Copyright Notices

Notice 1

Notice 2

Nanorobotics Control Design for Nanomedicine

by

Adriano Cavalcanti

Supervisor: Prof. Dr. Bijan Shirinzadeh
Co-Supervisor: Prof. Dr. Toshio Fukuda
Co-Supervisor: Prof. Dr. Luiz C. Kretly

A DISSERTATION SUBMITTED TO THE DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING, AT MONASH UNIVERSITY, FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN ENGINEERING

accepted on recommendation of

Prof. Dr. Sylvain Martel
Prof. Dr. Bradley Nelson

October 2009

Version 9.46

Monash University Victoria 3800 AUSTRALIA
October 2009

This work is dedicated to
the coming generations.

With love to..... Ester M. Paes Cavalcanti, my wife,
from who I have learned a quite few things,
such as determination and patience.

Camila, my little daughter,
who has suffered due our long absence
during my graduation time.

Just a few quotes...

"God does not play at dice."
Albert Einstein, *(1879-1955)*

"There is nothing permanent except change."
Heraclitus of Ephesus (ca. 525-475 B.C.)

"A scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die and a new generation grows up that is familiar with it."
Max Plank (1858-1947)

"A pessimist sees the difficulty in every opportunity; An optimist sees the opportunity in every difficulty."
Winston Churchill (1874-1965)

Abstract

The purpose of this thesis is to present a new paradigm for nanotechnology automation. Therefore, the work provides a computational methodology for control design of nanorobots with an application in medicine.

The subject under study concentrates its main focus on the control design of nanorobots for biomolecular assembly manipulation and the use of evolutionary agents as a suitable way to achieve the adaptive features required for the proposed model. Furthermore the work presents the use of neural networks as the most practical technique for the problem of robot motion optimization using a sensor based system. Thus, the author proposes a useful approach within advanced graphics simulation for nano-assembly automation with its focus on an application for nanomedicine. The motivation for such a study is the fact that with the emerging era of molecular engineering, the development of methodologies that facilitate analytical and empirical investigation, should help in the system architecture analysis, improving the evaluation of new approaches for insightful comprehension of nano-worlds. Therefore, it should provide a great impact for effective design of control instrumentation, helping in the development of nanotechnology.

The presented nanorobot model is required to survive and interact with a complex environment. Furthermore the nanorobot has to address a pre-defined set of tasks both in a competitive scenario and in a cooperative collective environment. In a three-dimensional environment our nanorobot monitors a determined number of organ inlets' nutritional levels, capturing and assembling new biomolecules into proteins that have to be delivered to the organ inlets with higher priority during each moment of our dynamic simulation. The nanorobot must avoid fuzzy obstacles, and must with proper time and manner react in real time for an environment requiring continuous control. In order to achieve the most appropriate pre-programmed set of behaviours the nanorobot uses a local perception through simulated sensors to effectively interact with the surrounding workspace. Thereby this work addresses distinct aspects of the main techniques required to achieve a consistent nano-planning systems design through the analysis of numerical results.

To provide a feasible design for the behaviour of a reactive nanorobot, the computational architecture adopted parallel processing as the natural way to achieve a modular design. This enables a functional orientation focused on each main aspect related to an intelligent sensor-based nanorobot's successful performance. For such an aim, it used feedback evolutionary decision control activation, neural motion control, and real time environment interaction methodologies. The application of stochastic models has provided an appropriate evolutionary agent behaviour, which was shown to be the most effective methodology for any situation when a more specific action description does not attend a large number of complex elements in a dynamic environment. The model includes stochastic techniques, addressing aspects inherent to quantum uncertainties present in the microscopic spaces. We have employed the proposed nanorobot in an evolved physically based simulated environment in a series of task-based non-trivial problems, and have studied the adaptive properties of distinct nanorobot behaviour with a design to address each environment with respective rules to trigger control activation for behavior activation and complexities. Thus the development of new concepts on nanomechatronics and automation theory is focused on the problem of molecular machine systems. A novel adaptive optimal methodology is described and the model validation is demonstrated successfully through the application of nanorobot control design for nanomedicine.

"Time flies when you are among good friends."

- Old Basque Saying

ACKNOWLEDGMENTS

The completion of this thesis was the result of the help, cooperation, faith and support of many people, thus I would like sincerely to thank all of them.

Foremost, my special thanks for Prof. Bijan Shirinzadeh as my supervisor during my time in his laboratory. He and the team of Monash University provided me with the required support for the development of innovative automation paradigms in the new field of medical nanorobotics automation. His contributions and enthusiasm in supervising my PhD thesis has ensured the success of this work.

I wish to specially acknowledge as my co-supervisor Prof. Toshio Fukuda at Nagoya University, for his support and interest in the development of this work towards the investigation of new control techniques for medical nanorobotics.

My special thanks also to Prof. Luiz C. Kretly at State University of Campinas, for his incentives, co-supervision with important and significant technical contributions on digital analog simulation and architecture instrumentation.

My thanks go equally to Prof. Mark Thompson, also from Monash University, for his encouraging words and for sharing his wise vision about research and development.

Thanks to the library teams of Flinders University and Monash University, which provided many insightful articles on the topic of nanobiotechnology.

The author also thanks for fruitful discussions, technical collaboration and suggestions: Arancha Casal, Bill Nace, Constantinos Mavroidis, Declan Murphy, Lior Rosen, Mingjun Zhang, Robert A. Freitas Jr., Seiichi Ikeda, Tad Hogg, and Warren W. Wood, for all the helpful comments provided during the development of this project.

To Edgars and Irmgard Krievins, Edison V. Monteiro, Heiko Schors, Helga Wahre, Hermann and Adelheid Sporer, Hermano M. F. Tavares, Joao A. C. Mangabeira, Jurandir F. R. Ferndandes, Lara Osborne, Maria Salgado, Marinea Santos, Richard and Margaret Farrell, Takaaki Ohishi, and Tim Kelleher, for their incentives and friendly support at the very beginning of this journey.

I thank all my family who kept believing that science, research and technology may be used to improve the human condition. Thank you all for being patient and understanding my absences during my time in the graduate school, especially my dad, Nitercílio, my mom, Dionice, my brother Ade, as well as my grandparents, Leopoldo and Amélia. My special gratitude to Ernestina and Gema, who have missed us through all this journey.

This project was partially supported by the Australian Research Council (ARC).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

The new field of nanoscience and nanoengineering began recently, opening new possibilities and challenges. The gap between the top down strategy and bottom-up strategy to build NEMS (nanometer-sized electromechanical structures) has been gradually reduced. Moreover, powerful tools to manipulate nanometer-sized objects are emerging. So, now we are entering a new stage to build NEMS and MEMS (microelectromechanical systems) based on nanotechnology. In the USA and Japan government research ministries have provided significant resources for the rapid development of nanotechnology, and in Europe the same serious approach is taken. A US$ 1 trillion market consisting of devices and systems with some kind of embedded nanotechnology is projected by 2015 [269], [130]. Another article announced that corporations intended to achieve revenues of around US$ 1 Billion of profit using commercial nanoproducts by 2012 [324]. Also, a first series of commercial nanoproducts has been announced as possible by 2007 [147], and nanoelectronics are incorporated since 2008 into products currently available in the marketplace.

In the medical area, the same miniaturization of devices is expected to have a direct impact on biomedical instrumentation and practices [367]. Hence, the first class of nanorobots, that are expected to have revolutionary applications in such areas as health care and environmental monitoring, are likely to emerge for the coming decades [307]. To reach this goal of building electronic devices in nanoscales, firms are forming collaborations and alliances that bring together new nanoproducts through the joint effort of corporations such as IBM, Motorola, Philips Electronics, Plastic Logic, Palo Alto Research Center Inc (PARC), Xerox Research Centre, Hewlett Packard, Dow Chemical, Bell Laboratories, Lucent Technologies, Royal Philips Electronics, E Ink Corp., DuPont, Rolltronics Corp., Intel Corp., Thin Film Electronics ABA, just to quote a few [274].

The key to advancing this technology is the development of new methodologies and nanomechatronics techniques that explore the nano-world [65]. Some efforts in the development of intelligent automated molecular systems design have been undertaken [308]. The increasing importance of prototyping techniques to enable rapid design can be observed, which must serve to address complex aspects of the physical principles used for the production of final 3D prototyping

[335]. The recent developments of a new branch in the field of computational nanotechnology known as nanoCAD [66] has drawn attention to the use of 3D visualization as a powerful tool applied to design of devices with nanoscale dimensions [72]. A major factor for the fast development of the nanotechnology field [70][66] should comprise a suitable strategy for the study and the design of tools for the development of integrative and multidisciplinary systems. Such systems should achieve a molecular manufacturing automation through the increasing and progressive implementation of interactive platforms for pratical nanomechatronics control and instrumentation.

While industry insiders cannot predict whether polymers or small molecules will rule the organic electronics universe in the end, all agree that the deciding factor will be manufacturing costs [274]. Controlled action at a distance, teleoperation, has been used in the past decades to extend man's reach into hostile or distant places. Tele-robotics systems for operating robots in hazardous environments, such as nuclear plants, and remote places, such as outer space, have been some of the different areas of robotics control application [331]. Now, the newest frontier to be conquered is not the macro-world or outer space, but the inside space, or nano-world. Since barriers within the nano-world increase significantly given difficulties to human direct interaction, there is a general agreement about the importance and necessity for the use of advanced simulation in the nanotechnology community. Analytical results and simulation should enhance new approaches as a practical pathway for control of future nanodevices into biomedical applications. Prototyping in 3D can also help automated planning and judgments about manufacturing feasibility, assisting chemical and biological assembly analyses in nanobiotechnology.

Two strategies, top down and bottom-up for creating nano systems have been presented [129] [113]. A combination of these two methods could be useful, i.e., firstly to fabricate building blocks through directed self-assembling to generate supramolecules (material goes bottom-up), and then to assemble them into more complex nano systems by smaller and smaller nanomanipulators (tool goes top down). Hence, nanomanipulation, or positional control at the nanometer scale, will be a key technology towards molecular nanotechnology.

Microsystems have been researched actively in the last 20 years, thus nanomanipulation is one of the most promising enabling technologies for MEMS based on NEMS and Nanotechnology. There are many application fields in this industry. The micromachines have the scale advantage to reduce the size of components. Miniaturization is essential for tasks to be carried out in narrow spaces. In most cases, in the early stage of the MEMS research, the key technology to build these microsystems was microfabrication based on lithography. However, recently there have been proposed a lot of new strategies to build microsystems. Based on the new fabrication methods, microactuators, microsensors, micro fluidic devices, and so forth have been produced. The accuracy of the fabrication process has been improved, and the processed devices have become more complex. Yet most of the microsystems are made using the top down strategy.

There was an idea to use the bottom-up strategy to build MEMS devices in the early '90s. They are supposed to be made from atoms or NEMS. In nature, physical things are made from atoms. So, this way of thinking is quite natural. Thus the emerging fields of nanoscience and nanoengineering is developing successfully, thereby the gap between the top down strategy and the bottom-up strategy is gradually reducing [308]. Moreover, nano-structured new materials are discovered and developed, and powerful tools to manipulate nanometer-sized objects are emerging [256]. So, now we are entering a new stage of integrating NEMS and MEMS based on nanotechnology. Innovative strategies required at present are summarised as follows: downsizing of the component (Micro to Nano); higher precision of machining accuracy; 3D manipulation and assembly technique; method and theory to overcome difference between the model and practice; different ideas in design approaches of nano-structured and

Figure 1.1: The diamond makers, left to right are respectively Drs. Francis P. Bundy, Herbert M. Strong, H. Tracy Hall, Robert Wentorf, Anthony Nerad and Jim E. Cheney.

functional materials; different ideas in nanomechatronics control; utilisation of the self-organisation phenomenon.

Among them, the 3D manipulation and assembly technique and theory to explore the nano-world will play an important role in nanotechnology. From this aspect, nanomanipulation is quite important. With the existing difficulties of exploring the nano-world, the use of computational nanotechnology is argued as an important tool in design [112].

The long-term purpose of nanomanipulation is to build novel functional nanometer scale structures and/or mechanisms, which would otherwise be unobtainable, using nanometer scale building blocks. The last version of nanomanipulators might be Drexler's assembler [113], which has been proposed as general purpose manufacturing devices that can build a wide range of useful products. Presently, nanomanipulation would be also helpful for the exploration of the nano-world. It might find applications in relatively simple nano structure fabrication and biology research in the near future.

Obviously when we are talking about nanotechnology as the manipulation of molecular structures, we cannot forget the historical work performed by the General Electric group in the fall of 1951 [171]. The G.E. team (see Figure 1.1) has become widely known as "the diamond makers" once they have assembled simple carbon molecules into diamonds. Many famous scientists and engineers took their turns at trying to make diamond from baser forms of carbon. Final reproducible success was not attained until the 1950s, after the G.E. scientist team had developed adequate thermodynamic understanding, the high-pressure-high-temperature apparatus, and the reaction path needed.

Nevertheless the first fine electromechanical practice on nanomanipulation came with a scanning tunneling microscope (STM) that was achieved by Eigler and Schweizer in 1989 [117]. They applied a STM at low temperature (4K) to position individual xenon atoms on a single-crystal nickel surface with atomic precision. The manipulation enabled them to fabricate rudimentary structures of their own design, atom by atom. The result is the famous set of images showing how 35 atoms were moved to form the three-letter logo "IBM", which also helped prove to the world that people indeed can move atoms. For such work they received the Nobel Prize in Physics in 1986.

A Nobel Prize in chemistry was attributed in 1996 [142], to the work realised by the IBM Zurich Research Center. The reason was their achievement: they have succeeded in positioning individual organic molecules at room temperature by purely mechanical means. They "hand-picked" for the first time an organic molecule with 173 atoms and a 1.5nm diameter, including a porphyrin core, for the experiment. Choosing 6 such molecules from a set randomly positioned on a copper surface, they pushed each molecule into position to form a ring. This configuration would not normally be found in nature [138]. It might also be possible to align and maintain these molecular gear teeth in atomically precise meshed positions. Since that feat, more researchers have used STM or other versions of nanomanipulators to create letters, and pictures, as well as exotic physical structures on surfaces using one atom at a time. Although still in its primitive stage, this was just the kind of submicroscopic manipulation that the physics Nobel Prize winner Richard Feynman was talking about in 1959 [129] [323]. Continued efforts are being made to develop atom- and molecule-manipulating tools that can be more effective and easier to use. Key technologies for micro and nanomanipulation include observation, actuation, measurement, system design, control, calibration, fabrication, communication, and human-machine interface, among others.

1.2 Nanotechnology Range of Applications

Nanotechnology has a goal of 3D manipulation of chemical moieties to build molecules/clusters and then to assemble them into larger devices and materials. Achieving this requires combining techniques of chemical synthesis with engineering methods that wield atomically precise positional control. Better understanding of mesoscopic phenomenon would help to make automatic, high-speed and precision manipulation possible [311].

Although Scanning Probe Microscopy (SPM) has been used widely for topographical imaging, atomic/molecular manipulation, and nanoscale lithography, the low raster speed became a major drawback, limited by present cantilever and system dynamics to about 50 Hz/line. To alleviate this problem, several groups of scientists are developing arrays of cantilever probes that are individually actuated and controlled [261][263]. Further development may provide tools for large-scale and high density manipulations.

Manufacturing technologies such as microelectromechanical systems (MEMS) are potentially capable of producing higher degree-of-freedom micromachines, which can exert molecular-level positional control and bridge mesoscopic extremes in handling nanoscale and microscale components. Extension of MEMS into nanometer-sized electromechanical structures (NEMS) will achieve that capability. In combination with chemical schemes and self-assembly concepts MEMS/NEMS will form an essential generation of hybrid machines for subsequent stages of nanotechnology development.

Nanotechnology will allow mankind to exploit the ultimate technological capabilities of electronic, magnetic, mechanical, and biological systems by providing different kinds of nanodevices and techniques [144]. While the best examples at present are clearly associated with the information technology industry, the potential for nanotechnology can be much broader. Nanotechnology will ultimately have a direct impact on our ability to enhance energy conversion, control pollution, produce food, and improve human health and longevity. The applications of nanotechnology are summarized as follows.

1.2.1 Nanoelectronic devices

To provide ever faster and cheaper computers, the size of microelectronic circuit components will soon need to reach the scale of atoms or molecules. The idea that a few molecules, or even a single molecule, could be embedded between electrodes and perform the basic functions of digital electronics-rectification, amplification and storage- was first put forward in the mid-1970's. The concept is now used for individual components, e.g. CNT-based nano electronic devices [354][114] [203].

Potential applications are in digital radar, electronic support measures (ESM) receivers, ATM data stream processing, wide bandwidth communications, digital image processing, waveform generation, and the broad area of analog to digital (A/D) applications.

1.2.2 Next generation storage media

Resonant tunneling devices are being explored with demonstrated successes in multivalued logic and various logic circuits and memory circuits. SET logic and memory concepts are being explored with a focus on memory applications [159][250]. Spin devices in the form of nanomagnetics using the magnetoresistive effect in magnetic multilayers have demonstrated their use for nonvolatile, radiation-hard memory. Integration of scanning probe tips into sizable arrays provides a mechanical information storage strategy [241]. Cross-bar architecture is realised with CNTs [313].

1.2.3 Quantum calculation machines

Quantum computing is a joint venture between computer science and quantum physics. Basically two issues motivate quantum computing:

- Quantum mechanical concepts must be applied to solve tractable computing problems.
- From a computer miniaturization point of view, the size limit of a bit of information is important. Recently, this issue has attracted increased attention, due to the current development of nanotechnology and the design problems of semiconductor and metal devices that are approaching the quantum size limit. Consequently, the idea of quantum computing, in which the elements that carry the information are atoms, has attracted the attention of many scientists. Quantum cellular automata and coupled quantum dot technology are being explored and their potential assessed for transistorless computing [103][100].

1.2.4 Surface measurement

The invention of the STM (Scanning-Tunneling Microscope) [41] and AFM (Atomic Force Microscope) [40] have spawned the development of a variety of new scanning probe microscopes (SPMs) [372]. As a class, the SPMs measure local properties with nanometer-scale spatial resolution by bringing a sharp tip in proximity (1-10 Å) to a solid surface.

The proximity of the tip and surface enables the SPMs to operate in ambient temperatures, which is impossible with vacuum-based surface analytical techniques. The STM and the AFM were initially limited to monitoring fine scale topography. But the broader class of scanning probes, derived

from these initial instruments, allows one to go beyond topography and examine many other local properties, including the following: electronic structure, optical properties, temperature, dielectric constants, magnetism, charge transfer and the Helmholtz layer, biological molecule folding/recognition, and chemical information.

1.2.5 Molecular machines

Synthesis and processing of nanostructures will employ a diverse array of material types - organic, inorganic, and biological. Increasing emphasis will be placed on synthesis and assembly at a very high degree of precision, achieved through innovative processing. The result will be the control of the size, shape, structure, morphology, and connectivity of molecules, supermolecules, nano-objects and nanostructured devices and molecular machines. Integration of top-down physical assembly concepts with bottom-up chemical and biological assembly concepts may be required to create fully functional nanostructures that are operational at mesoscopic scales. The combination of new nanoscale building blocks and new paradigms in assembly strategies will provide nanostructured materials and devices with new, unprecedented capabilities, limited only by our imagination.

Building blocks for nanostruture include: (i) polymeric materials, dendrimers, block copolymers [358], (ii) Nanocrystals [54], (iii) nanotubes and rods, and (iv) nanoparticle structures. Processing methods of nanostructures include assembly [2], templated growth of mesoporous materials [8], direct structuring and nanoimprint lithography.

1.2.6 Genetic analysis

During the last few years, scientists have developed the technology for rapidly mapping the genetic information in DNA and RNA molecules, including detection of mutations and measurement of expression levels. This technology uses DNA microchip arrays that adapt some of the lithographic patterning technologies of the integrated circuit industry. Work on new types of chemical arrays should expand this approach of parallel biological information processing to the analysis of proteins and other biomolecules. Miniaturization of allied analytical processes such as electrophoresis will lead to increases in throughput and reduced cost for other important methods of analysis, such as DNA sequencing and fingerprinting [361][297].

1.2.7 Nano-biology

The ability of DNA to undergo highly controlled and hierarchical assembly makes it ideal for applications in nanobiotechnology [379][325]. For example, DNA has been used to design lattices that readily assemble themselves into predictable two-dimensional patterns. These arrays are composed of rigid DNA tiles, formed by antiparallel strands of DNA linked together by a double-crossover motif analogous to the crossovers that occur in meiosis. The precise pattern and periodicity of the tiles can be modified by altering the DNA sequence, allowing the formation of specific lattices with programmable structures and features at a nanometer scale. This approach has the potential to lead to the use of designed DNA crystals as scaffolds for the crystallization of macromolecules, as materials for use as catalysts, as molecular sieves, or as scaffolds for the assembly of molecular electronic components or biochips in DNA-based computers. Similarly, biological-molecule-based scaffolding could take advantage of the unique structural characteristics of RNA molecules, of polypeptide chains, or of the highly specific interactions that occur between DNA and proteins or between RNA and

proteins. Devices that are currently in use to control the interactions of DNA on surfaces can have broader applications for controlling nanoassembly. These devices use electric fields to control the movement of particles toward or away from microscopic sites on the device surface. Charged biological molecules (DNA, RNA, protein) and analytes, cells, and other nanoscale or microscale charged particles can be precisely organized [318].

1.2.8 Medical supply

Scientists are using strategies learned from biological systems to design new materials and complex biological systems. Such information provides the basis to design components that can come together in only one way to form a desired three-dimensional nanoarchitectural material for systems and components. For example, spider silk is one of the strongest materials known. Its molecular structure is being used to design better composite polymer systems for increasing strength and utility. Nanoparticles considerably smaller than one micron in diameter have been used in revolutionary ways to deliver drugs and genes into cells. The particles can be combined with chemical compounds that are ordinarily insoluble and difficult for cells to internalise. The derivatized particles can then be introduced into the bloodstream with little possibility of clogging the capillaries and other small blood vessels, as in the case of insoluble powders. The efficacy and speed of drug action in the human body can thereby be dramatically enhanced. In similar ways, nanoparticles carrying DNA fragments can be used to incorporate specific genes into target cells [142].

1.2.9 Others

The trend to smaller and smaller structures, through miniaturization, well known in the microelectronics industry, can be evidenced by the rapid increase in computing power through reduction of the area and volume needed per transistor on chips. In the energy and chemicals areas, this same trend towards miniaturization, i.e., control of function and/or structure at the nanoscale, also is occurring, but for different reasons. Smallness in itself is not the goal. Instead, it is the realization, or now even the expectation that new properties intrinsic to nanostructures will enable breakthroughs in a multitude of different technologically important areas. Nanoenginnering is expected to lead into significant improvements in fields such as: solar energy conversion and storage; better energy-efficient lighting; stronger, lighter materials that will improve transportation efficiency; use of low-energy chemical pathways to break down toxic substances for remediation and restoration; and better sensors and controls to increase efficiency for manufacturing and processing [17].

1.3 Recent Developments and Motivation

Works in molecular manufacturing has emphasized the need for very small and very accurate manipulators that simultaneously have a wide range of motion to enable the assembly of molecular components [111]. New approaches for nanorobotic motion and control design have been proposed, where the models consider thermal noise as a significant source of positional uncertainty, comparing a robotic arm, Stewart platform and a five-strut crank model [256]. A precursor work for nanoassembly automation was presented for a modern molecular library for proteins, DNA, and RNA assembled by highly automated robotic equipment [101] with polymers approaching 10^{14} sequences and libraries with more than 2×10^5 members [210]. Another study has provided a detailed 2000 atom molecular

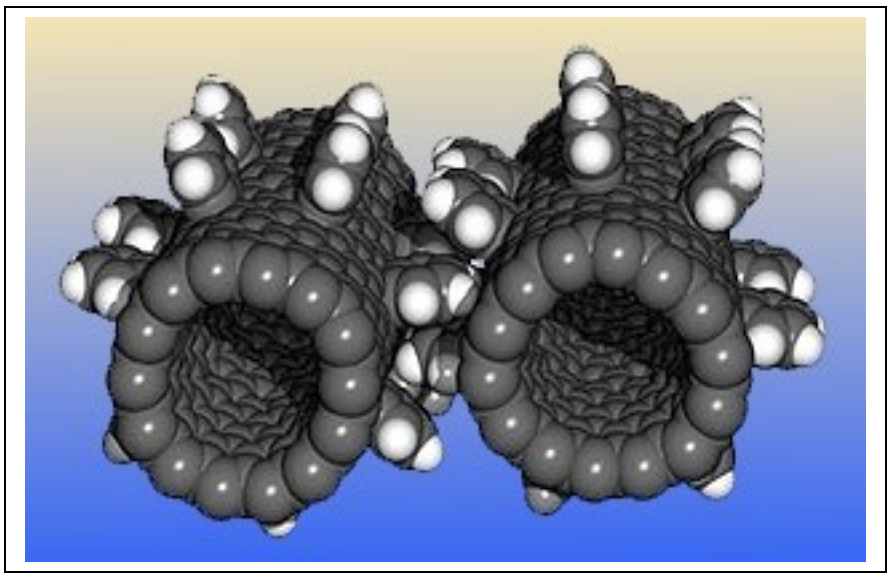

Figure 1.2: Nano-gear design by NASA Ames.

dynamics simulation to investigate the properties of molecular gears fashioned from carbon nanotubes with teeth added via a benzyne reaction known to occur with C_{60} [278]. Computationally, one gear is powered by forcing the atoms near the end of the nanotube to rotate (Figure 1.2), and a second gear is allowed to rotate by keeping the atoms near the end of the nanotube constrained to a cylinder (i.e., the ends of the shaft were constrained to not elongate but were allowed to move within a plane transverse to the tube symmetry axis). Other types of nanotube-based gear systems were also simulated.

Among other works the ORNL group simulated, was a fullerene motor consisting of two concentric graphite cylinders (shaft and sleeve) with one positive and one negative electric charge attached to the shaft [285][272]. Rotational motion of the shaft was induced by applying one, or sometimes two, oscillating laser fields. The shaft cycled between periods of undesirable rotational pendulum and good unidirectional motor-like behaviour.

Robotic manipulation and assembly of objects at the nanoscale is a branch of nanorobotics that has generated considerable interest and promises to produce revolutionary advances in miniaturization towards developing molecular machine systems. Practical approaches for nano-planning systems have been presented as a first step towards automating assembly tasks in nanorobotics [245], where there was presented a 2D assembly task automation.

The design of a molecular library by using 160,000 reactions has used a genetic algorithm approach consisting of coding 10 isocyanides, 40 aldehydes, 10 amines, and 40 carboxylic acids in a "bit-string" data structure [369], where it was suggested that the method could be fully automated with robotic handling and fluidic transport. A closely related work in the possible automation of nanoscale manipulation has proposed a fully autonomous motion manipulator system capable of performing 200,000 accurate measurements per second at the atomic scale [248].

Recent developments in the field of biomolecular computing [1] have positively demonstrated the feasibility of processing logic tasks by bio-computers [163], which is a promising first step to enable the manufacture of future nanoprocessors with increasing complexity, power for information storage, and data processing capacity, which could be considered as indispensable components for the real automation of nanosystems.

Intel Corp.'s prototype 90nm process facility has already produced a fully functional 52Mb Static Random Access Memory (SRAM) with transistor gate lengths of 50nm and SRAM cell sizes of

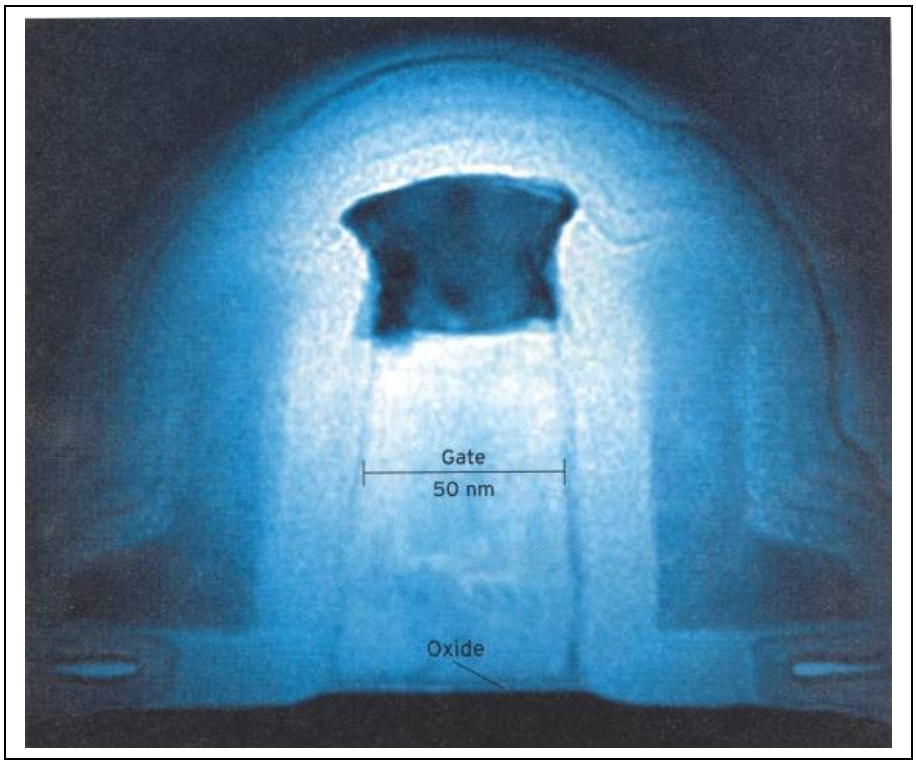

Figure 1.3: Nanotransistor shown in this scanning electron micrograph by Intel.

just $1\mu m^2$, or roughly half the cell size of today's most advanced SRAMs (see Figure 1.3). This downscaling will continue, according to the Semicondutor Industry Association's roadmap. High-performance ICs will contain by 2016, more than 8.8 billion transistors in an area 280 mm^2 (see Figure 1.4) - more than 25 times as many as are on today's chips built with 130-nm feature sizes [147].
A recent study has addressed the performance of fast electronic switching combined with slow mechanical nuclear vibrations, and switching was studied showing the aspects required for the development of nanoelectronics [92].

Developments in the sense of building biosensors [348] and nano-kinetic devices have advanced recently too [342][272][18][221], and could be considered by many researchers as a prerequisite for making nano-automation feasible to enable nanorobotics operation and locomotion. The paths for the development of nanobiomotors [272] have been explained through the study of biological processes that occur at the molecular levels [243]. The use of ATP synthase was used to convert chemical, osmotic and mechanical energy, providing the basis of initial nanomotors.

The application of artificial intelligence as the most appropriate means to enable some aspects of intelligent behaviour for the control of nanorobots, with the intention of facilitating a major improvement in the cost-effectiveness of molecular manufacturing, and finding a suitable assembly sequence for end-specified molecules, has been discussed and accepted in the nano community [94]. In this aspect, an acceptable approach is the use of agents as assemblers, where the most suitable model would be projected ideally as close as possible to concepts related to Artificial Life. Thus, in order to be useful, nanoscale assembler have to be controlled by robust, scalable, flexible software, which will enable the system to survive in very chaotic environments, and such characteristics could be better satisfied by the use of concepts like reinforcement noise proven models, adaptive control systems, ants, neural networks and genetic algorithms [69][65].

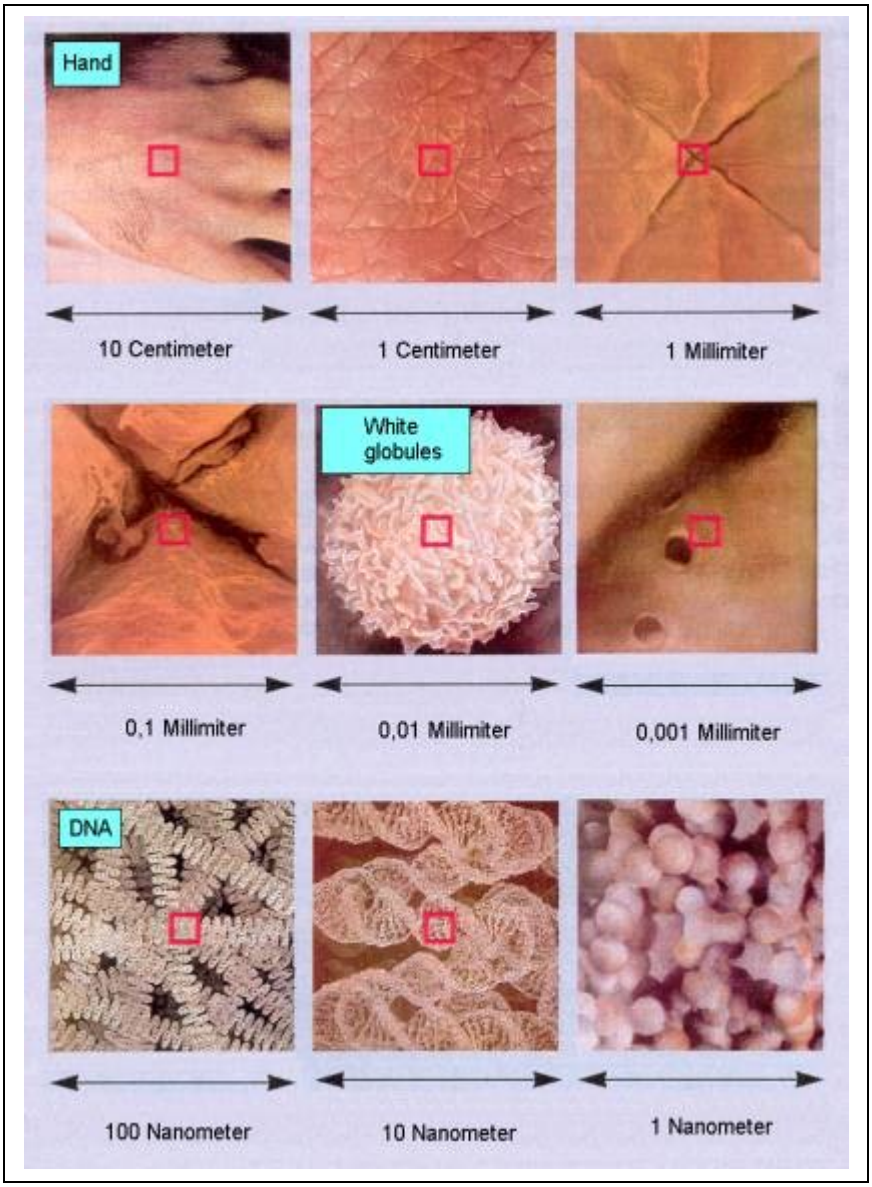

Figure 1.4: Nanometer size comparisons: macro, micro and nano by EAMES Office.

Taking the above points into consideration, micro/nano physics-based robotics and sensing with intelligent control is indispensable for micro/nano manipulation. Therefore, the development of automation for real-time sensory feedback intelligent control clearly became one of the highest key technological factors for the fast development of nanotechnology [245]. In this sense, to enhance the nano-automation systems and prototyping methodology, the present work will address the design of a fully automated nanorobotic system with the use of a 3D computational nanomechatronics approach for biomedical applications. The main theoretical analysis and mathematical aspect are detailed in the work supporting the predefined model for intelligent behaviour, as well as some distinct issues of specific techniques required to achieve a successful nano-planning system design with a simulation visualisation in real time. The model implementation is achieved with parallel processing and dynamic collision detection, showing the effectiveness of each module that comprises the system architecture for nanorobot control design, and what enables the nanorobot to react adaptively in a stochastic environment.

1.4 Proposed Approach

The aim of this work is to propose the analytical and computational study of a new control paradigm, using computer aided design and real time physically based simulation, for the implementation of new concepts and methodologies that can support the automation of collective medical nanorobotics. The nanorobot model is pre-programmed to perform a set of tasks related to the nanoassembly automation and control. Actually, a fully robust control model, enabling collective robotics features, is considered an important and complex problem to achieve massive molecular manipulation. For the effective development of nano and biotechnology we can consider as critical the implementation of new tools and systems adopting multidisciplinary methodologies. The demand for automatic manipulation, encoding and control of macro and nano structures of biological materials is paramount. In general lines, we can describe a nanorobotic molecular machine system as a system able to perform molecular manufacturing at the atomic scale, whose constituent robots are capable of interacting with the surrounding environment.

The main focus in this thesis deals with nanorobot control design for nanomedical application, where a set of pre-defined tasks is performed by nanorobots capturing proteins in a 3D microscopic environment. Afterwards, these same biomolecules should be delivered to a set of organ inlets requiring drug delivery or protein injection. For our analytical analysis, we chose nano-manipulation in a liquid workspace, which is mostly relevant to biomedical applications.

The proposed design has to be robust enough to operate in a complex environment with movement providing six-degrees-of-freedom. Taking into consideration all the characteristics described above, we adopted the use of non-deterministic approaches as the most feasible control technique. Like some techniques inspired by biological and natural models, and evolving some capabilities characteristic of artificial life and intelligent agents, it should inherit some ideal counterparts for such a nanorobotic model. Thus, a model using evolutionary techniques and artificial neural networks was adopted to be the most appropriate way for an adaptive model and is used in the proposed study. The reason for this choice is based on the fact that the nanorobot must be capable of reacting in real time, corresponding in accordance with different changes and requirements that come from the surrounding dynamic environment.

1.5 Contributions

We are presenting an innovative control model for medical nanorobotics, which is implemented with a simulation of adaptive intelligent behaviour in stochastic spaces. The investigation of new methodologies on control automation for molecular machines helps towards complex analytical design, and opens new paradigms, required for the fast development of nanotechnology with potential applications in biotechnology and nanomedicine [69][66]. More specifically, the core of present work consists of considering the main aspects required for the control design and system simulation architecture of mobile nanorobots. The nanorobot is comprised of nanoscopic components and has a size comparable with a bacteria to perform molecular manipulation for nanomedicine [142].

The scheduling problem considers the biomolecular manipulation, which is automatically performed with protein by an intelligent agent. The agent has, as its mission, the improvement of nutritional levels of organisms, performing the injection of appropriate substances to the pre-established delivery points. Furthermore, the present model is required also to incorporate either competitive or cooperative multiple-robotics concepts in order to enable a suitable design model for

testability, featuring a massively coordinated action of large proportions. Both aspects could be considered as a basic requirement for a successful automation model to achieve an efficient nanorobotics control design. Therefore contributions of this work could be outlined as follows:

- To present a control approach featuring a highly adaptive design, capable of competitive and reactive decisions in a dynamic and complex environment.
- To elaborate a control model for the interaction of coherent collective nanorobotics, advocating the use of stochastic methodologies to be the most robust and suitable automation methodology for stochastic 3D environments, as those related to medicine.
- Outlining the use of physically based simulation as a powerful technique to deals with nano-worlds representation and dynamic simulation.
- Exposing the necessity of real-time advanced graphics simulation in the nano-world as a valuable tool for the study to gain better insight into kinematics and stochastic aspects, where computational nanotechnology with nanomechatronics and scientific visualization becomes a very important tool for supporting detailed interactive analysis.
- To emphasize the importance of using a parallel modular approach as the most effective way to evolve a nanorobot design, which would be required to react adaptively in an uncertain environment.
- For demonstrating the local perception (sensor-based control) as the most feasible alternative for medical nanorobotics automation approach.
- Using evolutionary techniques and feedforward neural networks as methodologies require lower computational time and resources. This provides higher flexibility and adaptability for control investigation of medical nanorobotics.

Thus, the present study makes a detailed investigation of the main issues related to molecular machine systems design to provide a robust and flexible control methodology for the problem of nanorobot and molecular machine automation, addressing the key aspects related to this very promising and new field of research known as medical nanorobotics control automation.

1.6 Thesis Outline

The development of nanorobots presents difficult fabrication and control challenges. Such devices will operate in microenvironments whose physical properties differ from those encountered by conventional robots. Particularly interesting microenvironments are those involved in nanomedicine applications, where the robots should operate inside the body to provide significant new capabilities for diagnosis and treatment of diseases. Since nanorobots cannot yet be fabricated but are planned for next coming decades [307][367], evaluating possible designs and control algorithms requires using experimental analyses and simulators [65]. Such simulators can operate at various levels of detail to tradeoff physical accuracy, computational cost, number of robots and the time over which the simulation can follow the robot behaviour.

Hence, we propose as an investigation approach a physical simulation of a 3D microenvironment incorporating major differences between conventional robotics and the situation facing nanorobots for medical applications, specifically motion dominated by viscous rather than inertial forces in fluids. We focus on a typical task for such robots: locating specific targets in the

environment and maintaining chemical concentrations at desired levels near them. Our simulation shows nanorobot behaviours and runs quickly enough to allow rapid evaluation of various control algorithms.

Taking into account the most important cited points, the control model proposed could be described and divided into some main aspects and modules, covering the following issues:

- **Design:** for the nanorobot design, it was necessary to consider concepts related to the field of application. Thus, Chapter 2 provides information about the main aspects of nanotechnology related to nanomedicine, such as related to nano-world and quantum uncertainty, infering directly to important concepts for the automation of biomolecular manipulation. All those particularities of nanotechnology environments and science have to be taken into consideration for the development of our nanorobot design.

- **Sensing:** for the sensing aspects, the nanorobots require the approach of sensor-based control, which is more appropriate for medicine, as performed in the model by the use of dynamic physically based simulation, using hierarchical collision detection, as discussed with more details in Chapter 3.

- **Motion control:** for motion control, optimization was adopted in the use of neural network techniques. In Chapter 4 this is described and considered in the main aspects related to robot motion in complex and stochastic environments, as well as the motion with multiple robots in the same space. The neural network algorithm implications and principal considerations are then described in Chapter 5.

- **Decision:** the nanorobot model concept uses evolutionary techniques, which permit the nanorobot to decide in an adaptive fashion to react in accordance with the environmental changes controlled by a mathematical fitness function, which models a detailed pre-defined set of actions to be taken by the nanorobot based on a penalties and rewards approach. The genetic algorithms technique is described in Chapter 6, and the methodology application for nanorobotics in nanomedicine is discussed in more detail in Chapter 8.

- **Model architecture:** the use of parallel processing as a methodology necessary for the integration of the different modules for model simulation is a feasible approach for systems architecture implementation. Therefore in Chapter 7 we have a description on parallel processing applied to robotics. The necessary techniques and system integration aspects for the proposed system design are described in Chapter 8.

The validation of the model under analysis is discussed in Chapter 9, with the study of two different control approaches in distinct scenarios, as it describes how to optimize and adjust behaviour control of nanorobots in a stochastic environment. Chapter 10 outlines the main conclusions and future directions for the development of nanorobotics control design with automation for nanomedicine.

Chapter 2

Nanotechnology

2.1 Introduction

Nanotechnology is a very new field, comprised of an interdisciplinary set of sciences [66], such as computer science, physics, chemistry and biology. Regarding the impact of this new field of science on our society, this is something that can not be completely and precisely predicted. However, for the scientific community, the only thing that is clear is that the possibilities and the future results of nanotechnology indicate it as an exciting new research area. The potential fields of application in the nanobiotechnology ranges from the development of new materials in the field of metallurgy to advanced molecular machine systems in the field of medicine [69][74][65].

Nanotechnology will be a bottom-up technology, building upward from the molecular scale [111]. It will bring a revolution in human abilities like that brought by agriculture in the 19th century or power machinery in the 18th century. Although major discoveries can be expected, we have no direct experience of the molecular world, and this can make nanotechnology hard to visualize, and consequently hard to understand. Actually most scientists working with molecules face this problem. They can often calculate how molecules will behave, but to understand this behavior, more than numerical analysis is sought: they need Computer Aided Design (CAD) approaches to achieve interactive simulations. For this reason, the U.S. National Science Foundation (NSF) has launched a program in "Scientific Visualization" [112][279] to address computational nanotechnology, in part as an incentive to strengthen the importance of advanced computational tools focused on the problem of investigating the molecular world.

Molecules are objects that exert forces on one another. If your hands were small enough, you could grab the molecules, squeeze them, and compress them. Understanding the molecular world is much like understanding any other physical world: it is a matter of understanding size, shape, strength, force, and motion, or a matter of understanding the differences between sand, water, and rock, or between steel and soap bubbles.

Nowadays there are basically four approaches to nanosystems for molecular manipulation [311]. The first is through consecutive linking of covalent bonds that results in one huge molecule or macromolecular structure. The second involves hydrogen bonding, van der Waals bonding, electrostatic bonding, and non-covalent bonding to assemble large systems of molecules. A third method involves forcing chemical bonding by a so-called positional assembly, which involves using

scanning tunneling microscopy [112]. A fourth method involves biotechnology to manipulate molecular components and systems into synthetic chemical compounds.

A molecular machine system could be described as a system to perform molecular manipulation at atomic scales, the constituent entities of which are capable of performing a pre-established set of tasks. The International Technology Roadmap for Semiconductors, published periodically by the Semiconductor Industry Association (SIA, San Jose, Calif.), has revised its projection for the 2003 technology node from 100nm to 90nm [147]. "Technology node" refers to the set of processes needed to print the smallest feature. True to the 2001 Roadmap projections, many foundries including Intel, TSMC, Philips, IBM, STMicro, Motorola and LSI Logic, have geared up together and started a volume production of 90nm processes in 2003 [147]. Scientific and engineering knowledge are on a growth curve that is accelerating exponentially [110], and international progress in molecular nanotechnology will take the same path. Mushegian's study [277] suggests that the minimal artificial biological nanorobot will consist of at least 300 different nanoparts (protein). Recently one private company has already been formed to pursue the construction of these artificial nanorobots [120]. Studies on robotics motion planning problems are moving to the new and challenging area of nanorobotics (with the construction of purely mechanical nano-computers, and construction of DNA assemblies), where the main focus takes on the potential applications of robotics motion planning [304] in nanotechnology and molecular computing.

The most important challenge that has become evident as a vital problem for the fast development of nanotechnology with industrial applications is the automation of atoms manipulation [72][66]. The starting point of nanotechnology to achieve the main goal of building systems in nanoscale, is the automation and development of molecular machine systems, which could enable a manufacturing schedule of nano-device building blocks. Thus a molecular machine system, which is the most desired achievement in the nano-community, will be the most advanced result for nanotechnology development.

Building patterns and manipulating atoms using the SPM has been used quite successfully [308] as a promising approach for the construction of nanoelectromechanical systems (NEMS). However, these manual manipulations require much time and as it is a repetitive task, it tends to be monotonous and imprecise when performed manually for a large number of molecules. Hence, automation systems in such a situation would greatly improve the productivity and precision in atoms manipulation.

2.2 Physical and Chemical Properties

One of the main directions of research in the field of mechatronics is the miniaturization of robots, machines and devices. This new branch of science is called micro/nano-mechatronics [69][65]. With the use of Scanning Probe Microscopes (SPM) as in Atomic Force Microscopy and Scanning Tunneling Microscopy, geometrical and electrical magnetic properties of material can be measured down to atomic scale in 3D with precision at up to 0.01nm resolution. Thus if our main long-range goal is to build a nanorobot, then we have to do more than speculate on its capabilities. We need also to describe some of the main aspects needed to make molecular-scale machines.

The use of classical rigid-body dynamics and semiclassical mechanics are quite sufficient for studying the rotational dynamics for building molecular components [311], which could be proved through the use of concepts provided by chemistry, protein engineering and scanning probe methods. The molecules will arrange themselves according to their configuration and the temperature of the surrounding medium. Hence, we can see that a specific molecular reaction will take place regarding thermodynamic perspectives and chemical kinetics.

The appliance of chemical kinetics for the study of molecular collisions from beam-type experiments can be used to deduce the mechanism of chemical reactions. From this understanding of the mechanisms, we can design specific molecular structures, as we can expect to do for molecular nanotechnology. About the consideration of quantum chemistry, advanced mechanics simplifies the equations of motion by the Lagrangian and Hamiltonian forms [350], and these are the mathematical methods for quantum mechanics.

The major strength of molecular mechanics is that energy minimizations for large systems can be computed in a reasonable amount of time. The computed answers are usually accurate, although the force constants from the preceding interactions were determined by empirical methods. The essential methods of molecular dynamics have been discussed [165], in this manner the kinematics of hard-sphere and soft-sphere collisions could be computed along with the intermolecular potential and time dynamics.

The methods usually involve finite-difference computations, which consist of solving partial differential equations. All those concepts and calculations could be found at molecular-design programs like Alchemy III and HyperChem [194]. More practically the quantum mechanical calculations are usually approximated by various methods, including the convolution technique, Huckel methods, and group theory involving symmetry operators, which allow one to achieve reasonable numerical values [311]. Molecules and atoms are generally considered for such mechanical calculations as hard spheres.

2.3 Nano Manipulation

Micro/nano manipulation approaches can be classified depending on the starting point, process, interaction and operation as given in Figure 2.1 [144]. In respect to the starting point for manipulation, systems can be classified as bottom-up and top-down approaches [223].

We can consider the self-assembly technique as an example of the bottom-up approach. It is based on a manipulation technique, using a biochemical process that can be utilized for constructing micro/nano devices or materials. Several laboratories especially in the fields of chemistry (supramolecular chemistry) and biology are trying to use this approach [235] where it is promising in building highly-repetitive or symmetrical structures. For top-down, also known as positional nanoassembly, the approach using STM or AFM to manipulate atoms or molecules [322][309][343] [204] is allowing molecular structures that may never exist in nature. Hence, the technique [292][192] [206] [321][209][316][352][42][280] starts from familiar operations in the macroscopic world and moves towards smaller and smaller objects, and more and more precision of handling. In this approach Codourey [87] achieved a pick-and-place task for octagonal diamonds with 50µm diameter. Koyano proposed new micro object handling and teleoperation system with concentrated visual fields and new handling skills [222]. They achieved the fabrication of pyramidal 3D structures made of polymer particles of 2µm in diameter using Scanning Electron Microscope (SEM) and a two-arm micromanipulator system [267]. Zesch [380] utilized piezoresistive AFM cantilevers for force-controlled 2D pushing of microparts on a planar substrate. Zhou [381] tried to control the position of a microcantilever on a substrate precisely by integrating visual and force feedback.

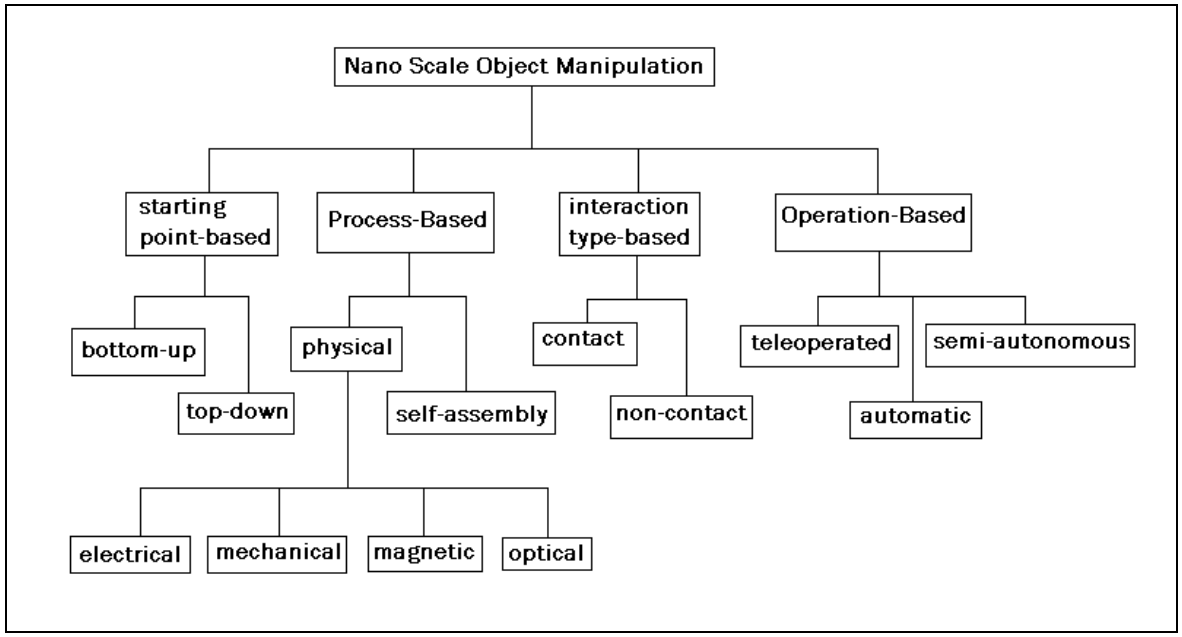

Figure 2.1: Micro/nano scale object manipulation approaches.

The positional nanoassembly, or positional nanomanipulation, uses forces such as electrical, mechanical, magnetic and optical forces. By physical manipulation, an external force for positioning or assembling objects in 2D or 3D, cutting, drilling, twisting, bending, pick-and-place, push and pull kinds of tasks are to be carried out. STM-based atom/molecule manipulation systems control the electrical force between the metallic STM probe and the substrate atom/molecules for pick-and-place, or push-and-pull kind of tasks [343][242][362]. AFM-based manipulation systems utilize the AFM tip for pushing or pulling nano objects such as particles [322][34][168][331] or cutting DNA [338] or fiber [288], by controlling the applied mechanical load on the sample. Actually using the AFM-based system, the smallest pushed particle manipulation is around 15 nm radius, moving atoms or molecules with respect to the tip size of the microscope and strong interatomic forces [204]. Inoue [195] utilised magnetic fields to control the motion of the microparts. Finally, laser-trapping approaches also can be used to move micro/nano particles in 2D or 3D by the applied laser light force [320].

Depending on the interaction type, non-contact and contact manipulation systems can be used. In the former, laser trapping (optical tweezers) or electrostatic or magnetic field forces are utilized. Yamamoto [378] can cut DNA using restriction enzymes on a laser-trapped bead. Stroscio [343] utilized electrical force between an STM probe tip and the surface atoms for non-contact manipulation of Xe or Ni atoms. As the contact manipulation, an AFM probe tip is utilized for positioning particles on a substrate by contact pushing or pulling operations [322][204][310].

The operation-based approaches, are the teleoperated and the automatic approaches [333]. Teleoperation technology at the initial phase is a promising tool for understanding quantum uncertainties and improving automatic manipulation strategies used by the human interface [330]. Teleoperation systems have the stages of direct teleoperation, and task-based-semiautonomous teleoperation systems, where in the former, an operator directly enters the control-loop of the micro/nanomanipulator, and in the latter, the operator only sends high-level task commands, and the manipulator performs the tasks in an autonomous way.

PROPERTY	Microscopes	AFM	STM	SEM	OM
	Highest Resolution	0.1nm	0.1nm	~ 5nm	~ 40nm
	Visible object Types	All	½ conductors	½ conductors	All
	Imaging type	Near-field	Near-field	Far-field	Far-field
	Object Interaction	Contact non-contact	Non-contact	Non-contact	Non-contact
	Imaging Environment	All	Vacuum or air	Vacuum	Air or liquid
	Imaging Principle	Interatomic forces	Tunneling current	Electron emission	Material-light interaction
	Imaging Dimensions	3D	3D	2D	2D

Table 2.1: Imaging devices used for micro/nano manipulation and their properties.

Hollis [181] utilized an STM probe as the slave-robot and 6-DOF fine motion device called Magic Wrist as the master device for feeling atomic scale topography in the operator's hand. The nanoManipulator group utilizes commercial AFM and a haptic device for a real-time haptic display [124]. They utilize a plane and probe model [247][161] for surface force feedback, but they do not have any report on scaled teleoperation control problems and micro/nano force modelling. The final goal is a fully automatic system that can enable the mass-production of micro/nano robots or machines.

For the imaging devices during micro/nano manipulation, OM (Optical Microscope), SEM (Scanning Electron Microscope), AFM (Atomic Force Microscope) and STM (Scanning-Tunneling Microscope) are the most frequently used microscopes. Their properties are reported in Table 2.1. Up to the 1μm scale an OM integrated with a CCD camera is enough with 30-100 frames/sec speed. OM can be used for submicron imaging using special techniques such as fluorescence labelling, and submicron imaging. Therefore SEM, AFM or STM is utilized where SEM has limitations in the sense of requiring a vacuum chamber, getting only a 2D image and works only for conducting and some semi-conducting objects.

Finally, the vision sensors can be classified as far-field or near-field sensors. Far-field sensors can get images of the manipulation tool and the manipulated object from another reference, while in the near-field case only the relative distance between the tool and object can be maintained during manipulation.

2.4 Nanosystems: Key Technologies

Strategies for nanomanipulations determine the "fingers" of nanomanipulation systems, presently, which generally include AFM cantilevers, nanotweezers, and lasers. Other parts of nanomanipulation generally include manipulators, sensing devices, control systems and a human-machine interface. Manipulators serve to position the end-effectors into the desired position, while microscopes are used for sensing the action and/or measuring the properties of the objects. Other devices served to facilitate the manipulations, e.g. a human-machine interface.

2.4.1 Nanomanipulators

There are mainly two families of positioning devices/nanomanipulators: the STM family (including STM, AFM, and other types of SPMs), and the robotic type.

- **STM family nanomanipulators**

From the time of their invention, STM and AFM or other type local probe microscopes had also been used as positioning devices. The advantage of this family is their incomparable resolution (tenths of an angstrom) for 3D surface topology observation. An STM can be used in air or in a vacuum for the manipulation of conductive or semi-conductive objects. An AFM can be used in any environment (in air, vacuum or liquid) for different kinds of objects.

- **Nanorobotic manipulators**

It is most significant to be able to manipulate nano scale objects in 3D space for constructing nano structures and devices. In order to undertake such manipulations, robotic manipulators with a multi-degree of freedom and nanometer scale resolutions, will be useful tools. The basic requirements for a nanorobotic manipulator for 3D manipulations include a nano-scale positioning resolution, a relatively large working space, enough DOFs for 3D positioning of the end effectors, and usually with multi-end-effectors for complex operations. However, such kinds of manipulators need microscopes as the real time observation system. Selectable microscopes include the SEM, TEM (Transmission Electron Microscope) or the OM. The vacuum chamber of a TEM is too narrow to be used for complex operations at present for the relatively large sizes of actuators, Skidmore [336] has succeeded in building a 3-DOF manipulator inside a TEM and it was cooperatively used with another manipulator inside a SEM with a transportable common substrate. The SEM is still the first choice for nanorobotic manipulators. The OM is seldom used, except in special situations.

2.4.2 Sensing Systems

The recent rapid advances in nanomanipulation are due in large part to the development of microscopes. Whether it is scanning probes, optical tweezers, high-resolution electron microscopes, or other new tools, instruments available to research workers in science and technology now permit them to create new structures, measure new phenomena, and explore new applications. To facilitate nanomanipulation, a whole range of imaging is useful for operators even when applying local probe type manipulators based on the STM or AFM. Guthold [160] is planning to insert their AFM into a SEM. An AFM cantilever with external measurement system (commonly using laser-photo diode in the AFM, or images of SEM in the robotic type [105][106], or cantilever built-in piezoresistive sensors [329], or independent strain gauges [145] for force sensing) provides useful information to avoid destruction of tools or samples. A micro tri-axial force sensor for a 3D bio-manipulation system also has been developed [9].

2.4.3 Control Systems

Since there is a lack of knowledge about the micro/nano-world, telemanipulation based on a master-slave method is still the only method to control the manipulators. For this reason, the

development of intelligent systems capable of automated molecular assembly is an open issue for study and development. Once considered an uncertainty which involves the nano-world resembling a complex environment, the most feasible sensing approach seems to be the use of nanorobotic sensor-based control instead of deterministic motion planning for a nanorobot conception [68]. A bilateral control system has been presented [10], and the use of collective robotics was emphasized for the development of intelligent control design [179][308] [13]. Collective robotics appears to be the more feasible methodology for a practical approach related to the appliance and development of a system capable of a massive automatic molecular manipulation for nanotechnology. The use of concepts derived from collective nanorobotics, and behavior control based on local rules, was also investigated for medicine dealing with a common goal to destroy malignant tissues in the human body [238].

The importance of such an issue is quite remarkable [67]. Therefore, the specification of a control paradigm to deal with the problems of uncertainty implicit in the nano-world, enables a fast massive automation of nanotechnology, and is one of the most challenging subjects in the control research community. For such advances, in so complex an issue, a cooperative interdisciplinary work team was demonstrated to be the best path to follow, and good achievements on the improvement of a new paradigm for the automation of a nanorobot control design should be developed [73][65]. Such a model will be forced to learn and to evolve even when problems occur in an unpredictable fashion. Thus non-deterministic approaches were demonstrated to be the best way to fulfil this kind of complex set of pre-programmed tasks, which are expected to address the rapid development in nanotechnology [74].

2.4.4 Human-Machine Interface

Guthold [160] tried to provide a SPM virtual-environment interface to improve interactivity. The system provided the virtual telepresence on the surface, scaled by a factor of about a million to one. The introduction of direct human-SPM interaction creates not only enhanced measurement capability (for instance, special transducers can provide a sense of touch to the nanomanipulator), but also an automated technology presaging nanofabrication and/or repair of nanostructures. A 3D bio-micromanipulation system integrated with a real-time virtual reality simulator was proposed [145]. The use of virtual reality and physically based simulation was argued [69] as the most efficient method, providing a fast development for the study of the nano-world kinematics. Computational systems should enable a better comprehension and a real time follow-up of phenomenon related to the nano-world [70][66].

2.5 A New Robotics Field

Going from the macro to the micro/nano-world, the main phenomenon is the reduction of the size of objects where the effect of length change is defined as a scaling effect. Figure 2.2 describes some major barriers to achieving the nano-world. Scaling effect for different physical and geometrical parameters can be seen in table 2.2 [368]. As observed from the table and figures, by decreasing the size, for example, inertial forces decrease with the power of 4, and angle/rotational information does not change. Considering these kinds of effects, following physical and other object property changes occur at the micro/nano scale due to the scaling effect:

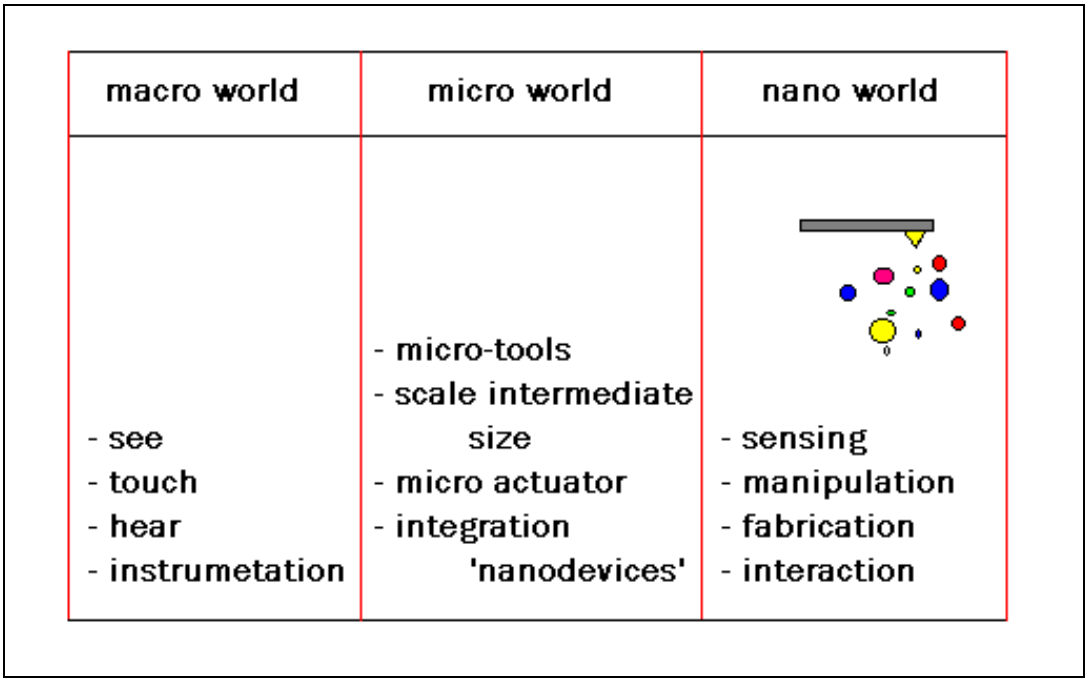

Figure 2.2: Barriers among macro, micro and nano-worlds.

- Other surface or adhesion forces such as van der Waals, electrostatic and capillary forces become dominant with respect to the inertial forces [126] [12]. As distinct from the macro scale object manipulation, when the objects to be gripped are in the nano-world, this dominance may result in a sticking effect.
- Going to a smaller nanometer or molecular scale, continuum mechanics is not valid. New physics utilizing electromechanical quantum mechanics and chemistry is necessary for exact analysis.
- Besides any applied load, frictional forces become affected by adhesion forces that differ from the macro world. Thus, a new definition of friction is needed for the nano scale.
- Resonant frequency increases with the length power of 1, which implies that the dynamics in the nano-world are very fast. Therefore, the approaches in quasi-static dynamics are more feasible for a teleoperated micro/nano manipulation control case.
- Specification increases in the sense of manipulation tasks such that sticking forces depend on the following aspects: object geometry and material type, object-gripper distance, environmental parameters such as temperature and humidity, and the environment type such as air, vacuum or liquid. Thus, manipulation, sensing, control and nano manipulator design strategy depend increasingly on the task specification and environment.
- Objects become more fragile and easily deformed in the case of imperfect shapes. Thus, sensor feedback is very important for uncertain objects and nano gripper shapes, and force-feedback is needed so as not to break or deform the objects and manipulation tools at the nano scale.

Parameter	Equation	Scaling Effect
Length	L	L
Surface area	αL^2	L^2
Volume	αL^3	L^3
Mass	ρV	L^3
Pressure	SP	L^2
Gravitational force	mg	L^3
Inertial force	$m\frac{d^2x}{dt^2}$	L^4
Viscosity force	$c\frac{S}{L}\frac{\partial x}{\partial t}$	L^2
Elastic force	$eS\frac{\Delta L}{L}$	L^2
Linear spring Constant	$\frac{2UV}{(\Delta L)^2}$	L
Eigen vibration frequency	$\sqrt{K/m}$	L^{-1}
Angular momentum	amr^2	L^5
Electrostatic force	$\frac{\varepsilon S}{2}\frac{V^2}{d^2}$	L^0
Electromagnetic force	$\frac{B}{2\mu}S_m$	L^2
Thermal expansion force	$eS\frac{\Delta L(T)}{L}$	L^2
Piezoelectric force	$eS\frac{\Delta L(E)}{L}$	L^2

Table 2.2: Scaling effects in the physical parameters.

- Rotational position is not affected by scaling, but the translational positioning is linearly scaled, which means that high precision linear positioning is necessary for the manipulation tool at the nano scale.

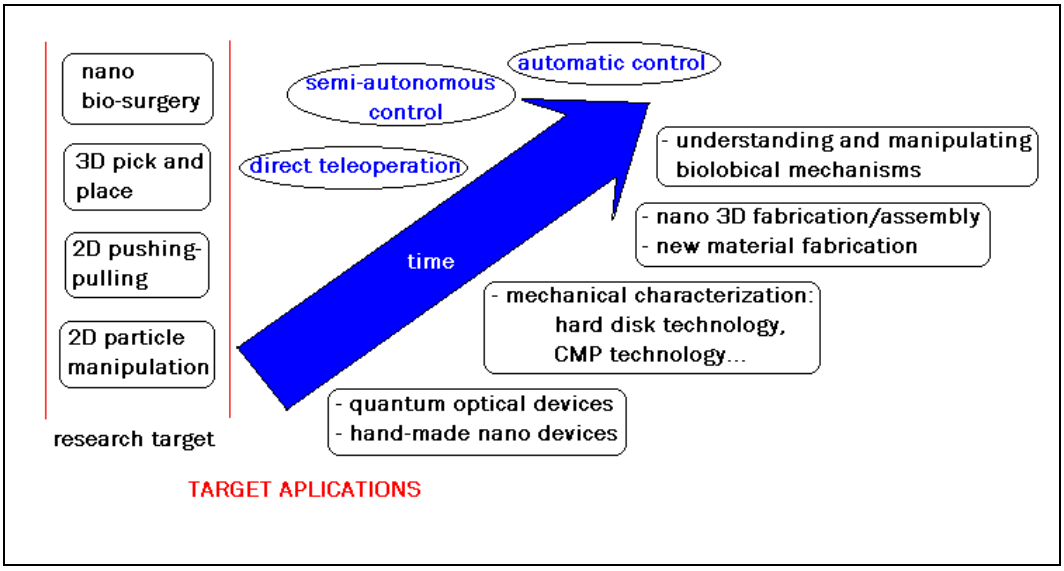

Figure 2.3: The target of the overall micro/nano manipulation project.

Taking the above points into consideration, micro/nano physics-based robotics and sensing with intelligent control is indispensable for nano manipulation. The following features are needed for the new robotics field:

- Design of nanoactuators taking into consideration the sticking effect and the specific task of reducing the sticking forces or controlling adhesive forces.
- Nanometer precision intelligent manipulation control with sensory feedback devices for quantum effects, nonlinear dynamics and disturbances. This should be addressed through teleoperated systems or real-time autonomous sensory feedback intelligent control.
- Robust and stable nonlinear controller design is required to attend to the nonlinearity and uncertainties at the actuators and the manipulation interactions.
- Cameras are replaced by nano physics-based microscopy.

At the molecular scale on molecular robotics cases, besides nano physics, chemistry has to be considered for controlled manipulation; thus nanorobotics requires interdisciplinary research from experts from different fields in a collaborative work. A general target for manipulation systems for nanotechnology development was established [331] (see Figure 2.3).

2.6 Nanomedicine

Molecular nanotechnology has been defined as the three dimensional positional control of molecular structures to create materials and devices with molecular precision. The human body is comprised of molecules, hence the availability of molecular nanotechnology will permit dramatic progress in human medical services. More than just an extension of "molecular medicine", nanomedicine will employ molecular machine systems to address medical problems, and will use molecular knowledge to maintain and improve human health at the molecular scale [142]. Nanomedicine will have extraordinary and far reaching implications for the medical profession, for the

definition of disease, for the diagnosis and treatment of medical conditions including ageing, for our very personal relationships with our own bodies, and ultimately for the improvement and extension of natural human biological structure and function.

The principal focus in medicine is going to shift from medical science to medical engineering. The design of medically active microscopic machines will be the consequent result of techniques provided from human molecular structure knowledge derived from the 20th and the beginning of the 21st century. For the feasibility of such achievements in nanomedicine two primary capabilities are required: fabrication of parts and assembly of parts. Through the use of different approaches such as biotechnology, supramolecular chemistry, and scanning probes, both capabilities had been demonstrated in a limited fashion as early as 1998 [142].

Despite quantum effects which impose a relative uncertainty of electron positions, such objections are resolved by recognizing that an atom has a predictable position due its nucleus having a large mass. The quantum probability function of electrons in atoms tends to drop off exponentially with distance outside the atom, giving atoms a moderately sharp "edge". Even in most liquids at their boiling points, each molecule is free to move only ~0.07 nm from its average position [151].

Many classical objections related to the feasibility of nanotechnology, such as quantum mechanics, thermal motions and friction, have already been considered and resolved [311] and discussions of techniques for manufacturing nanodevices are appearing in the literature with increasing frequency [140], [172]. Natural molecular machines could be found operating in living things: in the human body the closest similarity with a molecular machine is a ribosome. A ribosome acts as a general purpose factory building diverse varieties of proteins by bonding amino acids together in precise sequences under instructions encoded in the DNA [142]. Similarly a pre-established set of molecular manipulation tasks will be performed by the presented nanorobot with the task of protein delivery for organ inlets. Thereby the story of nanotechnology in medicine will be the story of achieving control at the molecular level, with nanorobots expected to serve as highly precise sensors and actuators for biomedical instrumentation. The easiest applications will be a system for diagnosis and health monitoring. More difficult applications will require that medical nanomachines perform surgical procedures and drug delivery.

One approach to nanomedicine would make use of microscopic mobile devices built using molecular-manufacturing equipment. Such molecular machines would either be biodegradable, self-collecting, or collected by something else once they had completed their work. Here, it is useful to think in terms of medical nanomachines that resemble small cells. Indeed a more recent estimation shows that future nanoprocessors will enable computers to be 100,000 times faster than any actual computers [208]. With their onboard nanobiosensors, they will be able to react to the same molecular signals that the immune system does, but with greater specification. Before being sent into the body on their mission, they should be programmed with a set of characteristics that allow them to distinguish their targets from everything else.

The advantages of the use of nanorobotics for medical problems are numerous [139]. We could mention briefly some of these as follows. A practical approach could be taken to exclude parasites, bacteria, viruses, and metastasizing cancer cells using medical nanorobots to limit the spread of blood borne diseases [141][142]. The establishment of a faster action against foreign antigens [93], thus greatly speeding up the natural immune system response to a large range of diseases. Eradication of most serious circulatory-related pathological conditions is possible, including all vascular disease, and heart disease, through the elimination of unconstrained metabolite and fluid circulation. Faster metabolite transport and distribution, significantly improves physical endurance and stamina [142] [143]. The direct control by the user of many hormonal- and neurochemical-mediated, and all blood-mediated, physiological responses will enable the extremely rapid detection of health problems [21].

2.7 Conclusion

As in any new field of technology composed of multidisciplinary sciences, nanotechnology has arisen in order to have an impact on our society's life style. The expectations are considerable and the motivation for this is the large range of possibilities expected to be achieved in the coming years. Among them, the possibilities that are expected in the field of medicine are exciting. With miniaturization, the new frontiers have become more challenging at in the same time more intriguing. With the development of smaller and smaller robots, and the nanomechatronics devices with manipulation at 3D nanoscale precision, the steps to achieve nanorobots have been considered [72] [92][243].

Many barriers have fallen recently, and many others are coming down [74][65]. The importance of a better understanding of nano-world mechanisms and behaviours is paramount. Computational systems can be used to improve interactivity at the nano scale, thus improving the developmental stages of medical nanorobotics. It should have a great impact on the fast development of nanotechnology, resulting in direct improvements to medicine.

A more suitable approach for such endeavours has been shown in the appliance of bottom-up and up-down joint efforts, where the concepts derived from both approaches point to a feasible path for the theoretical development of models capable of addressing the complexity of nano-worlds. Among other approaches, the use of computer graphics as a human interface that facilitates the interaction between the human and nano-worlds could be pointed out as a most justifiable way to identify the essence of nanomedicine instrumentation [70][66]. Nanotechnology is expected to enable us to employ molecular machine systems to address medical problems. Therefore, we should use scientific knowledge at the molecular scale to maintain and improve human health.

Chapter 3

Physically Based Simulation

3.1 Introduction

The study of non-penetrating rigid bodies in virtual reality using dynamic constrained simulation is a field of research in computer graphics that has an enormous impact on physically based simulation and a large range of works in this field have resulted in good outcomes [24][70][164][66]. Particularly in calculating the motion of many objects that move under changing constraints, and frequently collide, one of the key issues of the dynamic simulation method is the calculation of collision impulses between rigid bodies [23][302]. Depending on the type of applications, it is given many different names, such as interference detection, clash detection, intersection tests, etc. In robotics, an essential component of robot motion planning and collision avoidance is a geometric reasoning system which can detect potential contacts and determine the exact collision points between the robot manipulator and the obstacles in the workspace [29][62][344].

The problem of collision detection or contact determination between two or more objects is fundamental to computer animation, molecular modelling, computer simulated environments, and robot motion planning [59][176][273][294][295][360]. In mathematical terms the correlation between contact force and relative normal acceleration could be expressed as a linear programming task [25] [69][81][66], which allows the calcultation of the collision impulse that operates between rigid bodies colliding at multiple points. Furthermore the relation between the collision impulse and relative normal velocity also could be expressed as a linear complementary task.

A simple and fast algorithm was equally demonstrated for calculating the contact force with friction by formulating the relation between force and relative acceleration as a linear complementary task [22]. This model was based on Dantzig's algorithm for solving linear complementary problems, which is extended to systems with friction. Baraff's algorithm has achieved great performance for real-time and interactive simulation of two-dimensional mechanisms with contact force, friction force and collision impulse, although friction impulse at collision was not completely covered in such a model. Therefore a complementary algorithm was established covering the "impulse-based" aspects as well, which can trace in detail the change of friction force at a single colliding point by numerical integration of both the contact force and the friction force [266].

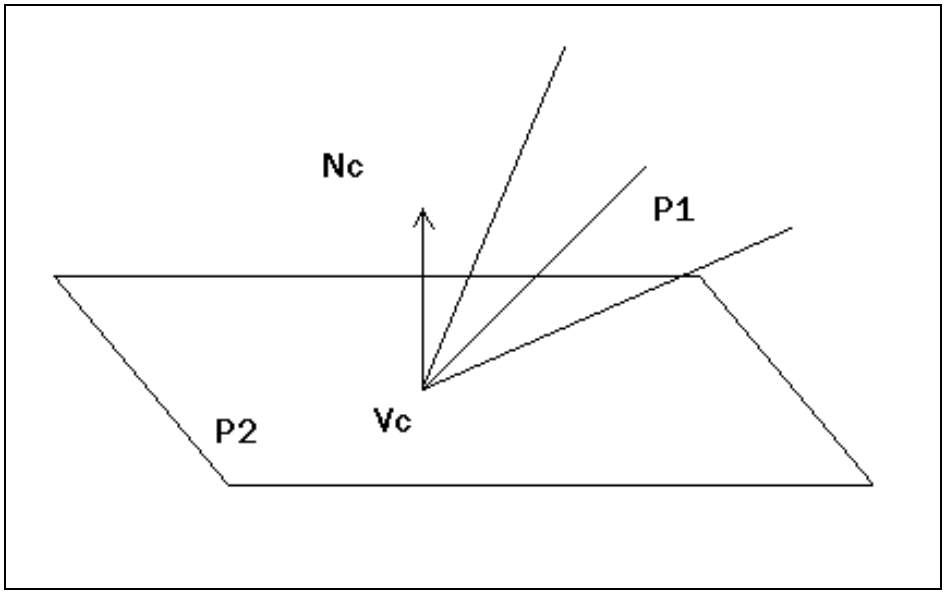

Figure 3.1: A point-plane contact between two polyhedra.

In the physical world, there are no perfectly planar faces or perfectly straight edges, and specifically at a nanoscopic level all contacts can be modelled as a composition of point contacts [70] [66]. Dynamic collision detection and non-penetrating constraint aspects were indispensable for model development once we decided to consider kinetics and frictional aspects required specially for molecular assembly manipulation and rigid body motion with hydrodynamics at low Reynolds numbers (Re) [79][142].

3.2 Representing Contacts

A contact in the $\Re^3$ space is represented as a finite conjunction set C of point- plane contact constraints. A typical point-plane constraint between two polyhedra is shown in Figure 3.1. The constraint c is defined by a vertex of contact v_c and the outward normal of the face n_c. A translation d of v_c rebord causes p_1 to penetrate p_2 at v_c exactly when $n_c^T d < 0$. The local motion ΔX causes a vertex v_c of P_1 to undergo a translation $d_c = J_c \Delta X$, where J_c is the constant 3×6 Jacobian matrix that relates the differential motion of P_1 to the motion of v_c. Thus ΔX causes P1 to penetrate P2 at v_c exactly when $n_c J_c \Delta X < 0$.

The presentation C of a contact between parts P_1 and P_2 is interpreted as follows. For each constraint $c \in C$, a local motion ΔX can relate to c in three ways, depending on the relative motion at the contacting point:

- Motion ΔX violates c if and only if $n_c^T J_c \Delta X < 0$. In other words, ΔX violates c exactly when part P_1 undergoing motion ΔX penetrates part P_2 at contact point v_c.
- Motion ΔX breaks c when $n_c^T J_c \Delta X > 0$. In this case, ΔX causes contact point v_c on P_1 to move in a local tangent to part P_2.

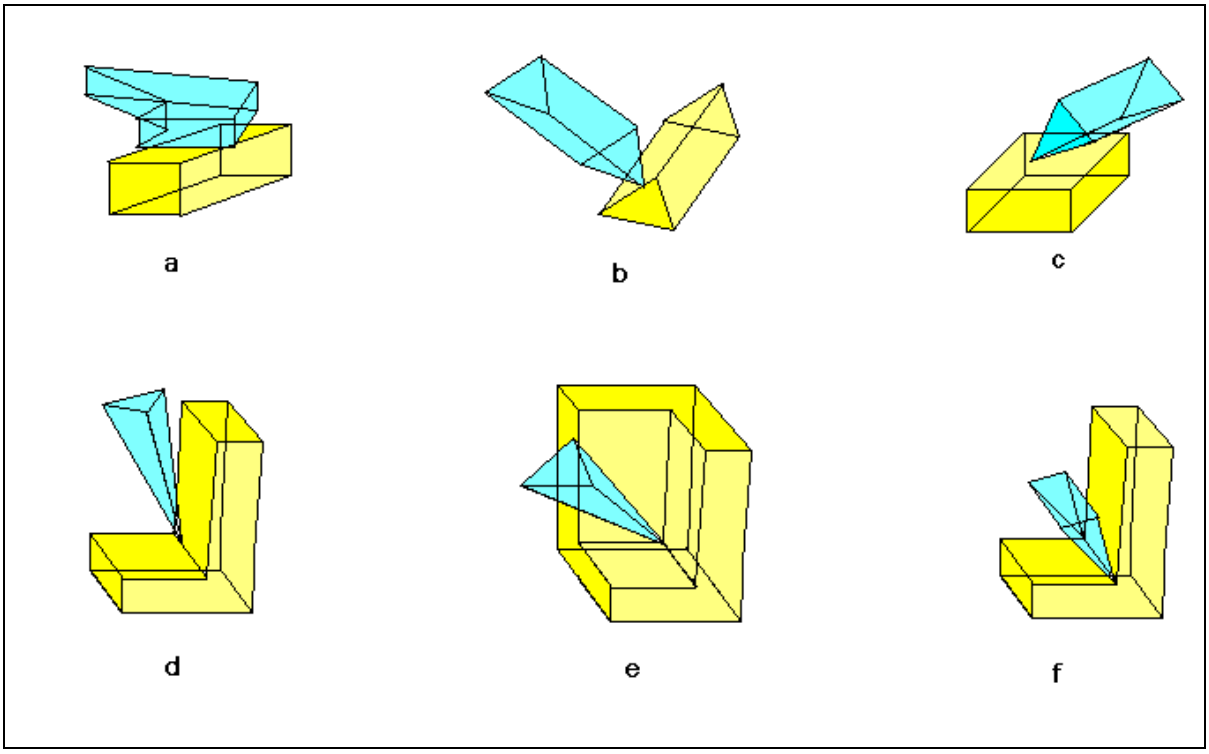

Figure 3.2: Contacts between polyhedra expressed as point-plane contacts.

- Motion ΔX slides on c when $n_c^T J_c \Delta X = 0$, i.e. when ΔX causes contact point v_c on P_1 to move in a local tangent to part P_2.

When ΔX breaks or slides on constraint c, we say that ΔX obeys c. The set of local motions that obey c form a closed half-space bounded by a hyperplane through the origin.

A local motion ΔX obey the contact C if and only if it obeys all constraints in C. C describes the contact between P_1 and P_2 if and only if the set of motions that obey C is equal to the local freedom of P_1 with respect to P_2. In this case the c local freedom of P_1 is given by the intersection of the closed half-spaces defined by the constraints $c \in C$. Note that the point-plane constraints representing a contact need not correspond to actual contact points on the parts; the only requirement is that the local motions that obey the contact be equal to the local freedom between the parts.

3.2.1 Polyhedral Contacts

The following types of contact between polyhedra can be described as sets of point-plane constraints:

Plane-point: the contact c between a planar face of P_1 with outward normal n_c and vertex v_c of P_2 is given by a constraint at v_c between a point of P_1 and plane of P_2 with outward normal $-n_c$.

Face-face: a contact between two polygonal planar faces is described by a set of point-plane constraints at the vertices of the convex hull of their contacting surface area (Figure 3.2a).

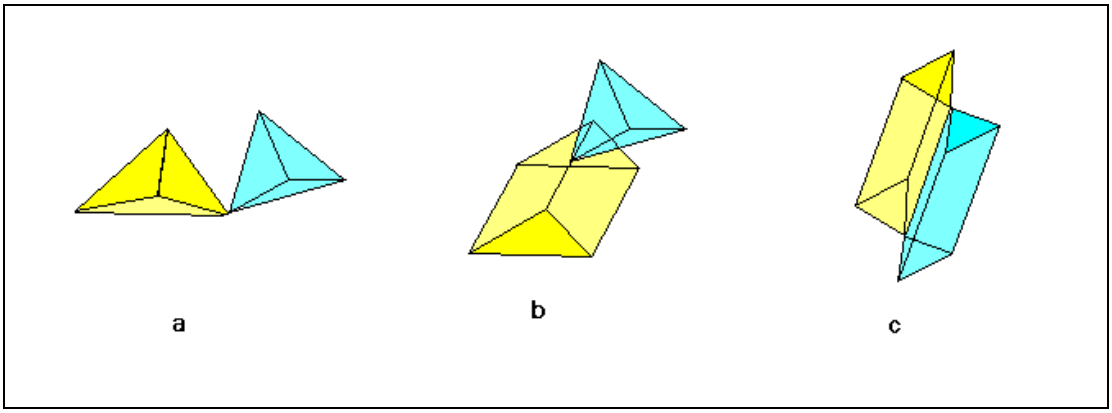

Figure 3.3: Polyhedral convex vertex-convex vertex, vertex-convex-edge, and aligned-convex-edges contacts.

Non-aligned convex edges: two convex edges touching at a point *p* are described by a point-plane constraint between *p* and the plane containing the two edges (Figure 3.2b).

Edge-face: A contact between a convex edge *e* and a planar face *f* is described by two point-plane constraints, one at each end of the intersection segment of *e* and *f* (Figure 3.2c).

Convex vertex-concave edge or vertex: if a convex vertex *v* is in contact with a concave edge or vertex, the constraint on local motion is equivalent to a set of point constraints between *v* and each of the faces meeting at the edge or vertex (Figures 3.3d and 3.3e).

Convex edge-concave edge: in a similar way, two edge-face contacts suffice to describe the constraint arising from a convex edge contacting a concave edge (Figure 3.2f).

A contact between two polyhedra that includes several of the contacts described above can be expressed as a set of constraints *C*, where *C* is the union of sets C_i each representing one of the above simple contacts. The remaining possible contacts between polyhedra are convex vertex-convex vertex, vertex-convex-edge, and aligned-convex-edges contacts (Figure 3.3). All but aligned-convex-edges contacts can be treated using finite disjunctions of point-plane constraints [374][177]. For instance, in Figure 3.3a, the motion of the contact vertex v_c on P_1 must obey at least one of the point-plane constraints between v_c and the planes of P_2 that meet at v_c.

3.2.2 Nonpolyhedral Contacts

In addition to the above contacts between polyhedra, several common contact types in non-polyhedral 3D-objects can be expressed in terms of point-plane constraints:

Cylinder-face: a cylinder contacting a plane in a line segment is equivalent for local motion purposes to an edge-plane contact along the contact line segment (Figure 3.4a).

Cylinder-cylinder: a round peg in a round hole has the same local freedom as a round peg in a triangular hole. Thus a cylinder-cylinder contact can be described as three cylinder-plane contacts, i.e. six point-plane contacts (Figure 3.4b).

Threaded cylinders: a contact between two threaded cylinders can be expressed as shown in Figure 3.4c. A cylinder-cylinder contact is combined with two point-plane contacts that together express the twisting constraint of the threads at a single point. The normals of the two thread contact

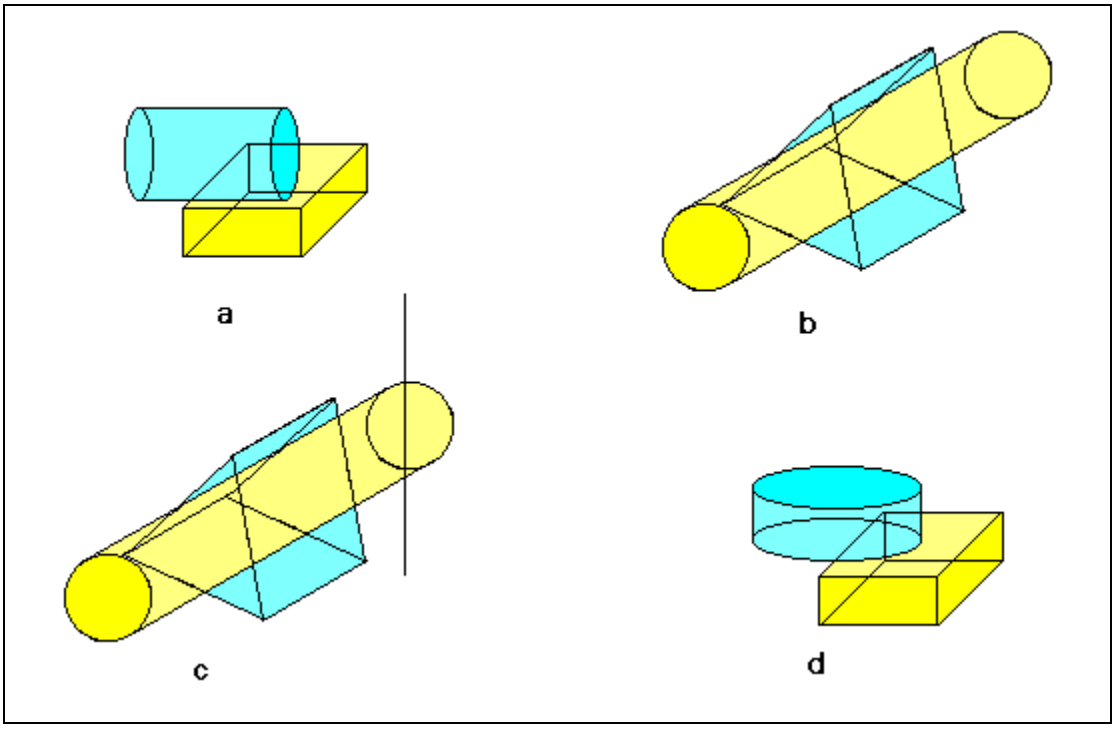

Figure 3.4: Typical non-polyhedral contacts.

constraints are opposing and have angle $\alpha = \arctan\frac{1}{p}$ with the axis of the cylinder, where p is the pitch of the threads. In the procedure to calculate local freedom described below, this representation of threaded contacts is never needed because motion is so strongly constrained.

Although products often have complicated, curved surface shapes, the great majority of contacts between parts fall into the cylindrical, planar, and threaded types above. The main exception is when the convex hull of the contact area of a face-face contact is not polygonal (Figure 3.4d). Such a contact is equivalent to an infinite number of point-plane contacts around the convex hull. A polygonal approximation of the convex hull allows such a contact to be described with some loss of accuracy.

Non-contacting surfaces of parts are often curved to satisfy requirements such as strength, aerodynamics, and aesthetics. Thus reasoning about curved surfaces is more important when extended motions are considered, since these surfaces may interfere with each other. In such cases approximate methods are often better suited [178].

3.3 Physically Based Simulation

Basically the problem of collision detection corresponds to determining whether there is any contact between two objects. We can express the exact conditions for dynamically contact forces as a vector C of contact force magnitude, which is correct if it satisfies some of the basic conditions discussed next.

Thus the main aspects for collision detection are described as follows. There is no object interpenetration through contact forces for a rigid body, and any contact force can only push any related object. The contact force could not be used to pull any 3D object, it affects just the contact points and nothing else. For dynamic collision detection the contact force expresses a continuous

behaviour related to the function of time. Such observations are necessary for any correct contact force function that intends to produce a dynamically correct motion.

It is possible to have a multiple correct contact force and when some similar circumstances arises the right solution is given using an equation of compatibility, that is precluded by the rigid body modelling. Nevertheless any correct result provided by the contact force *C* results in the same correct motion [25]. The motion of a rigid body subject to external forces is described by the Newton-Euler motion equations as follows:

$$\dot{v} = \frac{1}{m} \sum_{i=1}^{k} f(C)_i \tag{3.1}$$

$$\dot{w} = I^{-1} \left(\sum_{i=1}^{k} C_i \times f(C)_i - w \times Iw \right) \tag{3.2}$$

where $\dot{v}$ is the dotted velocity vector, $\dot{w}$ is the dotted normal contact distance vector, $f(C)_i$ are the external forces (including contact forces), C_i is the vector which points from the center of mass to the points where the force is applied, **I** denotes the inertia tensor, and *m* the object mass. We are interested to verify when the objects begin their motion if there is any contact between the objects. For rigid body simulation there are two types of contacts [61] that we could identify as tangential collision and boundary collision.

Tangential Collisions: this corresponds to a tangential intersection between two surfaces at a geometric contact point. The contact point lies in the interior of each surface and the normal vectors at that point are collinear. Equation 3.3 expresses a tangential intersection.

$$E(s,t) = P(u,v) \tag{3.3}$$

$$(E_s(s,t) \times E_t(s,t)) \bullet P_u(u,v) = 0 \tag{3.4}$$

$$(E_s(s,t) \times E_t(s,t)) \bullet P_v(u,v) = 0 \tag{3.5}$$

with E(s,t) and P(u,v) representing two parametric surfaces, we assume that the Bézier surface has an algebraic formulation in homogeneous coordinates as:

$$E(s,t) = (X(s,t), Y(s,t), Z(s,t), W(s,t)) \tag{3.6}$$

$$P(u,v) = (\overline{X}(u,v), \overline{Y}(u,v), \overline{Z}(u,v), \overline{W}(u,v)) \tag{3.7}$$

where E_s, E_t, P_u, P_v correspond to the partial derivatives and $\bullet$ corresponds to the dot product. Equation 3.3 corresponds to a contact between the two surfaces; equation 3.4 and 3.5 represent the fact that their normals are collinear. They are expressed as scalar triple products of the vector. This is an over constrained system and has a solution only when the two surfaces are touching each other tangentially. For such equations, after cross multiplication we get 3 polynomial equations of degree *2n* each. The dot product results in the addition of degrees of the numerator polynomials. Similarly for two algebraic surfaces, the problem of tangential intersection can be formulated as:

$$e(x,y,z) = p(x,y,z) = 0 \tag{3.8}$$

with

$$\begin{pmatrix} e_x(x,y,z) \\ e_y(x,y,z) \\ e_z(x,y,z) \end{pmatrix} = \theta \begin{pmatrix} p_x(x,y,z) \\ p_y(x,y,z) \\ p_z(x,y,z) \end{pmatrix} \tag{3.9}$$

Equation 3.8 and 3.9 correspond equally to an over constrained system.

Boundary Collisions: this intersection lies on the boundary curve of one of two surfaces. Thus given a Bézier surface, defined over the domain, $(s,t) \in [0,1] \times [0,1]$, we obtain the boundary curves by substituting *s* or *t* to be 0 or 1. Hence, the problem is reduced into the following equation:

$$E(s,1) = P(u,v) \tag{3.10}$$

Two objects collide if equations 3.3 or 3.10 for parametric surfaces and the 3.3 for algebraic surfaces have a common solution in their domain. Instead of a Lagrangian model, which complicates the calculations exponentially as the complexity of the system increases, the Newton-Euler method was chosen to derive system differential equations. Thus linear state feedback could be used for motion kinematics modelling.

3.3.1 Timely Dynamic Collision

Given a trajectory that each moving object will travel, we can determine the exact collision time [64]. It will require the computation of the initial separation and the possible collision time among all pairs of objects and the obstacles, assuming that the magnitude of relative initial velocity, relative maximum acceleration and velocity limits are given. If the path that each object travels is known in advance, then we can calculate a lower bound on collision time. This bound on collision time is calculated adaptively to speed up the performance of dynamic collision detection.

Let $a_{\max}$ be an upper bound on the relative acceleration between any two points on any pair of objects. The bound $a_{\max}$ can be easily obtained from bounds on the relative absolute linear $\vec{a}_{lin}$ and relative rotational accelerations $\vec{a}_{rot}$ and relative rotational velocities $\vec{w}_r$ of the bodies and their diameters: $\vec{a}_{\max} = \vec{a}_{lin} + \vec{a}_{rot} \times \vec{r} + \vec{w}_r \times (\vec{w}_r \times \vec{r})$ where *r* is the vector difference between the centers of mass of two bodies. Let *d* be the initial separation for a given pair of objects, and v_i (where $\vec{v}_i = \vec{v}_{lin} + \vec{w}_r \times \vec{r}$) the initial relative velocity of the closest points on these objects. Then we can bound the time t_c to collision as

$$t_c = \frac{\sqrt{v_i^2 + 2a_{\max}d} - v_i}{a_{\max}} \tag{3.11}$$

This is the minimum safe time that is added to the current time to give the wake-up time for this pair of objects. To avoid a "Zeno's paradox" condition [174] where smaller and smaller times are added and the collision is never reached, we must add a lower bound to the time increment. So rather

than just adding t_c as derived above, we added $t_w = \max(t_c, t_{\min})$, where $t_{\min}$ is a constant, e.g. 1m Sec or $\frac{1}{FrameRate}$ which determines the effective time resolution of the calculation.

As a side note here, we would like to mention the fact that since we can calculate the lower bound on collision time adaptively, we can give a fairly good estimate of exact collision time to the precision in magnitude of $t_{\min}$. In addition, since the lower bound on time to collision is calculated adaptively for the object most likely to collide first, it is impossible for the algorithm to fail to detect an interpenetration. This can be done by modifying $t_{\min}$, the effective time resolution, and the user defined safety tolerance ε according to the environment so as to avoid the case where one object collides with the other between time steps. The parameter ε is used because the polyhedra may be actually shrunk by ε amount to approximate the actual object. Therefore, the collision should be declared when the distance between two convex polytopes is less than 2ε. This is done to ensure that we can always report a collision or near-misses.

3.4 Collision Detection for Bounding Volumes

To compute exact collision contacts, using a bounding volume for an interference test is not sufficient, but it is rather efficient for eliminating those object pairs that are of no immediate interest from the point of collision detection. The bounding box can either be spherical or rectangular, or even elliptical, depending on the application and the environment. We prefer spherical and rectangular volumes due to their simplicity and suitability for implementation.

Consider an environment where most of the objects are elongated and only a few objects, probably just the robots in most situations, are moving, then rectangular bounding boxes are preferable. In a more dynamic environment like a vibrating parts feeder where all objects are rather "fat" [290] and bouncing around, then spherical bounding boxes are more desirable for such objects, e.g. the biomolecules. If the objects are concave or articulated, then a subpart-hierarchical bounding box representation (similar to subpart-hierarchical tree representation, with each node storing a bounding box) should be employed. The reasons for using each type of the bounding volumes are as follows.

Using a spherical bounding volume, we can pre-compute the box during the pre-processing step. At each step, we only update the center of each spherical volume and get the minimum and maximum x, y, z coordinates almost instantaneously by subtracting the measurement of radius from the coordinates of the center. This involves only one vector-matrix multiplication and six simple arithmetic operations: 3 additions and 3 subtractions.

However, if the objects are rather oblong, then a sphere is a rather poor bonding volume to use. Therefore, a rectangular bounding box is a better choice for elongated objects. To impose a virtual rectangular bounding volume on an object rotating and translating in space, involves a recomputation of the rectangular bounding volume. Recomputing a rectangular bounding volume is done by updating the maximum and minimum x, y, z coordinates at each time instance. This is a simple procedure that can be done at constant time for each body. We can update the "min" and "max" using the following approaches.

Since the bounding boxes are convex, the maximum and minimum in the x, y, z coordinates must be the coordinates of vertices. We can set up 6 imaginary boundary walls, each of these walls is located at the maximal and minimal x, y, z coordinates possible in the environment. Given the previous bounding volume, we can update each vertex of the bounding volume by performing only

half of the modified vertex-face tests, since all the vertices are always in the Voronoi regions of these boundary walls. We first find the nearest point on the boundary wall to the previous vertex, after motion transformation, on the bounding volume, then we verify if the nearest point on the boundary wall lies inside the Voronoi region of the previous vertex. If so, the previous vertex is still the extreme point, minimum or maximum in x, y, or z-axis. When a constraint is violated by the nearest point on the face (wall) to the previous extreme vertex, the next feature returned will be the neighbouring vertex, instead of the neighbouring edge. This is a simple modification to the point-vertex applicability criterion routine. It still preserves the properties of locality and coherence well. This approach of recomputing the bounding volume dynamically involves 6 point-vertex applicability tests.

Another simple method can be used based on the property of convexity. At each time step, all we need to do is to check if the current minimum or maximum vertex in x, y, or z coordinate still has the smallest or largest x, y, or z coordinates values in comparison to its neighbouring vertices. By performing this verification process recursively, we can recompute the bounding boxes at an expected constant rate. Once again, we are exploiting the temporal and geometric coherence and the locality of convex polytopes. Obviously we must update only the bounding volumes for the moving objects.

We speed up the performance of this approach by realising that we are only interested in one coordinate value of vertices for each update, say x coordinate while updating the minimum or maximum value in x-axis. Therefore, there is no need to transform the other coordinates, say y and z values, in updating the x extremal vertices during the comparison with neighbouring vertices. Therefore, we only need to perform 24 vector-vector multiplications. 24 comes from 6 updates in minimum and maximum in x, y, z coordinates and each update involves 4 vector-vector multiplications, assuming each vertex has 3 neighbouring vertices.

For concave and articulated bodies, we need to use a hierarchical bounding box structure, i.e. a tree of bounding boxes. Before the top level bounding boxes collide, there is no need to impose a bounding volume on each subpart or each link. Once the collision occurs between the parent bounding boxes, then we compute the bounding boxes for each child (subpart or linkage). At last we would like to briefly mention that in order to report “near-misses”, we should “grow” the bounding boxes by a small amount to ensure that we perform the exact collision detection algorithm when two objects are about to collide, not after they collide. Given the details of computing bounding volume dynamically, we will describe briefly some current algorithms that are applied for collision detection with their particularities, suitable applicability and relative computational effort required for each one.

3.4.1 Interval Tree for 2D Intersection Tests

Another approach is to extend the one-dimensional sorting and sweeping technique to higher dimensional space. However, as mentioned earlier, the time bound will be worse than $O(n)$ for two or three-dimensional sort and sweep due to the difficulty in making a tree structure dynamic and flexible for quick insertion and deletion of a higher dimensional box. Nevertheless, for more dense or special environments, such as a mobile robot moving around in a room cluttered with moving obstacles, such as biomolecules, it is more efficient to use an interval tree for 2-dimensional intersection tests to reduce the number of pairwise checks for overlapping. We can significantly reduce the extra effort in verifying the exchanges checked by the one-dimensional sort and sweep. Here we will briefly describe the data structure of an interval tree and how we use it for intersection testing of 2-dimensional rectangular boxes.

An interval tree is actually a range tree properly annotated at the nodes for fast search of real intervals. Assume that *n* intervals are given, as $[b_1, e_1], \cdots, [b_n, e_n]$ where b_i and e_i are the endpoints

of the interval as defined above. The range tree is constructed by first sorting all the endpoints into a list $x_1,\cdots,x_m$ in ascending order, where $m \le 2n$. Then, we construct the range tree top-down by splitting the sort list *L* into the left subtree L_l and the right subtree L_r, where $L_l = (x_1,\cdots,x_p)$ and $L_r = (x_{p+1},\cdots,x_m)$. The root has the split value $\frac{x_p + x_p + 1}{2}$. We construct the subtrees within each subtree recursively in this fashion till each leaf contains only an endpoint. The construction of the range tree for *n* intervals takes $O(n \log n)$ time. After we construct the range tree, we further link all nodes containing stored intervals in a doubly linked list and annotate each node if it or any of its descendants contain stored intervals. The embellished tree is called the *interval tree*.

We can use the interval tree for static query, as well as for the rectangle intersection problem. To check for rectangle intersections using the sweep algorithm, we take a sweeping line parallel to the *y*-axis and sweep in increasing *x* direction, and look for overlapping *y* intervals. As we sweep across the *x*-axis, *y* intervals appear or disappear. Whenever there is an appearing *y* interval, we check to see if the new interval intersects the old set of intervals stored in the interval tree, report all intervals it intersects as rectangle intersections, and add the new interval to the tree.

Each query of interval intersection takes $O(\log n + k)$ time where *k* is the number of reported intersection and *n* is the number of intervals. Therefore, reporting intersections among *n* rectangles can be done in $O(n \log n + k)$ where *K* is the total number of intersecting rectangles.

3.4.2 One-Dimensional Sort and Sweep

This approach is especially suitable for an environment where only a few objects are moving while most of the objects are stationary, e.g. a virtual walk-through environment. In computational geometry, there are several algorithms that can solve the overlapping problem for *d*-dimensional bounding boxes in $O(n \log^{d-1} n + s)$ time where *s* is the number of pairwise overlaps [116][182] [334]. This bound can be improved using coherence. Let a one-dimensional bounding box be [*b,e*] where *b* and *e* are the real numbers representing the beginning and ending points. To determine all pairs of overlapping intervals given a list of *n* intervals, we need to verify for all pairs *i* and *j* if $b_i \in [b_j, e_j]$ or $b_i \in [b_i, e_i]$, $1 \le i < j \le n$. This can be solved by first sorting a list of all b_i and e_i values, from the lowest to the highest. Then, the list is traversed to find all the intervals that overlap. The sorting process takes $O(n \log n)$ and $O(n)$ to sweep through a sorted list and $O(s)$ to output each overlap where *s* is the number of the overlap.

For a sparse and dynamic environment, we do not anticipate that each body will make a relatively large movement between time steps, thus the sorted list should not change much. Consequently the previous sorted list would be a good starting point to continue. To sort a "nearly sorted" list by *bubble sort* or *insertion sort* can be done in $O(n + e)$ where *e* is the number of exchanges.

All we need to do now is to keep track of "status change", i.e. from overlapping in the last time step to non-overlapping in the current time step and vice versa. We keep a list of overlapping intervals at all times and update it whenever there is a status change. This can be done in $O(n + e_x + e_y + e_z)$ time, where e_x, e_y, e_z are the number of exchanges along the *x*, *y*, *z* coordinate. Though the update

can be done in linear time, e_x, e_y, e_z can be $O(n^2)$ with an extremely small constant. Therefore, the expected run time is linear in the total number of vertices.

To use this approach in a three-dimensional workspace, we pre-sort the minimum and maximum values of each object along the *x*, *y*, *z* axis, as if we are imposing a virtual bounding box hierarchy on each body, sweep through each nearly sorted list every time step and update the list of overlapping intervals as we mentioned before. If the environment is sparse and the motions between time frames are "smooth", we expect the extra effort to check for collisions will be negligible. This "pre-filtering" process to eliminate the pairs of objects not likely to collide will run essentially in linear time. A similar approach has been mentioned by Baraff [23].

3.4.3 Uniform Spatial Subdivision

We can divide the space into unit cells (volumes) and place each object (bounding box) in some cell(s) [290]. To check for collisions, we have to examine the cell(s) occupied by each box to verify if the cells are shared by other objects. But, it is difficult to set a near-optimal size for each cell and it requires a tremendous amount of allocated memory. If the size of the cell is not properly chosen, the computation can be rather expensive. For an environment where almost all objects are of uniform size, like a vibrating parts feeder bowl or molecular modelling [360] [290], this is a rather ideal algorithm, especially to run on a parallel-computing machine. In fact, Overmars has shown that using a hash table to look up an entry, we can use a data structure of $O(n)$ storage space to perform the point location queries in constant time [290].

3.4.4 BSP-Trees and Octrees

One of the commonly used tree structure is the BSP-tree, Binary Space Partitioning tree, to speed up intersection tests in CSG Constructive Solid Geometry [356]. This approach constructs a tree from separating planes at each node recursively. It partitions each object into groups of parts that are close together in binary space. When the separation planes are chosen to be aligned with the coordinate axes, then a BSP tree becomes more or less like an octree.

One can think of an octree as a tree of cubes within cubes. But the size of the cube varies depending on the number of objects occupying that region. A sparsely populated region is covered by one large cube, while a densely occupied region is divided into smaller cubes. Each cube can be divided into 8 smaller cubes if necessary. So each node in the tree has 8 children (leaves).

Another modified version of the BSP-Tree proposed by Vanecek [364] is a multidimensional space-partitioning tree called Brep-Index. This tree structure is used for collision detection [45] between moving objects in a system called Proxima developed at Purdue University.

The problem with tree structures is that its update, insertion and deletion, is inflexible and cumbersome, especially for a large tree. The overhead of insertion and deletion of a node in a tree can easily dominate the run time, especially when a collision occurs. The tree structures also cannot capture the temporal and spatial coherence well.

3.5 Conclusion

The use of physically based simulation is a branch of computer graphics that has a great range of applications in different fields. For nanotechnology its impact could be of great importance for the nanoCAD design of new devices at nanoscale [70][66]. With the application of virtual reality and dynamic collision detection many aspects for a better comprehension of kinetics features in the nano-world could be improved. The representation of diverse contacts in 3D are easily accompanied with the use of a virtual reality approach, where the exact conditions for dynamically contacted forces could be expressed and satisfied in a precise fashion. Although there are different methodologies proposed for collision detection with bounding volumes, we have chosen the *Interval Tree for 2D Intersection*, this is the most suitable approach for the complexity of environments with different objects and robots moving all around the workspace, once its reduced number of pairwise checks improve the simulator performance. The field of computer graphics is providing a unique opportunity to contribute to a new field of science that promises to revolutionise our society [65]. The possibilities that are open for the new field of nanotechnology are impressive and computer graphics will play a decisive role in the fast development of nanotechnology [73][66].

Chapter 4

Motion Control

4.1 Introduction

Motion planning algorithms were initially developed in the context of robotic systems. Such algorithms process motion given a high-level goal and a geometric description of the objects involved [230]. In the context of computer animation, motion planning can be used to effectively compute primary intentional motions. Examples include computing collision-free motions to accomplish high-level navigation or object manipulation tasks [219][19][226], or connecting different body configurations [86][44]. Motion planning is particularly suited to such tasks, since there is a near infinite number of possible goal locations and obstacle arrangements in the environment. This combinatorial explosion of possibilities currently prohibits the direct use of pre-recorded motion sequences. Instead, flexible and efficient planning algorithms can be designed to compute collision-free motions for specific categories of task commands that apply to a broader set of situations.

Indeed path planning problems arise in such diverse fields as robotics [71], assembly analysis [42][65], virtual prototyping [69][66], pharmaceutical drug design [210], manufacturing [20], and computer animation [226]. Such problems involve searching the systems configuration space for a collision-free path connecting a given start and goal configuration [230]. Randomized algorithms for path planning have enjoyed success and popularity in the last several years due to their efficiency in handling problems with many degrees of freedom [26]. The randomized path planner of Barraquand and Latombe [28] was an early attempt to practically solve problems with high-dimensional configuration spaces. Their search technique alternated between following the gradient of an artificial potential field [215], and utilizing random walks in order to escape the basin of attraction of any local minima encountered. Variations of this planner were used to solve complex single and multi-arm manipulation tasks [218][219][359]. Unfortunately, pathological cases involving local minima can exist such that the probability of escaping them via random walk is extremely small [83][211].

In order to avoid the problems of local minima inherent with artificial potential fields, new sampling strategies were devised [212]. *Probabilistic roadmap* methods build a network of randomly-sampled free configurations in the configuration space [290][213]. After the network (roadmap) has been built, a path may be sought for using standard graph-search algorithms [349].

Theoretical results show that with a reasonable resolution of the geometry of the configuration space, a relatively small roadmap can correctly capture the connectivity of the free space with a high probability [189]. More precisely, the probability that a roadmap incompletely represents the connectivity of the free space decreases exponentially with the number of sample points in the roadmap. The main issue affecting coverage of the free space is the presence of narrow passages in the configuration space [188].

There are numerous variations on the basic roadmap strategy, most of which rely on different sampling techniques in an effort to reduce the computational costs [186][4]. The computed roadmaps are especially suitable when multiple path-planning queries are given for a robot in the same static environment, since searching a roadmap is very fast [188]. However, the overhead associated with building the roadmap is often too large for single-query planning problems in interactive environments [43].

Hsu developed a variant of the probabilistic roadmap planner that is better suited for solving single-query path planning problems [189]. It avoids the initial cost of preprocessing by incrementally building two trees respectively rooted at the start and the goal. As the trees grow, the planner periodically attempts to join them together to form a path. The idea of constructing search trees from the initial and goal configurations comes from classical Artificial Intelligence (AI) bidirectional search, and its use in motion planning methods [193]. In order to avoid oversampling any region of the configuration space, in such a method the planner incrementally samples around nodes in the trees according to a weighted probability that favors nodes that have few neighbors.

LaValle recently introduced the concept of Rapidly-exploring Random Trees (RRTs), a randomized sampling scheme originally designed for nonholonomic motion planning [232]. RRTs have also been applied to kinodynamic planning problems in configuration spaces of up to 12 dimensions. For both holonomic and nonholonomic planning, the sampling technique exhibits several desirable properties. Similar to the planner, the goal is to incrementally build a tree of free configurations in such a way that the expansion of the tree is heavily biased towards the unexplored regions of the space [189]. Due to the way that RRTs are constructed, the distribution of samples eventually converges toward a uniform distribution over C_{free} [232]. Further approaches were also proposed [71], and among others we could briefly mention for example genetic algorithms, tabu search, simulated annealing, neural networks that were applied successfully to the problem of motion control [65]. A mathematical description of the basic aspects related to the problem of motion control is detailed next.

4.2 Motion Control Description

Basic motion planning can be characterized as an optimal control problem. In this section it is important to note that we are not using control theory to study such issues as robot dynamics, controllability, or stability, but simply to recast the basic motion planning problem. Optimal control theory is a vast subject, and only some key definitions are provided here. A more thorough introduction to optimal control theory can be found in [56][228]. Let $X \subseteq \Re^n$ represent a state space in which $x_0 \in X$ represents the initial state of a system. Let $u : [0, t_f] \to \Re^n$ represent a *control function* in which $[0, t_f]$ represents an interval of time. The control at time *t* is given by *u(t)*, and the system state at time *t* is given by *x(t)*. The system equation can be represented as $\dot{x} = f(x(t), u(t))$,

which defines how the state will evolve over time. A *loss functional* is defined that evaluates any state trajectory and control function as follows:

$$L(x(\cdot),u(\cdot)) = \int_0^{t_f} l(x(t),u(t))dt + Q(x(t_f)) \tag{4.1}$$

The integrand *l(x(t),u(t))* represents an instantaneous cost, which when integrated can be imagined as the total amount of energy that is expended. The term $Q(x(t_f))$ is a final cost that can be used to induce a preference over trajectories that terminate in a particular portion of the state space by penalizing the final state of the system. One can also take $t_f = \infty$ and describe an asymptotic final state $x(\infty)$. Suppose that the initial state, x_0, is given. The optimal control design task is to select a control function $u(\cdot)$ that causes equation 4.1 to be minimized.

Considering that the basic motion planning could be expressed as a problem to compute a path $\tau[0,1] \rightarrow C_{free}$ such that $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$, when such a path exists, and that the natural choice for the state space as $X = C_{free}$. Furthermore if robot dynamics were also included in the problem specification, then *X* might be expanded to include time derivatives on the configuration space. Next we define a simple system equation, $f(x(t),u(t)) = u(t)$ for all *t*. This is not intended to be the most specific model of a particular robotic system, but rather it is used to encode the basic motion planning problem. We can assume for all *t* that the control input is either normalized, $\|u(t)\| = 1$, or $u(t) = 0$. The initial state of the system is fixed, $\tau(0) = q_{init}$. The loss functional can be simplified to $L(x(\cdot),u(\cdot)) = Q(x(t_f))$. We take $Q(x(t_f)) = 0$ if $x(t_f) = q_{goal}$, and $Q(x(t_f)) = 1$ otherwise. Thus our modelling partitioned the space of admissible controls into two classes: control functions that cause the basic motion planning problem to be solved receiving zero loss; otherwise, unit loss is received.

The motion planning problem requires a collision-free path. This can be obtained by mapping the space of control functions into the space of state trajectories for well-known obstacles. For a given *u(t), t > 0* and x_0, a state trajectory $x_u(t)$, $t > 0$ can be completely predicted. If $L(x_u(\cdot),u(\cdot)) = 0$, then the determined state trajectory is a solution to the basic motion planning problem, which can be expressed as $\tau(s) = x_u(st_f)$.

The previous formulation considered all control inputs that achieve the goal to be equivalent. By changing the loss functional, the optimal-path-length motion planning problem can be formulated:

$$L(x(\cdot),u(\cdot)) = \begin{cases} x(t_f) = q_{goal} \Rightarrow & \int_0^{t_f} \|\dot{x}\| dt \\ otherwise \Rightarrow & \infty \end{cases} \tag{4.2}$$

The term $\int_0^{t_f} \|\dot{x}\| dt$ measures the path length, and recall that $\dot{x}(t) = u(t)$ for all *t*. It is well known that the optimal path generally maps into the closure of C_{free} [230]. Because C_{free} is open, we define the state space for this case to be $X = C_{valid}$. A variety of other possibilities exist for defining the loss functional.

The motion planning problem can alternatively be characterized in discrete time. The discrete-time representations can simplify the development of computational methods. With the discretization of time, $[0,t_f]$ is partitioned into *stages*, denoted by $k \in \{1,...,K+1\}$. Stage k refers to time $(k-1)\Delta t$. If t_f is finite, the final stage is given by $K = \left\lfloor \frac{t_f}{\Delta t} \right\rfloor$. Let x_k represent the state at stage k. At each stage k, an *action* u_k can be chosen. Because

$$\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{x(t+\Delta t) - x(t)}{\Delta t} \tag{4.3}$$

the state transition equation can be approximated as

$$x_{k+1} = x_k + u_k \tag{4.4}$$

As an example of how this representation approximates the basic motion planning problem, consider the following example. Suppose $C_{free} \subseteq \Re^2$. It is assumed that $\|u_k\| = 1$ and, hence, the space of possible actions can be sufficiently characterized by the parameter $\Phi_k \in [0,2\pi]$. The discrete-time state transition equation becomes

$$x_{k+1} = x_k + \Delta t \begin{bmatrix} \cos(\Phi_k) \\ \sin(\Phi_k) \end{bmatrix} \tag{4.5}$$

At each stage, the direction of motion is controlled by selecting Φ_k. Any K-segment polygonal curve of length $K\Delta t$ can be obtained as a possible state trajectory of the system. If an action is included that causes motion, shorter polygonal curves can also be obtained.

A discrete-time representation of the loss functional must also be defined:

$$L(x_1,...,x_{K+1},u_1,...,u_K) = \sum_{k=1}^{K} l_k(x_k,u_k) + l_{K+1}(x_{K+1}) \tag{4.6}$$

in which l_k and l_{k+1} serve the same purpose as l and Q in the continuous-time loss functional.

The basic motion planning problem can be represented in discrete time by letting $l_k = 0$ for all $k \in \{1,...,K\}$, and defining the final term as $l_{K+1}(x_{k+1}) = 0$ if $x_k = q_{goal}$ and $l_{K+1}(x_{k+1}) = 1$ otherwise. This gives equal preference to all trajectories that reach the goal, and approximate the problem of planning as an optimal-length path, with $l_k = 0$ for all $k \in \{1,...,K\}$. The final term is then defined as $l_{K+1}(x_{k+1}) = 0$ if $x_k \in q_{goal}$, and $l_{K+1}(x_{k+1}) = \infty$ otherwise.

The previous formulations have shown equivalence between the basic motion planning problem and a specific version of the optimal control problem. Therefore, an algorithm that solves a basic motion planning problem equivalently solves a specific optimal control problem. For example, the visibility graph approach can be considered as a method for determining an optimal controller for a particular optimal control problem in $\Re^3$ with polygonal constraints on the state space and the loss functional.

One key difference between this optimal control problem and those typically considered in control theory literature is the set of geometric constraints on the state space that appear because of obstacles in the workspace. These constraints represent a twist to the control problem and must be confronted in a robotics context. Another difference is that a goal (or reference) trajectory that serves as a reference for comparing solutions is often specified for standard control problems; however, in basic motion planning the primary task is to select the trajectory (the collision-free path).

The optimal control formulations that are considered in this section are limited, however, in a number of ways: (1) only open-loop control functions are considered; (2) no form of uncertainty is assumed; and (3) only a single robot is being controlled; (4) the space of motion is in $\Re^3$.

4.3 Uncertainty Environments

The success of a motion planning approach in an implemented robotics system depends to a large extent on the manner in which various forms of uncertainty are modelled and treated. Motion planning under uncertainty therefore represents a very important extension of the basic problem and has received much attention from the motion planning community [233].

Two common representations of uncertainty have been applied to motion planning problems. One representation restricts parameter uncertainties to lie within a specified set. A motion plan is then generated that is based on *worst-case analysis* [63][122][231][239]. We refer to this representation as *nondeterministic uncertainty*. The other popular representation expresses uncertainty in the form of a probability density function. This often leads to the construction of motion plans through *average-case* or *expected-case analysis* [53][150][327]. We refer to this case as *probabilistic uncertainty*.

We will describe each of the sources of uncertainty in isolation, although in general any combination of these uncertainty types can be considered simultaneously in a motion planning formulation. Uncertainty can be introduced into a motion planning problem in a number of ways, thus we organize this uncertainty into four basic sources [233] for the discussion which follows.

4.3.1 Configuration-Sensing Uncertainty

Suppose that C_{free} is given. Under uncertainty in configuration sensing, incomplete or imperfect information is utilized by the robot to make an inference about its configuration. This information could come from sensor measurements or motion history. With a nondeterministic uncertainty model, the robot might have sufficient information to infer that $\boldsymbol{q}$ lies in some subset $Q \subset C_{free}$. For example, this representation of uncertainty is used to guarantee that the robot recognizably terminates in a goal region. With a probabilistic model, the robot might infer a posterior probability density over configurations [239], $p(\mathbf{q})$, that is conditioned on sensor observations [231], initial conditions [63], or additional knowledge [122]. Examples that handle configuration-sensing uncertainty with probabilistic representations could be found at [53][351][60].

4.3.2 Configuration-Predictability Uncertainty

Suppose that both C_{free} and the current configuration, $\boldsymbol{q} \in C_{free}$ are given. Motion commands can be given to the robot, but with control uncertainty the future configurations cannot, in general, be completely predicated [104]. With nondeterministic uncertainty, the robot may infer that some future

configuration will belong to a subset $Q \subset C_{free}$. The method of preimage backchaining constitutes a large body of work in which bounded uncertainties are propagated and combined with configuration-sensing uncertainty, to guarantee that the robot will achieve a goal [135][239]. With a probabilistic model, future configurations can be described by a posterior density over configurations, $p(\mathbf{q})$, that is conditioned on the initial configuration and the executed motion command [150].

4.3.3 Environment-Sensing Uncertainty

Analogous to configuration-sensing uncertainty, suppose that a space of possible environments, E, is known to the robot. Although a space of configurations is a well-defined concept, in robotics literature, we must define what is meant by a "space of environments". For the purpose of discussion, let E contain different possibilities for C_{free}. Under environment-sensing uncertainty, incomplete or imperfect information is utilized by the robot to make an inference about its environment. With a nondeterministic uncertainty model, the robot might have sufficient information to infer that the environment e belongs to some subset $F \subset E$. For example, a determined environment could be restricted to a plane populated with unknown polygonal obstacles, which are then discovered using visual "scans" to build a visibility graph for motion planning [300]. Sometimes unknown obstacles are allowed to be of arbitrary shape, and the sensor data consists of "tactile" information for a point robot [240]. With a probabilistic model [115], the robot might infer a posterior probability density [190], $p(e)$, over environments, which is conditioned on sensor observations [119], initial conditions, or additional knowledge [341].

4.3.4 Environment-Predictability Uncertainty

Suppose again that the space of environments E is known by the robot; however, in addition, the robot knows its current environment $e \in E$. Predictable motion commands might be given to the robot, but with environment-predictability uncertainty, future environments cannot be completely predicted. With nondeterministic uncertainty, the robot may infer that some future environment will belong to a subset $F \subset E$ [289]. With a probabilistic model, future environments can be described by a posterior density over environments $p(e)$ that can be conditioned on the initial environment, the robot configuration, or an executed motion command [33][53].

4.4 Sensor-Based Motion Control

The role of perception for a given robot with predefined motor actions is to determine what actions take place and when. In this style of action-oriented perception, the action defines the form of the perception in terms of what information is needed for the action to make its control decision. Thus sensing for intelligent robots may be defined as "the process of gathering or receiving data about the environment and the agent itself" [85]. Architecture for intelligent agents usually provides some method of interfacing the agent to the environment through sensors. Sensory information can be encoded at both a low level and a high level and utilised by high-level decision-making processes of the agent. We could summarise that the work of sensing the environment around a robot as two fundamental operations: gathering data about the environment, and interpreting the data. For a physical robot, gathering data involves devices such as cameras, laser rangefinders, and sonars, while

interpreting data involves software algorithms, e.g. image segmentation, 3D model reconstruction, object recognition, motion estimation, etc.

Previous researchers have argued the case for employing some kind of virtual perception for animated characters [305]. This may include one or more of simulated visual, acoustic, aural, olfactory, or tactile perception. Some research has been done on sensor-based motion [214] and manipulation [191], therefore perception is used to help decide what action the robot should take and when it should be performed, and such approach is also known in the literature as perceptual sensing [225].

The connection between local perception and global action is a redundant system of mobile robots through the mass effect of competitive or collective robotics. For any one robot, its locally derived perception may not decode the environment completely, due to limitations imposed by the robot's position within the environment (a spatial constraint). Nevertheless, since sensing in such a spatially distributed system increases the probability that some of the robots correctly respond to the environment, then the actions performed on the environment by those robots may allow others to sense stimulus changes upon the manipulated objects in the workspace. How local perception decodes stimuli depends on the approaches taken to integrating sensor data, and one of them could be by defining and specifying perceptual cues.

Defining and specifying perceptual cues involves three techniques for cue creation: feature extraction using threshold logic; orthogonal sensing as a means for integrating physical sensors; and additive cue construction specified as clauses in predicate calculus. The result is cues that answer yes/no type questions about what can be sensed in the robot's immediate vicinity. Functionally, perceptual cues are used for either activating motor behaviours or for causing state transitions among the robot's subtask controllers. Consequently, the local perception can be summarized as a way of determining the "what and when" for robot action sequences.

4.4.1 Perceptual Cue

A perceptual cue is a boolean value that indicates either the presence or absence of a pattern of stimuli. Perceptual cues (PCs) are context dependent features in sensor data that indicate a perceived event. Context is determined by the current state in task execution space. States in task execution are specified as steps in the task and implemented as subtask controllers. At the level of task description, PCs are used to determine which step of the task is being executed. Each subtask controller consists of a finite number of states, where each state is associated with a certain motor action and implemented as a primitive actuation behaviour. In a perceptual behaviour approach, sensor features detected by a perceptual cue map directly to motor actions.

Features are obtained by processing sensor data to produce a binary output. Sensor data is acquired from single or multiple sensors and is processed using simple threshold logic. Cues can be created by using data from different sensor types using boolean operators. Cues are context dependent in that they are specified or a specific task and a given environment. Sensor features which are not unique can be combined orthogonally or additively.

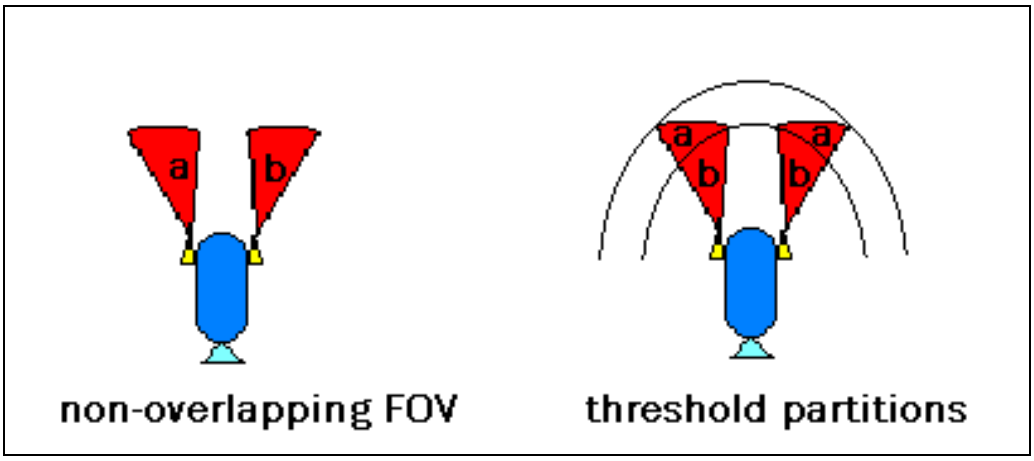

Figure 4.1: Sensing by orthogonal spatially sensors.

4.4.2 Orthogonal Sensing

In order to simplify sensor processing, binary cues created using threshold logic can be integrated by employing either spatially or modally orthogonal sensing strategies. The result integrates multiple sensors of the same type geometrically, by spatially partitioning the robot's perceptual field-of-view. Sensors of different type are combined to create cues in which all bit positions in the output vector are from dissimilar stimulus modalities. Their combination makes the extracted sensor feature temporally unique.

Sensing can be made spatially orthogonal by either arranging the same type of sensors geometrically with nonoverlapping fields-of-view or by partitioning the field-of-view with thresholds as shown in Figure 4.1. As an example of a spatially orthogonal sensor, consider a ring of eight sensors, each with a 45 degrees field-of-view and equally spaced on a circle – it is a common configuration found in commercial mobile platforms. The perceptual space is divided into eight discreet zones in which stimuli may be detected. If obstacle sensors were used, then each bit of an 8-bit vector could represent the presence of an obstacle within the assigned zone. Thus, 256 possible combinations are available for mapping to motor actions used in obstacle avoidance. The outputs are combined using boolean operators resulting in an unique feature in the sensor's output space from sensors of different modalities.

4.4.3 Additive Cue

Perceptual cues can also be defined by combining cues additively as a Horn clause [225]. In predicate calculus a Horn clause is any disjunction of the form: $\neg A \vee \neg B \vee \ldots \vee \neg C \vee D$. Each Horn clause has at most one positive literal, and can be rewritten as an equivalent formula: $A \wedge B \wedge \ldots \wedge C \rightarrow D$. Such a formula is a notational variant of Horn clauses used in Logic Programming and Fuzzy Logic [249]. The newly defined cue is the consequent variable D of previously defined cues represented by the variables A, B,…, C. Cues defined in this way represent the state of the task model and are generally used for behaviour activation in a world completed by

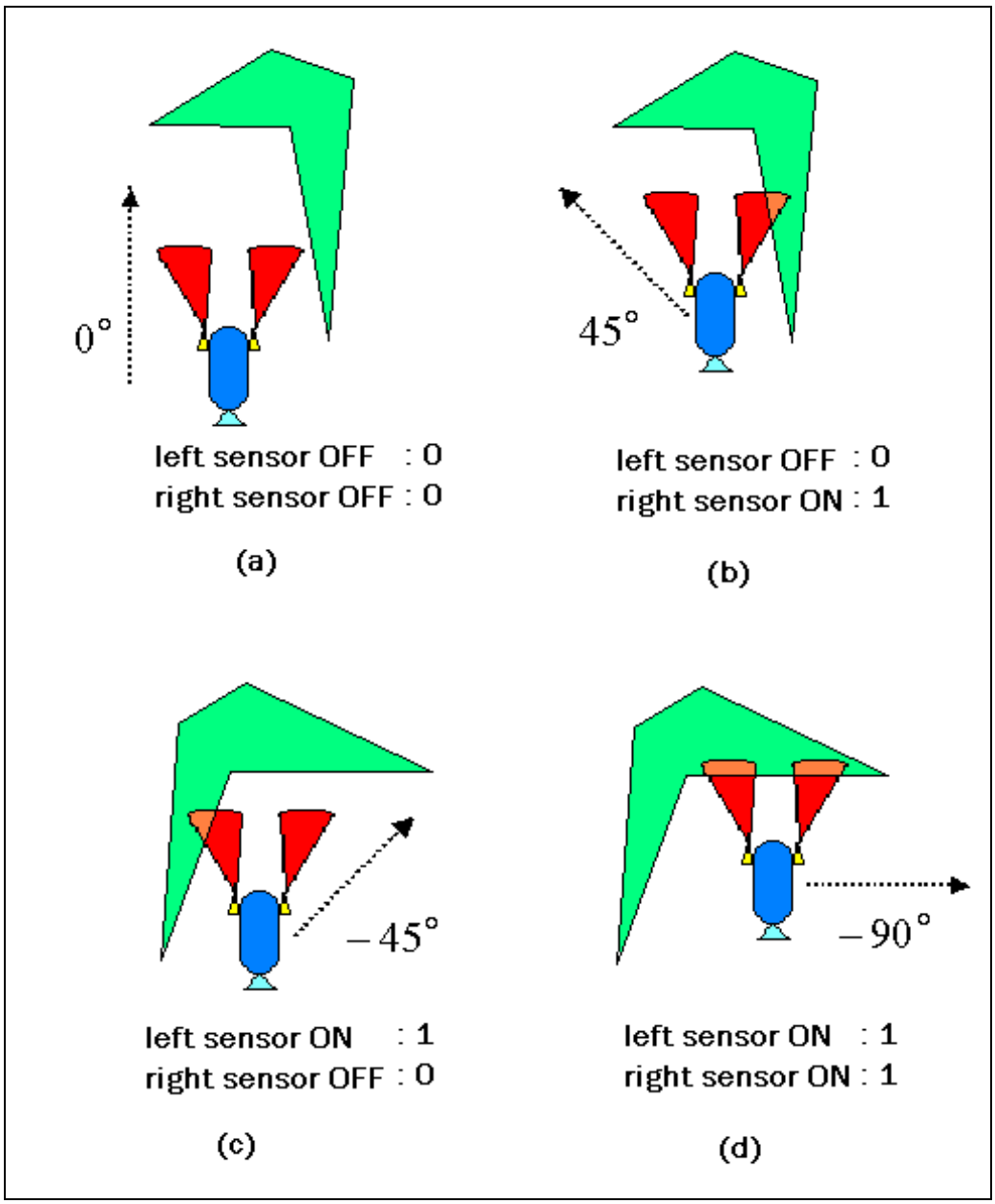

Figure 4.2: Robot orientation by sensor-based reaction.

objects, agents, and events. Thus a perceptual cue is a control decision used to trigger a motor behaviour and to control the transition among states in a task model. Motor behaviours remain active for a fixed period of time, at the end of which the cue's truth-value is reevaluated. Either the same cue is applicable or the stimulus conditions have changed, thereby activating another cue.

In executing a robot task, defined as a multistep procedure, stimulus conditions may also change sufficiently to indicate a state transition, where "state" represents a separate motion controller designed to accomplish one step in a task description. Using cues to trigger a behavioural response is a common mechanism for action in social insects [275] and for governing different phases of activity in tasks such as nest building [109].

The advantage of reducing motor behaviour control decisions to binary values is the cue's functional abstraction. In this manner, activation of a motor behaviour is not dependent on a specific perceptual cue, but rather on the decision that results from sensor processing. For example, a motor

behaviour created to make a robot rotate $\sin(\Phi)$, where Φ assumes a set of possible predefined values, changes the robot route avoiding a collision between the robot and some undesirable obstacle. If touch sensors are used then about the point of contact, it could specify when both touch sensors are in contact with the surface as illustrated in Figure 4.2, and return a binary "11" value. Thus to maintain a normal orientation with respect to obstacles, left and right contact information is used. The information can be provided by either touch sensors or acoustic sensors.

Contact with the left side only would be represented as binary "10" and contact with the right as a "01" value, with the no contact condition specified as a "00" value. The same contact information using acoustic sensors and phototaxis could be specified by determining the sensor's threshold value when in contact with a surface and creating cues that return "11" when the robot is in full contact with the surface in a similar manner.

The advantage is that the design of the motor behaviour does not change when different sensor types or alternate feature extraction techniques are used since the information needed by the motor behaviour is the same binary vector in both cases. Therefore the function of perceptual cues is to control behaviour activation and state transitions in a manner that allows for changes in perception design and implementation without affecting the control architecture's connection to motor action.

4.5 Multiple Robot Motion Planning

For multiple robot motion planning problems, we are concerned not only about collision with obstacles, but also about collisions that occur between robots. Thus let each robot, r_{Ω} , be a rigid object, capable of moving in a workspace that is a bounded subset of $\Re^3$. The position and orientation of the robot in the workspace are specified parametrically, by a point in an *n*-dimensional configuration space, C^i. Static obstacles in the workspace, compact subsets of $\Re^3$, prohibit certain configurations of the robot. The open subset of C^i that corresponds to configurations in which r_{Ω} does not intersect any obstacles, is referred to as the free configuration space, and is denoted by C^i_{valid}, which is the closure of C^i_{free}. We use C^i_{valid} in this work because optimality is more straightforward to consider. This distinction is primarily technical, because solutions that exist in C^i_{valid} can be considered as limit points for solutions in C^i_{free}. We assume that each robot has complete knowledge of C^i_{valid}, along with perfect configuration sensing and control.

A state space, *X*, is defined that simultaneously represents the configurations of all of the robots. Because collisions with obstacles are prohibited, a natural choice for the state space is

$$X = C^1_{valid} \times C^2_{valid} \times \cdots \times C^N_{valid} \tag{4.7}$$

in which $\times$ denotes the Cartesian product. Let the state space be represented as

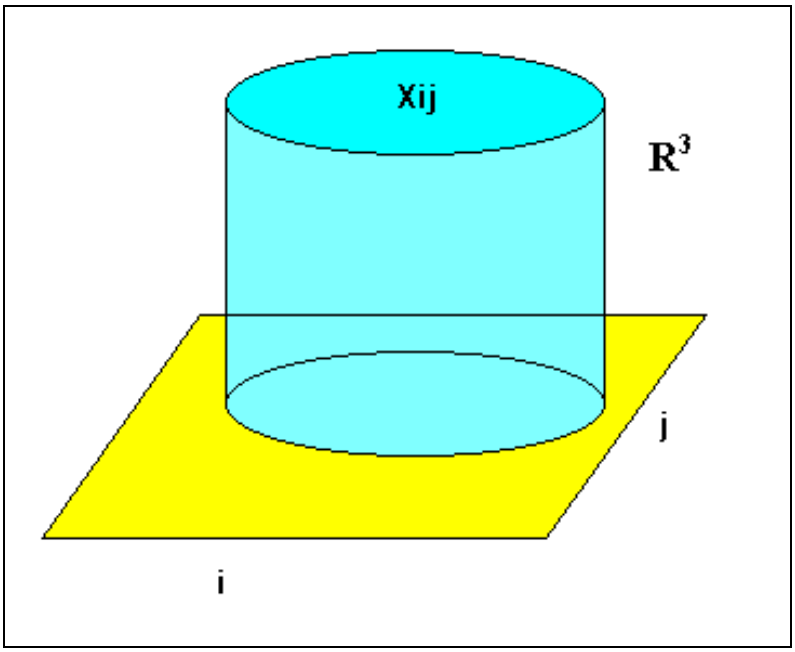

Figure 4.3: The set X_{ij} and its cylindrical structure on R^3.

$$X = X^1 \times X^2 \times \cdots \times X^N \tag{4.8}$$

Each subspace, X^i, of the state space yields the configuration of r_Ω . Let $r_{\Omega\circ}$ denote the interior of r_Ω . We define the open set corresponding to the exclusion of the boundary of r_Ω as

$$X_{coll}^{ij} = \{x \in X \mid r_{\Omega\circ=i}(x^i) \cap r_{\Omega\circ=j}(x^j) \neq 0\} \tag{4.9}$$

We use the notation $r_\Omega(x^i)$ to refer to the transformed robot, $r_{\Omega=i}$, at x^i. The equation 4.9 represents the set of states in which two robots collide. The collision subset, $X_{coll} \subset X$ is represented as the open set,

$$X_{coll} = \bigcup_{i \neq j} X_{coll}^{ij} \tag{4.10}$$

Hence, a state is in the collision subset if the interiors of two or more robots intersect. We define X_{valid} as the closed set, $X - X_{coll}$ - see Figure 4.3. The basic task is to bring each robot from some initial state $x_{init}^i \in X^i$ to some goal state $x_{goal}^i \in X^i$. While achieving this task, each robot is not permitted to collide with obstacles or other robots, which means that the state must remain within X_{valid} . In addition, explicit objectives must be taken into consideration when achieving this task.

We consider a state trajectory as a continuous mapping $x:[0,T] \to X$. A trajectory for an individual robot is represented as $x^i:[0,T] \to X^i$. An explicit choice for the final time, T, is usually

not needed in practice. For some problems, a final time may naturally exist, by which the robots must accomplish the basic task. Usually, however, we do not require a specific termination time, and can consider $T = \infty$. The motion of an individual robot, r_Ω , is specified through the state transition equation,

$$\dot{x}^i(t) = f^i(x^i(t), u^i(t)) \text{ for each } i \in \{1,..., N\}, \tag{4.11}$$

in which $u^i(t)$ represents a control function for r_Ω , which is chosen from a set of allowable controls.

Considering the geometric aspect of a motion planning problem [27], we assume that a robot is capable of switching between a fixed, maximum speed, $\left\|v^i\right\|$, and remaining motionless, what represents a typical characteristic in geometric motion planning [121][287]. If for example, a robot is allowed to translate and rotate, then finite bounds might be given that limit the translational and angular speeds [207].

4.5.1 Multiple Robot Coordination

A collective system that acts as a unit in a well-coordinated manner is displaying a coherent system behaviour. Such a system, composed of people, insects or robots, is thought to be more effective at achieving some goals than individuals acting alone. In robot tasks, like collective manipulation, is such a cooperative system possible without inter-robot communication or robot identification? Coherent behaviour is accomplished by viewing the system that solves the problem as two equally important parts consisting of the environment and the robot system as shown in Figure 4.4. A solution to a given task is considered to consist of two parts: the environment with actions on its input and changes in stimulus as its output, and the robot system with stimulus as input and actions on the environment as output.

The environment has actions performed in it on its input side, which result in changes that may be perceived on its output side. The robot system has perception on its input and produces actions in the environment as its output. In such a system the task to be accomplished is the desired change in the environment in response to input actions performed by the robots. The robot system is the procedural mechanism used to achieve those changes. In this synergistic system coherent behaviour becomes possible as the common task and its environment become the central coordinating mechanism.

Nature has generally provided us examples of a multi-agent system such as social insects [137], whose decentralized control is based solely on locally sensed information. Moreover, ants exhibit a group transport behaviour, used in both food and prey retrieval tasks, in which stagnation problems arise and are solved using simple recovery strategies [345]. Group transport is the most common cooperative movement of a global team goal observed by social insects. Very few studies have examined this behaviour which is found almost exclusively in ants, but those that have, have shown group transport to be an efficient way of moving a load with a small workforce [268][135]. Food is generally consumed within the nest and must be first torn apart before consumption. Ants must either transport the food item as a whole from its location or dismantle it into small enough pieces to be carried back to the nest by an individual. A detailed study of the movement patterns involved in group transport was carried out by Sudd in which it was concluded that although the behaviour of ants in group transport was similar to that of single ants, a cooperative team interaction could be attained for distinct kinds of pre-established tasks [347][345].

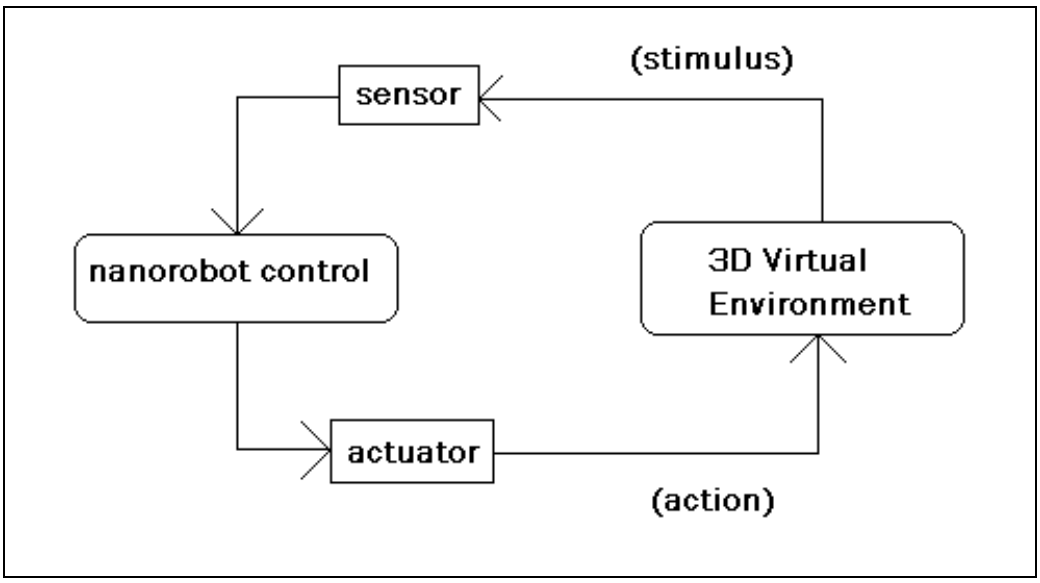

Figure 4.4: Input's stimulus and the robot's output action.

Sensing plays a key role in triggering the transition between different task construction or transport behaviour steps. It is reasonable, therefore, to speculate that such a mechanism may also be used as a means of synchronizing several asynchronous robots in the execution of a common task. A frequent question about social insects is how they collectively build sophisticated nests without centralized planning. Coordinating their building activities often involves simple rules applied without communicating directly with other workers as Brooks concluded after modelling the two dimensional structures built by ants using a bulldozing-building behaviour [52].

Nest building by ants that live in the flat crevices of rocks involves making perimeter walls around their colonies without the need to construct either a roof or floor. This type of two dimensional structure is highly conducive to laboratory observation and data collection, as nests could be built between two microscope glass slides separated with cardboard columns. The first stage of wall construction described involves an individual ant carrying a granule into the nest towards the cluster of nest mates. Once the ant is close it reverses its direction 180 degrees and begins to push the granule into other existing granules. This bulldozing behaviour was tested as a computer-simulation model producing a similar pattern of granules that formed perimeter walls. Thus, bulldozing behaviour is an example of how a simple rule for building can be used to produce a predictable result without direct communication between builders. Rather, indirect communications through the environment by way of the building structure serves to coordinate collective activity [52]. In this way both the environment and behavioural act used for task completion is part of the solution.

Attempts to model the states of both the environment and its cognizant occupants is not novel. Animal behaviour studies were done to define a motivational state as a combination of a physiological and perceptual state, with behaviour used to change states in motivational space [98]. This approach was extended to modelling the system behaviour by assigning state variables to environmental space, behaviour space and task space. Environmental space defines the constraints imposed on the system with regards to movement and topology. Behavioural space refers to the partition of the environment made by the animal's or robot's sensory system. Tasks are defined by their initial and final states using state variables that are relevant to the task.

Finite state automata (FSA) have been used to model perceptual tasks [31] and motivational behaviour in animals [123][99]. FSA was used to model the space-time relationship in a perceptual processing task on a mobile robot. This approach allows for perceptual tasks to be sequenced in a reactive control system. Thus finite state automata used to model the steps in a task as rules of interaction along with local perception to control the application of that action is a plausible model for a collective coherent behaviour.

4.6 Conclusion

The study of motion planning is a part of the control problem that has a great application for robotics control and for animation in computer graphics [66]. Many algorithms have been proposed dealing with motion planning optimization, and different approaches have demonstrated good results [4][43][71][186]. A mobile robot requires a more elaborated motion algorithm, once it is believed that it will work in a more complex environment. Thus for such cases, the best way to deal with uncertain environments is to use non-deterministic approaches. Generally non-deterministic methodologies will be supported by a set of sensors as an ancillary way to support the robot when dealing with unpredictable situations. This is mentioned in the research communities as action-oriented perception, which affects a robot's behaviour based on events in a multiple reaction to the surrounding environment.

While for a dynamic environment a local perception will help the robot mainly to avoid collisions, for workspaces composed of several mobile robots the sensor-based perception will be required to recognize any eventual change performed by the other robots, which could interfere directly or indirectly with the task attributed to any robot in the related environment. Although an easier approach would be a telemetric system with centralized information, for the problem related to nanorobotic automation applied to nanomedicine, the most feasible approach is a decentralised and local perception approach. For fast massive automation in nanotechnology, the study of competitive and collective robotics systems capable of supporting the complexity inherent in nano-worlds and to incorporate coherent behaviour interacting with the environment, is a field of growing interest that has to be further investigated in the coming years [69][70][72][66].

Chapter 5

Artificial Neural Networks

5.1 Introduction

The field of artificial neural networks (ANNs) is an interdisciplinary area of research. A thorough study of artificial neural networks involves knowledge about neurophysiology, cognitive science or psychology, control theory, computer science, artificial intelligence, statistics, mathematics, pattern recognition, computer vision, parallel processing, and hardware.

The successful use of neural networks for robotics is a feasible method for adaptive motion control for complex environments where some determined robot is required to interact with a certain set of stochastic events which have been done with satisfactory results [71][70]. Virtually every intelligent robot designer uses a different combination of controller paradigm and learning method. As a result the automation field for robotics is very broad and many working systems have been demonstrated for different kinds of applications [65][66].

The classical artificial intelligence (AI) approach to control autonomous systems is to break up the problem into functional modules such as sensory perception, environmental modelling, planning of actions and execution of those plans [51]. Various techniques from the AI toolbox are used within each module, such as knowledge representation schemes, and goal directed searching and reasoning [353]. However it has been argued that classical AI cannot produce systems that are robust in real-world environments [38].

It has been established that a large number of applications can benefit from the use of ANNs [95][183][175][246][162]. Artificial neural networks are massive parallel computing systems consisting of an extremely large number of simple processors with many interconnections between them. ANNs were designed with the goal of building “intelligent machines” to solve complex problems, such as control systems, pattern recognition and dynamic optimization.

5.2 Brief Historical Review

Humans, being inquisitive creatures, have long been interested in exploring where the mind originates and how the brain computes. These efforts may be traced back to Aristotle. Yet, the modern

era of computational neural modelling began in 1943 with the pioneering work of McCulloch and Pitts [252], who introduced a computational model of neuron and a logical calculus of neural networks. McCulloch-Pitts' classic paper was widely read at the time (and is still read), and for 15 years generated considerable interest in the detailed logic of networks consisting of simple neurons. Such networks were proved to be capable of any boolean function.

The next major milestone in ANNs was Rosenblatt's work on the Perceptron in 1958. The crowning achievement of Rosenblatt's work was the first proof of the perceptron convergence theorem. In 1960, Widrow introduced the least mean square (LMS) algorithm for the Adaptive Linear Element. Nilsson's book on machine learning [281] was the best-written exposition of linearly separable patterns in hypersurfaces. ANNs generated a great deal of enthusiasm in the 1960's. It appeared that such a machine could do any type of computation. However, this enthusiasm was dampened by Minsky's book [265], which demonstrated the fundamental limitations of the computing power of one-layer perceptrons. They showed that certain rather simple computations, such as the Exclusive-OR (XOR) problem, could not be solved by the one-layer perceptron. It was believed that such limitations could be overcome by multilayer perceptrons that employ intermediate layers of units (hidden units) between the input layer and output layer. But, a difficult problem encountered in designing a multilayer perceptron is the credit assignment problem. There was no learning algorithm known at that time to solve this problem, thus Minsky doubted that one could find a solution for that and thought it more profitable to explore other approaches to artificial intelligence. Because of this and other reasons, research into neural networks went into hibernation. However, the neural network field was not completely abandoned in the 1970's. A number of dedicated researchers continued to develop neural network models. Two important themes that emerged were associative content-addressable memory and self-organizing networks using competitive learning.

In the 1980's a number of important publications appeared, which changed the course of ANNs research. Perhaps more than any other publication, the 1982 paper by Hopfield [183] and the two-volume book by Rumelhart in 1986 [314] were the most influential publications. In 1982 Hopfield introduced the idea of an energy function from statistical physics to formulate a new way of understanding the computation of recurrent networks with symmetric synaptic connections. This formulation makes explicit the principle of storing information as dynamically stable attractors.

In 1986 Rumelhart reported the development of the backpropagation algorithm that popularized the use of the multilayer perceptron to solve a wide variety of pattern recognition problems. In fact, the development of the back-propagation algorithm has a colorful history. It was first developed by Werbos in 1974 in his Ph.D. thesis, and later rediscovered independently in two other places by Parker and by Lecun in 1985.

Over the last ten years, thousands of researchers from many diverse fields, such as neuroscience, psychology, medicine, mathematics, physics, computer science, and engineering, have been involved in developing neural network models, implementing the models in hardware, VLSI (Very Large Scale Integration Systems) and optics, and software, and solving a number of important applications of the ANN models.

5.3 Biological Models

Many existing robots and automatons are biologically inspired in some way. This is sometimes because researchers want an injection of new ideas into their designs, and sometimes because they want to model biological systems to help understand them better. For example, in [118] the body and spinal cord of the Lamprey (a kind of fish) were simulated. It was shown that the coupled oscillators in the signal cord could, in conjunction with sensory feedback, produce the correctly timed muscle

contractions necessary for swimming. Other Lamprey studies [283][282] explored brain stem control models and learning schemes to acquire the appropriate central pattern generator (CPG) parameters for correct swimming.

Another example is the study of a model insect walking in a neural network simulation, which controls a six-legged robot [91][32]. Realistic inter-leg coordination mechanisms are used and it is shown that interactions between the controlling network, the robot and the environment are important. Lewis has used genetic algorithms to synthesize gait-producing pattern generators in a hexapod robot [237].

Several groups have attempted to model the brain at a much higher level. For example, Hosogi describes a quasi-realistic cerebellum model used to control a robot manipulator [187]. It contains a self-organizing granule cell layer and a Purkinje layer that uses Hebb learning rules. The "Darwin" system is an ambitious attempt to create a complete artificial brain for various automatons [303]. Based on Edelman's theory of neuronal group selection, it has realistic cell and synaptic modification dynamics, and various realistic sensory and motor systems. The Darwin-III system contains 50 interconnected networks with some 50,000 cells and 620,000 synaptic junctions.

Many authors have created design paradigms based on biological principles. Crawford suggests a hierarchical controller using radial basis function networks for systems with many degrees of freedom, made up of a network of the simple single-joint controllers [89][90]. This approach was used to control a simulated human platform diver. Altman presents a distributed decision-making model for insects [3], based on a neural equivalent of Brooks' typical architecture model. Kalveram suggests that robot arm movements can be controlled by CPGs and reflex-like processes which allow high level centres to specify only the kinematics (not the dynamics) of movement [205]. Hallam gives a neuroethological approach for controlling a mobile robot using a neural network with quasi-realistic synapse modification [166].

A neuron is a special biological cell, the essence of life, with information processing ability. The introduction of neurons as basic structural constituents of the brain was credited to Ramon y Cajal who won the 1906 Nobel Prize for physiology and medicine, shared with Camillo Golgi, for the crucial discovery of the extensive interconnections within the cerebral cortex, the portion of the brain where approximately 90% of the neurons in the human are located.

A schematic drawing of a neuron is shown in the Figure 5.1. A neuron is composed of a cell body, or soma, and two types of out-reaching tree-like branches: axon and dendrites. The cell body has a nucleus that contains information on hereditary traits and plasma containing molecular equipment for the production of material needed by the neuron. The cell membrane contains various types of electrochemical pumps that can maintain equilibrium in charge concentrations inside and outside the cell.

A neuron receives signals (impulses) from other neurons through its dendrites (receivers), and transmits signals generated by its cell body along the axon (transmitter), which eventually branches into strands and substrands. At the terminals of these strands are the synapses. A synapse is a place of contact between two neurons (an axon strand of one neuron and a dendrite of another neuron). When the impulse reaches the synapse's terminal, certain chemicals, called neurotransmitters are released. The neurotransmitters diffuse across the synaptic gap, and their effect is to either enhance or inhibit, depending on the type of synapse, the receptor neuron's own tendency to emit electrical impulses. The effectiveness of a synapse can be adjusted by the signals passing through it so that synapses can learn from the activities in which they participate. This dependence on past history acts as a memory that is possibly responsible for the human ability to remember.

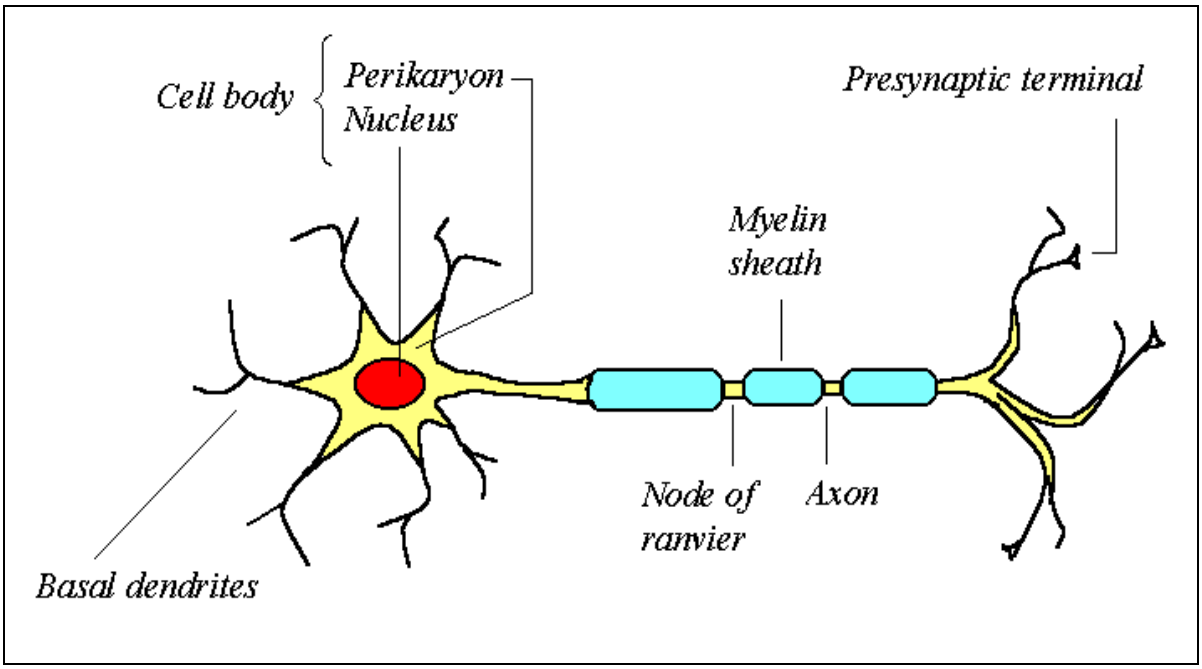

Figure 5.1: A sketch of a biological neuron.

The cerebral cortex in humans is a large flat sheet of neurons about 2 to 3 mm thick with a surface area of about 2,200 cm^2, about twice the area of a standard computer keyboard. This is an amazing creation of nature because a sphere with a volume of about 1.5 liters, the typical size of a human brain, has a surface area of only 634 cm^2. It is the walnut appearance of the human brain that provides the cerebral cortex with a surface area three times larger than a simple smooth spherical surface. The cerebral cortex contains about 10^{11} neurons, which is approximately the number of stars in the Milky Way! There are about 34 different types of neurons based solely on their shape, and as many as 100 types of functionally different neurons. Neurons are massively connected, much more complex and denser than today's telephone networks. Each neuron is connected to 10^3–10^4 other neurons. The number of interconnections depends on the location of the neuron in the brain and the type of neuron. In total, the human brain contains approximately 10^{14}–10^{15} interconnections.

Neurons communicate by a very short train of pulses, typically milliseconds in duration. The message is modulated on the frequency with which the pulses are transmitted. The frequency can vary from a few up to several hundred Hertz, which is a million times slower than the fastest switching speed in electronic circuits. However, complex perceptual decisions, such as face recognition, are made by a human brain very quickly, typically within a few hundred milliseconds. These decisions are made by a network of neurons whose operational speed is a few milliseconds. This implies that the computation involved cannot take more than about one hundred serial stages. In other words, the brain runs parallel programs that are about 100 steps long for such perceptual tasks. This is known as the hundred step rule [128]. The same timing considerations show that the amount of information sent from one neuron to another must be very small (a few bits). This implies that critical information is not transmitted directly, but captured and distributed in the interconnections, thus comes the name connectionist model. What is the magic that permits slow computing elements to perform extremely complex tasks rapidly? The key is the parallel and distributed representation and computation.

	Von Neumann computer	Biological computer
Processor	Complex	Simple
	High speed	Low speed
	One or a few	Large number
Memory	Separate from processor	Integrated into processor
	Localized	Distributed
	Non-content addressable	Content addressable
Computing	Centralized	Distributed
	Sequential	Parallel
	Stored programs	Self-learning
Reliability	Very vulnerable	Robust
Expertise	Numerical and symbolic manipulations	Perceptual problems
Operating environment	Well-defined, well-constrained	Poorly-defined, unconstrained

Table 5.1: Von Neumann computer versus biological computer.

5.4 Artificial Neural Networks

Modern digital computers have outperformed humans in the domains of numeric computational and related symbol manipulation. However, humans can effortlessly solve complex perceptual problems (e.g., recognizing a person in a crowd from a mere glimpse of his face) at such a fast speed and extent as to dwarf the world's fastest computer. Why does there exist such a remarkable difference in their performance? The biological computer employs a completely different architecture than the Von Neumann architecture (table 5.1). It is this difference that significantly affects the type of functions each computational model is best able to perform.

Numerous efforts have been made on developing "intelligent" programs based on the Von Neumann's centralized architecture. However, such efforts have not resulted in any general purpose intelligent programs. ANNs are inspired by biological evidence, and attempt to make use of some of the "organizational" principles that are believed to be used in the human brain. Our ability to model a biological nervous system using ANNs can increase our understanding of biological functions. For example, for many years experimental psychologists have used neural networks to model classical conditioning animal learning data [95]. The state-of-the-art in computer hardware technology (e.g. VLSI and optical) has made such modeling and simulation feasible. The long course of evolution has resulted in the human brain possessing many desirable characteristics, which are present neither in a Von Neumann computer nor in modern parallel computers. These characteristics include massive parallelism, distributed representation and computation, learning, ability, generalization ability, adaptability, inherent contextual information processing, fault tolerance, and low energy consumption. It is hoped that ANNs, motivated from biological neural networks, would possess some of these desirable characteristics from the human brain.

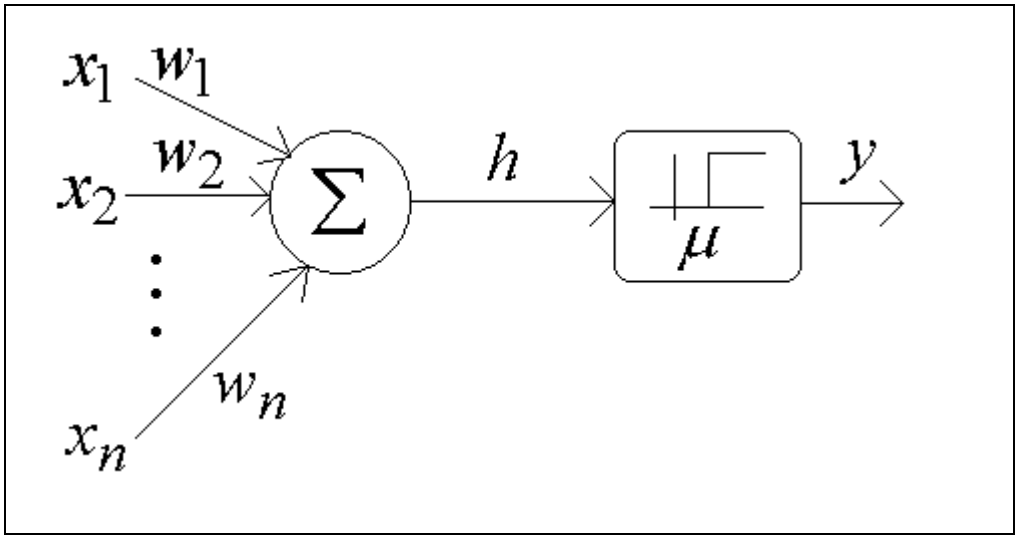

Figure 5.2: A neuron model.

5.4.1 Computational Models of Neurons

McCulloch and Pitts proposed a binary threshold unit as a computational model for a neuron [252]. A schematic diagram of a McCulloch-Pitts neuron is shown in Figure 5.2. This mathematical neuron computes a weighted sum of its n input signals, x_j, $j = 1, 2,\ldots, n$, and generated an output of "1" if this sum is above a certain threshold μ, and an output of "0" otherwise.

Mathematically,

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - \mu\right) \tag{5.1}$$

where $\theta(.)$ is a unit step function, and w_j is the synapse weight associated with the j^{th} input. For simplicity in notation, we often consider the threshold μ as another weight $w_0 = -\mu$ that is attached to the neuron with a constant input, $x_0 = 1$. Positive weights correspond to excitatory synapses, while negative weights model inhibitory synapses. McCulloch and Pitts proved that with suitably chosen weights, a synchronous arrangement of such neurons is, in principle, capable of universal computation. There is a crude analogy (table 5.2) to a biological neuron: wires and interconnections model axons and dendrites, connection weights represent synapses, and the threshold function approximates the activity in soma. The model of McCulloch and Pitts contains a number of approximated resolutions, which reflect biological neuron behaviour simplification. Some of these differences are:

- Biological neurons are not threshold devices, but have a graded response (essentially a nonlinear function of the inputs);
- Biological neurons perform a nonlinear summation of inputs and can even perform logical processing;

Biological neurons	Artificial neurons
Synapses	Connection weights
Axons	Output wires
Dendrites	Input wires
Soma	Activation function

Table 5.2: Analogy between biological neurons and artificial neurons.

- Biological neurons produce a sequence of pulses, not a simple output value;
- Biological neurons are updated asynchronously.

Nevertheless, the McCulloch-Pitts neuron model started a new era of computational neural modeling. The McCulloch-Pitts neuron has been generalized in many ways. An obvious generalization is to use activation functions other than the threshold function, e.g., a piecewise linear, sigmoid, or Gaussian, shown in Figure 5.3. The sigmoid function is by far the most frequently used function in ANNs. It is a strictly increasing function that exhibits smoothness and asymptotic properties. The standard sigmoid function is the logistic function, defined by

$$g(x) = \frac{1}{1 + \exp(-\beta x)} \tag{5.2}$$

where β is the slope parameter.

5.4.2 Network Architecture

An assembly of artificial neurons is called an artificial neural network. ANNs can be viewed as weight-directed graphs in which nodes are artificial neurons and directed edges (with weights) are connections from the outputs of neurons to the inputs of neurons. These are based on categories as shown in Figure 5.4, where there are feedforward networks in which no loop exists in the graph, and feedback (or recurrent) networks in which loops exist because of feedback connections.

The most common family of feedforward networks is a layered network in which neurons are organized into layers with connections strictly in one direction from one layer to another. In fact, all the networks with no loops can be rearranged in the form of layered feedforward networks with possible skip-layer connections. Figure 5.4 also shows typical networks of each category.

Different connectivities exhibit different network behaviours. Generally speaking, feedforward networks are static networks, i.e., given an input, they produce only one set of output values, not a sequence of values. Feedforward networks are memoryless in the sense that the response of a feedforward network to an input is independent of the previous state of the network. An exception is the time delay feedforward network in which dynamics occurs because of different delay factors of the neurons in the network. A positive aspect of feedforward networks is the fact that they have a good performance for combinatorial problems requiring low computational effort and processing demand [71].

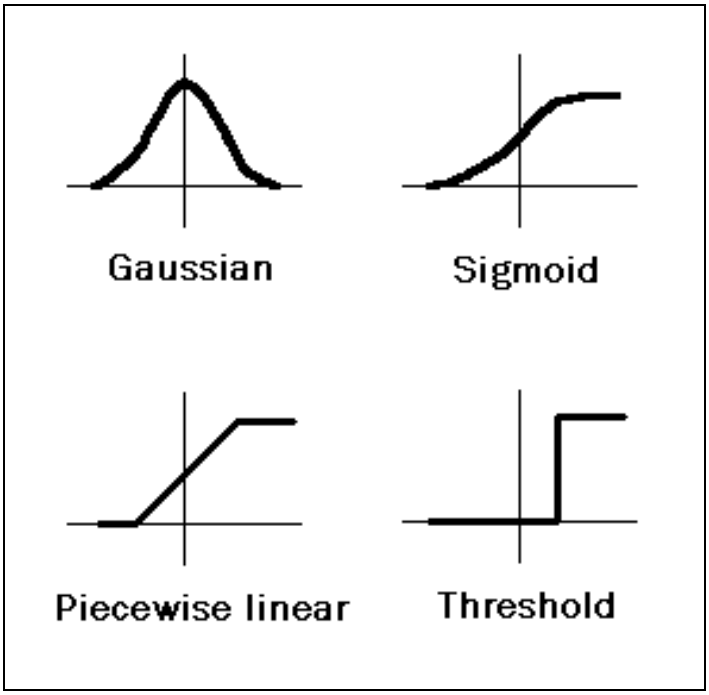

Figure 5.3: Different types of activation functions.

Recurrent networks are dynamic systems. Upon presenting a new input pattern, the outputs of the neurons are computed. Because of the feedback paths, the inputs to each neuron are then modified, which leads the network to enter a new state. This process is repeated until convergence. Obviously, different mathematical tools must be employed to treat these two different types of networks. Dynamic systems are often described by differential equations.

These network architectures can be either simulated in software or implemented in hardware - VLSI and optical. Software simulation of a network is always necessary before implementing it in hardware. A number of public and commercial software ANN simulators are available. More and more researchers have recognized the importance of hardware implementation, which is probably the only way to take full advantage of the capacities of ANNs. A difficulty in the VLSI implementation of ANNs is the massive connections. A fully connected network with N neurons requires N^2 connections!

This factor limits the number of neurons, typically a few hundred, that we can build on a single chip using the state-of the-art VLSI technology. An alternative is the optical implementation of ANNs. But it is still in the early stages.

Different network architectures require different learning algorithms. The next section will provide a general overview of the learning processes.

5.4.3 Learning

Ability to learn is a fundamental trait of intelligence, although what is meant by learning is often difficult to describe. A learning process, in the context of artificial neural networks, can be viewed as the problem of updating network architecture and connection weights so that a network can efficiently perform a specific task. Typically, learning in ANNs is performed in two ways. Sometimes, weights can be set primarily by the network designer through a proper formulation of the problem.

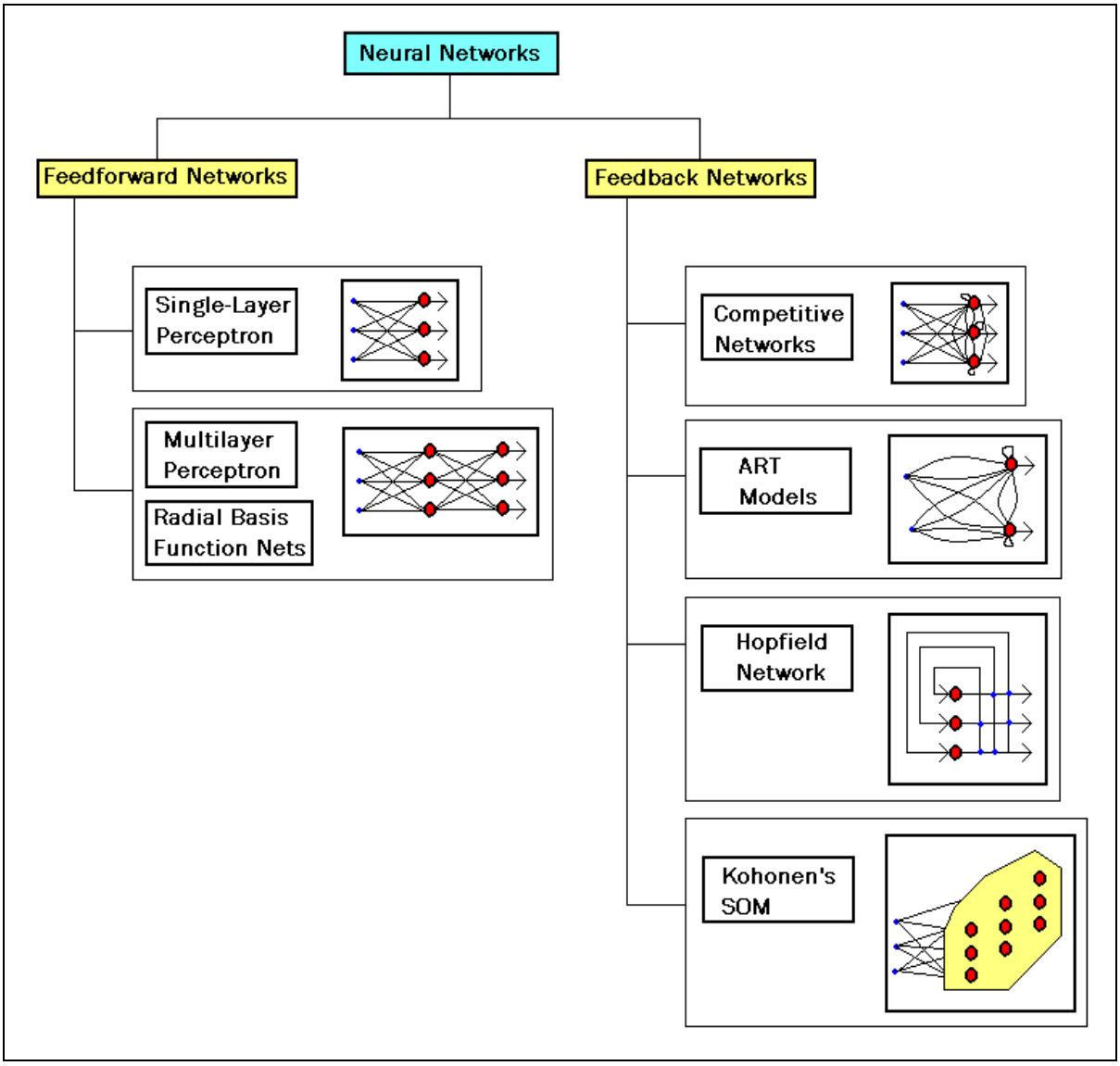

Figure 5.4: A taxonomy of network architectures.

However, most of the time, the network must learn the connection weights from the given training patterns. Improvement in performance is achieved over time through iteratively updating the weights in the network. The ability of neural networks to automatically learn from examples makes artificial neural networks very attractive and exciting. Instead of having to specify a set of rules, ANNs appear to learn from the given collection of representative examples. This is one of the major advantages of neural networks over traditional expert systems.

In order to understand or design a learning process, one must first have a model of the environment in which a neural network operates, i.e. what information is available to the neural network. We refer to this model as a learning paradigm [183]. Second, one must understand how weights in the network are updated, i.e. what is the learning rule that governs the updating process. A learning algorithm refers to a procedure in which learning rules are used for adjusting weights in the network. Finally, it is important to investigate how much the network can learn from examples (capacity), how many training samples are required (sample complexity), and how fast the system can learn (time complexity).

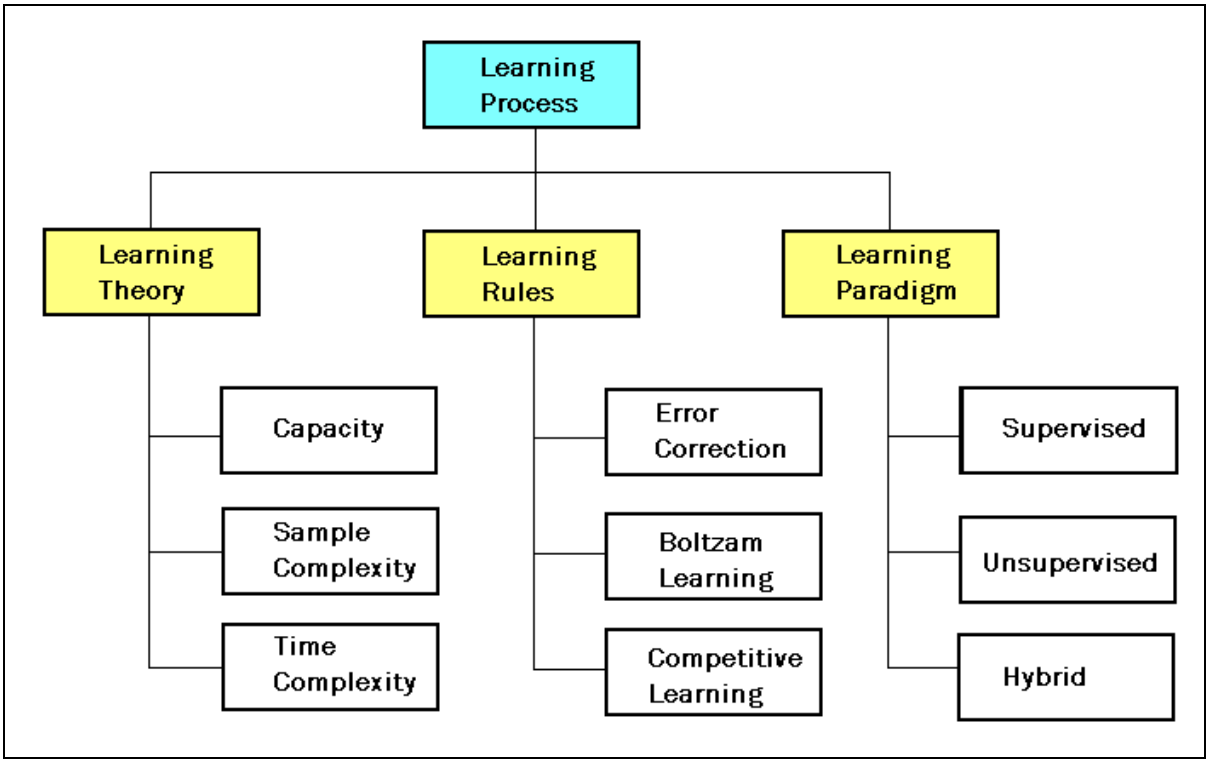

Figure 5.5: Learning issues.

The study of capacity, sample complexity, and time complexity is what a learning theory must deal with. Figure 5.5 illustrates these three aspects of a learning process. There are three main learning paradigms, namely: supervised, unsupervised, and hybrid learning.

- Supervised learning: In supervised learning, the network is provided with a correct answer to every input pattern. Weights are determined so that the network can produce answers as close as possible to the known correct answers. This is sometimes referred to as learning with a teacher. Reinforcement learning is a special case of supervised learning where the network is provided with only critiques on the correctness of network outputs.
- Unsupervised learning: In contrast, unsupervised learning does not require any correct answer associated with each input pattern in the training data set. It explores the underlying structure in the data, or correlations between patterns in the data, and organizes patterns into categories from these correlations.
- Hybrid learning: combines supervised learning and unsupervised learning. Typically, a portion of weights in the network is determined using supervised learning, while the others are obtained from unsupervised learning.

Learning theory must address three fundamental and practical issues associated with learning from samples: capacity, sample complexity, and time complexity. The first issue concerns whether the true solution is contained in the set of solutions that a network can deliver. If not, we can never hope to obtain the optimal solution. This remains a difficult and unsolved problem. Approximation capabilities of feedforward neural networks have been investigated by many researchers [175]. A fundamental result of these studies is that 3-layer, or even 2-layer, feedforward networks with an arbitrarily large

Learning paradigm	Learning Rule	Architecture	Learning Algorithm	Task
Supervised	Error-correction	Single- or multi-layer perceptron	- Perceptron learning algorithms - Backpropagation - Adaline and Madaline	- Pattern classification - Math approximation - Control
	Boltzmann	Recurrent	- Boltzmann learning algorithm	- Pattern classification
	Hebbian	Multi-layer	- Linear discriminant analysis	- Data analysis - Pattern classification
	Competitive	Competitive	- Learning vector quantization	- Within-class categorization - Data compression
		ART network	- Artmap	- Pattern classification - Within-class categorization
Unsupervised	Error-correction	Multi-layer feedforward	- Sammon's projection	- Data analysis
	Hebbian	Feedforward or Competitive	- Principal component analysis	- Data analysis - Data compression
		Hopfield net	- Associative memory learning	- Associative memory
	Competitive	Competitive	- Vector quantization	- Categorization - Data compression
		Kohonen SOM	- Kohonen's SOM	- Categorization - Data analysis
		ART networks	- Art1, Art2	- Categorization
Hybrid	Error-correction and Competitive	RBF (Radial Basis Function) network	- RBF Learning algorithm	- Pattern classification - Function approximation - Control

Table 5.3: Learning algorithms.

number of non-linear hidden units are capable of implementing any continuous mapping with a pre-specified accuracy under certain mild conditions.

The second issue, sample complexity, determines the number of training patterns needed to train the network in order to guarantee a valid generalization. Too few patterns may cause the "over-fitting" problem where the network performs well on the training data set, but poorly on independent test patterns drawn from the same distribution as the training patterns.

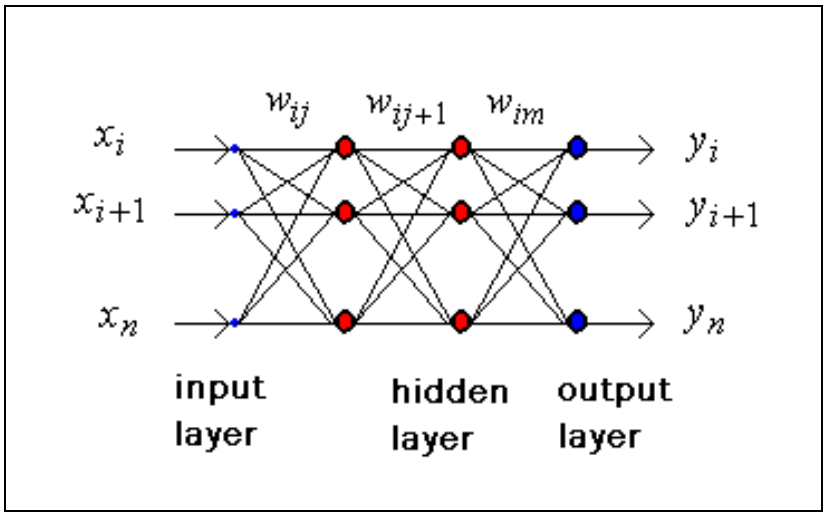

Figure 5.6: A typical 3-layer feedforward network architecture.

The third issue is the computational complexity of the learning algorithm used to estimate a solution from the training patterns. Many existing learning algorithms have a high computational complexity. For example, the popular backpropagation learning algorithm for feedforward networks is computationally demanding because of its slow convergence. Designing efficient algorithms for neural network learning is a very active research topic.

There are four basic types of learning rules as shown in figure 5.5: error-correction, Boltzmann, Hebbian, and competitive learning. Various learning algorithms and their associated network are summarized in Table 5.3. However this is by no means an exhaustive list of the learning algorithms available in the literature.

We notice that both the supervised and unsupervised learning paradigms employ learning rules based on error-correction, Hebbian and competitive learning. Learning rules based on error-correction can be used for training feedforward networks, while Hebbian learning rules have been used for all types of network architecture. However, each learning algorithm is designed for training a specific network architecture. Therefore, when we talk about a learning algorithm, it is implied that there is a particular network architecture associated with it. Each learning algorithm is also designed for performing one or a few specific tasks. The last column of Table 5.3 lists a number of tasks that each learning algorithm can perform.

5.4.4 Multilayer Perceptron

It has been recognized that multilayer feed forward networks are capable of forming arbitrarily complex decision boundaries and can represent any Boolean function [265]. The development of the back-propagation learning algorithm for determining weights in a multi-layer feedforward network has made these networks the most popular of all the networks.

Figure 5.6 shows a typical 3-layer perceptron. Is adopted by convention that the input nodes are not counted as a layer. In general a standard L-layer feedforward network consists of one input stage, $L - 1$ hidden layers, and one output layer of units that are successively connected fully or locally in

Step 1: Initialize the weights to small random values;

Step 2: Randomly choose an input pattern $x^{(\mu)}$;

Step 3: Propagate the signal forward through the network;

Step 4: Compute δ_i^L in the output layer $\left(o_i = y_i^L\right)$

$$\delta_i^L = g'\left(h_i^L\right)\left(d_i^\mu - y_i^L\right),$$

where h_i^L represents the net input to the i^{th} unit in the l^{th} layer.

Step 5: Compute the deltas for the preceding layers by propagating the errors backwards;

$$\delta_i^l = g'\left(h_i^l\right)\sum_j w_{ij}^{l+1}\delta_i^{l+1}$$

$$\text{for } l = (L-1),\ldots,1.$$

Step 6: Update weights using

$$\Delta w_{ji}^l = \eta\,\delta_i^l\, y_j^{l-1}$$

Step 7: Go to step 2 and repeat for the next pattern until the error in the output layer is below a pre-specified threshold or the maximum number of iterations is reached.

Table 5.4: Back-propagation algorithm.

a feedforward fashion with no connections between units in the same layer and no feedback connections between layers. We denote $w_{ij}^{(l)}$ as the weight on connection between the i^{th} unit in layer (l – 1) to j^{th} unit in layer l.

Recall that the task of a learning algorithm is to automatically determine the weights in the network such that a certain cost function is minimized.

Let $\left\{\left(x^{(1)}, d^{(1)}\right), \left(x^{(2)}, d^{(2)}\right), \ldots, \left(x^{(p)}, d^{(p)}\right)\right\}$, be a set of p training patterns, input-output pairs, where $x^{(1)} \in R^n$ is the input vector in the n-dimensional pattern space, and $d^{(i)} \in [0,1]^m$ is the desired output vector in the m-dimensional hyper-cube. For classification purposes, m is set to the number of classes. The squared-error cost function, which is most frequently used in the ANN literature [314], can be defined as

$$E = \frac{1}{2}\sum_{i=1}^{p}\left\|y^{(i)} - d^{(i)}\right\|^2 \tag{5.3}$$

The back-propagation algorithm is a gradient-descent method to minimize the above squared-error cost function in Equation 5.3. It is described in detail in the Table 5.4.

Multilayer feedforward networks with sigmoid activation functions can form smooth decision boundaries rather than piece-wise linear boundaries. There are many issues in designing feedforward networks. These issues include: how many layers are needed for a given task ?; how many units per layer?; what can we expect a network to generalize on data not included in the training net?; and how large should the training set be for "good" generalization? Although multilayer feedforward networks with a backpropagation algorithm have been widely used for classification and function approximation

[175], many design parameters still have to be determined by the trial-and-error method. Existing theoretical results only provide very loose guidelines for selecting these parameters in practice.

5.5 Intelligent Mobile Robots

Intelligent control is necessary for mobile robots that must survive in unstructured environments without continuous human guidance. Examples include robot sentries, intra-office delivery robots, and robot tour guides. Unlike the typical factory robot they may have to negotiate environments that are complex, changeable, full of obstacles and possibly hostile. Intelligent control is also necessary when there is a time lag between operator commands and robot execution. An example of intelligent adaptive control is the NASA's six-wheeled robotic rover, which was placed on the surface of Mars in July 1997 as part of the Mars Pathfinder project [251]. The rover had to be partly autonomous because of the large communication delay between Earth and Mars, where the time delay was between 6 and 41 minutes. The rover navigated between way-points specified by an earth bound operator, avoiding obstacles along the way. Another real motivation for the use of intelligent mobile robots in some specific situations can either have economic or security features. Unmanned landers have touched down on Mars for as little as US$ 250 million. But the estimated price tag for a manned journey to the red planet is estimated at around US$ 500 billion. Thereby further developments for spatial investigation require a more practical approach using robotics for unmanned missions [340].

Robots are already playing a growing role in complex tasks and hazardous environments. For example the U.S. arsenal includes intelligent robots to identify chemical and biological warfare agents. Since February 2004, the U.S. Defense Department has offered a US$ 1 million prize in a robot race with the aim to accelerate the development of autonomous robotic technologies [366]. The competition was launched by DARPA Defense Advanced Research Projects Agency, and the robot must complete a course comprised of both on-road and off-road segments which will include extremely rugged terrain and obstacles. The prize will go to the team whose robot fully and autonomously completes the course first.

Intelligent control is equally useful in telepresence applications, where a human operator issues high-level commands from a remote location and the robot complies using its own behavioural resources to implement the lower-level subtasks that are required. Furthermore a relatively new and speculative application of intelligent control is to control virtual agents. For example, in some situations an animator would prefer to be able to ask a virtual agent to "walk into the room and sit down" rather than specifying the detailed motions required for the action.

Numerous efforts have been made in developing "intelligent" programs based on Von Neumann's centralized architecture. Inspired by biological neural networks, researchers in a number of scientific disciplines are designing ANNs to solve a variety of problems in decision making, optimization, prediction, and control. Real-time sensory functions, process control, and motor control are the most meaningful tasks for neural computing. ANNs can be viewed as parallel and distributed processing systems that consist of a huge number of simple and massively connected processors.

An alternative for the design of intelligent controllers is the use of ANNs for "modular behaviour". The idea is that the controller contains modules which each perform some simple task oriented function. The robot's overall behaviour emerges from the interaction of these modules, rather than being specified explicitly. For example, Brooks' architecture consists of modules containing state machines and timers that each implement some simple behaviour [51][50]. This architecture has been used to make robust controllers for wheeled and legged robots, though the networks have to be carefully constructed to get the desired behaviour. Another example is Beer's artificial insect [38][36] [37]. It is a hexapod robot that is controlled by an artificial nervous system made up of about 80

biologically realistic neurons. The neural network is designed rather than trained: it contains groups of neurons dedicated to particular tasks, such as leg control, which interact to achieve overall coordinated movement. The robot is capable of walking with various gaits and performing simple tasks such as wall following and food finding.

These two examples, though different in implementation are similar in principle: they are designed from the bottom up, by adding units that interact with the existing structure to create new behaviour. Such systems are constructed especially to survive in their environments. They are not smart in the sense of classical AI, so they do not perform high level planning and problem solving. Instead they are smart in the sense that simple animals like insects are smart: able to robustly survive in, and interact appropriately with, their environments.

A lot of research has been done on using feed-forward neural networks as the adaptive component in a learning controller [262]. The network weights can be adjusted using the backpropagation algorithms, genetic algorithms [7], or various stochastic search algorithms [363] [373]. Supervised training is usually performed using error signals derived from the system's performance error, although other approaches that transfer expert information from a rule base are common. For example, Handelman trains a CMAC from a "knowledge base" to control a planar two-link manipulator [167]. A similar approach was implemented with the use of a fuzzy rule base [319] [197].

5.6 Conclusion

Various ANN models and learning algorithms have been successfully applied to a large variety of problems. Developments in ANNs have prompted a lot of enthusiasm as well as criticism. Many comparative studies provide an optimistic outlook for ANNs, while others offer a pessimistic view. For many tasks no single approach dominates the others. Thus the choice of the best technique should be driven by the nature of the given application. We should try to understand the capacities, characteristics, and applicability of various approaches developed in various disciplines, and maximally exploit the complementary advantages of these approaches in order to develop better intelligent systems. Such an effort may lead to a synergistic approach that combines the strengths of ANNs and other disciplines in order to achieve a significantly better performance for challenging problems [65][66]. Minsky has recognized that the time has come to build systems out of diverse components [264]. In such a synergistic approach, not only are individual modules important, but also a good methodology for integrating various modules is the key to success. It is clear that communication and cooperative work between ANNs and other methodologies will not only avoid repetitious work but also, more importantly, will stimulate and motivate individual disciplines.

Chapter 6

Evolutionary Techniques

6.1 Introduction

A mobile robot is more than just a mechanical device that can operate without being attached to a power supply or an external computer [196]. Although it may include those features, intelligent mobile robots are rather identified by the ability to adapt to their environment by finding optimal solutions, to develop a suitable control system, to define their low-level priorities, and possibly, to perform some self-monitoring [339].

As a reaction to partial failure of the classical artificial intelligence approach to develop robust control systems for intelligent robots by performing a functional decomposition [49], a novel approach called *behaviour-based* robotics emerged in the early 90's [47][244]. Whereas classical Artificial Intelligence (AI) is more concerned with a high-level definition of the environment and of the knowledge required by the system, behaviour-based robotics stresses the importance of continuous interaction between the robot and its environment by means of sets of reflexes applied in particular perceptual situations. Thus Subsumption Architecture was introduced as a behaviour-based approach, incrementally adding more situation-specific components to a control architecture for a more robust and adaptive robot performance [48]. However, this incremental design was done by hand, exploiting the designer's knowledge about the robot, the environment, and the task.

Within the behaviour-based methodology, a number of researchers have successfully employed an evolutionary approach to the development of control systems for the automation of the mobile robotics field [153][152][258]. The rich variety of structures that has been put under evolution, such as feed-forward neural networks, dynamic recurrent neurons, classifier systems and Lisp code, and the large number of evolved behaviours, such as locating food sources, obstacle avoidance, wall-following, object collection, etc, have empirically demonstrated the power and generality of the evolutionary methodology. However, evolved control systems may present robustness problems when environmental conditions change.

From the perspective of a control system, changes can be induced by several factors, among which are modification to the sensory appearance of objects, e.g. different light conditions, changes in

sensor response, re-arrangement of the environmental layout, transfer from simulated to physical robots, and transfer across different robotic platforms. Some authors have suggested ways to improve the robustness of evolved systems by adding noise [260][199] and by evaluating individuals in several different environments [357]. However, both techniques imply that one knows in advance what makes the evolved solution brittle in the face of future changes, in order to choose a suitable type of noise and environmental diversity.

Intelligent robots working in the real world must cope with changing conditions. In order to survive in such a dynamic environment evolved robots must show some adaptive features capable of coping with unpredictable new situations that differ from those encountered during the evolution process. Environmental changes can be a problem also for other approaches (programming, learning) to the extent in which the sources of change have not been considered during systems design. They are even more so for evolved systems because these often rely on environmental aspects that are often not predictable by an external observer.

For robotics automation one of the main advantages of evolution with respect to other adaptation methods, such as gradient descent techniques or reinforcement learning, is that the criterion function describing the desired behaviour need not be detailed, continuous, and differentiable [132]. Instead, you could model a range of desirable behaviours that the robot could decide on from a set of possible acts that would be better when applied to the presented situation [72]. Performance of Genetic Algorithms (GAs) has been compared with that of a back-propagation algorithm in different classification tasks [371]. The results showed that evolution is capable of generating better performance solutions with lower computational effort or processing time.

6.2 Brief Historical Review

Genetic Algorithms (GAs) have the aim of adaptive heuristics based on the evolutionary idea of natural selection and genetics. GAs have demonstrated good performance even when the problem dimension grows, and for this reason such methodology has achieved success in a large range of NP-complete and NP-Hard problems [259] [149]. GAs were first introduced as a parallel search technique based on the Darwinian principle of selective reproduction of the fittest individuals [180]. A GA operates on a population of artificial chromosomes, which are strings that encode the properties of a population of individuals. An artificial chromosome can be thought of as the individual's D.N.A., composed of a number of genes, each one containing a symbol or allele from a set of possible symbols. Although several kinds of encoding methods have been used, the most common one consists of encoding each gene of the chromosome using a set composed of two alleles, 0 and 1. This method is known as binary encoding [196].

GAs have achieved special success and emphasis as a heuristic technique applicable for complex problems covering the deficiency of deterministic methods, avoiding stagnation in a local optimum, thus providing a global near-optimum solution with lower computational effort for optimization problems [82]. Some of the many possible kinds of problems which could be solved with the use of GAs are discussed briefly next.

6.2.1 Robotics and Artificial Life Applications

Menczer has used steady-state GA to evolve sensory characteristics of artificial organism in an environment with controlled complexity [255]. The environment model used is called a latent energy environment. The behaviour of two types of sensors is interesting in this study: avoidance and

enforcement. Contact sensors are presented in the robot that is required to learn avoidance tasks. Reinforcement learning is used to train motor actions that are desirable. Ambient sensors are presented in the robot providing a feasible approach for a better interaction with the environment. Any changes in motor characteristic of this type of organism can only be achieved via evolution. Steady-state GA was used in such a study as follows. Each individual, represented by a string, must acquire energy from atoms in the environment beyond a fixed threshold before it can asexually reproduce. If the energy level within an individual is lower than the threshold, that individual will die. The chromosome of each individual contains two parts, one in floating-point format, and the other in binary format. Mutation is done by randomly added uniformly distributed noise to the chromosome. The types of atoms the sensors sensed are coded into the binary part.

Grefenstette has used GA to evolve rule sets in the SAMUEL system [156]. The evolving rule sets contain the rules for collision avoidance and the finding of energy resources. The SAMUEL system consists of two modules: the execution system module and the off-line system module. The execution system module contains the actual robot and environment. The off-line system module consists of a robot simulation module and a GA module for rule sets evaluation. The initial population is a heterogeneous population that is automatically generated. After the off-line learning is accomplished, the rule sets are tested on the actual system. Ramsey has modified this learning system to include real-time modification to the robot model [299]. This learning strategy is called case-based anytime learning.

Jakobi has introduced a new encoding scheme in GA with a hybrid approach where the evolutionary process is influenced by a recurrent neural network [200]. This neural network can be used as a robot controller. The robot controller can be used to control a robot to perform corridor following tasks and object avoidance tasks. Within each cell there is a genomic regulatory network (GRN). A GRN is composed of a number of units, with each unit containing a single string genome. One genome is responsible for the production of one protein. Protein that is produced by one unit regulates other genes in the different units. Proteins within each cell are divided into different classes that affect the gross behaviour of the cell. Signal proteins diffuse out of one cell and into another, resulting in an interaction between cells. Initially a single cell is placed in a controlled environment, which contains a number of predefined cellular developments including cell division and cell movement. Interaction between cells will eventually lead to cell differentiation. Once a cell is differentiated, a number of densities are grown out of each cell. When a dendrite from one cell contacts another cell, a synaptic connection is established. After every cell has been fully developed, thresholds and weights are assigned to each cell and dendrite, respectively.

6.2.2 Cellular Automata Applications

Cellular automata are an abstract way of analysing the simultaneous execution of local rules. A cellular space is a uniform array of cells arranged in some forms of topology and dimension. For cellular automaton (CA), each cell in the cellular space contains an identical automation. The next state of each automaton is defined by a function of its current state and the current state of other automata in a predefined neighbourhood. As discussed earlier (chapter 1), the idea of quantum computing, in which the elements that carry the information are atoms, has attracted the attention of many scientists. Quantum cellular automata and coupled quantum dot technology are being explored and their potential assessed for transistorless computing [103][100].

Andre has used genetic programming with automatically defined functions to produce a state-transition rule of linear cellular automaton for solving majority classification problems [5][6]. The

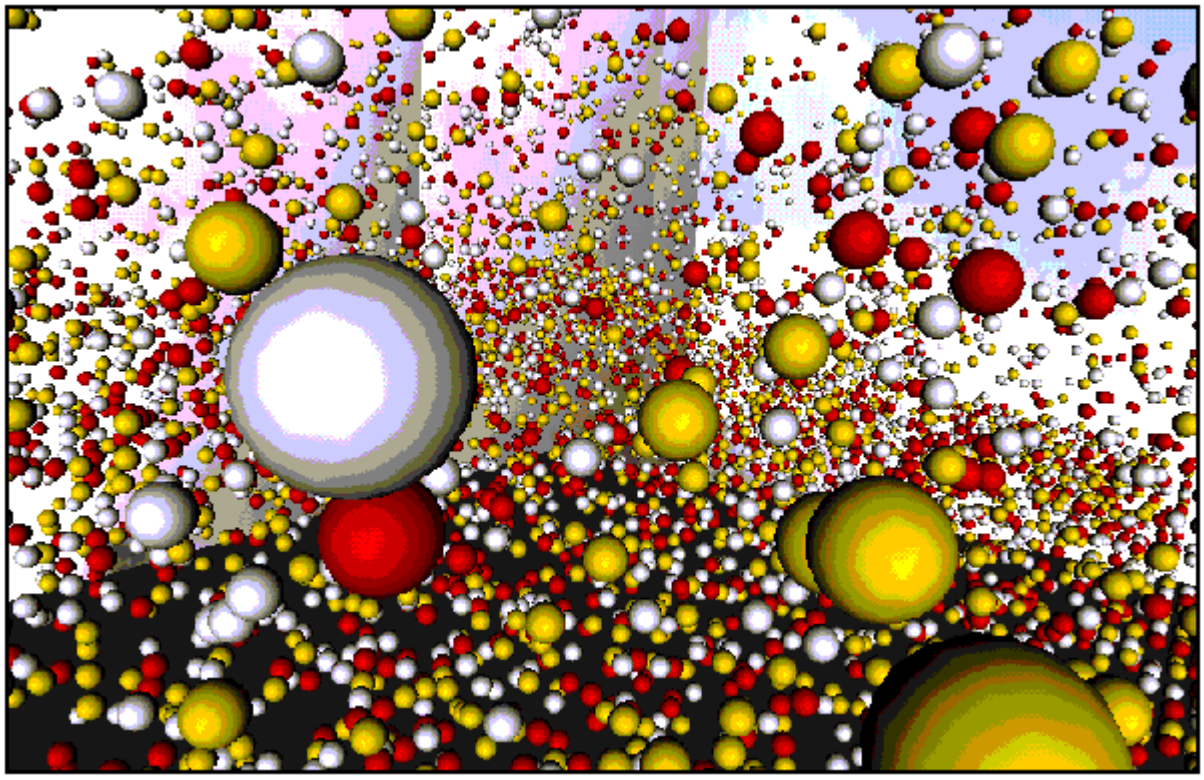

Figure 6.1: Navigation in the search space of a NP-Hard problem - each sphere represents a solution.

cellular space consists of 149 automaton in linear arrangement. Each automaton has either state 0 or state 1 at any given time.
The next state of each automaton depends on its own current state and the states of its six neighbouring automata, three to the left and three to the right. In this case, the program tree contains a result-producing branch and automatically defined functions. The resulting state-transition rule can solve the majority classification problem with a higher accuracy than the Gacs-Kurdyumov-Levin (GKL) rule and other known human-written rules.

Das has also studied the behaviour of cellular automata via the use of linear cellular automaton [97]. Unlike the work by Andre [5][6], Das uses GA to evolve the state-transition rules. A chromosome of each individual represents the output bits from all rules in a rule set in lexicographic order of neighbourhood configuration. Since in this case, each rule output depends on the states of seven cells, each individual will have chromosome of length 2^7. Das has applied this technique to majority classification tasks. Das has utilised the same technique on synchronisation tasks [96]. Further analysis of majority classification tasks and synchronisation tasks using cellular automata and GA can be found in Hordijk [184].

6.3 Genetic Algorithms Representation

Imagine any desired solution for some combinatorial problem, where if we start from a feasible solution, we could achieve another combination that results in a better solution to the problem under analysis. This analysis to find an optimal combination is done through the study of a set of possible solutions for the focused problem. The process for the resolution of searching a problem depends

Genetic Algorithms	Mathematics
1 population of solutions (individuals): chromosomes	1 set of solutions
1 chromosome	1 solution
1 gene	1 part of a solution

Table 6.1: Comparison between genetic algorithms terms with their correlation in math.

mainly on the environment where the search is happening, and what conditions need o be satisfied by the expected solution. However, it does not matter what methodology is applied to such combinatorial searching processes, because every search method will require at least:

- Some way to represent the possible solutions for the problem;
- Operators that could generate new candidate solutions;
- Evaluation of a generated solution;

According to its generality, the searching algorithms could be classified as weak, when they can be applied to a large range of problems, or strong, when they are projected into a short range of well specified applications, which make use of a greater quantity of information related to a specific situation. Gradually as the knowledge about the search space develops (Figure 6.1), the algorithms become specialized in the search of this specified space. Such algorithms grow in efficiency with a loss of generality, i.e. it is possible to use stronger or weaker searching algorithms.

Another important classification is the possible inclusion of some probabilistic component in a related searching algorithm. When a decision is generated from some value with a number randomly generated, we say that such an algorithm is stochastic, and in the opposite situation, we say that the algorithm is deterministic. In the first case the numeric random generator is always initialized with some value (e.g. supplied by the user) and therefore every algorithm execution could theoretically result in a different result.

Generally we could identify a thread-off that could be characterized as a prerequisite for a good searching algorithm: intensification and diversification. Intensification could be observed as the work of intensifying the local search, trying to get the best of all possible local solutions. The diversification is the work of exploring the global space for probable solutions. Thus some algorithm that has just one of these characteristics won't be able to obtain a good performance for NP-complete and NP-Hard problems. In optimization a good capacity of diversification in the search space is required if it is desired to find an extreme maximum/minimum global value, which should represent at least a good solution.

In GAs the *crossover* operator performs the recombination of two possible solutions, which will be used as the base for a new solution for a combinatorial problem. Normally this operator presents intensification characteristics to achieve the best solution, once there is a greater incentive for the recombination and generation of new son solutions from better adapted parents. This intensification tends to generate an accented loss of diversity, which is continuously equilibrated with the appliance of the mutation process in the descendent solutions.

The mutation alters the state of some components of some solutions. For the elaboration of the mutation algorithm, it is possible to determine what kind of mutation is to be used: a heavy mutation or a simple mutation. In the case of the problem under study we have chosen a simple mutation, where the state of only one of the components in a determined solution is inverted.

Genetic Algorithm Pseudo-code
Begin
- Generate an initial solution population with size *n*
- Evaluate the solutions in this solutions population

While < this generation does not converge >
Begin
do
Begin
- Select parents to produce the actual generation
- Apply crossover
- Evaluate the new solutions (individuals)
- Replace the old individuals in the population by new individuals

end
While < parameter not satisfied >
end
end

Table 6.2: Genetic Algorithm Pseudo-code.

The mutation change in the solution is a random choice on what solution and what component will suffer such a mutation, as is discussed in the following paragraph. When there is a loss of diversity, i.e. the population has indeed converged to a region that is near a local minimum, the GA applies a diversification through the use of a heavy mutation. Therefore a GA has as its main characteristic the property to avoid an optimum local, thus exploring a much greater range of points in the search space, which results in a greater probability of achieving an optimum global solution.

The GAs represent tool classes which are very versatile and robust, that are utilized for the solution of optimization problems.

In Table 6.1 is shown a set of GA expressions and their comparison with mathematical related terms. A GA applied to some determined problem must be composed from the following elements:

- A genetic representation for a feasible solution about some problem.
- A solution population.
- An evaluation function for the evolutionary population behaviour, what is called the "fitness" function.
- A genetic operator that generates new solutions, which is called the "crossover" operator.
- Diversification operator, which is called the "mutation" operator.
- Parameters definition, such as: population size, stop criterion, chromosome renovation criterion, and diversification criterion.

A pseudo-code which is capable of describing in general terms the existing GA as described in Table 6.2.

chromosse	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
solution	1	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0

Table 6.3: Encoding synaptic weights on a genotype are encoded as binary numbers.

chromosse	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
solution	42	90	13	8	51	2	23	10	84	39	62	70	15	38	20	75

Table 6.4: Encoding synaptic weights on a genotype are encoded as real numbers.

Organ Inlet	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Supply	0	0	1	1	0	0	0	1	1	0	0	1	1	1	0	0
State %	41	46	36	27	39	49	47	37	43	46	41	37	36	27	39	49

Table 6.5: Organ inlets representing the nanorobot attending decision at the current time.

6.4 Genetic Algorithms Codification

The first stage to resolve a specified problem using GAs consist in the establishment of a codification/representation fashion to each element in the search space of the model in question, once the GA engine will affect and operate on its codification, which itself represents a solution. The GA could be made using an integer or real codification of their variables. Hence, the solution can assume integer values or continuous percentage (tables 6.3 and 6.4).

Normally a problem solution is associated with a chromosome *p* represented by a vector or a list in the space $\Re^n : p = \{x_1, x_2, \ldots, x_n\}$ where each x_i represents a gene, which is a real variable that characterizes a problem solution. Generally an advantage of a real representation is its more intuitive conception, what makes possible and easier the appliance of prior knowledge derived from the appliance field, for the use of a crossover and mutation operator within the specified problem context [81].

For the problem under study where the nanorobots are required to operate a set of *n* organ inlets, a chromosome represents the configuration and state of an inlet organ at time step *t* in the scenery simulated. Supposing that we lead with a problem with $n = 16$ organ inlets, thus a chromosome is represented by a vector with 16 elements with each element corresponding the organ inlet nutritional level. The value "1" means that such organ inlet was scheduled at time *t* to be attended by the nanorobot *i*, and "0" meaning that such organ inlet is not included in the priority list to be attended at current time (table 6.5).

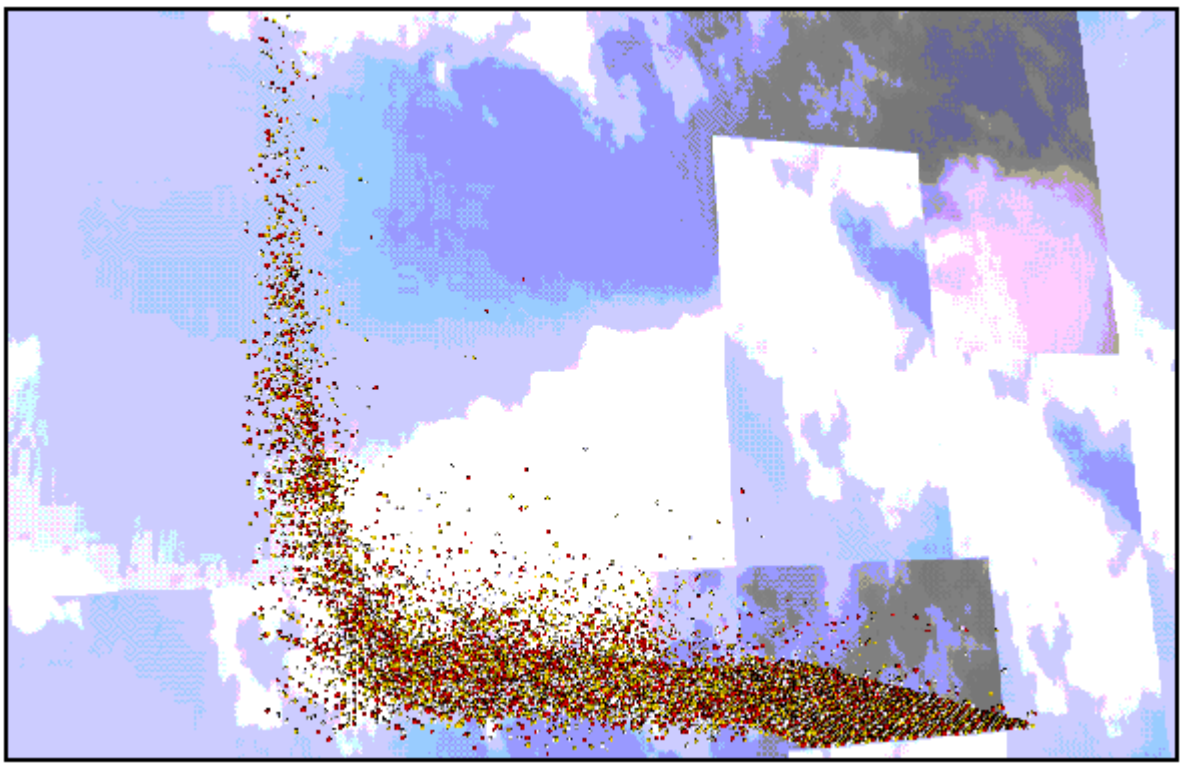

Figure 6.2: Evolutionary behaviour for minimization problems - each point represents a solution cost.

In the first line is represented the organ inlet identification, while in the second the supply's values "0" or "1" represents when the nanorobot has chosen to attend or not attend each determined organ inlet with a nutritional delivery in the current time step. In the third line the state represents the relative nutritional state of each organ inlet influenced by the nanorobot decision on attending or not attending the actual organ inlet demand.

6.4.1 Genetic Algorithm Initial Population

The population of a GA is a set of feasible solutions represented by a chromosome as a vector with size *n* composed of continuous or integer variables for a determined problem. To generate an initial GA population in most cases, this is done through a very simple procedure. Normally it is used from randomized up to heuristic algorithms for such an aim. For example, with the introduction of an "interesting individual" (i.e. solution) into the initial population, which could be an approximated known solution with some previous expert information, such an initial population could tend to converge faster to develop the best global solution.

Each execution of a GA can be repeated on a computer by just taking for it the same "seed" for the random number generator. Another seed value introduces a new sequence of random numbers and consequently will bring in a new initial population. To obtain a better insight about the GA behaviour on some specific application, all that we need is to do a statistic analysis on a set of results corresponding to different seed values. Observe that, independently of the seed value in use, a robust GA must experience little influence by such random mechanisms, thus returning every solution with a good result.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Father A	1	0	1	1	1	0	0	0	1	1	0	1	1	1	0	0
Father B	0	0	1	0	0	0	0	1	1	1	1	1	0	1	0	1
Son	1	0	1	0	1	0	0	1	1	1	0	1	0	1	0	0

Table 6.6: A crossover new solution generation.

6.4.2 Function of Chromosomes Evaluation

This function is responsible for the chromosome classification process and must identify the quality of each chromosome in the population. By comparison with mathematical programming, such a function is designated as the *fitness function* in a GA engine and is equivalent to an objective function that wishes to minimize or maximize some parameter. On the other hand it is also possible to make a multi-goal function where the model tries to maximize some α parameter while minimizing a β parameter at the same time.

The objective function must be determined in such a way that through a successive sample using an evolutionary process we could obtain some insight into which direction the optimal solution could be located. For example, an evaluation function that only returns a binary solution with a value "1" for just one right solution and "0" (such as "guess a keyword" game) for all other cases is incapable of being a GA application, once we have lost any kind of guidance engine for such a situation. Thus the search becomes completely blind and any feasible solution becomes merely a chance play.

6.5 Genetic Operators

Once defined, the representation of population elements is possible to construct genetic operators, which will be acting about a chromosome population. Therefore such operators are able to generate new individuals (solutions). The development of a genetic operator is very closely influenced by the solution representation for the original problem, which is encoded in a chromosome format. It means that for some solution *S* we have to encode it into a chromosome *p*, and after some updating performed in *p* by the genetic operator the new chromosome *p'* will be decoded to obtain the new solution *S'* for the associated problem. Basically there are two kinds of conventional genetic operators to generate new solutions, which are discussed next. In the sequence we also describe the roulette approach, which is the engine applied to choose parents for reproduction and mutation in the GA methodology.

6.5.1 Crossover

The expression *crossover* is normally applied in the evolutionary literature representing the operator that generates new solutions through combinatorial manipulations. These new solutions came from their parents. Thus the crossover operator aims to promote the genetic material recombination from distinct parents, in order to generate one or more "sons" (new solutions). Depending on the

Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Individual (P)	1	0	1	1	1	0	0	0	1	1	0	1	1	1	0	0
Individual (P')	1	0	1	1	1	0	1	0	1	1	0	1	1	1	0	0

Table 6.7: A "mutation operator" in action.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Individual (P')	1	0	1	1	1	0	1	0	1	1	0	1	1	1	0	0
Individual (P")	1	0	0	1	1	1	1	0	1	1	0	1	1	1	0	0

Table 6.8: Accumulative action for "mutation operator".

evaluation parameters of the *fitness function*, these new solutions could result in better or worse solutions for the problem under study.

The better one will be given a greater reproduction probability in comparison with the worst one, thus the population is going to achieve better solutions through the evolutionary process (Figure 6.2).

For the crossover operator two or more chromosomes are chosen and with a probability p_c, they are submitted to the crossover operation. One position in the recombination process is randomly selected and the parents' genetic material is recombined as demonstrated in the Table 6.6.

In this example the positions 2, 3, 6, 7, 9, 10, 12, 14 and 15 have the same value from both fathers. In this way, the values were copied automatically to the generated son. The father A was selected to transmit his characteristics to the positions 1, 5, 11 and 16. Meanwhile the father B was chosen for the positions 4, 8 and 13. For the positions where the fathers would have distinct values, the selection was done randomly with equivalent probability for each father.

6.5.2 Mutation

One or more operators for mutations are normally introduced in a GA engine. Normally they operate over an unique solution generating a new solution for a specified population. This new solution is then evaluated and then reincluded in the actual solutions population. The mutation operators modify one or more chromosome's genes, and such modification is based on a randomized process or on a pre-defined rule. Thus if we have an individual *P*, which belongs to a determined population, and the mutation operator has decided to alter one of the chromosome's genes values (e.g. in the 7th position), we would obtain a new individual with a modified solution *P'* (see Table 6.7).

In the same way the mutation operator could use a duplicate effect for the same solution, thereby generating a newer solution *P''* through a permutation with genes in the solution *P'*. For example, if we recombine the genes $x_3 = 1$ and $x_6 = 0$ we will have the result obtained in Table 6.8. The occurrence of such a mutation process over a specified gene (chromosome) is done in a probability manner p_m such that the incidence is small. The idea originated in genetic concepts and tries to preserve some genetic diversity in the population, which tends to lose such diversity when the

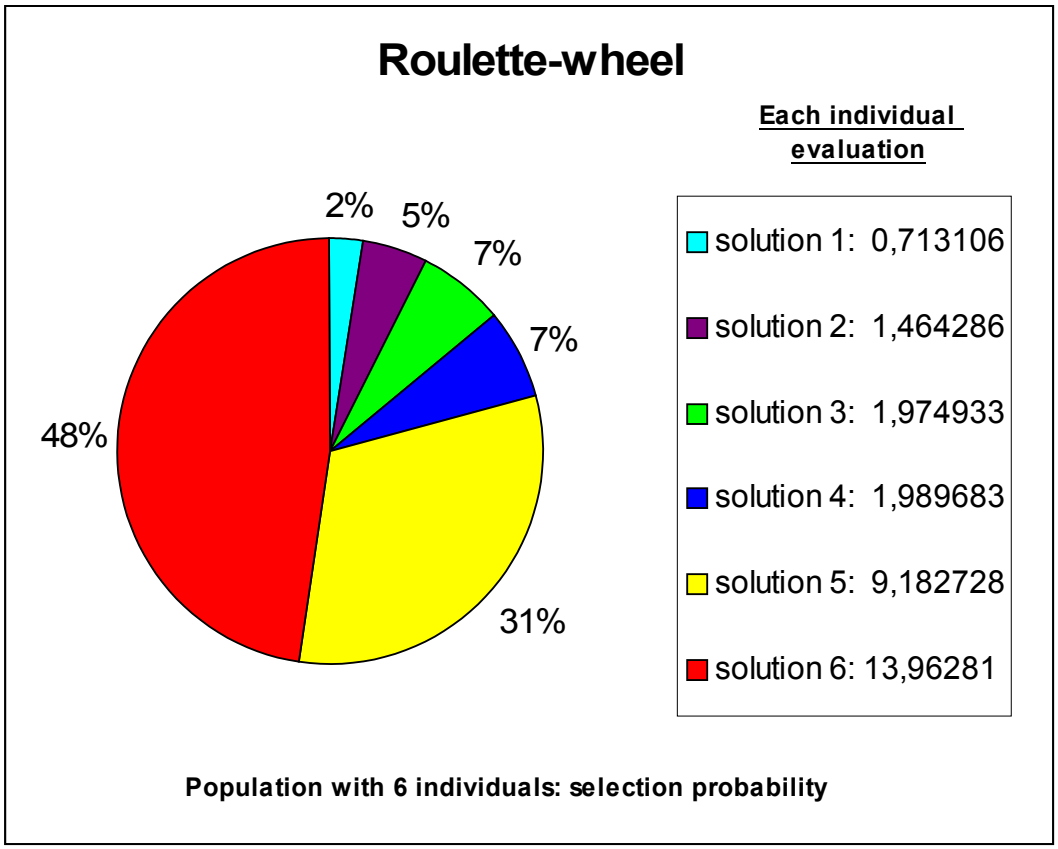

Figure 6.3: Roulette-wheel selection.

evolution goes forward and individuals become more and more similar, which tends to achieve a good solution for a determined problem for cases where the applied GA achieves success. Furthermore, when a significant loss of diversity occurs, the algorithm could use a heavy mutation action as a diversification alternative for this specific generation. This approach aims to achieve new regions in the search space with an increment of diversity in the population of the solution.

The mutation provides a diversification tool for the search space once it generates new individuals. These new individuals could reveal promising regions in the search for the best solution.

6.5.3 Roulette-Wheel Selection

The roulette-wheel is the process where it is chosen what parents will have the crossover or mutation operator applied. This choice is done in a randomized form based on probability intervals. These probability intervals observe the value relative to the fitness function evaluation for each individual related to its population. Thus in the GA classical literature the fitness function is closely related to the roulette-wheel selection. The fitness function by definition must be maximized. Therefore, the individuals (solutions) from a determined population that have a greater value are those that become nearest to the optimum point in the solution search space, which means that such solutions have a higher adaptability. For these individuals the roulette-wheel will always attribute a greater chance to be chosen in the process of generating a new population, thus they have a higher probability to become the fathers for the crossover operator.

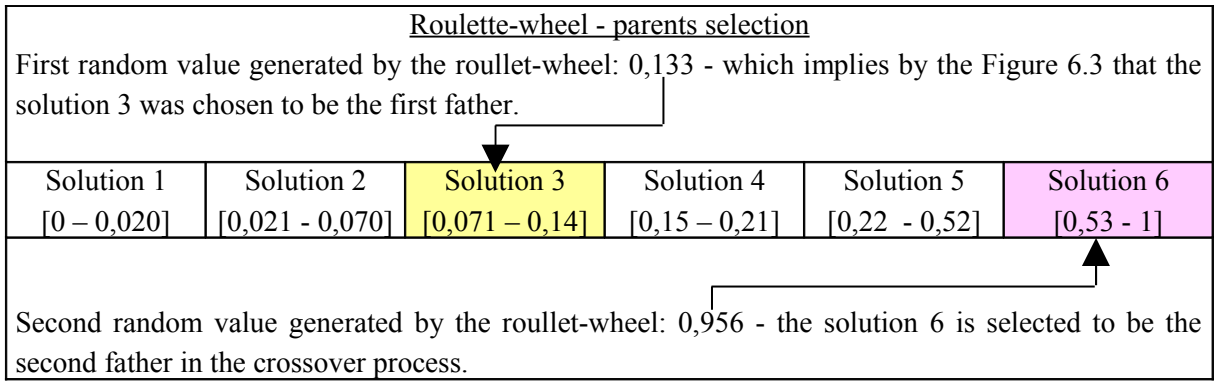

Figure 6.4: Roulette-wheel parents selection for the next crossover.

More precisely, the width of each slot is proportional to the fitness of the population. Since copies are made by spinning the wheel and selecting the individual corresponding to the slot where the ball stops, individuals with higher fitness have more reproduction chances. For example, the probability attribution for each individual is decided as follows: consider the evaluation value for all of those 6 individuals (Figure 6.3), where we have obtained the value 29,28755 as being the total sum from each individual fitness (solution value). Sharing the fitness value from a solution using the total sum of all those fitness values, we obtain the contributing fraction of this individual in relationship to the whole population's fitness value. Once having generated the probability intervals, we generate random numbers with a uniform distribution probability U[0,1], where this value will determine the father's choice for the next crossover (Figure 6.4).

This approach assures that better adapted individuals tend to have a greater probability of being chosen. However, individuals with lower values are also given a chance to be chosen for crossover. Obviously, they receive a lower probability in comparison with those with a higher fitness value. The justification for this approach, where individuals with lower fitness are also permitted to be chosen for the crossover process, is done in order to avoid too fast a convergence into a very similar population, locking the search into an optimum local solution, which would imply automatically in the loss of the best global solution that is what is really of interest in our optimization search.

In general terms, we could affirm that heuristic methods show a satisfactory performance, when we must lead with combinatorial problems [365]. In the majority of those cases a heuristic approach results in excellent near optimal solutions with significantly lower computational effort for a large range of problems.

6.6 Parameters Definition

In GA there are many parameter adjustments, such as crossover proportion, mutation proportion etc. For the crossover proportion, we mean what percentage of a population will be involved in the recombination process for each generation. The mutation proportion determines with what probability an individual will suffer a mutation in one of his genes in each generation. Experimental results have demonstrated that for crossover weight values from 60% up to 80% demonstrated good performance, while for the mutation the suitable values are quite small, i.e. generally as small as 5% [80].

An important question also is related to the suitable population size that must be used in a GA. Experimental tests have demonstrated that the greater the population size, the better will be the achieved solutions. On the other hand, the greater the defined population size, the greater will be the computational time required for each complete population iterations. Another important parameter for

the GA engine definition is the number of iteration, i.e. the stop criterion. As well as the prior parameter, this also has an inverse proportion related to solution improvement versus a greater computational effort consumption.

The adjustment of such parameters are in the majority of cases better specified for each situation, thus having to observe some specific restriction related to the problem under study, could be the better option. For example, some real time problems must have a very fast answer from GA leading to a control system response, therefore the time required for processing must be lower for a smaller population. Such a parameter could be completely different for a short time decision system, which requires a solution for the next 1 hour and so forth. What happens in many cases is the use of good sense and previous knowledge about the problem in question for the GA parameter specification.

6.7 Conclusions

A fundamental requirement for intelligent mobile robots is a decision by the robot on how to interact with its environment. Many situations require the robot to decide what kind of task demands a higher priority than another, and so forth. Approaches based on the classical paradigms were not completely suitable for unpredictable and dynamic environments. Other approaches consider this reaction as the new paradigm to build intelligent systems. One classical instance of this kind of architecture is the subsumption architecture that was proposed by Brooks and has been successfully implemented on robots of many research institutes [47]. The base of the subsumption architecture is "behaviour". Each behaviour reacts in a particular situation and the global control is a composition of such behaviours. Different systems, from finite state machines to fuzzy controllers [198], have been used for the implementation of these behaviours, and the rules of these behaviours may be designed by a human expert, designed "ad-hoc" for the problem, or learned using different artificial intelligence techniques.

Machine learning has been applied to shape the behaviour of adaptive agents. Some of these techniques become inapplicable for learning about reactive behaviour problems because they require more information than the problem constraints allow. Thus, it would seem reasonable to use an automatic system that gradually builds up a control system for a robust and dynamic agent by exploiting the changing interactions between the environment and the agent itself. Some approaches use GAs [249], Classifier Systems to learn controllers [317][270] or Neural Networks to adjust behaviours to attend to real time environmental demands [271].

Dynamic and interactive concepts are crucial in the case of mobile robots. Unlike robotic manipulators, mobile robots often operate in open, unpredictable, and dynamic environments. These dynamics between the robot and its environment are not easy for a human designer to completely analyse. Instead a better approach is to define a robust high level range of instructions and goals that must be accomplished by the robot. In this scenario, one cannot program a robot to move along a given path, but instead, must allow the robot the possibility of making low-level decisions depending on the conditions encountered in the surrounding environment. Learning and evolutionary approaches [65] can exploit these kinds of interaction complexities at high and low decision levels to generate a satisfactory adaptive behaviour, as has been demonstrated in recent works [73][69][66].

Chapter 7

Parallel Processing

7.1 Introduction

Computer animation, and in particular physical simulation, can impose extreme loads on processing hardware. Motion dynamics and collision detection for complex environments where mobile robots need to be interactive with each other and the objects surrounding them, can be the most computationally intensive stage of the production process of an animation sequence. Methods for decreasing the time spent on such activities must be a major priority if future requirements for complex, interactive scenes are to be met.

Despite the expected future increase in available processing power, demands will always tend to exceed the capabilities of current state-of-the-art machines. Although these are significant improvements in processing speed, sequential processors are far from rendering sufficient computing capacity for an advanced robot system. Any methods that increase the efficiency and speed of generation of physical simulation for animation are therefore of great importance [78]. In several cases, fundamentally new concepts have to be developed, so that a parallelization is possible. The use of parallel processing can offer potential increases in processing speed in proportion to the number of processors used.

7.2 Parallel Processing Characterisation

In order to use a parallel system, it is important to demonstrate the feasibility of parallelizing existing problem solutions in robotics. In several cases, fundamentally new concepts have to be developed, so that a parallelization is possible. is surveyed in . The historical development of control structures of automated manufacturing has influenced specially designed computer architecture for robot control [102][154].

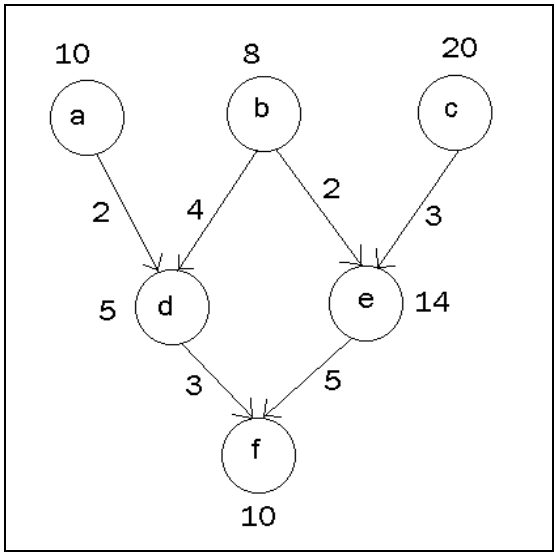

Figure 7.1: Directed task graph.

A classification scheme for robot control architecture has been proposed to cover the extreme viewpoints of the historical development, hierarchical and distributed control [185]. Additionally, function-oriented and behaviour-oriented approaches are distinguished. Altogether, this results in four different classes. For parallel processing each function or each behaviour can be performed by an extra processing element (PE). Thus computer programs that could be running distinct PEs in a multiprogramming environment should be identified as suitable applications for parallel processing techniques and will be described as a flow shop problem in the sequence.

In general flow shop problems there may be n jobs each requiring m tasks T_{1i}, T_{2i}, ..., T_{mi}, $1 \leq i \leq n$, to be performed. Task T_{ji} is to be performed on processor $P_j, 1 \leq j \leq m$. The time required to complete task T_{ji} is t_{ji} . A schedule for the n jobs is an assignment of tasks to time intervals on the processors. No processor may have more than one task assigned to it in any time interval. Additionally, for any job I the processing of task Tji, $j > 1$, cannot be started until task $Tj\text{-}1$, i has been completed. A non-preemptive schedule is a schedule in which the processing of a task on any processor is not terminated until the task is complete. A schedule for which this need not be true is called preemptive. In our case, we are dealing with a non-preemptive model.

The *finish time*, $f_i(Z)$, of job i is the time at which all tasks of job i have been completed in schedule Z. The finish time value $V(Z)$, of a schedule Z is given by

$$V(Z) = \max_{1 \leq i \leq n} \{f_i(Z)\} \tag{7.1}$$

The mean flow time $M(Z)$, is defined to be

$$M(Z) = \frac{1}{n} \sum_{1 \leq i \leq n} f_i(Z) \tag{7.2}$$

An optimal scheduling finish time for a given set of jobs is a non-preemptive schedule Z for which $V(Z)$ is minimum. The general problem of obtaining an optimal finish time for parallel processing scheduling is computationally difficult with dynamic programming.

$$Pred(b_f) = x_{f-1}$$

$$Succ(b_f) = \{u_{f_i} \mid i \in I\}$$

$$U = \{u_{f_i} \mid i \in I, Pred(u_{f_i}) = b_f, Succ(u_{f_i}) = \{s_{f_h} \mid h \in I\}\}$$

$$S = \{s_{f_i} \mid i \in I, Pred(s_{f_i}) = \{u_{f_i} \mid h \in I\}, Succ(s_{f_i}) = \{r_{f_g} \mid g \in I\}\}$$

$$R = \{r_{f_i} \mid i \in I, Pred(r_{f_i}) = \{s_{f_h} \mid h \in I\}, Succ(r_{f_i}) = x_f\}$$

$$Pred(x_f) = \{r_{f_i} \mid i \in I\}$$

$$Succ(x_f) = b_{f+1}$$

Table 7.1: Predecessors and successors set of tasks description.

To our problem we could represent a job with a directed task graph. Jobs could be sometimes application problems, and sometimes part of the application problem. Figure 7.1 shows an example of a task graph. Each node of the graph stands for a task. The letters inside the node show the task identification and the value next to the node shows the processing time for that task. The arcs from one node to another indicate communication between tasks and task precedence. For example, task *d* communicates with task *a*, task *b* and task *f*, and is not executable until both task *a* and *b* have been completed. The number beside an arc shows the communication cost between the tasks. Since a loop structure in the job can be expanded to a set of tasks without any loop, we assume that the task graph has no directed cycle, and is, therefore, a directed acyclic graph.

We assign tasks to processors and determine the execution order of the tasks, so that the total computation time is minimized and the assignment constraints are satisfied. The assignment of tasks implies a partition of tasks into subsets, where each subset contains the tasks assigned to a processor. From the relation between the assignment constraints and the computation time, we can choose the appropriate architecture. The relation between the number of processors and the total computation time shows whether the problem is appropriate for parallel processing with that type of architecture. Generally if the computation time decreases in inverse proportion to the number of processors, parallel processing is a good solution for the problem [134]. On the other hand, if the computation time is independent of the number of processors, as in the case of a simple linear task graph, parallel processing is not applicable to the problem.

7.3 Processing Requirements of the System

An appropriate notation is required to analyse formally the problem of applying a parallel processing solution to computer interactive robotics control in a graphic environment. It should provide the method for assessing the suitability of applications for parallel processing and a way of mapping the tasks of the problem domain to processing elements [227].

$$J(i) = \{j \mid j = i+1, i+2, ..., i+(n-1)\}$$

$$K(i) = \{k \mid k = i+1, i+2, ..., i+(n-1)\}$$

$$M(k) = \begin{cases} k \geq 0 \rightarrow m = k \\ k < 0 \rightarrow m = k + n \end{cases}$$

$$U' = \{u'_{f_i} \mid i \in I, Pred(u'_{f_i}) = b_f, \quad Succ(u'_{f_i}) = \{\{s'_{f_{j \bmod n}} \mid j \in J(i)\}, s'_{f_i}\}\}$$

$$S' = \{s'_{f_i} \mid i \in I, Pred(s'_{f_i}) = \{\{u'_{f_m} \mid m \in \{M(k) \mid k \in K(i)\}, u'_{f_i}\}\},$$

$$Succ(s'_{f_i}) = \{\{r'_{f_m} \mid m \in \{M(k) \mid k \in K(i)\}, r'_{f_i}\}\}$$

$$R' = \{r'_{f_i} \mid i \in I, Pred(r'_{f_i}) = \{\{s'_{f_{j \bmod n}} \mid j \in J(i)\}, s'_{f_i}\}\}, \quad Succ(r'_{f_i}) = x_f\}$$

Table 7.2: The revised task predecessor and successor relationships.

7.3.1 Analysis of the Task

A task graph G for the process of producing a single frame of an animation sequence can be defined as:

$$G = (T, C, e, h, d) \tag{7.3}$$

$$R^* = \{x \mid x \in R, x \geq 0\} \tag{7.4}$$

where:

T: the set of tasks;

C: the set of task communications: $C \subset T \times T$;

e: a task execution time function: $e : T \rightarrow R^*$;

h: a communication overhead time function: $h : C \rightarrow R^*$;

d: a communication delay time function: $d : C \rightarrow R^*$;

*R**: is a set of non-negative real numbers.

In the case of the generation of a single frame of animation, *f*, for a system of *n* "agent" objects (which represents our nanorobots), *T* is defined as:

$$I = \{i \mid 0 \leq i \leq (n-1)\} \tag{7.5}$$

$$T = \{b_f, x_f, U, S, R\} \tag{7.6}$$

where:

b_f: the initial trigger for frame f;

x_f: the final synchronisation for the frame;

U: the set of initial object update tasks, $\{u_{f_i} \mid i \in I\}$;

S: the interaction response tasks, $\{s_{f_i} \mid i \in I\}$;

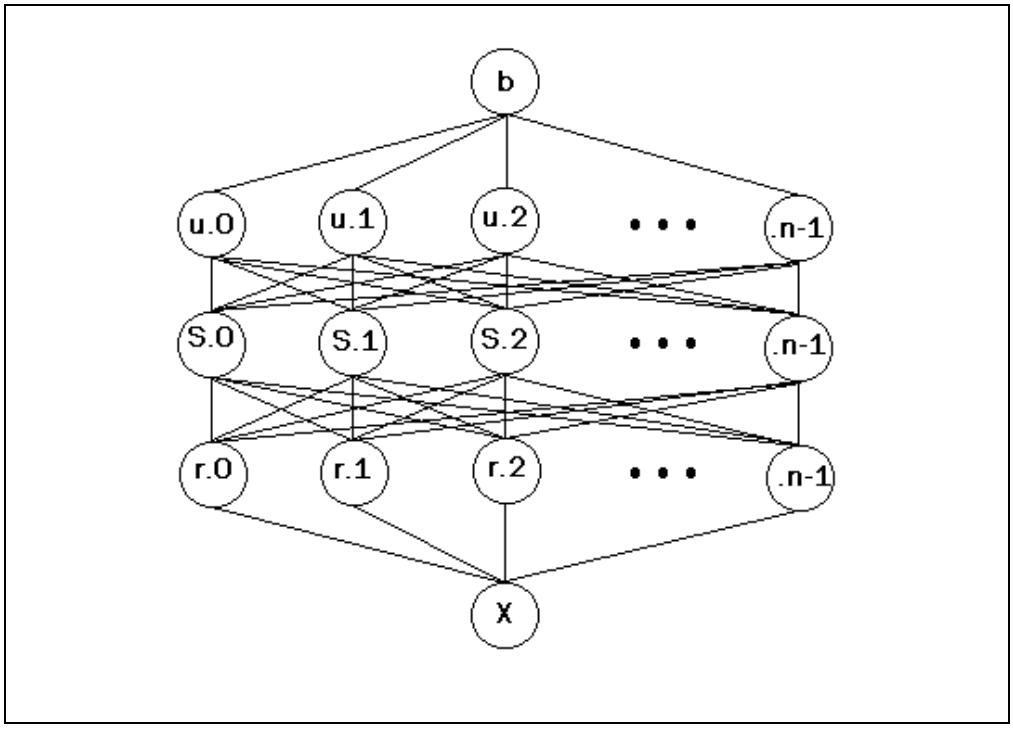

Figure 7.2: Task graph for *n* PEs system with modified sense and response stages.

R: the interaction response tasks, $\{r_{f_i} \mid i \in I\}$;

The predecessors and successors for the tasks can be identified in Table 7.1.

This gives the task graph shown in the Figure 7.2. This represents the worst-case scenario, since every object in the system receives data from every other object to provide it with the data for the sense stage. If the operation of the sense and response stages is implemented as commutative operations, then they need only be carried out for each pair of objects once. It is then possible to divide the sense and response stage work for the entire system equally amongst the agent objects [291].

This results in the replacement of the naive *s* sense tasks with a new *s'* sense task that processes data from (n-1)/2 other objects. It must be remembered, however, that each of the u_{f_i}, s_{f_i} and r_{f_i} tasks refer to the same object, i.e. they have common data. Therefore in addition to each of the (n-1)/2 assignments for each task there is the additional task successor corresponding to the present task for each object. The new predecessor and successor requirements are then given by the relationships in Table 7.2.

It is now apparent that the communications requirements for the successors of the tasks in *S'* are the same as those of these predecessors. A processing element assignment can now be defined that is consistent with the task graph.

Since the tasks *R'* are indirect successors to the set of tasks *U'*, and there is no direct communication requirement between the members of *U'* and those of *R'*, there is no direct relationship between the processor assignment function. There is a requirement, however, for the relationship between *pa* (predecessor assigned task) for each element in *U'* and each element of *S'* and each element of *R'*.

$$W = (P, L)$$

$$P = \{1,2,..., p\}$$

$$L = \{p_1, p_2 \mid p_1, p_2 \in P, p_1 < p_2, \exists t_1, t_2 \in T,$$
$$(pa(t_1 = p_1) \wedge (pa(t_2) = p_2) \wedge (((t_1, t_2) \in C) \vee ((t_2, t_1) \in C))\}$$

$$T_a = \{t \mid t \in T, (pa(t) = p_t) \wedge (p_t \in P),$$
$$c \in Succ(t), (pa(c) = p_c) \wedge ((p_t, p_c) \in L), d \in$$
$$Pred(t), (pa(d) = p_d) \wedge ((p_t, p_d) \in L)\}$$

Table 7.3: Processor network assignment relationships.

These relationships can be expressed for a processor network *W*, with a set of *p* processors, *P*, and a set of inter-processor links *L*. This is given in Table 7.3, and merely specifies the requirement that each task assigned to a particular processor has a communication channel to its predecessors and its successors.

Since these have been defined, *pa* for each set of tasks can be defined. It is assumed that it is desirable to minimise the number of inter-processing element links. Hence, since the successors of a given member of *S'* are in the same set as its predecessors, it is logical to assign the predecessors and successors of each member of *S'* to the same processor. To start with, assign each element of *S'* to a particular processor:

$$\forall s' \in S', pa(s') = p_s \tag{7.7}$$

Since, for each *i*, u_{fi}, s_{fi} and r_{fi} execute consecutively and share data, it is logical to implement them on the same processing element. With the other predecessor and successor relationships, the assignments become the general expression detailed in Table 7.4. Where in Table 7.4 L_a is the set of actual links required for a given set of processor assignments. The link requirements, L_a, for any processor that has an assigned task u_{fi} is that there must be a link connecting that processor to the processor s'_{fj}, where s'_{fj} is a successor to u'_{fi}. Since $pa(u'_{fi})$ has already been defined as equal to $pa(r'_{fi})$, this specifies all the inter-processor links for the task graph. So any suitable network of processing elements *P*, will have:

$$L = \{l \mid i \in P, \{j \mid j \in J(i), l = (p_i, p_j)\}\} \tag{7.8}$$

with a suitable architecture and task assignment algorithm defined, the implications for processing performance can be analysed.

$$\forall_i \in I, \forall_j \in I, pa(u'_{f_i}) = pa(s'_{f_i}) = pa(r'_{f_i}) = p_i$$

$$(pa(s'_{f_i}) = p_i) \wedge ((pa(\mathrm{Pred}(s'_{f_i})) = pa(Succ(s'_{f_i})) = p_j)) \rightarrow ((p_i, p_j) \in L_a)$$

$$L_a = \{l \mid i \in I, pa(u'_{f_i}) = pa(r'_{f_i}) = p_i, \{p_i \mid j \in J(i), pa(s'_{f_j}) = p_j, l = (p_i, p_j)\}\}$$

Table 7.4: Task assignments general expression.

$$TM_s(t_1) \equiv \begin{cases} 0 : \mathrm{Pred}(t_1) = NULL \\ \max\limits_{t_2 \in \mathrm{Pred}(t_1)} (TM_e(t_2) + d((t_2, t_1))) : otherwise \end{cases};$$

$$TM_e(t_1) \equiv TM_s(t_1) + \sum_{t_2 \in \mathrm{Pred}(t_1)} h(t_2, t_1) + e(t_1) + \sum_{t_3 \in Succ(t_1)} h(t_1, t_3),$$

$$TM(G, P, A) \equiv \max_{t \in T}(TM_e(t)) = \max_{t \in \{u \mid u \in T, Succ(u) = NUL\}} (TM_e(t))$$
$$= TM_e(x)$$
$$= \max_{r \in \mathrm{Pred}(x)} (TM_e(r) + d(r, x) + \sum_{r \in \mathrm{Pred}(x)} h(r, x) + e(x);$$

$$TM_e(r) = \max_{s \in \mathrm{Pred}(x)} TM_e(s) + d(s, r) + \sum_{s \in \mathrm{Pred}(r)} h(s, r) + e(r) + \sum_{x \in Succ(r)} h(r, e),$$

$$TM_e(s) = \max_{u \in \mathrm{Pred}(s)} TM_e(u) + d(u, s) + \sum_{u \in \mathrm{Pred}(s)} h(u, s) + e(s) + \sum_{r \in Succ(s)} h(r, s),$$

$$TM_e(u) = TM_e(b) + d(b, u) + h(b, u) + e(u) + \sum_{s \in Succ(u)} h(u, s),$$

$$TM_e(b) = e(b) + \sum_{u \in Succ(b)} h(b, u),$$

Table 7.5: Task-processor start and end times of a task.

7.3.2 Implications for Processing Performance

The system that results from the assignment algorithm can be assessed for its performance advantage over a sequential system. The start and the end times of a given task t and the computation time for a task could be defined by a graph *G* [227], a set of processors *P* and a set of task-processor assignments are expressed in the Table 7.5, along with the final computation time requirements.

The processing time of the production process for a single frame in a sequential system is given by the formula:

$$TM_{seq} = e(b) + \sum_{i=0}^{n-1} (e(u_i) + e(s_i) + e(r_i)) + e(x) \tag{7.9}$$

where:

$$
\begin{aligned}
TM_{par} = {} & e(b) + \sum_{i \in I}((h(b,u_i) + \max_{i \in I}(d(b,u_i) + h(b,u_i) + e(u_i) \\
& + \sum_{j \in J(i)} h((u_i, s_{j \bmod n}) + \max_{j \in j(i)}(d(u_i, s_{j \bmod n}) \\
& + \sum_{m \in M(K(j))} h(u_{m \bmod n}, s_{j \bmod n}) + e(s_{j \bmod n}) \\
& + \sum_{m \in M(K(j))} h(s_{j \bmod n}, r_{m \bmod n}) + \max_{m \in M(K(j))} d(s_{j \bmod n}, r_{m \bmod n}) \\
& + \sum_{v \in J(m)} h(s_{v \bmod n}, r_{m \bmod n}) + e(r_{m \bmod n}) + h(r_{m \bmod n}, x))) \\
& + d(r_i, x) + + \sum_{i \in I} h(r_i, x) + e(x)
\end{aligned}
$$

Table 7.6: Processing time for one processor per task assignment.

$e(b)$ = time taken by task *b*.

$e(u_i)$ = time taken for each update task *u*.

$e(s_i)$ = time taken for each sense task.

$$e(s_i) = \sum_{j=0}^{n-1} e(s_i''(j)) \tag{7.10}$$

$e(s_i)$ is the time to process data corresponding to each other object.

$$e(r_i) = \sum_{k=0}^{n-1} e(r_i''(k)) \tag{7.11}$$

$e(x)$ = time taken for task *x*.

$e(r_i)$ is the time to produce a response for each other object.

For the parallel case *I* which the tasks are assigned $(n-1)/2$, the minimum processing time (i.e. the case where there is one processing element per assignment) is given in Table 7.6.

Hence the processing performance increase that can be expected depends on *h* and *d* for each pair of communicating processes and on the relationship of the maximum time for each stage with the mean time for each stage. For a system where *d* and *h* are independent of the task-processor assignments, the best possible speed-up will be when each stage takes an equal amount of time, and if this is the case the corresponding sequential and parallel computation times are given by the relationships in Table 7.7.

This represents an order *n* speed-up for the update, sense and response phases, in a situation with *n* processors. This, of course, is the maximum that could be expected. What is more important is that the communication overhead is of order *n* too, when the communications between tasks provide an effective inter-object data exchange of order n^2. Hence, although the actual speed up will be reduced by the communication overhead, this overhead only increases in proportion to the number of object tasks run concurrently. For the general case of *p* processing elements:

$$
\begin{aligned}
&\forall\, i \in I, e(u_i) = e(u), e(s_i) = e(s), e(r_i) = e(r) \\
&\forall\, t_1, t_2 \in T, h(t_1, t_2) = h, d(t_1, t_2) = d \\
&TM_{seq} = e(b) + n(e(u)) + e(ns'') + e(nr'') + e(x) \\
&\quad = e(b) + ne(u) + n^2 e(s'') + n^2 e(r'') + e(x) \\
&TM_{par} = e(b) + nh + d + h + e(u) + ((n-1)/2)h + d + (((n-1)/2)h \\
&\quad + e(((n-1)/2)s'')) + ((n-1)/2)h + d + ((n-1)/2)h \\
&\quad + e(((n-1)/2)r'') + h + d + nh + e(x) \\
&\quad = e(b) + e(u) + ((n-1)/2)e(s'') + ((n-1)/2)(r'') \\
&\quad + e(x) + 4((n-1)/2)h + 2(n+1)h + 4d
\end{aligned}
$$

Table 7.7: Sequential and parallel computation times for equal processing per object task.

$$
\begin{aligned}
TM_{par} = {} & e(b) + (n/p)(e(u) + ((n-1)/2)e(s'') + ((n-1)/2)e(r'') + e(x) \\
& + 2ph + 2(n/p)h + 4(n/p)((n-1)/2)h + 4d
\end{aligned}
\tag{7.12}
$$

So for p processors, the update, sense and responses stages can offer an order p speed-up. In many animation applications these are the major tasks in the system, certainly this is the case in physical simulation systems [70]. In other applications, for example in the growing field of intelligent agents [46][58][158][74][77], these will also be the most processor intensive tasks since it is in these that the majority of data processing is going on to produce the final animation sequence [66]. This method therefore offers an important performance improvement.

7.4 Parallel Processing for Robotics Control

Distributed high-level robot control systems make use of functional parallelism by dividing the system into functionally different modules that run on different PEs. A system with high efficiency can be achieved by concentrating on large and efficient software blocks with little inter-modular communication [20]. Those systems typically provide extensible inter-processor communication bandwidths. In the following, some classes will be distinguished:

- *Shared memory systems* communicate through a common memory. An efficient system hierarchically organized can solve the dynamics of a robot [154]. Besides analyzing the different buffering strategies, this work investigates pipelining the data flow for achieving maximum parallelism.
- Another possibility for running distributed robot task controllers that are independent from the network structure is the commonly used *distributed operating systems*. They have the advantage that basic communication and coordination functions are provided at the system level. Systems like DCE [328], therefore, provide a mechanism for *remote procedure calls*. With this simple form, only synchronous communication is supported and deadlocks are possible.

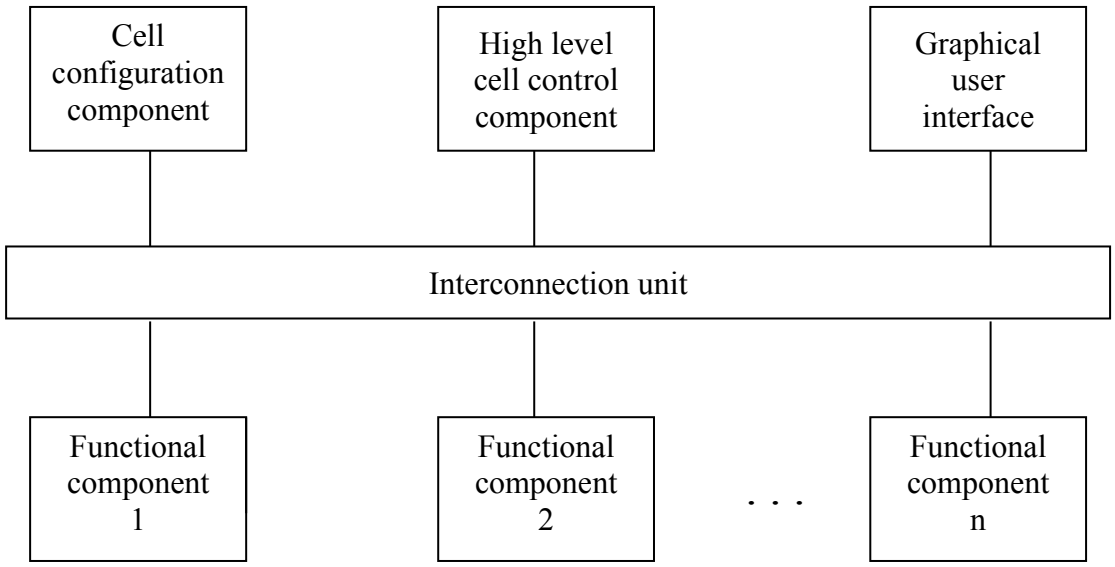

Figure 7.3: Concept for parallel robot control architecture consisting of multiple components and an interconnection unit.

Even more convenient to program are *distributed object-oriented systems* [217][76][69]. They offer all the possibilities of common distributed systems and support remote object access. Though often remote object access can be used asynchronously, the system performance still is dependent on a low level communication structure and protocol [146]. A flexible software architecture was designed, which allows the software to be easily adapted to system changes [65][66]. Some concepts could be observed in most parallel systems:

- Having a component called "Workcell Manager" which orchestrates most of the cooperation activities with the other components.
- Fixed cycle patterns and fixed programs for the interaction of the components solving a common task.
- Interfaces of the components defined with a few simple and common operations, and components designed with as few characteristics as possible about the cell.

More loosely coupled system architecture is defined by the *multi-agent systems* [229] and especially the blackboard systems [286]. The subsystems called *agents* are of similar size to the components of our concept and also use asynchronous communication. Another system architecture similar to the adopted approach is the distributed simulation system [254], where a distributed and flexible system divided into functional subsystems is suggested, which can be run in parallel. One main problem investigated is the temporal and functional consistency of the world model. The data-flow must correspond directly to the flow of the material. Temporal consistency is monitored and managed by a timer component. Such a system may work well for advanced manipulation tasks, like cooperating robot manipulators.

7.4.1 Parallel Architecture

Before discussing parallel control architecture, it is important to explain what control architecture is. According to [102], control architecture makes a control system from control components. The architecture determines the interrelationships between the components and the mechanisms for coordination.

Requirements on robot control architecture can be described from a general point of view [125], for manufacturing systems [131], and for software architectures of robot control [131]. Important requirements from the parallel processing point of view are required to attend flexibility [173]. The goal of our system concept is to provide scalable and flexible high level robot control architecture for a complex manipulation task in a 3D virtual environment with real time adaptive reaction. In order to provide high efficiency, our system is divided into subsystems with different functionality that may run on different PEs and communicate by an efficient message passing protocol.

A subsystem is also called a component. A component implements a set of related functions and can either be a physical or a logical component [254]. Logical components run as different processes in order to have the possibility to run them on different PEs for higher efficiency. Additionally, a component process can be parallelized at the algorithm level like automatic motion and decision control (see chapter 8).

The system of components from this abstract point of view is shown in Figure 7.3. Each component is linked to the interconnection unit. This interconnection unit may solely transfer messages from one component to another (communication) or may have some intelligence and decision capabilities (coordination). The set up of the system and the distribution of the component processes is done by the cell configuration component that is invoked only at the beginning of the cell process and is idle after the system has been investigated. Thus, our system establishes a flat hierarchy: the main process control is at the top level and the other components at the second level with little functional dependencies in order to support short response times.

In order to keep our system extensible and its components exchangeable, all software components need to have an identical structure. Each component provides a set of functions, which can be used by other components. Together with the interconnection unit, this enables a component to fulfill a given task by cooperating with the other components. For example, with the nanorobot design, such a model may generate a coherent behaviour comprised of distinct components, such as motion control, dynamic decision, and collision detection, among others. In order to make the robot's architecture faster, it has been subdivided into parallel subcomponents. Thus the neural network component is used to provide an optimization for the robot route trajectory (see chapter 5 and 8). The nanorobot has also a sensor-based component, which uses an *Interval Tree for 2D Intersection Tests* for collision detection (see chapter 3) with a hierarchical distance computation in the 3D workspace, based on the given virtual model interaction between the robot and the environment [174] [72] [66]. Moreover last but not least, the decision component integrates the robot evolutionary behaviour in a reactive fashion with the dynamic environment [75] [65].

Step 1: Process sensing
Robot 1: sense the 3D environment
Robot 2: sense the 3D environment
.
.
.
Robot n: sense the 3D environment
Step 2: Get response
Robot 1: calculate response
Robot 2: calculate response
.
.
.
Robot n: calculate response

Table 7.8: Processes of sensing and reacting in parallel with the environment.

7.4.2 Parallel Sensing for Virtual Robots

In order to enable a real time interaction between a set of intelligent agents in our virtual environment, there is the necessity for what we can identify as an *interaction phase*. In such a phase, there is an exchange of data between the nanorobots in the system. This means that each nanorobot requires access to the data of each of the other nanorobots. To analyses this problem further, the interaction phase can be broken down into two stages. In a physical simulation system these would typically involve the detection and resolution of collisions, but the stages could equally incorporate any other forms of sensory information and response, hence they shall be defined as the sense and response stages.

The sense stage is when a given agent is retrieving sufficient information about its environment to enable it to perform the response stage. A naive approach to this would be for every robot in the system to acquire the necessary information from every other robot action in the system. This would provide the desired result but would result in many cases of redundancy of data. It means that every action performed by any robot in the environment must be realized in a fashion which will permit the recognition by the other agents through the use of local perception. Thus each nanorobot is required to react with changes occurring in its surrounding environment. Such an approach would allow the nanorobots to perform the sense stage in parallel.

However, if the security of data is maintained, then the acquisition of data from other nanorobots will require that the other agents in the system actually provide the data to the sensing robot. This means that during each robot's active sense stage some processing time will be spent serving data to other objects. Hence the concurrent nature of the sense stage can rapidly become communication bound. Of course, any intelligent implementation of such a system would provide mechanisms to reduce this communication to a minimum. Firstly, only data need be processed for agents that have changed their state information since the last interaction stage. Secondly, most data operations need only be processed once for each agent (as in the case of the collision detection operation). But the inter-agent communications would still be significant in any system with a nontrivial collection of complex objects.

It is possible to circumnavigate this problem to some degree by providing shared memory storage for the data of all the agents. This would allow the sense stage to gather data without direct inter-agent communication and (allowing for resolution of memory contention) this could be carried out in parallel. The problem with this approach is that it destroys the security of data for the nanorobots. To solve this problem a security manager class was implemented, which is responsible for data consistency. Since nanorobots may typically change the objects' status that is under manipulation during an animation (e.g. the organ inlets' nutritional levels), the information about such objects will not be known in advance. Some data will be required to be obtained by each agent through its sensing systems upon such manipulated objects, what is implied in a shared exchanging data process. The sense stage will therefore involve indirect inter-agent communications, both during the gathering of information and the redistribution of results.

The response stage will include some isolated processing per agent that acts on the data gathered during the sense stage. For example for collision detection, this stage would involve nanorobots calculating their new velocities resulting from the data obtained during the collision detection stage. For the evolutionary decision process, it will require the sensing and gathering of information on the organ inlets' nutritional levels in order for a better strategy for action the next time-step in the dynamic environment.

The entire procedure ends with the scene collecting the information generated by the agents' updated operations, which signals that they are ready for the next frame generation sequence. The interaction phase can therefore be written as described in Table 7.8. These requirements are independent of the design paradigm used for the system and are valid for real time and frame-based systems that use discrete time increments for the simulation process.

7.5 Conclusion

Processing demands could increase extremely rapidly for the physical simulation of mobile robots when using computer animation. Specially for the automation of robotics systems, which are characterized as systems composed of several functional modules, the use of methods for decreasing the operation time is even more important. After identifying each functional module, the parallelization of such complex systems is an intensive field of research in computer science, even with an increasing level of processing power and memory capacity. For our problem the same approach was used based on such concepts, considering the complexity of the scenery under study. Hence breaking down the whole complex system into smaller functional parts, enables faster management and implementation, providing an architecture which is easier to test and to verify the robustness of each module. Afterwards such an approach seems to be a more suitable architecture for modeling robotics animation in computer graphics. To improve the simulator performance not only the systems were projected to run in parallel but also we have taken care to minimize as much as possible the intercommunication among the different modules, which implies an improvement in the performance of the system in question.

Chapter 8

Proposed Control Design

8.1 Introduction

This chapter summarizes distinct aspects of the main techniques required to achieve a successful nano-planning system design for a nanorobot model testing the reliability of an adaptive behaviour under diverse circumstances as a robust agent. It also illustrates the required architecture for a 3D visualization in real time. A new approach, using advanced graphics simulations for the problem of nano-assembly automation and its application in medicine with concepts derived from mobile robotics, is discussed. Therefore the problem under study concentrates its main focus on dynamic control for nanorobot optimal performance as a suitable way to achieve a large range of tasks and biomolecular manipulation in a dynamic environment. In our described workspace representing a simplification of the human body, the nanorobot performs a pre-established set of tasks building nutrient molecules, crudely analogous to the work done by a ribosome which is a natural assembler. Hence we discuss in this chapter the main aspects involved in successful nanorobotics control modelling, proposing the main concepts required for a new paradigm on the challenging development of molecular machine systems design.

8.2 Virtual Environment

A molecular machine systems could be described as a system capable of performing molecular manufacturing on an atomic scale [111]. With reference to nanorobotics control, it was demonstrated that computation is relatively cheap for macroscale robotic actuators while arm motion is relatively cheap for nanoscale robotic actuators [143]. Thus the moment-by-moment computer control of arm trajectories is the appropriate paradigm for macroscale robots, but not for nanoscale robots [142]. For nanoscale robots, the appropriate manipulator control is often trajectory trial and error, also known as sensor based motion control [214][69][67].

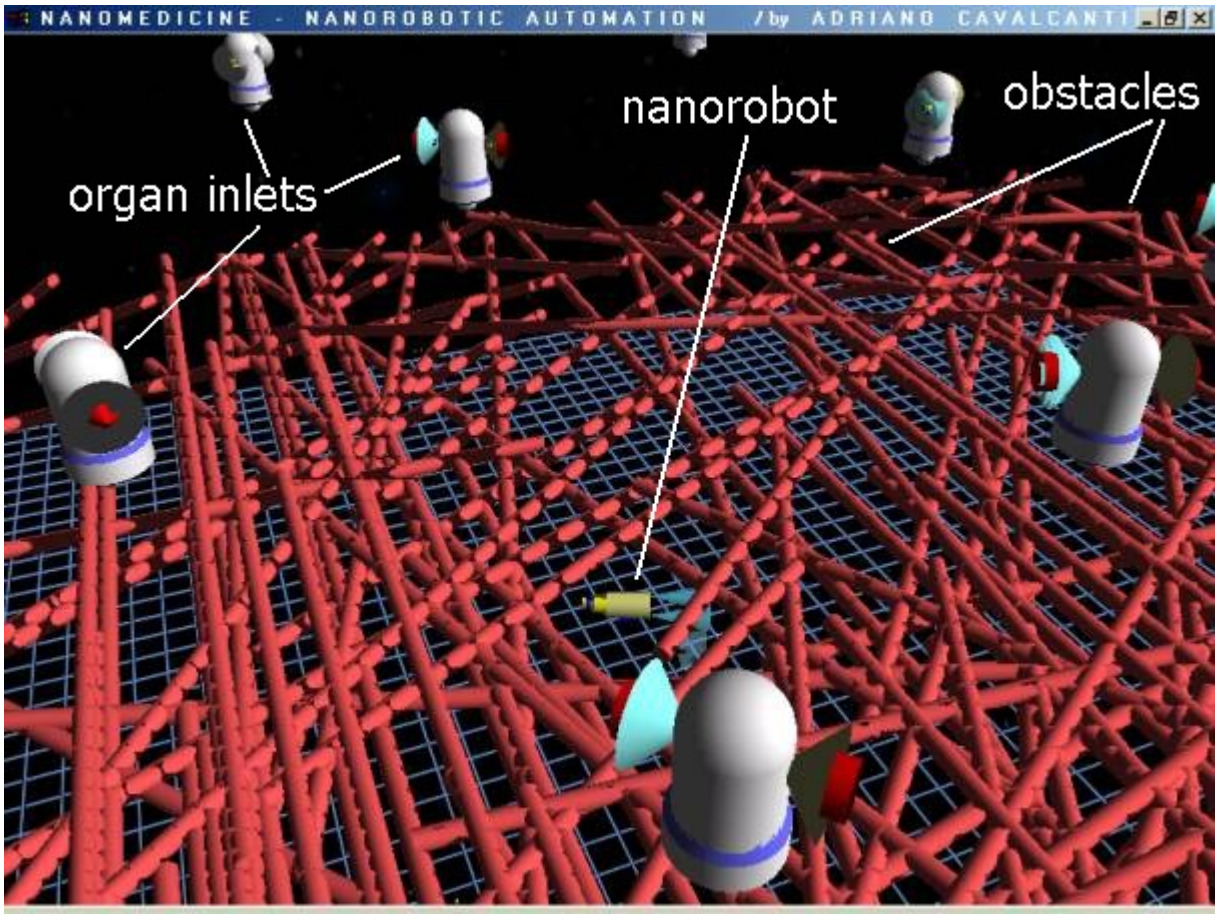

Figure 8.1: Top camera view in the virtual environment.

Actually there are three main design approaches in nanomanipulation for liquid and air environments, these are robotic arm, Stewart platform and a five-strut crank model [256]. In respect to the simulated environment, for our experiments we have chosen a nanomanipulation in liquid medium, which is most relevant within the presented application in nanomedicine [70][66].

There is a general agreement about the importance and necessity of the use of advanced graphical simulation that can accurately reflect the results of experiments in automated planning to permit judgments about manufacturing feasibility assisting chemical and biological assembly analyses in nanotechnology [94]. Nanoscale object manipulation systems have been successfully applied with the use of computer graphics for teleoperation, where the requirements for such systems have been clearly established [330]. Virtual Reality was used for our nanorobot design where the use of macro and microrobotics concepts is considered as a practical approach once the theoretical and practical aspects are focused on its domain of appliance. The virtual environment in our study is inhabited by nanorobots, biomolecules, obstacles, and organ inlets. Each nanorobot measures 650 nm in length and 160 nm in diameter. The biomolecule has a diameter of ~10 nm and each obstacle has a diameter of 120 nm. The organ inlets are 400 nm in height and width with inlet orifices 720 nm in diameter.

The nanorobot should be robust enough to operate in an environment with movements of six-degrees-of-freedom. For the input and user interface, the mouse and keyboard was adopted, and the camera view can also change its position in the y-axis related to the user's view height (Figure 8.1).

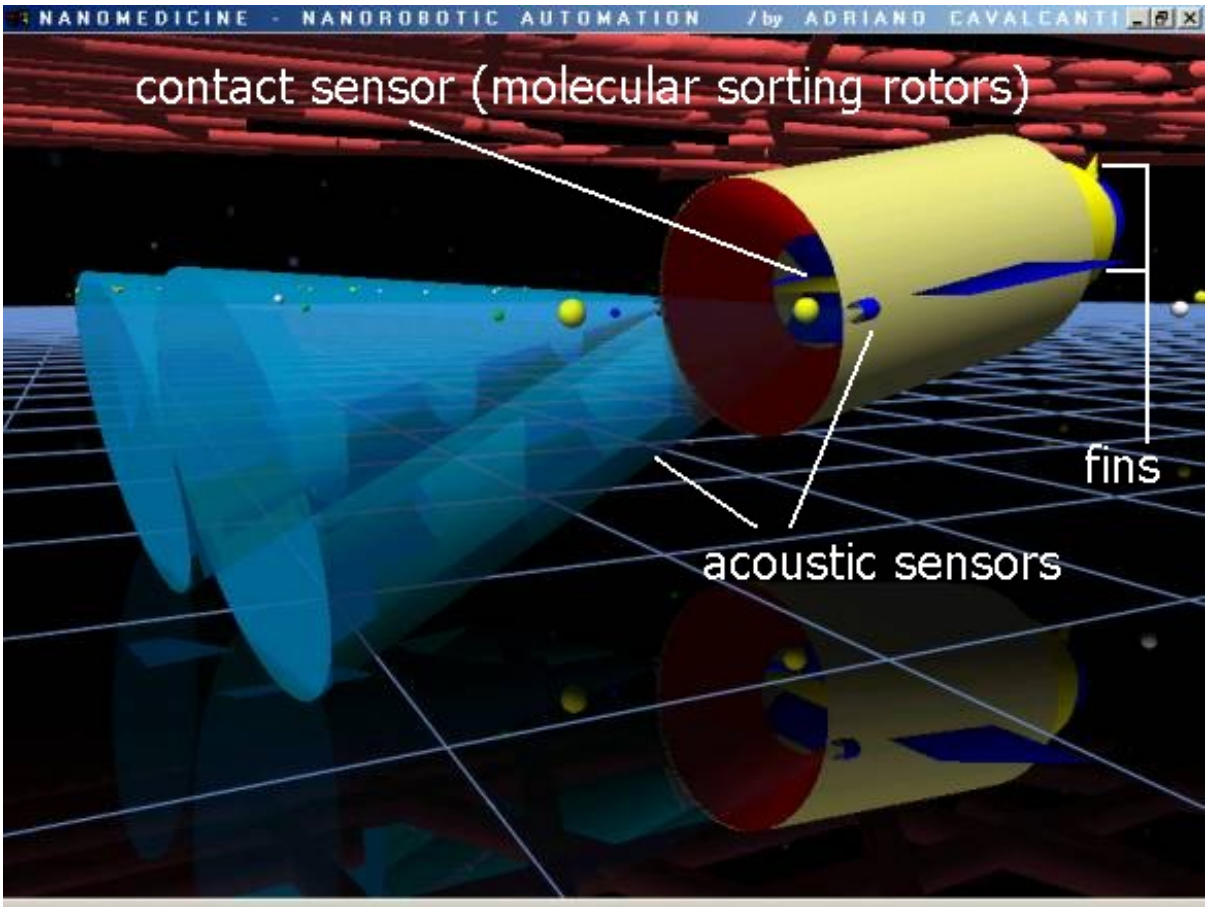

Figure 8.2: Molecular identification through collision contacts.

Therefore a suitable starting point for our hypotheses formulations and assembly system automation experiments was to consider the nanorobot design derived from biological models and comprised of some basic nanoscale components such as *molecular sorting rotors* (see Figure 8.2); to distinguish among different molecule types, the molecular sorting rotor presents a series of chemotactic sensors whose binding sites have different affinity for each kind of molecule [142].

The nanorobot exteriors considered in our design assume a diamond-based material [111] which may provide a smoother surface that minimizes fibrinogen (and other blood protein) adsorption and bioactivity, thus ensuring sufficient biocompatibility for the nanorobot to avoid immune system attack [142].

Some concepts provided from underwater robotics [55], [57], [370] were also assumed for nanorobot locomotion. Observing kinematics aspects, the nanorobot kinetic response can be predicted using state equations, positional constraints, inverse kinematics and dynamics, while some individual directional component performance can be simulated using control system models of transient and steady-state response [55]. For the kinetics aspects the nanorobot lives in a world of viscosity, where friction, adhesion, and viscous forces are paramount and gravitational forces are of little or no importance [142]. The main argument for using concepts based on underwater robotics as a good starting point for design, is the liquid environment in which the agents will be under operation performing the biomolecular assembly tasks [75]. In order to enable the nanorobot to function, it can provide its own energy demands, via the chemical combination of oxygen and glucose [142], both of which are plentiful in the human body.

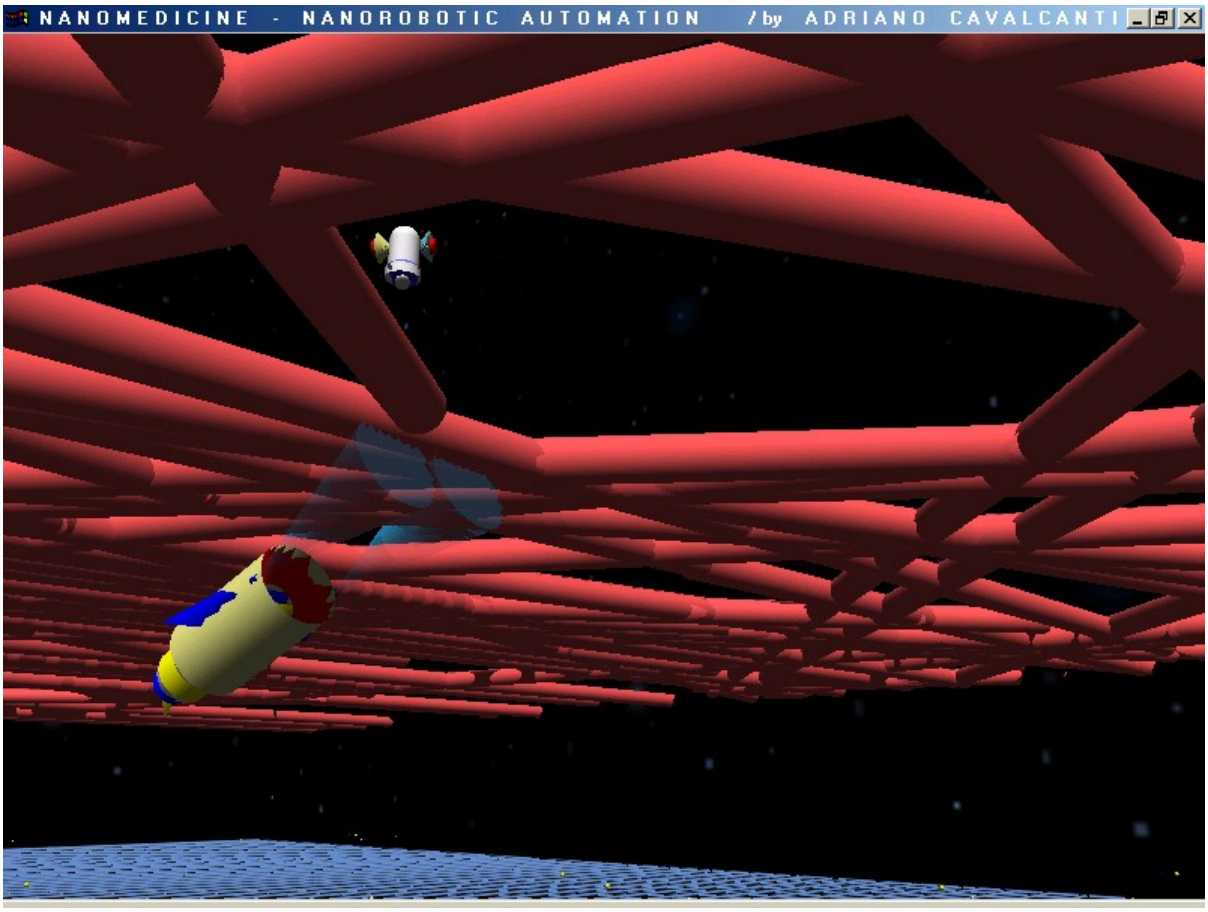

Figure 8.3: Robot obstacle avoidance - sensing obstacles.

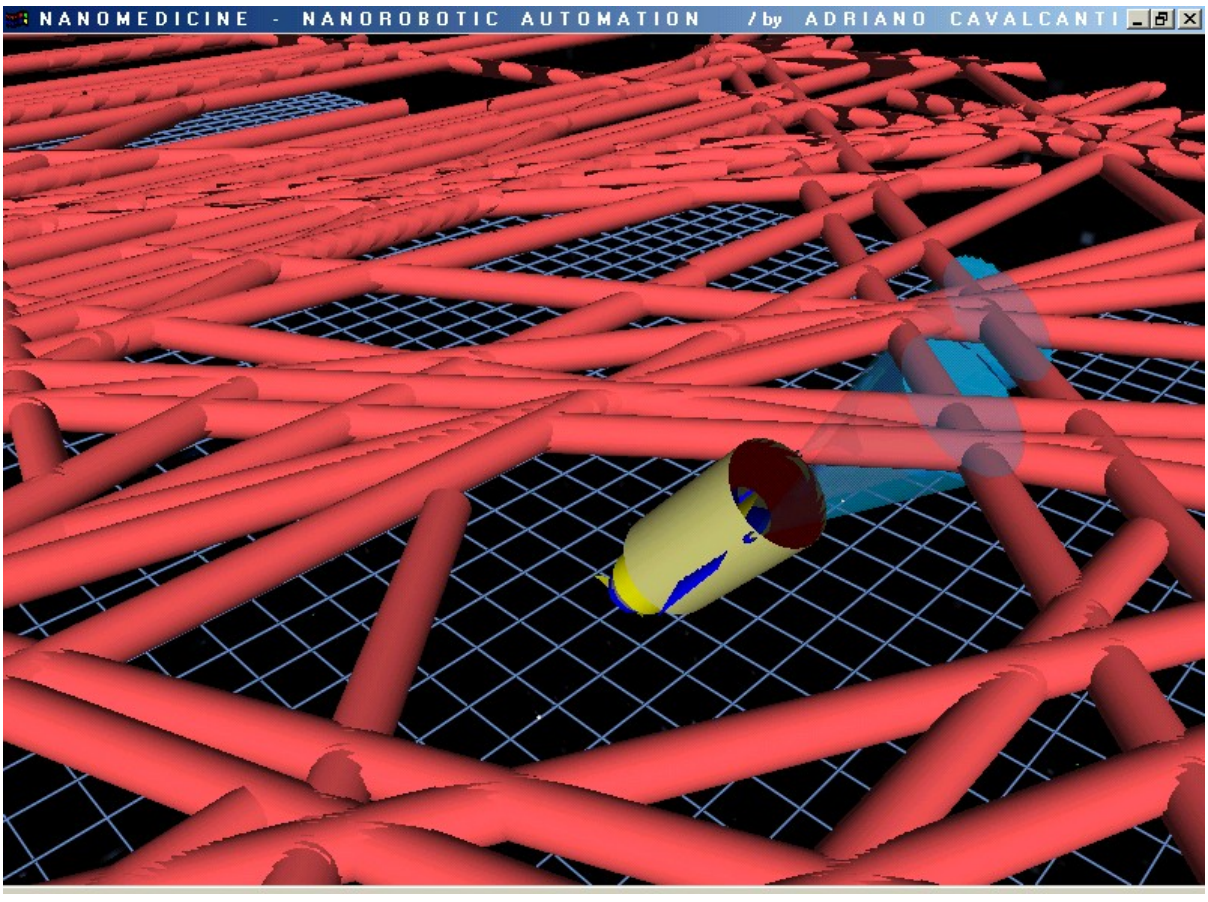

Figure 8.4: Robot obstacle avoidance - finding path.

A *nanoactuator* will be carried internally inside the nanorobot, and the nanorobot pushes the assembled molecule to the delivery point. The obstacles will be located in unknown probabilistic positions and the nanorobot has to avoid any collision with possible obstacles (Figures 8.3 and 8.4).

The nanorobot uses a macrotransponder navigational system that has been adapted for the main aspects of nanorobot positioning, which may keep high positional accuracy to each nanorobot's orientation [142]. Such a system might involve an externally generated signal from beacons placed at fixed positions outside the skin [257]. The uptake kinetics of a low molecular weight using a magnetic resonance contrast agent can predict the delivery of protein drugs to solid tumors. This can provide spacial information for inside body diagnosis of patient's chemotherapy developments. Hence, a similar approach can also be useful for nanorobot navigational purposes. Such approach uses Image Registration that belongs to the sub-field of computer science know as Computer Vision. It is directly related to pattern recognition in the sense of treatment of image. The methodologies normally adopted on Image Registration is to use sample techniques with data extraction through digital-analog data processing and approximation techniques. Thus the delivery positions that represent organ inlets requiring proteins to be injected are located for the nanorobot in a well-known position, whether these organ inlets are scheduled or not, for injection at time t. They will change their delivery orifice's colours making it open or closed. Thus assembled molecules are delivered to specific locations by a nanorobot's docking at 2 micron^2 (~1.4-micron square) embedded at appropriate spatial intervals across the organ inlets orifice, which will be open for the delivery and be closed automatically within the nanorobot's delivery act. The assembled molecule can be pumped by the molecular sorting rotors in ~10 seconds [139].

The trajectories and position of each molecule were generated randomly and each one will have also a probabilistic motion acceleration. The nanorobot navigation uses plane surfaces (three fins total) and bi-directional propellers, which are comprised of two simultaneously counter-rotating screw drives for the propulsion; propellers applicable as a propulsion system has been considered to nanorobots [142][148]; although propellers are adopted in the current work, other different approaches can be possible. Among other different types of propulsion system, an interesting possibility to be considered is cilia-based of flagella [367], which should be further investigated. The nanorobot has sensors which will inform it if a collision occurs and if it is an obstacle to generate a new trajectory plan, or if it could be a molecule which has to be captured and assembled (Figure 8.2).

The nanorobot will live in a world dominated by viscosity, as well as bacteria do. In this world a very low Reynolds number (Re) is assumed for the kinetic calculations [298], where the fluid mechanics in small structures can usually be described by the classical continuum equations [111]. The ratio of inertial to viscous forces is determined by Re which could be expressed in equation 8.1.

$$\mathrm{Re} = r\rho v / \eta \tag{8.1}$$

where η is the viscosity of the fluid, v is the velocity, ρ is the fluid density, and r is a characteristic dimension or fluid density. Re indicates whether the flow will be laminar or turbulent around an object of a given shape [79]. For nanoscale dimensions in fluids of ordinary viscosities and velocities, Re is low and the flow laminar [142]. Given a sudden stop, the nanorobot will "coast" to a halt in a time $t_{coast} = r\,\mathrm{Re}_{nano}$ and by equation 8.2:

$$t_{coast} = \frac{\rho L^2}{15\eta} = 0.1 \tag{8.2}$$

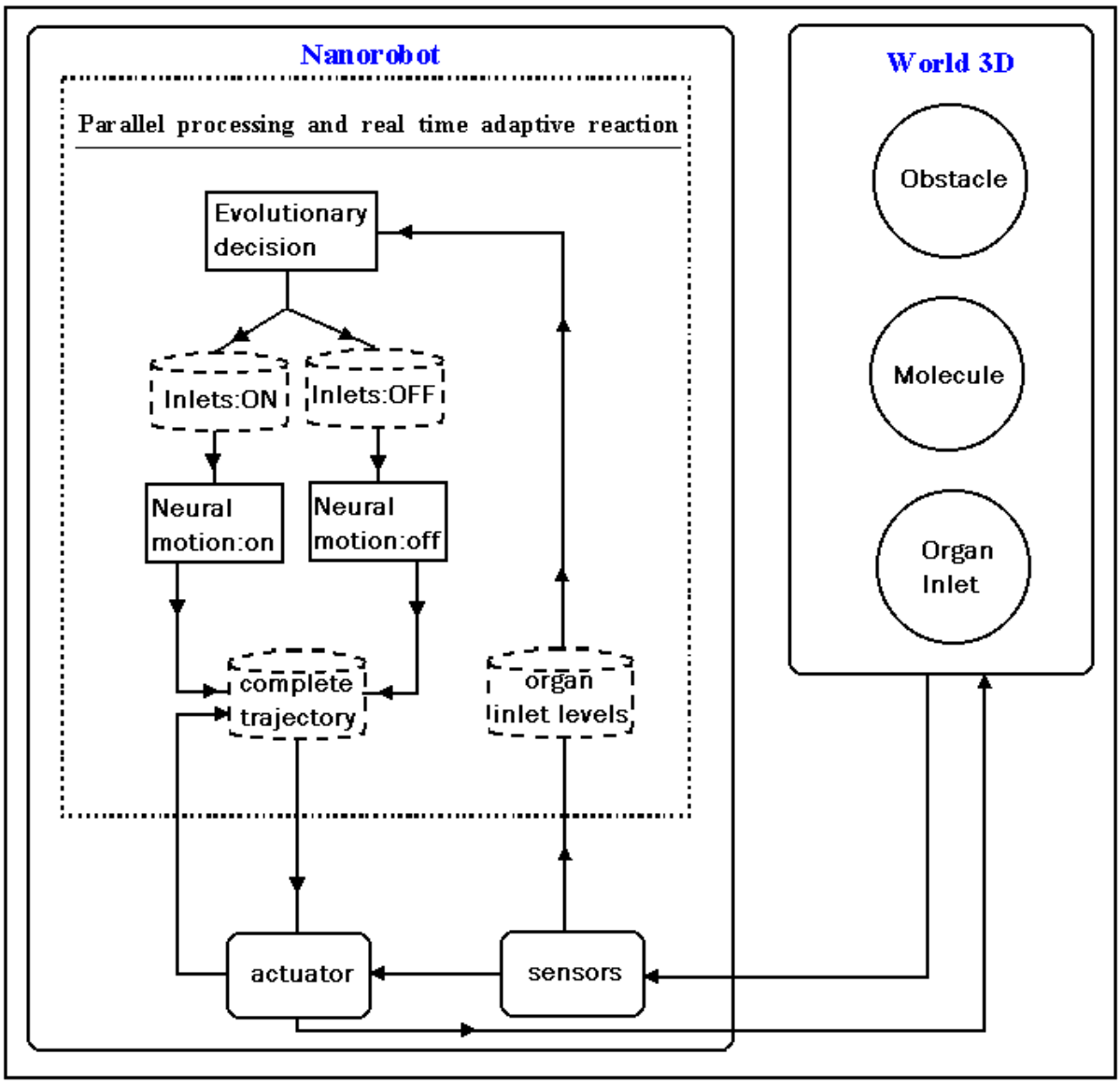

Figure 8.5: System architecture (nanorobot's functional parallel architecture).

where 0.1 is expressed in microsecond, and in distance $x_{coast} \cong v_{nano} t_{coast} = 1\,\text{nm}$ [39]. Thus with n as the rotational frequency, if the nanorobot is rotating at a frequency $n_{nano} = 100$ Hz when its rotational power source is suddenly turned off, n_{nano} decays exponentially to zero in a time $t_{coast} \cong 0.1$ microsecond and stops after turning, as expressed by equation 8.3:

$$q_{coast} = \frac{2pn_{nano} r \,\text{Re}_{nano}^{2}}{15\eta} \cong 40 \tag{8.3}$$

where q is the rotational motion, p is the pressure and 40 is expressed in microradians.

8.3 Evolutionary Decision

We intend to construct and validate a nano-planning system, where the use of robust evolutionary agents will enable a better-tuned validation of the nanorobot control system under study.

8.3.1 Robust Evolutionary Behaviour

Incorporating robust behaviour in a complex real-world system requires accurate and timely reactions to stochastic environmental events [236][293][84], thereby advances in artificial intelligence and real time systems have become important and successful tools dealing with such problems, therefore use of concepts derived from Evolutionary Techniques, Artificial Life and Ants have received special attention in the research community [75][66].

The evolutionary model used for nanorobot control decisions is cited in the literature as Genetic Algorithms (GA). A GA relies on concepts derived from evolution and genetics [76][65], thus providing behavioural learning from events and actions through time. In a GA every solution is seen as an individual with its own genetic characteristics and belonging to a certain population. In the implemented architecture (Figure 8.5) we used real time and parallel processing techniques [377], which were intended to provide a simulation scenery as close as possible to a real situation, where the agents react adaptively to any event and change in the environment with the model visualization in real time [78]. Each solution in the GA model is expressed as a chromosome regarding the agent decision on how, when and what organ inlets to serve in the dynamic scenery, and each decision required to be taken by the nanorobot is always attaining the programmed set of actions rigidly pre-established in our design by the fitness function, as is described through the equations 8.4 to 8.13.

$$Max \quad f(r_\Omega) = \sum_{t=1}^{n} \sum_{i=1}^{m} \psi w_i^t - \left|y^t\right| - \left|z_i^t\right| \tag{8.4}$$

$$\text{s.t.} \quad z_i^t = w_i^{t+1} - w_i^* \tag{8.5}$$

$$y^t = Q^t - d \tag{8.6}$$

$$Q^t = \sum x_i^t \le L \tag{8.7}$$

$$x_i^t = \mu_i^t x_i^{\max} \tag{8.8}$$

$$\mu_i^t \le \Delta_i^{\max} \tag{8.9}$$

$$w_i^{t+1} = w_i^t - \gamma\psi s_i^t + \psi x_i^t \tag{8.10}$$

$$w_i^{\min} \le w_i^t \le w_i^{\max} \tag{8.11}$$

$$\mu_i^t \in \{\{0,100\} \vee \{0,1\}\} \tag{8.12}$$

$$\Omega \in \{A, B\} \tag{8.13}$$

where

w_i^t :	nutritional state of the organ inlet *i* at time *t*.
y^t :	surplus/deficit to the desired assembled mean.
z:	keep the nutritional levels close to the target.
w_i^* :	defined desirable organ inlets' nutritional target level.
r, t, i:	subscript denoting respectively robot, time, and organ inlet.
max, min:	upper and lower bound parameter.
A, B:	define the kind of nanorobot.

n: size of time in the simulated scenery.
m: total of organ inlets to be fed.
L: robot load capacity.
x_i^t : substance amount injected in the organ inlet *i*.
Q^t : total assembled molecule by *r* in *t*.
s_i^t : substance consumed in the organ inlet *i*.
d : desired assembled substances rate.
γ : parameter to look ahead at nutritional levels.
μ_i^t : boolean variable.
ψ: determines specific model performance for *r*; ψ value is defined respectively in the chapter 9 by the equations 9.3 and 9.4 depending on each respective control approach.
Ω: determines if *r* is of kind *A* or *B*.
Δ: maximum to be injected at organ *i* in *t*.

Equation 8.4 represents our fitness function, where the nanorobot optimizes the protein levels for the selected organ inlets, what could mean a maximization or a minimization of the same variable depending on the parameter ψ, and the variable *y* induces the nanorobot to catch a number of molecules as closely as possible to the desired delivery mean, while z brings the nutritional levels as close as possible to w_i^*. Equation 8.5 informs the nanorobot how close its action is in bringing the organ inlets' levels to the desirable nutritional target. As it has been pointed out by the results in Chapter 9, there is a direct correlation between a greater number of nanorobots acting in the workspace and the desirable nutritional levels' target improvement. Equation 8.6 sets up the specified amount to be transported and assembled at time *t* for the nanorobot. Equation 8.7 is the total sum of captured molecules that will be assembled attending the nanorobot load capacity. Equation 8.8 is the amount specified for each organ inlet *i* with injection at time *t*. Equation 8.9 expresses the maximum that could be injected into the organ inlet *i* at time *t*. Equation 8.10 is the nutritional state for the organ inlet *i* due to the action performed by *r*. Equation 8.11 shows the minimum and maximum nutritional levels needed for the organ inlets. Equation 8.12 is the genetic random operating values. Equation 8.13 defines what kind of features the present nanorobot has in *r*, such characteristics are described in detail in the next chapter (see Chapter 9).

As we shall see, the action based on sensor local perception has generated an adaptive coherent nanorobot behaviour, which was observed by the proposed model simulation (see Chapter 9). The study of coherent multi-robot behaviour in a single global environment is a relatively new field of research [225], which has advanced most concepts related to the use of local perceptions for reactive agents.

8.3.2 Behaviour Activation

The nanorobot model uses a local perception technique, thus the first approach to sensor integration involves two orthogonal sensing strategies: spatially and modally orthogonal sensors. Spatially orthogonal refers to a geometric arrangement of sensors which carves the robot's perceptual field-of-view into discrete non-overlapping regions.

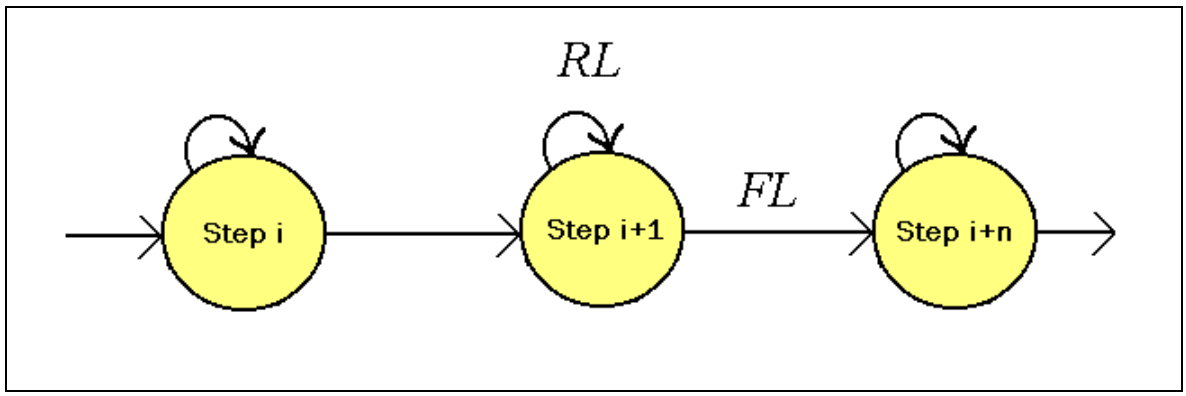

Figure 8.6: Perceptual cues forward flow.

Modally orthogonal is the integration of sensor data from dissimilar sensor types. By combining sensor data using these approaches, features from the environment's stimulus output are extracted and used by the robot's motor decision process.

The second method of integrating sensor data uses previously defined perceptual cues additively by concatenating binary decisions (cue outputs) into vectors that can be used to control state transitions in the robot's task model. Biomolecules-pushing is the task used to demonstrate the feasibility of the perceptual cue framework, by defining the cues used in the task independently from the actions performed by the nanorobots. In this manner, what the system of robots is required to do is defined by an evolutionary robot's task controller, but how the robots accomplish the task is not explicitly defined, it is instead dictated by sensor-based behaviour activation.

- **Controlling behaviour activation**

  Behaviour activation refers to the process of deciding which behaviour is to become active in the current context, i.e. in the currently executing controller. Each behaviour has an associated perceptual cue that activates the behaviour to produce a motion command as output. More than one behaviour may become active during the control loop. A priority scheme among the behaviours within the current executing controller determines which action is executed by the robot. Thus, in a known environment a robot's action is based on a perceptual process that uses local sensing to look for specific features in sensor data.

- **Controlling task state transitions**

  Perceptual cues used on control state transitions in task execution are specified as predicates with perceptual preconditions that must be satisfied. Each task is decomposed into subtasks and a controller is designed for each subtask. Control system processing is handled in discrete steps, with control either remaining within the current subtask controller or passing onto the next one, as specified in the task model digraph using a forward (FL) or repeat (RL) edge. The cue used or the transition in each subtask controller, or step *i*, is related to its predecessor by:

$$FL_i = FL_{i-1} \wedge c_i \tag{8.14}$$

$$
\begin{aligned}
ST_1 &\leftarrow \neg c_1 \\
ST_2 &\leftarrow c_1 \wedge \neg c_2 \\
ST_3 &\leftarrow c_1 \wedge c_2 \wedge \neg c_3 \\
&\vdots \qquad \vdots \\
ST_{n-1} &\leftarrow c_1 \wedge c_2 \wedge c_3 \wedge \cdots \wedge \neg c_{n-1} \\
ST_n &\leftarrow c_1 \wedge c_2 \wedge c_3 \wedge \cdots \wedge \neg c_n
\end{aligned}
$$

Table 8.1: Logical *AND* perceptual cues.

for I= 1,2,…,n where n is the number of subtasks and c_i is a new perceptual cue for step i.

$$FL_i = FL_{i-1} \wedge \neg c_i \tag{8.15}$$

specified in this manner the forward edge, illustrated in Figure 8.6, is the cue signalling step transition and signifies that a locally detectable event has occurred indicating step completion. Each step in the task is modelled as a state in a finite state machine with perceptual cues used for state transition. The perceptual cue causing a forward transition (FL) is simply a concatenation of another boolean variable to the previous step's forward perceptual cue. The repeat edge indicates that the current action is to be repeated since the specified change in stimulus (the detectable event) has not occurred.

When a task is modelled as a multistep procedure, with each step represented as a state in a task digraph, then the current state (ST) is specified as a logical *AND* of the perceptual cues, or i = 1,2,…,n where n is the number of subtasks, as detailed in the Table 8.1. In choosing a minimal set of sensors for the transport task, the robot's activities of avoiding obstacles, locating the molecules to be moved, and transporting it to a goal location, are considered. Each behaviour that could be taken must be enumerated, with inputs to the behaviours specified as binary input variables. This establishes the minimal number of binary variables that each behaviour requires for a determined action. For example, the possible actions of the *AVOID* behaviour are *idle, left-turn, right-turn* therefore requiring two binary input variables allowing for a maximum of four actions. In a similar manner, the molecule locating behaviours use two input variables and the pushing behaviour uses one.

The nanorobot includes external sensors to inform it of collisions and to identify when it has encountered an obstacle which will require new trajectory planning. Aspects of the non-structured opaque surrounding workspace, like the interior of the human body where the nanorobot is acting, must be considered in the navigational sensing design. In robotics fields there are often many kinds of sensors such as infrared, computer vision, acoustic, chemical sensors, and so forth which are normally used for robotics navigational purposes. Optical sensors have been widely applied in terrestrial mobile robotics but these have an extremely limited range in a liquid environment. Types of sensors such as laser rangefinders [55] could be also used for underwater robotics but not for nanorobotic sensing because, for instance, the laser energy might excite or chemically alter the surrounding biomolecules that the nanorobot is trying to capture. Optical sensors may also be unfavorable for nanorobot design because lighting requires excessive power and has limited range, and because vision based systems are unreliable in opaque environments which may restrict their use except at short distances.

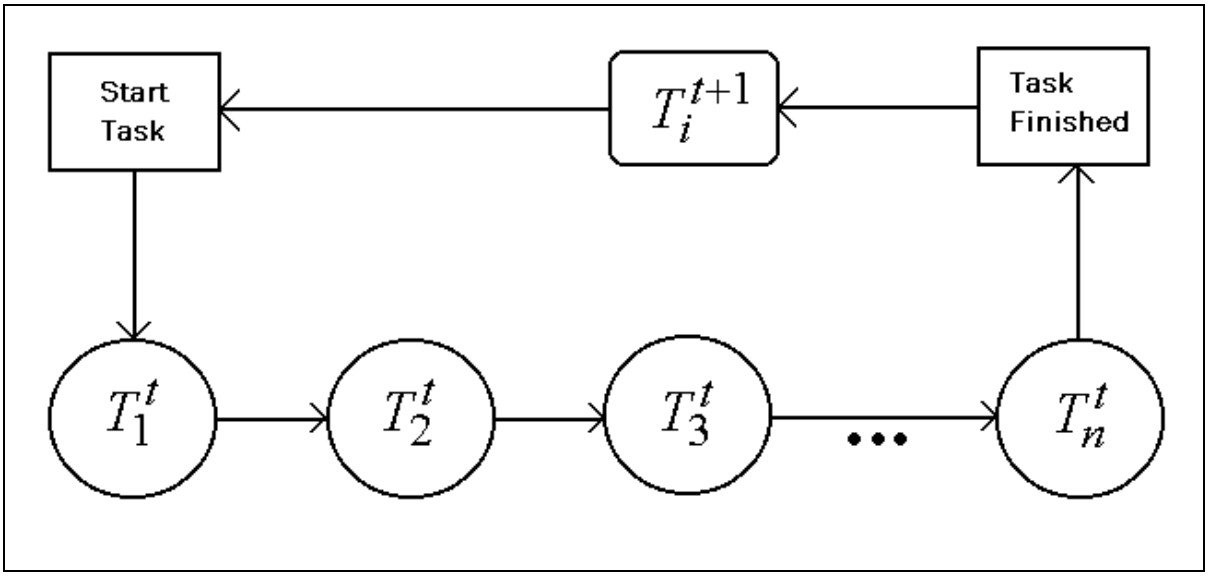

Figure 8.7: Tasks are described as a sequence of steps, with each step possibly composed of additional subtasks (T_i).

Although the infra-red sensor seems preferable for macroscale terrestrial robots, for underwater robots the most common sensor approach involves the use of sonar systems. Similarly the most addressable approach for nanorobots in nanomedicine is to use acoustic waves [142]. The blue cones shown in Figure 8.2 represent regions that the robot's sonar can "hear". Scientific visualization techniques permit rapid and precise geometric analysis for a sonar classification system [55]. The present approach provides a medical nanorobotics control model in accordance with engineering, physics concepts, and current trends in nanotechnology. For communication, as well as for navigational purposes, the use of nanoacoustics for nanorobot interactions can effectively achieve resolutions of 700 nm [284].

8.4 Task Description and Decomposition

Task description and decomposition can be divided into task-related and tool-related knowledge [253]. In other words, what is to be done and how to do it. Tasks-related knowledge can be described in terms of externally observable desired changes in the environment, independent of the procedural mechanism used to accomplish them. This is synonymous with Wilson's sensory state machines in which the environment is considered as a machine with the effects of robot actions considered as input and changes in observable stimulus as output [375].

Our model assumes that the task under consideration can be described as a sequence of steps. A finite state machine (FSM) will then be designed to accomplish each step with transitions between steps triggered by perceptual cues. Each step may, of course, be composed of substeps or subtasks also to be performed sequentially. In this manner a task may be described in fine detail as required by its decompositional analysis. This results in a task description having the hierarchical structure illustrated in Figure 8.7.

In the presented model, the task description is specified in a directed graph, called a task description graph (TDG), with vertices representing a stimulus-object and its position to be manipulated by the system, and edges in the graph representing possible actions that effect those manipulations.

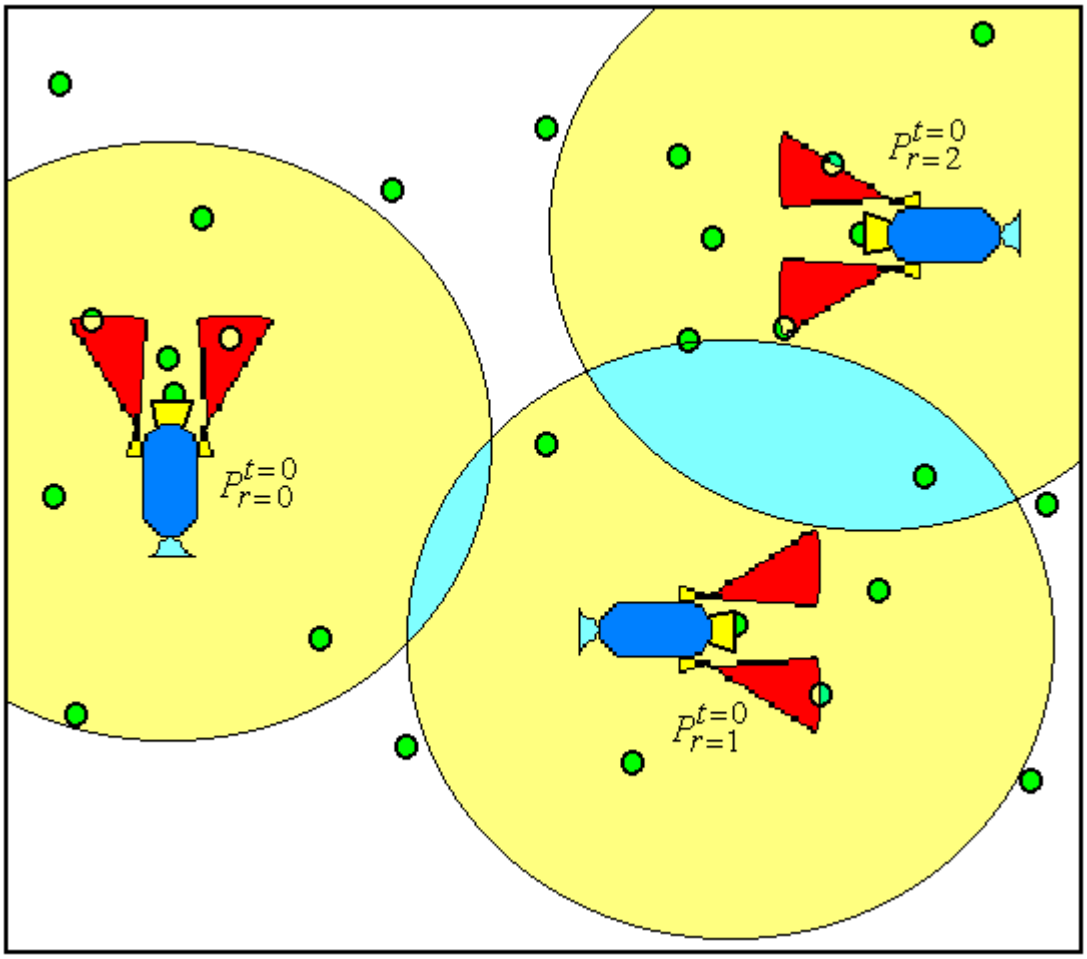

Figure 8.8: Illustrated is the nondirected molecule-transport where the nanorobots catch the molecule from different directions.

Tool-related knowledge is specific to the mechanism employed by the system and refers to robot actions in the environment and therefore is procedural in nature. A task is decomposed into finite state controllers that accomplish the desired changes specified in the TDG. Both the execution of individual subtask controllers and the transition between them are accomplished with perceptual cues. Perceptual cues and their finite state machine controllers are called *Q*-machines and together with a task description graph provide a model that considers the environment and robots together in its solution to the specified task.

In the class of manipulation tasks being modelled here, objects to be manipulated are described as stimulus-objects and states are determined by position, time and performance metric. Since states are vectors, there are an infinite number of states in the environment. However, in the molecule-capturing task the states of interest are: initial, final, intermediate and stagnating. Thus the states correspond to several actual positions of the object being manipulated in an *X*, *Y*, *Z* coordinate system. A task to be accomplished by the system is described by defining the initial and goal positions of the object being manipulated. As well, stagnating conditions are identified as positions in the graph requiring special actions, i.e. stagnation recovery behaviours. In the nanoassembly case that we discuss in items 8.4.1. and 8.4.2, two actions are used to manipulate the molecule: A_1 *capture_molecule* and A_2 *remake_motion*.

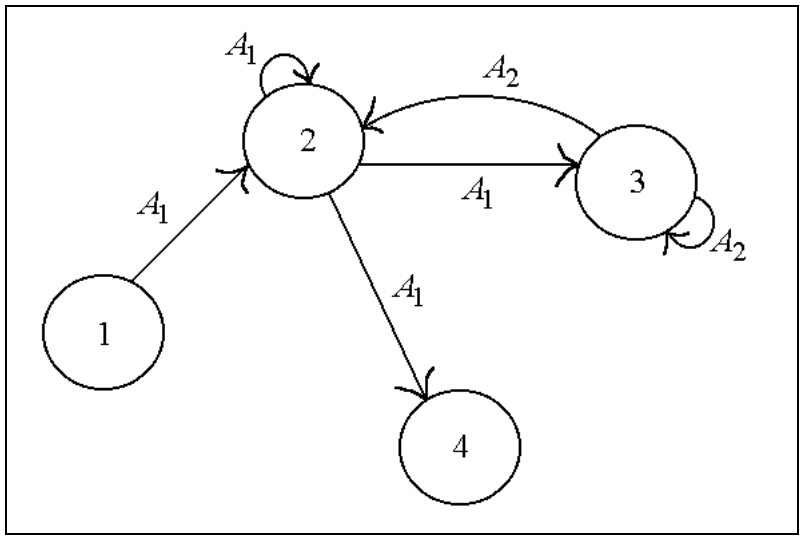

Figure 8.9: The task description graph where vertices represent an object (molecule) and position, and edges represent actions that effect changes in an object's position.

Task description is an external global point-of-view and describes changes in the environment without regard to the mechanism that causes those same changes. The description of the task is captured in a directed graph defined as follows.

Task description graph (TDG): is a directed graph G with *n* vertices and *m* edges. The vertex set $V(G) = \{v_1, \cdots, v_n\}$ describes the state uniquely determined by position, time and performance metric, of an object (S) perceived as a stimulus to be manipulated, and the edge set $A(G)_i = \{a_1, \cdots, a_m\}$ describes the actions needed to manipulate the object *i* without speaking about the actor or actors. *V(G)* contains an initial state and a goal state, each of which can be associated with a set of positions, times and performance metrics according to the precision to which the values are known.

8.4.1 Nondirected Molecule-Capturing

Nondirected molecule-capturing involves pushing a molecule from an initial position for a fixed distance in any direction. The task is considered successful if the molecule *S* is pushed to a fixed distance *R* in a given amount of time *T*. Distance *R* is the radius of a circle with the center at an initial position P_{t_0} as illustrated in Figure 8.8. The goal position *P* is any position that satisfies $R \geq \left| P - P_{t_0} \right|$ which is simply the distance between the goal and initial molecule positions.

Each vertex in the TDG show in the Figure 8.9 specifies an unique condition as defined by changes in radius from the initial position $\Delta r = \left| P - P_{t-\delta} \right|$ per time period δ summarized as:

Initial molecule position, which may take any value in the 3D position space with $\Delta r = 0$

$$v_1 : (S_1, P_t) \,|\, \Delta r = 0, t = t_0 . \tag{8.16}$$

Intermediate molecule positions, which may take any value in the 3D position space with $\Delta r > 0$

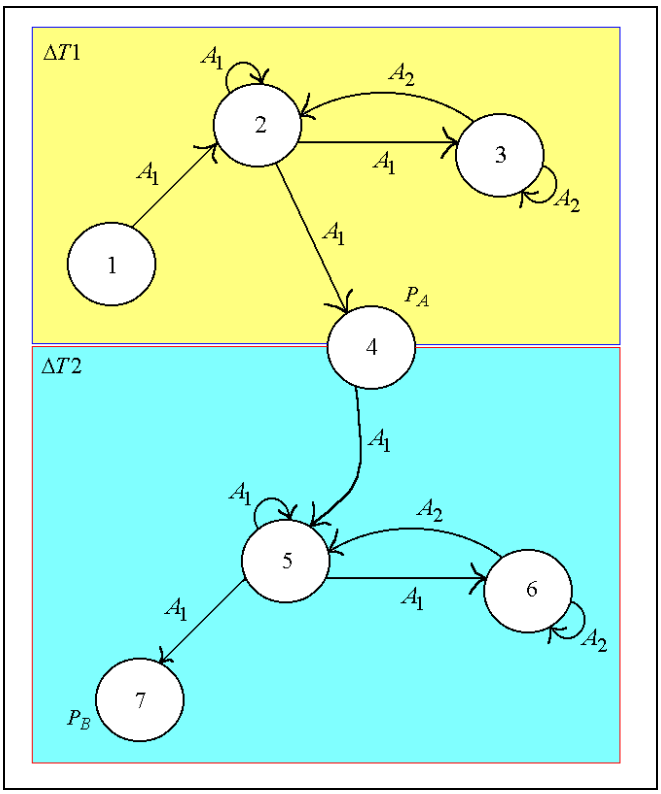

Figure 8.10: The task description graph for directed molecule-pushing.

$$v_2 : (S_1, P_t) \mid \Delta r > 0, t = t_0 + \delta \ . \tag{8.17}$$

Goal molecule positions, which may be specified as any value in the 3D position space where $\Delta r \geq R$ and $t_n - t_0 \leq T$

$$v_3 : (S_1, P_t) \mid \Delta r \geq R, t = t_n \ . \tag{8.18}$$

Stagnating molecule position, which describes a position that has not changed in time period δ resulting in $\Delta r = 0$

$$v_4 : (S_1, P_t) \mid \Delta r = 0, t = t_0 + k\delta \ . \tag{8.19}$$

where $k\delta$ is the time period before stagnation is detected, i.e. a timeout.

The stagnation condition in equation 8.19 occurs when robots capturing the molecules in opposing directions detect an imminent collision with another robot moving in a contrary direction, thereby producing an undesirable event, that is, the collision between robots. The problem occurs due to the nondirected nature of the task. The solution is a recovery behaviour whose output is a robot action that changes the orientation of the navigation force and is labelled as A_2 in Figure 8.9.

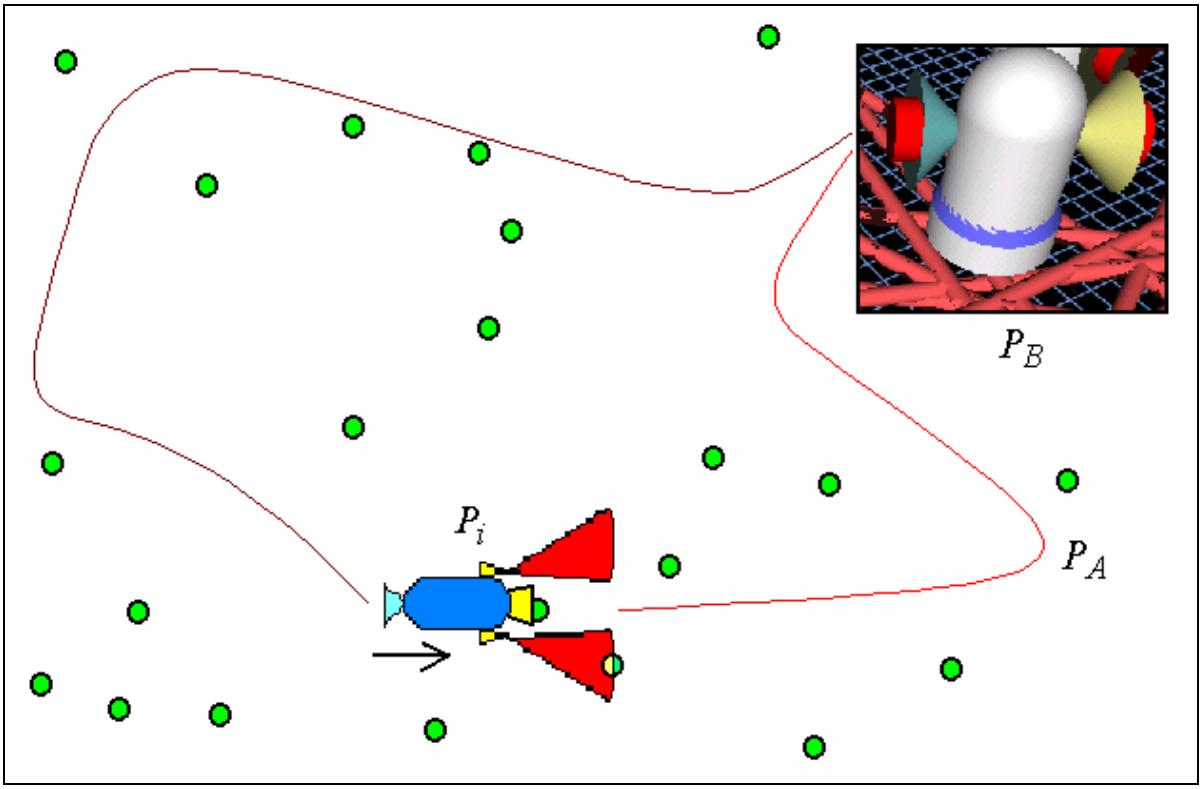

Figure 8.11: Illustrated is the directed molecule-pushing task where the robot pushes the molecule first to position P_A and then to position P_B.

8.4.2 Directed Molecule-Capturing

The task can be changed to directed molecule-capturing towards specific positions and can include a temporal sequence of positions as indicated in Figure 8.10. The molecule is pushed from an unknown initial position labelled as v_1 to the position described by vertex v_4 in time period $\Delta T1$ and then to position described by vertex v_7 in the time period $\Delta T2$. Positions $v_{2,5}$ describe intermediate positions during execution, while position v_3 and v_6 refer to stagnating positions from which recovery actions are required.

In this example, the molecule is first moved from an initial unknown position P_i to a known position P_A and then moved to a second position P_B (Figure 8.11).

8.5 Environment Sensing

Using a sensor-based model our nanorobot can explore the unknown environment dynamically, and incrementally build its own internal model of the world. Due to the sensor-based local perception, the nanorobot must on each time-step verify the organ inlets that belongs to its attribution, to make feasible the next step in *decision planning* (Figure 8.12). Thus each nanorobot visits in a shorter time the organ inlets that were pre-attributed to that nanorobot in order to gather information for the next time-step decision from the 3D workspace.

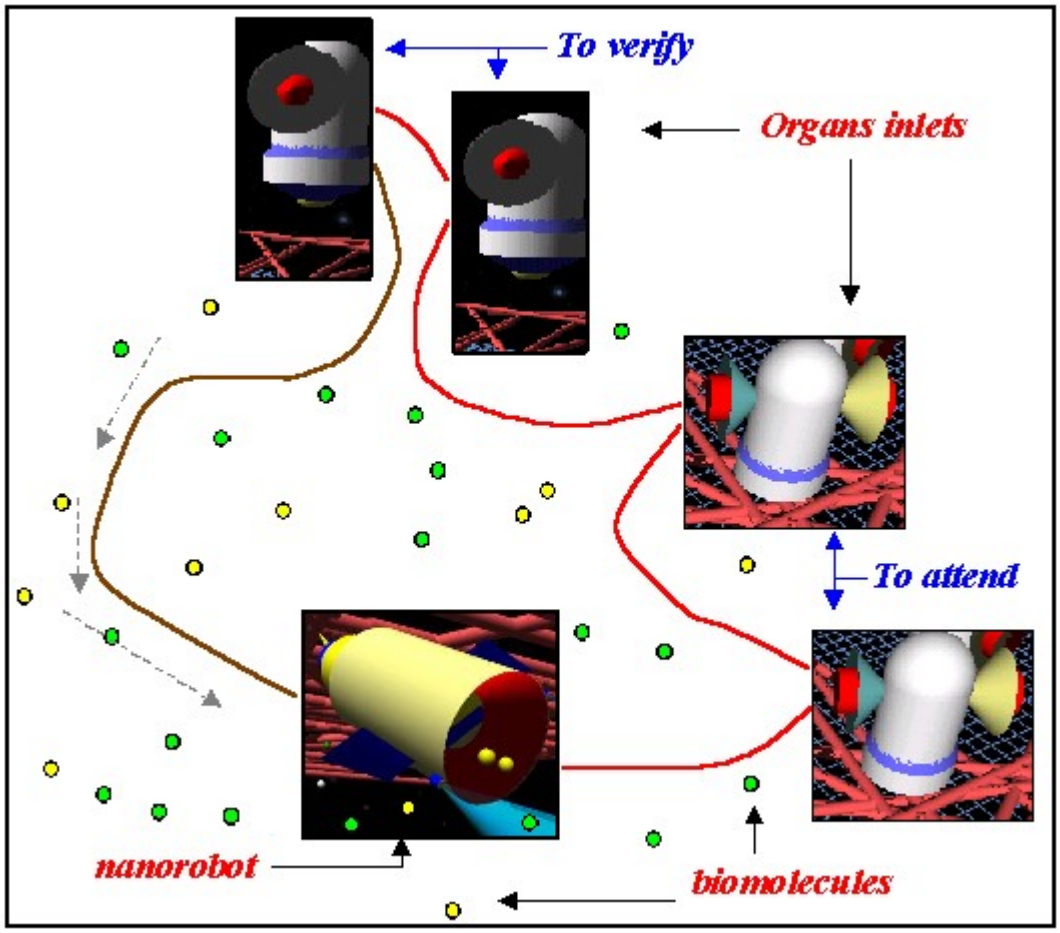

Figure 8.12: Directed molecule capture-delivery, and the environment sensing with complete tour.

The nanorobot firstly visits the organ inlet selected to be supplied, attending to each demand, afterwards the nanorobot visits any other organ inlet still not visited during the time-step simulation in order to evaluate their nutritional levels. Thereby the referred nanorobot has attained enough knowledge necessary to decide the best action plan for the next step. The organ inlet state level is read through contact sensors, with the sensor-based local perception.

8.5.1 Memory Behavior

Memory behavior begins with the sonar recognition of 3D objects using a set of previously recorded memory sonar object data identifications. These are based on geometric collision detection for the process of evaluating different sensing actions that the nanorobot can take, and choosing the kind of low-level action that maximizes the information acquired by an effective agent performance. Three dimensional data could be obtained by a high resolution acoustic camera, such as the Echoscope [169], with the recognition process performed through CAD-based vision techniques [133]. The acoustic camera is formed by a two-dimensional array of transducers sensitive to signals backscattered from the scene previously insonified by a high-frequency acoustic pulse [276]. A related system has been proposed for transcellular acoustic microscopy for nanomedicine [142].

For each object class, and for each level of detail, an aspect graph is built off-line. Each node of the aspect graph corresponds to a characteristic view of the object at the given level of detail,

including views that are close to one another. This allows the nanorobot's recognition system to act quickly and effectively with low computational effort to show the most adequate low level action based on the memory behavior approach. Thus, when the acoustic or chemical sensors (molecular sorting rotors) [142] touch an object in the scenery, the nanorobot will identify what kind of object the signal is related to. Thereafter the memory activation permits recognition of distinct objects, allowing the nanorobot to perform suitable activities upon encountering a molecule, an obstacle, an organ inlet, or another colliding robot.

As a feasible approach to operate the graphic simulator as fast as possible and to keep the nanorobotics animation efficient, we have done set strings representing the kind of objects which comprise the workspace where the nanorobots are interacting. Thus, let K denote the set of all kind of 3D objects in the environment. Each nanorobot maintains a set M of observations checked incrementally from the output of the sensor-based module. Thus the nanorobot rule-based behavior is influenced by equation 8.20:

$$M \leftarrow K \tag{8.20}$$

$$K \subset o_i^t = \{p, q, v, s, u\} \tag{8.21}$$

$$M \begin{cases} K = o_1 & \Rightarrow & r_\Omega = \omega_1, \\ K = o_2 & \Rightarrow & r_\Omega = \omega_2, \\ \vdots & \vdots & \vdots \\ K = o_n & \Rightarrow & r_\Omega = \omega_n. \end{cases} \tag{8.22}$$

$$\omega_i = A(G)_i \tag{8.23}$$

where

i, t: subscript denoting: object identification and time.

r_Ω: nanorobot.

o: each possible 3D object to be sensed by the r_Ω.

ω: kind of activated behavior that r_Ω has to perform.

$A(G)_i$: actions needed to manipulate the object i.

K: set with the kind of object comprising the 3D nano-world.

M: memory behavior.

p: actual position from i.

q: properties of i.

v: velocity of i.

s: translation of i.

u: rotation of i.

Component t represents the variable time in our model, while i is the related object that the nanorobot is sensing. The attribute q contains specific properties and qualities of the object, including information about the typical reaction features from the object when it is manipulated, such as whether the object is a deformable or rigid body, and other kinds of information that allows the nanorobot to take the correct action required. For example, if the observed object is a molecule, then q might identify the object as something that can be captured, and so forth. Object properties are flexible and can be utilized by reasoning engines to enable the nanorobot to make correct decisions when interacting with the surrounding workspace.

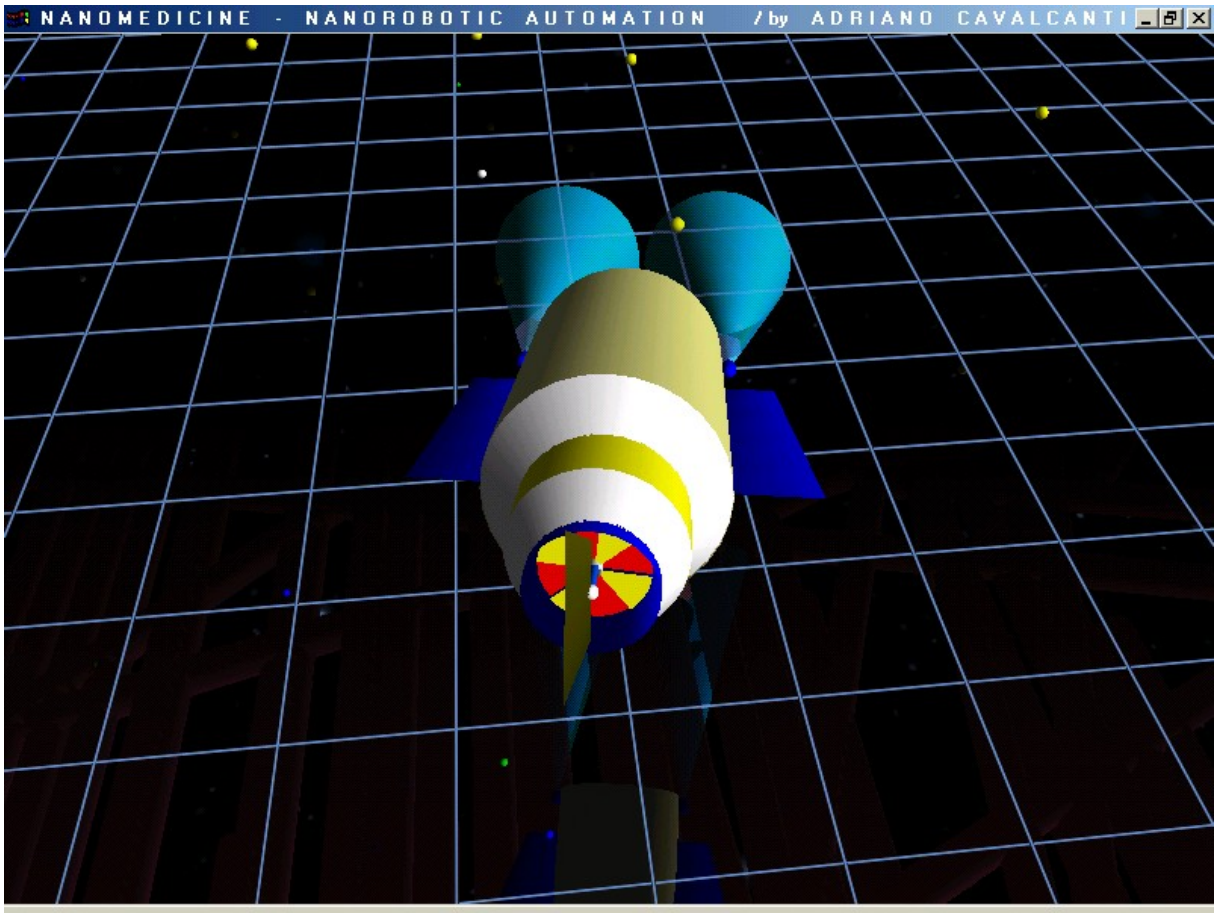

Figure 8.13: Nanorobot's sensor - back view.

The translation *s* is the observed position and orientation of the object represented by o_i and the same idea is applied for *u*. The linear and angular velocities observed from o_i are represented by the variable *v*. *M* represents the nanorobot sensing memory related to the object o_i under consideration.

The nanorobot's perception sensing is continuously evaluated by the sensor-based module engine, which detects collisions on each sensor and relates them to the referred 3D object in *M* (see Figure 8.2 and 8.13). Thus the object identification from any currently contacted object is returned from the set *K*. As expressed in equation 8.21, each o_i in *K* is combined with its corresponding object's state description. Thus the characteristic from o_i is compared and recognized based on the values contained in *K*. For example, after *M* has been supported by *K*, then the navigation path-planning module is invoked using the information returned by *M* in order to know whether the sensed object leads to an obstacle, a molecule, or an organ inlet, and what subsequent action to take. Thus, each nanorobot plans determined action interactively in the nano-world based on its own sensor-based local perception and its memory behavior. The data flow of the algorithm is shown in Figure 8.14.

The interaction of the nanorobot with its environment supported by the sensing system and memory behavior is a suitable approach mainly because the nanorobot's workspace is a stochastic environment populated by objects which could appear, disappear, or move around in an unforeseeable fashion. There are many proposed architectures for implementing rule-based models in the literature, the suitability of each one depending on the kind of application [201], [226], [315].

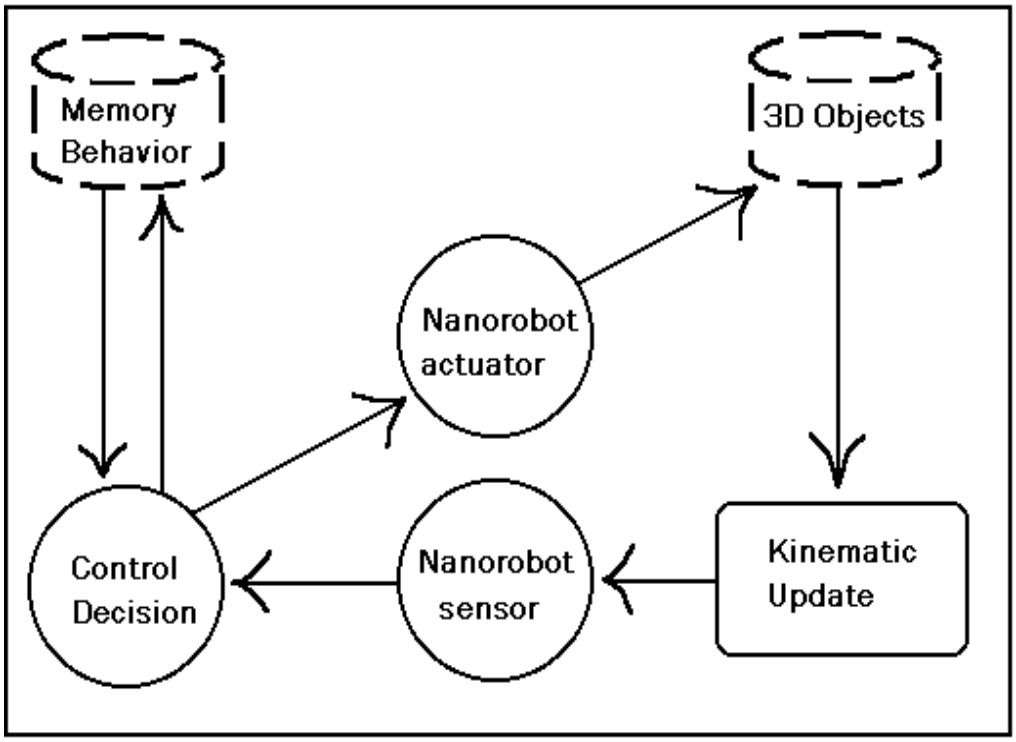

Figure 8.14: The basic sense-plan-control loop for the stochastic environment.

In the artificial intelligence literature, there has been a growing movement towards knowledge representation languages that support an explicit representation of uncertainty [220]. The emphasis has been on probabilistic representations, which allow the agent's reasoning process to utilize such techniques as conditioning for incorporating new information, and expected utility maximization for making decisions [202][88]. Concepts such as Bayesian belief networks that provide a compact representation of complex probability distributions could be used to allow the agent to make inferences based on observations of the environment. In our approach the nanorobot uses an integrated multi-modular functional architecture, in which the information collected by the nanorobot in the time-step simulation t-1 serves as the basis for the evolutionary decision planning and neural motion control at the time step t.

8.6 Neural Motion Control

A connectionist model using an artificial neural network (ANN) was chosen for the solution of motion control and shortest-path problem, where we are going to lead with a dynamic combinatorial problem for each time-step simulation. The classical problem of finding an optimal three-dimensional shortest path avoiding polygonal obstacles was demonstrated as typical NP-hard [14]. The use of a non-deterministic approach to solve the motion control seems to be the appropriate technique in such cases, in the sense that among other heuristic methods the use of ANNs were successfully used for motion and animation of physically-based models in virtual environments [157]. Suppose that a coordination problem has been posed in which the state space, *X*, is defined, along with initial goal states, x_{init} and x_{goal}.

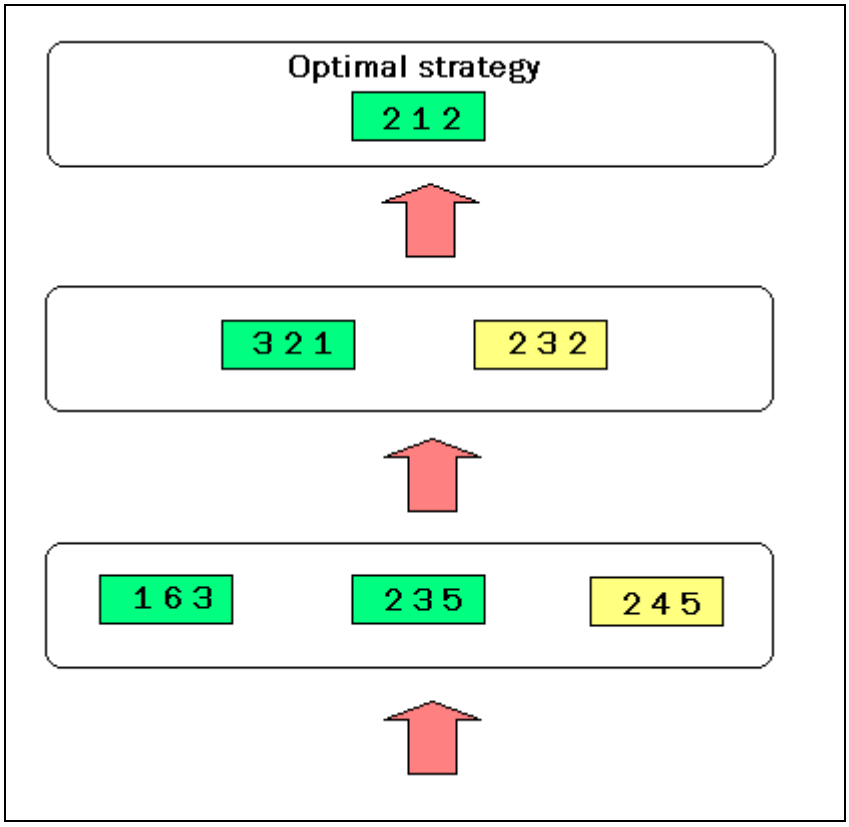

Figure 8.15: Obtaining strategies that are minimal related to the ordering that exists on $\Gamma / \sim$.

The goal of each robot is to choose some control function u^i that achieves the goal x^i_{goal}. We will use the notation γ^i to refer to a robot evolutionary strategy, which represents a possible choice of a control function that incorporates state feedback, represented as $u^i(t) = \gamma^i(x,t)$. In terms of control laws, this is equivalent to a closed-loop controller. In principle, extensions that incorporate incomplete or imperfect information feedback can be made [30] [234] [233]. We refer to $\gamma = \{\gamma^1, \gamma^2, ..., \gamma^N\}$ as a strategy. Let Γ denote the set of all allowable strategies.

A stationary strategy is a special form of strategy that depends only on state, and not on a particular time. For the motion planning problems that we are considering, although in relation to the time step simulation represented by $\gamma^t = \{\gamma^1, \gamma^2, ..., \gamma^N\}$ we have dynamically time-varying strategy solutions, non-stationary strategy, the solutions are naturally stationary related to each specific time step *t* solution, which is dynamically considered and defined at the moment *t*-1.

For a given x_{init} and strategy γ, the entire trajectory, *x(t)*, can be determined. If we assume that x_{init} and x_{goal} are given, then $L^i(\gamma)$ can be written, instead of using the form $L^i(x_{init}, x_{goal}, u^1, ..., u^N)$. Unless otherwise stated, we assume that $L^i(\gamma)$ refers to the loss associated with implementing γ, to bring the robot from some fixed x_{init} to x_{goal}. Hence, we can consider the loss functional as a function on Γ.

In general, there will be many strategies in Γ that produce equivalent losses. Therefore, we define an equivalence relation, $\sim_\Gamma$, on all pairs of strategies in Γ. We say that $\gamma \sim_\Gamma \gamma'$ if and only if $L^i(\gamma) = L^i(\gamma') \; \forall \; i$ (i.e., γ and γ' are equivalent). The equivalence relation, $\sim_\Gamma$, induces a partition of Γ into classes that represent equivalent losses. We denote the quotient strategy space by $\Gamma / \sim$, whose elements are induced equivalence classes. An element of $\Gamma / \sim$ will be termed a quotient strategy and will be denoted as $[\gamma]_\Gamma$, indicating the equivalence class that contains γ. Consider a strategy, γ, which produces $L^1(\gamma) = 10$ and $L^2(\gamma) = 10$, and another strategy, γ', which produces $L^1(\gamma') = 6$ and $L^2(\gamma') = 6$. From a global perspective, it is not clear which strategy would be preferable. Robot r^1 would prefer γ, while r^2 would prefer γ'. Both robots would, however, prefer either strategy to a third alternative that produced $L^1(\gamma'') = 5$ and $L^2(\gamma'') = 5$. These comparisons suggest that there exists a natural partial ordering on the space of strategies (see Figure 8.15).

Our interest is in finding the set of strategies that are minimal with respect to this partial ordering; these comprise all the useful strategies, because any other strategies would not be preferred by any of the robots. We define a partial ordering on the space $\Gamma / \sim$. The minimal elements with respect to $\Gamma / \sim$ will be considered as the solutions to our problem. For a pair of elements $[\gamma]_L, [\gamma']_L \in \Gamma / \sim$ we declare that $[\gamma]_L \leq [\gamma']_L$ *if* $L^i(\gamma) \leq L^i(\gamma')$ for each *i*. Two quotient strategies, $[\gamma]_L$ and $[\gamma']_L$, are incomparable if there exist some *i, j* such that $L^j(\gamma) < L^j(\gamma')$ and $L^j(\gamma) > L^j(\gamma')$. Hence, we can consider $[\gamma]_L$ to be either better than, worse than, equivalent to, or incomparable to $[\gamma']_L$. We can also apply the terms worse and better to representative strategies of different quotient strategies; for example we could say that γ is better than γ' if $[\gamma]_L \leq [\gamma']_L$. We can say that $[\gamma^*]_L$ is a minimal strategy if, for all $[\gamma]_L \neq [\gamma^*]_L$ such that $[\gamma]_L$ and $[\gamma^*]_L$ are not incomparable, we have $[\gamma^*]_L \leq [\gamma]_L$.

8.6.1 Feedforward Neural Networks

In our case we have implemented a feedforward or acyclic network due to its suitability for probabilistic calculations. We assume that a robot is capable of a switching between a fixed, maximum speed, $\|v^i\|$, and remaining motionless (this represents a typical resolution in geometric motion planning [27][121][207][108]). If, for example a robot is allowed to translate and rotate, then finite bounds might be given that limit the translational and angular speeds.

We next express the performance criteria for the robots. For each robot, r^i, we define a loss functional of the form

$$L^i(x_{init}, x_{goal}, u^1, \ldots, u^N) = \int_0^T g^i(t, x^i(t), u^i(t))dt + \sum_{j \neq i} c^{ij}(x(\cdot)) + q^i(x^i(T)) \tag{8.24}$$

which maps to the extended reals, and

$$c^{ij}(x(\cdot)) = \begin{cases} 0 & \text{if } x(t) \in X_{valid} \text{ for all } t \\ \infty & \text{otherwise} \end{cases} \tag{8.25}$$

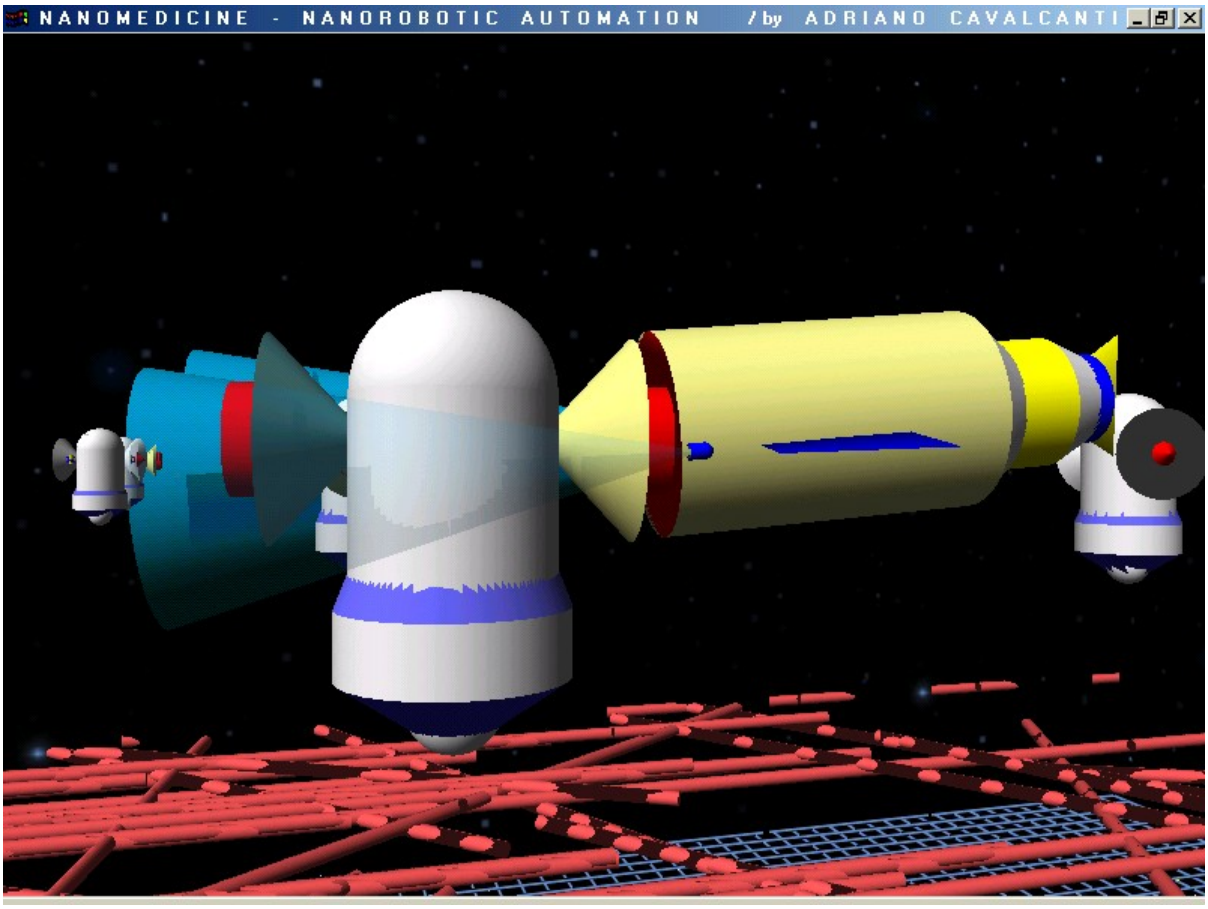

Figure 8.16: Nanorobot molecule delivery to the organ inlet (represented by the white cylinder).

and

$$q^i(x^i(T)) = \begin{cases} 0 & if \quad x^i(T) = x^i_{goal} \\ \infty & otherwise \end{cases} \tag{8.26}$$

The function g^i represents a continuous cost function, which is a standard form that is used in optimal control theory. We additionally require, however, that

$$g^i(t, x^i(t), u^i(t)) = 0 \qquad if \qquad x^i(t) = x^i_{goal} \tag{8.27}$$

This implies that no additional cost is received while robot r^i "waits" at x^i_{goal} until time T.

The middle term in the equation 8.24, $c^{ij}(x(\cdot))$ penalizes collisions between the robots. This has the effect of preventing any robots from considering strategies that lead to collision. The function also in the equation 8.24 $q^i(x^i(T))$ represents the goal in terms of performance. If a robot, r^i, fails to achieve its goal x^i_{goal}, then it receives infinite loss.

The model particularly implemented here is known as a Feedforward Network with logistic belief characteristics [170] that requires a lower computational effort in comparison with a backpropagation approach. The properties of Feedforward Neural Networks could be described by equation 8.28.

```
timeSeconds=Φ;
time_begin = time(NULL);
do{//Generate Feedforward Solutions
j=0;
for(move=0;move<nDestiny;move++)
{  neuronActiv=randomLayer(nDestiny-move);
   // Take the activated neurons.
search.sequence[j]=neuronSelect[neuronActiv];
  for(i=neuronActive;i<(nDestiny-move)-1;i++)
  {  neuronSelect[i]=neuronSelect[i+1];
  }
  j++;
}
// Compare the actual cost and take this
// solution if it has the best cost.
reckonNeuralCost();
time_end = time(NULL);
}while(time_end - time_begin < timeSeconds);
```

Table 8.2: Feedforward network algorithm.

$$pa(X_j) \subseteq \{X_1, X_2, ..., X_{j-1}\} \tag{8.28}$$

where *X* represents a vector, consisting of the two-valued random variables *X1, X2,..., Xn*, defining a topology composed of *N* stochastic neurons.

With *n* representing the range of a hidden layer, which leads the network to be optimized at the time-step *t*, it is related to each destiny to be achieved for the nanorobot through the simulation. The units in the network are organized into a two-dimensional *n* rows by *m* columns matrix A_{mn}, where *n* and *m* is the cost matrix of destinations to be performed by each evolutionary nanorobot, which tries to complete its set of tasks successfully as fast as possible. Let the output of the unit in row *i* and column *j* be v_{ij} = 1, where $i \neq j$. This means that the referred destiny is visited at the i^{th} stop, with v_{ij} = 0 otherwise. Therefore, a solution cost for the nanorobot routing could be expressed by equation 8.29.

$$\min R^t = \sum_i \sum_j v_i w_{ij} \tag{8.29}$$

The best solution was achieved by running our simulation based on the distance from each intended goal in the virtual environment configuration (Figure 8.16). A Feedforward Network pseudo code is described in Table 8.2.

8.7 Conclusion

The development of a control model and the design of a nanorobot automation has been described, emphasising the necessity to attain flexibility and robustness at the same time for any intelligent nanomolecular machine system prototyping, once considered to be complex, in relation to many aspects of the nano-worlds. The control model will be required to perform molecular manufacturing at nanoscale, which is even more evident for the specific problem that we are focusing on, that is related to the application of nanorobots in nanomedicine.

We have included in the control model the most suitable methodologies regarding the main aspects of adaptability, considering evolutionary and learning concepts, through the use of algorithms that are considered to be most applicable to complex problems, enabling low processing efforts for NP-Complete and NP-Hard problems. Moreover, the advantage of our approach is to provide a modular behaviour for the nanorobot, where the nanorobot is required to react in a well-tuned way with the events that came up from the surrounding uncertain 3D environment. For such a goal the sensor-based concept, also know as local perception sensing was used. The simulator implementation has required a higher performance, therefore the model described was developed using C++ [224], OpenGL [376], and RAPID [301].

The use of computer graphics for the design of the model has significantly helped in our understanding of many of those aspects related to nanoscale modelling such as: how a six-degree-of-freedom model has to interact in an environment where the agent must ensure a satisfactory response, even in a world dominated by quantum mechanics. The main expected requisites to be considered for a complex representation of kinematics aspects at nanoscales was specified and detailed. Furthermore the follow-up and theoretical considerations, as well as numerical simulations and experimentation at nanoscale could be very hard to understand intuitively, and even more so, to be designed and modelled, without the helpful use of computer graphics.

Thus, the main prerequisite and techniques for a detailed study of theoretical and practical characteristics related to the investigations of new paradigms for the development of control models applied to the coming new field of nanorobotics, has been considered and discussed. We expect that the design approach presented here could serve in some way as an inspiring framework for further studies on the field of nanosystems control design for mobile nanorobotics automation. The next chapter has a detailed discussion on the numerical results and the model achievements.

Chapter 9

Results Discussion

9.1 Introduction

This chapter shows how the proposed intelligent nanorobotics control design can be used to generalise and unify the analysis and computation methods for an adaptive behaviour model that has to survive in a stochastic way across different classes of motion planning problems with different kinds of interaction rules, such as competitive and collective robotics environments. The emphasis here is on defining models and formulating concepts, and validating the modelling through simulation and numerical analyses. The discussion in this chapter also provides a basis for future research on the direction taken in this dissertation.

Section 9.2 describes the main aspects related to the decision control model used by the nanorobot. Section 9.3 describes the competitive scenery where the nanorobot is required to react adaptively in an environment in which an adversary is found [69], thus the nanorobot control decision is validated showing its reactive characteristics and effectiveness in a competitive dynamic environment, where the two presented agents compete against each other [65]. Section 9.4 describes the collective robotics scenery [66] where the nanorobots in a collective fashion must work cooperatively to achieve a massively parallel assembly task in a 3D environment to improve the organ inlets' nutritional levels [74]. Section 9.5 shows the numerical optimization for the motion control problem [71], where the nanorobot must visit the set of organ inlets that has to be attended and verified, to provide a well-tuned decision for the next time-step in the simulation scenery. Section 9.6 concludes this chapter by discussing issues that result from generalizing and comparing new paradigms on the problem related to the design of nanorobots [73][66], declaring the presented work as a feasible framework for the complex problem of nanorobotics control modelling applied to nanomedicine [65].

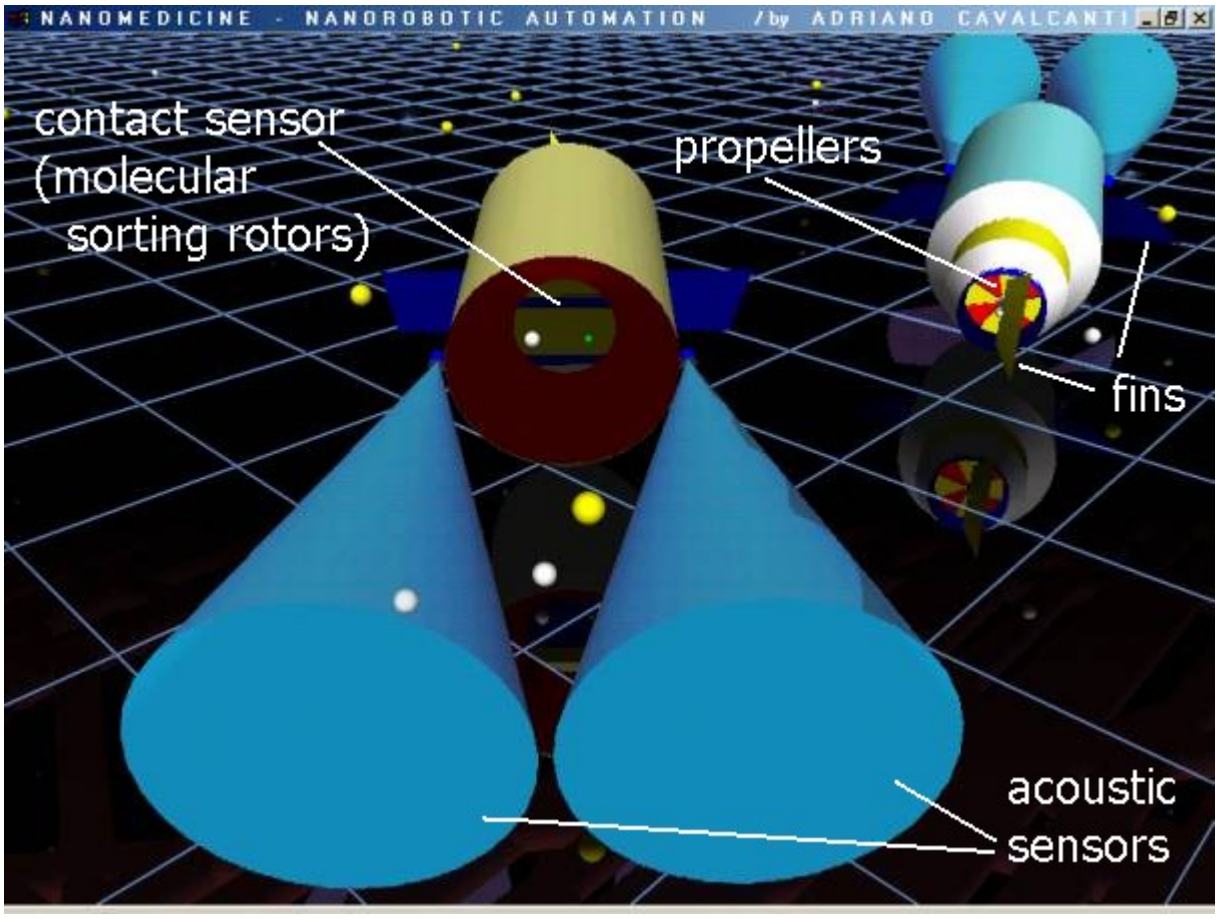

Figure 9.1: Nanorobots' design - acoustic sensors, molecular sorting rotor, fins and propellers.

9.2 Evolutionary Decision for Dynamic Problems

The described nanorobots (Figure 9.1) should react adaptively in an uncertain and dynamic environment with a well defined pre-programmed set of actions, such as finding, capturing, and transporting biomolecules, which are required for the biomolecular assembly task and delivery. In our architectural implementation, we use real time and parallel processing techniques to provide a real time coherent adaptive behaviour with the visualization of the dynamic 3D virtual environment in real time. Basically each nanorobot r is responsible for handling some different molecules, where such molecules are designated as e, g and h as the kind of molecules to be assembled by r, therefore:

$$\beta \quad \begin{cases} r_{\Omega} = A & \Rightarrow \quad \beta = e, \\ r_{\Omega} = B & \Rightarrow \quad \beta = h, \end{cases} \tag{9.1}$$

$$\delta = g. \tag{9.2}$$

where A and B represent distinct nanorobot types depending on the interaction rule specified for each investigated scenery. Thus A and B will imply two different nanorobot behavioural characteristics, which will be determined and discussed in more detail in the sequence of this chapter. Our fitness/objective function observes the same mathematical modelling discussed in the mathematical model described in Chapter 8 in equation 8.4, where the nanorobots optimize the protein levels for the

selected organ inlets. The nanorobot has to decide what organ inlets to attend at each time-step once the delivery for the organ inlets in most of the cases assumes not a continuous but only an integer value and the nanorobot cannot attend all the organ inlets at the same time. Just two of each ten organ inlets was randomly specified to have a possible continuous nutritional injection working as the gap variable for the delivery mean problem, which becomes more complex and significant once the nanorobot decides what organ inlets to attend at each time-step of the dynamic decision. For the organ inlets' nutritional levels in both scenes discussed next, we have adopted as the most ideal nutritional target levels at 50% of the relative organ inlet nutritional capacity. In general trems it is assumed that levels lower than 20% or higher than 80% are characterised as a possible deficiency or overdose case. Therefore, as we will observe, in very rare occurrences when such a circumstance arises, i.e. nutritional levels outside the 20% to 80% operational ranges, the medically beneficial nanorobotic behaviour acts immediately to bring the nutritional level into the desirable nutritional ranges. Those numerical simulations will be described in more detail throughout this chapter.

The delivery mean was established, that each 10 organ inlets are expected to deliver an amount of 100 proteins as a relative symbolic amount to set up a target, which has to be managed by our nanorobots. Thus it does not matter what amount of nanorobots are in the simulated scenery, their loading capacity will be adjusted in order to correspond to the respective targets in the simulation. Obviously the nanorobots' load capacity won't be larger nor smaller than the delivery target, which implies that the nanorobots must make a well tuned decision on how to administer their delivery capacity, considering that at each time-step in the simulation any robot cannot attend all the organ inlets in the 3D environment at the same moment on the same simulated interval of time. So, the nanorobot has to decide what organ inlets can stay out of the delivery list at the actual interval of time and what organ inlet cannot wait till the next time in the dynamic simulation. For example, for a situation where we have 30 organ inlets with one nanorobot agent, the nanorobot agent's load capacity is adjusted to a maximum value of 300 proteins at each round simulated; for 60 organ inlets, the nanorobot's load capacity is 600 proteins for each respective round. Of course, the same modelling applies for each nanorobot in the environment. Nevertheless for scenery with collective robotics, where for each nanorobot is attributed a fixed number of 10 organ inlets, each nanorobot will have a load/delivery capacity pre-established as 100 proteins for each round simulated. It does not matter if the environment is comprised of 30 or 60 organ inlets.

The functional decision module uses genetic algorithms as a suitable approach for dynamic behaviour dealing with a highly complex environment, which provides a nanorobot decision in an efficient fashion with low time processing effort. Thus the proposed model is not dealing with any kind of nanorobot self-replication behaviour [110], instead the model uses the evolutionary approach strictly for the combinatorial analyses.

9.3 Competitive Scenery

The competitive scenery presents a pair of nanorobots competing against each other (Figure 9.2) in the sense that while one agent tries to improve the nutritional state of a set of organ inlets in the represented living three-dimensional environment, the nanorobot adversary tries to debilitate it through the injection of inappropriate assembled substances into the same organ inlets. The aim of such an approach is to demonstrate the timely reactive characteristics of the presented nanorobot model.

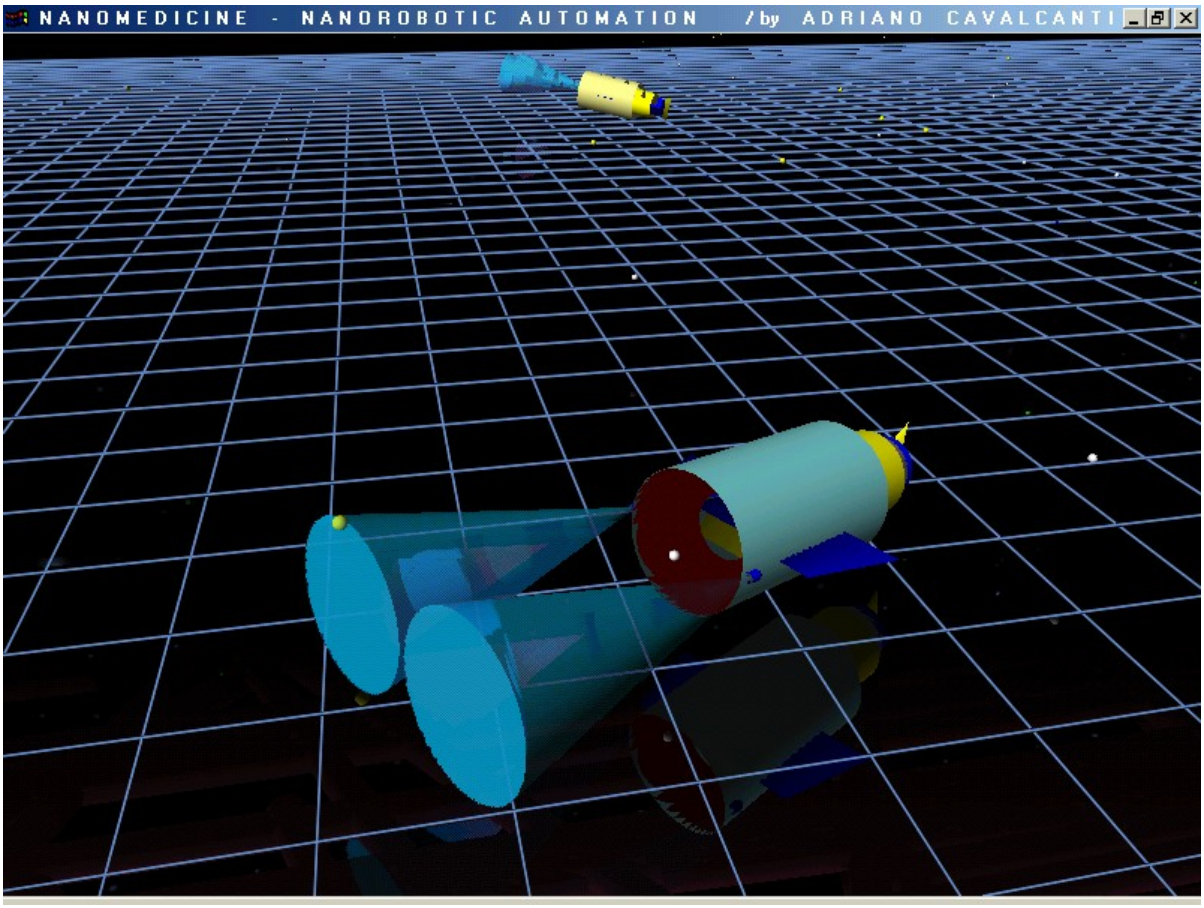

Figure 9.2: Competitive agent and adversary in action.

The model behaviour must achieve a satisfactory adaptive reaction in front of a set of disturbances produced by an adversary, which is acting dynamically in the same environment.

9.3.1 Environment Description

The competitive scenery is comprised of 2 nanorobots, 30 organ inlets and *n* obstacles (Figure 9.3). Therefore we could express the combinatorial complexity of the present problem as 2^{30} for each nanorobot in the scenery. The complexity of the problem refers to the fact that each nanorobot must decide what action to take in the next time-step in the simulation scenery. Both nanorobots are required to move in a 3D environment, thereby swimming in an environment with a motion control of 6 degrees-of-freedom to avoid obstacles in order to deliver the assembled biomolecules delivery to the organ inlets (Figure 9.4).

9.3.2 Nanorobots Interaction Rule

We intend to construct and validate a nano-planning system, where the use of competitive evolutionary agents will enable a better-tuned validation of the nanorobot control model under study. The competitive nanorobot interactive rule is described in Table 9.1. The *min* denotes the minimum defined to be captured by each nanorobot at time step *t*.

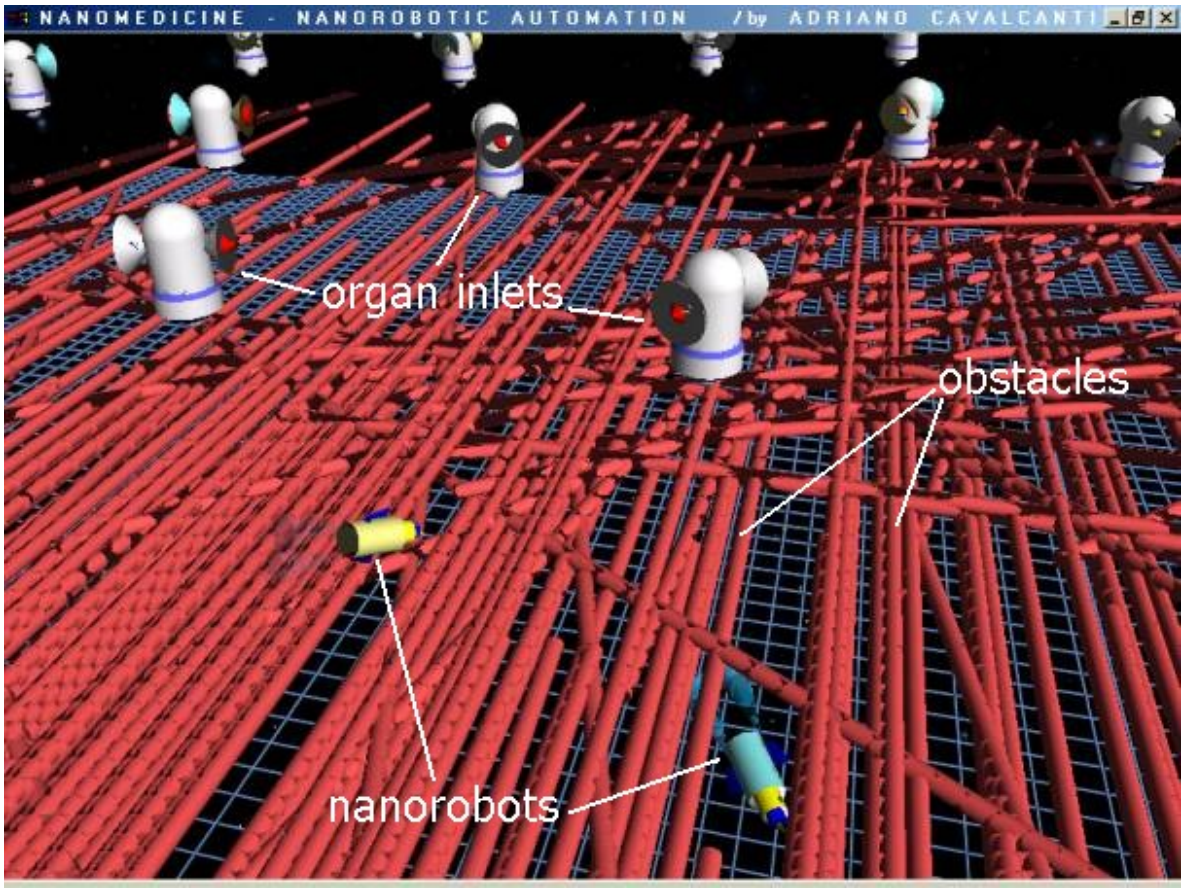

Figure 9.3: Competitive scenery, top camera view.

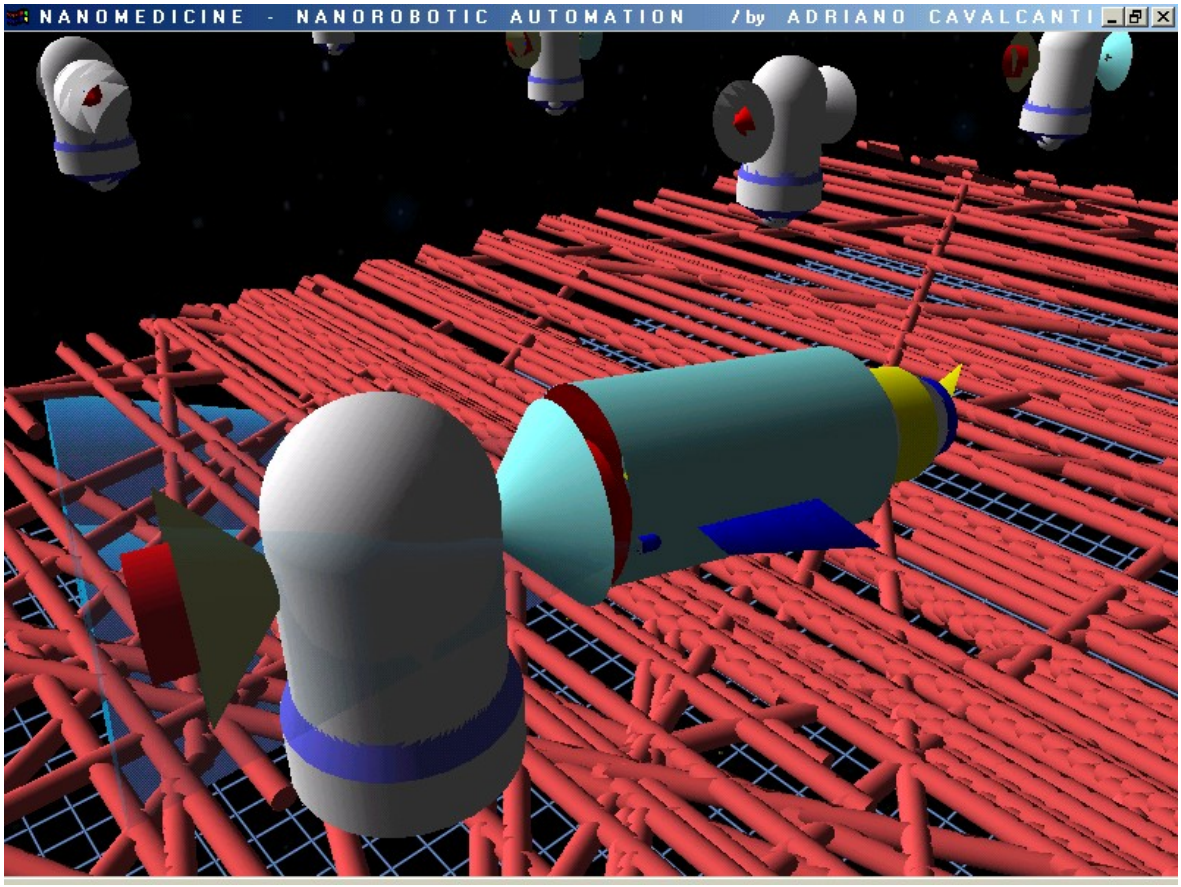

Figure 9.4: Nanorobot adversary delivery to the organ inlet - represented by the white cylinder.

Step 1:	r_{Ω} walk randomly to capture β and δ;
Step 2:	if ∑ β = ∑ δ → assemble f(r_{Ω})= β + δ;
Step 3:	if ∑ f(r_{Ω}) < min repeat step 1;
Step 4:	r_{Ω} achieve next delivery goal regarding the delivery queue;
Step 5:	if $\Omega = B$ go to step 7, otherwise next step;
Step 6:	if delivery_NOT_overdose = true → next step; otherwise go to step 8;
Step 7:	delivery: f(r_{Ω}) = f(r_{Ω}) -1;
Step 8:	if f(r_{Ω})>0 repeat step 4;
Step 9:	repeat step 1;

Table 9.1: competitive scenery nanorobot interaction rule.

Both agents react adaptively based on their local perception of any action performed by an adverse decision with the environment visualization in real time [78]. The parameter Ψ in the equation 9.3 defines the distinct behaviour of nanorobots *A* and *B*, which is denominated here respectively as the agent and adversary competitive function, directly influencing the evolutionary decision control model defined by the equation 8.4 from Chapter 8.

$$\Psi \quad \begin{cases} \Omega = A \Rightarrow & \Psi = 1; \\ \Omega = B \Rightarrow & \Psi = -1; \end{cases} \tag{9.3}$$

As we are going to see, the action performed using the sensor-based system has generated a competitive nanorobot coherent behaviour, which was observed in the proposed model simulation.

9.3.3 Nanorobots Competitive Results

Tables 9.2 and 9.3 respectively show the nanorobot agent and nanorobot adversary decisions about the injection of assembled substances into the set of 30 organ inlets. As we observe, the action performed by the agent is influenced by the decision made by the nanorobot adversary at the same time-step in the dynamic environment. On one side the nanorobot adversary is injecting an amount of assembled substances that convey elements into the organ inlets that take off the actual nutritional state, minimizing their levels. On the other side the nanorobot agent is observing with its local perception sensors the organ inlets' nutritional level in order to maximize it, and trying to bring all levels as close as possible to the optimal target level, which was established as 50%. In the actual scenery with 30 organ inlets one could see in Tables 9.4 and 9.5 that both the nanorobot agent and adversary have achieved their protein delivery target. In the same Tables 9.4 and 9.5 one could see the highest and lowest nutritional levels resulting from each nanorobot's action respectively.

Inlet	time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	delivery	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	1	0
	level	30	42	34	46	37	49	20	32	44	56	27	39	51	42	54	46	37	49	41	32	44	56	27	39
2	delivery	0	0	1	0	1	1	1	0	0	1	0	0	0	0	1	1	0	1	0	0	1	0	0	1
	level	56	62	58	55	52	49	37	42	48	44	41	46	43	49	45	42	48	44	41	46	43	40	45	42
3	delivery	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0	0
	level	46	42	48	45	41	48	34	40	47	53	39	46	42	48	55	61	57	44	50	57	53	49	46	52
4	delivery	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0	0	1
	level	43	56	66	61	50	45	52	47	53	44	39	45	56	45	52	63	58	52	64	62	46	58	70	50
5	delivery	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0
	level	25	38	52	41	31	45	34	24	37	51	40	30	44	57	23	36	50	63	29	43	56	46	35	49
6	delivery	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1
	level	59	75	58	58	74	57	56	72	72	71	54	54	70	53	52	68	68	51	50	66	49	49	65	48
7	delivery	0	1	1	0	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	1
	level	57	54	45	48	45	49	40	44	47	45	42	45	43	40	44	47	45	42	45	49	46	44	47	38
8	delivery	0	1	0	0	1	0	1	0	1	0	1	0	0	1	1	0	1	1	0	0	0	0	1	0
	level	56	53	60	56	54	60	61	68	68	64	65	61	70	56	54	62	56	55	60	61	65	73	60	56
9	delivery	0	1	0	0	1	1	0	1	0	1	0	1	0	0	1	0	0	1	0	1	1	1	0	1
	level	59	58	57	63	56	55	60	54	59	52	57	57	56	61	60	60	65	64	69	69	62	61	66	66
10	delivery	0	1	1	1	0	1	0	0	1	0	0	0	1	0	1	0	1	0	0	0	1	0	0	1
	level	56	53	50	47	54	51	48	55	41	48	45	52	49	56	43	50	36	44	40	48	34	41	48	45
11	delivery	1	0	0	0	1	0	1	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	1	0
	level	30	22	34	46	37	49	20	32	44	56	47	19	30	42	54	46	37	49	20	32	44	56	27	39
12	delivery	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	0	0	1	0	0	0
	level	48	53	50	46	52	40	45	42	48	44	50	38	43	40	45	33	39	44	41	46	43	49	45	51
13	delivery	1	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0
	level	46	52	48	45	41	48	44	50	37	43	49	36	42	48	45	51	48	54	60	66	63	59	56	52
14	delivery	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	0	1	1	0	0	0	1
	level	54	56	53	47	57	51	59	54	66	46	55	48	57	51	55	67	47	56	53	29	38	47	57	54
15	delivery	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	0
	level	25	38	52	41	31	45	34	48	61	27	40	54	19	33	47	36	50	15	29	43	56	22	35	49
16	delivery	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0
	level	59	75	58	58	74	57	56	72	55	55	71	54	53	69	52	52	68	67	50	50	66	49	48	64
17	delivery	0	1	0	0	1	1	0	0	0	0	1	1	1	0	0	0	1	0	1	1	0	1	0	0
	level	57	47	51	55	52	49	46	50	47	51	48	45	36	40	44	47	45	48	45	43	46	44	47	45
18	delivery	0	1	0	0	1	0	0	1	0	1	0	1	0	1	1	0	0	0	1	0	0	1	0	0
	level	56	51	59	56	53	49	56	48	54	48	55	52	55	51	38	39	44	52	51	59	65	62	66	72
19	delivery	1	0	0	1	1	0	0	1	0	1	0	1	0	0	0	1	0	1	0	1	1	1	0	1
	level	53	58	57	57	50	55	60	54	59	58	63	57	62	67	66	66	65	64	69	63	62	55	60	60
20	delivery	1	0	0	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1
	level	46	43	50	47	54	40	48	44	41	48	45	42	49	46	53	60	47	54	51	58	55	62	59	55
21	delivery	1	0	0	0	1	0	1	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	1
	level	30	42	34	46	37	49	41	52	44	35	47	39	30	42	54	25	37	49	41	52	44	35	47	39
22	delivery	0	1	0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	0	0	1	0	1	0	0
	level	56	44	50	55	52	49	54	42	48	44	50	55	52	49	54	51	48	53	58	46	52	40	45	51
23	delivery	0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0
	level	56	52	48	55	61	57	54	50	47	53	49	56	62	58	55	61	67	54	60	66	73	59	66	72
24	delivery	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0
	level	54	48	56	47	56	67	61	69	47	55	59	71	55	49	60	55	66	60	71	49	43	53	48	50
25	delivery	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0
	level	49	38	52	41	31	45	58	72	37	27	40	54	44	33	47	60	74	39	29	43	56	70	35	49
26	delivery	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0
	level	59	59	58	74	74	73	73	56	55	71	71	70	53	53	69	52	51	67	67	50	49	65	48	48
27	delivery	0	0	1	0	0	0	0	1	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0	0
	level	57	54	45	48	52	56	59	50	47	51	42	45	43	46	50	41	45	35	39	43	40	44	41	45
28	delivery	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	level	56	41	37	42	39	46	53	40	45	54	50	58	54	58	55	43	39	43	48	44	40	36	44	40
29	delivery	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0
	level	59	52	51	51	50	55	54	48	53	58	57	63	62	67	66	60	59	64	63	57	56	55	54	54
30	delivery	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
	level	56	43	40	47	44	40	48	44	51	48	45	52	49	56	53	40	36	44	51	48	44	41	48	45

Table 9.2: Competitive scenery with organ inlets' nutritional levels for the nanoroborot adversary.

Inlet	time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	delivery	1	0	1	1	1	0	1	1	1	0	1	1	1	1	1	0	1	1	0	1	1	0	1	1
	level	51	42	54	66	57	49	41	52	64	56	47	59	71	62	74	46	57	69	41	52	64	56	47	59
2	delivery	1	1	0	1	1	0	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0	1	1	1
	level	65	70	58	64	61	49	45	51	56	44	50	46	52	57	54	51	56	44	50	55	43	49	54	51
3	delivery	0	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	0	1	1
	level	46	52	58	45	51	48	44	50	57	53	49	56	52	58	65	71	57	54	60	66	63	49	56	62
4	delivery	1	1	0	1	0	1	0	1	1	0	1	1	1	1	1	1	0	1	1	0	1	1	0	0
	level	61	72	66	73	50	58	52	59	65	44	51	62	67	58	69	80	58	69	80	62	64	75	70	50
5	delivery	1	1	1	0	1	1	0	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0	1	0
	level	49	62	76	41	55	69	34	48	61	75	40	54	68	57	47	60	74	63	53	67	80	46	59	49
6	delivery	1	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0
	level	75	75	58	74	74	57	73	72	72	71	54	70	70	53	69	68	68	51	67	66	49	65	65	48
7	delivery	1	0	1	0	1	0	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0	1	0	1
	level	63	54	51	48	52	49	46	50	54	45	48	52	43	46	50	54	45	48	52	56	46	50	47	45
8	delivery	1	1	0	1	1	1	1	1	0	1	0	1	0	1	1	1	1	1	1	1	1	0	0	1
	level	66	64	60	69	64	72	72	80	68	77	65	74	70	69	66	70	66	64	65	69	77	73	60	67
9	delivery	1	0	1	0	1	1	0	1	0	1	1	0	1	1	0	1	1	1	1	0	1	1	1	1
	level	65	58	63	63	62	61	60	60	59	58	63	57	62	67	60	66	71	70	75	69	68	67	72	72
10	delivery	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	1	1	1
	level	67	63	60	57	64	51	58	55	51	48	55	63	59	56	53	50	47	44	51	48	44	51	59	55
11	delivery	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1
	level	30	42	54	66	57	49	41	52	64	76	47	39	51	62	74	46	57	49	41	52	64	56	47	59
12	delivery	1	1	0	1	0	1	0	1	1	1	0	1	0	1	0	1	1	0	1	1	1	0	1	1
	level	56	62	50	55	52	49	45	51	56	53	50	46	43	49	45	42	48	44	50	55	52	49	54	60
13	delivery	1	1	1	0	1	1	1	0	1	1	0	1	1	0	1	1	1	1	1	1	0	1	0	1
	level	56	62	58	45	51	57	54	50	47	53	49	46	52	48	55	61	57	64	70	76	63	69	56	62
14	delivery	1	1	0	1	1	1	0	1	0	1	1	1	0	1	1	0	1	1	0	1	1	1	1	0
	level	62	69	53	63	67	65	59	72	66	61	67	63	57	61	72	67	61	74	53	43	53	63	74	54
15	delivery	1	1	1	0	1	1	1	1	0	1	1	0	1	1	0	1	0	1	1	1	0	1	1	1
	level	49	62	76	41	55	69	58	72	61	51	65	54	44	57	47	60	50	39	53	67	56	46	59	73
16	delivery	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0
	level	75	75	58	74	74	57	73	72	55	71	71	54	70	69	52	68	68	67	50	66	66	49	65	64
17	delivery	0	1	1	1	1	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1
	level	57	54	58	61	58	49	53	50	54	58	55	45	43	46	50	54	51	55	52	49	53	50	47	51
18	delivery	1	1	0	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0
	level	67	63	59	67	53	60	63	58	63	58	67	59	66	51	43	48	56	64	63	69	73	70	76	72
19	delivery	1	0	1	0	1	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	0	1	1	1
	level	59	58	63	57	56	61	60	60	65	64	63	63	68	67	72	66	71	70	69	69	62	61	66	66
20	delivery	0	1	1	1	0	1	1	0	1	1	0	1	1	1	1	0	1	0	1	1	1	0	1	1
	level	46	53	60	57	54	51	58	44	51	59	45	52	59	56	63	60	57	54	61	68	65	62	69	66
21	delivery	1	0	1	1	1	1	1	1	0	1	1	0	1	1	0	1	1	1	1	1	0	1	1	1
	level	51	42	54	66	57	69	61	73	44	56	67	39	51	62	54	46	57	69	61	73	44	56	67	59
22	delivery	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	1	0
	level	56	53	58	55	61	57	54	51	56	53	58	64	61	57	63	60	56	62	58	55	52	49	54	51
23	delivery	0	1	1	1	0	1	1	0	1	1	1	1	0	1	1	1	0	1	1	1	0	1	1	0
	level	56	62	58	65	61	67	64	50	57	63	59	66	62	68	65	71	67	64	70	76	73	69	75	72
24	delivery	0	1	1	1	1	0	1	0	1	1	1	0	0	1	0	1	0	1	0	0	1	1	1	1
	level	54	61	63	61	72	67	75	69	61	64	76	71	55	66	60	71	66	77	71	49	58	69	56	68
25	delivery	0	1	1	0	1	1	1	0	0	1	1	1	0	1	1	1	0	0	1	1	1	0	1	1
	level	49	62	76	41	55	69	82	72	37	51	65	78	44	57	71	84	74	39	53	67	80	70	59	73
26	delivery	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1
	level	59	59	75	74	74	73	73	56	72	71	71	70	53	69	69	52	68	67	67	50	66	65	48	64
27	delivery	0	0	1	1	1	1	0	1	1	0	1	1	1	1	0	1	0	1	1	0	1	0	1	0
	level	57	54	51	55	58	62	59	57	54	51	48	52	49	53	50	47	45	42	45	43	46	44	47	45
28	delivery	0	0	1	0	1	1	0	1	1	0	1	0	1	0	0	0	1	1	0	0	0	1	0	1
	level	56	41	46	42	50	56	53	49	58	54	61	58	62	58	55	43	47	51	48	44	40	48	44	44
29	delivery	0	0	1	0	1	0	0	1	1	0	1	0	1	0	0	0	1	0	0	0	0	1	0	1
	level	59	52	57	51	56	55	54	54	59	58	63	63	68	67	66	60	65	64	63	57	56	61	54	60
30	delivery	0	0	1	0	0	1	0	1	0	0	1	0	1	0	0	0	1	1	0	0	0	1	0	1
	level	56	43	50	47	44	51	48	55	51	48	55	52	59	56	53	40	47	54	51	48	44	51	48	55

Table 9.3: Competitive scenery with organ inlets' nutritional levels for the nanoroborot agent.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Delivery goal	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300
adversary lowest	25	22	34	41	31	40	20	24	37	27	27	19	19	33	23	25	36	15	20	29	34	22	27	38
adversary highest	59	75	66	74	74	73	73	72	72	71	71	71	70	69	69	68	74	67	71	69	73	73	70	72

Table 9.4: Competitive scenery with highest and lowest levels for the nanoroborot adversary.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Delivery goal	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300
agent lowest	30	41	46	41	44	48	34	44	37	44	40	39	43	46	43	40	45	39	41	43	40	44	44	44
agent highest	75	75	76	74	74	73	82	80	72	77	76	78	71	69	74	84	74	77	80	76	80	75	76	73

Table 9.5: Competitive scenery with highest and lowest levels for the nanoroborot agent.

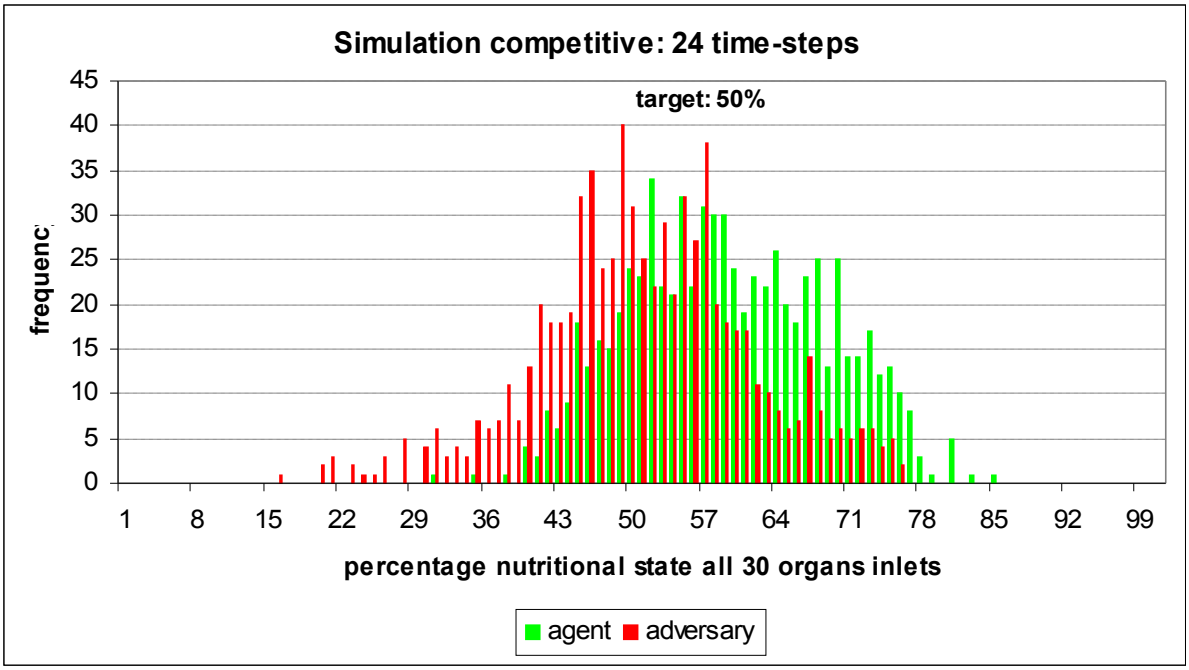

Figure 9.5: Histogram of competitive scenery with organ inlets' nutritional levels.

For a total of 720 organ inlets' nutritional levels observed through a scenery of 24 intervals of time, we can observe that there are only 6 occasions when the nanorobot adversary could put the nutritional level in or under the pre-established nutritional level of 20%, which determines a risk of nutritional insufficiency - see Table 9.2. This represents a percentage of 0,0083% for the time in the simulated scenery. One could observe in the same instant the nanorobot agent has acted above such negative insufficiencies registered in the organ inlets nutritional levels, thus in the same time-step where it was observed that such insufficiencies were caused by the nanorobot adversary, i.e. time-steps {7, 12, 13, 18, 19}, we could observe that no one insufficiency remains after the nanorobot agent reaction - see Table 9.4 and Table 9.5.

The same satisfactory behaviour could be observed regarding the organ inlets' nutritional level related to overloads. In Table 9.3 we should note that from the nutritional levels of 720 organ inlets observed through a scenery of 24 intervals of time, only 7 occurrences of levels rounding 80% were registered. Representing 0.0097% for all nutritional level performances registered.

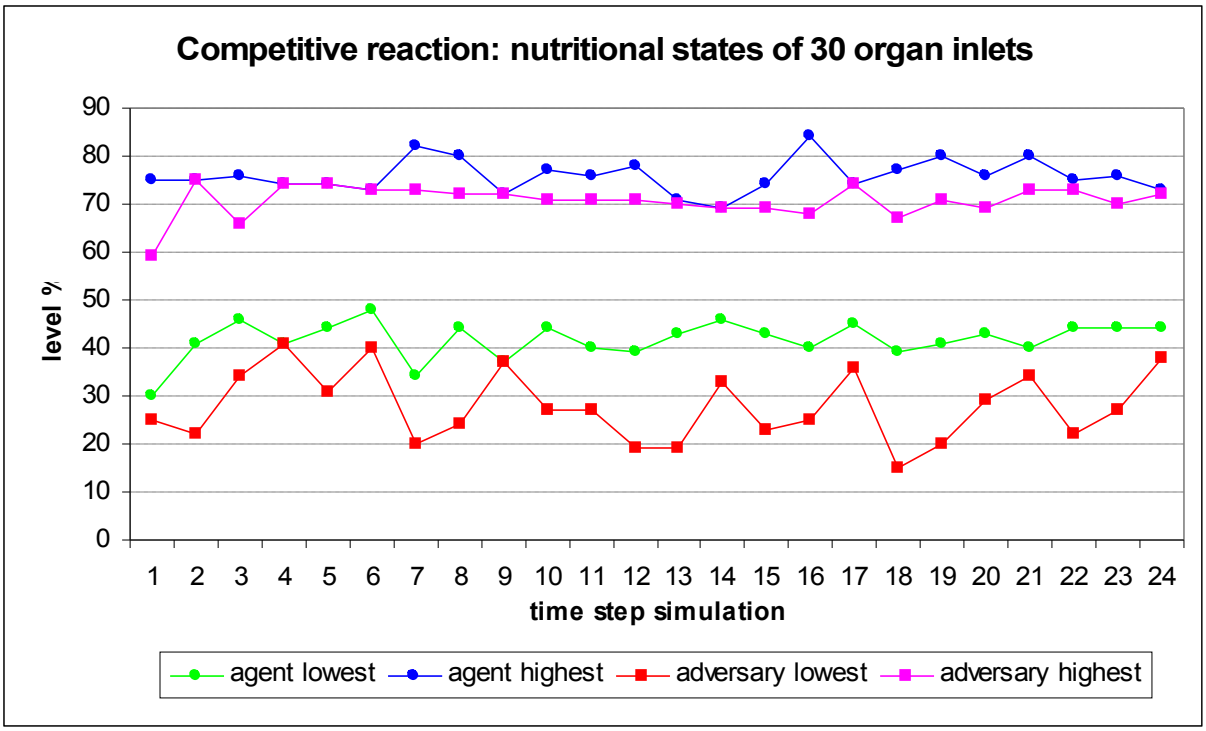

Figure 9.6: Upper and lower organ inlets' nutritional levels for competitive scenery.

As it might be observed, even in such cases, the nanorobot has carried out an action for the next time-step simulation by not injecting any proteins into the organ inlets, thereby reducing the levels - see Table 9.3. Furthermore the nanorobot agent has ensured that not one organ inlet has achieved levels near of an insufficient nutritional state, as we could see in Table 9.5.

A general adversary objective between both competitive nanorobots could be better ilustrated by Figure 9.5, where both characteristic behaviours could be better observed. While the nanorobot agent is clearly forcing the organ inlets' levels to increase, an equivalent counter reaction occurs with the injections administered by the nanorobot adversary. Although most of the nutritional levels remains at the desired level of 50%, the influence of both nanorobots is clear, in the sense that one is trying overall to maximize the nutritional levels and the other is acting to reduce it.

The aspect of paying more attention to the highest or lowest level registered at the organ inlets through the simulated scenery, is to be considered as the most important for nanorobot control behaviour. We can observe that the nanorobots react in a well tuned fashion, in the sense that within a satisfactory time when the nanorobot adversary achieves a lower nutritional level, the agent reacts to the respective organ inlet increasing that nutritional level - see the Figure 9.6. Here we could equally observe that the highest and lowest levels for the agent and adversary have a distinct performance in the sense that the levels for the action performed by each nanorobot is respectively maximized or minimized in a significant way.

Figure 9.7: Competitive agent and adversary robustness.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Delivery goal	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600
adversary lowest	15	12	18	8	7	19	11	14	27	17	6	9	21	12	13	7	7	19	11	23	22	12	17	15
adversary highest	49	54	55	64	60	60	63	62	62	61	61	60	60	59	62	62	61	57	60	57	58	59	59	62

Table 9.6: Competitive robustness with highest and lowest levels for the nanoroborot adversary.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Goal	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600
agent lowest	37	33	32	32	28	30	24	32	27	33	31	29	32	33	33	26	28	30	31	33	33	33	25	29
agent highest	63	64	67	64	69	63	66	63	64	66	66	68	62	63	65	64	65	64	67	63	63	66	62	63

Table 9.7: Competitive robustness with highest and lowest levels for the nanoroborot agent.

9.3.4 Nanorobots Competitive Control Robustness

To evaluate the stability and robustness of the model, we have increased the complexity of the problem (Figure 9.7). Thus the number of organ inlets has increased 100%, i.e. now we have 60 organ inlets that will be visited by one nanorobot agent and one nanorobot adversary at each turn of the dynamic environment, thereby each nanorobot load capacity has equally grown by up to 600 nutrients at each time-step simulation.

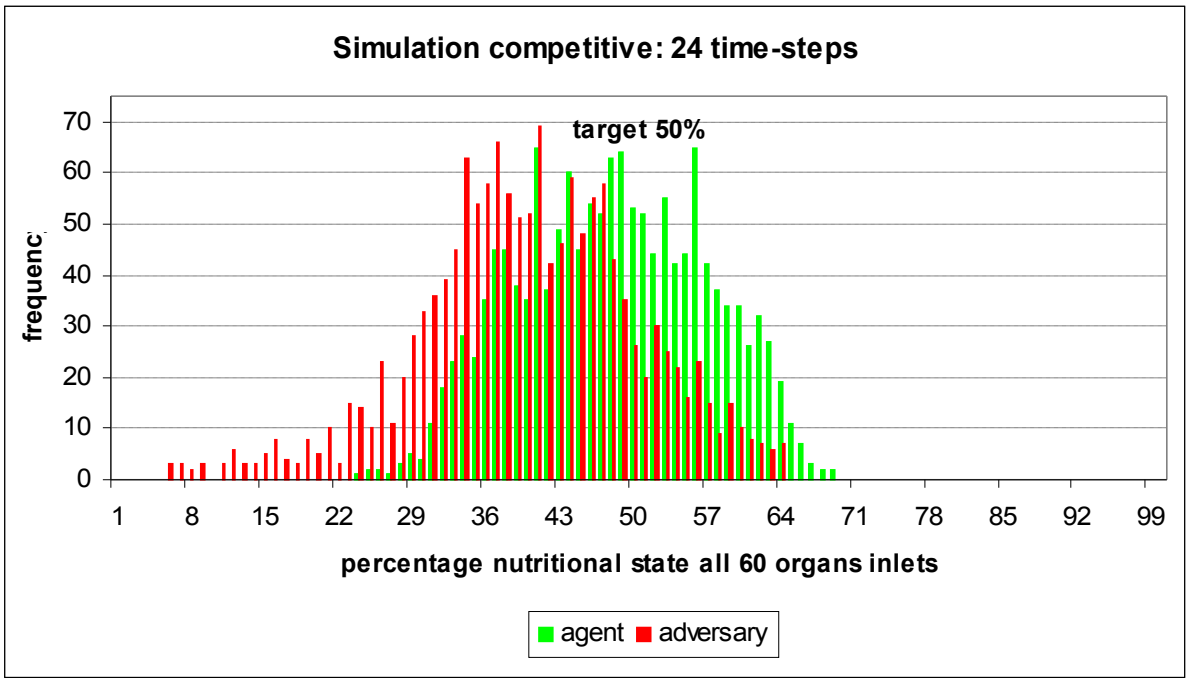

Figure 9.8: Histogram of competitive robustness with organ inlets' nutritional levels.

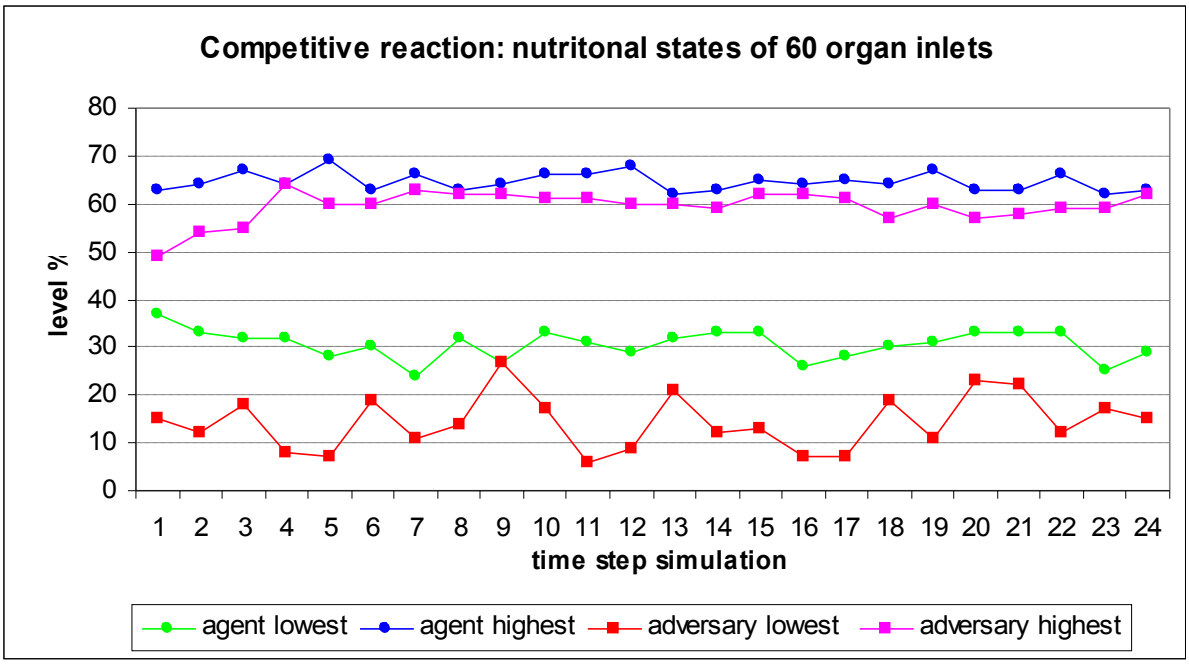

Figure 9.9: Upper and lower organ inlets' nutritional levels for competitive robustness.

As can be seen in Table 9.6, although the nanorobot adversary has shown a certain degree of deeper nutritional levels, with a more frequent observation of levels ranging under 20%, this means a higher possible registration of an insufficient nutritional state, but on the other side the nanorobot agent has demonstrated its capability to bring all organ inlets' nutritional levels into the desirable range of 20% to 80% - see Table 9.7. With reference to substances delivery goal, it was attended equally successfully by both nanorobots as detailed in the same tables.

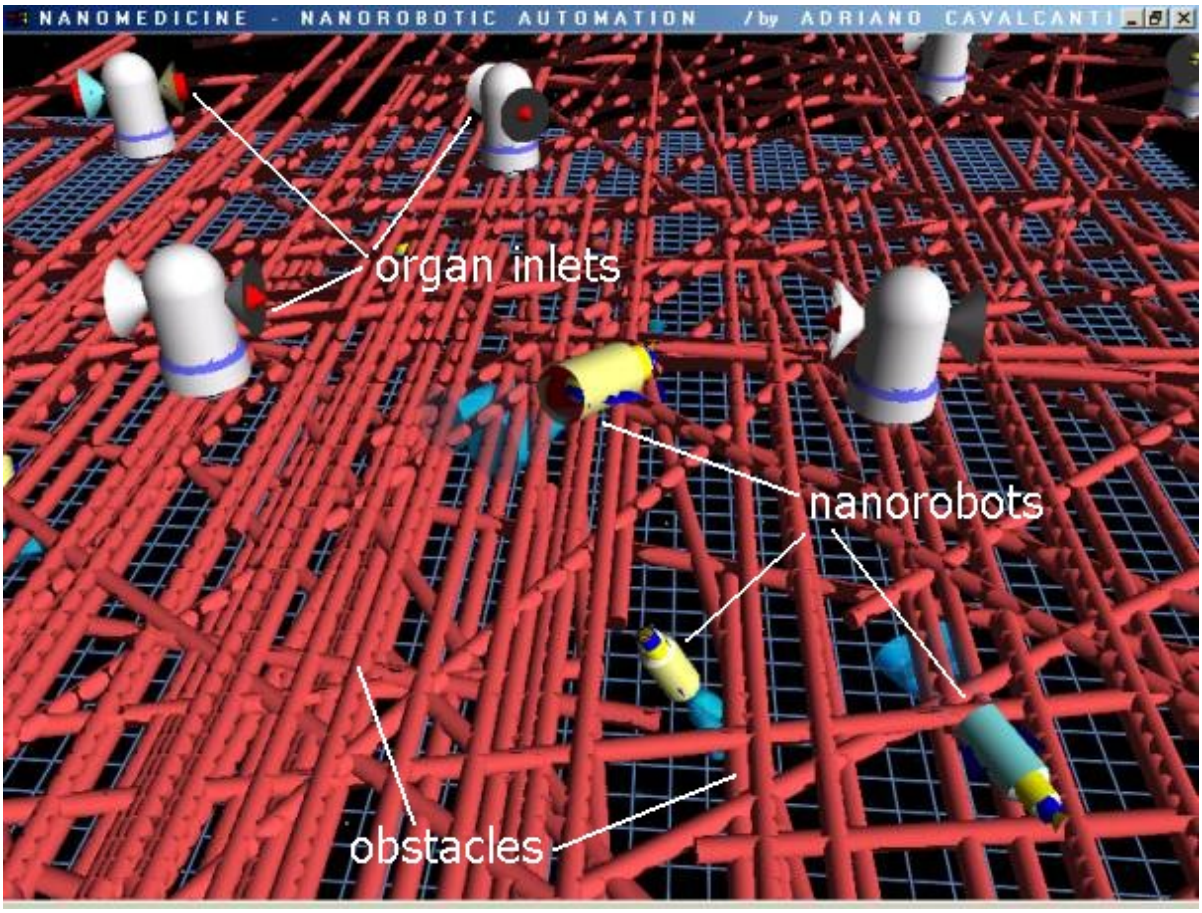

Figure 9.10: Collective scenery, top camera view.

The same satisfactory behaviour could be observed in a more detailed fashion in Figures 9.8 and 9.9, where the nanorobot agent is able to keep the levels of all the organ inlets in a very satisfactory range of operation, bringing all the nutritional levels to the desirable level, even though the nanorobot adversary would keep trying to reduce the organ inlets' nutritional state.

9.4 Collective Robotics Scenery

The approach presented here is suggested by the necessity of the massive nanoassembly task, which would be required to be performed by groups of collective nanorobots in a timely parallel set of operations in a stochastic environment (Figure 9.10). Many of the concepts presented in collective robotics consider studies based on social insects with the aim of enabling multi-robotics groups to cooperate amongst themselves. Stigmergy, a term coined by the biologist P. Grassé [155], means to promote work by the effect of previous work, is a principle which is finding its way from the field of social insects to collective robotics [35][355]. With their limited repertoire of behavioural acts, social insects display an amazing competence in building nest structures. From the simple nests produced by the blind bulldozing of ants to the termite homes that stand over several meters tall, all of these result from common tasks coordination that does not appear to depend on interaction between the agents but rather on the object they act upon [136][346].

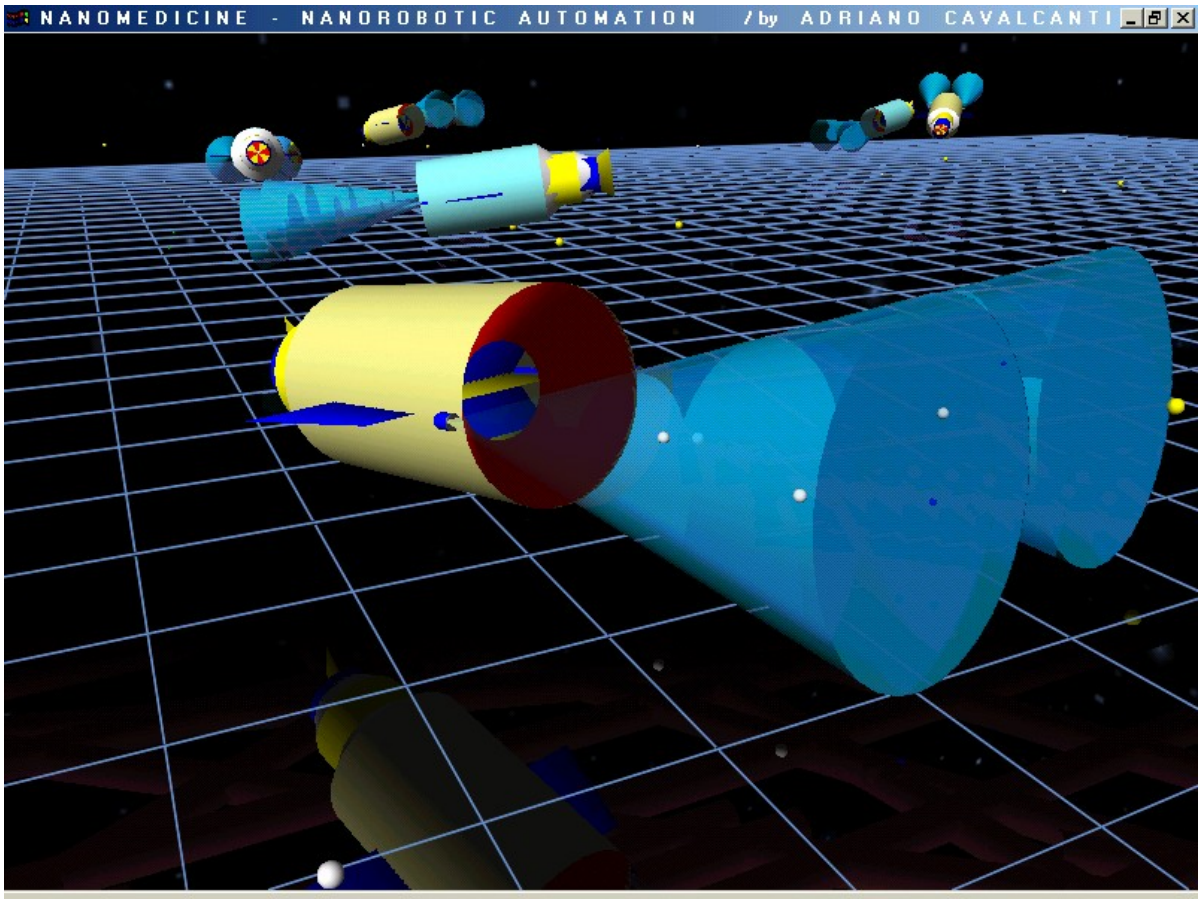

Figure 9.11: Collective team behaviour.

9.4.1 Environment Description

The collective scenery is comprised of 2 teams with each team formed by 3 nanorobots. Thus we have a total of 6 nanorobots in the environment that are responsible for 30 organ inlets requiring proteins. While in the competitive scenery the nanorobots compete against each other, here the nanorobot teams have to work together in a sequenced coordinated assembly task (Figure 9.11), avoiding the set of stochastic obstacles present in the 3D virtual nano-world in order to deliver protein to the organ inlets. We have the first nanorobot team where the nanorobots belonging to this team are identified as nanorobots *A* and those other ones belonging to the second group are identified as nanorobots *B*. Basically what distinguishes the *A* and *B* nanorobots is the kind of pre-defined molecular assembly task to be performed by each one. Furthermore nanorobot *A* and *B* have to present a sequenced delivery fashion in relation to its organ inlets. In both cases they are trying to maximize the organ inlets' nutritional level, attending a pre-defined nutritional consumption from the set of organ inlets present in the environment.

The use of local perception should in most cases be quite sufficient for the overall set of tasks that our nanorobots are designed to perform (Figure 9.12 and Figure 9.13). An explicit communication between each nanorobot partner sending the signal is required just when a delivery is completed for the determined organ inlet, whereupon nanorobot *B* awaits a message from nanorobot *A* confirming that *A* has finished the delivery to the given organ inlet. Acoustic communication sensors [143] mounted within the nanorobot hull permit the nanorobot to communicate with its partner whether or not the organ inlet has received the required substance. This permits the nanorobots to maintain the correct delivery sequence of assembled substances into the organ inlets (Figure 9.14).

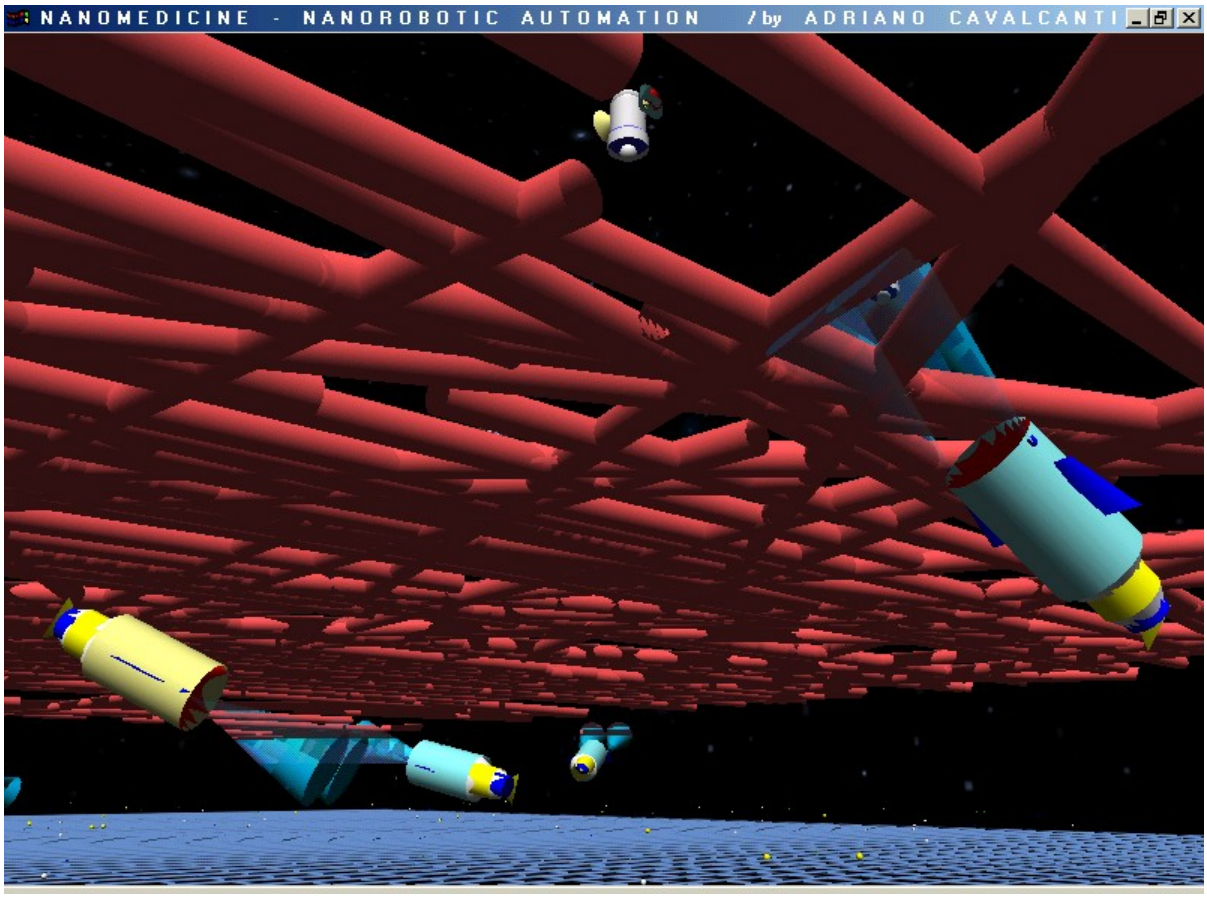

Figure 9.12: Collective scenery, sensing obstacles.

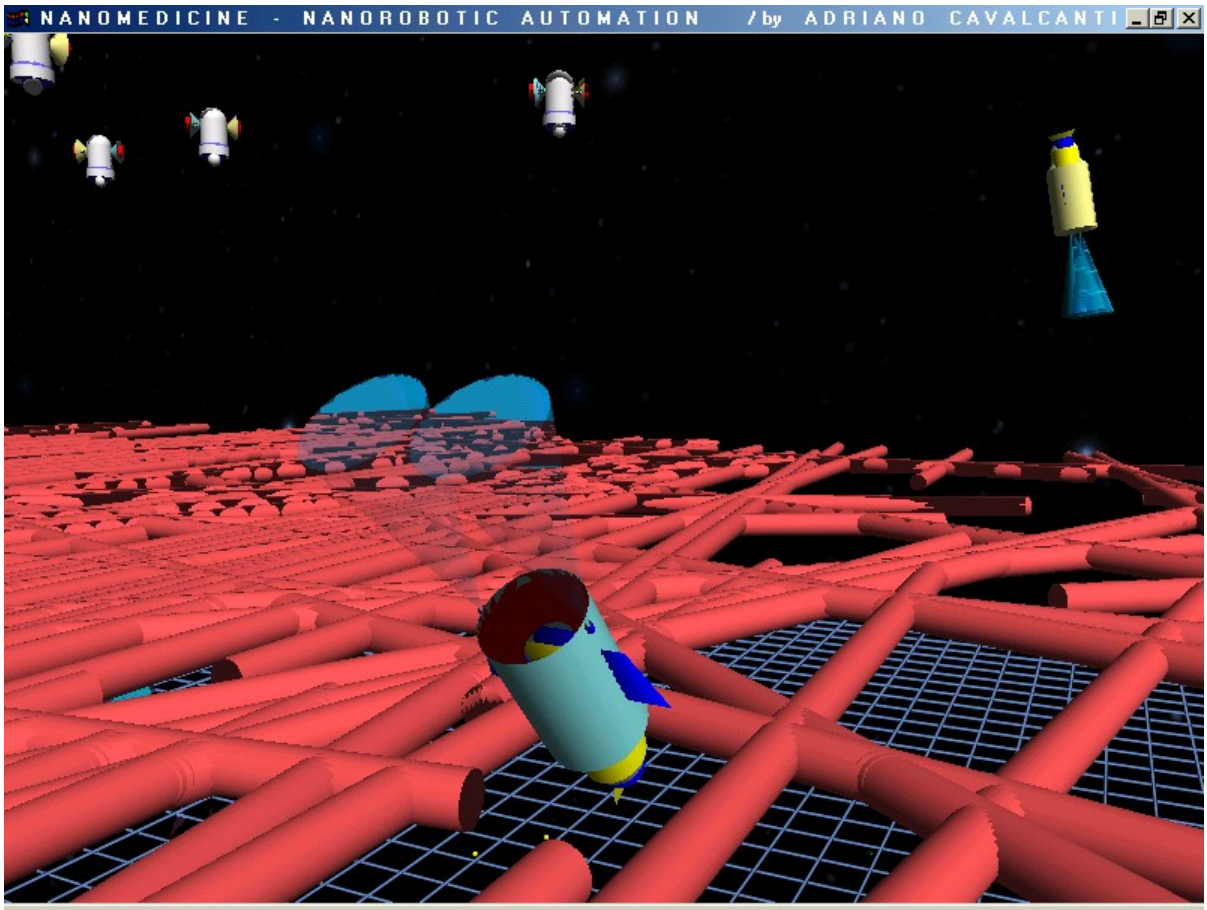

Figure 9.13: Collective scenery, obstacle avoidance.

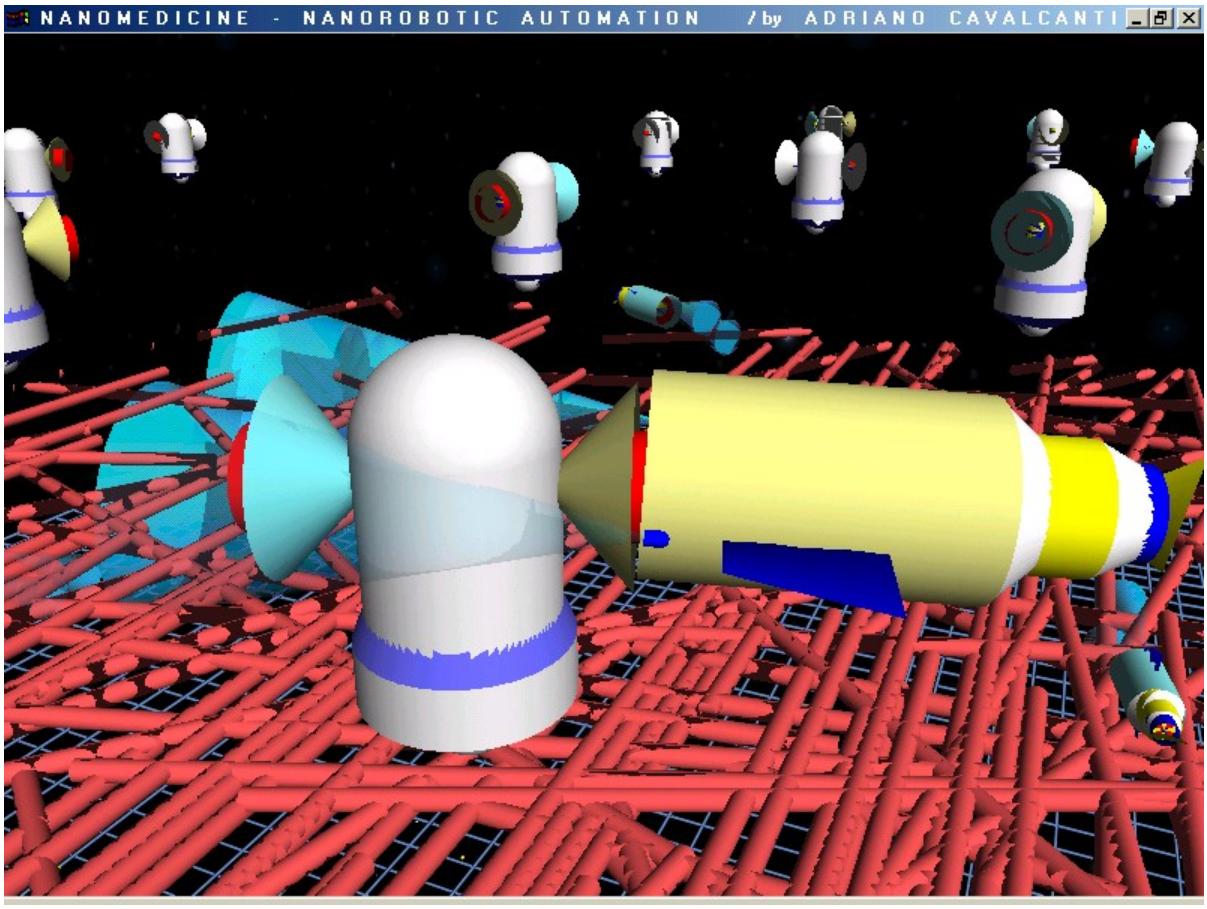

Figure 9.14: Collective scenery, nanorobot molecule delivery.

Step 1:	r_Ω walk randomly to capture β and δ;
Step 2:	if $\sum \beta = \sum \delta \rightarrow$ assemble $f(r_\Omega) = \beta + \delta$;
Step 3:	if $\sum f(r_\Omega) < \min \rightarrow$ repeat step 1;
Step 4:	r_Ω achieve next delivery goal;
Step 5:	if delivery_permition = true →
	delivery: $f(r_\Omega) = f(r_\Omega) - 1$;
Step 6:	if $f(r_\Omega) > 0 \rightarrow$ repeat step 4;

Table 9.8: Collective robotics interaction rule.

By using the nanorobot's local perception as much as possible and by sending the fewest possible messages to other nanorobots, unnecessary communication between the agents is minimized, thus optimizing energy consumption by the nanorobots. Nanorobots satisfy their energy requirements via the chemical combination of oxygen and glucose [142], both of which are plentiful in the human body.

9.4.2 Nanorobots Interaction Rule

The approach for a nanomedicine problem here could be described as two multi-robots teams that must cooperate interactively to feed a set of organ inlets in the virtual environment under study. Research on multi-robot teams working cooperatively to achieve a single global task suggests that we should consider emulating the methods of the social insects [326], because nature is showing us how to build decentralized and distributed systems that are robust and capable of accomplishing tasks through the interaction of agents with the same structures and pre-programmed actions and goals. Kube [225] has pointed out that a careful decomposition of the main problem task into subtasks with action based on local sensor-based perception could generate multi-robot coherent behaviors.
We have decomposed the total set of organ inlets, assigning for each pair of nanorobots a specified number of organ inlets to be attended by the nanorobots at each time-step during the simulation. Each pair is comprised of nanorobots from teams *A* and *B*. The parameter Ψ in the equation 9.4 defines the nanorobot *A* and *B* behaviour, which is designated here as cooperative teams, directly influencing the evolutionary decision control model defined by the equation 8.4.

$$\Psi \quad \{ \quad \Omega = A \vee B \Rightarrow \qquad \Psi = 1; \tag{9.4}$$

The organ inlets selected to be fed at time *t* have to be fed first by the agent *A* and so forth. Both agents must take care to avoid supplying too much or too little of the injected substances. The multi-robot team behaviour interaction rule is described at Table 9.8. The study of mobile multi-robot behaviour in a single global environment is a relatively new field of research [225], which has advanced most of the concepts related to the use of local perception for reactive agents.

9.4.3 Nanorobots Collective Results

The collective nanorobotics performance has achieved the most desirable performance, as we can be observed in Table 9.9. The organ inlets have registered excellent behaviour in all of the 720 nutritional levels observed through a simulation of 24 time-steps, with no one level raging outside the desirable operational levels, i.e. 20% to 80%. Indeed the highest level registered in an organ inlet was a nutritional level of 60%, and on the other side, the lowest level observed was 37% - both indicating a most desirable control performance on the stochastic environment realised by the cooperative nanorobots teams. The delivery mean was equally successful, as detailed in Table 9.10.

In Figure 9.15 we could observe how well tuned was the collective nanorobots cooperative work, demonstrating that a collective approach is an ideal approach for massively parallel actions for nanotechnology automation specifications, where most of the nutritional levels are ranging significantly around the target mark of 50% in relation to the organ inlets relative capacity. We could estimate that the deviation was around 10 points above and 10 points below the considered nutritional target value - see Figure 9.16.

Inlet	time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	delivery	0	1	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1
	level	47	44	48	45	49	53	50	54	51	48	52	56	53	57	48	52	56	53	57	47	51	55	52	56
2	delivery	0	1	0	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	1	0	1	1
	level	45	47	46	45	47	46	48	46	45	47	46	48	44	46	48	44	42	44	46	45	47	43	45	46
3	delivery	0	1	1	0	1	1	1	0	1	1	0	1	1	0	1	1	0	1	1	1	1	0	1	1
	level	48	47	49	45	47	49	48	46	45	47	43	45	47	42	45	47	42	44	46	45	47	46	45	47
4	delivery	1	1	1	1	1	0	1	1	1	1	1	0	0	1	0	1	1	1	0	1	0	1	1	1
	level	51	48	49	51	55	47	48	52	50	54	58	50	48	52	50	47	51	55	50	51	49	47	49	51
5	delivery	1	0	1	1	1	0	1	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1
	level	54	51	47	52	56	45	49	46	42	47	43	48	52	49	37	42	46	43	47	52	48	45	50	54
6	delivery	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
	level	49	55	55	54	49	54	54	48	54	53	53	53	53	47	53	52	52	52	46	52	52	51	51	46
7	delivery	1	1	1	0	1	1	1	0	0	1	0	1	1	1	1	1	1	1	0	1	1	1	1	0
	level	51	50	51	48	49	50	50	49	45	47	44	45	46	45	44	45	45	46	43	44	45	44	45	45
8	delivery	1	1	1	1	0	1	0	1	1	0	1	0	1	1	1	0	1	0	1	0	1	1	1	1
	level	49	50	51	54	52	55	51	52	55	54	53	52	49	51	54	49	50	46	49	47	50	48	49	51
9	delivery	0	0	1	0	1	1	1	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1	0	0
	level	49	47	47	47	48	48	50	48	49	49	47	49	48	48	50	50	51	51	51	53	52	54	54	52
10	delivery	1	1	1	0	1	0	1	1	0	1	1	1	1	1	0	1	0	1	1	1	0	1	1	1
	level	48	51	50	45	48	46	45	48	43	46	48	47	49	52	51	50	49	51	50	52	48	50	53	51
11	delivery	1	1	0	1	0	1	0	1	1	0	1	1	0	1	1	0	1	1	1	1	1	1	1	1
	level	53	57	48	52	49	53	43	47	51	42	46	50	40	44	48	38	42	46	43	47	51	48	52	56
12	delivery	0	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	1	0	1	1
	level	48	47	46	45	47	43	45	46	42	44	46	42	44	46	42	44	45	44	43	45	47	43	45	46
13	delivery	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	0	1	1	0	1	1	1	1	0
	level	48	50	49	45	43	45	48	46	45	47	43	45	47	46	48	47	45	48	46	45	47	49	48	47
14	delivery	1	0	1	0	1	0	1	1	1	0	1	1	0	1	0	1	1	0	1	0	1	0	1	1
	level	48	46	50	48	51	46	49	52	55	53	52	56	54	55	53	50	54	52	53	45	49	47	52	51
15	delivery	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1	1
	level	46	51	47	52	56	53	57	54	58	55	51	48	52	57	53	50	46	51	55	60	48	45	50	54
16	delivery	0	1	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1
	level	49	55	55	54	49	54	54	48	54	53	53	53	53	47	53	52	52	47	52	52	52	51	46	51
17	delivery	1	1	0	1	1	1	1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	0	1	1
	level	51	50	47	48	47	48	50	49	45	47	44	43	44	45	44	45	45	46	47	46	47	44	45	47
18	delivery	1	0	1	1	0	1	0	1	1	0	1	1	1	1	0	1	1	0	1	0	1	0	1	0
	level	48	47	49	48	46	48	47	46	49	48	50	50	52	54	50	52	54	49	51	50	50	49	48	47
19	delivery	1	0	1	0	1	1	1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1	0	0
	level	51	49	49	49	48	48	50	50	47	47	47	47	48	50	48	48	49	49	51	53	52	54	54	52
20	delivery	1	0	1	1	1	1	1	0	1	1	0	1	1	0	1	0	1	1	1	0	1	1	1	0
	level	52	51	50	52	54	53	56	55	54	56	51	54	56	52	54	50	52	54	53	52	51	54	56	51
21	delivery	1	1	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	0	1	1	0
	level	47	51	41	45	49	46	50	47	44	48	39	43	47	51	48	52	56	46	50	54	51	48	52	43
22	delivery	1	1	1	0	1	1	1	1	0	1	1	1	0	1	1	0	1	0	1	0	1	1	0	1
	level	51	50	52	51	50	52	53	55	54	53	55	57	53	54	56	52	54	53	52	51	50	52	50	49
23	delivery	1	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	1	0	1	1	1	0	1	1
	level	51	50	52	48	50	49	48	46	45	44	46	48	47	49	48	50	49	48	50	52	51	49	51	54
24	delivery	1	1	1	1	0	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	1	0	0
	level	52	51	55	57	52	53	51	55	53	53	56	54	51	49	54	57	52	55	54	51	54	57	55	48
25	delivery	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	1	1	1	0	1	0	1	1	1
	level	54	51	55	52	48	53	49	54	50	47	51	56	60	49	45	50	54	59	47	52	40	45	50	54
26	delivery	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1
	level	49	49	49	54	54	54	48	54	54	53	53	48	53	53	53	52	47	52	52	52	52	51	46	51
27	delivery	0	0	0	1	1	0	1	1	0	1	1	0	1	1	1	1	1	0	1	1	0	1	1	1
	level	49	48	47	46	47	46	45	46	45	45	46	43	44	45	44	45	47	46	45	46	45	46	45	45
28	delivery	0	0	0	1	1	0	1	0	0	1	1	1	0	1	1	1	0	1	0	1	1	1	0	0
	level	48	47	46	45	46	45	47	46	44	44	47	47	42	44	44	47	45	46	44	43	44	47	46	45
29	delivery	0	0	0	1	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	1	0	0
	level	49	49	49	49	48	48	50	50	49	49	49	51	48	50	50	52	49	51	51	51	52	54	52	52
30	delivery	0	0	0	1	1	0	1	0	0	1	1	1	0	1	1	1	0	1	1	0	1	1	0	0
	level	48	47	46	45	48	46	49	48	47	46	48	50	49	52	54	56	52	54	57	52	55	57	53	51

Table 9.9: Collective robotics scenery with organ inlets' nutritional levels.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Delivery goal	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300	300
Lowest level	45	44	41	45	43	43	43	46	42	42	39	42	40	42	37	38	42	43	43	43	40	43	45	43
Highest level	54	57	55	57	56	55	57	55	58	56	58	57	60	57	56	57	56	59	57	60	55	57	56	56

Table 9.10: Collective robotics scenery with highest and lowest levels.

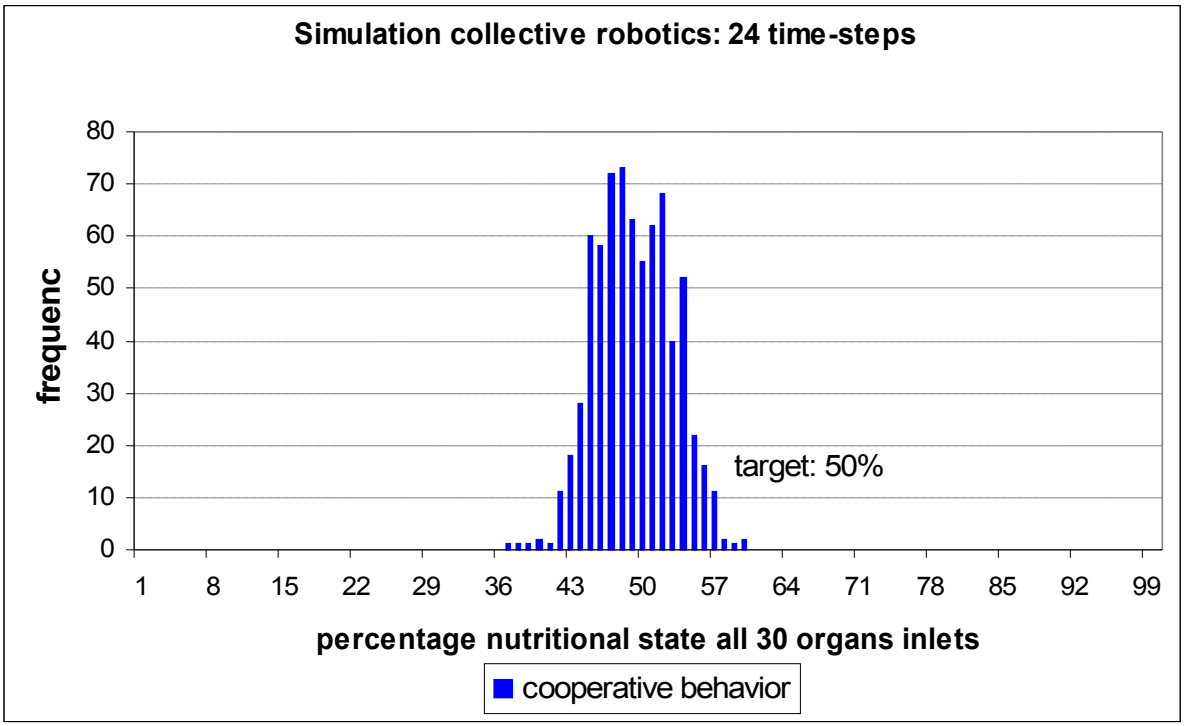

Figure 9.15: Histogram of collective robotics scenery with organ inlets' nutritional levels.

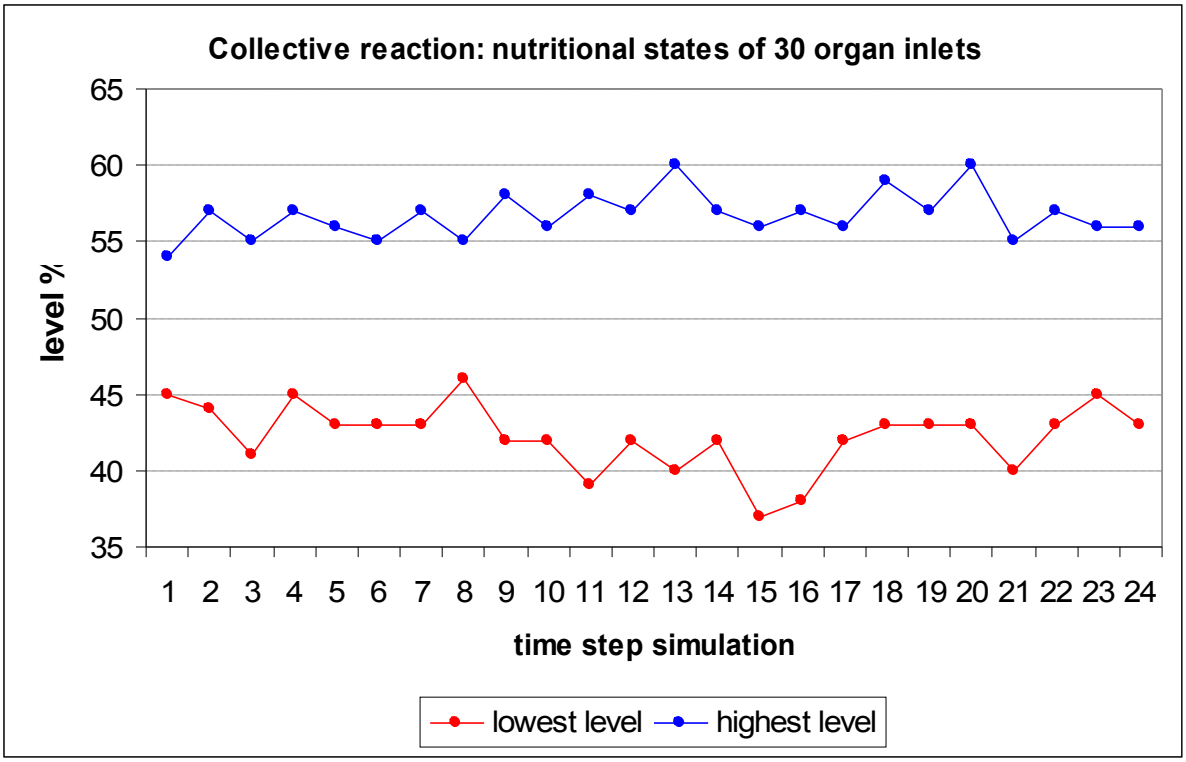

Figure 9.16: Upper and lower organ inlets' nutritional levels for collective robotics scenery.

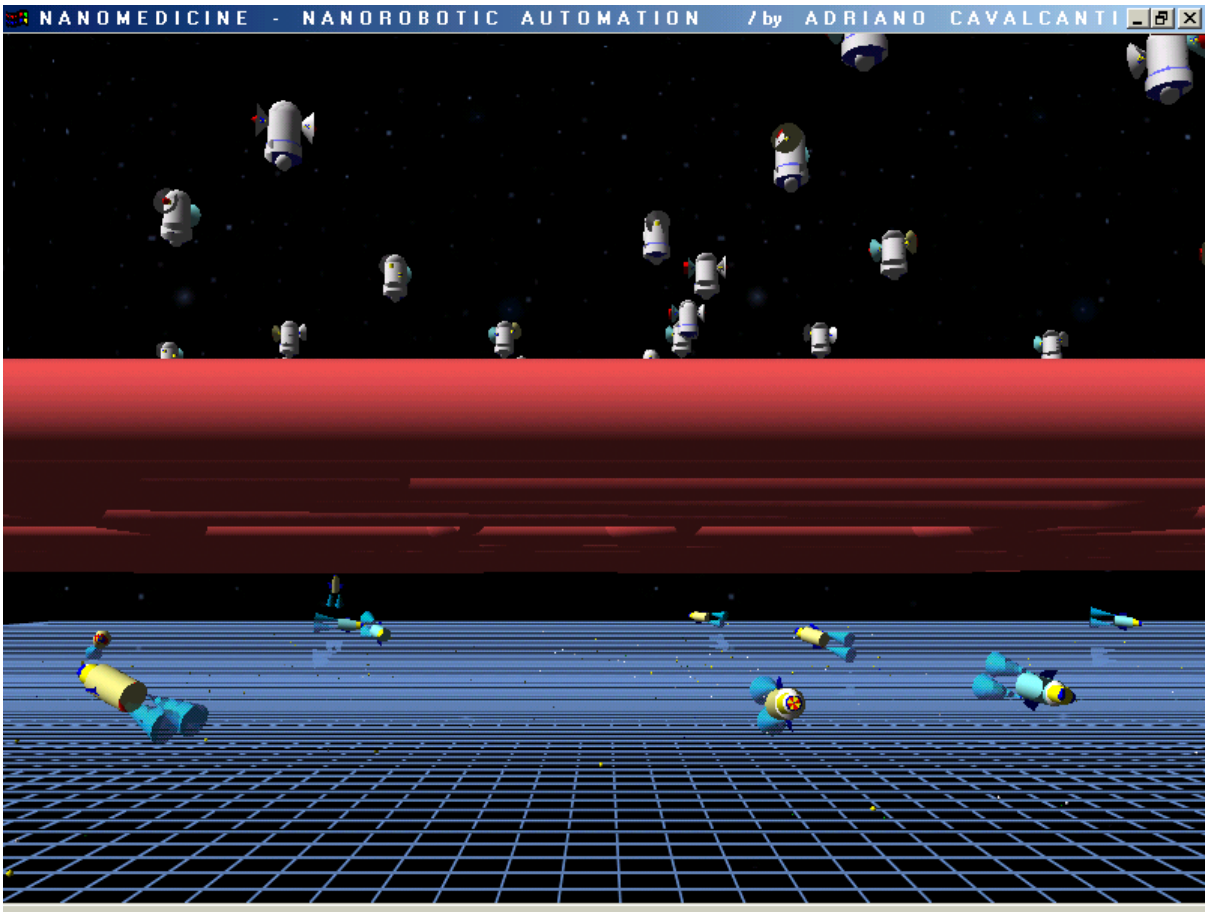

Figure 9.17: Collective team robust behaviour.

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
Delivery goal	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600	600
Lowest level	40	43	41	38	42	43	38	41	42	42	39	42	40	41	42	38	42	43	43	41	40	42	43	42
Highest level	54	57	55	60	56	56	57	55	58	57	59	56	55	57	55	56	58	59	57	55	58	57	58	56

Table 9.11: Collective robotics robustness with highest and lowest levels.

9.4.4 Nanorobots Collective Control Robustness

We have simulated a larger scenery with a total of 12 nanorobots shared equally into 2 teams, which were designated respectively nanorobot teams *A* and *B*, to evaluate the robustness of collective nanorobotics (Figure 9.17). The environment was comprised of 60 organ inlets, and *n* obstacles positioned randomly. Thus nanorobot *A* must cooperate with nanorobot *B* forming a kind of nanorobot couple, which has to carry out the nutritional control of 10 organ inlets that were pre-assigned to each one. The quantity to be delivered by the nanorobots increases to 600 nutrients by each time-step on the simulation. They have obviously to decide what organ inlet to attend immediately and what organ inlet to set for the next time-step in the dynamic environment. The desired delivery mean was always successfully attended by the nanorobot teams as observed by the *delivery goal* in Table 9.11. In the same table it can be seen, the lowest organ inlet's level observed was 38% and the highest level was 60%, which are satisfactory values ranging within the considered range of 20% to 80%, indicating no insufficiency or overdoses.

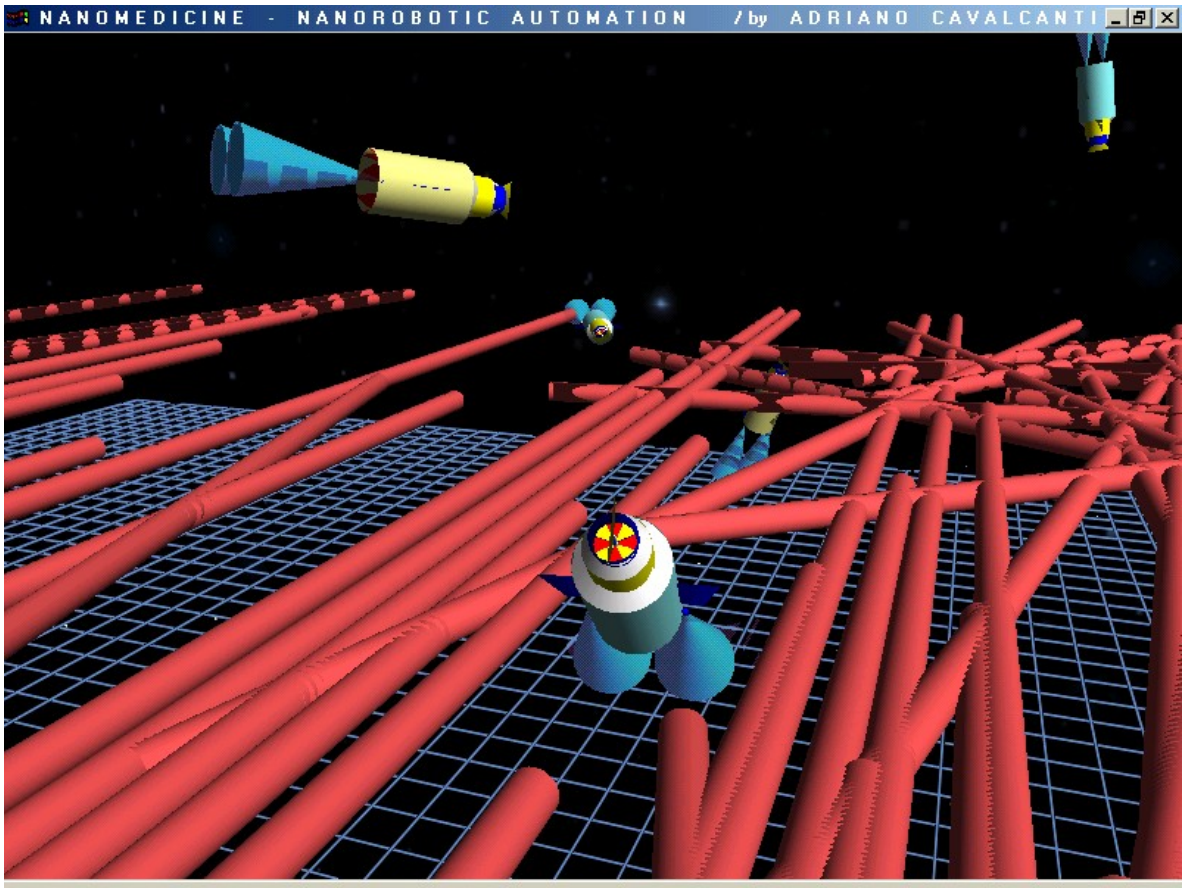

Figure 9.18: Nanorobot goes back to search and capture more molecules.

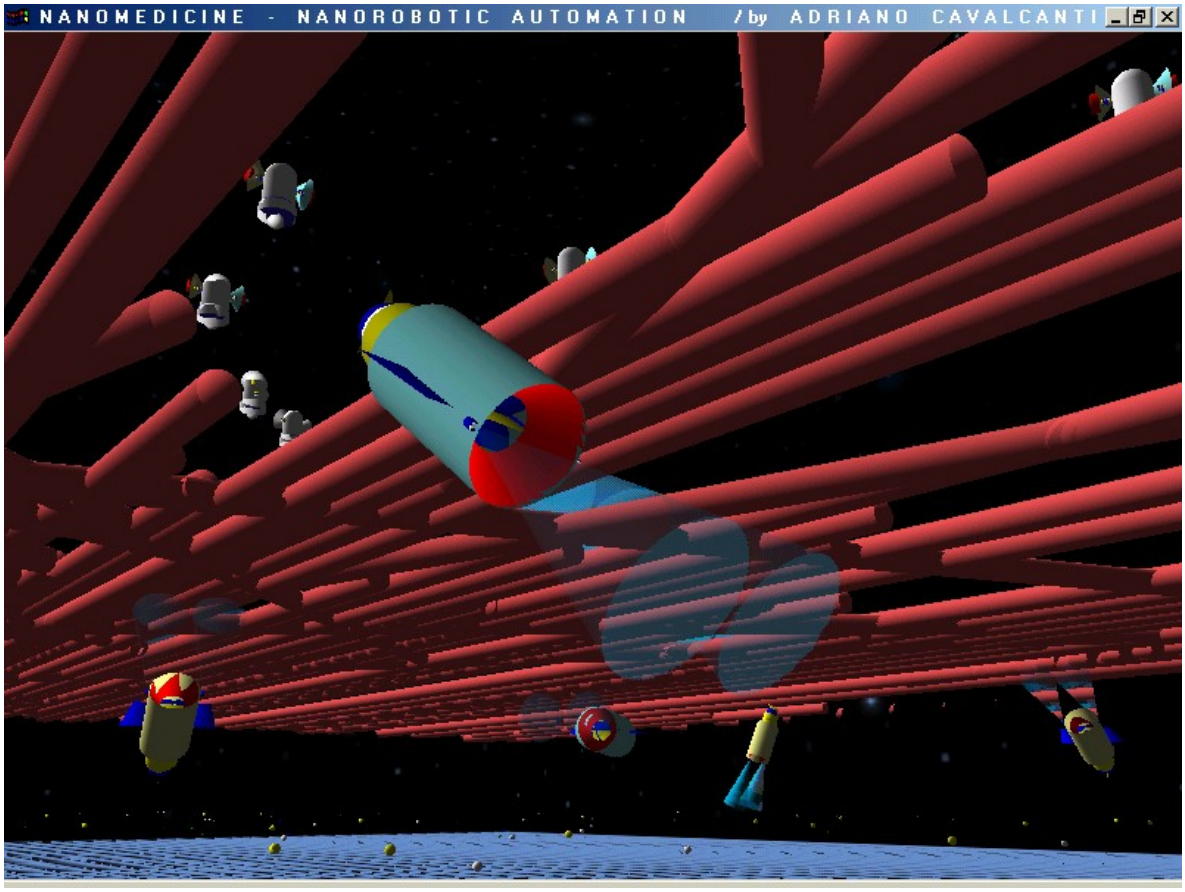

Figure 9.19: Nanorobot avoiding collision to attend delivery goal assembling more nutrients.

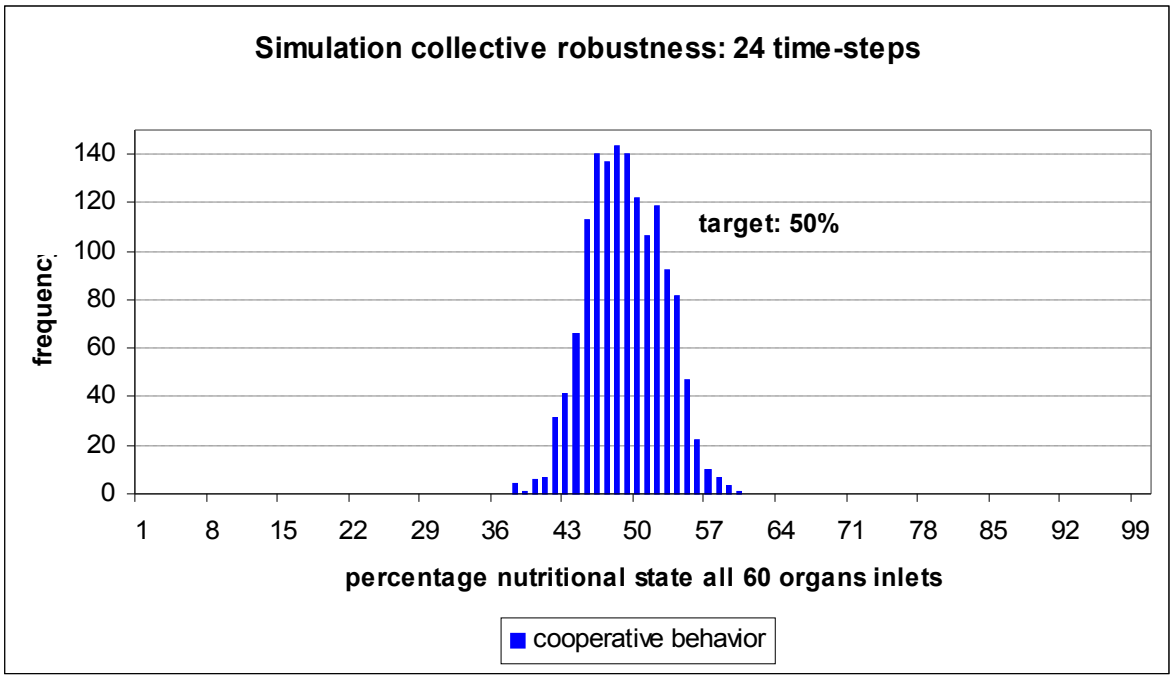

Figure 9.20: Histogram of collective robotics robustness with organ inlets' nutritional levels.

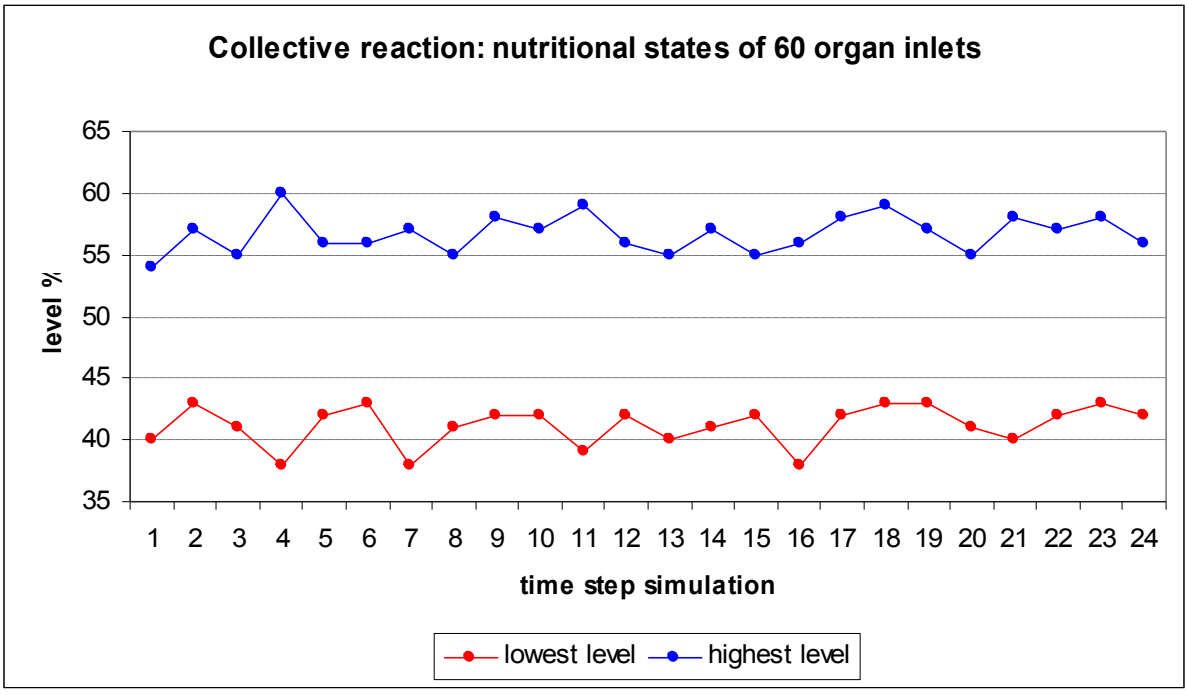

Figure 9.21: Upper and lower organ inlets' nutritional levels for collective robotics robustness.

After performing the tasks related to nutrient delivery and organ inlet's levels verification, the nanorobot navigates through the 3D environment to capture more molecules attending the *delivery goal* (Figures 9.18 and 9.19).

At the same time the control behaviour registered here is quite similar to the values observed in Table 9.10, where the lowest value observed was 37% and the highest 60%. Such performance similarity between the instances with 60 organ inlets and the prior one with 30 organ inlets could be

explained, once we have the collective nanorobotics scenery for both instances and the capability to proportionately increase the number of nanorobots with the increase in problem complexity.

We could observe in Figure 9.20 that the control model has achieved a similar behaviour, in the sense that most of the values have ranged in the same intervals registered by the scenery with the 30 organ inlets. Thus we could affirm that the model has the robustness required with coherent behaviour to a desirable massive assembly automation for nanotechnology. Comparing Figure 9.21 and Figure 9.16 we could clearly observe that the model has demonstrated its stability in the sense that the upper and lower registered nutritional levels maintain basically the same satisfactory performance features.

9.5 Neural Motion Results

The nanorobot required a motion control model based on either of two main aspects: optimization of the trajectory distance, and real time analyses for a required trajectory which enables the delivery of assembled biomolecules with the avoidance of obstacles. Coherent behaviour in complex real-world systems requires accurate and timely reactions to environmental events. The use of Artificial Neural Networks appears to be a suitable approach for nanorobot motion analysis in a 6-degrees-of-freedom environment [71]. The reactive layer is comprised of a low-level approach which is required to interact with a continuous time complex environment [16]. Overall, this design results in a flexible, hard real-time execution system that exhibits graceful performance.

Considering that the motion problem of the core of our discussion is comprised of a dynamic sub-set of sequenced trajectories, we have expressed the solution as the cost minimization of a *complete trajectory*. This *complete trajectory* is comprised of a first route, that we nominate as the *delivery route*, representing the organ inlets that were set-up to be attended in the present simulation time, and the second route, which we could identify as the *verification route*, being respectively the set of organ inlets not selected for any delivery in the current evolutionary dynamic decision. The *complete trajectory* is necessary in order to provide important information for the evolutionary decision making for an effective action in the next time-step dynamic scenery.

As we have discussed in Item 8.5 from the previous chapter, each nanorobot visits in a shorter time the organ inlets that were pre-attributed to that nanorobot in order to gather information for the next time-step decision from the 3D workspace. Thus for a larger workspace, the nanorobot *r* does not visit all the organ inlets comprising the environment, but only the ones included into the *r*'s *complete trajectory*, i.e. the attribution that *r* has to supervise.

The architecture and model implemented comprise concepts derived from real time systems [337] in the sense that it is a controller that should meet appropriately its behaviour in response to changes in the dynamics of the process and the nature of the disturbances [15], such as uncertain obstacles in the environment that would require an adaptive motion in regard to unpredictable events and other agents, therefore avoiding collisions, to successfully achieve a pre-established set of dynamic goals.

Table 9.12 shows the sequential optimization performed by our neural networks to complete the delivery route, where the multi-layer was instructed to minimize the layer energy, thus optimizing the trajectory to be performed by our nanorobot. The best trajectory is achieved at the 11th sequence, with a cost of 77040 nm for the trajectory related to visiting the organ inlets, which must be supplied at present time-step simulation. The results in Table 9.12 show a cost optimization of 37%.

A similar performance could be observed in Table 9.13 about the verification route, where the trajectory is related to the organ inlets that the nanoborot will visit, in order to verify their nutritional levels. Here the optimization has achieved 36% for the trajectory cost minimization.

Trajectory neural optimization – organs inlet „ON“																			cost
sequence 1	6	5	8	16	3	2	13	7	4	9	10	19	12	15	11	20	1	17	122240
sequence 2	9	15	13	1	7	12	5	11	3	17	19	10	4	8	6	20	16	2	106040
sequence 3	2	3	12	4	13	16	8	10	19	11	15	9	1	17	5	7	6	20	101880
sequence 4	10	9	4	2	7	3	5	13	12	17	15	6	11	19	20	8	16	1	99560
sequence 5	7	10	8	11	6	13	9	19	2	20	4	15	16	1	12	3	5	17	94720
sequence 6	5	3	9	15	4	7	20	13	12	2	1	8	10	11	16	6	19	17	94280
sequence 7	11	10	4	6	8	13	19	7	12	16	1	2	15	5	9	3	20	17	91440
sequence 8	6	10	11	4	13	15	9	17	2	5	16	1	7	12	20	3	19	8	87240
sequence 9	19	12	1	2	20	10	8	4	7	11	9	13	6	3	17	5	16	15	85960
sequence 10	8	19	10	11	6	13	17	5	2	16	20	4	7	12	1	3	15	9	80720
sequence 11	3	5	17	15	9	16	2	7	4	1	20	13	19	6	10	8	11	12	77040

Table 9.12: Neural motion optimization for delivery route - distance cost in nm.

Trajectory neural optimization – organs inlet „OFF“													Cost
sequence 1	28	29	18	0	24	21	14	22	26	27	23	25	76400
sequence 2	27	22	28	21	14	0	23	24	26	25	29	18	74440
sequence 3	21	14	28	0	18	27	25	22	29	24	26	23	71240
sequence 4	0	14	23	24	25	18	29	21	27	28	26	22	70040
sequence 5	25	28	18	0	14	21	23	27	26	22	29	24	61320
sequence 6	14	28	18	27	26	29	24	25	22	21	0	23	61120
sequence 7	23	14	21	27	26	18	24	29	25	22	28	0	58640
sequence 8	23	14	0	21	22	29	28	24	25	18	27	26	55520
sequence 9	29	23	0	18	28	24	25	26	27	22	14	21	51840
sequence 10	21	14	0	28	22	27	26	25	18	29	24	23	51520
sequence 11	21	22	26	27	25	18	29	24	28	23	0	14	49960
sequence 12	23	21	14	0	29	18	27	26	25	24	28	22	49040

Table 9.13: Neural motion optimization for verification route - distance cost in nm.

Complete trajectory sequence																													Cost
3	5	17	15	9	16	2	7	4	1	20	13	19	6	10	8	11	12	22	28	24	25	26	27	18	29	0	14	21	23	120360

Table 9.14: Neural motion optimization with complete trajectory - distance cost in nm.

Once the routes to be taken by the nanorobots for supply of organ inlets are verified, we join both trajectories considering the best connection for the verification route with the last point on the delivery route, i.e. the verification route could be set in forward or backward sequence, depending on the nearest position between the last organ inlet in the delivery route and the first or last organ inlet in the verification route sequence. In the case shown in the Table 9.14, the best sequence was to connect organ inlet 12 with organ inlet 22, instead of 12-23 sequence, which has resulted in achieving a *complete trajectory* with the lowest cost. Thus for this case we have a verification route in backward sequence.

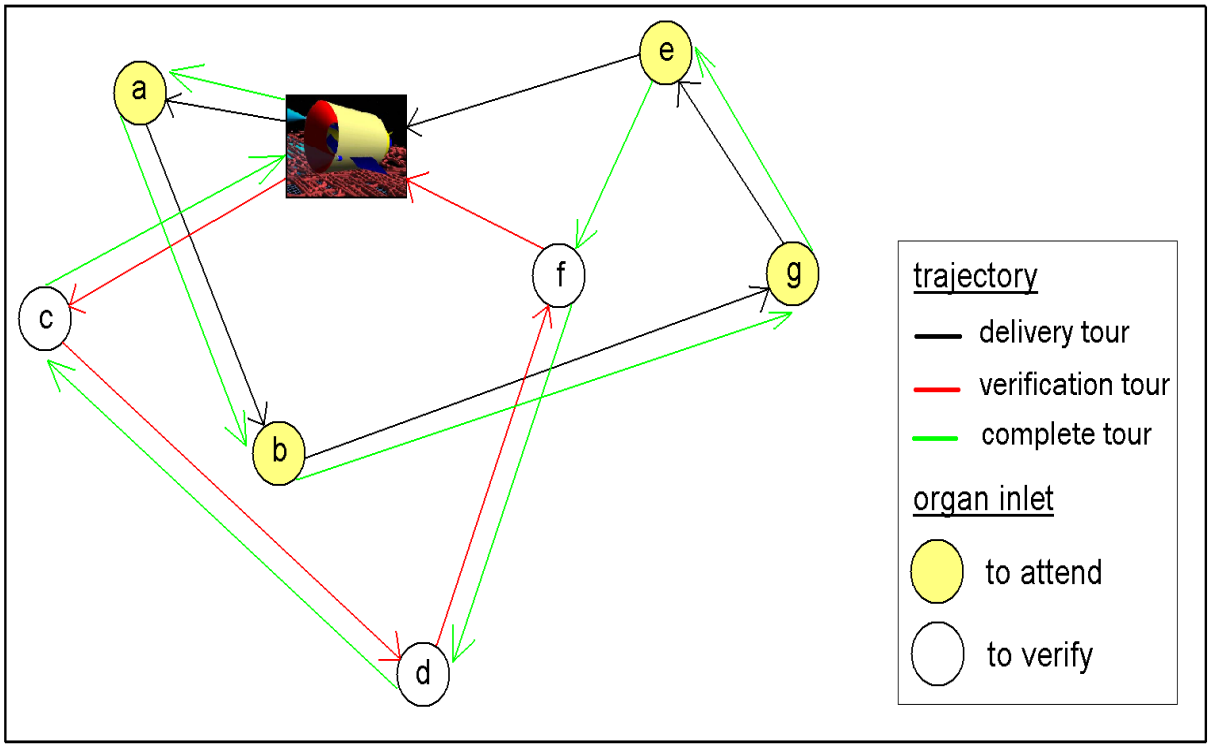

Figure 9.22: Complete trajectory comprised by delivery tour and verification tour.

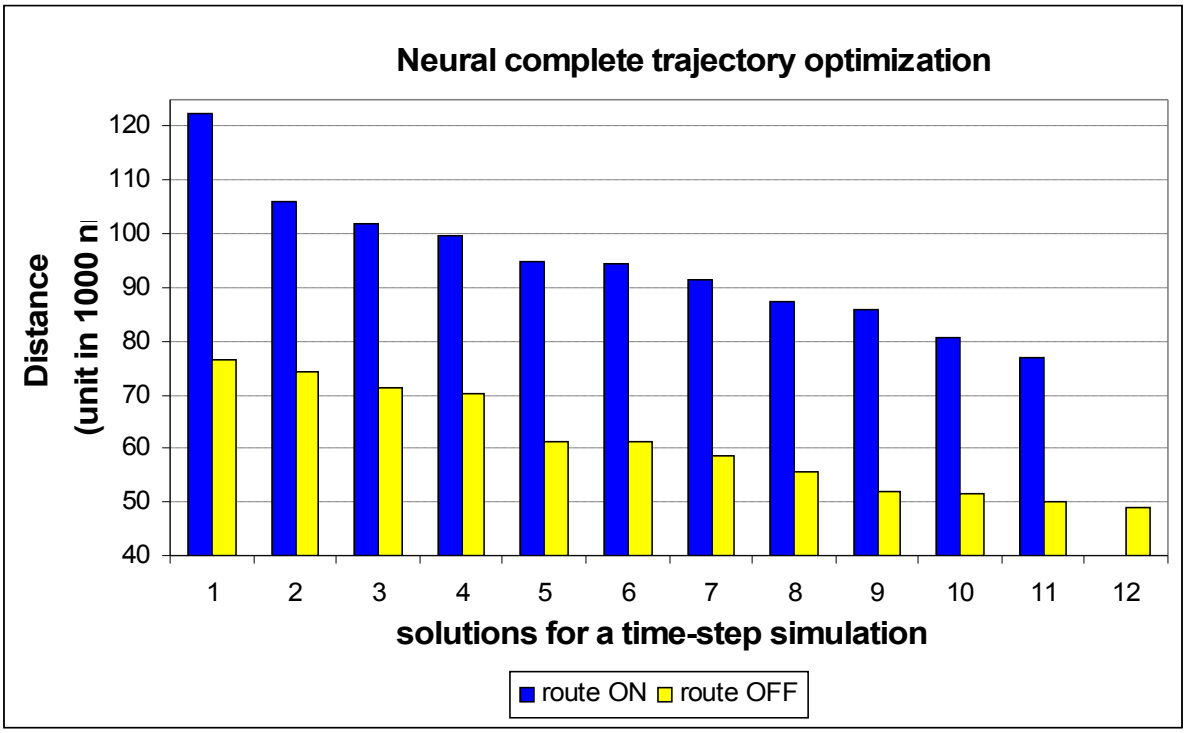

Figure 9.23: Neural motion cost minimization.

Figure 9.22 shows an illustrative representation of the parallel trajectory process that the nanorobot receives by the neural control module to improve its performance. As could be observed the *complete trajectory* has a smaller number of connected directed edges than the total sum of the directed edges connecting the *delivery route* and the total sum of the directed edges connecting the *verification route* together, which implies a lower cost for the *complete trajectory* as a whole trajectory, rather than the separated sum of *delivery route* and *verification route* costs.

In Figure 9.23 we see the *delivery route* and the *verification route* evaluation with its trajectory optimization, which represents the sequenced cost minimization observed in Tables 9.12 and 9.13. The neural motion control has achieved suitable results with a low processing requirement, providing shortest-path values ~ 37% better than a *greedy* solution [77] for the route distance minimization problem. A similar performance for the motion control problem has been observed for the different scenarios under study [72][69][66].

9.6 Conclusion

This chapter has demonstrated through its numerical results that a promising field of research of increasing interest such as nanorobotics automation could be specified and designed in more detail with the use of techniques derived from control theory using stochastic and flexible algorithms, such as neural networks and evolutionary programming. With reference to the intrinsic problem related to the study of nano-worlds and the question directly influenced by the observation of nanoscale events, the use of virtual reality and computer graphics could be shown as an essential tool for a better insight into phenomena related to the emerging field of nanotechnology, thus enabling an easier elaboration of new concepts considering complex and uncertain environments, through better observation.

On one side, the competitive nanorobotics scenery is a practical approach to evaluate the stability of the robust behaviour of the nanorobot for the adaptive control model under study. Thus the competitive environment has been shown as a powerful tool to verify in a detailed fashion the model's coherent performance attending a large range of aspects inherent to nano-worlds, such as dealing with uncertainty and stochastic events. Meanwhile on the other side, the collective nanorobotics approach could be observed equally as a desirable way to lead with a high complex set of assembly tasks intended to achieve a massive assembly automation, where a number of agents could work cooperatively, through a well shared set of tasks and an equal pre-programmed set of actions, and achieve a greater performance over a stochastic and complex environment. Thus the analyses and results in this chapter indicate that the approaches described might also be a promising systems design for assembly automation in nanotechnology.

Chapter 10

Conclusion

10.1 Perspective

The general purpose throughout this dissertation has been to provide a feasible design approach for the most important aspects directly related to the fast development of nanotechnology with an application to medicine. Thus we have centred our discussions on the study of possible automation models and tools to follow up control analyses focusing on the theory of the development of molecular machines capable of working and accomplishing successfully a set of pre-programmed task in a stochastic environment. One of the aims of the presented work was also to serve as a framework for the new stage of future nanotechnology in the sense of nanosystems automation, which is considered by the research community to be one of the most important paradigms to achieve feasible nanoassemblers.

The emerging field of nanotechnology has brought new challenges and possibilities never thought of before. We are going to see in coming years exciting achievements, with industries and governments in a mature joint partnership, each doing their part in this endeavour, making significant investments for a worthwhile effort. Obviously such important investments reflect just a small part of all revenues expected by the same government and private initiatives. Although the first steps for this molecular manufacturing in the sense of building blocks, has come with positive results in the 80's and 90's. Now we are faced with a more complex duty that is the next generation of nanotechnology advances, in the sense of building nanobioelectronics and molecular machines, which are expected to achieve the most important aspects of the expanding nanotechnology development.

When we take note of some important aspects of a large nanotechnology development, there is general agreement about the necessity of new approaches to conquer a higher level of automation for molecular manipulation comprising many uncertaint aspects related to quantum mechanics calculations inherent in the nano-world. In such aspects the key technology is new devices and theories to explore and automate such environments.

As a practical approach we have examined many of the coherent behaviours based on nanorobotic performances using virtual reality as the most actual and feasible way for a detailed exploration of nano-worlds. Working in a complex 3D environment, a set of tasks was presented to a

group of nanorobots, specifically for the task of automation assembly, to find and capture molecules, avoid collision with any kind of obstacles, molecules delivery for a pre-defined set of organ inlets, and dynamic decision making based on the nanorobot's local perception.

10.2 Dissertation Role

The main aspects required for a new nanorobot control design paradigm have been considered throughout the discussions and experimentation in this dissertation. In Chapters 1 and 2 we have an overview of the main aspects of the emerging new field of nanotechnology, pointing out the importance of new paradigms for nanoassembly automation, as well as the important role that computer graphics could play in the actual stage of chemical and mechanical experimentation.

In Chapter 3 we have discussed the main aspects of physically based simulation, and its application for the study of dynamic virtual environments, specifically the use of bounding box volume with the *2D intersection tests* as the most practical approach for collision detection in robotics applications. In Chapter 4 we have described the main mathematical considerations about motion control, and its importance in relation to mobile robotics design.

In Chapters 5 and 6 we have described the algorithms that we have chosen to compose the nanorobot "brain", in the sense of motion control and dynamic decisions, where we have respectively discussed the main aspects related to neural networks and to evolutionary programming. In Chapter 7 we have presented important issues related to parallel processing, which is an important aspect required to integrate the whole simulator and nanorobot architecture with nanorobot respective functional architecture and distinct parallel modules.

Finally, we have presented in detail the control model and nanorobot design in Chapter 8, where we pointed out the importance of adaptive characteristics inherent in a nanorobot model that must survive in a dynamic stochastic environment. Obviously the computer graphics were a valuable tool for nanorobot design, in the sense that virtual reality has enabled an easier and practical prototyping than any other imaginable approach. Going forward in the dissertation we have achieved several conclusions with the numerical results obtained through the graphic simulator, which was discussed in detail in Chapter 9. Next we highlight the main aspects obtained from the development of proposed designs analysing the model's performance according to the dissertation achievements.

10.3 Research Achievements

The concepts developed in this dissertation have provided a useful characterisation of the many aspects related to the design of control systems for the development of molecular machines. Thus a successful design methodology for nanorobotic evolutionary development decision and motion control comprised of functional parallel modules was implemented. We have postulated two main paths to investigate the dynamics of robust nanorobotic control: the first was the study of competitive nanrobotics scenery, and the second was the collective nanorobotics scenery. Both scenarios utilising evolutionary techniques for the dynamic decision problem, and neural networks for the motion control problem, have their worth. In the following sequence we discuss the model performance for each scenario investigated.

10.3.1 Competitive Evolutionary Behaviour

The competitive scenery was demonstrated as a suitable approach for the aim of verification of model adaptability, once it was possible to observe that the model could react in a satisfactory fashion against an adversary as skilful as our nanorobot acting upon the environment, thus requiring from a nanorobot decision model the most fitting behaviour considering the information gathered from the environment through sensor-based local perception.

A starting point for our model validation was a scenery where the nanorobot has acted upon 30 organ inlets, which were experiencing, at the same time, the counter action of a nanorobot adversary with the same capabilities inherent in the nanorobot agent. Furthermore we have included more complexity in this scenery, in the sense of verifying the decision model robustness through an adaptive behaviour within a greater problem instance, and for such model validation we have observed the nanorobot performance for an environment with 60 organ inlets, where in a similar competitive environment the nanorobot has also achieved satisfactory control accross a greater number of organ inlets.

In both instances the nanorobot decision model has reacted with a real time response attending successfully to all problem constraints that were considered, which provided accurate proof of the stability and adaptability of the model, once its satisfactory behaviour was demonstrated in a reactive stochastic environment.

10.3.2 Collective Evolutionary Behaviour

Once we have verified that the model could adapt in a circumstance where competitive agents operate in the same environment, we have addressed a second scenery where our experiment with the nanorobots has converged to show a collective function. Therefore the concept of collective robotics was adopted as an effective coordinated action approach for a massive nanoassembler manipulation.

We observed that the collective behaviour of the nanorobot has achieved a collaborative and well coordinated performance, once all nanorobots have the same behaviour characteristics and share the same set of targets, which were equally pre-programmed. Thereby, they share a common goal acting upon the same set of objects (organ inlets) in a dynamic environment. For a more successful accomplishment of the task, we have shared subsets of organ inlets with the same number of organ inlets and attributed a subset to each nanorobot, with the intention of a well coordinated shared effort, which could be observed as an appropriate approach especially if it was considered that the nanorobots' interaction with their surrounding world was based on a local perception approach.

The main aspects expected to be controlled in this environment were attended successfully by collective nanorobotics modelling, where the evolutionary module also has demonstrated a tuned performance for a multinanorobots workspace, where through time the organ inlets' nutritional levels have remained in the desirable ranges of 20% to 80%, conserving most of the values around the target level of 50%. Thereby we could affirm that the collective robotics approach seems to be a promising approach to achieve massive nanoassembler automation.

10.3.3 Neural Motion Performance

The neural motion control was successfully used in two sceneries with two different levels of complexity with real time responses for the circumstance where the nanorobots should go around capturing molecules and visiting a pre-defined set of delivery points, avoiding random obstacles,

collision with other mobile nanorobots, and trying at the same time to minimize the time required to accomplish such tasks. These requirements have been fulfilled by the neural networks approach, where the nanorobots have completed their trajectories with a cost minimization of a mean value of 37%, which shows an impressive improvement in comparison with a *greedy* solution for motion control optimization. A positive aspect of feedforward neural networks is its suitable application requiring low computational effort for NP-hard and NP-Complete problems, which is the case for motion in 6-degrees-of-freedom.

10.4 Main Contribution

The approach presented through this thesis is a practical technique for investigating the behaviour of nanorobots. Including aspects of the physical environment, in conjunction with graphical visualization, provides a feasible methodology for automation and control design in nanotechnology. Unlike some prior simulators for simple robots, the simulator presented here does not assume robots are restricted to a fixed grid or behave as simple cellular automata with very simple environments. Our simulator differs also from most of those used with larger robots, e.g., for operating in office environments or for robot soccer, by its focus on viscous forces and emphasis on motion in three dimensions.

Our work has proposed a new paradigm for the challenging issue related to the development of control models for nanorobotics automation with biomedical applications. Thus this work has discussed a nanorobotics control investigation focusing on the aspects of a robust and adaptive model for the nanoassembly manipulation problem. The dissertation has presented an innovative contribution and investigation of the central aspects influencing the possible approaches and design of control for the development of molecular machine systems.

Altogether, the dissertation has brought the reader's attention to essential aspects related to molecular assembly manipulation on behalf of nanorobotics and nanoassembly automation as one of the most important questions for the fast development of nanotechnology.

10.5 Conclusions and Future Works

Computational Nanotechnology is a new research branch originated from Computer Graphics and Simulation, that provides an extremely useful and important enabling approach to make feasible many of the chemical, physical, and kinetic analyses for theoretical and practical investigation of nano-worlds. Many intriguing aspects of nanomanipulation and automation are still open and awaiting pioneers that are willing to develop work on a very exciting emerging field with plenty of new possibilities to be discovered. In such contest, the use of advanced 3D Computer Aided Design systems for interactive visualization are expected to play an important role for the further development in scientific research for the coming years.

Nanorobots monitoring nutrient concentrations in a three dimensional workspace is a possible application of nanorobots to medicine and other biomedical problems. Ongoing developments in hardware and with the use of distributed processing could also allow increasingly the number of nanorobots or the level of detail to improve investigations for many medical questions (Figure 10.1). Future work with more detailed simulator versions could then provide a more specific evaluation for particular cases.

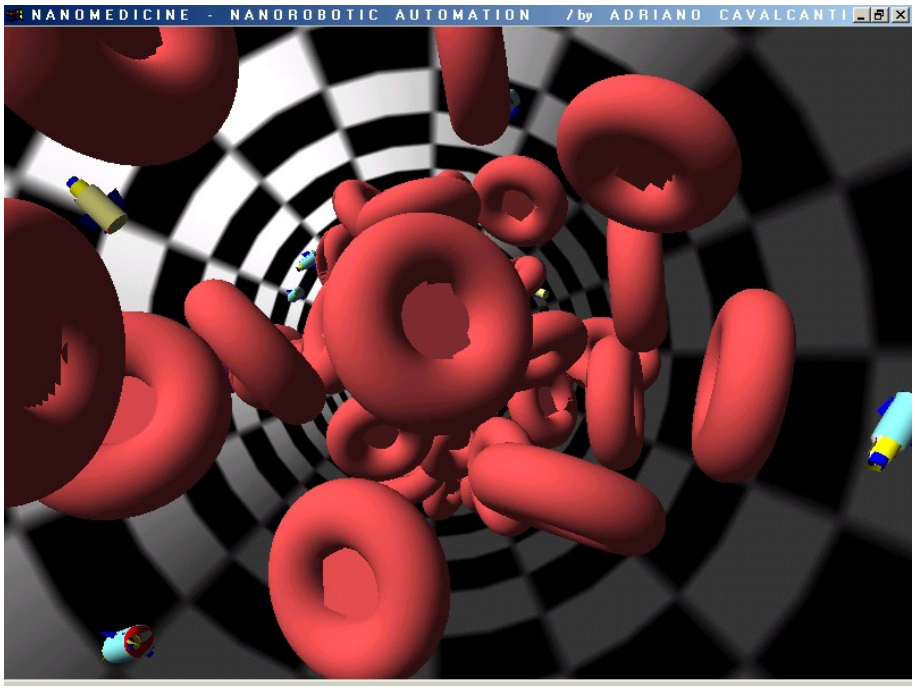

Figure 10.1: Red blood cells and nanorobots inside a textured vessel wall.

Nanorobots supervising the human body through delivering assistance to organs that require some kind of repair is also a possible application of nanorobots in medicine, among other biomedical problems. Another interesting nanorobotic application is its use to process specific chemical reactions in the human body as ancillary devices for injured organs. Nanorobots equipped with nanosensors could be used to detect glucose demand in diabetes patients. Nanorobots could also be applied in chemotherapy to combat cancer through superior chemical dosage administration, and a similar approach could be taken to enable nanorobots to deliver anti-HIV drugs. New works applying nanorobots to coronary problems (Figure 10.2) is also a very interesting possibility.

A set of nanorobots could be used as well to link together and form larger structures. For instance, as an initial response to tears in the vessel wall. In this case, the controls based on chemical signals described here readily extend to artificial “scents” passed among nanorobots via direct contact as applied to modular robots to form a variety of structures. Effective grouping of modular nanorobots is a difficult control problem requiring recognition of other nanorobots and requiring independent locomotion capabilities beyond those usually considered for modular robotics (where movement often requires prior physical contact with other modules). While originally proposed for large modular robots, these techniques readily apply to nanorobots since they do not require a centralized planner with extensive computational capabilities and knowledge of the environment.

More generally, nanorobots could be of different types, some could act as scouts with a wide range of sensors, others to provide additional power, and others with large supplies of signalling molecules or broader communication capabilities. Such heterogeneous teams face additional control issues of resource allocation, for which market-based control should be well-suited due to the large numbers of nanorobots. In our work, we addressed collective behaviour through effective communication for homogeneous nanorobots, hence all have the same control program and capabilities.

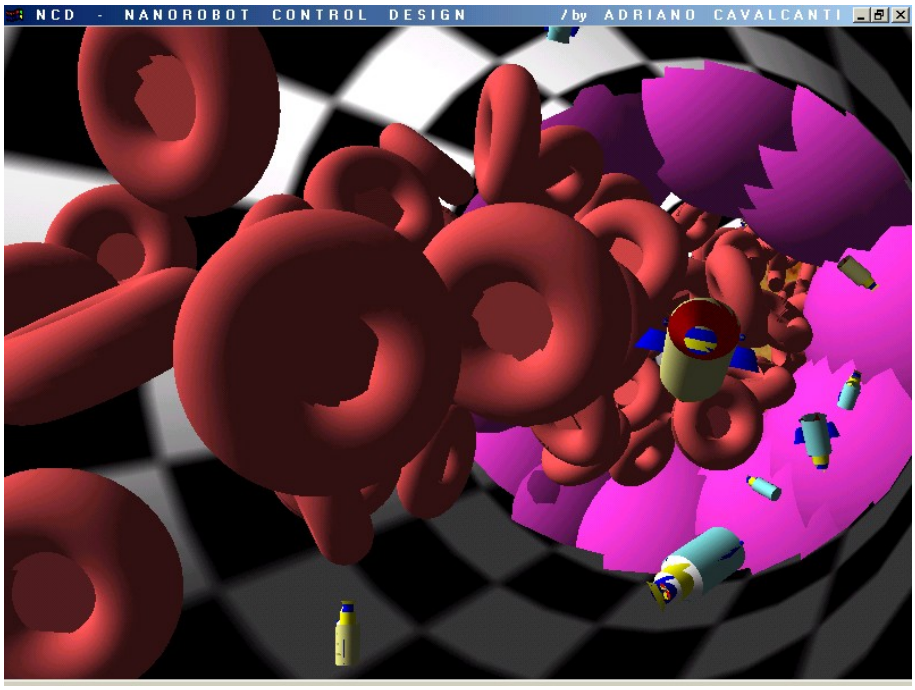

Figure 10.2: Artery 3D rendering with 60% occlusion, red blood cells and nanorobots.

10.6 Nanotechnology Research - The Bridge for New Frontiers

New aspects of control theory and automation models on many different facets of nanotechnology are still to come, specifically for nanomedicine considering the complex range of problems to be solved on such issues. Therefore many different aspects must be investigated in the future and the expectation is that the advent of nanotechnology will bring together more than ever engineers and medical experts in order to open new frontiers for mankind. The design and development of complex nanosystems with high performance can be well analysed and addressed via simulation to help pave the way for the future use of nanorobots in biomedical engineering applications.

In reality nanorobots are not a new invention from any nanotechnology expert; the fact is that their nature has been shown to us: the construction of a molecular machine is really possible and workable. Indeed the existence of the virus, bacterium, and ribosome, gives us an insight into complex mechanisms and molecular machines that have performed complex tasks for millions of years. Even though it may take some time for the international scientific community to dominate completely the requisites to fulfil such a challenge of building functional nanorobots, all the actual interdisciplinary developments evolved are important works that promise revolutionary advances in our current technology.

The methodology developed through this research provides the basis and motivation for further research and investigation of control system for nanorobots. A broad range of possible nanorobot applications can be achieved in the future, but only if we start now moving in that direction. A tree will never give you an apple just a few hours after you ringing the phone. It demands nurturing, wisdom and time.

Appendix A

Environment Dynamics

Enviroment Interaction

The 3D environment contains nanorobots, obstacles, biomolecules and specific medical targets, named organ-inlets. Organ-inlets represent medical targets, displaced stochastically as target drug delivery points for medical applications. These organ-inlets are closed when the chemical concentrations are near the desired levels. Otherwise, it will open to allow the required injection of protein drugs. For real biomedical instrumentation, targets can be for cancer a tumour nodule, or for cardiology, it will be plaques of fat in the coronary vascular system. A target can assume different sizes and shapes according with the biomedical application. In our study the organ-inlets act as a general purpose target for biomedical identification. This approach provides a practical test-bed environment for nanorobots control design. Further physically based simulation and related references is found in the dissertation on Chapters 3 and 8.

Diffusing Signals

A key choice in chemical signalling is the measurement time and detection threshold at which the signal is considered to be received. Due to background concentration, some detection occurs even without the target signal. As a threshold, we use the diffusive capture rate ϕ for a sphere of radius R in a region with concentration as:

$$\phi = 4\pi DRC \tag{1}$$

where the concentration for other shapes such as cylinders are about the same. All moving objects (i.e., the nanorobots and biomolecules) in the workspace have neutral buoyancy. In regard of circulatory system about vessels geometries and the nanorobot sizes for medical purposes, the lumen diameters ranges from the vena cava with ~3cm in the heart, to ~10μm of capillary vessels. In the present study the nanorobot is transporting proteins.

TABLE 1. Parameters

Chemical signal	
production rate	$\dot{Q} = 10^4 molecule \quad s^{-1}$
diffusion coefficient	$D = 100 \mu m^2 s^{-1}$
background concentration	$6 \times 10^{-3} molecule \quad \mu m^{-3}$
Parameter	**Nominal value**
average fluid velocity	$v = 1000 \mu m \quad s^{-1}$
vessel diameter	$d = 10 \mu m, 20 \mu m, 40 \mu m$
workspace length	$L = 60 \mu m$
density of nanorobots	$3 \mu m^3$

The virtual environment as testbed includes a randomized network of obstacles. This construction creates a random network of obstacles in the plane bisecting environment, which is quite appropriate as a basis for nanorobot control feedback purposes of motion analyzes and obstacle avoidance. The environment presents also spheres with 10nm diameter as bio-molecules that the nanorobots can use to supply medical targets with proteins. These spheres move with the fluid, and follow the laminar flow considered additional Brownian motions. In a typical molecular dynamics simulation, a set of molecules is introduced initially with a random velocity for each molecule and the intermolecular interactions can be expressed, using Lennard-Jones potential:

$$V(r) = 4\varepsilon \left[\left(\frac{r}{\sigma} \right)^{-12} - \left(\frac{r}{\sigma} \right)^{-6} \right] \tag{2}$$

Except for coronary artery blood flow, in the typical biomedical applications the blood flow is laminar, especially for smaller vessels where the fluid velocity is typically lower.

Hence, in order to simulate the nanorobot intervention and interaction with the workspace, we used different organ-inlets as delivery targets, where depending on their protein demand, they will be emitting chemical and thermal signals. We simulated distinct cases to validate our study given the Eq. (1) and parameters on Table 1 for diffusing signals.

Fluid Dynamics

The fluid in the workspace moves through the vessel with velocity 1mm/sec, as is typical of flow in small blood vessels. The fluid is described by the classical continuum equations. The continuity condition $\nabla \cdot v = 0$ and the Navier-Stokes equation are applied for the velocity *v* of the fluid:

$$\frac{\partial v}{\partial t} + (v \cdot \nabla) v = f - \frac{1}{\rho} \nabla P + \frac{\eta}{\rho} \nabla^2 v, \tag{3}$$

where η is the fluid's viscosity, ρ its density, P is the pressure and f is the external force, per unit mass, imposed on the fluid. The three components of the Navier-Stokes equation and the continuity condition give four equations for the three components of the velocity and the pressure. In contrast to the conventional and large-scale robots, the nanorobot's world is dominated by viscosity while inertial and gravitational forces are negligible. The Reynolds number, defined as:

$$\mathrm{Re} = L\rho v / \eta \,, \tag{4}$$

for objects of size L with velocity v, characterizes this behavior by giving the ratio of inertial to viscous forces. The Re is low for nanoscale robots operating in fluids of ordinary viscosities. The surrounding liquid has density of 1 g/cm^{-3} and viscosity of 1 centipoise, or equivalent to 10^{-2} g/cm^{-1} s^{-1}. As an example, if a nanorobot of size 1μm moving at 1 mm s^{-1} in liquid flow has Re=10^{-3}, much less than 1 and hence viscous forces dominate. As boundary conditions, the flow velocity v matches the velocity of each object in the fluid at the object's surface. We also impose a constant input velocity along the pipe as a boundary condition to maintain the fluid flow. This condition is maintained by a pressure gradient imposed on the fluid.

Interaction in Viscous Flow

Our environment contains two types of moving objects: the nanorobots and the small spheres representing biomolecules. These objects are subject to both deterministic and random forces. The deterministic forces arise from the fluid motion and, in the case of the nanorobots, from their powered locomotion.

The inertial force on the object of size L moving with velocity v with respect to the fluid is of order $F_{inertial} \cong \rho v^2 L^2$ and the viscous drag force is of order $F_{viscous} \cong \eta v L$. Thus to keep moving, a nanorobot of size $L \cong$ 1μm and velocity $v \cong$ 1mm s^{-1} with respect to the fluid must apply $F_{inertial} \cong 1\,fN$ (femtonewtons, $1\,fN = 10^{-15}\,N$) and a much larger $F_{viscous} \cong 10^3\,fN$ of motive force. As a consequence of this dominance of viscosity, when a force F is applied to an object, it quickly reaches a terminal velocity where that force is canceled by the drag from the fluid. As an illustration of this behavior, if motive power to a swimming spherical nanorobot with radius L=1μm, and the velocity v=1cm s^{-1} with respect to the fluid, is suddenly stopped, then the nanorobot will "coast" to a halt with respect to the fluid in a time t_{coast} as:

$$t_{coast} = \frac{\rho L^2}{15\eta} = 0.1 \text{ microsecond} \tag{5}$$

and in distance $x_{coast} \cong v\, t_{coast} = 1nm$. A comparable result applies to other shapes, e.g., the nanorobot and the smaller sized biomolecules. Thus an applied force quickly results in motion with constant

velocity. It contrasts with the behavior when inertial forces dominate: an applied force produces a constant acceleration. A similar observation applies to rotations: a given torque rapidly produces a constant angular velocity rather than a constant angular acceleration.

Two main forces act on a nanorobot while it is not in contact with other objects. First is the force F_r produced by the robot itself, which is taken to be directed along the axis of the cylinder. Second is the drag from the fluid given by:

$$F_{drag} = -A_{drag}\,\eta L v \,, \tag{6}$$

where v is the velocity vector of the nanorobot with respect to the fluid, $v = v_{robot} - v_{fluid}$. The quantity A_{drag} is a geometric factor depending on the orientation of the nanorobot with respect to the fluid and is typically of order 1, e.g., for a sphere of diameter *L*, A_{drag} is 3π when no other objects are nearby. For other situations, A_{drag} has roughly the same magnitude, but the exact value must be determined numerically. To see how this is done, consider a small area *dA* on the surface of an object, treated as a vector oriented perpendicular to the surface. The fluid imposes a force vector $-T\,dA$ on that area, where *T* is a matrix representing the stress tensor for the fluid motion at the surface of the object. For incompressible fluids, its components are expressed:

$$T_{k,l} = P\delta_{k,l} - \eta\left(\frac{\partial v_k}{\partial x_l} + \frac{\partial v_l}{\partial xk}\right), \tag{7}$$

where is $\delta_{k,l} = 1$ when *k*=*l* and is 0 otherwise.

In general, the velocity gradient and pressure vary over the surface of the object. The total drag force requires integrating the force on each part of the object. The difference in forces around the object can also give rise to a torque, causing the object to rotate as it moves through the fluid. The total force acting on the robot is $F_r + F_{drag}$, which is zero when the nanorobot velocity equals to:

$$v_{robot} = v_{fluid} + F_r /(A_{drag}\,\eta L) \,. \tag{8}$$

The biomolecules move passively along with the fluid, i.e., their velocity is equal to v_{fluid}. Both the passive obstacles and other nanorobot are potential sources of collision and additional force. In particular, a collision with the wall of the pipe or one of the obstacles sets to zero the component of the object's velocity perpendicular to the wall or obstacle. When a biomolecule collides with a nanorobot, the biomolecule velocity perpendicular to the robot is set to zero; it may also be absorbed by the robot if it was identified as a protein. In addition to these deterministic forces, stochastic forces due to thermal motion of molecules in the fluid give rise to additional random motions, i.e., Brownian

motion. As an indication of the size of these motions, the average displacement of a particle of radius L over a time t when the fluid has temperature T is:

$$b = \left(\frac{kTt}{3\pi\eta L} \right)^{1/2}, \tag{9}$$

where k is Boltzmann's constant and b is displacement. Operating at typical body temperature, this gives displacement for the nanorobot of about $\sqrt{t}$ μm when t is measured in seconds, and $8\sqrt{t}$ μm for the biomolecules.

Collecting biomolecules is part of the robot task. A nanorobot at rest with respect to the fluid will encounter biomolecules due to their diffusion. The laminar fluid flow itself only moves the molecules along streamlines which go through the workspace. An estimate of the rate at which such a nanorobot will encounter diffusing biomolecules is of order $10LDC$, where L is the size of the robot, D is the diffusion coefficient in liquid, about 10^{-10} m^2 s^{-1}, and C is the concentration of molecules around the robot. In the simulation, C=10^{16}m^{-3} enabling the nanorobots to have an interactive response in collecting them for a further target identification, and protein drug delivery. In our simulation, the collisions between biomolecules and the robots are determined from their individual motions, including the diffusion from Brownian motion.

Object Motion

Our physically-based simulation includes kinetics and frictional aspects for object motion with hydrodynamics at low Reynolds number. Specifically, the dynamics of the objects in our environment is determined by the object positions and, for the nanorobots, their choice of locomotion force as determined from their control program. Unlike the case of inertial forces, there is no need to consider accelerations. The dynamics in the environment is processed as the boundary conditions for determining the fluid motion from Eq. (3). Given the fluid motion, we determine the net force and torque on each nanorobot, after which Eq. (8) gives its new velocity. Each biomolecule's velocity matches that of the fluid at its position. For both the nanorobots and biomolecules, this velocity is also subject to constraints from any collisions. From the perspective of each object, this process amounts to a function that evaluates its velocity v_{object} in terms of the state of the system. Using a time step of Δt, we then update the object positions according to:

$$P = object \leftarrow F + v_{object}\Delta t + \varepsilon, \tag{10}$$

P is the current position of the nanorobot, which changes given the following parameters: ε represents a random vector chosen from a Gaussian distribution with mean of 0 and average length $\sqrt{\Delta t}$ μm

(with Δt measured in seconds) for the nanorobots and 8 times as large for the biomolecules. For the nanorobots, a similar evaluation is based on the torques applied by the fluid, the nanorobots themselves as part of their locomotion, and any collisions given their angular velocities. The angular velocity then gives the change in orientation after the time Δt, and F_r is the applied force for manipulating an object. As mention before $F = F_r + F_{drag}$, and therefore F_{drag} can assume positive or negative values depending the direction of the nanorobot in relation to the bloodstream. For a 3D enviroment the forces applied to the nanorobot can be represented by:

$$F = F_x i + F_y j + F_z k \tag{11}$$

Where the F_x, F_y, F_z are the (x, y, z) components of the force. The nanorobot position is updated dynamically and has the respective values of each coordinate that comprises the coordinate system represented as follows:

$$P = X, Y, Z \tag{12}$$

According with the equation (10), the nanorobot motion is obtained from Newton's second law, which states that the summation of all external forces acting on a body is equal to the time rate of the momentum of the body, plus angular velocity - i.e., the rate of change of the momentum of a particle is proportional to the resultant force acting on the particle and is in the direction of that force. Therefore, momentum represents the product of total mass M by the velocity of the center of the mass giving us the object translation:

$$\sum F = ma \tag{13}$$

Momentum the total momentum of any closed system (one not affected by external forces) cannot change, and this law also applies electrodynamics, quantum mechanics, quantum field theory, and general relativity. For nanoworlds, the linear velocity of the nanorobot can be broken into relative velocities, related to his coordinates system, representing applied velocities using *u*, *v*, *w* respectively. Mathematically we can the vector velocity in terms of subcomponents:

$$V_M = ui + vj + wk \tag{14}$$

where (**i**, **j**, **k**) are the unit vectors along the respective nanorobot body axes.

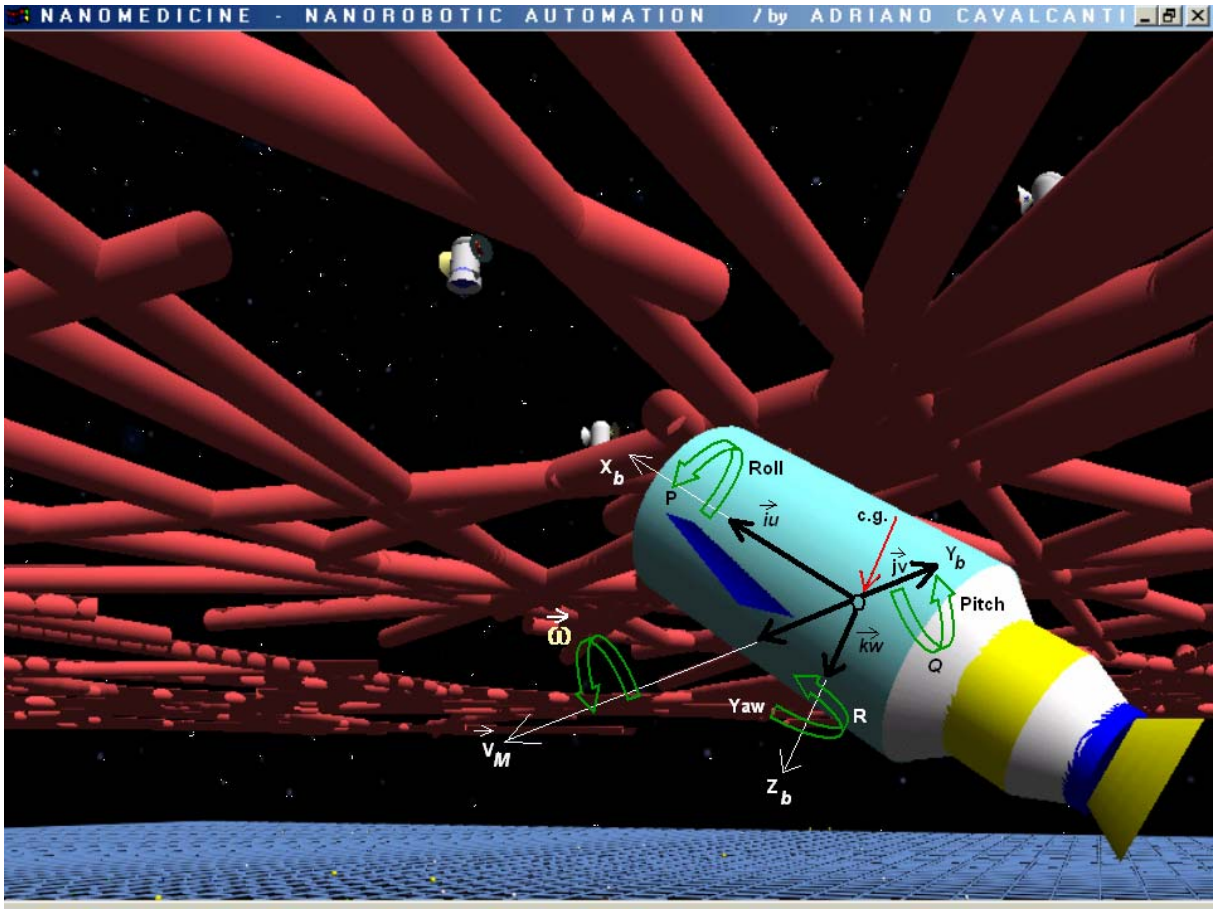

Figure 1: Representation of the unit vectors along the respective 3D nanorobot body axes.

$$\left|V_M\right| = V_M = (u^2 + v^2 + w^2)^{1/2} \tag{15}$$

Which is illustrated in the figure 1. The moment of external forces comprises the rolling moment (*L*), pitching moment (*M*) and the yawing moment (*N*):

$$\sum \Delta M = Li + Mj + Nk \tag{16}$$

In a similar manner, the nanorobot's angular velocity vector ω can be broken up into the components *P*,*Q*, and *R* about the (*Xb*, *Yb* , *Zb*) axes, respectively, as follows:

$$\omega = Pi + Qj + Rk \tag{17}$$

where *P* is the roll rate, *Q* is the pitch rate, and *R* is the yaw rate. For these parameters represented in the figure 1, the values for angular velocities ψ, ϕ, θ are obtained with the integration of $(d\psi/dt, d\phi/dt, d\theta/dt)$:

$$\frac{d\psi}{dt} = (Q\sin\phi + R\cos\phi)/\cos\theta \quad \rightarrow \quad \psi = \psi_0 + \int_0^t \left(\frac{d\psi}{dt}\right) dt \tag{18}$$

$$\frac{d\phi}{dt} = P + \left(\frac{d\psi}{dt}\right)\sin\theta \quad \rightarrow \quad \phi = \phi_0 + \int_0^t \left(\frac{d\phi}{dt}\right) dt \tag{19}$$

$$\frac{d\theta}{dt} = Q\cos\phi - R\sin\phi \quad \rightarrow \quad \theta = \theta_0 + \int_0^t \left(\frac{d\theta}{dt}\right) dt \tag{20}$$

The angular velocity in terms of Euler angles, comprises a time dependent system and interferes with the positioning of the nanorobot, and can be expressed in a matrix as follows:

$$\frac{d}{dt}\begin{bmatrix} X_e \\ Y_e \\ Z_e \end{bmatrix} = C_e^b \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{21}$$

with

$$\frac{dX_e}{dt} = (\cos\theta\cos\psi)u + (\cos\psi\sin\phi\sin\theta - \sin\psi\cos\phi)v + (\cos\psi\cos\phi\sin\theta + \sin\psi\sin\phi)w \tag{22}$$

$$\frac{dY_e}{dt} = (\cos\theta\sin\psi)u + (\sin\psi\sin\phi\sin\theta - \cos\psi\cos\phi)v + (\sin\psi\cos\phi\sin\theta - \cos\psi\sin\phi)w \tag{23}$$

$$\frac{dZ_e}{dt} = -(\sin\theta)u + (\sin\phi\cos\theta)v + (\cos\theta\cos\phi)w \tag{24}$$

From the equation 21, for a trajectory of objects in a dynamic environment the motion is described by:

$$X_e = X_{e,0} + \int_0^t \left(\frac{dX_e}{dt}\right) dt \tag{25}$$

$$Y_e = Y_{e,0} + \int_0^t \left(\frac{dY_e}{dt}\right) dt \tag{26}$$

$$Z_e = Z_{e,0} + \int_0^t \left(\frac{dZ_e}{dt}\right) dt \tag{27}$$

where axes coordinates are updated in relation to time. The physical simulation uses the computational approach based on a real time clock and independent of the fps (frames per second) rate in the

rendering pipeline that update the objects positions. Thus, there is no relation between fps and the simulator timer to update the physical environment. This allows showing the behavior in fast or slow motion, as the user requests, without changing the physical simulation. The simulator maintains a list of positions and orientations of all objects in the task environment, including the nanorobots. This list maintains all information relevant to the nanorobot interactions in the workspace. It also introduces new bio-molecules with the fluid as it enters the environment extremity. The simulator consists of several modules that simulate physical behaviors, determine sensory information for each nanorobot, run the control programs to determine the nanorobot sensing activation, provide a visual display of the environment, and record the chemical levels monitored by the nanorobots. The processing approach for the environment involves a multithreaded system, which provides dynamic updates for the nanorobot real-time sensing and activation. This same concept and implementation is applied in relation to other nanorobots, as well as to the surrounding workspace.

Appendix B

Decision Control

Source Code

The sequence discloses two classes that contain the events and structures related to nanorobot interactive decision control and evolutionary optimization. The code was implemented using an object oriented approach, which helps to keep the software development and multi-agent based processing effective.

The program CrobotDecisionSensing is organized into two distinct files, saved with the extension “cpp” and “.h”. As the name suggests, it is used for the nanorobots which perform sensing interactive actions. The second program called CrobotDecisionEvolutionary, is where control optimization takes place, with a focus on the control of protein levels for each organ inlet whitin the simulated environment. Comments throughout the code are used, and clear names for variables and functions are adopted as a feasible way to make it easier to undertake maintenance of the software. Mathematical and logical explanation of the code related to evolutionary computation, control decision, competitive and collective behaviours, were described in Chapters 6, 8 and 9.

CLASS: CrobotDecisionSensing

```
//***************************************************************************//
//***************************************************************************//
// FILE CRobotDecisionSensing.h
#ifndef __CROBOTDECISIONSENSING_H
#define __CROBOTDECISIONSENSING_H
//***************************************************************************//
//***************************************************************************//
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
//***************************************************************************//
//***************************************************************************//
#include "CRobotDecisionEvolutionary.h"
#include "CParallelCriticalSection.h"
#include "CDataSetupSimulator.h"
//***************************************************************************//
```

```
//*************************************************************************//
class CRobotDecisionSensing
{
	private:
		int n_inlets_continue;
		int global_timeLength;
		int n_timeframes;
		int type;
		int complexity;

		// aim_NonLinear		robotDeliveryGoal*complexity
		int goal_NonLinear;
		// aim_Linear
		int goalGA;
		// Generate numbers [0,1]
		int RangeOperation_Integer;
		// Generate numbers [0,100] :: nonIntegerControl
		int RangeOperation_Continue;

		CRobotDecisionEvolutionary nanorobot_IntegerControl, nanorobot_nonIntegerControl;

		void Setup(int agent);
		void initData(int nanorobotID);
		void linkIntConstOBJincrease(int timer);
		void LinkIntConstOBJdecrease(int timer);
		void linkIncrease(int timer);
		void linkDecrease(int timer);

		void chooseInlet(int timer);
		void delivery(int timer);
		void setConstantOutput(void);
		int takeSmaller(void);
		void freeData(void);

		void writeCheckFile(void);
		void writeCheckFileOpen(void);
		void writeCheckFileClose(void);

	public:
		CRobotDecisionSensing() {}
		~CRobotDecisionSensing() {}

		CParallelCriticalSection *criticSection;
		CDataSetupSimulator *setup;

		int *ProteinLevelLevelInlet;//[setup.n_inletTotal];
		int *sendSocketMaxProteinUB;
		int *outputInlet;//[setup.n_inletTotal];
		int *ProteinLevelLevel;//[setup.n_inletTotal];
		int *constantOutput;//[setup.n_inletTotal];
		int *reagent;//[setup.n_inletTotal];
		int *controlActived;//[setup.n_inletTotal]
		int *sort;//[setup.n_inletTotal]

		void writeCheckFile2(int timer);

		void adaptiveAPI(int nanorobotID);

};
//*************************************************************************//
//*************************************************************************//
```

```
#endif // __CROBOTDECISIONSENSING_H
//***************************************************************************//
//***************************************************************************//
// FILE CRobotDecisionSensing.cpp
#include "CRobotDecisionSensing.h"
//***************************************************************************//
//***************************************************************************//
void CRobotDecisionSensing::Setup(int agent)
{	// here the word Continue means: not Integer Programming

	complexity=setup->GComplexity;
	global_timeLength=setup->Nhours;
	n_timeframes=setup->Ndays;
	// n. intervention
	type=setup->n_InletIntervention;
	// Generate numbers [0,1]
	RangeOperation_Integer=2;
	// Generate numbers [0,100]
	RangeOperation_Continue=101;
	// Objective itens to be delivered
	goal_NonLinear=(setup->robotDeliveryGoal*complexity);
	// Itens to be delivered: plus the lack variables
	//goalGA=(int)goal_NonLinear*(float)1.09;
	goalGA=(int)goal_NonLinear*(float)setup->gapVarible[setup->setProcessTime];//1.18;
	// lack variables in the math model
	n_inlets_continue=2*complexity;
}
//***************************************************************************//
//***************************************************************************//
void CRobotDecisionSensing::initData(int nanorobotID)
{
	//-------------------------------------------------/
	criticSection=receive_criticSection;
	setup=receive_setup;
	//-------------------------------------------------/
	ProteinLevelLevelInlet=(int*)malloc(setup->n_inletTotal*sizeof(int));

	if(ProteinLevelLevelInlet == NULL)
	{ printf("\n insufficient memory for ProteinLevelLevelInlet");
	  exit(1);
	}
	//-------------------------------------------------/

	sendSocketMaxProteinUB=(int*)malloc(setup->n_inletTotal*sizeof(int));

	if(sendSocketMaxProteinUB == NULL)
	{ printf("\n insufficient memory for sendSocketMaxProteinUB");
	  exit(1);
	}
	//-------------------------------------------------/
	outputInlet=(int*)malloc(setup->n_inletTotal*sizeof(int));
	if(outputInlet == NULL)
	{ printf("\n insufficient memory for outputInlet");
	  exit(1);
	}
	//-------------------------------------------------/
	ProteinLevelLevel=(int*)malloc(setup->n_inletTotal*sizeof(int));
	if(ProteinLevelLevel == NULL)
	{ printf("\n insufficient memory for ProteinLevelLevel");
	  exit(1);
	}
```

```
        //-------------------------------------------------/
        constantOutput=(int*)malloc(setup->n_inletTotal*sizeof(int));
        if(constantOutput == NULL)
        { printf("\n insufficient memory for constantOutput");
          exit(1);
        }
        //-------------------------------------------------/
        controlActived=(int*)malloc(setup->n_inletTotal*sizeof(int));
        if(controlActived == NULL)
        { printf("\n insufficient memory for controlActived");
          exit(1);
        }
        //-------------------------------------------------/
        sort=(int*)malloc(setup->n_inletTotal*sizeof(int));
        if(sort == NULL)
        { printf("\n insufficient memory for sort");
          exit(1);
        }
        //-------------------------------------------------/
        reagent=(int*)malloc(setup->n_inletTotal*sizeof(int));
        if(reagent == NULL)
        { printf("\n insufficient memory for reagent");
          exit(1);
        }
        //-------------------------------------------------/
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::freeData(void)
{       free(ProteinLevelLevelInlet);
        free(sendSocketMaxProteinUB);
        free(outputInlet);
        free(ProteinLevelLevel);
        free(constantOutput);
        free(reagent);
        free(controlActived);
        free(sort);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::adaptiveAPI(int nanorobotID)
{       int pack,currentTime_xls;
        int breakTime,timer,status,Inlet;

        initData(nanorobotID);
        Setup(nanorobotID);

        nanorobot_IntegerControl.initDataGeneticObj(setup,
                                        RangeOperation_Integer,
                                        setup->n_inletTotal,
                                        global_timeLength+1,
                                        goalGA,
                                        type);
        nanorobot_IntegerControl.Agent=protectHealth;

        for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
        {
                ProteinLevelLevelInlet[Inlet]=nanorobot_IntegerControl.organ[Inlet].vol;
                sendSocketMaxProteinUB[Inlet]=nanorobot_IntegerControl.organ[Inlet].max;
        }
```

```
	if(!nanorobot_IntegerControl.Agent)
	{	criticSection->initProteinLevel(setup->n_inletTotal,
					ProteinLevelLevelInlet,
						sendSocketMaxProteinUB);
	}

	nanorobot_nonIntegerControl.initDataGeneticObj(setup,
					RangeOperation_Continue,
					n_inlets_continue,
					global_timeLength+1,
					goal_NonLinear,
					type);
	nanorobot_nonIntegerControl.Agent=protectHealth;
	//-----------------------------------------------------//
	breakTime=global_timeLength+1;
	currentTime_xls=0;

	for(pack=0;pack<n_timeframes;pack++)
	{
	    for(timer=1;timer<breakTime;timer++)
	    {
		currentTime_xls++;
		nanorobot_IntegerControl.setTimer(timer);
		nanorobot_nonIntegerControl.setTimer(timer);
		chooseInlet(timer); // Run -  AG : Interger and Non Integer
		delivery(timer);

nanorobot_IntegerControl.mean_AG_count_LOOP+=(float)nanorobot_IntegerControl.AG_count_LOOP;

		if(!setup->keep_parallelThreadings)
		{	// stop the genetic_search, stop threadings
			pack=n_timeframes;
			timer=breakTime;
		}
	     }

	    nanorobot_IntegerControl.mean_AG_count_LOOP/=(float)global_timeLength;
	    nanorobot_IntegerControl.Excell(breakTime,pack,currentTime_xls,goal_NonLinear);
	}
	freeData();
	status=criticSection->keepSimulator(1);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::writeCheckFileClose(void)
{	fclose(checkAGincreaser);
	fclose(checkAGdecreaser);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::writeCheckFile(void)
{	// choose output of variables to analysis
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::writeCheckFileOpen(void)
{
	int i;
	char fileName1[40]="AGincreaser";
	char fileName2[40]="AGdecreaser";
	char extension[5];
```

```
	for(i=0;i<setup->sizeExtension;i++)
	{	extension[i]=setup->extension[i];

	}

	strcat(fileName1,extension);
	checkAGincreaser=fopen(fileName1,"w");
	strcat(fileName2,extension);
	checkAGdecreaser=fopen(fileName2,"w");

}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::linkIntConstOBJincrease(int timer)
{	int inletID,step,organID;
	//-------------------------------------------------------------//
	// Add all Activated Inlet
	for(inletID=0;inletID<nanorobot_IntegerControl.size;inletID++)
	{	if(nanorobot_IntegerControl.organ[inletID].selected)
		{
nanorobot_nonIntegerControl.proteins_integer+=nanorobot_IntegerControl.organ[inletID].output;//2
		}
	}
	inletID=0;
	// Subtract Continuous Activated Inlet
	for(organID=0;organID<complexity;organID++)
	{	if(nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected)
		{ nanorobot_nonIntegerControl.proteins_integer-=nanorobot_IntegerControl.organ[3+setup-
> tenOrgansInlet*organID].output;
		  nanorobot_nonIntegerControl.organ[inletID].selected=1;

nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].vol;
		  nanorobot_nonIntegerControl.organ[inletID].reagent=
					nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].reagent;
		}
		else
		{  nanorobot_nonIntegerControl.organ[inletID].selected=0;

nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].vol;		nanorobot_nonIntegerControl.organ[inletID].reagent=
					nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].reagent;
		}
		inletID++;
		if(nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected)
		{ nanorobot_nonIntegerControl.proteins_integer-=nanorobot_IntegerControl.organ[7+setup-
> tenOrgansInlet*organID].output;
		  nanorobot_nonIntegerControl.organ[inletID].selected=1;


nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].vol;
		  nanorobot_nonIntegerControl.organ[inletID].reagent=
					nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].reagent;
		}
		else
		{  nanorobot_nonIntegerControl.organ[inletID].selected=0;
```

```
nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[7+setup-
>tenOrgansInlet*organID].vol;
                        nanorobot_nonIntegerControl.organ[inletID].reagent=
                                                nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].reagent;
                    }
                    inletID++;
            }
            //-------------------------------------------------------------//
            puts("\n");
            for(inletID=0;inletID<n_inlets_continue;inletID++)

{   nanorobot_nonIntegerControl.organInlet2[inletID].vol=nanorobot_nonIntegerControl.organ[inletID].vol;

nanorobot_nonIntegerControl.organInlet2[inletID].reagent=nanorobot_nonIntegerControl.organ[inletID].reagent
;
            }
            //-------------------------------------------------------------//
            nanorobot_nonIntegerControl.AGAPI();
            //-------------------------------------------------------------//
            inletID=0;
            for(organID=0;organID<complexity;organID++)
            {   nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].xTotal=0;
                nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].goal=0;

                if(nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected)
                {    nanorobot_IntegerControl.Result[timer-1].ON_OFF[3+setup->tenOrgansInlet*organID]=1;
                    for(step=0;step<type;step++)
                    {   nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].x_t[step]=
                                                    nanorobot_IntegerControl.organ[3+setup>tenOrgansInlet *
organID].output *
                                    nanorobot_nonIntegerControl.organ[inletID].selected/100;
                        nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].xTotal+=
                                    nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].x_t[step];
                        nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].goal+=
                                    nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].x_t[step];
                    }
            }
            else
            {       nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected=0;
                    nanorobot_IntegerControl.Result[timer-1].ON_OFF[3+setup->tenOrgansInlet*organID]=0;
             }

             inletID++;

             nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].xTotal=0;
             nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].goal=0;

            if(nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected)
            {    nanorobot_IntegerControl.Result[timer-1].ON_OFF[7+setup->tenOrgansInlet*organID]=1;
                for(step=0;step<type;step++)
                {
                  nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].x_t[step]=
                            nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet * organID].output *
                                        nanorobot_nonIntegerControl.organ[inletID].selected/100;
                   nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].xTotal+=
                            nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].x_t[step];
                     nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].goal+=
```

```
                        nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].x_t[step];
                }
            }
            else
            {       nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected=0;
                    nanorobot_IntegerControl.Result[timer-1].ON_OFF[7+setup->tenOrgansInlet*organID]=0;
             }
             inletID++;
          }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::LinkIntConstOBJdecrease(int timer)
{       int inletID,step,organID;
        //-------------------------------------------------------------//
        // Add all Activated Inlet
        for(inletID=0;inletID<nanorobot_IntegerControl.size;inletID++)
        {       if(nanorobot_IntegerControl.organ[inletID].selected)

{   nanorobot_nonIntegerControl.proteins_integer+=nanorobot_IntegerControl.organ[inletID].output;//2
                }
        }
        inletID=0;
        // Subtract Continuous Activated Inlet
        for(organID=0;organID<complexity;organID++)
        {       if(nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected)
                {   nanorobot_nonIntegerControl.proteins_integer-=nanorobot_IntegerControl.organ[3+setup-
> tenOrgansInlet*organID].output;
                    nanorobot_nonIntegerControl.organ[inletID].selected=1;

nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].vol;
                    nanorobot_nonIntegerControl.organ[inletID].reagent=
                                        nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].reagent;
                }
                else
                {   nanorobot_nonIntegerControl.organ[inletID].selected=0;

nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].vol;
                    nanorobot_nonIntegerControl.organ[inletID].reagent=
                                        nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].reagent;
                }
                inletID++;
                if(nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected)
                {   nanorobot_nonIntegerControl.proteins_integer-=nanorobot_IntegerControl.organ[7+setup-
> tenOrgansInlet*organID].output;
                    nanorobot_nonIntegerControl.organ[inletID].selected=1;

nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].vol;
                    nanorobot_nonIntegerControl.organ[inletID].reagent=
                                        nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].reagent;
                }
                else
                {    nanorobot_nonIntegerControl.organ[inletID].selected=0;
```

```
nanorobot_nonIntegerControl.organ[inletID].vol=nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].vol;
                        nanorobot_nonIntegerControl.organ[inletID].reagent=
                                                nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].reagent;
                    }
                    inletID++;
            }
            //-------------------------------------------------------------//
            puts("\n");

            for(inletID=0;inletID<n_inlets_continue;inletID++)

{   nanorobot_nonIntegerControl.organInlet2[inletID].vol=nanorobot_nonIntegerControl.organ[inletID].vol;

nanorobot_nonIntegerControl.organInlet2[inletID].reagent=nanorobot_nonIntegerControl.organ[inletID].reagent
;
            }
            //-------------------------------------------------------------//
            nanorobot_nonIntegerControl.AGAPI();
            //-------------------------------------------------------------//
            inletID=0;
            for(organID=0;organID<complexity;organID++)
            {   nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].xTotal=0;
                nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].goal=0;
                if(nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected)
                {       nanorobot_IntegerControl.Result[timer-1].ON_OFF[3+setup->tenOrgansInlet*organID]=1;
                        for(step=0;step<type;step++)
                        {   nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].x_t[step]=
                                        nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].output *

            nanorobot_nonIntegerControl.organ[inletID].selected/100;
nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].xTotal+=
                                        nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].x_t[step];
                                nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].goal+=
                                        nanorobot_IntegerControl.organ[3+setup->
tenOrgansInlet*organID].x_t[step];
                        }
                }
                else
                {   nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected=0;
                    nanorobot_IntegerControl.Result[timer-1].ON_OFF[3+setup->tenOrgansInlet*organID]=0;
                }
                inletID++;

                nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].xTotal=0;
                nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].goal=0;

                if(nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected)
                {       nanorobot_IntegerControl.Result[timer-1].ON_OFF[7+setup->tenOrgansInlet*organID]=1;
                        for(step=0;step<type;step++)
                        {
                                nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].x_t[step]=
                                        nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].output *

            nanorobot_nonIntegerControl.organ[inletID].selected/100;
                                nanorobot_IntegerControl.organ[7+setup>tenOrgansInlet*organID].xTotal+=
                                        nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].x_t[step];
```

```
                        nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].goal+=
                                        nanorobot_IntegerControl.organ[7+setup->
tenOrgansInlet*organID].x_t[step];
                    }
                }
                else
                {   nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected=0;
                    nanorobot_IntegerControl.Result[timer-1].ON_OFF[7+setup->tenOrgansInlet*organID]=0;
                }
            inletID++;
        }
        //-------------------------------------------------------------//
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::linkIncrease(int timer)
{       int inletID,step,organID;
        int gapVariables=0;

        for(inletID=0;inletID<nanorobot_IntegerControl.size;inletID++)
        {  nanorobot_IntegerControl.organ[inletID].goal=0;
           if(nanorobot_IntegerControl.organ[inletID].selected)

{   nanorobot_IntegerControl.organ[inletID].goal=type*nanorobot_IntegerControl.organ[inletID].output;//2
                nanorobot_IntegerControl.Result[timer-1].ON_OFF[inletID]=1;
                for(step=0;step<type;step++)
                {  nanorobot_IntegerControl.organ[inletID].x_t[step]=
nanorobot_IntegerControl.organ[inletID].output;

        nanorobot_IntegerControl.organ[inletID].xTotal+=nanorobot_IntegerControl.organ[inletID].x_t[step];
                                        }
           }
        }
        //-------------------------------------------------------------//
        for(organID=0;organID<complexity;organID++)
        {       gapVariables+=nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected;
                gapVariables+=nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected;
        }
        // there must be: gapVariables and goalAG>goal_NonLinear
        if(nanorobot_IntegerControl.Result[timer-1].goal_integer>goal_NonLinear && gapVariables)
        {       linkIntConstOBJincrease(timer);
        }
        //-------------------------------------------------------------//
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::linkDecrease(int timer)
{       int inletID,step,organID;
        int gapVariables=0;

        for(inletID=0;inletID<nanorobot_IntegerControl.size;inletID++)
        {  nanorobot_IntegerControl.organ[inletID].goal=0;

           if(nanorobot_IntegerControl.organ[inletID].selected)

{  nanorobot_IntegerControl.organ[inletID].goal=type*nanorobot_IntegerControl.organ[inletID].output;//2
                nanorobot_IntegerControl.Result[timer-1].ON_OFF[inletID]=1;
                for(step=0;step<type;step++)
                {   nanorobot_IntegerControl.organ[inletID].x_t[step]=
nanorobot_IntegerControl.organ[inletID].output;
```

```
nanorobot_IntegerControl.organ[inletID].xTotal+=nanorobot_IntegerControl.organ[inletID].x_t[step];
                                        }
                }
        }
        //-------------------------------------------------------------//
        for(organID=0;organID<complexity;organID++)
        {       gapVariables+=nanorobot_IntegerControl.organ[3+setup->tenOrgansInlet*organID].selected;
                gapVariables+=nanorobot_IntegerControl.organ[7+setup->tenOrgansInlet*organID].selected;
        }
        // there must be: gapVariables and goalAG>goal_NonLinear
        if(nanorobot_IntegerControl.Result[timer-1].goal_integer+(goal_NonLinear/2)>goal_NonLinear &&
gapVariables)
        {        LinkIntConstOBJdecrease(timer);
        }
        //-------------------------------------------------------------//
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::chooseInlet(int timer)
{ int inletID,ordem;
         nanorobot_IntegerControl.Init_xTotal_t();
        // Chama AGAPI - Probl. Variaveis Inteiras
        /*-------------------------------------------------------------------*/
        // alteracoes para rodar com o genetico
        ordem=0;
        nanorobot_nonIntegerControl.proteins_integer=0;
        nanorobot_IntegerControl.AGAPI();

        for(inletID=0;inletID<nanorobot_IntegerControl.size;inletID++)
        { nanorobot_IntegerControl.Result[timer-1].ON_OFF[inletID]=0;
        }
        if(nanorobot_IntegerControl.Agent)
        {   linkIncrease(timer);
        }
        else
        {    linkDecrease(timer);
        }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::writeCheckFile2(int timer)
{       FILE *checkAdap;
        int Inlet,i;

        //-------------------------------------------------------------------/
        //--------  itoa(var,probl,2)   ------------------------------------/
        char kindData[5];
        char probl[5];
        //char fileName[15]={"adaptSocket"};
        char fileName[40]="adaptSocket";
        //-------------------------------------------------//
        char extension[5];
        for(i=0;i<setup->sizeExtension;i++)
        {       extension[i]=setup->extension[i];
        }
        //-------------------------------------------------//
        sprintf(kindData,"%d",nanorobot_IntegerControl.Agent);
        strcat(fileName,kindData);
        sprintf(probl,"%d",timer);
        strcat(fileName,probl);
        strcat(fileName,extension);
```

```
	checkAdap=fopen(fileName,"w");

	for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
	{	fprintf(checkAdap,"\n AG[%d].x_t[0]*type=
%d\t",Inlet,nanorobot_IntegerControl.organ[Inlet].x_t[0]*type);
		fprintf(checkAdap,"nanorobot_IntegerControl.organ[%d].reagent=
%d\n",Inlet,nanorobot_IntegerControl.organ[Inlet].reagent);
	}
	fprintf(checkAdap,"\nnanorobot_IntegerControl.Result[timer-1].goal_nonInteger = %d\n\n",
                                                    nanorobot_IntegerControl.Result[timer-
1].goal_nonInteger);
	fprintf(checkAdap,"ag_count_LOOP = %.1f\n",nanorobot_IntegerControl.AG_count_LOOP);
	fprintf(checkAdap,"ag_count_LOOP = %.1f\n",nanorobot_nonIntegerControl.AG_count_LOOP);
	fclose(checkAdap);
}
//*************************************************************************//
//*************************************************************************//
int CRobotDecisionSensing::takeSmaller(void)
{	int Inlet;
	int lowerProteinLevel=99999;
	int lowerInlet;
	int nextStepProteinLevel;
	for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
	{  if(!controlActived[Inlet])
	   {	nextStepProteinLevel=((nanorobot_IntegerControl.organ[Inlet].vol-
nanorobot_IntegerControl.organ[Inlet].input) *100) /

	nanorobot_IntegerControl.organ[Inlet].max;
			nextStepProteinLevel*=1;
			if(lowerProteinLevel>nextStepProteinLevel)
			{	lowerProteinLevel=nextStepProteinLevel;
				lowerInlet=Inlet;
			}

	    }
	}
	controlActived[lowerInlet]=1;
	return(lowerInlet);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::setConstantOutput(void)
{	int higherProteinLevel,Inlet,sum,whatDifferInlet,differ,i;
	higherProteinLevel=0;
	//-----------------------------------------------------------//
	// sort from smaller to greatest
	for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
	{	controlActived[Inlet]=0;
	}
	for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
	{	sort[Inlet]=takeSmaller();
	}
	//-----------------------------------------------------------//
	Inlet=setup->n_inletTotal;
	sum=0;
	for(i=0;i<setup->n_inletTotal;i++)
	{	--Inlet;
		if(sum<goal_NonLinear)
		{	if(!nanorobot_IntegerControl.organ[sort[Inlet]].x_t[0])
			{	sum+=nanorobot_IntegerControl.organ[sort[Inlet]].input;
				whatDifferInlet=sort[Inlet];
```

```
		constantOutput[sort[Inlet]]=nanorobot_IntegerControl.organ[sort[Inlet]].input;
				}
			}
		}
		if(sum)
		{	differ=sum-goal_NonLinear;
			constantOutput[whatDifferInlet]=nanorobot_IntegerControl.organ[whatDifferInlet].input-
differ;
		}
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionSensing::setConstantOutput(void)
{	int Inlet,sum=0;
	for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
	{	constantOutput[Inlet]=nanorobot_IntegerControl.organ[Inlet].input/2;
		sum+=constantOutput[Inlet];
	}
}
//*************************************************************************//
//*************************************************************************//
// all values in delivery are in cubic meter
void CRobotDecisionSensing::delivery(int timer)
{	int inletID,t,caudalTotal,Inlet,wait;
	for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
	{	outputInlet[Inlet]=nanorobot_IntegerControl.organ[Inlet].x_t[0];
		constantOutput[Inlet]=0;
		ProteinLevelLevel[Inlet]=0;
		reagent[Inlet]=0;
		nanorobot_IntegerControl.organ[Inlet].reagent=0;
	}
	if(!nanorobot_IntegerControl.Agent)
	{	setConstantOutput();
	}
	do
	{	wait=criticSection->setupProteinLevel(timer, nanorobot_IntegerControl.Agent, setup->
n_inletTotal,type,
							outputInlet, ProteinLevelLevel, reagent, constantOutput);
		if(wait)
		{   //printf("\n sleep#1 agent = %d  time = %d",nanorobot_IntegerControl.Agent,timer);
			Sleep(10000);// sleep time in milliseconds
		}
	}while(wait);
	for(Inlet=0;Inlet<setup->n_inletTotal;Inlet++)
	{   nanorobot_IntegerControl.organ[Inlet].vol=ProteinLevelLevel[Inlet];
	    nanorobot_IntegerControl.organ[Inlet].reagent=reagent[Inlet];
	}
	//fclose(check);

	for(t=0;t<type;t++)
	{  caudalTotal=0;
	   for(inletID=0;inletID<nanorobot_IntegerControl.size;inletID++)
	   {	caudalTotal+=nanorobot_IntegerControl.organ[inletID].x_t[t];
		caudalTotal+=constantOutput[inletID];
	   }
	   //---------------------------------------------/
	   // Format output for Excell
	   nanorobot_IntegerControl.Result[timer-1].goal_nonInteger=caudalTotal;
	   //---------------------------------------------/
	}// End of the first "for" loop
```

```
        //writeCheckFile2(timer);
        // Format output for Excell
        for(inletID=0;inletID<nanorobot_IntegerControl.size;inletID++)
        {  nanorobot_IntegerControl.Result[timer-1].Vol_Perc[inletID]= 100 *

        nanorobot_IntegerControl.organ[inletID].vol/nanorobot_IntegerControl.organ[inletID].max;
        }
}
```

CLASS: CrobotDecisionEvolutionary

```
//*************************************************************************//
//*************************************************************************//
// FILE CRobotDecisionEvolutionary.h
//*************************************************************************//
//*************************************************************************//
#ifndef __CROBOTDECISIONEVOLUTIONARY_H
#define __CROBOTDECISIONEVOLUTIONARY_H
//*************************************************************************//
//*************************************************************************//
// ClassGenetic Specification
//*************************************************************************//
//*************************************************************************//
#include <process.h>
#include <windows.h>
#include <winuser.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include <time.h>
//*************************************************************************//
//*************************************************************************//
#include "CDataRandomize.h"
#include "CDataSetupSimulator.h"
//*************************************************************************//
//*************************************************************************//
class CRobotDecisionEvolutionary
{       private:
                //101 ::random number 0 - 100
                int rangeOperation;
                // Population size = number of individuals
                int num_seq;
                // crossover percentage
                float  Kporcentagem;
                // mutation percentage
                float  Kmut_rate;
                // number of mutation
                int    num_mut;
                // 15 seconds: determine how longer to process
                long   time_monitor;
                long genetic_timing;
                // n solution forbidden to recombine
                int    num_forbidden;
                // fitness do melhor solution_ID
                float       bestfitness;
                int type;
```

```
        float *slice,*fitness_main,*fitness_integer;
        int *selected,*ordered,*blocked_father;
        int *chromosome_aux,*solution_global,*best_chromosome;
        int *solution_passing,*start_inlet;
        int *solution_temp;

        struct population{    int *column;
        }*population_main, *population_int;

        float InitInletProteinLevel;
        int SendGoal;
        int timer;

        // Methods :: Genetic methods
        void start_population(void);
        void genetics(void);
        void fc_verify(void);
        void mutation_swap(int n_swaps);
        void include_solution(int index);
        void crossover_UX(void);
        void crossover_OX(void);
        void ordering_population(void);
        int select_solution(void);
        float calculate_fitness(int last, int show_it);
        void verifygapVariables(void);
        float integerFitnessIncrease(int last, int show_it);
        void setnotActived(void);
        int notevenActived(void);
        float continuousFitnessIncrease(int last, int show_it);
        float integerFitnessDecrease(int last, int show_it);
        float continuousFitnessDecrease(int last, int show_it);
        // Data methods
        void constAGSGBD(void);
        void intSGBD(void);
        void InletPosition(void);

public:
        // public variable declaration
        // public variables don't require methods to be accessed
        CRobotDecisionEvolutionary() {}
        ~CRobotDecisionEvolutionary() {}

        // radom methods
        CDataRandomize randGA;
        CDataSetupSimulator *setup;

        struct outputexcell{        int goal_nonInteger, goal_integer;
                                    int *ON_OFF,
                                    *Vol_Perc;
        }*Result;
        struct inletsK{     // Comunication variables
                            int selected,vol,max,input,output;
                            int lb,ub,mean_day;
                            int reagent;
                            int x_t[4],xTotal,goal;
                            int restarted, init_Zero;
                            // Inlet position
                            int xLocate,zLocate,yLocate;
                            int avoidEverOFF;
        }*organ,*organInlet2;
```

```
                // Number of inletOrgan = chromossomes: 10 numero de tarefas
                int size, Complexity;// = 2 * Complexity;
                int proteins_integer;
                // Set if the Class instance will be a Saver or Decreaser
                int Agent;
                float AG_count_LOOP,mean_AG_count_LOOP;
                // Methods.
                void initDataGeneticObj(CDataSetupSimulator *receive_setup, int MyRange_Operation,
                                        int NumberInlet, int breakTime, int mySendGoal, int ptype);
                void AGAPI();
                void setTimer(int pTimer);
                void Excell(int breakTime,int pack,int currentTime_xls,int goal_NonLinear);
                // method required just by integer problem
                void Init_xTotal_t(void);
                //void writeCheckFile(float fitness,int distanceFlow,int kindProcess);
                //friend CDataSetupSimulator;
};
//*************************************************************************//
//*************************************************************************//
#endif // __CROBOTDECISIONEVOLUTIONARY_H


//*************************************************************************//
//*************************************************************************//
// FILE CRobotDecisionEvolutionary.cpp
#include "CRobotDecisionEvolutionary.h"
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::initDataGeneticObj(CDataSetupSimulator *receive_setup,
                                                    int MyRange_Operation,  int NumberInlet,
                                                    int breakTime, int mySendGoal, int ptype)
{       int i;
        setup=receive_setup;
        // set to the random methods if the problem is continuous or integer
        rangeOperation=MyRange_Operation;
        // Number of inletOrgan = chromossomes: 10 numero de tarefas
        size = NumberInlet;
        // Set a percentual on the initial ProteinLevel level for every Inlet
        InitInletProteinLevel=(float)setup->startProteinLevel[setup->n_InletIntervention];
        // Set the Amount to be delivered
        SendGoal=mySendGoal;

        if(Agent)
        {       SendGoal*=1;
                SendGoal*=1;
        }
        // Set the number of regions: each region has x_inletOrgan
        // "x_inletOrgan"=(NumberInlet/Complexity)
        Complexity=setup->GComplexity;
        type=ptype;
        // Population size = number of individuals
        num_seq= 200;
        // crossover percentage
        Kporcentagem= (float) 0.3;
        // mutation percentage
        Kmut_rate= (float) 0.1;
        // number of mutation
        num_mut =       1;
        // 15 seconds: determine how longer to process
        time_monitor =setup->timeProcessingSeconds[setup->setProcessTime];
        num_forbidden =         0; // numero de indiv. que nao podem se recombinar
```

```
bestfitness =(float) 1500000; // fitness do melhor solution_ID
// number Iteractions
AG_count_LOOP=(float)0;
mean_AG_count_LOOP=(float)0;
// it's set forever value as "0" for all "Integer Objects"
proteins_integer=0;
// Memory allocation for Structs
//-----------------------------------------------------/
fitness_main=(float*)malloc(num_seq*sizeof(float));
if(fitness_main == NULL)
{  printf("\n insufficient memory for fitness_main");
   exit(1);
}
//-----------------------------------------------------/
fitness_integer=(float*)malloc(num_seq*sizeof(float));
if(fitness_integer == NULL)
{  printf("\n insufficient memory for fitness_integer");
   exit(1);
}
//-----------------------------------------------------/
slice=(float*)malloc(num_seq*sizeof(float));
if(slice == NULL)
{ printf("\n insufficient memory for slice");
  exit(1);
}
//-----------------------------------------------------/
ordered=(int*)malloc(num_seq*sizeof(int));
if(ordered == NULL)
{ printf("\n insufficient memory for ordered");
  exit(1);
}
//-----------------------------------------------------/
blocked_father=(int*)malloc(num_seq*sizeof(int));
if(blocked_father == NULL)
{ printf("\n insufficient memory for blocked_father");
  exit(1);
}
//-----------------------------------------------------/
selected=(int*)malloc(num_seq*sizeof(int));
if(selected == NULL)
{ printf("\n insufficient memory for selected");
  exit(1);
}
//-----------------------------------------------------/
chromosome_aux=(int*)malloc(size*sizeof(int));
if(chromosome_aux == NULL)
{ printf("\n insufficient memory for chromosome_aux");
  exit(1);
}
//-----------------------------------------------------/
best_chromosome=(int*)malloc(size*sizeof(int));

if(best_chromosome == NULL)
{ printf("\n insufficient memory for best_chromosome");
  exit(1);
}
//-----------------------------------------------------/
solution_global=(int*)malloc(size*sizeof(int));

if(solution_global == NULL)
{ printf("\n insufficient memory for solution_global");
```

```
  exit(1);
}
//-----------------------------------------------------/
start_inlet=(int*)malloc(size*sizeof(int));
if(start_inlet == NULL)
{ printf("\n insufficient memory for start_inlet");
  exit(1);
}
//-----------------------------------------------------/
solution_passing=(int*)malloc(size*sizeof(int));
if(solution_passing == NULL)
{ printf("\n insufficient memory for solution_passing");
  exit(1);
}
//-----------------------------------------------------/
solution_temp=(int*)malloc(size*sizeof(int));
if(solution_temp == NULL)
{ printf("\n insufficient memory for solution_temp");
  exit(1);
}
//-----------------------------------------------------/
organ=(struct inletsK*)malloc(size*sizeof(struct inletsK));
if(organ == NULL)
{ printf("\n insufficient memory for organ");
  exit(1);
}
//-----------------------------------------------------/
organInlet2=(struct inletsK*)malloc(size*sizeof(struct inletsK));
if(organInlet2 == NULL)
{ printf("\n insufficient memory for organInlet2");
  exit(1);
}
//-----------------------------------------------------/
population_main=(struct population*)malloc(num_seq*sizeof(struct population));
if(population_main == NULL)
{ printf("\n insufficient memory for population_main");
  exit(1);
}
for(i=0;i<num_seq;i++)
{           population_main[i].column = (int *)malloc(setup->n_inletTotal*sizeof(int));
            if(population_main[i].column == NULL)
            { printf("\nSim Memoria Para population_main[%d].column",i);
              exit(1);
            }
}
//-----------------------------------------------------/
population_int=(struct population*)malloc(num_seq*sizeof(struct population));
if(population_int == NULL)
{ printf("\n insufficient memory for population_int");
  exit(1);
}
for(i=0;i<num_seq;i++)
{   population_int[i].column = (int *)malloc(setup->n_inletTotal*sizeof(int));
    if(population_int[i].column == NULL)
    { printf("\nSim Memoria Para population_int[%d].column",i);
      exit(1);
    }
}
//-----------------------------------------------------/
Result=(struct outputexcell*)malloc(breakTime*sizeof(struct outputexcell));
if(Result == NULL)
```

```
        { printf("\n insufficient memory for Result");
          exit(1);
        }
        for(i=0;i<breakTime;i++)
        {       Result[i].ON_OFF = (int *)malloc(setup->n_inletTotal*sizeof(int));
                if(Result[i].ON_OFF == NULL)
                { printf("\nSim Memoria Para Result[%d].ON_OFF",i);
                  exit(1);
                }
        }
        for(i=0;i<breakTime;i++)
        {       Result[i].Vol_Perc = (int *)malloc(setup->n_inletTotal*sizeof(int));
                if(Result[i].Vol_Perc == NULL)
                { printf("\nSim Memoria Para Result[%d].Vol_Perc",i);
                  exit(1);
                }
        }
        //-------------------------------------------------/
        // case rangeOperation [0,1], is processed the initSGBD
        if(rangeOperation-2)
        {       //initialize continuous problem database
                constAGSGBD();
        }
        else
        {       //initialize integer problem database
                intSGBD();
        }
        for(i=0;i<size;i++)
        {       organ[i].avoidEverOFF=0;
        }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::setTimer(int pTimer)
{       timer=pTimer;
}
//*************************************************************************//
//*************************************************************************//
// Old Genetic main method
void CRobotDecisionEvolutionary::AGAPI(void)
{   int         cont2;
    start_population();
    bestfitness = (float)100000;
    //opencheck(rangeOperation);
    genetics();
    for (cont2 = 0; cont2 < size; cont2++)
    {     start_inlet[cont2]=solution_global[cont2] = best_chromosome[cont2];
    }
    // Reckons the best result
    calculate_fitness( 1, 3);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::genetics()
{   int                     cont1, cont2, cont3, cont4;
    int                     index_insercao;
    time_t                  time_begin, time_end;
    //=============================================//
    // apply heavy mutation to the whole population   //
    //=============================================//
    for (cont1 = 0; cont1 < num_seq; cont1++)
```

```
    {  for (cont3 = 0; cont3 < size; cont3++)
       { solution_global[cont3] = population_main[cont1].column[cont3];
       }
       mutation_swap(10*size);
       for (cont2 = 0; cont2 < size; cont2++)
       { solution_global[cont2] = solution_passing[cont2];
        }
       //Kbusca_local();
       for (cont3 = 0; cont3 < size; cont3++)
       { population_main[cont1].column[cont3] = solution_global[cont3];
       }
       fitness_main[cont1] = calculate_fitness(0, 0);
    }

    do
    {   cont1 = 0;
        time_begin = time(NULL);
        do
        {       index_insercao = 0;
                ordering_population();
                for (cont3 = 0; cont3 < (int)((float)Kporcentagem*(float)num_seq); cont3++)
                {       crossover_OX();
                        for (cont2 = 0; cont2 < size; cont2++)
                        { solution_global[cont2] = solution_passing[cont2];
                        }
                        if (((float)randGA.randomizeAPI(10000))/10000< Kmut_rate)
                        {       mutation_swap(num_mut);
                                for (cont2 = 0; cont2 < size; cont2++)
                                { solution_global[cont2] = solution_passing[cont2];
                                }
                        }
                        //Kbusca_local();
                        for (cont4 = 0; cont4 < size; cont4++)
                        { population_int[index_insercao].column[cont4] = solution_global[cont4];
                        }
                        index_insercao++;
                }
                include_solution(index_insercao);
                //================================//
                // calculate fitness                    //
                //================================//
                for (cont3 = 0; cont3 < num_seq; cont3++)
                {       for (cont4 = 0; cont4 < size; cont4++)
                        { solution_global[cont4] = population_main[cont3].column[cont4];
                        }
                        fitness_main[cont3] = calculate_fitness(0, 0);
                }
                fc_verify();
                time_end = time(NULL);
                cont1++;
          }while((time_end - time_begin < time_monitor) && setup->keep_parallelThreadings)
    }while((AG_count_LOOP<500000) && setup->keep_parallelThreadings);

    genetic_timing=0;//time_end - time_begin;
    }
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::start_population(void)
{     int cont1, cont2;
      for (cont1 = 0; cont1 < num_seq; cont1++)
      {   for (cont2 = 0; cont2 < size; cont2++)
```

```
        {population_main[cont1].column[cont2] = randGA.randomizeAPI(rangeOperation);
        }
        blocked_father[cont1] = 0;
        num_forbidden = 0;
      }
}
//*************************************************************************//
//*************************************************************************//
float CRobotDecisionEvolutionary::continuousFitnessIncrease(int last, int show_it)
{       int     LowerBound, distance;
        int     inletID, ProteinLevelLevel;
        float   fitness;
        int     aux_2;
        int     maximum, index;

        for(inletID=0;inletID<size;inletID++)
        {       if(!organ[inletID].selected)
                {       solution_global[inletID]=0;
                }
                organInlet2[inletID].selected=solution_global[inletID];
                solution_temp[inletID] = organInlet2[inletID].selected;
        }
        maximum = 0;
        index=20;
        LowerBound=0;
        ProteinLevelLevel=0;
        aux_2 = 0;

        for (inletID = 0; inletID < size; inletID++)
        {       aux_2 += organInlet2[inletID].selected*organInlet2[inletID].output/100;
                ProteinLevelLevel+=(int)(organ[inletID].vol-
(organ[inletID].input*0.7+organ[inletID].reagent*0.3)*2+
                                        (organInlet2[inletID].selected*organInlet2[inletID].output/100));
        }
        ProteinLevelLevel/=100;
        AG_count_LOOP++;
        distance=(int)abs(SendGoal-(aux_2+proteins_integer));
        // minimizing fitness
        fitness =  (float)(4*distance)+(float)ProteinLevelLevel;
        if ((show_it == 1)||(show_it == 3))
        {
                if(show_it==3)// final result
                {  for(inletID=0;inletID<size;inletID++)
                   { organ[inletID].selected=organInlet2[inletID].selected;
                   }
                }
        }
        return(fitness);
}
//*************************************************************************//
//*************************************************************************//
float CRobotDecisionEvolutionary::integerFitnessIncrease(int last, int show_it)
{       int     LowerBound;
        int     inletID, organID;
        float   fitness;
        int     distance;
        int     neverActived;
        int     aux_2;
        int     maximum, index;
        int     ProteinLevelOverFlow;
        int     goal_integer,goal_nonInteger;
```

```
        for(inletID=0;inletID<size;inletID++)
        {       organ[inletID].selected=solution_global[inletID];
                solution_temp[inletID] = organ[inletID].selected;
        }
        for(inletID=0;inletID<size;inletID++)
        {  if(!organ[inletID].selected)
           {    if(organ[inletID].vol-
(organ[inletID].input*0.7+organ[inletID].reagent*0.3)*2+organ[inletID].output*type
                                <=(float)organ[inletID].max*setup->lowerBound[setup->
n_InletIntervention])
                {  // too low level and not attended
                   solution_global[inletID]=solution_temp[inletID]=organ[inletID].selected=1;
                }
            }
            if(organ[inletID].selected)
            {    if(organ[inletID].vol-
(organ[inletID].input*0.7+organ[inletID].reagent*0.3)*2+organ[inletID].output*type
                          >=((float)organ[inletID].max * setup->upperBound[setup->n_InletIntervention]))
                 {       // too overloaded and injected again
                         solution_global[inletID]=solution_temp[inletID]=organ[inletID].selected=0;
                 }
             }
        }
        verifygapVariables();
        for(inletID=0;inletID<size;inletID++)
        {       organ[inletID].selected=solution_global[inletID];
                solution_temp[inletID] = organ[inletID].selected;
        }
        maximum=0;index=20;LowerBound=0;
        aux_2 = 0;
        ProteinLevelOverFlow=0;
        for (inletID = 0; inletID < size; inletID++)
        {  if(organ[inletID].selected)
           { aux_2 += organ[inletID].output;
           }
        }
        //=============================================================================/
        for(organID=0;organID<Complexity;organID++)
        {   for(inletID=0+setup->tenOrgansInlet*organID;inletID<setup->
tenOrgansInlet*(organID+1);inletID++)
            {    if(organ[inletID].selected)
                 {ProteinLevelOverFlow=(int)pow(organ[inletID].vol-
(organ[inletID].input*0.4+organ[inletID].reagent*0.6) * 4+
                                                        organ[inletID].output*type,2);
                 }
            }
        }
        //=============================================================================/
        neverActived=notevenActived();
        //=============================================================================/
        // We try to minimize to fitness
        goal_integer=SendGoal;
        goal_nonInteger=SendGoal;
        distance=(int)abs(goal_integer - aux_2);
        fitness=(float)200*distance;
        fitness+=(float)ProteinLevelOverFlow/200;
        fitness+=neverActived;
        fitness*=(float)0.15;
        AG_count_LOOP++;
        //=============================================================================/
```

```
	if ((show_it == 1)||(show_it == 3))
	{	//writeCheckFile(fitness,distance,1);
		//writeCheckFile();
		if(show_it==3) // final result
		{	Result[timer-1].goal_integer=aux_2;
			setnotActived();
		}
	}
	return(fitness);
}
//*************************************************************************//
//*************************************************************************//
float CRobotDecisionEvolutionary::continuousFitnessDecrease(int last, int show_it)
{	int	LowerBound, distance;
	int	inletID, ProteinLevelLevel;
	float	fitness;
	int	aux_2;
	int	maximum, index;
	for(inletID=0;inletID<size;inletID++)
	{	if(!organ[inletID].selected)
		{	solution_global[inletID]=0;
		}
		organInlet2[inletID].selected=solution_global[inletID];
		solution_temp[inletID] = organInlet2[inletID].selected;
	}
	maximum = 0;
	index=20;
	LowerBound=0;
	ProteinLevelLevel=0;
	aux_2 = 0;
	for (inletID = 0; inletID < size; inletID++)
	{	aux_2 += organInlet2[inletID].selected*organInlet2[inletID].output/100;
		ProteinLevelLevel+=(int)
(organ[inletID].vol(organ[inletID].input*0.7+organ[inletID].reagent*0.3)*2-
					(organInlet2[inletID].selected*organInlet2[inletID].output/100));

	}
	ProteinLevelLevel/=100;
	//=========================================================================//
	AG_count_LOOP++;
	distance=(int)abs(SendGoal/2-(aux_2+proteins_integer));
	// minimizing fitness
	fitness =  (float)(4*distance)+(float)ProteinLevelLevel;
	//=========================================================================//
	if ((show_it == 1)||(show_it == 3))
	{
		if(show_it==3)// final result
		{   for(inletID=0;inletID<size;inletID++)
		    { organ[inletID].selected=organInlet2[inletID].selected;
		    }
		}
	}
	return(fitness);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::verifygapVariables(void)
{	int organID,gap,vector[2]={3,7};
	int gapVariables=0;
	for(organID=0;organID<Complexity;organID++)
	{   if(Agent) //saver
```

```
    {    // is not added, but it is too full
         if(organ[3+setup->tenOrgansInlet*organID].selected)
         {     if(organ[3+setup->tenOrgansInlet*organID].vol-
                    (organ[3+setup->tenOrgansInlet*organID].input*0.7+
                    organ[3+setup->tenOrgansInlet*organID].reagent*0.3)*2+
                    organ[3+setup->tenOrgansInlet*organID].output*type
                         >=((float)organ[3+setup->tenOrgansInlet*organID].max
                         *setup->upperBound[setup->n_InletIntervention]))
               {    // it is too full and attended again
                    solution_global[3+setup->tenOrgansInlet*organID]=0;
               }
          }
          if(organ[7+setup->tenOrgansInlet*organID].selected)
          {       if(organ[7+setup->tenOrgansInlet*organID].vol-
                    (organ[7+setup->tenOrgansInlet*organID].input*0.7+
                    organ[7+setup->tenOrgansInlet*organID].reagent*0.3)*2+
                    organ[7+setup->tenOrgansInlet*organID].output*type
                         >=((float)organ[7+setup->tenOrgansInlet*organID].max
                              *setup->upperBound[setup->n_InletIntervention]))
                { // it is overloaded and attended again
                  solution_global[7+setup->tenOrgansInlet*organID]=0;
                }

          }
}

if(!Agent) //Decreaser
{         // is not subtrated, but it is too full
          if(!organ[3+setup->tenOrgansInlet*organID].selected)
          {         if(organ[3+setup->tenOrgansInlet*organID].vol+
                         (organ[3+setup->tenOrgansInlet*organID].input*0.7+
                              organ[3+setup->tenOrgansInlet*organID].reagent*0.3)*2-
                              organ[3+setup->tenOrgansInlet*organID].output*type
                              >=((float)organ[3+setup->tenOrgansInlet*organID].max
                              *setup->upperBound[setup->n_InletIntervention]))
                    {         // too overloaded and attended again
                              solution_global[3+setup->tenOrgansInlet*organID]=1;
                    }
          }
          if(!organ[7+setup->tenOrgansInlet*organID].selected)
          {         if(organ[7+setup->tenOrgansInlet*organID].vol+
                         (organ[7+setup->tenOrgansInlet*organID].input*0.7+
                              organ[7+setup->tenOrgansInlet*organID].reagent*0.3)*2-
                              organ[7+setup->tenOrgansInlet*organID].output*type
                              >=((float)organ[7+setup->tenOrgansInlet*organID].max
                              *setup->upperBound[setup->n_InletIntervention]))
                    {         // too overloaded and attended again
                              solution_global[7+setup->tenOrgansInlet*organID]=1;
                    }
            }

  }

  //-----------------------------------------------------------------//
  gapVariables+=solution_global[3+setup->tenOrgansInlet*organID];
  gapVariables+=solution_global[7+setup->tenOrgansInlet*organID];
}

if(!gapVariables)
{         gap=randGA.randomizeAPI(2);
          organID=randGA.randomizeAPI(Complexity);
```

```
                    solution_global[vector[gap]+setup->tenOrgansInlet*organID]=1;
            }
}
//*************************************************************************//
//*************************************************************************//
float CRobotDecisionEvolutionary::integerFitnessDecrease(int last, int show_it)
{  int     LowerBound;
   int     inletID, organID;
   float   fitness;
   int     distance;
   int     neverActived;
   int     aux_2;
   int     maximum, index;
   int     ProteinLevelOverFlow;
   int     goal_integer,goal_nonInteger;

   for(inletID=0;inletID<size;inletID++)
   {   organ[inletID].selected=solution_global[inletID];
       solution_temp[inletID] = organ[inletID].selected;
   }
   // This "for" is the first point that change from Save to Decrease
   for(inletID=0;inletID<size;inletID++)
   {   if(organ[inletID].selected)// is being subtracted, but it is too empty
       {    if(organ[inletID].vol+(organ[inletID].input*0.7+organ[inletID].reagent*0.3)*2-
organ[inletID].output*type
                              <=(float)organ[inletID].max*setup->lowerBound[setup->n_InletIntervention])
            {   // it is empty and not attended
                solution_global[inletID]=solution_temp[inletID]=organ[inletID].selected=0;
            }
       }
       if(!organ[inletID].selected)// is not subtrated, but it is too full
       {    if(organ[inletID].vol+(organ[inletID].input*0.7+organ[inletID].reagent*0.3)*2-
organ[inletID].output*type
                              >=((float)organ[inletID].max*setup->upperBound[setup->n_InletIntervention]))
            {  // it is overloaded and being injected again
               solution_global[inletID]=solution_temp[inletID]=organ[inletID].selected=1;
            }
       }
   }
   //=========================================================================//
   verifygapVariables();
   for(inletID=0;inletID<size;inletID++)
   {       organ[inletID].selected=solution_global[inletID];
           solution_temp[inletID] = organ[inletID].selected;
    }
    //=========================================================================//
    maximum = 0;
    index=20;
    LowerBound=0;
    aux_2 = 0;
    ProteinLevelOverFlow=0;
   for (inletID = 0; inletID < size; inletID++)
   {    if(organ[inletID].selected)
        { aux_2 += organ[inletID].output;
        }
    }
    //=========================================================================/
    // Here is the second point where there is change from Save to Decrease
    for(organID=0;organID<Complexity;organID++)
    {       for(inletID=0+setup->tenOrgansInlet*organID;inletID<setup->tenOrgansInlet*(organID+1);inletID++)
            {           if(organ[inletID].selected)
```

```
                {       ProteinLevelOverFlow=(int)pow(organ[inletID].vol+(organ[inletID].input*0.4 +
                                                organ[inletID].reagent*0.6) *4-
organ[inletID].output*type,2);
                }

        }
    }
    neverActived=notevenActived();
    // We try to minimize to fitness
    goal_integer=SendGoal/2;
    goal_nonInteger=SendGoal/2;
    distance=(int)abs(goal_integer - aux_2);

    fitness=(float)200*distance;
    fitness+=(float)ProteinLevelOverFlow/200;
    fitness+=neverActived;
    fitness*=(float)0.15;
    AG_count_LOOP++;
   if ((show_it == 1)||(show_it == 3))
   {
      if(show_it==3) // final result
      {   Result[timer-1].goal_integer=aux_2;
          setnotActived();
      }
   }
   return(fitness);
}
//*************************************************************************//
//*************************************************************************//
int CRobotDecisionEvolutionary::notevenActived(void)
{       int inletID;
        int neverActived=0;
        for(inletID=0;inletID<size;inletID++)
        {   if(!organ[inletID].selected)
                {       neverActived+=organ[inletID].avoidEverOFF*3*Complexity;
                }
        }
        return(neverActived);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::setnotActived(void)
{       int inletID;
        for(inletID=0;inletID<size;inletID++)
        {   if(organ[inletID].selected)
            {   organ[inletID].avoidEverOFF=0;
            }
            else
            {   organ[inletID].avoidEverOFF+=1;
            }
        }
}
//*************************************************************************//
//*************************************************************************//
        // output is the active varible from each agent, thus:
        //              agent saver(1)=  active<<output(++)>>; passive<<input (--)>>
        //              agent Decreaser(0)= active<<output(--)>>; passive<<input (++)>>
//*************************************************************************//
//*************************************************************************//
float CRobotDecisionEvolutionary::calculate_fitness(int last, int show_it)
{       if(Agent)
```

```
        {       if(rangeOperation-2)
                { return(continuousFitnessIncrease(last,show_it));
                }
                else
                { return(integerFitnessIncrease(last,show_it));
                }
        }
        else
        {       if(rangeOperation-2)
                { return(continuousFitnessDecrease(last,show_it));
                }
                else
                { return(integerFitnessDecrease(last,show_it));
                }
        }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::ordering_population(void)
{   int     cont1, cont2;
    float   maximum;
    int     index_maximum;

    for (cont1 = 0; cont1 < num_seq; cont1++)
    {  selected[cont1] = 0;
    }
    for (cont1 = 0; cont1 < num_seq; cont1++)
    {  maximum = (float)-15000;
       index_maximum = -1;
       for (cont2 = 0; cont2 < num_seq; cont2++)
       {  if ((selected[cont2] == 0) && (fitness_main[cont2] >= maximum))
          {   maximum = fitness_main[cont2];
              index_maximum = cont2;
          }
       }
       selected[index_maximum] = 1;
       ordered[cont1] = index_maximum;
    }
}
//*************************************************************************//
//*************************************************************************//
int CRobotDecisionEvolutionary::select_solution(void)
{  int                  cont1;
   int          last_elemento;
   float                 aux;
   float    total =(float)      0;
   float accumulative =(float)      0;
   //=========================//
   // reckon the total fitness
     total = (float)0;
     accumulative = (float)0;
     last_elemento = -1;
     for (cont1 = 0; cont1 < num_seq; cont1++)
     {   if (blocked_father[cont1] == 0)
         { total += (float)1/(float)(fitness_main[cont1]+(float)1);
         }
         slice[cont1] =(float) 0;
     }
     //=============================================================//
     // reckon the slice (percentage) of each solution for the roulette well
     for (cont1 = 0; cont1 < num_seq; cont1++)
```

```
    {   if (blocked_father[cont1] == 0)
        { slice[cont1] = (float)1/(float)(fitness_main[cont1]+(float)1)/(float)total;
        }
    }
    //=======================================//
    // run roulette and pick a solution
    aux = (((float)randGA.randomizeAPI(10000))/10000);
    cont1 = 0;
    while ((accumulative < aux) && (cont1 < num_seq))
    {   if (blocked_father[cont1] == 0)
        { accumulative = accumulative + slice[cont1]; last_elemento = cont1;
        }
        cont1 = cont1 + 1;
    }
    if (cont1 == num_seq)
    { cont1 = last_elemento+1;
    }
    return(cont1-1);
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::crossover_OX(void)
{   int         cont1, cont2;
    int          solution_ID_1, solution_ID_2;
    int                 position1;
    int                 position2;
    int                 passing;

    solution_ID_1 = select_solution();
    blocked_father[solution_ID_1] = 1;
    solution_ID_2 = select_solution();
    blocked_father[solution_ID_2] = 1;
    num_forbidden = 0;
    for (cont2 = 0; cont2 < num_seq; cont2++)
    {   if (blocked_father[cont2] == 1)
        { num_forbidden++;
        }
    }
    if (num_forbidden > (int)((float)num_seq*(float)0.7))
    {   for (cont1 = 0; cont1 < num_seq; cont1++)
        { blocked_father[cont1] = 0;
        }
        num_forbidden = 0;
    }
    for (cont2 = 0; cont2 < size; cont2++)
    {   chromosome_aux[cont2] = 0;
    }
    position1 = 0; position2 = 0;

    while(position1 == position2)
    {   position1 = randGA.randomizeAPI(size);
        position2 = randGA.randomizeAPI(size);
    }
    if(position1 > position2)
    {   passing = position1;
        position1 = position2;
        position2 = passing;
    }
    for (cont2 = position1; cont2 < position2+1; cont2++)
    {   chromosome_aux[cont2] = population_main[solution_ID_1].column[cont2];
    }
```

```
      for (cont2 = 0; cont2 < position1; cont2++)
      {    chromosome_aux[cont2] = population_main[solution_ID_2].column[cont2];
      }
      for (cont2 = position2+1; cont2 < size; cont2++)
      {    chromosome_aux[cont2] = population_main[solution_ID_2].column[cont2];
      }
      for (cont2 = 0; cont2 < size; cont2++)
      { solution_passing[cont2] = chromosome_aux[cont2];
      }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::crossover_UX(void)
 {    int           cont1, cont2;
      int             solution_ID_1, solution_ID_2;
      solution_ID_1 = select_solution();
      blocked_father[solution_ID_1] = 1;
      solution_ID_2 = select_solution();
      blocked_father[solution_ID_2] = 1;

      if (((float)randGA.randomizeAPI(10000))/10000< 0.5)
      { blocked_father[solution_ID_1] = 0;
      }
      else
      { blocked_father[solution_ID_2] = 0;
      }
      if (((float)randGA.randomizeAPI(10000))/10000< 0.5)
      { blocked_father[0] = 1;
      }
      else
      { blocked_father[0] = 0;
      }
      num_forbidden = 0;
      for (cont2 = 0; cont2 < num_seq; cont2++)
      {    if (blocked_father[cont2] == 1)
           { num_forbidden++;
           }
      }
      if (num_forbidden > (int)((float)num_seq*(float)0.7))
      {   for (cont1 = 0; cont1 < num_seq; cont1++)
          { blocked_father[cont1] = 0;
          }
          num_forbidden = 0;
       }
      for (cont2 = 0; cont2 < size; cont2++)
      {    if (((float)randGA.randomizeAPI(10000))/10000< 0.5)
           { chromosome_aux[cont2] = population_main[solution_ID_1].column[cont2];
           }
           else
           { chromosome_aux[cont2] = population_main[solution_ID_2].column[cont2];
           }
      }
      for (cont2 = 0; cont2 < size; cont2++)
      { solution_passing[cont2] = chromosome_aux[cont2];
      }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::include_solution(int index)
{     int                cont1, cont2;
      float                    fit;
```

```
    int                 new;

    for (cont1 = 0; cont1 < index-1; cont1++)
    {   new = 1;
        for (cont2 = 0; cont2 < size; cont2++)
        { solution_global[cont2] = population_int[cont1].column[cont2];
        }
        fit = calculate_fitness(0,0);
        for (cont2 = 0; cont2 < num_seq; cont2++)
        {       if (fitness_main[cont2] == fit)
                { new = 0;
                }
        }
        if (new)
        {       for (cont2 = 0; cont2 < size; cont2++)
                {       population_main[ordered[cont1]].column[cont2] = solution_global[cont2];
                        fitness_main[ordered[cont1]] = fit;
                }
        }
    }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::mutation_swap( int n_swaps)
{   int     cont2;
    int position1, position2; // mutation position
    int       passing;

    for (cont2 = 0; cont2 < n_swaps; cont2++)
    {   position1 = randGA.randomizeAPI(size);
        position2 = randGA.randomizeAPI(size);
        passing = solution_global[position2];
        solution_global[position2] = solution_global[position1];
        solution_global[position1] = passing;
    }
    for (cont2 = 0; cont2 < size; cont2++)
    {   solution_passing[cont2] = solution_global[cont2];
    }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::fc_verify(void)
{   int cont1, cont2;
    for (cont1 = 0; cont1 < num_seq; cont1++)
    {   if(fitness_main[cont1] < bestfitness)
        {       bestfitness = fitness_main[cont1];
                fitness_main[0] = fitness_main[cont1];
                for (cont2 = 0; cont2 < size; cont2++)
                {       best_chromosome[cont2] = population_main[cont1].column[cont2];
                        population_main[0].column[cont2] = population_main[cont1].column[cont2];
                        solution_global[cont2] = best_chromosome[cont2];
                }
                calculate_fitness(1, 1);
        }
    }
    for (cont2 = 0; cont2 < size; cont2++)
    {   population_main[0].column[cont2] = best_chromosome[cont2];
    }
    fitness_main[0] = bestfitness;
}
//*************************************************************************//
```

```
//*************************************************************************//
void CrobotDecisionEvolutionary::constAGSGBD(void)
{       int inletID, organID;

        for(organID=0;organID<Complexity;organID++)
        {       // maximum inletID
                organInlet2[organID].max=organ[organID].max= setup->max[organID];
                // output inletID
                organInlet2[organID].output=organ[organID].output= setup->output[organID];
                // input inletID
                organInlet2[organID].input=organ[organID].input= setup->input[organID];
                // lower_bound inletID
                organInlet2[organID].lb=organ[organID].lb= setup->lb[organID];
                // upper_bound inletID
                organInlet2[organID].ub=organ[organID].ub= setup->ub[organID];
                // mean_day inletID
                organInlet2[organID].mean_day=organ[organID].mean_day= setup->meanDay[organID];
        }
        // start up nutritional levels currentTime zero
        for(inletID=0;inletID<size;inletID++)
        {       organInlet2[inletID].vol=organ[inletID].vol=(organ[inletID].max*InitInletProteinLevel);
                organInlet2[inletID].reagent=0;
        }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::intSGBD(void)
{       int organID;
        // output is the active varible from each agent, thus:
        //              agent saver(1)=  active<<output(++)>>; passive<<input (--)>>
        //              agent Decreaser(0)= active<<output(--)>>; passive<<input (++)>>
        for(organID=0;organID<Complexity;organID++)
        {       // mean output inletID
                organ[0+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[1+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[2+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[3+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[4+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[5+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[6+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[7+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[8+setup->tenOrgansInlet*organID].output= setup->output[organID];
                organ[9+setup->tenOrgansInlet*organID].output=setup->output[organID];
                // mean input inletID
                organ[0+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[1+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[2+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[3+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[4+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[5+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[6+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[7+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[8+setup->tenOrgansInlet*organID].input= setup->input[organID];
                organ[9+setup->tenOrgansInlet*organID].input= setup->input[organID];
                // maximum inletID
                organ[0+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[1+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[2+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[3+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[4+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[5+setup->tenOrgansInlet*organID].max= setup->max[organID];
```

```
                organ[6+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[7+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[8+setup->tenOrgansInlet*organID].max= setup->max[organID];
                organ[9+setup->tenOrgansInlet*organID].max= setup->max[organID];

                // lower bound inletID
                organ[0+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[1+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[2+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[3+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[4+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[5+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[6+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[7+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[8+setup->tenOrgansInlet*organID].lb= setup->lb[organID];
                organ[9+setup->tenOrgansInlet*organID].lb=setup->lb[organID];

                // upper bound  inletID
                organ[0+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[1+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[2+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[3+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[4+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[5+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[6+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[7+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[8+setup->tenOrgansInlet*organID].ub= setup->ub[organID];
                organ[9+setup->tenOrgansInlet*organID].ub=setup->ub[organID];

                // mean_day inletID
                organ[0+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[1+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[2+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[3+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[4+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[5+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[6+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[7+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                organ[8+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];
                        organ[9+setup->tenOrgansInlet*organID].mean_day= setup->meanDay[organID];

        }
        // start volume currentTime zero
        for(inletID=0;inletID<Complexity;inletID++)
        {       organ[inletID].vol=(int)(organ[inletID].max*(float)InitInletProteinLevel);
                organ[inletID].reagent=0;
        }
}
//*************************************************************************//
//*************************************************************************//
void CRobotDecisionEvolutionary::Init_xTotal_t(void)
{  int inletID,t;
   for(inletID=0;inletID<Complexity;inletID++)
   {    organ[inletID].xTotal=0;
        for(t=0;t<4;t++)
        {  organ[inletID].x_t[t]=0;
        }
   }
}
//*************************************************************************//
//*************************************************************************//
```

```
void CRobotDecisionEvolutionary::Excell(int breakTime, int pack, int currentTime_xls,int goal_NonLinear)
{	int timer,inletID,smallerVol,higherVol,Histogram[101];
	int negative,negativeAmount;
	int transbord,transbordAmount;
	FILE *writeout;
	negative=negativeAmount=0;
	transbord=transbordAmount=0;
	for(inletID=0;inletID<101;inletID++)
	{	Histogram[inletID]=0;
	}
	//---------------------------------------------------------------------/
	// itoa : convert int to char
	char protectHealth[5];
	char probl[5];
	char fileName[15]={"Results"};
	char extension[5];
	for(int i=0;i<setup->sizeExtension;i++)
	{	extension[i]=setup->extension[i];
	}
	sprintf(protectHealth,"%d",Agent);
	strcat(fileName,protectHealth);
	sprintf(probl,"%d",pack);
	strcat(fileName,probl);
	strcat(fileName,extension);
	//---------------------------------------------------------------------/
	writeout=fopen(fileName,"w");
	fprintf(writeout,"\ttime\t");
	for(timer=0;timer<breakTime-1;timer++)
	{ fprintf(writeout,"%d\t",timer+1+currentTime_xls-breakTime+1);
	}
	fprintf(writeout,"\n");
	fprintf(writeout,"\tPL\t");
	for(timer=0;timer<breakTime-1;timer++)
	{ fprintf(writeout,"%d\t",Result[timer].goal_nonInteger);
	}
	fprintf(writeout,"\n");
	fprintf(writeout,"\tAG\t");
	for(timer=0;timer<breakTime-1;timer++)
	{ fprintf(writeout,"%d\t",Result[timer].goal_integer);
	}
	fprintf(writeout,"\n");
	fprintf(writeout,"goal_nonInteger: %d\n",goal_NonLinear);
	fprintf(writeout,"type:%d\n",type);
	fprintf(writeout,"Complexity:%d\n",Complexity);
	fprintf(writeout,"mean_AG_count_LOOP  :  %.2f\n",mean_AG_count_LOOP);
	fprintf(writeout,"genetic_timing:  %l\n",genetic_timing);
	fprintf(writeout,"Ongans Inlet\n");
	if(!pack)
	{	for(inletID=0;inletID<size;inletID++)
		{ // <1> or <0> based on the previous state
		  organ[inletID].init_Zero=0;
		  organ[inletID].restarted=0; // receive the previous total of sum
		}
	}
	//-------------------------------------------------------//
	for(inletID=0;inletID<size;inletID++)
	{ fprintf (writeout," %d\t",inletID);
		fprintf (writeout,"delivery\t");
		for(timer=0;timer<breakTime-1;timer++)
		{	fprintf(writeout,"%d\t",Result[timer].ON_OFF[inletID]);
			if(organ[inletID].init_Zero) //  (!Result[timer].ON_OFF[inletID-1])
```

```
                        {       if(Result[timer].ON_OFF[inletID])
                                {  organ[inletID].restarted++;
                                }
                        }
                        organ[inletID].init_Zero=Result[timer].ON_OFF[inletID];
                }
                fprintf(writeout,"\n");
                fprintf (writeout,"\tProteinLevelLevel\t");
                for(timer=0;timer<breakTime-1;timer++)
                {       fprintf(writeout,"%d\t",Result[timer].Vol_Perc[inletID]);
                        if((Result[timer].Vol_Perc[inletID]<0)||(Result[timer].Vol_Perc[inletID]>100))
                        {       if(Result[timer].Vol_Perc[inletID]<0)
                                { negative++;
                                  negativeAmount+=abs(Result[timer].Vol_Perc[inletID]);
                                }
                                else
                                { transbord++;
                                  transbordAmount+=Result[timer].Vol_Perc[inletID];
                                }
                        }
                        else
                        {       Histogram[Result[timer].Vol_Perc[inletID]]++;
                        }
                }
                fprintf(writeout,"\n");
        }
        //-------------------------------------------------------//
        fprintf (writeout,"\tlowerProteinLevel\t");
        for(timer=0;timer<breakTime-1;timer++)
        {       smallerVol=200;
                for(inletID=0;inletID<size;inletID++)
                {       if(smallerVol>Result[timer].Vol_Perc[inletID])
                        {       smallerVol=Result[timer].Vol_Perc[inletID];
                        }
                }
                fprintf(writeout,"%d\t",smallerVol);
        }
        fprintf(writeout,"\n");
        fprintf (writeout,"\thigherProteinLevel\t");
        for(timer=0;timer<breakTime-1;timer++)
        {       higherVol=0;
                for(inletID=0;inletID<size;inletID++)
                {       if(higherVol<Result[timer].Vol_Perc[inletID])
                        {       higherVol=Result[timer].Vol_Perc[inletID];
                        }
                }
                fprintf(writeout,"%d\t",higherVol);
        }
        fprintf(writeout,"\n");
        //-------------------------------------------------------//
        // histogram
        fprintf(writeout,"\nHistogram\n");
        for(inletID=0;inletID<101;inletID++)
        { fprintf(writeout,"%d\t",inletID);
        }
        fprintf(writeout,"\n");
        for(inletID=0;inletID<101;inletID++)
        { fprintf(writeout,"%d\t",Histogram[inletID]);
        }
        fprintf(writeout,"\n\nnegative \tProteinLevel \tfrequency =\t %d\tamount =\t
%d\n",negative,negativeAmount);
```

```
        fprintf(writeout,"\ntransbord \tProteinLevel \tfrequency =\t %d\t          amount =\t
%d\n",transbord,transbordAmount);
        //-------------------------------------------------//
        // show the total of long time ago inletID not actived
        fprintf(writeout,"\n\n show the total of long time ago inletID not actived\n");
        fprintf(writeout,"\n\n");
        for(inletID=0;inletID<size;inletID++)
        {  fprintf(writeout,"%d\t",inletID);
        }
        fprintf(writeout,"\n");
        for(inletID=0;inletID<size;inletID++)
        {  fprintf(writeout,"%d\t",organ[inletID].avoidEverOFF);
        }
        fprintf(writeout,"\n\n");
        //-------------------------------------------------//
        // control of last activation
        fprintf(writeout,"\n\n\n total of initialization by inletID && final state\n\n ");
        for(inletID=0;inletID<size;inletID++)
        { fprintf(writeout,"%d\t\t %d\n",organ[inletID].restarted,organ[inletID].init_Zero);
        }
        fclose(writeout);
}
//*************************************************************************//
// End of ClassGenetic Specification
//*************************************************************************//
//*************************************************************************//
```

Appendix C

Parallel Processing

Source Code

The implemented class was designated ***CParallelManager***, which is responsible for the management of the shared data by a set of distinct components that run in parallel. Among other events, it activates and runs the method ***adaptiveAPI*** from the class ***CrobotDecisionSensing***, which interacts with motion control for agent simulation – see it at ***neuralAPI*** method from the class ***CNeural***. The source codes for both classes are shown respectively in Appendix A and C. As can be observed through the source code, parallel processing synchronizes the events running dynamically in parallel, and also generates the random database used in the 3D virtual environment. Key aspects and concepts of system architecture related to parallel processing were described in Chapter 7.

Start: Parallel Processing

```
//*************************************************************************//
//*************************************************************************//
// Activate parallel processing
//*************************************************************************//
//*************************************************************************//
#include <windows.h>
// library for parallel processing: process.h
#include <process.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
//*************************************************************************//
//*************************************************************************//
#include "CSetup.h"
CSetup setup;
#include "CParallelManager.h"
CParallelManager criticSection;
#include "CSetPosition.h"
#include "CNeural.h"
```

```
#include "CAdaptive.h"

#include "CcreatePosition.h"

CAdaptive adaptiveControl[setup.n_nanorobots];
CSetPosition locate;
CcreatePosition moleculePosition,obstaclesPosition;
CNeural neuralMotion[setup.n_nanorobots];
//*************************************************************************//
//*************************************************************************//
void InletPositionThread(void* pParams);
void moleculePositionThread(void* pParams);
void obstaclesPositionThread(void* pParams);
void activateAgent(void* pParams);
void agentMotionControl(void* pParams);
//*************************************************************************//
//*************************************************************************//
void initMultithreading(void);
//*************************************************************************//
//*************************************************************************//
void initMultithreading(void)
{       int shootDown=0;
        setup.SetupAPI();

        //---------------------------------------//
        criticSection.initData(setup.NumberInlettotal);

        criticSection.firstwaitworld3D();

        // Create Environment Coordinates
        _beginthread(InletPositionThread,0,NULL);
        shootDown++;

        // Create molecule startPosition to be Assembled
        _beginthread(moleculePositionThread,0,NULL);
        shootDown++;

        // Create obstacles startPosition
        _beginthread(obstaclesPositionThread,0,NULL);
        shootDown++;
        // Start Adaptive Agents
        setup.agentType=0;
        for(i=0;i<setup.n_nanorobots;i++)
        {       setup.nanorobotID=i;
                _beginthread(activateAgent,0,NULL);
                shootDown++;
                _beginthread(agentMotionControl,0,NULL);
                shootDown++;

                if(setup.agentType)
                {       setup.agentType=0;
                }
                else
                {       setup.agentType=1;
                }
        }

        criticSection.initkeepSimulator(shootDown);
        shootDown=0;

        shootDown=criticSection.keepSimulator(shootDown);
```

```
}
//*************************************************************************//
//*************************************************************************//
void moleculePositionThread(void* pParams)
{
        moleculePosition.positionAPI(setup.n_molecule,
                                        setup.minDistance_molecule,
                                        setup.objMOl, setup.distanceRadiusMol);
}
//*************************************************************************//
//*************************************************************************//
void obstaclesPositionThread(void* pParams)
{
        obstaclesPosition.positionAPI(setup.n_obstacle,
                                        setup.minDistance_obstacle,
                                        setup.objObst, setup.distanceRadiusObst);
}
//*************************************************************************//
//*************************************************************************//
void InletPositionThread(void* pParams)
{
        locate.positionAPI();
}
//*************************************************************************//
//*************************************************************************//
void activateAgent(void* pParams)
{
        adaptiveControl[setup.nanorobotID].adaptiveAPI(setup.agentType);
}
//*************************************************************************//
//*************************************************************************//
void agentMotionControl(void* pParams)
{       neuralMotion[setup.nanorobotID].neuralAPI(setup.nanorobotID);
}
//*************************************************************************//
//*************************************************************************//
```

Class: CParallelManager

```
//*************************************************************************//
//*************************************************************************//
// FILE CParallelManager.h
//*************************************************************************//
//*************************************************************************//
CRITICAL_SECTION cs;
//*************************************************************************//
//*************************************************************************//
class CParallelManager
{
        private:

                int Sequencecontrol;

                int wait1_initworld3D;
                int wait2_initworld3D;
                int wait3_initworld3D;
                int wait4_initworld3D;

                int waitNanorobotProteinIncrease;
                int waitNanorobotProteinDecrease;
```

```
        int sizeworld3D;

        int RouteControlIncrease;
        int RouteControlDecrease;


        struct dataInlet
        {       // Protein levels on each Inlet and
                // the last "+" or "-" Protein update
                int positive,negative,Protein,wait,maxProtein;
        }*InletSocket;

        int *InletON3D;

        int nTotalAgent;
        struct agentTimer
        {       // Increase: 0; Decrease: 1; graphic: 2;
                int timerAdaptive;
                int timerNeural;
        }*setTime;

        struct locateObj3D
        {       int *x,*z,*y;
        };
        locateObj3D Inletposition;
        locateObj3D molPosition;
        locateObj3D obstaclePosition;

        struct locateBasexz
        {       int x,z,y;
        }basePosition,robotBasePosition;

        struct Inlet_trajectory
        {
                int *whatInlet;
                int totalInlet;
                int waitNeural;
        }IncreasePath, DecreasePath, IncreasePathSolution, DecreasePathSolution;

        int simulatorStatus;

        struct costTrajectory
        {
                int *NM;
        }*MyInletConnection,
        IncreaseBaseConnection,DecreaseBaseConnection;

        int distancesMatrix;

        FILE *verify_file,*Increaseoute,*DecreaseRoute;

public:

        CParallelManager() {};
        ~CParallelManager() {};

        int setupProtein(int timer, int agent,
                                        int TotalInlet,int nOperation,
                                        int *outputInlet, int *ProteinLevel,
                                        int *robotDecrease);
        void initProtein(int TotalInlet,int *ProteinLevelInlet,int *maxProteinLevel);
```

```
		void initData(int TotalInlet);
		void freeData(void);

		void putMoleculePosition(int kind_of_Obj,
					int *object_x,
					int *object_z,
					int *object_y,
					int objTotal,
					int sizeworld);

		void putInletPosition(int startBase, int *Inletposition_x,
				int *Inletposition_z,int *Inletposition_y,
				int InletTotal, int sizeworld);

		void putInletDistances(int n,int *distance, int InletTotal, int startBase);

		int initSocketNeural(int agent,int n, int *matrixlineNM,int *distanceBase);
		int setupNeural(int agent, int *ntotalInlet, int *on_what);
		int neuralRouteSolution(int *Inletsequence, int nInletinRoute,int agent);

		void initkeepSimulator(int shootDown);
		int keepSimulator(int shootDown);

		void firstwaitworld3D(void);
		int updateTimer3D(int agent, int timer);

		int initVirtualWorld(int *sizeworld,
				int *ProteinDecrease,int *ProteinIncrease,
				int moleculeTotal,
				int *molPos_x,int *molPos_z,int *molPos_y,
				int obstacleTotal,
				int *obstaclePos_x,int *obstaclePos_z,int *obstaclePos_y,
				int InletTotal,
				int *Inletposition_x,
				int *Inletposition_z,
				int *Inletposition_y,
				int *baseposition_x,
				int *baseposition_z,
				int *baseposition_y,
				int *robotBasePosition_x,
				int *robotBasePosition_z,
				int *robotBasePosition_y);

		int updateRoute3D(int *Inletsequence, int *totalinRoute,int agent);
		int updateProtein3D(int *ProteinLevel,int *InletON,int agent,int totalInlet);

		friend CSetup;
};
//*************************************************************************//
//*************************************************************************//
void CParallelManager::firstwaitworld3D(void)
{	int agent,i,j;

	EnterCriticalSection(&cs);

	for(i=0;i<setup.NumberInlettotal;i++)
	{	for(j=0;j<setup.NumberInlettotal;j++)
		{	MyInletConnection[i].NM[j]=0;
		}
	}
```

```
	wait1_initworld3D=1;
	wait2_initworld3D=1;
	wait3_initworld3D=1;
	wait4_initworld3D=1;

	IncreasePathSolution.waitNeural=1;
	DecreasePathSolution.waitNeural=1;

	waitNanorobotProteinIncrease=1;
	waitNanorobotProteinDecrease=1;

	for(agent=0;agent<nTotalAgent;agent++)
	{	setTime[agent].timerAdaptive=0;
		setTime[agent].timerNeural=0;
	}

	LeaveCriticalSection(&cs);
}
//*************************************************************************//
//*************************************************************************//
void CParallelManager::initProtein(int TotalInlet,int *ProteinLevelInlet,int *maxProteinLevel)
{	int Inlet;

	EnterCriticalSection(&cs);

	for(Inlet=0;Inlet<TotalInlet;Inlet++)
	{	InletSocket[Inlet].Protein=*ProteinLevelInlet++;
		InletSocket[Inlet].maxProtein=*maxProteinLevel++;
	}

	Sequencecontrol=0;
	wait4_initworld3D=0;
	IncreasePath.waitNeural=1;
	DecreasePath.waitNeural=1;

	LeaveCriticalSection(&cs);

}
//*************************************************************************//
//*************************************************************************//
void CParallelManager::initkeepSimulator(int shootDown)
{
	EnterCriticalSection(&cs);

	simulatorStatus=shootDown;

	LeaveCriticalSection(&cs);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::keepSimulator(int shootDown)
{
	EnterCriticalSection(&cs);

	simulatorStatus-=shootDown;

	LeaveCriticalSection(&cs);

	return(simulatorStatus);
}
```

```
//*************************************************************************//
//*************************************************************************//
int CParallelManager::initSocketNeural(int agent, int n, int *matrixlineNM, int *distanceBase)
{	int Inlet,wait;

	EnterCriticalSection(&cs);

	if(!MyInletConnection[setup.NumberInlettotal-1].NM[setup.NumberInlettotal-1])
	{	// Inlet position wasn't initialized
		wait=1;
	}
	else // last position in the matrix was completed
	{	wait=0;
		if(agent)
		{	for(Inlet=0;Inlet<setup.NumberInlettotal;Inlet++)
			{	(*matrixlineNM++)=MyInletConnection[n].NM[Inlet];
				(*distanceBase++)=IncreaseBaseConnection.NM[Inlet];
			}
		}
		else
		{	for(Inlet=0;Inlet<setup.NumberInlettotal;Inlet++)
			{	(*matrixlineNM++)=MyInletConnection[n].NM[Inlet];
				(*distanceBase++)=DecreaseBaseConnection.NM[Inlet];
			}
		}
	}
	LeaveCriticalSection(&cs);

	return(wait);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::setupNeural(int agent, int *ntotalInlet, int *on_what)
{	int wait,Inlet;

	EnterCriticalSection(&cs);

	wait=0;

	if(agent)
	{	if(!IncreasePath.waitNeural)
		{	IncreasePath.waitNeural=1;
			// to do goes here
			for(Inlet=0;Inlet<IncreasePath.totalInlet;Inlet++)
			{	(*on_what++)=IncreasePath.whatInlet[Inlet];
			}
			(*ntotalInlet)=IncreasePath.totalInlet;
		}
		else
		{	wait=1;
		}


	}
	else
	{	if(!DecreasePath.waitNeural)
		{	DecreasePath.waitNeural=1;
			// to do goes here
			for(Inlet=0;Inlet<DecreasePath.totalInlet;Inlet++)
			{	(*on_what++)=DecreasePath.whatInlet[Inlet];
			}
```

```
                        (*ntotalInlet)=DecreasePath.totalInlet;
                }
                else
                {       wait=1;
                }
        }

        LeaveCriticalSection(&cs);

        return(wait);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::updateTimer3D(int agent, int timer)
{       int updateTimer3D=0;

        EnterCriticalSection(&cs);

        if((setTime[agent].timerAdaptive-timer)&&(setTime[agent].timerNeural-timer))
        {       updateTimer3D=1;
        }

        LeaveCriticalSection(&cs);

        return(updateTimer3D);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::updateProtein3D(int *ProteinLevel,int *InletON,int agent,int totalInlet)
{       int wait,i;

        EnterCriticalSection(&cs);

        if(agent)
        {       wait=waitNanorobotProteinIncrease;

                if(!waitNanorobotProteinIncrease)
                {       for(i=0;i<totalInlet;i++)
                        {       // percentual ProteinLevel
                                (*ProteinLevel++)=InletSocket[i].Protein*100/InletSocket[i].maxProtein;
                                (*InletON++)=InletON3D[i];
                        }
                        waitNanorobotProteinIncrease=1;
                }
        }
        else
        {       wait=waitNanorobotProteinDecrease;

                if(!waitNanorobotProteinDecrease)
                {
                        for(i=0;i<totalInlet;i++)
                        {
                                (*ProteinLevel++)= InletSocket[i].Protein * 100/
                                                        InletSocket[i].maxProtein;
                                (*InletON++)=InletON3D[i];
                        }
                        waitNanorobotProteinDecrease=1;
                }
        }

        LeaveCriticalSection(&cs);
```

```
	return(wait);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::setupProtein(int timer, int agent, int TotalInlet,int nOperation,
                                   int *outputInlet, int *ProteinLevel,int *robotDecrease)
{	int Inlet,wait,numberInlet;

	EnterCriticalSection(&cs);

	wait=0;
	numberInlet=0;

	if(agent)
	{	// Agent Increase
		// wait if the agent Decrease hasn't visited
		if(Sequencecontrol && IncreasePath.waitNeural && waitNanorobotProteinIncrease)
		{	// Neural control
			IncreasePath.totalInlet=0;

			for(Inlet=0;Inlet<TotalInlet;Inlet++)
			{	InletSocket[Inlet].positive=((*outputInlet++)*nOperation);
				InletSocket[Inlet].Protein+=InletSocket[Inlet].positive;
				(*ProteinLevel++)=InletSocket[Inlet].Protein;

				// Neural Control
				// inform inlets to be attended in the motion control
				InletON3D[Inlet]=0;
				if(InletSocket[Inlet].positive)
				{	IncreasePath.whatInlet[IncreasePath.totalInlet++]=Inlet;
					InletON3D[Inlet]=1;
				}

				// I need to inform what was the last negative setup
				// to the Increase choose the better next action plane
				// so, it returns for the pointer in the AG
				// the action performed by the robotDecrease
				(*robotDecrease++)=InletSocket[Inlet].negative;

			}
			setTime[agent].timerAdaptive++;
			Sequencecontrol=0;
			IncreasePath.waitNeural=0;

			waitNanorobotProteinIncrease=0;
		}
		else
		{	wait=1;
		}
	}
	else
	{	// Agent Decrease

		// wait if the agent Increase hasn't finished
		if((!Sequencecontrol) && DecreasePath.waitNeural &&
                                        waitNanorobotProteinDecrease)
		{	// Neural Control
			DecreasePath.totalInlet=0;

			for(Inlet=0;Inlet<TotalInlet;Inlet++)
```

```
                        {       InletSocket[Inlet].negative=((*outputInlet++)*nOperation);
                                InletSocket[Inlet].Protein= InletSocket[Inlet].Protein -
                                                        InletSocket[Inlet].negative;
                                (*ProteinLevel++)=InletSocket[Inlet].Protein;

                                // Neural Control
                                // inform inlets to be attend in the motion control
                                InletON3D[Inlet]=0;
                                if(InletSocket[Inlet].negative)
                                {       DecreasePath.whatInlet[DecreasePath.totalInlet++]=Inlet;
                                        InletON3D[Inlet]=1;
                                }

                                // it return for the pointer in the AG
                                // the action performed by the robotDecrease
                                (*robotDecrease++)=InletSocket[Inlet].positive;

                        }
                        setTime[agent].timerAdaptive++;
                        Sequencecontrol=1;
                        DecreasePath.waitNeural=0;

                        waitNanorobotProteinDecrease=0;
                }
                else
                {       wait=1;
                }
        }

        LeaveCriticalSection(&cs);

        return(wait);
}
//*************************************************************************//
//*************************************************************************//
void CParallelManager::freeData(void)
{       int i;

        EnterCriticalSection(&cs);

        for(i=0;i<setup.NumberInlettotal;i++)
        {       free(MyInletConnection[i].NM);
        }
        free(MyInletConnection);

        free(InletSocket);
        free(setTime);

        free(IncreasePath.whatInlet);
        free(DecreasePath.whatInlet);
        free(IncreasePathSolution.whatInlet);
        free(DecreasePathSolution.whatInlet);

        free(IncreaseBaseConnection.NM);
        free(DecreaseBaseConnection.NM);
        free(Inletposition.x);
        free(Inletposition.z);
        free(Inletposition.y);

        free(molPosition.x);
        free(molPosition.z);
```

```
	free(molPosition.y);

	free(obstaclePosition.x);
	free(obstaclePosition.z);
	free(obstaclePosition.y);

	free(InletON3D);

	LeaveCriticalSection(&cs);

}
//*************************************************************************//
//*************************************************************************//
void CParallelManager::initData(int TotalInlet)
{
	int i;

	EnterCriticalSection(&cs);

	nTotalAgent=setup.numberAgent;
	//-----------------------------------------------/
	InletSocket=(struct dataInlet*)malloc(
			TotalInlet*sizeof(struct dataInlet));
	if(InletSocket == NULL)
	{ printf("\n not enough memory");
	  exit(1);
	}
	//-----------------------------------------------/
	setTime=(struct agentTimer*)malloc(
			nTotalAgent*sizeof(struct agentTimer));
	if(setTime == NULL)
	{ printf("\n not enough memory");
	  exit(1);
	}
	//-----------------------------------------------/
	IncreasePath.whatInlet=(int *)malloc(setup.NumberInlettotal*sizeof(int));
	if(IncreasePath.whatInlet == NULL)
	{ printf("\n not enough memory");
	  exit(1);
	}
	//-----------------------------------------------/
	DecreasePath.whatInlet=(int *)malloc(setup.NumberInlettotal*sizeof(int));
	if(DecreasePath.whatInlet == NULL)
	{ printf("\n not enough memory");
	  exit(1);
	}
	//-----------------------------------------------/
	IncreasePathSolution.whatInlet=(int *)malloc(setup.NumberInlettotal*sizeof(int));
	if(IncreasePathSolution.whatInlet == NULL)
	{ printf("\n not enough memory");
	  exit(1);
	}
	//-----------------------------------------------/
	DecreasePathSolution.whatInlet=(int *)malloc(setup.NumberInlettotal*sizeof(int));
	if(DecreasePathSolution.whatInlet == NULL)
	{ printf("\n not enough memory");
	  exit(1);
	}
	//-----------------------------------------------/
	MyInletConnection=(struct costTrajectory *)
				malloc(setup.NumberInlettotal
```

```
                                *sizeof(struct costTrajectory));
if(MyInletConnection==NULL)
{       printf("\n not enough memory");
        exit(1);
}
for(i=0;i<setup.NumberInlettotal;i++)
{
        MyInletConnection[i].NM = (int *)malloc(
                                setup.NumberInlettotal*sizeof(int));
        if(MyInletConnection[i].NM == NULL)
        { printf("\n not enough memory");
          exit(1);
        }
}
//-----------------------------------------------------/
IncreaseBaseConnection.NM=(int *)malloc(setup.NumberInlettotal*sizeof(int));
if(IncreaseBaseConnection.NM == NULL)
{ printf("\n not enough memory");
  exit(1);
}
//-----------------------------------------------------/
DecreaseBaseConnection.NM=(int *)malloc(setup.NumberInlettotal*sizeof(int));
if(DecreaseBaseConnection.NM == NULL)
{ printf("\n not enough memory");
  exit(1);
}
//-----------------------------------------------------/
molPosition.x=(int *)malloc(setup.n_molecule*sizeof(int));
if(molPosition.x == NULL)
{ printf("\n not enough memory");
  exit(1);
}
molPosition.z=(int *)malloc(setup.n_molecule*sizeof(int));
if(molPosition.z == NULL)
{ printf("\n not enough memory");
  exit(1);
}
molPosition.y=(int *)malloc(setup.n_molecule*sizeof(int));
if(molPosition.y == NULL)
{ printf("\n not enough memory");
  exit(1);
}
//-----------------------------------------------------/
obstaclePosition.x=(int *)malloc(setup.n_obstacle*sizeof(int));
if(obstaclePosition.x == NULL)
{ printf("\n not enough memory");
  exit(1);
}
obstaclePosition.z=(int *)malloc(setup.n_obstacle*sizeof(int));
if(obstaclePosition.z == NULL)
{ printf("\n not enough memory");
  exit(1);
}
obstaclePosition.y=(int *)malloc(setup.n_obstacle*sizeof(int));
if(obstaclePosition.y == NULL)
{ printf("\n not enough memory");
  exit(1);
}
//-----------------------------------------------------/
Inletposition.x=(int *)malloc(setup.NumberInlettotal*sizeof(int));
if(Inletposition.x == NULL)
```

```
        { printf("\n not enough memory");
          exit(1);
        }
        Inletposition.z=(int *)malloc(setup.NumberInlettotal*sizeof(int));
        if(Inletposition.z == NULL)
        { printf("\n not enough memory");
          exit(1);
        }
        Inletposition.y=(int *)malloc(setup.NumberInlettotal*sizeof(int));
        if(Inletposition.y == NULL)
        { printf("\n not enough memory");
          exit(1);
        }
        //-------------------------------------------------/
        InletON3D=(int *)malloc(setup.NumberInlettotal*sizeof(int));
        if(InletON3D == NULL)
        { printf("\n not enough memory");
          exit(1);
        }
        //-------------------------------------------------/
        LeaveCriticalSection(&cs);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::initVirtualWorld(int *sizeworld,
                        int *ProteinDecrease,int *ProteinIncrease,
                        int     moleculeTotal,
                        int     *molPos_x,int *molPos_z,int *molPos_y,
                        int     obstacleTotal,
                        int     *obstaclePos_x,int *obstaclePos_z,int *obstaclePos_y,
                        int InletTotal,
                        int *Inletposition_x,
                        int *Inletposition_z,
                        int *Inletposition_y,
                        int *baseposition_x,
                        int *baseposition_z,
                        int *baseposition_y,
                        int *robotBasePosition_x,
                        int *robotBasePosition_z,
                        int *robotBasePosition_y)
{
   EnterCriticalSection(&cs);

        int wait = wait1_initworld3D + wait2_initworld3D +
                   wait3_initworld3D + wait4_initworld3D;

        // initialize the world object position in 3D
        if(wait)
        {       // wait, the position wasn't generated
        }
        else
        {       int j;

                (*sizeworld)=sizeworld3D;
                //-----------------------------------------//
                // initialize the molecule to be assembled points' coordinates
                for(j=0;j<obstacleTotal;j++)
                {       (*obstaclePos_x++)=obstaclePosition.x[j];
                        (*obstaclePos_z++)=obstaclePosition.z[j];
                        (*obstaclePos_y++)=obstaclePosition.y[j];
                }
```

```
                //-----------------------------------------//
                // initialize the molecule to be assembled points' coordinates
                for(j=0;j<moleculeTotal;j++)
                {       (*molPos_x++)=molPosition.x[j];
                        (*molPos_z++)=molPosition.z[j];
                        (*molPos_y++)=molPosition.y[j];
                }
                //-----------------------------------------//
                // initialize the delivery points' coordinates
                for(j=0;j<InletTotal;j++)
                {       (*Inletposition_x++)=Inletposition.x[j];
                        (*Inletposition_z++)=Inletposition.z[j];
                        (*Inletposition_y++)=Inletposition.y[j];
                        // Protein_level in percentual
                        (*ProteinDecrease)=InletSocket[j].Protein*100/InletSocket[j].maxProtein;
                        (*ProteinIncrease++)=InletSocket[j].Protein*100/InletSocket[j].maxProtein;
                }
                //-----------------------------------------//
                // agent start position
                (*baseposition_x)=basePosition.x;
                (*baseposition_z)=basePosition.z;
                (*baseposition_y)=basePosition.y;
                // reagent start position
                (*robotBasePosition_x)=robotBasePosition.x;
                (*robotBasePosition_z)=robotBasePosition.z;
                (*robotBasePosition_y)=robotBasePosition.y;
                //-----------------------------------------//

        }

        LeaveCriticalSection(&cs);

        return(wait);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::updateRoute3D(int *Inletsequence, int *totalinRoute,int agent)
{       int wait;

        EnterCriticalSection(&cs);

        // update dynamically the world state

        // update the Protein levels

        // update the trajectory route for Decrease and the Increase agents
        if(agent)
        {
                if(!IncreasePathSolution.waitNeural)
                {       (*totalinRoute)=IncreasePathSolution.totalInlet;
                        for(int i=0;i<IncreasePathSolution.totalInlet;i++)
                        {       (*Inletsequence++)=IncreasePathSolution.whatInlet[i];
                        }
                        IncreasePathSolution.waitNeural=1;
                        wait=0;
                }
                else
                {       wait=1;
                }
        }
        else
```

```
	{
		if(!DecreasePathSolution.waitNeural)
		{	(*totalinRoute)=DecreasePathSolution.totalInlet;
			for(int i=0;i<DecreasePathSolution.totalInlet;i++)
			{	(*Inletsequence++)=DecreasePathSolution.whatInlet[i];
			}
			DecreasePathSolution.waitNeural=1;
			wait=0;
		}
		else
		{	wait=1;
		}
	}

	LeaveCriticalSection(&cs);

	return(wait);
}
//*************************************************************************//
//*************************************************************************//
int CParallelManager::neuralRouteSolution(int *Inletsequence, int nInletinRoute,int agent)
{	int wait;

	EnterCriticalSection(&cs);

	if(agent)
	{
		if(IncreasePathSolution.waitNeural)
		{
			IncreasePathSolution.totalInlet=nInletinRoute;
			for(int i=0;i<IncreasePathSolution.totalInlet;i++)
			{	IncreasePathSolution.whatInlet[i]=(*Inletsequence++);
			}
			wait=IncreasePathSolution.waitNeural=0;
			setTime[agent].timerNeural++;
		}
		else
		{	wait=IncreasePathSolution.waitNeural;
		}
	}
	else
	{
		if(DecreasePathSolution.waitNeural)
		{
			DecreasePathSolution.totalInlet=nInletinRoute;
			for(int i=0;i<DecreasePathSolution.totalInlet;i++)
			{	DecreasePathSolution.whatInlet[i]=(*Inletsequence++);
			}
			wait=DecreasePathSolution.waitNeural=0;
			setTime[agent].timerNeural++;
		}
		else
		{	wait=DecreasePathSolution.waitNeural;
		}
	}

	LeaveCriticalSection(&cs);

	return(wait);
}
//*************************************************************************//
```

```
//*************************************************************************//
void CParallelManager::putInletPosition(int startBase,
                                        int *Inletposition_x,
                                        int *Inletposition_z,
                                        int *Inletposition_y,
                                        int InletTotal,
                                        int sizeworld)
{
   EnterCriticalSection(&cs);

        sizeworld3D=sizeworld;

        switch(startBase)
        {       case 0:
                {
                        for(int j=0;j<InletTotal;j++)
                        {       Inletposition.x[j]=(*Inletposition_x++);
                                Inletposition.z[j]=(*Inletposition_z++);
                                Inletposition.y[j]=(*Inletposition_y++);
                        }
                        break;
                }
                case 1:
                {
                        basePosition.x=(*Inletposition_x);
                        basePosition.z=(*Inletposition_z);
                        basePosition.y=(*Inletposition_y);

                        break;
                }
                case 2:
                {
                        robotBasePosition.x=(*Inletposition_x);
                        robotBasePosition.z=(*Inletposition_z);
                        robotBasePosition.y=(*Inletposition_y);

                        wait1_initworld3D=0;

                        break;
                }
        }

        LeaveCriticalSection(&cs);
}
//*************************************************************************//
//*************************************************************************//
void CParallelManager::putMoleculePosition(int kind_of_Obj,
                                           int *object_x,
                                           int *object_z,
                                           int *object_y,
                                           int objTotal,
                                           int sizeworld)
{       int j;

        EnterCriticalSection(&cs);

        switch(kind_of_Obj)
        {       case 0:
                {       // molecules to be assembled
                        for(int j=0;j<objTotal;j++)
                        {       molPosition.x[j]=(*object_x++);
```

```
                                molPosition.z[j]=(*object_z++);
                                molPosition.y[j]=(*object_y++);
                                wait2_initworld3D=0;
                        }
                        break;
                }
                case 1:
                {       // obstacles
                        for(int j=0;j<objTotal;j++)
                        {       obstaclePosition.x[j]=(*object_x++);
                                obstaclePosition.z[j]=(*object_z++);
                                obstaclePosition.y[j]=(*object_y++);
                                wait3_initworld3D=0;
                        }
                        break;
                }
                case 2:
                {       // other objects
                        break;
                }
        }
        LeaveCriticalSection(&cs);
}
//*************************************************************************//
//*************************************************************************//
void CParallelManager::putInletDistances(int n,int *distance,int InletTotal, int startBase)
{       int j;

        EnterCriticalSection(&cs);

        switch(startBase)
        {
                case 0:
                {       for(j=0;j<InletTotal;j++)
                        {       DecreaseBaseConnection.NM[j]=distance[j];
                        }
                        break;
                }
                case 1:
                {       for(j=0;j<InletTotal;j++)
                        {       IncreaseBaseConnection.NM[j]=distance[j];
                        }
                        break;
                }
                case 2:
                {       for(j=0;j<InletTotal;j++)
                        {       MyInletConnection[n].NM[j]=distance[j];
                        }
                        break;
                        // the distances was fully initialized
                }
        }
        LeaveCriticalSection(&cs);
}
//*************************************************************************//
//*************************************************************************//
```

Appendix D

Motion Control

Source Code

The sequence discloses the source code for the class ***CNeural*** that contains the dynamic feed-forward neural network used in motion control for simulated nanorobots. The mathematical model and neural network concepts were described in Chapter 5. The calculation is based on agents theory for each nanorobot interacting within the 3D workspace, and uses memory behavior and task decomposition for environment sensing as described in Chapter 8. The source code here interacts directly with the class ***CrobotDecisionSensing***, which is presented in Appendix A.

CLASS: CNeural

```
//*************************************************************************//
//*************************************************************************//
// Artificial Neural Networks
//*************************************************************************//
//*************************************************************************//
class CNeural
{
        private:
                int *whatInlet,*neuronSelect;
                int agent;
                int sizeSampleNeural;
                long   neuralTime;

                struct data_nm
                {       int *baseNM;
                }distance;

                struct distanceInlet
                {       int *nm;
                }*distanceInlet;

                struct sequenceNeural
                {       int *sequence;
                        int cost;
                }best,search;
```

```
                struct sampleCost
                {       int cost;
                }*sampleSearch;

                int *searchSequence;

                FILE *neuralMatrix;
                FILE *fileANN;

        public:
                CNeural() {}
                ~CNeural() {}

                // random methods
                CRandomize randNeural;

                int *InletMatrixDistance,*distanceSocketBase;
                int nInletRoute;

                void neuralAPI(int nanorobotID);
                void allocMemory(void);
                void initData(int nanorobotID);
                void freeMemory(void);
                void setupNeural(void);

                int neuralForward(void);
                int reckonNeuralCost(int output);
                void searchEngine(int timer);

                friend CCritical;

};
//*************************************************************************//
//*************************************************************************//
void CNeural::allocMemory(void)
{       int i;

        whatInlet=(int*)malloc(setup.NumberInlettotal*sizeof(int));
         if(whatInlet == NULL)
        {       printf("\n not enough memory for whatInlet");
                exit(1);
        }
        //-----------------------------------------------------//
        neuronSelect=(int*)malloc(setup.NumberInlettotal*sizeof(int));
         if(neuronSelect == NULL)
        {       printf("\n not enough memory for neuronSelect");
                exit(1);
        }
        //-----------------------------------------------------//
        InletMatrixDistance=(int*)malloc(setup.NumberInlettotal*sizeof(int));
        if(InletMatrixDistance == NULL)
        {       printf("\n not enough memory for InletMatrixDistance");
                exit(1);
        }
        //-----------------------------------------------------//
        distanceSocketBase=(int*)malloc(setup.NumberInlettotal*sizeof(int));
         if(distanceSocketBase == NULL)
        {       printf("\n not enough memory for distanceSocketBase");
                exit(1);
        }
```

```
	//-----------------------------------------------------//
	distance.baseNM=(int*)malloc(setup.NumberInlettotal*sizeof(int));
	if(distance.baseNM == NULL)
	{	printf("\n not enough memory for distance.baseNM");
		exit(1);
	}
	//-----------------------------------------------------//
	distanceInlet=(struct distanceInlet*)malloc(setup.NumberInlettotal*sizeof(
		struct distanceInlet));
	if(distanceInlet == NULL)
	{	printf("\n not enough memory for distanceInlet");
		exit(1);
	}
	for(i=0;i<setup.NumberInlettotal;i++)
	{	distanceInlet[i].nm=(int*)malloc(setup.NumberInlettotal*sizeof(int));
		if(distanceInlet[i].nm == NULL)
		{	printf("\n not enough memory for distanceInlet[i].nm",i);
			exit(1);
		}
	}
	//-----------------------------------------------------//
	searchSequence=(int*)malloc(setup.NumberInlettotal*sizeof(int));
	if(searchSequence == NULL)
	{	printf("\n not enough memory for searchSequence");
		exit(1);
	}
	//-----------------------------------------------------//
	best.sequence=(int*)malloc(setup.NumberInlettotal*sizeof(int));
	if(best.sequence == NULL)
	{	printf("\n not enough memory for best.sequence");
		exit(1);
	}
	//-----------------------------------------------------//
	search.sequence=(int*)malloc(setup.NumberInlettotal*sizeof(int));
	if(search.sequence == NULL)
	{	printf("\n not enough memory for best.sequence");
		exit(1);
	}
	//-----------------------------------------------------//
	sampleSearch=(struct sampleCost*)malloc(sizeSampleNeural*sizeof(
			struct sampleCost));
	 if(sampleSearch == NULL)
	{	printf("\n not enough memory for sampleSearch");
		exit(1);
	}
	//-----------------------------------------------------//
}
//*************************************************************************//
//*************************************************************************//
void CNeural::freeMemory(void)
{	free(whatInlet);
	free(InletMatrixDistance);
	free(distanceSocketBase);
	free(distance.baseNM);
	free(distanceInlet);
	free(searchSequence);
	free(sampleSearch);
	free(neuronSelect);
}
//*************************************************************************//
//*************************************************************************//
```

```
void CNeural::setupNeural(void)
{
	// determine how long to process in seconds
	neuralTime=setup.timeProcessingSeconds;
	// size of sizeSampleNeural
	sizeSampleNeural=100;
}
//*************************************************************************//
//*************************************************************************//
void CNeural::initData(int nanorobotID)
{
	int wait,j,Inlet;
	agent=nanorobotID;

	for(Inlet=0;Inlet<setup.NumberInlettotal;Inlet++)
	{
		do
		{	wait=criticSection.initSocketNeural(agent,
				Inlet,
				InletMatrixDistance,
				distanceSocketBase);
			if(wait)
			{	printf("\nsleep#2 agent = %d  initData ",agent);
				Sleep(10000);
			}
		}while(wait);

		for(j=0;j<setup.NumberInlettotal;j++)
		{	distanceInlet[Inlet].nm[j]=InletMatrixDistance[j];
			distance.baseNM[j]=distanceSocketBase[j];
		}
	}
}
//*************************************************************************//
//*************************************************************************//
void CNeural::neuralAPI(int nanorobotID)
{	int timer,wait,status;
	int dayID;

	setupNeural();
	allocMemory();
	initData(nanorobotID);

	for(dayID=0;dayID<setup.Ndays;dayID++)
	{
		//-----------------------------------------------------------------//
		// strings for the dynamic file name
		char saveLife[3];
		char probl[5];
		char file_name[25]={"resultANN"};

		char extension[5];
		for(int i=0;i<setup.sizeExtension;i++)
		{	extension[i]=setup.extension[i];
		}

		sprintf(saveLife,"%d",agent);

		strcat(file_name,saveLife);

		sprintf(probl,"%d",dayID);
```

```
                strcat(file_name,probl);

                strcat(file_name,extension);

                fileANN=fopen(file_name,"w");
                //-------------------------------------------------------------------//

                for(timer=0;timer<setup.Nhours;timer++)
                {
                        do
                        {       wait=criticSection.setupNeural(agent,&nInletRoute,whatInlet);
                                if(wait)
                                {   printf("\nsleep#3 agent = %d  time = %d",agent,timer);
                                        Sleep(15000);// sleep time in milliseconds
                                }
                        }while(wait);
                        searchEngine(timer);
                }
        }

        printf("\n executed neural - agent = %d",agent);

        fclose(fileANN);

        freeMemory();

        status=criticSection.keepSimulator(1);
}
//*************************************************************************//
//*************************************************************************//
void CNeural::searchEngine(int timer)
{
        int i,Inlet,solution,sample;
        time_t          time_begin, time_end;

        // do for “n” seconds (n = neuralTime) the Neural Forward
        time_begin = time(NULL);

        sample=0;
        best.cost=99000;

        do
        {       // make 100 network analyses and take the better result
                for(solution=0;solution<100;solution++)
                {       // take the result if it is better than prior
                        if(neuralForward())
                        {
                                int trash=reckonNeuralCost(1);

                                for(Inlet=0;Inlet<nInletRoute;Inlet++)
                                {       best.sequence[Inlet]=search.sequence[Inlet];
                                }
                                best.cost=search.cost;
                                sampleSearch[sample++].cost=search.cost;

                                fprintf(fileANN,"\n==========================\n");
                                fprintf(fileANN,"sequence\t");
                                for(i=0;i<nInletRoute;i++)
                                {       fprintf(fileANN,"%d\t",best.sequence[i]);
                                }
                                fprintf(fileANN,"\n route price\t");
```

```
                                fprintf(fileANN,"%d\n",best.cost);

                                fprintf(fileANN,"\n==========================\n");
                        }
                }

                time_end = time(NULL);
        }while(time_end - time_begin < neuralTime);

        for(i=0;i<sample;i++)
        {       fprintf(fileANN,"%d\n",sampleSearch[i].cost);
        }
        fprintf(fileANN,"\n===== BEST SOLUTION - timer: %d =======\n",timer);
        for(i=0;i<nInletRoute;i++)
        {       fprintf(fileANN,"%d\t",best.sequence[i]);
        }
        fprintf(fileANN,"\n%d\n",best.cost);
        fprintf(fileANN,"\n=======================================\n");

        criticSection.neuralRouteSolution(best.sequence,nInletRoute,agent);
}
//*************************************************************************//
//*************************************************************************//
int CNeural::neuralForward(void)
{       int Inlet,neuronActive,j,i;

        for(Inlet=0;Inlet<nInletRoute;Inlet++)
        {       neuronSelect[Inlet]=whatInlet[Inlet];
        }

        j=0;
        // generate a Sigmoid Belief Neural Network Solution
        for(Inlet=0;Inlet<nInletRoute;Inlet++)
        {       neuronActive=randNeural.randomizeAPI(nInletRoute-Inlet);
                // take the activated neurons
                search.sequence[j++]=neuronSelect[neuronActive];

                for(i=neuronActive;i<(nInletRoute-Inlet)-1;i++)
                {       neuronSelect[i]=neuronSelect[i+1];
                }
        }

        return(reckonNeuralCost(0));
}
//*************************************************************************//
//*************************************************************************//
int CNeural::reckonNeuralCost(int output)
{       int i;

        search.cost=0;
        // sum the cost relative at the start point to basePosition
        search.cost+=distanceSocketBase[search.sequence[0]];

        if(output)
        {       fprintf(fileANN,"\n cost from one point to next point in the route");
                fprintf(fileANN,"\n%d",distanceSocketBase[search.sequence[0]]);
        }

        // sum the cost on the middle route
        for(i=0;i<nInletRoute-1;i++)
        {       search.cost+=distanceInlet[search.sequence[i]].nm[search.sequence[i+1]];
```

```
                if(output)
                {       fprintf(fileANN,"\t%d",distanceInlet[search.sequence[i]].nm[
                                                        search.sequence[i+1]]);
                }
        }
        // sum the cost relative at the end point to basePosition
        search.cost+=distanceSocketBase[search.sequence[nInletRoute-1]];

        if(output)
        {fprintf(fileANN,"\t%d\n",distanceSocketBase[search.sequence[nInletRoute-1]]);
        }

        if(search.cost<best.cost)
        {       i=1;
        }
        else
        {       i=0;
        }

        return(i);
}
//*************************************************************************//
//*************************************************************************//
```

Appendix E

Three Dimensional Rendering

Source Code

The sequence discloses the ***CObjNanorobot3D*** source code used to model and render the 3D nanorobots. The concept of bounding boxes and 3D modelling is presented in Chapter 3. The nanorobot interactions inside the virtual environment are based on real time physically based simulation. The nanorobot as an agent receives input about the actions to be performed from the classes ***CrobotDecisionSensing*** and ***CNeural***, which were presented in Appendix A and C. The model uses a modular approach, and the communications from the distinct parts of the system transfers events through the critical section, which uses the class ***CParallelManager*** – present in Appendix B.

CLASS: CobjNanorobot3D

```
//************************************************************************//
//************************************************************************//
// Nanorobot 3D Rendering with OpenGL and C++
//************************************************************************//
//************************************************************************//
class CObjNanorobot3D:public CObjBase3D
{
        private:

                float color_r,color_g,color_b;
                float pos_x,pos_z,pos_y,movePos_x,movePos_z,movePos_y;

                void Nanorobot(int NanorobotID);
                void drawNanorobot(int NanorobotID);
                void sensor(int NanorobotID);
                void sensingField(int colorBoundBox,float position2);

                void directinalPropeller(void);
                void backPropeller(void);
                void middlePropeller(int whatPropeller);

                void headNanorobot(int NanorobotID);
                void externalNeck(int NanorobotID);
```

```
                void propellerNanorobot(NanorobotID);
                void armNanorobot(float positionedY,
                                  float positionedZ,
                                  float degree,
                                  int armID);

                void mouthNanoRobot(void);
                void connectHead(float baseCylinderRadius,
                                 float topCylinderRadius,
                                 float locateX,
                                 int setConnection);
                void updateLocation(float take_x,float take_y,float take_z);

        public:

                // contructor
                CObjNanorobot3D() : CObjBase3D(){}

                void NanorobotAPI(int NanorobotID,float take_x,float take_y,float take_z);
                void initNanoRobot(void);
};
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::initNanoRobot(void)
{
        initCObj3DCBase();
        color_r=(float)setup->r;
        color_g=(float)setup->g;
        color_b=(float)setup->b;

}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::NanorobotAPI(int NanorobotID,
                                   float take_x,float take_y,float take_z)
{
        updateLocation(take_x,take_y,take_z);

        Nanorobot(NanorobotID);
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::updateLocation(float take_x,float take_y,float take_z)
{
        pos_x=(float)setup->x;
        pos_z=(float)setup->z;
        pos_y=(float)setup->y;

        movePos_x=take_x;
        movePos_y=take_y;
        movePos_z=take_z;

}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::Nanorobot(int NanorobotID)
{
        drawObj3DAPICBase(NanorobotID);
        drawNanorobot(NanorobotID);
}
//*************************************************************************//
```

```
//************************************************************************//
void CObjNanorobot3D::drawNanorobot(int NanorobotID)
{
	float positionedY,positionedZ,degree;
	int i;
	int armID;
	// begin the drawing
	glPushMatrix();

		// here is located the main linking parts of the robot
		glTranslatef((float)movePos_x, (float)movePos_y, (float)movePos_z);
		// method from class base
		motionPerformedCBase(NanorobotID);
		// put the starting rotation for the nanoRobot
		glRotatef(setup->degree, 1.0, 0.0, 0.0);
		//-----------------------------------//
		headNanorobot(NanorobotID);
		externalNeck(NanorobotID);
		propellerNanorobot(NanorobotID);
		sensor(NanorobotID);

		armID=0;
		degree=(float)0;
		positionedY=(float)0.0;
		positionedZ=(float)0.13;
		armNanorobot(positionedY,positionedZ,degree,armID);

		armID=1;
		degree=(float)90;
		positionedY=(float)0.13*(1-2);
		positionedZ=(float)0.0;
		armNanorobot(positionedY,positionedZ,degree,armID);

		armID=2;
		degree=(float)180;
		positionedY=(float)0.0;
		positionedZ=(float)0.13*(1-2);
		armNanorobot(positionedY,positionedZ,degree,armID);
		//-----------------------------------//
	// dispose the current matrix
	glPopMatrix();

}
//************************************************************************//
//************************************************************************//
void CObjNanorobot3D::armNanorobot(float positionedY,float positionedZ,float degree, int armID)
{

	float stack,open,sphereSize;
	float closeArmDegree, rotOuterArm;
	float color_r,color_g,color_b;
	int setConnection;
	float cylinderHight=(float)setup.rotorSize*0.7;
	float baseCylinderRadius=(float)(setup.rotorSize/5)*1.5;
	float topCylinderRadius=(float)(setup.rotorSize/5)*1.5;
	float breakCylinderParts=(float)32;

	closeArmDegree=(float)90;
	// inner arm
	sphereSize=(float)baseCylinderRadius/9*1.7;
	glPushMatrix();
```

```
glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

glColor3f(1,1,0);
glTranslatef((float)pos_x-2.75,
                (float)pos_y+positionedY,
                        (float)pos_z+positionedZ);//0.4);
glRotatef(degree, 1.0, 0.0, 0.0);
glRotatef(closeArmDegree, 0.0, 1.0, 0.0);
glRotatef(rotInnerArmCBase, 0.0, 1.0, 0.0);
gluSphere(g_normalObject, sphereSize*1.2, 32, 20);
//-----------------------------------------------//
// inner arm
glPushMatrix();
        glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

        glColor3f(0,0,1);
        glTranslatef((float)pos_x,(float)pos_y, (float)pos_z);
        gluCylinder(g_normalObject,
                                baseCylinderRadius/9,
                                topCylinderRadius/9,
                                cylinderHight/2,
                                breakCylinderParts, 4);
glPopMatrix();
//-----------------------------------------------//
//-----------------------------------------------//
// outer arm
switch(armID)
{
        case 0:
        {       rotOuterArm=rotOuterArm0CBase;
                break;
        }
        case 1:
        {
                rotOuterArm=rotOuterArm1CBase;
                break;
        }
        case 2:
        {
                rotOuterArm=rotOuterArm2CBase;
                break;
        }
}

glPushMatrix();
        glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

        glColor3f(1,1,0);
        glTranslatef((float)pos_x,(float)pos_y, (float)pos_z+cylinderHight/2);
        glRotatef(rotOuterArm, 0.0, 1.0, 0.0);
        gluSphere(g_normalObject, sphereSize, 32, 20);
        //-----------------------------------------------//
        // inner arm
        glPushMatrix();
                glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

                glColor3f(0,1,1);
                glTranslatef((float)pos_x,(float)pos_y, (float)pos_z);
                gluCylinder(g_normalObject,
                                        baseCylinderRadius/9,
                                        topCylinderRadius/9,
```

```
                                                    cylinderHight*1.3,
                                                    breakCylinderParts, 4);
                        glPopMatrix();

                        glPushMatrix();
                                glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

                                glColor3f(1,1,0);
                                glTranslatef((float)pos_x,(float)pos_y, (float)pos_z+cylinderHight*1.3);
                                gluSphere(g_normalObject, sphereSize, 32, 20);
                        glPopMatrix();
                        //-----------------------------------------------//

                glPopMatrix();
                //-----------------------------------------------//
                //-----------------------------------------------//

        glPopMatrix();
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::sensor(int NanorobotID)
{
        int minus,colorBoundBox;
        float degree;

        float position2=(float)(setup->rotorSize*1.5)*1.5;
        position2*=(float)1.0735;

        // first sensor
        colorBoundBox=0;
        minus=1-2;
        position2*=minus;
        sensingField(colorBoundBox,position2);
        // second sensor
        colorBoundBox=1;
        position2*=minus;
        sensingField(colorBoundBox,position2);

        // first sensor
        colorBoundBox=0;
        minus=1;
        position2*=minus;

        degree=180;
        glRotatef(degree, 0.0, 0.0, 1.0);

        glTranslatef((float)pos_x+1.5,
                        (float)pos_y,
                        (float)pos_z+2.4);

        position2=0;
        sensingField(colorBoundBox,position2);

        degree=90;
        glRotatef(degree, 0.0, 0.0, 1.0);
        glTranslatef((float)pos_x-1.4,
                        (float)pos_y-3.7,
                        (float)pos_z-2.4);
        sensingField(colorBoundBox,position2);
```

```
	degree=180;
	glRotatef(degree, 0.0, 0.0, 1.0);
	glTranslatef((float)pos_x-1.4*2,
			(float)pos_y,
			(float)pos_z);
	sensingField(colorBoundBox,position2);

}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::sensingField(int colorBoundBox,float position2)
{
	float degree,minus;
	float color_r,color_g,color_b;
	float innerRadius,outerRadius,slices,loops, sphereSize;

	float breakCylinderParts=(float)32;
	float cylinderHight=(float)setup->rotorSize*1.5;
	float baseCylinderRadius=(float)setup->rotorSize*0.5;
	float topCylinderRadius=(float)setup->rotorSize*0.5;

	float positioning=(float)((2*(setup->rotorSize*1.5*2))*1.2)/5;

	cylinderHight=(float)12.0;
	baseCylinderRadius=(float)2.8;
	topCylinderRadius=(float)0;

	degree=(float)270;

	color_r=(float)0.9;
	color_g=(float)0.0;
	color_b=(float)0.0;

	cylinderHight=(float)0.4;
	baseCylinderRadius=(float)0.2;
	topCylinderRadius=(float)0.2;

	degree=90;

	//-------------------------------------------------------//
	// sensingField light: metalic base - outer
	glPushMatrix();
		glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

		glColor3f((float)0.0,(float)0.0,(float)1.0);
		glTranslatef((float)pos_x+3.5,
					(float)pos_y-position2,
					(float)pos_z);

		glRotatef(degree, 0.0, 1.0, 0.0);

		gluCylinder(quadratic->g_normalObject,
					baseCylinderRadius,
					topCylinderRadius,
					cylinderHight,
					breakCylinderParts, 4);

	glPopMatrix();
	//-------------------------------------------------------//
	// sensingField light: metalic base - inner mouth
	cylinderHight=(float)0.17;
```

```
	baseCylinderRadius=(float)0.04;
	topCylinderRadius=(float)0.2;

	glPushMatrix();
		glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

		glColor3f((float)0.9,(float)0.9,(float)0.9);
		glTranslatef((float)pos_x+3.72,
					(float)pos_y-position2,
					(float)pos_z);

		glRotatef(degree, 0.0, 1.0, 0.0);

		gluCylinder(quadratic->g_normalObject,
					baseCylinderRadius,
					topCylinderRadius,
					cylinderHight,
					breakCylinderParts, 4);
	glPopMatrix();
	//-------------------------------------------------------//
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::propellerNanorobot(NanorobotID)
{
	float degree;
	float innerRadius,outerRadius,slices,loops,startAngle,sweepAngle;
	float stack,open;
	float color_r,color_g,color_b;
	int setConnection,i,j;
	float cylinderHight=(float)setup->cylinderHight;
	float baseCylinderRadius=(float)setup->baseCylinderRadius;
	float topCylinderRadius=(float)setup->topCylinderRadius;
	float breakCylinderParts=(float)setup->breakCylinderParts;

	degree=-90;

	// propeller neck
	glPushMatrix();
		glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

		glColor3f(1,1,1);
		glTranslatef((float)pos_x-cylinderHight,(float)pos_y, (float)pos_z);

		glRotatef(degree, 0.0, 1.0, 0.0);
		glRotatef(rotRotorCBase/3, 0.0, 0.0, 1.0);
		gluCylinder(g_normalObject,
					(baseCylinderRadius*0.2)*2,
					(topCylinderRadius*0.2)*2,
					(cylinderHight)/3,
					breakCylinderParts, 4);

		//-----------------------------------------//
		// propellers
		innerRadius=(float)(baseCylinderRadius*0.2)*2;
		outerRadius=(float)(baseCylinderRadius*0.2)*5;
		slices=(float)32;
		loops=(float)20;
		startAngle=(float)0;
		sweepAngle=(float)45;
		for(i=0;i<2;i++)
```

```
		{
			for(j=0;j<4;j++)
			{
				startAngle=(float)90*j+45*i;
				// partial disk 1
				glPushMatrix();
					glMateriali(GL_FRONT_AND_BACK,GL_SHININESS,rand());

					glColor3f(1,1*i,0);

					glTranslatef((float)pos_x,(float)pos_y, (float)pos_z+0.10+i*0.10);

					glRotatef(30, 0.0, 0.0, 1.0);
					gluPartialDisk(g_normalObject,
							innerRadius,
							outerRadius,
							slices,
							loops,
							startAngle,
							sweepAngle);
				glPopMatrix();
			}
		}
		//----------------------------------------//
	glPopMatrix();
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::externalNeck(int NanorobotID)
{
	float degree;
	float stack,open;
	float color_r,color_g,color_b;
	int setConnection;
	float cylinderHight=(float)setup.rotorSize*0.7;
	float baseCylinderRadius=(float)setup->baseCylinderRadius;
	float topCylinderRadius=(float)(setup.rotorSize/5)*1.5;
	float breakCylinderParts=(float)32;

	degree=-90;

	stack=(float)3.3;
	open=(float)(baseCylinderRadius*0.2)*2;

	// middle body
	glPushMatrix();
		glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

		glColor3f(1,1,0);
		glTranslatef((float)pos_x-cylinderHight*1.3,(float)pos_y, (float)pos_z);

		glRotatef(degree, 0.0, 1.0, 0.0);
		gluCylinder(g_normalObject,
				baseCylinderRadius*0.2,
				topCylinderRadius*0.2,
				cylinderHight*2.5,
				breakCylinderParts, 4);
	glPopMatrix();

	setConnection=1;
	connectHead((baseCylinderRadius*0.2),
```

```
                                open,
                                cylinderHight*(3.8),
                                setConnection);

        glPushMatrix();
                glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

                glColor3f(1,0,0);
                glTranslatef((float)pos_x-cylinderHight*stack,(float)pos_y, (float)pos_z);

                glRotatef(degree, 0.0, 1.0, 0.0);
                gluCylinder(g_normalObject,
                                        (baseCylinderRadius*0.2)/2,
                                        (topCylinderRadius*0.2)/2,
                                        (cylinderHight*2.5)/2,
                                        breakCylinderParts, 4);
        glPopMatrix();

        setConnection=1;
        connectHead((baseCylinderRadius*0.2)/2,
                                0,// middle body
                                cylinderHight*(1.25+stack),
                                setConnection);

}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::headNanorobot(int NanorobotID)
{
        float degree;
        float color_r,color_g,color_b;
        int setConnection;
        float cylinderHight=(float)setup.rotorSize*0.7;
        float baseCylinderRadius=(float)(setup.rotorSize/5)*1.5;
        float topCylinderRadius=(float)(setup.rotorSize/5)*1.5;
        float breakCylinderParts=(float)32;

        degree=-90;
        // external body
        glPushMatrix();
                glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

                glColor3f((float)0.9,(float)0.7,(float)0.7);
                glTranslatef((float)pos_x,(float)pos_y, (float)pos_z);

                glRotatef(degree, 0.0, 1.0, 0.0);
                gluCylinder(g_normalObject,
                                        baseCylinderRadius,
                                        topCylinderRadius,
                                        cylinderHight,
                                        breakCylinderParts, 4);
        glPopMatrix();

        setConnection=1;
        connectHead(baseCylinderRadius,
                                0,
                                cylinderHight,setConnection);
        setConnection=0;
        connectHead(baseCylinderRadius,
                                0,
                                cylinderHight,setConnection);
```

```
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::mouthNanoRobot(void)
{
	float degree;
	float cylinderHight=(float)setup.rotorSize*1.5*2;
	float baseCylinderRadius=(float)setup.rotorSize*1.5;
	float topCylinderRadius=(float)setup.rotorSize*1.5;
	float breakCylinderParts=(float)32;

	degree=-90;
	glPushMatrix();
		glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

		glColor3f(1,0,0);
		glTranslatef((float)pos_x+4,(float)pos_y, (float)pos_z);

		glRotatef(degree, 0.0, 1.0, 0.0);
		gluCylinder(g_normalObject,
					baseCylinderRadius*1.5,
					topCylinderRadius*0.7,
					cylinderHight*0.6,
					breakCylinderParts, 4);
	glPopMatrix();
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::directinalPropeller(void)
{	middlePropeller(0);
	middlePropeller(1);

	backPropeller();
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::connectHead(float baseCylinderRadius,
                                  float topCylinderRadius,
                                  float locateX,
                                  int setConnection)
{
	float degree;
	float denominator;
	float cylinderHight=(float)setup.rotorSize;
	float breakCylinderParts=(float)32;

	if(setConnection)
	{
		denominator=(float)1-2;
		locateX*=denominator;
		degree=-90;

	}
	else
	{
		denominator=(float)1;
		locateX=0;
		degree=90;
	}
```

```
	glPushMatrix();
		glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

		glColor3f(1,1,1);
		glTranslatef((float)pos_x+locateX,(float)pos_y, (float)pos_z);

		glRotatef(degree, 0.0, 1.0, 0.0);
		gluCylinder(g_normalObject,
					baseCylinderRadius,
					topCylinderRadius,
					cylinderHight*0.3,
					breakCylinderParts, 4);
	glPopMatrix();
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::middlePropeller(int whatPropeller)
{

	float degree,minus;
	float color_r,color_g,color_b;

	float positioning=(float)(2*(setup.rotorSize*1.5*2))*1.2;
	positioning/=(float)5;
	float breakCylinderParts=(float)2;
	float cylinderHight=(float)setup.rotorSize*1.5;
	float baseCylinderRadius=(float)setup.rotorSize*0.5;
	float topCylinderRadius=(float)setup.rotorSize*0.5;
	cylinderHight=(float)3.9;
	baseCylinderRadius=(float)0.8;
	topCylinderRadius=(float)1.9;
	float position2=(float)(setup.rotorSize*1.5)*1.5;
	degree=(float)270;//90+180

	if(whatPropeller)
	{	minus=(float)-1;
	}
	else
	{	minus=(float)1;
	}

	color_r=(float)0.9;
	color_g=(float)0.9;
	color_b=(float)0.0;

	position2*=(float)1.2;

	glPushMatrix();
		glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);

		glColor3f(0,0,1);
		glTranslatef((float)pos_x+(positioning*1.2),
					 (float)pos_y-position2*minus,
					 (float)pos_z);
		glRotatef(degree, 0.0, 1.0, 0.0);
		glRotatef(rotateMiddlePropeller,0.0, 1.0, 0.0);

		gluCylinder(g_normalObject,
					baseCylinderRadius,
					topCylinderRadius,
					cylinderHight,
```

```
                                        breakCylinderParts, 4);
        glPopMatrix();
}
//*************************************************************************//
//*************************************************************************//
void CObjNanorobot3D::backPropeller(void)
{
        float degree;
        float rotor_r,rotor_g,rotor_b;

        float positioning=(float)2*(setup.rotorSize*1.5*2)*1.2;
        float breakCylinderParts=(float)2;
        float cylinderHight=(float)setup.rotorSize*1.5;
        float baseCylinderRadius=(float)setup.rotorSize*1.5;
        float topCylinderRadius=(float)setup.rotorSize*1.5;

        baseCylinderRadius*=0.7;
        topCylinderRadius*=0.7;
        cylinderHight*=1.2;
        baseCylinderRadius*=(float)0.7;
        cylinderHight*=0.5;
        positioning-=cylinderHight*1.4;
        baseCylinderRadius*=1.2;
        topCylinderRadius*=1.4;
        degree=(float)90;

        glPushMatrix();
                glMateriali(GL_FRONT_AND_BACK, GL_SHININESS, rand() % 128);
                glColor3f(0.9,0.9,0.0);
                glTranslatef((float)pos_x-positioning*1.2,
                                        (float)pos_y,
                                        (float)pos_z);
                glRotatef(degree, 1.0, 0.0, 0.0);
                degree=(float)270;
                glRotatef(degree, 0.0, 1.0, 0.0);
                glRotatef(rotateBackPropeller, 0.0, 1.0, 0.0);
                gluCylinder(g_normalObject,
                                        baseCylinderRadius,
                                        topCylinderRadius,
                                        cylinderHight,
                                        breakCylinderParts, 4);
        glPopMatrix();
}
//*************************************************************************//
//*************************************************************************//
```

Publication List

Journals

11 Adriano Cavalcanti, Bijan Shirinzadeh, Toshio Fukuda, Seiichi Ikeda, "Nanorobot for Brain Aneurysm", *International Journal of Robotics Research*, Sage, Vol. 28, no. 4, pp. 558-570, April 2009.

10 Adriano Cavalcanti, Bijan Shirinzadeh, Luiz C. Kretly, "Medical Nanorobotics for Diabetes Control", *Nanomedicine: Nanotechnology, Biology and Medicine*, Elsevier, Vol. 4, no. 2, pp. 127-138, June 2008.

9 Adriano Cavalcanti, Bijan Shirinzadeh, Mingjun Zhang, Luiz C. Kretly, "Nanorobot Hardware Architecture for Medical Defense", *Sensors*, MDPI(Basel), Vol. 8, no. 5, pp. 2932-2958, May 2008.

8 Adriano Cavalcanti, Bijan Shirinzadeh, Robert A. Freitas Jr., Tad Hogg, "Nanorobot Architecture for Medical Target Identification", *Nanotechnology*, IOP, Vol. 19, no. 1, 015103 (15p.), January 2008.

7 Adriano Cavalcanti, Bijan Shirinzadeh, Robert A. Freitas Jr., Luiz C. Kretly, "Medical Nanorobot Architecture Based on Nanobioelectronics", *Recent Patents on Nanotechnology*, Bentham Science, Vol. 1, no. 1, pp. 1-10, February 2007.

6 Adriano Cavalcanti, Robert A. Freitas Jr., Luiz C. Kretly, "Nanobotics Control Design: A Practical Approach Tutorial", *Robotics Today*, Dearborn, Mich.: SME Society of Manufacturing Engineers, 4th Quarter, Vol. 18, no. 4, October 2005.

5 Adriano Cavalcanti, Robert A. Freitas Jr., "Nanorobotics Control Design: A Collective Behavior Approach for Medicine", *IEEE Transactions on Nanobioscience*, Vol. 4, no. 2, pp. 133-140, June 2005.

4 Adriano Cavalcanti, "Nanorobotics", In 3-D Simulations, *Topic In Depth, NSF - The NSDL Scout Report for Math, Engineering and Technology*, Vol. 4, no. 8, The University of Wisconsin-Madison, Madison WI, USA, April 2005.

3 Adriano Cavalcanti, "Assembly Automation with Evolutionary Nanorobots and Sensor-Based Control applied to Nanomedicine", *IEEE Transactions on Nanotechnology*, Vol. 2, no. 2, pp. 82-87, June 2003.

2 Adriano Cavalcanti, Robert A. Freitas Jr., "Nanosystem Design with Dynamic Collision Detection for Autonomous Nanorobot Motion Control using Neural Networks", *Computer Graphics and Geometry*, MEPhI, Vol. 5, no. 1, pp. 50-74, May 2003.

1 Adriano Cavalcanti, Robert A. Freitas Jr., "Autonomous Multi-robot Sensor-Based Cooperation for Nanomedicine", *International Journal of Nonlinear Science and Numerical Simulation*, Vol. 3, no. 4, pp. 743-746, August 2002.

Conference Proceedings

18 Adriano Cavalcanti, Bijan Shirinzadeh, Toshio Fukuda, Ikeda Seiichi, "Hardware Architecture for Nanorobot Application in Cerebral Aneurysm", *IEEE Nano Int'l Conf. on Nanotechnology*, Hong Kong, China, pp. 237-242, Aug. 2007.

17 Adriano Cavalcanti, Bijan Shirinzadeh, Tad Hogg, Julian A. Smith, "Hardware Architecture for Nanorobot Application in Cancer Therapy", *IEEE-RAS Int'l Conf. on Advanced Robotics*, Jeju, Korea, pp. 200-205, August 2007.

16 Adriano Cavalcanti, Bijan Shirinzadeh, Declan Murphy, Julian A. Smith, "Nanorobots for Laparoscopic Cancer Surgery", *IEEE ICIS Int'l Conf. on Computer and Information Science*, Melbourne, Australia, pp. 738-743, July 2007.

15 Adriano Cavalcanti, Bijan Shirinzadeh, Tad Hogg, Luiz C. Kretly, "CMOS-based Nanorobot to Combat Cancer", *Australian Workshop on Fluid Mechanics*, A Complex Dynamical System, Melbourne, Australia, December 2006. (invited talk)

14 Adriano Cavalcanti, Tad Hogg, Bijan Shirinzadeh, Hwee C. Liaw, "Nanorobot Communication Techniques: A Comprehensive Tutorial", *IEEE ICARCV Int'l Conf. on Control, Automation, Robotics and Vision*, Grand Hyatt, Singapore, pp. 2371-2376, December 2006.

13 Adriano Cavalcanti, Lior Rosen, Bijan Shirinzadeh, Moshe Rosenfeld, "Nanorobot for Treatment of Patients with Artery Occlusion", *Springer Proceedings of Virtual Concept*, Cancun, Mexico, November 2006.

12 Adriano Cavalcanti, Warren W. Wood, Luiz C. Kretly, Bijan Shirinzadeh, "Computational Nanomechatronics: A Pathway for Control and Manufacturing Nanorobots", *IEEE CIMCA Int'l Conf. on Computational Intelligence for Modelling*, Control and Automation, IEEE Computer Society, Sydney, Australia, pp. 185-190, November 2006.

11 Adriano Cavalcanti, Tad Hogg, Bijan Shirinzadeh, "Nanorobotics System Simulation in 3D Workspaces with Low Reynolds Number", *IEEE-RAS MHS Int'l Symposium on Micro-Nanomechatronics and Human Science*, Nagoya, Japan, pp. 226-231, November 2006.

10 Adriano Cavalcanti, Tad Hogg, Luiz C. Kretly, "Transducers Development for Nanorobotic Applications in Biomedical Engineering", *IEEE NDSI Nanoscale Devices and System Integration*, Houston TX, USA, April 2005. (invited talk)

9 Adriano Cavalcanti, Lior Rosen, Luiz C. Kretly, Moshe Rosenfeld, Shmuel Einav, "Nanorobotic Challenges in Biomedical Applications, Design and Control", *IEEE ICECS Int'l Conf. on Electronics, Circuits and Systems*, Tel-Aviv, Israel, December 2004.

8 Lior Rosen, Adriano Cavalcanti, Moshe Rosenfeld, Shmuel Einav, "Pro-Inflammatory Cytokines and Soluble Adhesion Molecules as Activating Triggers for Nanorobots", *BMES Conf. on Biomedical Engineering: New Challenges for the Future*, Philadelphia PA, USA, October 2004.

7 Adriano Cavalcanti, Robert A. Freitas Jr., Luiz C. Kretly, "Nanorobotics Control Design: A Practical Approach Tutorial", *ASME 28th Biennial Mechanisms and Robotics Conference*, Salt Lake City Utah, USA, September 2004.

6 Adriano Cavalcanti, Tad Hogg, "Simulating Nanorobots in Fluids with Low Reynolds Number", *11th Foresight Conf. on Molecular Nanotechnology*, Burlingame CA, USA, October 2003.

5 Arancha Casal, Tad Hogg, Adriano Cavalcanti, "Nanorobots as Cellular Assistants in Inflammatory Responses", *IEEE BCATS Biomedical Computation at Stanford 2003 Symposium*, IEEE Computer Society, Stanford CA, USA, October 2003.

4 Adriano Cavalcanti, "Nanorobotics Control Techniques with NanoCAD for Biomedical Applications", *Int'l Symposium Frontiers of NanoEngineering 2003*, Campinas, Brazil, pp. 24-27, October 2003.

3 Adriano Cavalcanti, Robert A. Freitas Jr., "Collective Robotics Coherent Behaviour for Nanosystems with Sensor-Based Neural Motion", *IEEE - Int'l Conf. on Artificial Intelligence Systems*, IEEE Computer Society Press, Divnomorskoe, Russia, pp. 185-190, September 2002.

2 Adriano Cavalcanti, Robert A. Freitas Jr., "Autonomous Multi-robot Sensor-Based Cooperation for Nanomedicine", *ASME/IEEE ICMNS Int'l Conf. on Micro and Nano Systems*, Kunming, China, pp. 139-142, August 2002.

1 Adriano Cavalcanti, "Assembly Automation with Evolutionary Nanorobots and Sensor-Based Control applied to Nanomedicine", *IEEE - Nano 2002 Int'l Conf. on Nanotechnology*, Washington D.C., USA, pp. 161-164, August 2002.

Plenary Lectures

3 Adriano Cavalcanti, "Robots in Surgery", Plenary Lecture, Euro Nano Forum 2005, *Nanotechnology and the Health of the EU Citizen in 2020*, Edinburgh, Scotland, UK, September 2005. (invited talk)

2 Adriano Cavalcanti, "Neural Motion and Evolutionary Decision in Robotic Competition applied for Molecular Machine System Design", Plenary Lecture, *IEEE - CACSD Int'l Conf. on Computer Aided Control System Design, Glasgow*, Scotland, UK, pp. 5-14, September 2002. (invited plenary)

1 Adriano Cavalcanti, Robert A. Freitas Jr., "Nanosystem Design with Dynamic Collision Detection for Autonomous Nanorobot Motion Control using Neural Networks", Plenary Lecture, *ACM SIGGRAPH - Graphicon Int'l Conf. on Computer Graphics*, Novgorod, Russia, pp. 75-80, September 2002. (invited plenary)

Referenced in Post-Graduate Programs

9 Adriano Cavalcanti, Lior Rosen, Luiz C. Kretly, Moshe Rosenfeld, Shmuel Einav, "Nanorobotic Challenges in Biomedical Applications, Design and Control", In *ECE2195 Biomedical Computing*, Fall 2007, Department of Electrical and Computer Engineering, University of Pittsburgh, Pittsburgh PA, USA, November 2007.

8 Adriano Cavalcanti, Robert A. Freitas Jr., Luiz C. Kretly, "Nanorobotics Control Design: A Practical Approach Tutorial", In *ECE5930 Nanomechatronics*, Spring 2006, Department of Electrical and Computer Engineering, Utah State University, Logan UT, USA, April 2006.

7 Adriano Cavalcanti, "Nanorobotics", In *HUAS 6375 Imagery and Iconography*, 3D Simulation as Visual Language: Explaining Complex Ideas Simply, Fall 2004, Graduate Program in Arts and Technology, The University of Dallas, Dallas TX, USA, February 2005.

6 Adriano Cavalcanti, Robert A. Freitas Jr., "Autonomous Multi-robot Sensor-Based Cooperation for Nanomedicine", In *CS 549 Nanorobotics Course*, Simulation of Nanorobots, Fall 2004, Department of Computer Science, University of Southern California, Los Angeles CA, USA, October 2004.

5 Adriano Cavalcanti, Tad Hogg, "Simulating Nanorobots in Fluids with Low Reynolds Number", *Nanorobotics: Nanotechnology, Chemistry Biology*, Info Center ETHZ, Swiss Federal Institute of Technology, Zurich, Switzerland, September 2004.

4 Arancha Casal, Tad Hogg, Adriano Cavalcanti, "Nanorobots as Cellular Assistants in Inflammatory Responses", *Nanorobotics: Nanotechnology, Chemistry Biology*, Info Center ETHZ, Swiss Federal Institute of Technology, Zurich, Switzerland, September 2004.

3 Adriano Cavalcanti, "Assembly Automation with Evolutionary Nanorobots and Sensor-Based Control applied to Nanomedicine", In *Nanorobotics Lab*, Mechanical Engineering Department, Carnegie Mellon University, Pittsburgh, USA, July 2004.

2 Adriano Cavalcanti, "Assembly Automation with Evolutionary Nanorobots and Sensor-Based Control applied to Nanomedicine", In *EE 821 Biomedical Engineering Systems Course*, Get Ten Papers to Review, Department of Electrical Engineering, National Central University, Jhongli City, Taiwan, China, February 2004.

1 Adriano Cavalcanti, Robert A. Freitas Jr., "Autonomous Multi-robot Sensor-Based Cooperation for Nanomedicine", In *CS 599 Papers*, *Nanorobotics Course*, Lecture on Simulation of Nanorobots, Department of Computer Science, University of Southern California, Los Angeles CA, USA, March 2003.

Invited Seminars

3 Adriano Cavalcanti, "Computational Nanomechatronics for Nanorobots in Medicine", *Special Seminar*, Department of Bioengineering, Rice University, Houston TX, USA, April 2005.

2 Adriano Cavalcanti, "Nanorobotics: Virtual Environments and Control Techniques", *Seminary on Computer Science*, Computational Laboratory, Swiss Federal Institute of Technology, Zurich, Switzerland, May 2003.

1 Adriano Cavalcanti, "Autonomous Nanorobotic Control for Competitive Molecular System Design", *Seminary in Dynamics*, Department of Mechanical Engineering, Darmstadt University of Technology, Darmstadt, Germany, May 2002.

Invited Interviews/Article

20 Nanorobot for Brain Aneurysm, Emerging Technology Trends, Chris Jablonski, *ZDNet*, March 2009.

19 Nanorobots to improve health care, *Roland Piquepaille's Technology Trends*, May 2008.

18 Software Provides Peek into the Body – and the Future, Special Feature: Emerging Technologies, *Medical Product Manufacturing News*, Canon Communications LLC, Vol. 12, no. 2, pp. 22-23, March 2008.

17 Nanorobot Manufacturing for Medicine, Advanced Manufacturing Technology, *Technical Insights*, Frost & Sullivan, January 2008.

16 Researchers Eye Software for Nanorobots, Featured Articles, *NanoScienceWorks.Org*, Taylor & Francis Group, January 2008.

15 Nanorobots for drug delivery?, Emerging Technology Trends, Roland Piquepaille, *ZDNet*, December 2007.

14 Virtual 3D nanorobots could lead to real cancer-fighting technology, *Science Physics Tech Nano News*, PhysOrg, December 2007.

13 Nanorobot for Drug Delivery and Diagnosis, Lab Talk, *Nanotechweb*, IOP, December 2007.

12 Medical Nanorobotics for Diabetes, Nanotechnology Interviews, *The International Nanotechnology Business Directory*, NanoVIP, January 2007.

11 Manufacturing Technology for Medical Nanorobots, News Journal, *APNF Asia Pacific Nanotechnology Forum*, Vol. 6, n. 1, January 2007.

10 Nanorobots for Cardiology, *NanoScience Today*, November 2006.

9 Developments on Nanorobots with System on Chip May Advance Cancer Diagnosis, Cancer Treatment, *Health Care News Articles*, *eMaxHealth*, October 2006.

8 Medical Nanorobotics Feasibility, Interviews, Your Gateway to Everything Nanotech, *Nanotechnology Now*, November 2005.

7 Nanorobot pioneer reveals status of simulator, stem cell work, Views on Nanotechnology, *NanoDelta*, February 2005.

6 New Nanorobotic Ideas, Big Things Happen in Small Places, *Nanotechnology News Network*, October 2004.

5 Nanorobot pioneer reveals status of simulator, stem cell work, The Global Nanobiotechnology Intelligence Source, *NanoBiotech News*, NHI Publications, Vol. 2, n. 36, pp. 4-5, September 2004.

4 Nanorobotics, *NanoScience Today*, September 2004.

3 Nanorobots Inside our Bodies?, *Roland Piquepaille's Technology Trends*, August 2004.

2 Robots in the Body, *Genome News Network*, August 2004.

1 Nanorobotics Control, *Infosatellite News*, July 2004.

Citation List

Citations

List of works with citations to Adriano Cavalcanti's research:

82 A. O. Tarakanov, L. B. Goncharova, Y. A. Tarakanov, "Carbon nanotubes towards medicinal biochips", *Nanomedicine & Nanobiotechnology*, Wiley Interdisciplinary Reviews, Vol. 2, no. 1, pp. 1-10, November 2009. http://www3.interscience.wiley.com/journal/122684364/abstract

81 Z. Chen, L. Zhang, Y. Sun, J. Hu, D. Wang, "980-nm Laser-Driven Photovoltaic Cells Based on Rare-Earth Up-Converting Phosphors for Biomedical Applications", *Advanced Functional Materials*, Wiley InterScience, Vol. 19, no. 23, pp. 3815-3820, November 2009. http://www3.interscience.wiley.com/journal/122671477/abstract

80 S. Bewick, R. Yang, M. Zhang, "Complex mathematical models of biology at the nanoscale", *Nanomedicine & Nanobiotechnology*, Wiley Interdisciplinary Reviews, Vol. 1, no. 6, pp. 650-659, October 2009. http://www3.interscience.wiley.com/journal/122615953/abstract

79 R. Torrecillas, J. S. Moya, L. A. Díaz, J. F. Bartolomé, A. Fernández, S. Lopez-Este, "Nanotechnology in joint replacement", *Nanomedicine & Nanobiotechnology*, Wiley Interdisciplinary Reviews, Vol 1, no. 5, pp. 540-552, September 2009. http://au.wiley.com/WileyCDA/Section/id-397791.html

78 J. S. Murday, R. W. Siegel, J. Stein, J. F. Wright, "Translational nanomedicine: status assessment and opportunities", *Nanomedicine: Nanotechnology, Biology and Medicine*, Elsevier, Vol. 5, no. 3, pp. 251-273, September 2009. http://www.nanomedjournal.com/article/S1549-9634%2809%2900106-3/abstract

77 M. Shahini, W. W. Melek, J. T. W. Yeow, "Micro-force compensation in automated micro-object positioning using adaptive neural networks", *Smart Materials & Structures*, IOP, Vol. 18, no. 9, 095023 (14pp), September 2009. http://www.iop.org/EJ/abstract/0964-1726/18/9/095023

76 S. H. Kang, M. S. Islam, "Biosensors on Array Chip by Dual-color Total Internal Reflection Fluorescence Microscopy", *Biochip Journal*, Vol. 3, no. 2, pp. 97-104, June 2009. http://biochips.or.kr/website/04journal04.php?code=in_journal&mode=vie&number=95

75 C. Stephanidis, "*The Universal Access Handbook*", CRC Press, Taylor & Francis, June 2009. http://www.amazon.com/gp/product/0805862803

74 M. Eshaghian-Wilner, *"Bio-Inspired and Nanoscale Integrated Computing*", Wiley, June 2009. http://www.amazon.com/Bio-Inspired-Nanoscale-Integrated-Computing-Nature-Inspired/dp/0470116595

73 S. V. Dorozhkin, "Calcium orthophosphate-based biocomposites and hybrid biomaterials", *Journal of Material Science*, Springer, Vol. 44, no. 9, May 2009. http://www.springerlink.com/content/l0g172523566qv74

72 M. Hamdi, A. Ferreira, "Multiscale Design and Modeling of Protein-based Nanomechanisms for Nanorobotics", *International Journal of Robotics Research*, Vol. 28, no. 4, pp. 436-449, April 2009. http://ijr.sagepub.com/cgi/content/abstract/28/4/436

71 G. S. Nitschke, "*Neuro-Evolution for Emergent Specialization in Collective Behavior Systems*", PhD Thesis, Doctoral Theses Sciences, Vrije Universiteit Amsterdam, Amsterdam, Netherlands, March 2009. http://dare.ubvu.vu.nl//handle/1871/13153

70 O. Elhage, N. Hegarty, "Robotic Technology", *Urologic Robotic Surgery in Clinical Practice*, Springer London, March 2009. http://www.springerlink.com/content/k61860wl3n241583

69 R. Majumdar, J. S. Rathore, N. N. Sharma, "Simulation of Swimming Nanorobots in Biological Fluids", *IEEE ICARA Int'l Conf. on Autonomous Robots and Agents*, Wellington, New Zealand, pp. 79-82, February 2009. http://www.ieeexplore.ieee.org/xpl/freeabs_all.jsp?isnumber=4803907&arnumber=4803912

68 B. Fisher, "Biological Research in the Evolution of Cancer Surgery: A Personal Perspective", *Cancer Research*, American Association for Cancer Research, Vol. 68, no. 24, pp. 10007-10020, December 2008. http://cancerres.aacrjournals.org/cgi/content/abstract/68/24/10007

67 G. Pistoia, "*Battery Operated Devices and Systems: From Portable Electronics to Industrial Products*", Elsevier, December 2008. http://www.amazon.com/Battery-Operated-Devices-Systems-Electronics/dp/0444532145

66 C. Hill, A. Amodeo, J. V. Joseph, H. R. H. Patel, "Nano- and microrobotics: how far is the reality?", *Expert Review of Anticancer Therapy*, Vol. 8, no. 12, pp. 1891-1897 December 2008. http://www.expert-reviews.com/doi/abs/10.1586/14737140.8.12.1891

65 K.H.S. Hla, Y. Choi, J. S. Park, "Mobility Enhancement in Nanorobots by Using Particle Swarm Optimization Algorithm", *IEEE Int'l Conf. on Computational Intelligence and Security*, Suzhou, China, pp. 35-40, December 2008. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4724610

64 L. Žlajpah, "Simulation in robotics", *Mathematics and Computers in Simulation*, Vol. 79 , no. 4, pp. 879-897, December 2008. http://linkinghub.elsevier.com/retrieve/pii/S0378475408001183

63 K. Haymar S. Hla, Y. S. Choi, J. S. Park, "Obstacle Avoidance Algorithm for Collective Movement in Nanorobots", *IJCSNS International Journal of Computer Science and Network Security*, Vol. 8, no.11, pp. 302-309, November 2008. http://search.ijcsns.org/02_search/02_search_03.php?number=200811043

62 N. Solomon, "System, methods and apparatuses for integrated circuits for nanorobotics", *US Patent 20080244500*, October 2008. http://www.freepatentsonline.com/y2008/0244500.html

61 N. Solomon, "System and methods for collective nanorobotics for medical applications", *US Patent 20080241264*, October 2008. http://www.freepatentsonline.com/y2008/0241264.html

60 N. Solomon, "System and methods for collective nanorobotics for electronics applications", *US Patent 2008024330*, October 2008. http://www.freepatentsonline.com/y2008/0243303.html

59 C. A. Ouzounis, "The Emergence of Bioinformatics: Historical Perspective, Quick Overview and Future Trends", *Bioinformatics in Cancer and Cancer Therapy*, Humana Press, October 2008. http://www.springerlink.com/content/k53r31532155020j

58 C. A. Piña-García, E.-J. Rechy-Ramírez, V. A. García-Vega,, "Comparing Three Simulated Strategies for Cancer Monitoring with Nanorobots", *Lecture Notes in Computer Science*, Springer Berlin, October 2008. http://www.springerlink.com/content/l70703751p71m752

57 T. L. Dawson, "Nanomaterials for textile processing and photonic applications", *Coloration Technology*, Wiley-Blackwell, Vol. 124, no. 5, pp. 261-272, October 2008. http://www.ingentaconnect.com/content/bpl/cte/2008/00000124/00000005/art00001

56 K. A. Eaton, P. A. Reynolds, S. K. Grayden , N. H. F. Wilson, "A vision of dental education in the third millennium", *British Dental Journal*, Nature, Vol. 205, no. 5, pp. 261-271, September 2008. http://www.nature.com/bdj/journal/v205/n5/abs/sj.bdj.2008.736.html

55 B. Nerlich, "Powered by Imagination: Nanobots at the Science Photo Library", *Science as Culture*, Taylor & Francis, Vol. 17, no. 3, pp. 269-292, September 2008. http://www.informaworld.com/smpp/content~db=all?content=10.1080/09505430802280743

54 F. Walsh, S. Balasubramaniam, D. Botvich, T. Suda, T. Nakano, S. F. Bush, M. Ó Foghlú, "Hybrid DNA and Enzymatic based Computation for Address Encoding, Link Switching and Error Correction in Molecular Communication", *ACM Nano-Net Int'l Conf. on Nano-Networks*, Boston, USA, September 2008. http://www.ece.gatech.edu/research/labs/bwn/nanos/papers/HybridDNA.pdf

53 K.H.S. Hla, Y. Choi, J. S. Park, "Self Organized Mobility in Nanosensor Network Based on Particle Swarm Optimization and Coverage Criteria", *IEEE Int'l Conf. on Networked Computing and Advanced Information Management*, Gyeongju, Korea, pp. 636-641, September 2008. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4624083

52 H. Ahari, F. Dastmalchi, Y. Ghezelloo, R. Paykan, M. Fotovat, J. Rahmannya, "The application of silver nano-particles to the reduction of bacterial contamination in poultry and animal production", *Food Manufacturing Efficiency*, IFIS, Vol. 2, no. 1, pp. 49-53, August 2008. http://www.atypon-link.com/IFIS/doi/abs/10.1616/1750-2683.0028

51 I. F. Akyildiz, F. Brunetti, C. Blázquez, "Nanonetworks: A new communication paradigm", *Computer Networks*, Vol. 52, no. 12, pp. 2260-2279, August 2008. http://linkinghub.elsevier.com/retrieve/pii/S1389128608001151

50 A. L. Despotuli, A. V. Andreeva, "Prospects of Deep-Sub-Voltage Nanoelectronics and Related Technologies", *Nanoelektronika*, no. 2, pp. 1-15, August 2008. http://www.nanometer.ru/2008/02/08/nanoelektronika_5900/PROP_FILE_files_2/subvolt.pdf

49 B. L. Jennings-Spring, "Methods, treatments, and compositions for modulating Hedgehog pathways", *US Patent 20080138379*, June 2008. http://www.freepatentsonline.com/y2008/0138379.html

48 J. B. Elder, D. J. Hoh, B. C. Oh, A. C. Heller, C. Y. Liu, M. L. J. Apuzzo, "The future of cerebral surgery: A kaleidoscope of opportunities", *Neurosurgery*, Vol. 62, no. 6, pp. 1555-1579, June 2008. http://www.ncbi.nlm.nih.gov/pubmed/18695575

47 B. Layton, "Recent Patents in Bionanotechnologies: Nanolithography,Bionanocomposites, Cell-Based Computing and Entropy Production", *Recent Patents on Nanotechnology*, Vol. 2, no. 2, pp. 72-83(12), June 2008. http://www.ingentaconnect.com/content/ben/nanotec/2008/00000002/00000002/art00001

46 J. Jeon, J.-B. Lee, M.J. Kim, "A 20 μm movable micro mobile", *Technical Proceedings of the 2008 NSTI Nanotechnology Conference and Trade Show*, NSTI-Nanotech, Nanotechnology 2008, Vol. 3, pp. 190-193, June 2008. http://www.nsti.org/Nanotech2008/showabstract.html?absno=1426

45 N. Wickramasinghe, E. Geisler, "*Encyclopedia of Healthcare Information Systems*", June 2008. http://www.amazon.com/Encyclopedia-Healthcare-Information-Systems-Wickramasinghe/dp/1599048892

44 N. Solomon, "Nanorobotic System", *World International Patent WO/2008/063473*, May 2008. http://www.wipo.int/pctdb/en/wo.jsp?WO=2008063473

43 J. S. Murday, S. O. Moldin, "Re-Engineering Basic and Clinical Research to Catalyze Translational Nanoscience", *NSF Report*, University of Southern California, March 2008. http://www.nsf.gov/crssprgm/nano/reports/reengineering_basic_and_clinical_4_9_09.pdf

42 N. N. Sharma, R. K. Mittal, "Nanorobot Movement: Challenges and Biologically inspired solutions", *International Journal on Smart Sensing and Intelligent Systems*, Vol. 1, no. 1, March 2008. http://www.s2is.org/Issues/v1/n1/papers/paper6.pdf

41 J. B. Elder, C. Y. Liu, M. L. J. Apuzzo, "Neurosurgery in The Realm of 10^{-9}, Part 2: Applications of Nanotechnology to Neurosurgery-Present and Future", *Neurosurgery*, Vol. 62, no. 2, pp. 269-285, February 2008. http://www.neurosurgery-online.com/pt/re/neurosurg/abstract.00006123-200802000-00009.htm

40 R. J. Andrews, "Neuroprotection at the Nanolevel - Part I Introduction to Nanoneurosurgery", *Annals of the New York Academy of Sciences*, Vol. 1122, pp. 169-184, Dec. 2007. http://www.blackwell-synergy.com/doi/abs/10.1196/annals.1403.012

39 S. Das, A. J. Gates, H. A. Abdu, G. S. Rose, C. A. Picconatto, J. C. Ellenbogen, "Designs for Ultra-Tiny, Special-Purpose Nanoelectronic Circuits", *IEEE Transactions on Circuits and Systems I*, Vol. 54, no. 11, pp. 2528-2540, November 2007. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4383238

38 Tad Hogg, "Distributed Control of Microscopic Robots in Biomedical Applications", *Advances in Applied Self-organizing Systems*, Springer, part II, pp. 147-174, November 2007. http://www.springerlink.com/content/w50k3338v0417130

37 V. Shanthi, S. Musunuri, "Prospects for Medical Robots", *Journal of Nanotechnology Online*, Azonano, Vol. 3, pp. 1-9, November 2007. http://www.azonano.com/Details.asp?ArticleID=2035

36 C. Chibaya, S. Bangay, "A probabilistic movement model for shortest path formation in virtual antlike agents", *ACM Int'l Conf. of the South African institute of computer scientists and information technologists*, Sunshine Coast, South Africa, Vol. 226, pp. 9-18, October 2007. http://portal.acm.org/citation.cfm?id=1292491.1292493

35 J. Goicoechea, C. Ruiz Zamarreño, I.R. Matias and F.J. Arregui, "Minimizing the photobleaching of self-assembled multilayers for sensor applications", *Sensors and Actuators B: Chemical*, Elsevier, Vol. 126, no. 1, pp. 41-47, September 2007. http://linkinghub.elsevier.com/retrieve/pii/S0925400506007209

34 M.O. Killijian, N. Rivière, M. Roy, "Experimental Evaluation of Resilience for Ubiquitous Mobile Systems", *Workshop on Ubiquitous Systems Evaluation*, Innsbruck, Austria, pp. 283-287, September 2007. http://www.laas.fr/~mkilliji/publications.html

33 H. Heusala, "Technology Trends and Design Aspects of Data Processing Cores of Future Small Smart Objects", *Int'l Workshop on Design and Integration Principles for Smart Objects*, Innsbruck, Austria, September 2007. http://www.nanorobotdesign.com/references/DIPSO-Paper4.pdf

32 E. Brickner, "*The Populating Problem A Study In Multi-Nano-Robotics*", Master Thesis, Computer Science Department, Technion Israel Institute of Technology, September 2007. http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2007/MSC/MSC-2007-09.pdf

31 H. Heusala, "Technology Trends and Design Aspects of Data Processing Cores of Future Small Smart Objects", *Int'l Workshop on Design and Integration Principles for Smart Objects*, Innsbruck, Austria, September 2007. http://eis.comp.lancs.ac.uk/workshops/dipso/dipso2007/pdf/DIPSO-Paper4.pdf

30 M. Shahini, W. W. Melek, J. T. W. Yeow, "A Neural Network-based Learning Controller for Micro-sized Object Micromanipulation", *IEEE IJCNN 2007 Int'l Conf. on Neural Networks*, Orlando FL, USA, pp. 3035-3040, August 2007. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4371444

29 N. Wickramasinghe, S. Choudhary, E. Geisler, "Bionanotechnology: its applications and relevance to healthcare", *International Journal of Biomedical Engineering and Technology*, Inderscience, Vol. 1, no. 1, pp. 41-58, June 2007. http://www.ingentaconnect.com/content/ind/ijbet/2007/00000001/00000001/art00003

28 X. Yuan, S. X. Yang, "Multirobot-Based Nanoassembly Planning with Automated Path Generation", *IEEE/ASME Transactions on Mechatronics*, Vol. 12, no 3, pp. 352-356, June 2007. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4244399

27 P. Caamaño, A. Prieto, J. A. Becerra, R. Duro, F. Bellas, "Evolutionary Tool for the Incremental Design of Controllers for Collective Behaviors", *Lecture Notes in Computer Science*, Springer, Vol. 4527, pp. 587-596, June 2007. http://www.springerlink.com/content/r2ph304169153734

26 D. Murphy, B. Challacombe, T. Nedas, O. Elhage, K. Althoefer, L. Seneviratne, P. Dasgupta, "Equipment and technology in robotics", *Arch. Esp. Urol.*, Vol. 60, no. 4, pp. 349-354, May 2007. http://www.ncbi.nlm.nih.gov/pubmed/17626526

25 J.Q. Liu, K. Shimohara, "Molecular Computation and Evolutionary Wetware: A Cutting-Edge Technology for Artificial Life and Nanobiotechnologies", *IEEE Transactions on Systems, Man and Cybernetics Part C-Applications and Reviews*, Vol. 37, no. 3, pp. 325-336, May 2007. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4154945

24 T. Hogg, P. J. Kuekes, "Mobile microscopic sensors for high resolution in vivo diagnostics", *Nanomedicine: Nanotechnology, Biology and Medicine*, Elsevier, Vol. 2, no. 4, pp. 239-247 December 2006. http://www.nanomedjournal.com/article/PIIS1549963406001444/abstract

23 S. Hede, N. Huilgol, "Nano: The new nemesis of cancer", *Journal of Cancer Research and Therapeutics*, Vol. 2, no. 4, pp. 186-195, December 2006. http://www.cancerjournal.net/article.asp?issn=0973-1482;year=2006;volume=2;issue=4;spage=186;epage=195;aulast=Hede

22 J. R. Vaughn, "*Over the Horizon: Potential Impact of Emerging Trends in Information and Communication Technology on Disability Policy and Practice*", National Council on Disability, Washington DC, December 2006. http://www.ncd.gov/newsroom/publications/2006/pdf/emerging_trends.pdf

21 T. Hogg, "Coordinating Microscopic Robots in Viscous Fluids", *Autonomous Agents and Multi-Agent Systems*, Springer, Vol. 14, no. 3, pp. 271-305, Jun. 2007. http://www.hpl.hp.com/research/idl/papers/microSensors

20 D. Murphy, B. Challacombe, M. S. Khan, P. Dasgupta, "Robotic Technology in Urology", *Postgraduate Medical Journal*, BMJ, Vol. 82, n. 973, pp. 743-747, November 2006. http://pmj.bmj.com/cgi/content/abstract/82/973/743

19 P. Couvreur, C. Vauthier, "Nanotechnology: Intelligent design to treat complex disease", *Pharmaceutical Research*, Vol. 23, no. 7, pp. 1417-1450 July 2006. http://nano.cancer.gov/resource_center/sci_biblio_enabled-therapeutics_abstracts.asp

18 S. P. Leary, C. Y. Liu, M. L. I. Apuzzo, "Toward the emergence of nanoneurosurgery: Part III - Nanomedicine: Targeted nanotherapy, nanosurgery, and progress toward the realization of nanoneurosurgery", *Neurosurgery*, Vol. 58, no. 6, pp. 1009-1025 June 2006. http://www.neurosurgery-online.com/pt/re/neurosurg/abstract.00006123-200606000-00001.htm

17 A. S. G. Curtis, M. Dalby, N. Gadegaard, "Cell signaling arising from nanotopography: implications for nanomedical devices", *Nanomedicine Journal*, Future Medicine, Vol. 1, no. 1, pp. 67-72, June 2006. http://www.futuremedicine.com/doi/abs/10.2217/17435889.1.1.67

16 E.-Y. Kwon, Y.-T. Kim, D.-E. Kim, "Study on the Elastic Characteristics of Living Cells using Atomic Force Microscope Indentation Technique", *KSTLE International Journal*, Vol. 7, no. 1, pp. 10-13, June 2006. http://www.dbpia.co.kr/view/ar_view.asp?arid=757571

15 J. D. Bronzino, "*Tissue Engineering and Artificial Organs, The Biomedical Engineering Handbook*", Taylor & Francis CRC Press, May 2006. http://www.amazon.com/Tissue-Engineering-Artificial-Biomedical-Handbook/dp/0849321239

14 G. M. Patel, G. C. Patel, R. B. Patel, J. K. Patel, M. Patel, "Nanorobot: A versatile tool in nanomedicine", *Journal of Drug Targeting*, Vol. 14, no. 2, pp. 63-67, February 2006. http://www.ncbi.nlm.nih.gov/pubmed/16608733

13 H.-W. Jiang, S.-G. Wang , W. Xu, Z.-Z. Zhang, L. He, "Research and progress in bio-nano-robot", *Robot*, Vol. 27, no. 6, pp. 569-574. Nov-December 2005. http://md1.csa.com/partners/viewrecord.php?requester=gs&collection=TRD&recid=2006056126991MT

12 N. A. Weir, D. P. Sierra, J. F. Jones, "A Review of Research in the Field of Nanorobotics", *Sandia Report*, Office of Scientific and Technical Information, US Department of Energy, October 2005. http://www.osti.gov/energycitations/product.biblio.jsp?osti_id=875622

11 R. J. Narayan, "Pulsed laser deposition of functionally gradient diamond-like carbon-metal nanocomposites", *Diamond and Related Materials*, Elsevier, Vol. 14, no. 8, pp. 1319-1330 August 2005. http://top25.sciencedirect.com/index.php?cat_id=5&subject_area_id=15&journal_id=09259635

10 H. Jiang, S. Wang, W. Xu, Z. Zhang, L. He, "Construction of medical nanorobot", *IEEE Int'l Conf. on Robotics and Biomimetics*, Hong Kong, China, pp.151-154, July 2005. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1708613

9 A. Cuschieri, "Laparoscopic Surgery: Current Status, Issues and Future Developments", *Surgeon*, Vol. 3, no. 3, pp. 125-138, June 2005. https://www.thesurgeon.net/site/CMD=ORA/ArticleID=c5e9ca42-b108-4ac4-b74d-0e01cc0e2adc/1008/default.aspx

8 W. W. Wood, "Nanorobots: A New Paradigm for Hydrogeologic Characterization?", *Ground Water*, Wiley InterScience, Vol. 43, Issue 4, pp. 463, July 2005. http://www3.interscience.wiley.com/journal/118644114/abstract

7 A. Galstyan, T. Hogg, K. Lerman, "Modeling and Mathematical Analysis of Swarms of Microscopic Robots", *IEEE Swarm Intelligence Symposium*, pp. 201-208, Pasadena CA, USA, June 2005. http://www.isi.edu/~lerman/papers/lerman05SIS.pdf

6 I. D. Villar, I. R. Matias, F. J. Arregui, R. O. Claus, "ESA-based in-fiber nanocavity for hydrogen-peroxide detection", *IEEE Transactions on Nanotechnology*, Vol. 4, no. 2, pp. 187-193 March 2005. http://cat.inist.fr/?aModele=afficheN&cpsidt=16598341

5 A. Ummat, G. Sharma, C. Mavroidis, A. Dubey, "Bio-Nanorobotics: State of the Art and Future Challenges", *Biomedical Engineering Handbook*, *Bio-Nano Robotics*, CRC Press, March 2005. http://www.coe.neu.edu/Research/robots/papers/2123.pdf

4 M. Zhang, C. L. Sabharwal, W. M. Tao, T. J. Tarn, N. Xi, G. Li, "Interactive DNA sequence and structure design for DNA nanoapplications", *IEEE Transactions on Nanobioscience*, Vol. 3, no. 4, pp. 286-292 December 2004. http://www.ncbi.nlm.nih.gov/pubmed/15631140

3 R. J. Narayan, P. N. Kumta, C. Sfeir, D.-H. Lee, D. Olton, D. Choi, "Nanostructured ceramics in medical devices: Applications and prospects", *JOM*, Vol. 56, no. 10, pp. 38-43 October 2004. http://www.springerlink.com/content/y30717j73442k2ng

2 K. E. Drexler, "Nanotechnology: From Feynman to Funding", *Sage Bulletin of Science Technology and Society*, Vol. 24, no. 1, pp. 21-27 February 2004. http://www.metamodern.com/d/04/00/FeynmanToFunding.pdf

1 W. J. Li, N. Xi, W. K. Fung, T. S. Wong, "Nanorobotics and Nanomanipulation", Encyclopedia of Nanoscience and Nanotechnology, American Scientific Publishers, Vol. 7, no. 15, pp. 351-365, 2004. http://www.ingentaconnect.com/content/asp/enn/2004/00000007/00000001/art00018

References

[1] L. M. Adleman. On Constructing a Molecular Computer. *DNA Based Computers*, 1995, http://olymp.wu-wien.ac.at/usr/ai/frisch/local.html .

[2] A.P. Alivisatos, K.P. Johnsson, X.G. Peng, T.E. Wilson, C.J. Loweth, M.P. Brchez, and P.G. Schultz. Organization of Nanocrystal Molecules Using DNA. *Nature*, Vol. 382, pp. 609-611, 1996.

[3] J. S. Altman and J. Kien. New models for motor control. *Neural Computation*, Vol. 1, pp. 173-183, 1989.

[4] N. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *Proc. of IEEE Int. Conf. Robotics and Automation*, pp. 113-120, Minneapolis, MN, 1996.

[5] D. Andre, F. H. Bennett III, and J. R. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In *Proceedings of the First Annual Conference on Genetic Programming*, pp. 3-11. Cambridge, MA: MIT Press, 1996.

[6] D. Andre, F. H. Bennett III, and J. R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Proceedings of the 5th International Workshop on the Synthesis and Simulation of Living Systems - Artificial Live V*, Cambridge, MA: MIT Press, 1996.

[7] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. In *IEEE Transactions on Neural Networks*, Vol. 5, no. 1, pp. 54-65, January 1994.

[8] D. M. Antonelli, and J. Y.Ying. Mesoporous Material. In *Current Opinion in Colloid and Interface Science*, Vol. 1, pp.523-529, 1996.

[9] F. Arai, A. Kawaji, T. Sugiyama, Y. Onomura, M. Orgawa, and T. Fukuda. 3D Micromanipulation System under Microscope. In *MHS'98 Int. Symp. on Micromechatronics and Human Science*, pp.127-134, 1998.

[10] F. Arai, K. Morishima, T. Kasugai and T. Fukuda. Bio-Micro-Manipulation (New Direction for Operation Improvement). In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robotics and Systems - IROS*, Vol. 3, pp. 1300-1305, 1997.

[11] F. Arai, D. Andou, Y. Nonoda, T. Fukuda, H. Iwata, and K. Itoigawa. Micro endeffector with micro pyramids and integrated piezoresistive force sensor. In *Proc. of the IEEE/RSJ Int. Con. on Intelligent Robots and Systems*, pp. 842-849, 1996.

[12] F. Arai, D. Ando, and T. Fukuda. Micro manipulation based on micro physics: Strategy based on attractive force reduction and stress measurement. In *Proc. of the IEEE Int'l. Conf. on Robotics and Automation*, pp. 236-241, 1995.

[13] R. C. Arkin and J. Diaz. Line-of-Sight Constrained Exploration for Reactive Multiagent Robotic Teams. In *Proc. of Int'l Conference on Autonomous Agents and Multi-Agent Systems*, 2001. www.cc.gatech.edu/ai/robot-lab/online-publications/aamas.pdf .

[14] T. Asano, D. Kirkpatrick, and C. K. Yap. D1-optimal motion for a rode. In *Proc. of the 12th Annual Symposium on Computational Geometry*, New York, ACM Press, pp. 252-263, 1996.

[15] K. J. Aström and B. Wittenmark. *Adaptive Control*. 2nd edition, Addison-Wesley Inc., NY, 1995.

[16] E. M. Atkins. *Plan Generation and Hard Real-Time Execution with Application to Safe, Autonomous Flight*. PhD Thesis, Department of Computer Science and Engineering, The University of Michigan, 1999.

[17] G. S. Attard. Mesoporous Platinum Films from Lyotropic Liquid Crystalline Phases. *Science*, Vol.278, pp. 838, 1997.

[18] G. D. Bachand and C. D. Montemagno. Constructing organic/inorganic NEMS devices powered by biomolecular motors. *Biomedical Microdevices*, Vol. 2, no. 3, pp. 179-184, 2000.

[19] S. Bandi. *Discrete Object Space Methods for Computer Animation*. PhD thesis, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1998.

[20] D. Bar-On, D. Gershon, A. Israeli, and G. Zuniga. TRACK II: A multi-processor robot controller. In *Proc. CompEuro Int. Conf. on Computers in Design*, Manufacturing, and Production, pp. 86-93, Prix-Erry, France, May 1993.

[21] D. Braha, Y. Bar-Yam. The Statistical Mechanics of Complex Product Development: Empirical and Analytical Results. *Management Science*, Vol. 53, no. 7, pp. 1127-1145. July 2007.

[22] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *ACM SIGGRAPH Computer Graphics Proceedings*, Annual Conf. Series, pp. 23-34, 1994.

[23] D. Barraf. *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD Thesis, Department of Computer Science, Cornell University, Ithaca, NY, 1992.

[24] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. *ACM Computer Graphics*, Vol. 24, no. 4, pp. 19-28, 1990.

[25] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *ACM SIGGRAPH Computer Graphics Proceedings*, Vol. 23, pp. 223-232, 1989.

[26] J. Barraquand, L. E. Kavraki, J. C. Latombe, T. Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. In *Int. J. of Robotics Research*, Vol. 16, no. 6, pp. 759-774, 1997.

[27] J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. In *IEEE Trans. Syst., Man, Cybern.*, Vol. 22, no. 2, pp. 224-241, 1992.

[28] J. Barraquand and J. C. Latombe. Robot motion planning: A distributed representation approach. In *Int. J. of Robotics research*, Vol. 10, no. 6, pp. 628-649, December, 1990.

[29] R. Barzel and A. Barr. A modelling system based on dynamic constraints. *ACM Computer Graphics*, Vol. 22, no. 4, pp. 31-39, 1988.

[30] T. Basar and P. R. Kumar. On worst case design strategies. *Comput. Math. Applic.*, Vol. 13, no. (1-3), pp. 239-245,1987.

[31] T. Basar and G. J. Olsder. *Dynamic Noncooperative Game Theory*. Academic Press, London, 1982.

[32] U. Bässler. The walking-(and searching-) pattern generator of stick insects, a modular system composed of reflex chains and edogenous oscillators. In *Biological Cybernetics*, Vol. 69, pp.305-317, 1993.

[33] K. Basye, T. Dean, J. Kirman, and M. Lejter. A decision-theoretic approach to planning, perception, and control. *IEEE Expert*, Vol. 7, no. 4, pp. 58-65, August 1992.

[34] C. Baur and A. Bugaciv. Nanoparticle manipulation by mechanical pushing: Underlying phenomena and real-time monitoring. *Nanotechnology*, Vol. 9, pp. 360-364, 1998.

[35] R. Beckers, O.E. Holland, and J.L. Deneubourg. From local actions to global tasks: Stigmergy and collective robotics. In *Proc. of the Fourth International Workshop on the Synthesis and Simulation of Living Systems Artificial Life IV*, pp. 181-189, 1994.

[36] R. D. Beer, H. J. Chiel, R. D. Quinn, K. S. Espenchied, and Patrick Larsson. A distributed neural network architecture for hexapod robot locomotion. *Neural Computation*, Vol. 4, pp. 356-365, 1992.

[37] R. D. Beer, H. J. Chiel, and L. S. Sterling. An artificial insect. *American Scientist*, Vol. 79, pp.444-452, 1991.

[38] R. D. Beer. *Intelligent as Adaptive Behaviour: An Experiment in Computational Neuroethology*. Academic Press, 1990.

[39] H. C. Berg. Dynamic properties of bacterial flagellar motors. *Nature*, Vol. 249, no. 452, pp. 77-79, May 1974.
[40] G. Binnig. Atomic Force Microscope. In *Phys. Rev. Letts.*, Vol. 56, no. 9, pp. 930-933, 1986.
[41] G. Binnig, H. Rohrer, C. Gerber, and E. Weibel. Surface Studies by Scanning Tunneling Microscopy, In *Phys. Rev. Letts.*, Vol. 49, no. 1, pp. 57-61, 1982.
[42] K. F. Bohringer, R. S. Fearing, and K. Y. Goldberg. Parallel microassembly. In *Workshop on Precision Manipulation at Micro and Nano Scales, IEEE Int. Conf. on Robotics and Automation*, pp. 110-135, 1998.
[43] V. Boor, M. Overmars, and A. F. van der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *Proc. of IEEE Int. Conf. Robotics and Automation*, Detroit, MI, 1999.
[44] R. Boulic, R. Mas, and D. Thalmann. Complex character positioning based on a compatible flow model of multiple supports. *IEEE Transactions on Visualization and Computer Graphics*, pp. 245-261, July-September 1997.
[45] W. Bouma and Jr. G. Vanecek. Collision detection and Analysis in a physically based simulation. In *Proceeding of the Second Eurographics Workshop on Animation and Simulation*, Vienna, Austria, 1992.
[46] P. Bourgine. Autonomy, abduction, adaption. In *Proc. of Int. Conf. on Computer Animation'94 IEEE Computer Society Press*, pp. 104-111, Los Alamitos, CA, 1994.
[47] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47, 139-159, 1991.
[48] R. A. Brooks. Intelligence without reason. In Mylopoulos, J., & Reiter, R. (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, San Mateo, CA, Morgarn Kaufmann, 1991.
[49] R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6, 3-15, 1990.
[50] R. A. Brooks. A robot that walks; emergent behaviors from a carefully evolved network. *Neural Computing*, Vol. 1, pp. 253-262, 1989.
[51] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, Vol. 2, no. 1, pp. 14-23, March 1986.
[52] R. A. Brooks. Solving the find-path problem by good representation of free space. *IEEE Trans. Syst., Man, Cybern.*, Vol. 13, no. 3, pp. 190-197, 1983.
[53] R. C. Brost and A. D. Christiansen. Probabilistic analysis of manipulation tasks: A research agenda. In *IEEE Int. Conf. Robot. & Autom.*, Vol. 3, pp. 549-5556, 1993.
[54] L. Brus. Semiconductor Colloids: Individual Nanocrystals, Opals and Porous Silicon. *Current Opinion in Colloid and Interface Science*, Vol.1, pp.197-201, 1996.
[55] D. P. Brutzman, Y. Kanayama and M. J. Zyda. Integrated Simulation for Rapid Development of Autonomous Underwater Vehicles. *IEEE Autonomous Underwater Vehicle Conference*, IEEE Oceanic Engineering Society, Washington DC, pp. 3-10, June 1992.
[56] A. E. Bryson and Y.C. Ho. *Applied Optimal Control, Hemisphere Publishing Corp.*, New York, NY, 1975.
[57] R. B. Byrnes, D. L. MacPherson, S. H. Kwak, M. L. Nelson, and R. B. McGhee. An Experimental Comparison of Hierarchical and Subsumption Software Architectures for Control of an Autonomous Underwater Vehicles. *IEEE Oceanic Engineering Society Symposium on Autonomous Underwater Vehicles*, Washington D.C., pp. 135-141, 1992.
[58] T. Calvert, R. Ovans, and S. Mah. Towards the autonomous animation of multiple figures. In *Computer Animation'94,* IEEE Computer Society Press, pp. 69-75, Los Alamitos, CA, 1994.
[59] S. A. Cameron. A study of the clash detection problem in robotics. In *IEEE ICRA Proc. of the Int'l Conf. on Robotics and Automation*, pp. 488-493, 1985.
[60] G. Campa, M. L. Fravolini, B. Seanor, M. R. Napolitano, D. D. Gobbo, G. Yu, and S. Gururajan. On-line learning neural networks for sensor validation for the flight control system of a B777 research scale model. *International Journal of Robust Nonlinear Control*, Vol. 12, no. 11, pp. 987-1007, September 2002.
[61] J. F. Canny, M. C. Lin. An Opportunistic Global Path Planner. *Algorithmica*, Special Issue on Computational Robotics, Vol. 10, no. 2-4, pp. 102-120, October 1993.
[62] J. F. Canny and M. C. Lin. An opportunistic global path planner. *Algorithmica*, Special issue on Computational Robotics, Vol. 10, no. 2-4, pp. 102-120, Aug/Sept/Oct 1993.

[63] J. F. Canny. On computability of fine motion plans. In *IEEE Int. Conf. Robot. & Autom.*, pp. 177-182, 1989.

[64] J. F. Canny. The Complexity of Robot Motion Planning. MIT Press, Cambridge, MA, 1988.

[65] A. Cavalcanti, B. Shirinzadeh, T. Fukuda, S. Ikeda. Nanorobot for Brain Aneurysm. *International Journal of Robotics Research*, Sage, Vol. 28, no. 4, pp. 558-570, April 2009.

[66] A. Cavalcanti, B. Shirinzadeh, L. C. Kretly. Medical Nanorobotics for Diabetes Control. *Nanomedicine: Nanotechnology, Biology and Medicine*, Elsevier, Vol. 4, no. 2, pp. 127-138, June 2008.

[67] A. Cavalcanti, B. Shirinzadeh, M. Zhang, L. C. Kretly. Nanorobot Hardware Architecture for Medical Defense. *Sensors*, MDPI(Basel), Vol. 8, no. 5, pp. 2932-2958, May 2008.

[68] A. Cavalcanti, B. Shirinzadeh, R. A. Freitas Jr., T. Hogg. Nanorobot Architecture for Medical Target Identification. *Nanotechnology*, IOP, Vol. 19, no. 1, 015103 (15p.), January 2008.

[69] A. Cavalcanti, B. Shirinzadeh, T. Fukuda, I. Seiichi. Hardware Architecture for Nanorobot Application in Cerebral Aneurysm. *IEEE Nano Int'l Conf. on Nanotechnology*, Hong Kong, China, pp. 237-242, Aug. 2007.

[70] A. Cavalcanti, B. Shirinzadeh, T. Hogg, J. A. Smith. Hardware Architecture for Nanorobot Application in Cancer Therapy. *IEEE-RAS Int'l Conf. on Advanced Robotics*, Jeju, Korea, pp. 200-205, August 2007.

[71] A. Cavalcanti, B. Shirinzadeh, D. Murphy, J. A. Smith. Nanorobots for Laparoscopic Cancer Surgery. *IEEE ICIS Int'l Conf. on Computer and Information Science*, Melbourne, Australia, pp. 738-743, July 2007.

[72] A. Cavalcanti, B. Shirinzadeh, R. A. Freitas Jr., L. C. Kretly. Medical Nanorobot Architecture Based on Nanobioelectronics. *Recent Patents on Nanotechnology*, Bentham Science, Vol. 1, no. 1, pp. 1-10, February 2007.

[73] A. Cavalcanti, B. Shirinzadeh, T. Hogg, L. C. Kretly. CMOS-based Nanorobot to Combat Cancer. *Australian Workshop on Fluid Mechanics*, A Complex Dynamical System, Melbourne, Australia, December 2006.

[74] A. Cavalcanti, T. Hogg, B. Shirinzadeh, H. C. Liaw. Nanorobot Communication Techniques: A Comprehensive Tutorial. *IEEE ICARCV Int'l Conf. on Control, Automation, Robotics and Vision*, Grand Hyatt, Singapore, pp. 2371-2376, December 2006.

[75] A. Cavalcanti, L. Rosen, B. Shirinzadeh, M. Rosenfeld. Nanorobot for Treatment of Patients with Artery Occlusion. *Springer Proceedings of Virtual Concept*, Cancun, Mexico, November 2006.

[76] A. Cavalcanti, W. W. Wood, L. C. Kretly, B. Shirinzadeh. Computational Nanomechatronics: A Pathway for Control and Manufacturing Nanorobots. *IEEE CIMCA Int'l Conf. on Computational Intelligence for Modelling*, Control and Automation, IEEE Computer Society, Sydney, Australia, pp. 185-190, November 2006.

[77] A. Cavalcanti, T. Hogg, B. Shirinzadeh. Nanorobotics System Simulation in 3D Workspaces with Low Reynolds Number. *IEEE-RAS MHS Int'l Symposium on Micro-Nanomechatronics and Human Science*, Nagoya, Japan, pp. 226-231, November 2006.

[78] A. Cavalcanti, R. A. Freitas Jr., L. C. Kretly. Nanobotics Control Design: A Practical Approach Tutorial. *Robotics Today*, Dearborn, Mich.: SME Society of Manufacturing Engineers, 4th Quarter, Vol. 18, no. 4, October 2005.

[79] A. Cavalcanti, R. A. Freitas Jr. Nanorobotics Control Design: A Collective Behavior Approach for Medicine. *IEEE Transactions on Nanobioscience*, Vol. 4, no. 2, pp. 133-140, June 2005.

[80] A. Cavalcanti. Assembly Automation with Evolutionary Nanorobots and Sensor-Based Control applied to Nanomedicine. *IEEE Transactions on Nanotechnology*, Vol. 2, no. 2, pp. 82-87, June 2003.

[81] Adriano Cavalcanti, Robert A. Freitas Jr. Nanosystem Design with Dynamic Collision Detection for Autonomous Nanorobot Motion Control using Neural Networks. *Computer Graphics and Geometry*, MEPhI, Vol. 5, no. 1, pp. 50-74, May 2003.

[82] A. Cavalcanti, R. A. Freitas Jr. Autonomous Multi-robot Sensor-Based Cooperation for Nanomedicine. *International Journal of Nonlinear Science and Numerical Simulation*, Vol. 3, no. 4, pp. 743-746, August 2002.

[83] D. Chalou and M. Gini. Parallel robot motion planning. In *Proc. of IEEE Int'l Conf. on Robotics and Automation*, Atlanta, GA, pp. 24-51, 1993.

[84] W. Chen, K. Lewis. A Robust Design Approach for Achieving Flexibility in Multidisciplinary Design. *AIAA Journal American Institute of Aeronautics and Astronautics*, Vol. 37, no. 8, pp. 982-990, 1999.

[85] S. Chenney. *Sensing for autonomous agents in virtual environments*. http://http.cs.berkeley.edu/~schenney/autonomous/sensing.html, 1996.

[86] W. Ching and N. Badler. Fast motion planning for anthropometric figures with many degrees of freedom. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2340-2345, 1992.

[87] A. Codourey, M. Rodriguez, and I. Pappas. Human machine interaction for manipulations in the microworld. In *Proc. of the IEEE Int'l Workshop on Robot and Human Communication*, pp. 244-249, 1996.

[88] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer-Verlag, 1999.

[89] L. S. Crawford and S. S. Sastry. *Learning controllers for complex behavioral systems*. Technical report, ERL Memo Number M96/73, U.C. Berkeley, 1996.

[90] L. S. Crawford and S. S. Sastry. Biological motor control approaches for a planar diver. In *Proc. of the 34th IEEE Int'l Conference on Decision and Control*, pp. 3881-3886, December 1995.

[91] H. Cruse, D. E. Brunn, and C. Bartling. Walking: A complex behavior controlled by simple networks. *Adaptive Behavior*, Vol. 3, no. 4, pp. 385-418, 1995.

[92] A. I. Csurgay, W. Porod. The Circuit Paradigm in Modelling Coupled Nanomechanic-Nanoelectronic Dynamics. In *Proc. of IEEE Nano 2002 Int'l Conf. on Nanotechnology*, Washington, USA, pp. 79-82, August 2002.

[93] J. G. Cyster. Chemokines and cell migration in secondary lymphoid organs. *Science*, no. 286, 2098-2102, December 1999.

[94] A. Czarn, C. MacNish. From Nanotechnology to Nano-Planning. In *9th Computer Science Research Conf. in University of Western Australia*, Nedlands, Western Australia, Department of Computer Science, The University of Western Australia, Vol. 1, pp. 73-85, 1998.

[95] *DARPA Neural Network Study*. AFCEA International Press, 1988.

[96] R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson. Evolving globally synchronized cellular automata. In L. J. Eshelman (Ed.) *Proceedings of the Sixth International Converence on Genetic Algorithms*. San Francisco CA, Morgan Kaufmann, 1995.

[97] R. Das, M. Mitchell, and J. P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H. P. Schwefel and R. Männer (Eds.), *Lecture Notes in Computer Science 866 – Parallel Problem Solving from Nature – PPSNIIIII, International Conference on Evolutionary Computation, The Third Conference Parallel Problem Solving from Nature*, pp. 344-353. Berlin, Germany, 1994.

[98] T. L. Dean and M. P. Wellman. *Planning and Control*. Morgan Kaufman, San Mateo, CA, 1991.

[99] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall Publications, Englewood Cliffs, NJ, 1982.

[100] M. H. Devoret, R. J. Schoelkopf. Amplifying Quantum Signals with the Single-Electron Transistor. *Nature*, Vol. 406, pp.1039-1046, 2000.

[101] S. H. Dewitt, A. W. Czarnik. Combinatorial Organic Synthesis Using Parke-Davis's DIVERSOMER Method. *Accounts Chem. Res.*, 29, pp. 114-122, 1996.

[102] D. M. Dilts, N. P. Boy, H. H. Whoirms. The evolution of control architectures for automated manufacturing systems. In *Journal of Manufacturing Systems*, Vol. 10, 1991.

[103] D. Divincenzo. Quantum Computation. *Science*, Vol.270, p.255, 1995.

[104] B. R. Donald. *Error Detection and Recovery for Robot Motion Planning with Uncertainty*. PhD thesis, Department of electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1987.

[105] L. X. Dong, F. Arai and T. Fukuda. 3D Nanorobotic Manipulations of Multi-Walled Carbon Nanotubes. In *ICRA'2001 - Proc. of the Int'l Conf. on Robotics and Automation*, 2001.

[106] L. X. Dong, F. Arai and T. Fukuda. 3D Nanorobotic Manipulations of Nanometer Scale Objects. *J. of Robotics and Mechatronics*, Vol. 13, no. 2, pp. 146-153, 2001.

[107] L. X. Dong, F. Arai and T. Fukuda. 3D Nanorobotic Manipulation of Nano-order Objects inside SEM. *Proc. of the 2000 Int'l Symp. on Micromechatronics and Human Science*, pp. 151-156, 2000.

[108] P. A. O'Donnel and T. Lozano-Perez. Deadlock-free and collision-free coordination of two robot manipulators. In *IEEE Int. Conf. Robot. & Autom*, pp. 484-489, 1989.

[109] H. A. Downing and R. L. Jeanne. Nest construction by the paperwasp, Plistes: a test of stigmergy theory. *Animal Behaviour*, Vol. 36, pp. 1729-1739, 1988.

[110] K. E. Drexler, D. Forrest, R. A. Freitas Jr., J. S. Hall, N. Jacobstein, T. McKendree, R. Merkle, C. Peterson. *A Debate about Assemblers*. Institute for Molecular Manufacturing, 2001, www.imm.org/SciAmDebate2/whitesides.html .

[111] K. E. Drexler. *Nanosystems: molecular machinery, manufacturing, and computation*. Wiley & Sons, 1992.

[112] K. E. Drexler, C. Peterson, and B. Pergamit. *Unbounding the future: the Nanotechnology Revolution*. William Morrow and Company, New York, USA, 1991, http://www.foresight.org/UTF/Unbound_LBW/download.html .

[113] K. E. Drexler. Molecular Engineering: an Approach to the Development of General Capabilities for Molecular Manipulation. *Proc. Natl. Acad. Sci.*, USA, Vol.78, No.9, pp. 5275-5278, 1981.

[114] X. F. Duan, Y. Huang, Y. Cui, J. F. Wang and C. M. Lieber. Indium Phosphide Nanowires as Building Blocks for Nanoscale Electronic and Optoelectronic Devices. *Nature*, Vol. 409, pp. 66-69, 2001.

[115] H. F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE Trans. Robot. & Autom.*, Vol. 4, no. 1, pp. 23-31, February 1988.

[116] H. Edelsbrunner. A new approach to rectangle intersections, part i&ii. *Int'l J. Computer Math.*, Vol. 13, pp. 31-45, 1984.

[117] D. M. Eigler and E. K. Schweizer. Positioning Single Atoms with a Scanning Tunnelling Microscope. *Nature*, Vol. 344, pp.524-526, 1990.

[118] Ö. Ekeberg, A. Lansner, and S. Grillner. The neural control of fish swimming studied through numerical simulations. *Adaptive Behavior*, Vol. 3, no. 4, pp. 363-384, 1995.

[119] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, Vol. 22, no. 6, pp. 46-57, June 1989.

[120] EngeneOS, Inc.; http://www.engeneos.com .

[121] M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *IEEE Int. Conf. Robot. & Autom.*, pp. 1419-1424, 1986.

[122] M. A. Erdmann. *On motion planning with uncertainty*. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, August 1984.

[123] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual serving in robotics. *IEEE Trans. Robot. & Autom.*, Vol. 8, no. 3, pp. 313-326, June 1992.

[124] M. Falvo, R. Superfine, and S. Washburn. The nanomanipulator: A teleoperator for manipulating materials at the nanometer scale. In *Proc. of the Int. Symp. On the Science and Technology o Atomically Engineered Materials*, pp. 579-586, Nov. 1995.

[125] R. E. Fayek, R. Liscano, G. M. Karam. A system architecture for a mobile robot based on activities and a blackboard control unit. In *IEEE Int. Conf. on Robotics and Automation*, Vol. 2, pp.267-274, 1993.

[126] R. S. Fearing. Survey of sticking effects for micro parts handling. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and systems*, pp. 212-217, 1995.

[127] J. T. Feddema, P. Xavier, and R. Brown. Assembly planning at the micro scale. In *Workshop on Precision Manipulation at Micro and Nano Scales, IEEE Int. Conf. on Robotics and Automation*, pp. 56-69, 1998.

[128] J. Feldman, M. A. Fanty, and N. H. Goddard. Computing with structured neural networks. *IEEE Computer*, pp 91-103, March 1988.

[129] R. P. Feynman. There's Plenty of Room at the Bottom. *Caltech's Engineering and Science*, Feb., pp.22-36, 1960.

[130] G. Fishbine. *The Investor's Guide to Nanotechnology & Micromachines*. Wiley & Sons, 2001.

[131] S. Fleury, M. Herrb, R. Chatila. Design of a modular architecture for autonomous robots. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 3508-3513, 1994.

[132] D. Floreano. Reducing human design and increasing adaptivity in evolutionary robotics. In Gomi, T. (Ed.), *Evolutionary Robotics*, AAI Books, Ontario, Canada, pp. 187-220, 1997.

[133] L. D. Floriani, G.G. Pieroni, V. Murino, E. Puppo. Virtual environment generation by CAD-based methodology for underwater navigation. in *Proceedings IX European Signal Processing Conference*, pp.1105-1108, Rodi, Greece, 1998.

[134] I. Foster. Ch.5 Compositional C++. *Designing and Building Parallel Programs.* Addison Wesley, http://www.mcs.anl.gov/~itf/dbpp, 1995.

[135] A. Fox and S. Hutchinson. Exploiting visual constraints in the synthesis of uncertainty-tolerant motion plans. *IEEE Trans. Robot. & Autom.*, Vol. 1, no. 11, pp. 56-71, February 1995.

[136] N. R. Franks, A. Wilby, B. W. Silverman, and C. Tofts. Self-organizing nest contruction in ants: sophisticated building by blind buldozing. *Animal Behaviour*, Vol. 44, pp. 357-375, 1992.

[137] N. R. Franks. Teams in social insects: Group retrieval of prey by army ants. *Behavioral Ecology and Sociobiology*, Vol. 18, pp. 425-429, 1986.

[138] P. Fraundorf. *Scanning Tunneling Microscope*. 1997, http://www.umsl.edu/~fraundor/stm97x.html .

[139] R. A. Freitas Jr., C. J. Phoenix. *Vasculoid: A Personal Nanomedical Appliance to Replace Human Blood*. 2002, http://www.jetpress.org/volume11/vasculoid.html.

[140] R. A. Freitas Jr., The future of nanofabrication and molecular scale devices in nanomedicine. Studies in Health Technology and Informatics, Published in: Renata Bushko, ed., *Future of Health Technology*, pp. 45-59, IOS Press, Amsterdam, The Netherlands, 2002, http//www.rfreitas.com/Nano/FutureNanofabNMed.htm.

[141] R. A. Freitas Jr., Microbivores: Artificial Mechanical Phagocytes using Digest and Discharge Protocol. Zyvex preprint, March 2001, http://www.zyvex.com/Publications/articles/ Microbivores.html .

[142] R. A. Freitas Jr., *Nanomedicine*, Vol. I: Basic Capabilities, Landes Bioscience, Georgetown, TX, 1999, http://www.nanomedicine.com .

[143] R. A. Freitas Jr., Exploratory design in medical nanotechnology: A mechanical artificial red cell, Artificial Cells, Blood Substitutes, and Immobil. *Biotech*, Vol. 26, pp. 41-430, 1998 http://www.foresight.org/Nanomedicine/Respirocytes.html.

[144] T. Fukuda, F. Arai, L. Dong. Nano Robotic World - from Micro to Nano. Plenary Lecture, *ICRA 2001 Proc. of the Int'l Conf. on Robotics and Automation*. http://www.icra2001.org

[145] T. Fukuda, F. Arai. Prototyping Design and Automation of Micro/Nano Manipulation System. *Proc. of IEEE Int'l Conf. on Robotics and Automation (ICRA'00)*, Vol. 1, pp.192-197, 2000.

[146] M. Gaedke, R. Wicke. *Softwaretechnik für Entwicklung, Betrieb und Wartung von Anwendungen in World-Wide Web*. Master's Thesis, Telecooperation Office, University of Karlsruhe, Germany, 1997.

[147] L. Geppert. The Amazing Vanishing Transistor Act. Cover story, *IEEE Spectrum Magazine*, pp. 28-33, October 2002.

[148] A. Ghosh, P. Fischer. Controlled Propulsion of Artificial Magnetic Nanostructured Propellers. *Nano Letters*, Vol. 9, no. 6, pp. 2243-2245, March 2009.

[149] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading MA, 1989.

[150] K. Y. Goldberg. *Stochastic Plans for Robotic Manipulation*. PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August 1990.

[151] A. Goldbeter. *Biochemical Oscillations and Cellular Rhythms: The Molecular Bases of Periodic and Chaotic Behavior*. Cambridge University Press, N.Y., 1996.

[152] T. Gomi. *Evolutionary Robotics II - from Intelligent Robots to Artificial Life*. AAI Books, Kanata, Canada, 1998.

[153] T. Gomi. Non-Cartesian Robotics - the first 10 years. In Gomi, T. (Ed.), *Evolutionary Robotics, From Intelligent Robots to Artificial Life*, Kanata, Canada, AAI Books, 1998.

[154] J. H. Graham. Special computer architectures for robotics: Tutorial and survey. In *IEEE Int'l Symp. on Industrial Electronics (ISIE'97)*, Guimaraes, Portugal, University of Minho, 7-11 July, pp. 702-707, 1989.

[155] P. Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensiset cubitermes sp. La théorie de la stigmergie : essai d'interpré. *Insectes Sociaux*, Vol. 6, pp. 41-81, 1959.

[156] J. J. Grefenstette, and A. Schultz. An evolutionary approach to learning in robots. *Machine Learning Workshop on robot Learning*, New Brunswick, NJ, 1994.

[157] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. NeuroAnimator: Fast neural network emulation and control of physics-based models. In M. Cohen, ed., *Proc. of ACM SIGGRAPH 98 Conf.*, pp. 142-148, 1998.

[158] A. Guillot and J.-A. Meyer. Computer simulations of adaptive behavior in animats. In *Computer Animation'94*, IEEE Computer Society Press, Los Alamintos, CA, pp. 122-132, 1994.

[159] L. J. Guo. A single Electron Transistor Memory Operating at Room Temperature. *Science*, Vol. 275, no. 5300, pp. 649-651, 1997.

[160] M. Guthold. Controlled Manipulation of Molecular Samples with the nanomanipulator. *IEEE/ASME Trans. on Mechatronics*, Vol.5, No.2, pp. 189-198, 2000.

[161] M. Guthold, M. R. Falvo. Controlled manipulation of molecular samples with the nanomanipulator. In *Proc. of the IEEE/ASME Int'l Conf. on Advanced Intelligent Mechatronics*, 1999.

[162] M. T. Hagan, H. B. Demuth, O. D. Jesús. An introduction to the use of neural networks in control systems. *International Journal of Robust Nonlinear Control*, Vol. 12, no. 11, pp. 959-985, September 2002.

[163] M. Hagiya. From Molecular Computing to Molecular Programming. *Proc. 6th DIMACS Workshop on DNA Based Computers*, held at the University of Leiden, Leiden, The Netherlands, pp. 198-204, 2000.

[164] J. K. Hahn. Realistic animation of rigid bodies. *ACM Computer Graphics*, Vol. 22, no. 4, pp. 299-308, 1988.

[165] J. M. Haile. *Molecular Dynamics Simulations, Elementry Methods*. John Wiley and Sons, New York, 1992.

[166] B. E. Hallam, J. R. P. Halperin, and J. C. T. Hallam. An ethological model for implementation in mobile robots. *Adaptive Behavior*, Vol. 3, no. 1, pp. 51-79,1994.

[167] D. A. Handelman, S. H. Lane, and J. J. Gelfand. Integrating neural networks and knowledge-based systems for intelligent robotic control. *IEEE Control Systems Magazine*, April 1990.

[168] L. T. Hansen and A. Kuhle. A technique for positioning nanoparticles using an atomic force microscope. *Nanotechnology*, Vol. 9, pp. 337-343, December 1998.

[169] R. K. Hansen and P. A. Andersen. A 3-D underwater acoustic camera - properties and applications. In P. Tortoli and L. Masotti, editors, *Acoustical Imaging*, Plenum Press, pp. 607-611, 1996.

[170] S. Haykin. *Neural Networks a Comprehensive Foundation*. 2nd edition, Prentice Hall, New Jersey, USA, 1999.

[171] R. M. Hazen. *The Diamond Makers*. Cambridge U.P., ISBN 0-521-65474-2, NY, 1999.

[172] A. Hellemans. German Team Creates New Type of Transistor-Like Device. News Analysis, *IEEE Spectrum Magazine*, January 2003, pp. 20-21.

[173] D. Henrich, and T. Höniger. Parallel processing approaches in robotics. In *Proc. of the IEEE Int. Symp. on Industrial Electronics (ISIE'97)*, Guimaraes, Portugal, University of Minho, pp 702-707, July 1997.

[174] D. Henrich and X. Cheng. Fast distance computation for on-line collision detection with multi-arm robots. *IEEE Int'l Conf. on Robotics and Automation*, pp. 2514-2519, Nice, France, May 1992.

[175] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, 1991.

[176] B. V. Herzen, A. H. Barr, and H.R. Zatz. Geometric collisions for time-dependent parametric surfaces. *ACM Computer Graphics*, Vol. 24, no. 4, pp. 39-48, August 1990.

[177] H. Hirukawa, T. Matsui, and K. Takase. Automatic determination of possible velocity and applicable force of frictionless objects in contact from a geometric mode. *IEEE Trans. on Robotics and Automation*, Vol. 10, no. 3, pp. 309-322, June 1994.

[178] R. L. Hoffman. A common sense approach to assembly sequence planning. In L. S. Homem de Mello and S. Lee, editors, *Computer-Aided Mechanical Assembly Planning*, pp. 289-314, Kluwer, 1991.

[179] H. Bojinov, A. Casal, T. Hogg,. Multiagent Control of Self-reconfigurable Robots. *Proc. of Int'l Conf. on Multiagent Systems*, pp. 441-455, July 2001.

[180] J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[181] R. L. Hollis, S. Salcudean, and D.W. Abraham. Toward a tele-nanorobotic manipulation system with atomic scale force feedback and motion resolution. In *Proc. Of the IEEE Int. Conf. On MicroElectromechanical Systems*, pp. 115-119, 1990.

[182] J. E. Hopcroft, J. T. Scwartz, and M. Sharir. Efficient detection of intersections among spheres. *International Journal of Robotics Research*, Vol. 2, no. 4, pp. 77-80, 1983.

[183] J. J. Hopfield. Neural Networks and Physical systems with emergent collective computational abilities. In *Prof. Natl. Acad. Sci*. USA 79, pp. 2554-2558, 1982.

[184] W. Hordijk, J. P. Crutchfield, and M. Mitchell. Embedded-particle computation in evolved cellular automata. *Physics and Computation'96*, 1996.

[185] A. Hörmann. Steuerung und Systemarchitektur von Fortgeschrittenen autonomen Systemen. *Robotersysteme 5*, Springer-Verlag, pp. 1723-185, 1989.

[186] T. Horsch, F. Schwarz, and H. Tolle. Motion planning for many degrees of freedom: Random reflections at c-space obstacles. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA'94)*, pp. 3318-3323, San Diego, CA, April 1994.

[187] S. Hosogi, N. Watanabe, and M. Sekiguchi. A neural network model of the cerebellum performing dynamic control of a robotic manipulator by learning. *Fujitsu Science and Technology Journal*, Vol. 29, no. 3, pp. 201-208, September 1993.

[188] D. Hsu, L. E. Kavraki, J. C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In P.K. Agarwal, L.E.Kavraki, and M.T. Mason, editors, *Robotics: The Algorithms Perspective, Workshop on Algorithmic Foundations of Robotics*, A. K. Peters, Natick, MA, pp. 141-153, 1998.

[189] D. Hsu, J. C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. of computational Geometry and Applications*, Vol. 9, no. 4-5, pp. 495-512, 1997.

[190] H. Hu and M. Brady. A Bayesian approach to real-time obstacle avoidance for a mobile robot. *Autonomous Robots*, Vol. 1, no. 1, pp. 69-92, 1994.

[191] Z. Huang, R. Boulic, N. Magnenat Thalmann, and D. Thalmann. A multi-sensor approach for grasping and 3D interaction. In *Proc. Computer Graphics International'95*, Leeds, 1995.

[192] I. W. Hunter, S. Lafontaine, P. M. Nielsen, P. J. Hunter, and J. M. Hollerbach. Manipulation and dynamic mechanical testing of microscopic objects using tele-micro-robot system. *IEEE Control Systems Magazine*, Vol. 10, no. 2, pp. 3-9, 1990.

[193] Y. K. Hwang and N. Ahuja. Gross motion planning: A survey. *ACM Computing Surveys*, Vol. 24, no. 3, pp. 219-291, 1992.

[194] HyperChem. *Computational Chemistry*. Publication #HC40-00-03-00, Hypercube, Inc. Waterloo, Ontario, Canada, 1994.

[195] T. Inoue, K. Iwatani, I. Shimoyama, and H. Miura. Micromanipulation using magnetic field. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 679-684, 1995.

[196] J. U. Intza, D. Floreano. Evolutionary Robots with fast Adaptive Behavior in New Environments. In *Proc. of Int'l Conf. on Evolvable Systems: from Biology to Hardware (ICES2000)*, Edingurgh, Scotland (UK), pp. 251-256, April 2000.

[197] H. Ishibuchi, R. Fujioka, and H. Tanaka. Neural networks that learn from fuzzy if-then rules. *IEEE Transactions on fuzzy systems*, Vol. 1, no. 2, pp. 85-97, May 1993.

[198] S. Ishikawa. A method of Autonomous Mobile Robot Navigation by using Fuzzy Control. *Advanced Robotics*, Vol.9, no. 1, pp. 29-52,1995.

[199] N. Jakobi. Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics. In Husbands, P., & Harvey, I. (Eds.), *Advances in Artificial Life: Proceedings of the 4th European Conference on Artificial Life*, pp. 348-357, MIT Press, 1997.

[200] N. Jakobi. Harnessing morphogenesis. *International Conference on Information Processing in Cells and Tissues*, Liverpool, UK, 1995.

[201] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, Vol. 1, no. 1, pp. 7-38, 1998.

[202] F. Jensen. *An Introduction to Bayesian Networks*. Springer Verlag, 1996.

[203] C. Joachim. Electronics Using Hybrid-Molecular and Mono-Molecular Devices. *Nature*, Vol. 408, pp. 541-548, 2000.

[204] T. Junno, K. Deppert, L. Montelius, and L. Samuelson. Controlled manipulation of nanoparticles with an atomic force microscopy. *App. Physics Letters*, Vol. 66, no. 26, pp. 3627-3629, June 1995.

[205] K. T. Kalveram. Controlling the dynamics of a two-joined arm by central patterning and reflex-like processing. *Biological Cybernetics*, Vol. 65, no. 1, pp. 65-71, 1991.

[206] K. Kaneko, H. Tokashiki, K. Tanie, and K. Komoriya. A development of experimental system for macro-micro teleoperation. In *Proc. of the IEEE Int. Workshop on Robot and Human Communication*, pp. 30-35, 1995.

[207] K. Kant and S. W. Zucker. Toward efficient trajectory planning: the path-velocity decomposition. *Int. J. Robot. Res.*, Vol. 5, no. 3, pp. 72-89, 1986.

[208] R. Kargl. Die Computer der Zukunft: Wie ihre grenzenlose Power unser Leben erleichtern soll. *PM Magazin*, pp. 28-32, August 2002.

[209] T. Kasaya, H. Miyazaki, S. Saito, and T. Sato. Micro object handling under SEM by vision-based automatic control. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 2189-2196, 1999.

[210] S. Kauffman. Random Chemistry. *Drug Discovery Design*, Vol. 2, pp. 319-326, 1994.

[211] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration space. *IEEE Trans. On Robotics and Automation*, Vol. 12, no. 4, pp. 566-580, August 1996.

[212] L. E. Kavraki. *Random networks in configuration space for fast path planning*. PhD thesis, Dept. of Computer Science, Stanford University, Stanford, CA, January 1995.

[213] L. E. Kavraki and J. C. Latombe. *Randomized preprocessing of configuration space for fast path planning*. Technical report. Dept. of Computer Science, Stanford University, September 1993.

[214] M. Khatib, B. Bouilly, T. Simeon, R. Chatila. Indoor Navigation with Uncertainty using Sensor Based Motions. *Proc. IEEE Int'l Conf. on Robotics and Automation*, pp. 3379-3384, 1997.

[215] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, Vol. 5, no.1, pp. 90-98, 1986.

[216] K. H. Kim. The Distributed Time-Triggered Simulation Scheme Facilitated by TMO Programming. *IEEE Fourth International Symposium on Object-Oriented Real-Time Distributed Computing*, pp. 57-61, Magdeburg, Germany, May, 2001.

[217] Y. Kim, J. Y. Jo, V. B. Velasco, N. A. Barendt, A. Podgurski, G. Ozsoyoglu, F. L. Merat. A flexible software architecture for agile manufacturing, In *Proc. Int. Conf. on Robotics and Automation*, pp. 3043-3047, Albuquerque, USA, 1997.

[218] Y. Koga. *On Multi-Arm Manipulation Planning*. PhD thesis, Stanford University, Stanford, CA, USA, 1994.

[219] Y. Koga, K. Kondo, J. Kuffner, and J. C. Latombe. Planning motions intentions. In *Proc. SIGGRAPH'94*, pp. 395-408, 1994.

[220] D. Koller and A. Pfeffer. Object-oriented bayesian networks. In *Proc. of 13th Annual Conference on Uncertainty in AI (UAI)*, Providence, Providence, Rhode Island, August 1997.

[221] N. Koumura, R. W. Zijlstra, R. A. van Delden, N. Harada, B. L. Feringa. Light-driven monodirectional molecular rotor. *Nature*, Vol. 401, no. 6749, pp. 152-155, September 1999.

[222] K. Koyano and T. Sato. Micro object handling system with concentrated visual fields and new handling skills. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, pp. 2541-2548, 1996.

[223] M. Krantz. Building a better world-atom by atom. *Times*, pp. 62-63, December 2, 1996.

[224] D. Kruglinski, G. Sheperd, and S. Wingo. *Programming Microsoft Visual C++*. 5th Edition, Microsoft Press, Washington, USA, 1998.

[225] C. R. Kube and H. Zhang. Task Modelling in Collective Robotics. *Autonomous Robots*, Vol. 4, no. 1, pp. 53-72, 1997.

[226] J. J. Kuffner Jr., Goal-directed navigation for animated characters using real-time path planning and control. In *Proc. of CAPTECH'98 Workshop on Modelling and Motion Capture Techniques for Virtual Environments*. Springer-Verlag, November 1998.

[227] T. Kunii, S. Nishimura, T. Noma. The design of a parallel processing system for computer graphics. In *Parallel Processing for Computer Graphics Theory and Applications (Proceedings of InterGraphics'83)*, pp. 360-373. Springer-Verlag, April 1983.

[228] H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. Wiley, New York, NY, 1972.

[229] T. W. Längle. *Verteilte Steuerungskonzept für kmplexe inhomogene Roboter-systeme*. PhD Thesis, University of Karlsruhe, VDI-Fortschriftts-berichte Nr. 8/776114, VDI-Verlag, Düsseldorf, Germany, 1997.

[230] J. C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.

[231] J. C. Latombe, A. Lazanas, and S. Shekhar. Robot motion planning with uncertainty in control and sensing. *Artificial Intelligence*, Vol. 52, no. 1, pp. 1-47, 1991.

[232] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In *Proc. of the IEEE International Conf. on Robotics and Automation (ICRA'99)*, Detroit, MI, May 1999.

[233] S. M. LaValle and R. Sharma. Motion planning in stochastic environments: Theory and modelling issues. In *IEEE Int'l Conf. On Robotics and Automation*, pp. 3057-3062, 1995.

[234] S. M. LaValle and S. A. Hutchinson. An objective-base stochastic framework for manipulation planning. In *Proc. IEEE/RSJ/GI Int'l Conf. on Intelligent Robots and Systems*, pp. 1772-1779, September 1994.

[235] J. M. Lehn. Supramolecular chemistry and nanotechnology. In *France-Japan workshop on from Nano to Macroscale Science and Technology through Micro Systems*, pp. 14-15, Apr. 1997.

[236] L. Lewis and A. Parkinson. Robust Optimal Design with a Second Order Tolerance Model. *Research in Engineering Design*, no. 6, pp. 25-37, 1994.

[237] M. A. Lewis, A. H. Fagg, and G. A. Bekey. Genetic algorithms for gait synthesis in a hexapod robot. In Yuan F. Zheng, editor, *Recent trends in mobile robotics*, Chapter 11, pp. 317-331. World Scientific, 1993.

[238] M. A. Lewis and G. A. Bekey, *"The Behavioral Self-Organization of Nanorobots Using Local Rules"*, In Proc. of IEEE Int'l Conf. on Intelligent Robots and Systems, pp. 1333-1338, Raleigh, NC, 1992.

[239] T. Lozano-Perez, M. T. Mason, and R. H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, Vol. 3, no. 1, pp. 3-24, 1984.

[240] V. J. Lumelsky and A. A. Stepanov. Path planning strategies for a point mobile automaton moving amidst unknown obstacles o arbitrary shape. *Algorithmica*, Vol. 2, pp. 403-430, 1987.

[241] M. I. Lutwyche. Highly Parallel Data Storage System Based on Scanning Probe Arrays. *Applied Physics Letters*, Vol. 777, pp.3299-3301, 2000.

[242] I. W. Lyo and P. Avouris. Filed-induced nanometer to atomic-scale manipulation of silicon surfaces with the STM. *Science*, Vol. 253, no. 5016, pp. 173-176, 12 July 1991.

[243] M. A. Lyshevski. Brownian Motor Analysis and its Application to Nanosystems. *IEEE Nano 2002 International Conference on Nanotechnology*, Washington, USA, pp. 151-155, August 2002.

[244] P. Maes. Behavior-based artificial intelligence. In Maes, P. (Ed.), *Designing Autonomous Agents*, MIT Press-Bradford Books, Cambridge, MA, 1990.

[245] J. H. Makaliwe and A. A. G. Requicha. Automatic planning of nanoparticle assembly tasks. *Proc. IEEE Int'l Symp. on Assembly and Task Planning*, Fukuoka, Japan, pp. 288-293, 2001.

[246] J. Mao and A. K. Jain. Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks*, Vol. 6, no. 2, pp. 296-317, March 1995.

[247] W. R. Mark, S. C. Randolph, and M. Finch. Adding force feedback to graphics systems: Issues and solutions. In *Computer Graphics Proceedings SIGGRAPH*, pp. 447-452, 1996.

[248] S. Martel, P. Madden, L. Sosnowski, I. Hunter, and S. Lafontaine. NanoWalker: a fully autonomous highly integrated miniature robot for nano-scale measurements. *Proc. of the European Optical Society and SPIE Int'l Symposium on Envirosense, Microsystems Metrology and Inspection*, Munich, Germany, Vol. 3825, pp. 64-76, 1999.

[249] V. Matellan, C. Fernandez, and J. M. Molina. Genetic Learning of Fuzzy Reactive Controllers. *Robotics and Autonomous Systems*, Vol. 225, no. 1-2, 33-41, 1998.

[250] K. Matsumoto. Room-Temperature Single-Electron Memory Made by Pulse-Mode Atomic Force Microscopy Nano Oxidation Process on Atomically Flat α-Alumina Substrate. *Applied Physics Letters*, Vol. 76, no. 2, pp. 239-241, 2000.

[251] L. Matthies. Mars microrover navigation : Performance evaluation and enhancement. *Proceedings of the IEEE/RSJ International Conf. on Robots and Systems (IROS)*, August 1995.

[252] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanet in nervous activity. *Bulletin of Mathematical Biophysics*, Vol. 5, no. 4, pp. 115-133, December 1943.

[253] D. McFarland and T. Bosser. *Intelligent Behavior in Animals and Robots*. Mit Press, 1993.

[254] U. Mehlhaus. Verteilte Programmierung zur Integration von Simulation und Steuerung von Robotern. Reprinted in *VDI Fortschrittsberichte*, Vol. 10, 1994.

[255] F. Menczer, and R. K. Belew. Evolving sensors in environments of controlled complexity. In *Artificial Life IV: Proceedings of the 4th International Workshop on Synthesis and Simulation of Living Systems*, Cambridge, MA: MIT Press, pp. 210-221, 1994.

[256] R. C. Merkle. A New Family of Six Degree of Freedom Positional Devices. *Nanotechnology*, Vol. 8, no. 2, pp. 47-52, 1997.

[257] R. C. Merkle. Nanotechnology and Medicine. *Advances in AntiAging Medicine*, Mary Ann Liebert Press, Vol. 1, pp. 277-286, 1996.

[258] J. A. Meyer and P. Husbands. Autonomous mobile robotics architecture for a functional approach. *First European Workshop on Evolutionary Robotics*. Springer Verlag, 1998.

[259] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 1996.

[260] O. Miglino, H. H. Lund, and S. Nolfi. Evolving Mobile Robots in Simulated and Real Environments. *Artificial Life*, Vol. 2, no. 4, pp. 417-434, 1996.

[261] S. A. Miller. Microelectromechanical Scanning Probe Instruments for Array Architectures. *Rev. Sci. Instrum.*, Vol. 68, no. 11, pp. 4155-4162, November 1997.

[262] W. T. Miller, R. Sutton, and Paul Werbos. *Neural Networks for Control*. MIT Press, Cambridge, Massachusetts, 1990.

[263] S. C. Minne. Automated Parallel High-speed Atomic Force Microscopy. *Appl. Phys. Lett.*, Vol. 72, no. 2340, pp.2340-2342, May 1998.

[264] M. Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, Vol. 65, no. 2, pp. 34-51, 1991.

[265] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[266] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proc. of Symposium on Interactive 3D Graphics*, pp. 392-398, 1995.

[267] H. Miyazaki and T. Sato. Pick and place shape forming of three-dimensional micro structures form fine particles. In *Proc. of the IEEE Int. Conf. on robotics and Automation*, pp. 2535-2540, April 1996.

[268] M. W. Moffett. Cooperative food transport by an asiatic ant. *National Geographic Research*, Vol. 4, no. 3, pp. 386-394, 1988.

[269] N. Mokhoff. Education Overhaul Urged for Nanotech Revolution. *EE Times*, February 2003, http://www.theworkcircuit.com/news/OEG20030206S0026 .

[270] J. M. Molina, A. Sanchis, A. Berlanga, and P. Isasi. An enhanced classifier system for autonomous robot navigation in dynamic environments. *Intelligent Automation and Soft Computing*, Autosoft Press, Vol. 6, no. 2, pp. 113-124, 2000.

[271] F. Mondada and P. I. Franzi. Mobile Robot Miniaturization: a tool for investigation in control algorithms. *Proc. of the Second International Conference on Fuzzy Systems*, San Francisco, USA, pp. 182-187, 1998.

[272] C. D. Montemagno and G. D. Bachand. Constructing nanomechanical devices powered by biomolecular motors. *Nanotechnology*, Vol. 10, no. 3, pp. 225-231, 1999.

[273] M. Moore and J. Wilhelms. Collision detection and response for computer animation. *ACM Computer Graphics*, Vol. 22, no. 4, pp. 289-298, 1988.

[274] S. K. Moore. Just One Word - Plastics. Special R&D Report, Organic Electronics, *IEEE Spectrum Magazine*, pp.55-59, September 2002.

[275] J. C. Moser. Pheromones of social insects. In *Control of Insect Behaviour by Natural Products*, Academic Press, pp. 161-178, 1970.

[276] V. Murino and A. Trucco. A Geometric Approach to the Surface Fitting Problem in Underwater 3-D Acoustic Images. *Measurement Science and Technology*, Vol. 10, No. 12, pp. 1135-1141, December 1999.

[277] A. R. Mushegian. The minimal genome concept. *Curr. Opin. Genet*, Vol. 9, no. 6, pp. 709-714, December 1999.

[278] Nasa Ames, *NASA Ames Computational Molecular Nanotechnology Team*, NAS Nanotechnology Gallery, http://www.nas.nasa.gov/Groups/SciTech/nano .

[279] National Science Foundation, http://www.eng.nsf.gov/sbir .

[280] B. J. Nelson and Y. Zhou. Task-based micromanipulation control strategies for assembly of hybrid mems. *In Workshop on Precision Manipulation at Micro and Nano Scales, IEEE Int. Conf. on Robotics and Automation*, pp. 5-29, 1998.

[281] N. J. Nilsson. *Learning Machines : Foundations of Trainable Pattern-Classifying Systems*. McGraw-Hill, New York, 1965.

[282] J. Nishi, Y. Uno, and Ryoji Suzuki. Mathematical models for the swimming pattern of a lamprey: II. Control of the central pattern generator by the brainstem. *Biological Cybernetics*, Vol. 72, no. 1, pp. 11-18, November 1994.

[283] J. Nishi, Y. Uno, and R. Suzuki. Mathematical models for the swimming pattern of a lamprey: I. Analysis of collective oscillators with time-delayed interaction and multiple coupling. *Biological Cybernetics*, Vol. 72, no. 1, pp. 1-9, November 1994.

[284] T. B. Norris. Nanoacoustics: towards imaging nanostructures using picosecond ultrasonics. *The Journal of the Acoustical Society of America*, Vol. 119, no. 5, pp. 3284-3285, May 2006.

[285] R. E. Tuzun, D. W. Noid, and B. G. Sumpter. Dynamics of a laser driven molecular motor. *Nanotechnology*, Vol. 6, no. 2, pp. 52-63, April 1995.

[286] M. Occello, M. C. Thomas. Intelligent control in robotics through real-time distributed blackboards, In *IMACS/SICE Int. Symp. on Robotics, Mechatronics and Manufacturing Systems'92*, pp. 567-572, Kobe, Japan, 1992.

[287] P. A. O'Dunlaing and C. K. Yap. A retraction method for planning the motion of a disc. *Journal of algorithms*, Vol. 6, pp. 104-111, 1982.

[288] T. Ondarcuhu and C. Joachim. Combing a nanofiber in a nanojunction. *Nanotechnology*, Vol. 10, no. 1, pp. 39-44, January 1998.

[289] J. B. Oommen, S. S. Iyengar, N. S. V. Rao, and R. L. Kashyap. Robot navigation in unknown terrains using learned visibility graphs. Part I: The disjoint convex obstacle case. *IEEE J. of Robot. & Autom.*, Vol. 3, no. 6, pp. 672-681, 1987.

[290] M. Overmars. *A random approach to motion planning*. Technical report, Dept. Computer Science, Utrect University, Utrect, The Netherlands, October 1992.

[291] I. J. Palmer and R. L. Grimsdale. REALISM: Reusable Elements for Animation using Local Integrated Simulation Models. In *Computer Animation'94*, Los Alamitos, CA, 1994, IEEE Computer Society Press, 1994.

[292] I. Pappas and A. Codourey. Visual control of a microrobot operating under a microscope. In *Proc. of the IEEE//RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 993-1000, 1996.

[293] A. Parkinson. Robust Mechanical design Using Engineering Models. *Transactions of ASME Journal of Mechanical Design*, Vol. 117, issue B, pp. 48-54, June 1995.

[294] A. Pentland. Computational complexity versus simulated environment. *Computer Graphics*, Vol. 22, no. 2, pp. 185-192, 1990.

[295] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, Vol. 23, no. 3, pp. 185-192, 1990.

[296] M. S. Phadke. *Quality Engineering using Robust Design*. Englewood Cliffs, New Jersey, Prentice Hall, 1989.

[297] D. Porath, A. Bezryadin, S. Vries, and C. Dekker. Direct Measurement of Electrical Transport Through DNA Molecules. *Nature*, Vol. 403, no. 6770, pp. 635-638, 2000.

[298] M. Ramia, D. L. Tullock, and N. P. Thien. The Role of Hydrodynamic interaction in the locomotion of microorganisms. *Biophys. J.* , Vol. 65, no. 2, pp. 755-778, August 1993.

[299] C. L. Ramsey and J. J. Grefenstette. Cased-based anytime learning. In D. W. Aha (Ed.), *Cased-based Reasoning: Papers from the 1994 Workshop.* Menlo Park, CA: AAAI Press, 1994.

[300] N. S. V. Rao, S. S. Iyengar, J. B. Oommen, and R. L. Kashyap. On terrain model acquisition by a point robot amidst polyhedral obstacles. *IEEE J. of Robot. & Autom.*, Vol. 4, no. 4, pp. 450-455, August 1988.

[301] RAPID Robust Accurate Polygon Interface Detection, http://www.cs.unc.edu/~geom/OBB/OBBT.html.

[302] W. E. Red. Minimum distances for robot task simulation. *Robotica*, Vol. 1, pp.231-238, October 1983.

[303] G. N. Reeke, O. Sporns, and G. M. Edelman. Synthetic neural modeling: The 'Darwin' series of recognition automata. *Proceedings of the IEEE Int'l Conf. on Automation and Control*, Vol. 78, no. 9, pp. 1498-1530, September 1990.

[304] J. Reif and Z. Sun. *Nano-Robotics Motion Planning and its Applications in Nanotechnology and Biomolecular Computing*. Department of Computer Science, Duke University, 2002, http://www.cs.duke.edu/~reif/paper/sunz/NanoRobotics.html .

[305] O. Renault, N. M. Thalmann, and D. Thalmann. A vision-based approach to behavioural animation. *Visualization and Computer Animation*, Vol. 1, no. 1, pp. 18-21, 1990.

[306] L. Reppesgaard. Nanobiotechnology: Die Feinmechaniker der Zukunft nutzen Biomaterial als Werkstoff. *Computer Zeitung*, no. 36, pp. 22, 2 September 2002.

[307] A. A. G. Requicha. Nanorobots, NEMS and Nanoassembly. *IEEE ICRA International Conference on Robotics and Automation*, Vol. 91, no. 11, pp. 1922-1933, December 2003.

[308] A. A. G. Requicha, R. Resch, N. Montoya, B. E. Koel, A. Madhukar, and P. Will. Towards hierarchical nanoassembly. *IEEE/RSJ Int'l Conf. on Intelligent Robots & Systems*, Kyongju, Korea, pp. 34-39, 1999.

[309] A. A. G. Requicha, C. Baur, and A. Bugacov. Nanorobotic assembly of two-dimensional structures. In *proc. of the IEEE Int. Conf. on Robotics and Automation*, Vol. 4, pp. 3368-3374, May 1998.

[310] R. Resch and C. Baur. Manipulation of nano particles using dynamic force microscopy: Simulation and experiments. *App. Phys*. A, Vol. 67, no. 3, pp. 265-271, September 1998.

[311] E. A. Rietman. *Molecular Engineering of Nanosystems*. Biological Physics Series, Springer-Verlag, New York, USA, 2001.

[312] Y. Rollot, S. Regnier, P. Bidaud, and J. C. Guinot. Some conditions of micromanipulation by adhesion. *In Proc. of the Symp. Franco-Israelien*, France, 1998.

[313] T. Rueckes. Carbon Nanotube-Based Nonvolatile Random Access Memory for Molecular Computing Science. *Science*, Vol. 289, no. 5476, pp. 94-97, July 2000.

[314] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Exploration in the microstructure of cognition*. MIT Press, Cambridge, MA, 1986.

[315] S. Russel and P. Norvig. *Artificial Intelligence: a Modern Approach*. Prentice-Hall, 1995.

[316] S. Saito, H. Miyazaki, and T. Sato. Pick and place operation of a micro object with high reliability and precision based on micro physics under SEM. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Vol. 4, pp. 2736-2743, 1999.

[317] A. Sanchis, J. M. Molina, P. Isasi, and J. Segovia. RTCS: a Reactive with Tags Classifier System. *Jounal of Intelligent and Robotic Systems*, Vol. 27, no. 4, pp. 379-405, April 2000.

[318] J. T. Jr. Santini. A Controlled-Release Chip. *Nature*, Vol. 397, pp.335-338, 1999.

[319] M. A. Sartori and P. J. Antsaklis. Implementations of learning control systems using neural networks. *IEEE Control Systems Magazine*, Vol. 12, no. 2, pp. 49-57, April 1992.

[320] K. Sasaki, H. Fujiwara, and H. Masuhara. Optical manipulation of a lasing particle and its application to near-field microsprectroscopy. *J. Vac. Sci. Technol. B*, Vol. 15, no. 6, pp. 2786-2790, Nov./Dec. 1997.

[321] T. Sato, J. Ichikawa, M. Mitsuishi, and Y. Hatamura. A new micro-teleoperation system employing a hand-held force-feedback pencil. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Vol. 2, pp. 1728-1733, May 1994.

[322] D. M. Schafer, R. Reifenberger, A. Patil, and R. P. Andres. Fabrication of two-dimensional arrays of nanometric-size clusters with the atomic force microscopy. *App. Physics Letters*, Vol. 66, no. 8, pp. 1012-1014, February 1995.

[323] J. Scheppach. Nanotechnik: Die Wunderwelt der winzigen Giganten. *P.M. Magazin*, pp. 18-24, Oktober 2002 http://www.pm-magazin.de/de/heftartikel/artikel_id293.htm .

[324] Scientific American. O Brasil na era da nanotecnologia. *Scientific American Brasil*, 26 September 2002, http://www2.uol.com.br/sciam/brasil.htm .

[325] N. C. Seeman. Nucleic Acid Junctions and Lattices. *J. Theor. Biol*. Vol. 99, no. 2, pp. 237-247, November 1982.

[326] T. D. Seely, S. Camazine, and J. Sneyd. Collective Decision-making in honey bees: how colonies choose among nectar sources. *Behavioral Ecology and Sociobiology*, Vol. 28, no. 4, pp. 277-290, April 1991.

[327] R. Sharma. Locally efficient path planning in an uncertain, dynamic environment using a probabilistic model. *IEEE Trans. Robot. & Autom.*, Vol. 8, no. 1, pp. 105-110, February 1992.

[328] A. Shill. *DCE - Das OSF Distributed Computing Environment*. Springer-Verlag, 1993.

[329] M. Sitti, S. Horiguchi and H. Hashimoto. Controlled Pushing of Nanoparticles: Modeling and Experiments. *IEEE/ASME Trans. on Mechatronics*, Vol. 5, no. 2, pp. 199-211, 2000.

[330] M. Sitti and H. Hashimoto. Teleoperated nano scale object manipulation. In *Recent Advances on Mechatronics*, Springer Verlag Pub., Singapore, pp. 322-335, 1999.

[331] M. Sitti and H. Hashimoto. Two-dimensional fine particle positioning under optical microscope using a piezoresistive cantilever as a manipulator. *J. of Micromechatronics*, Vol. 1, no. 1, pp. 25-48, 2000.

[332] M. Sitti and H. Hashimoto. Two-dimensional fine particle positioning using a piezoresistive cantilever as a micro/nano-manipulator. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Detroit, pp. 2729-2735, May 1999.

[333] M. Sitti, M. Hoummady, and H. Hashimoto. Trends on mechatronics for micro/nano telemanipulation: Survey and requirements. In *IFAC Information Control in Manufacturing 1998*, edited by G. Morel and F.B. Vernadat, Pergamon Pub., Belgium, pp. 235-240, 1998.

[334] H. W. Six and D. Wood. Counting and reporting intersections of d-ranges. *IEEE Trans. on Computers*, Vol. 31, no. 3, March 1982.

[335] V. Skala, M. Kuchar, and Jan Hradek. Geometry Reconstruction in Rapid Prototyping with Hash Function. In *Proc. of IASTED International Conference on Computer Graphics and Imaging*, Honolulu, USA, pp.240-245, August 2001.

[336] G. D. Skidmore, M. Ellis and J. von Ehr. Free Space Construction with Carbon Nanotubes. *Science and Application of Nanotubes*, Springer-Verlag, NY, 2000.

[337] B. Soucek. *Neural and Concurrent Real-Time Systems*. John Wiley & Sons Inc., NY, 1989.

[338] R. W. Stark and S. Thalhammer. The AFM as a tool for chromosomal dissection - the influence of physical parameters. *Appl. Phys. A*, Vol. 66, issue S1, pp. 579-584, 1998.

[339] L. Steels. Building agents out of autonomous behavior systems. In L. Steels and R. Brooks (Eds.), The "artificial life" route to "artificial intelligence". *Building situated embodied agents*, pp. 102-137, Lawrence Erlbaum, New Haven, 1993.

[340] R. Stenger. Who should explore space, man or machine?. *CNN Technology*, 18th Feb. 2003, http://www.cnn.com/2003/TECH/space/02/18/sprj.colu.space.future/index.html .

[341] A. Stentz. Optimal and efficient path planning for partially-known environments. In *IEEE Int. Conf. Robot. & Autom*., Vol. 4, pp. 3310-3317, May 1994.

[342] R. Stracke, K. J. Böhm, J. Burgold, H. Schacht, and E. Unger. Physical and Technical parameters determining the functioning of a knesin-based cell-free motor system. *Nanotechnology* 11, UK, pp. 52-56, 2000.

[343] J. A. Stroscio and D.M. Eigler. Atomic and molecular manipulation with the scanning tunneling microscope. *Science*, Vol. 254, no. 5036, pp. 1319-1326, November 1991.

[344] D. Sturman. A discussion on the development of motion control systems. In *SigGraph Course Notes: Computer Animation: 3-D Motion Specification and Control*, number 10, 1987.

[345] J. H. Sudd. The transport of prey by ants. *Behaviour*, Vol. 25, no. 3-4, pp. 234-271, 1965.

[346] J. H. Sudd. How insects work in groups. *Discovery*, Vol. 25, pp. 15-19, June 1963.

[347] J. H. Sudd. The transport of prey by an ant, pheidole crasindoa. *Behaviour*, Vol. 16, no. 3-4, pp. 295-308, 1960.

[348] J. Sun, M. Gao, and J. Feldmann. Electric Field Directed Layer-by-Layer Assembly of Highly Fluorescent CdTe Nanoparticles. *Journal of Nanoscience and Nanotechnology*, Vol. 1, no. 2, pp. 21-27, 2001.

[349] P. Svestka, *A probabilistic approach to motion planning for car-like robots*. Technical report, Dept. Computer Science, Utrect Univ., Utrect, The Netherlands, April 1993.

[350] K. R. Symon. *Mechanics*. Third Edition, Addison-Wesley, Reading, MA, 1971.

[351] H. Takeda and J. C. Latombe. Sensory uncertainty field for mobile robot navigation. In *IEEE Int. Conf. Robt. & Autom.*, pp. 2465-2472, Nice, France, May 1992.

[352] T. Tanikawa, T. Arai, and T. Masuda. Development of micro manipulation system with two-finger micro hand. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Vol. 2, pp. 850-855, November 1996.

[353] S. L. Tanimoto. *The Elements of Artificial Intelligence Using Common Lisp*. 2nd Edition, Computer Science Press, 1995.

[354] S. J. Tans. Individual Single-Wall Carbon Nanotube as Quantum Wires. *Nature*, Vol. 386, pp.474, April 1997.

[355] G. Theraulaz and E. Bonabeau. Coordination in distributed building. *Science*, Vol. 269, no. 4, pp. 686-688, August 1995.

[356] W. Thibault and B. Naylor. Set operations on polyhedra using binary space partitioning trees. *SIGGRAPH'87 Computer Graphics*, Vol. 21, no. 4, pp. 153-162, July 1987.

[357] A. Thompson. On the automatic design of robust electronics through artificial evolution. In M. Sipper (Ed.), *Proceedings of the 2nd International Conference on Evolvable Systems: From biology to hardware (ICES98)*, Spriger-Verlag, Berlin, pp. 13-24, 1998.

[358] D. A. Tomalia. Starburst Cascade Dendrimers - Fundamental Building-Blocks for a New Nanoscopic Chemistry Set. *Advanced Materials*, Vol.6, no. 7-8, pp. 529-539, 1994.

[359] T. C. Tsao and M. G. Safonov. Unfalsified direct adaptive control of a two-link robot arm. *International Journal of Adaptive Control and Signal Processing*, Vol. 15, no. 3, pp. 319-334, May 2001.

[360] G. Turk. *Interactive collision detection for molecular graphics*. Master's thesis, Computer Science Department, University of North Carolina at Chapel Hill, 1989.

[361] S. W. Turner, A. M. Perez, A. Lopez and H. G. Craighead. Monolithic Nanofluid Sieving Structures for DNA Manipulation. *J. Vac. Sci. Technol. B*, Vol. 16, no.6, pp.3835-3840, 1998.

[362] H. Uchida, D.H. Huang, J. Yoshinobu, and M. Aono. Single-atom manipulation on the si(111)7x7 surface by the STM. *Surface Science*, Vol. 287-288, part 2, pp. 1056-1061, May 1993.

[363] K. P. Unnikrishnan and K. P. Venugopal. Alopex: A correlation-based learning algorithm for feedforward and recurrent neural networks. *Neural Computation*, Vol. 6, no. 3, pp. 469-490, 1994.

[364] Jr. G. Vanecek. Brep-index: A multi-dimensional space partitioning tree. *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD Applications*, Austin Texas, pp. 35-44, 1991.

[365] S. Voss. Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization. *Meta-Heuristics International Conference*, Kluwer Academic Pub, 1998.

[366] J. Walker. Pentagon: $1 million prize in robot race. *CNN Technology*, January 13, 2003. http://edition.cnn.com/2004/TECH/ptech/03/14/darpa.race/index.html, www.darpa.mil/grandchallenge .

[367] B. Watson, J. Friend, and L. Yeo. Piezoelectric ultrasonic resonant motor with stator diameter less than 250 μm: the Proteus motor. J*ournal of Micromechanics and Microengineering*, Vol. 19, no. 2, 022001 (5pp), January 2009.

[368] M. Wautelet. Scaling laws in the macro-, micro- and nanoworlds. *European Journal of Physics*, Vol. 22, no. 6, pp. 601–611, 2001.

[369] L. Weber, S. Wallbaum, C. Broger, and K. Gubernator. Optimization of the Biological Activity of Combinatorial Compound Libraries by a Genetic Algorithm. *Angew. Chem. Int. Ed. Engl.*, Vol. 34, no. 20, pp. 2280-2282, December 2003.

[370] L. L. Whitcomb. Underwater Robotics: out of the research laboratory and into the Field. *IEEE Int'l Conf. on Robotics and Automation*, pp. 85-90, April 2000.

[371] D. Whitley and T. Starkweather. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, Vol. 14, no. 3, pp. 347-361, 1990.

[372] H. K. Wickramasinghe. Scanned-Probe Microscopes. *Scientific American*, Vol. 261, pp.98-105, 1989.

[373] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, Vol. 8, no. 3-4, pp. 229-256, May 1992.

[374] R. H. Wilson and J. F. Rit. Maintaining Geometric Dependencies in Assembly Planning. In L. S. Homem de Mello and S. Lee, editors, *Computer-Aided Mechanical Assembly Planning*, pp. 217-242, Kluwer, 1991.

[375] S. W. Wilson. The animat path to AI. In *First International Conference on Simulation of Adaptive Behavior*, MIT Press, pp. 15-21, 1991.

[376] M. Woo, J. Neider, T. Davis, and D. Shreiner. *OpenGL Programming Guide*. 3rd Edition, Addison-Wesley, Boston, USA, 1999.

[377] C. Wurll, D. Henrich, H. Wörn. Parallel on-line Motion Planning for Industrial Robots. *3rd ASCE Specialty Conf. on Robotics for Challenging Environments, Robotics 98*, New Mexico, USA, pp. 314-320, 1998.

[378] T. Yamamoto. Molecular surgery of DNA using restriction enzymes. In *Proc. of the France-Japan Workshop on from Nano to Macro Science and Tech. Through Microsystems*, pp. 38, 1998.

[379] H. Yan, X. Zhang, Z. Shen, N. C. Seeman. A Robust DNA Mechanical Device Controlled by Hybridization Topology. *Nature,* Vol. 415, no. 6867, pp. 62-65, January 2002.

[380] W. Zesch and R.S. Fearing. Alignment of microparts using force controlled pushing. In *SPIE Conf. on Microrobotics and Micromanipulation*, Boston, Vol. 3519, pp. 148-156[, November 1998.

[381] Y. Zhou, B. J. Nelson, and B. Vikramaditya. Fusing force and vision feedback for micromanipulation. In *Proc. of the IEEE Int. Conf on Robotics and Automation*, Vol. 2, pp. 1220-1225, May 1998.