Autonomously Finding New Control Modes for the Rowing Blade “Ro” by an Agent-Oriented Evolutionary System

Rogério Perino de Oliveira Neves

Advisor: Prof. Dr. Tsugukiyo Hirayama

Department of Systems Design for Ocean-Space
Yokohama National University

Japan, July 2006

This thesis is submitted to the Faculty of the Graduate School of Yokohama National University, in partial fulfillment of the requirements for the degree of Doctor of Engineering. This thesis is entirely my own work and except where otherwise stated describes my own research.

Rogério Perino de Oliveira Neves

Acknowledgements

In first place, I would like to express gratitude to my supervisor, Professor Dr. Tsugukiyo Hirayama, whose guidance was fundamental for this research.

Secondly, I would like to thank Prof. Hirakawa and Prof. Takayama for the assistance they provided in developing this work. Their expertise and technical support were fundamental in many levels of this project, such as designing models and planning experiments.

I must acknowledge ex-members of the laboratory who provided support and friendship, helping me in many aspects of my adaptation to life in Japan, Dr. Nomiyama, Dr. Nishimura and Mr. Kabuto.

I recognize that this research would not have been possible without the financial assistance of Ministry of Education of Japan, the technical infrastructure of Japanese Institute of Navigation (JIN), the hospitality of Yokohama National University Foreign Student House (Gumyoji Kaikan) and the guidance of Yokohama National University Ryugakusei Center’s professors and employees. I hereby express my gratitude to those institutions.

Finally, I would like to dedicate this thesis to my beloved Saori, whose guidance and support were essential in every aspect of my life in Japan, whose love gave me inspiration, whose shine enlightened my path, and whose presence was my safe haven, and without her I could never have finished this work.

Autonomously Finding New Control Modes for the Rowing Blade “Ro” by an Agent-Oriented Evolutionary System

Abstract

Defining control systems for previously human performed tasks requires experienced operators working together with engineers and/or programmers, who apply their particular expertise to solve the specific control problem under consideration. Yet, this procedure generates human-oriented solutions, as result the defined controller will simulate human conducted control rather than generate optimized, machine-oriented signals specific for the electro-mechanical system. In addition, for unparalleled designs, such as new or improved control systems, not based in humanistic processes, the inexistence of operators renders the traditional human definition processes inefficient. Furthermore, many control systems involves a huge number of inter-dependent variables, incompatible with the familiar four-dimension coordinate visualization, making visualization of dependencies incomprehensible for human abstraction and therefore unsuitable for conventional treatment.

We present here an alternative, autonomous control system training model, based on natural evolution, Multi-Agent Systems and Distributed Computing, which is able to profit on modern computer architectures, such multi-processed and distributed systems, to deal with evolutionary search issues in a reduced timeframe. The method produces optimized machine-specific oriented control codes by simultaneously exploring the multi-variable space with multiple instances of agents, in a coordinated search in a higher resolution than traditional Genetic Algorithms.

Here we describe the method in practice, as we apply it to solve the “Ro” control problem, a simple one-oar robotic rowing system designed for experimentation, that allow us to test and compare the autonomously discovered control modes with the traditional, human conducted rowing.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

DF	Degrees of Freedom
FS	File-system
GA	Genetic Algorithm
NP	Number of Processors
MAS	Multi-Agent Systems
OS, DOS	Disk Operation System
RO-BOT	Robotic “Ro” Rowing Mechanism

List of Symbols

α	*Blade horizontal position (along X axis)*
β	*Blade elevation (along Y axis)*
θ	*Blade rotation*
$\vec{L}$	*Lift force*
$\vec{D}$	*Drag force*
$\vec{F}$	*Total Impulse Vector (Lift + Drag)*
$\hat{n}$	*Unitary vector normal to the profile*
$\hat{d}$	*Unitary vector pointing the direction of evaluation*
p	*Value of pressure*
$\partial\Omega$	*Frontier of the domain*
Px	*Point coordinate along X*
Py	*Point coordinate along Y*
Pz	*Point coordinate along Z*
POS_0	*Positioning for actuator 0*
POS_1	*Positioning for actuator 1*
POS_2	*Positioning for actuator 2*

Chapter 1

Introduction

Overview

In this initial chapter the goal and main motivations for developing an autonomous training system for the robotic Ro control and a quick overview of the current available research in the field are presented. The traditional and current approaches are explained and the thesis organization is also presented.

1.1. Objective

The objective of this research is to implement and experiment an autonomous definition method to search for the optimal, machine-oriented combination of signals to efficiently control a robotic Ro.

1.2. Motivation

As genetic algorithms become increasingly popular, they are applied to higher complex problems that may require considerable computations [1]. In cases where the GA involves calling to complex fitness functions, such as the hydrodynamic test considered here, parallel implementations become necessary to reach high-quality solutions in reasonable times.

Reducing the evaluation time requires massive computer power that is often unavailable. As alternative to multi-processed systems, that are expensive and rare, computer networks are widely available and relatively cheap. Likewise, computer networks are usually available in working and researching environments. The possibility of using an already available computer network to expand search capabilities was one of the main motivations for this project [2] [3].

In order to use the network to achieve solutions in a reduced timeframe a parallel search procedure need to be coordinated [6]. Multi-agent systems offer an elegant and efficient new approach to handle the problem. Object-oriented programming (OOP) is employed to enable agent to agent communication locally and across systems. A Multi-Agent approach inherits concepts such as autonomy, mobility and cooperation [7], as the Agent-oriented algorithm spreads to simultaneously search the multi-dimensional space, migrating among systems as new computers become available, while constantly communicating between instances to keep track of the progress. Once the agents work cooperatively, they are constantly communicating by changing objects, in order to avoid redundant area searches and recalculations of already evaluated points, OOP is a well established programming doctrine that enables the communication of objects even in such distributed environments.

1.3. Previous Works

The concept of evolutionary search by genetic algorithms is quickly spreading through different areas of science, as the computer industry offers increasing computer power for decreasing costs, the upcoming availability allows an evolutionary approach [1] to be put in use for a wider, more complex range of problems. But as the complexity of the algorithms and evaluation functions increase, the hardware and time requirements grow exponentially [2].

Innumerous researches are in progress in order to use newly available multi-processed systems to achieve hi-quality solutions in reasonable times, in an effort named Parallel Genetic Algorithms or PGA [6]. Yet, computer systems with large number of processors have high costs and short lifespan [2]. Nevertheless the efforts have given many working solutions [15] [17], faster and cheaper models of parallel genetic systems are yet to come. Here, an alternative is presented.

1.4. Approach

Usually referred as sculling, propelling a ship with one single oar positioned in her stern is very popular in Asian countries, such as China and Japan.

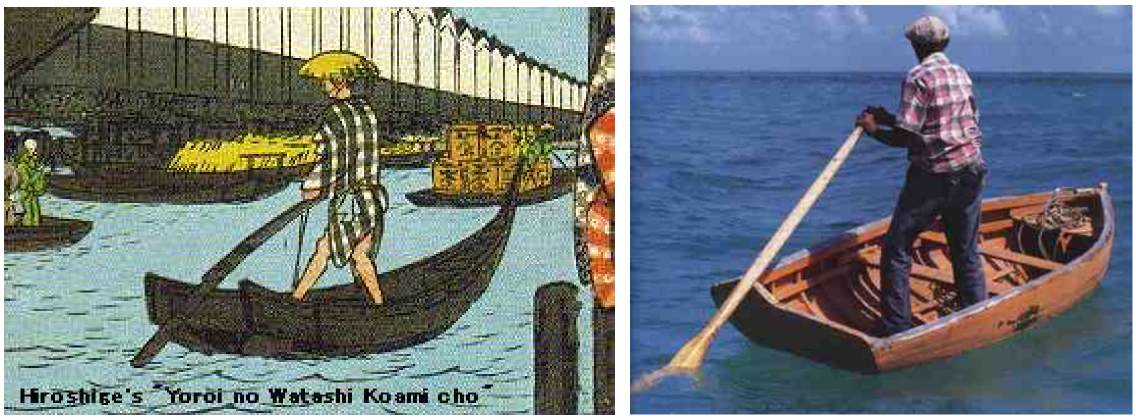


Figure 1 – Hiroshige's "Yoroi no Watashi Koami cho" and a fisherman depicts a Ro in Japanese art and in practical use nowadays.

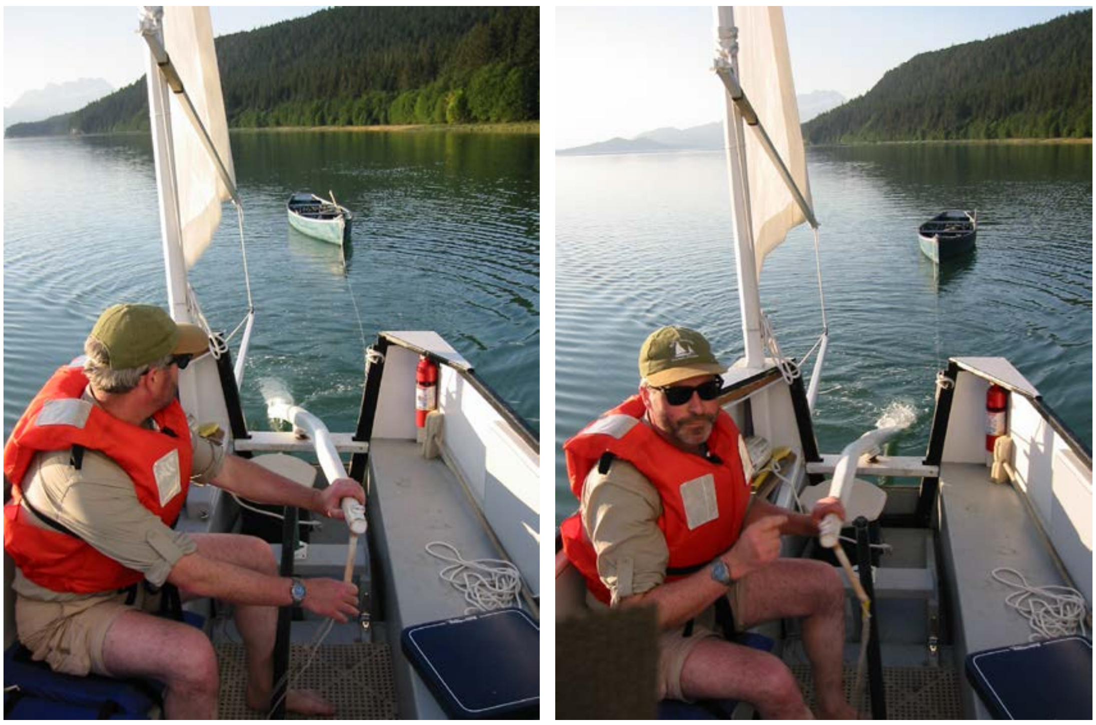


Figure 2 – Modern boat propelled by sculling a western version of Ro.

The Ro origins are unknown, but it is known from ancient times as it is depicted in many works dating from several ages of history, especially in Japan, China and ancient Egypt, Figure 1 shows a depiction of Ro and an actual photo. Today it can still be found in use not only in traditional folkloric regions, as shown in Figure 2.

At first glance, the humanistic control may look intuitive as it is ease to perform, in a way that one can quickly learn by a simple trial and error process. The definition of control signals for a robotic actuator, on the contrary, needs a skilled operator, capable of describing the linguistic operation rules in a comprehensible fashion, in order to a programmer to be able to implement such rules into the software of a controller.

In the traditional method, the programmer adjusts manually the set of command control codes to be sent to the actuators, interactively, until the desired result is obtained, matching the procedure described by the operator. The operator/programmer will most likely get the hardware to mimic human-performed control, producing human-oriented signals in the process. This method is based purely on observation and intuition and always produces humanistic results, rather than optimal results specifically designed for the hardware in consideration.

To solve the Ro problem, we propose the use of an autonomous training method, based on natural evolution [1], Genetic Algorithms (GA) [4], Multi-Agent Systems (MAS) [7] and computer simulation for evaluation of fitness in a distributed framework [2] [3], which is able among other advantages to find optimal machine-oriented solutions for scrutinized control problems.

The described method is specially recommended to address systems that involve, for instance: Dynamical control, multi-dimensional spaces, multiple coordinate systems, inter-dependent variables and an unknown best solution [4] [5], and such liken systems that often challenge human visualization capabilities.

The method allows obtaining a variety of different solutions with different benefits each, by scanning simultaneously several regions of the multi-variable space, optimizing the system by different sets of criteria, such max speed, acceleration, energy, fuel consumption, endurance, attrition, etc [4].

The current application is a simple example of control system, which allows us to compare the proposed procedure with the traditional, human-performed procedure. Once it is proven ship-worthy, the same method can be extended to treat more complex control systems by applying these same guidelines (e.g. navigation, collision avoidance, docking procedures).

1.5. Thesis Organization

The objective, motivations and a brief introduction to the approach adopted in this research are described in this first chapter named Introduction.

In Chapter 2, the fundamental concepts and technologies utilized are explained. The chapter gives a quick explanation of each topic being applied in this work, making further chapters easier to comprehend.

The implementation of the project is then explained in detail in Chapter 3.

Chapter 4 shows the results obtained into the simulation, explaining each mode and its particular characteristics.

Chapter 5 shows the results obtained by applying the previously found modes in the actual robotized model, and compare the simulated results with the experimental results.

In the Chapter 6, an analysis is presented by the computational point of view, pointing the advantages in using the Multi-Agent approach to solve the problem n a reduced time.

A discussion about the experiment, as well as the final conclusions and suggestions for future works are presented in Chapter 7.

Chapter 2

Fundamental Concepts

Overview

In this chapter the fundamental concepts applied in the current framework are introduced. A quick review of the theory and practical applications of each discipline are explained in order to facilitate comprehension of the implementation process. The information presented is summarized to describe solely the concepts applied in this framework. A comprehensive explanation as well as the state of the art research in each of the related areas can be found in the provided references.

2.1. Working with genetic Algorithms

Genetic Algorithms take its roots on Darwin's theory of evolution, and, in addition to prove Darwin's concepts, it provides engineers with a powerful tool to solve a sort of problems. Even though their mechanics are simple, Genetic Algorithms are complex non-linear algorithms that are controlled by many parameters, which are not always understood [6]. Parameters are related to population, distribution trough space, crossover and mutation rates, and not always have well defined values, changing according to particular attributes of one specific algorithm, some values may only be found by successive trial.

The baseline is that in a gene-pool, by always selecting the best individuals according to some pre-defined criteria, we introduce an artificial-selection. In addition, applying noise and scatter operators to modify the remaining individuals it leads to successively better solutions.

The gene-pool is the collection of all available individuals, divided in populations and represented in a string format, also called DNA. The concept of DNA is that a data structure is common to each and every individual and describes its particular characteristics. For engineering, it is merely a string of variables representing the many aspects of the addressed problem. Segments of the string that express some specific feature are called CHROMOSOMES and can be of any sub size of the string. The values in the gene pool can be started randomly or aided manually, by adding some already known good or partial solutions in order to speed the process. However, adding human defined points may cause a system addiction [1].

It is interesting to provide more than one mechanism to change the gene-pool, increasing the possibility of improving, or evolving the existing population. Once again, nature already has given useful answers. The most popular operators: Mutation (add noise) and Crossover (recombination). In short, each operator performs as described:

- Mutation: Introduces a random factor at some point of the DNA array, allowing for the possibility of increase the overall efficiency. Figure 3 shows examples of mutation operators.

- Crossover: Combine two existing DNA arrays into new array or arrays, the resulting solution will be a hybrid of the predecessors, containing features from both. Figure 4 illustrates the crossover operator in use.

DNA	CROMOSSOMES			
ORIGINAL	FF FA	1C A8	B6 06	0A 51 …
MUTATED P(10%)	FF FA	1C A8	B6 09	0A 51 …
MUTATED P(01%)	FF FA	E3 1F	B6 06	0A 51 …

Figure 3 – Example: a DNA array mutated by (bitwise) noise operator, large probability of a small change, small probability of a big mutation.

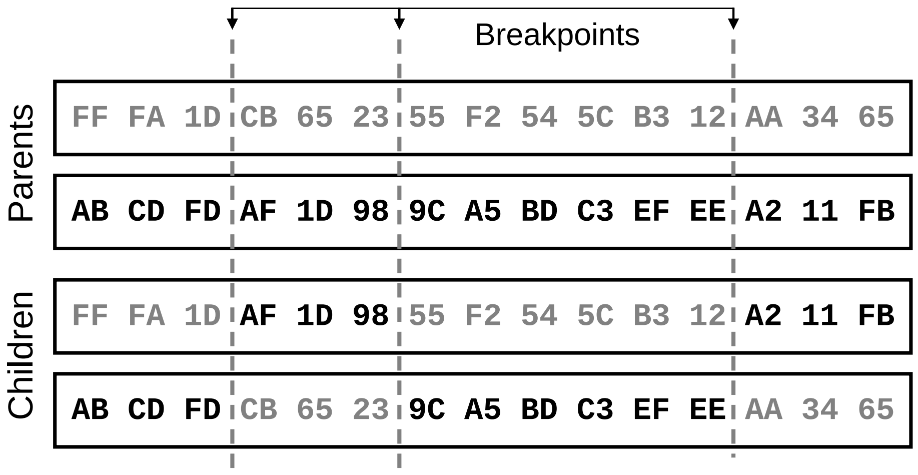

Breakpoints
Parents
Children
FF FA 1D CB 65 23 55 F2 54 5C B3 12 AA 34 65
AB CD FD AF 1D 98 9C A5 BD C3 EF EE A2 11 FB
FF FA 1D AF 1D 98 55 F2 54 5C B3 12 A2 11 FB
AB CD FD CB 65 23 9C A5 BD C3 EF EE AA 34 65

Figure 4 – Example of crossover operation, here two possible rearrangements are generated with random breakpoints.

Considering a multi-dimensional coordinate system, shaped by an efficiency function, the two rules represent consecutively a random walk in space, allowing the algorithm to explore the region for performance peaks, and a random jump, allowing it to explore different areas of the solution space.

One of the most important factors that determine the performance of the genetic algorithm performs is the diversity of the population. If the average distance between individuals is large, the diversity is high; if the average distance is small, the diversity is low. Getting the right amount of diversity is a matter of trial and error. If the diversity is too high or too low, the genetic algorithm might not perform well.

By default, the Genetic Algorithm creates a random initial population using the creation function. One can specify the range of the vectors in the initial population in the Initial range field in Population options. In the current approach, Agents will have different ranges for parallel populations, with localized diversity in each partition of the multi-dimensional space. The details are presented in the next sections.

By constantly applying the rules (select, mutate and crossover) new different solutions are achieved. A small percentage of these solutions statistically lead to better efficiencies, the artificial-selection criteria will grant that only the improved ones survive. So, if in a new generation we achieve a solution that is, to say, 1% better than in the previous generation, it will be kept, and the old, obsolete solutions will eventually be discarded.

The GA approach produces improved solutions for each generation, what generally leads to the optimal set. However, the time required to achieve the optimal combination is theoretically impossible to predict, due to the random nature of the improvement. As odd as it sounds, the optimal set can be achieve in a few generations, or only achieve after scanning considerable amount of the multi-variable space. Since the optimal value is unknown, the maximum efficiency progression must be observed, when its value seems unchanged for several generations (umber depends on previous progress history), it may indicate that a local or global maximum efficiency was reached. The result in any case is an optimal set or a sub-optimal set. For small, localized populations, the system may get stuck in some sub-optimal area of the space, never reaching out for the global optimum. Resetting the whole gene-pool a few times may spot different solutions, or prove some recursively achieved solution is indeed the optimal solution for the problem. Considering that the population may in any case get trapped into a sub-optimal region of the space, several randomly initiated attempts are recommended to confirm whatever the values found are indeed the optimal set of values for the algorithm.

Figure 5 shows a simple example of GA operation over a 2 dimensional surface data vs. fitness, in a three-dimensional plot.

Detailed information about GA, its roots, concepts and applications can be found in [1], [4], [5] and [6].

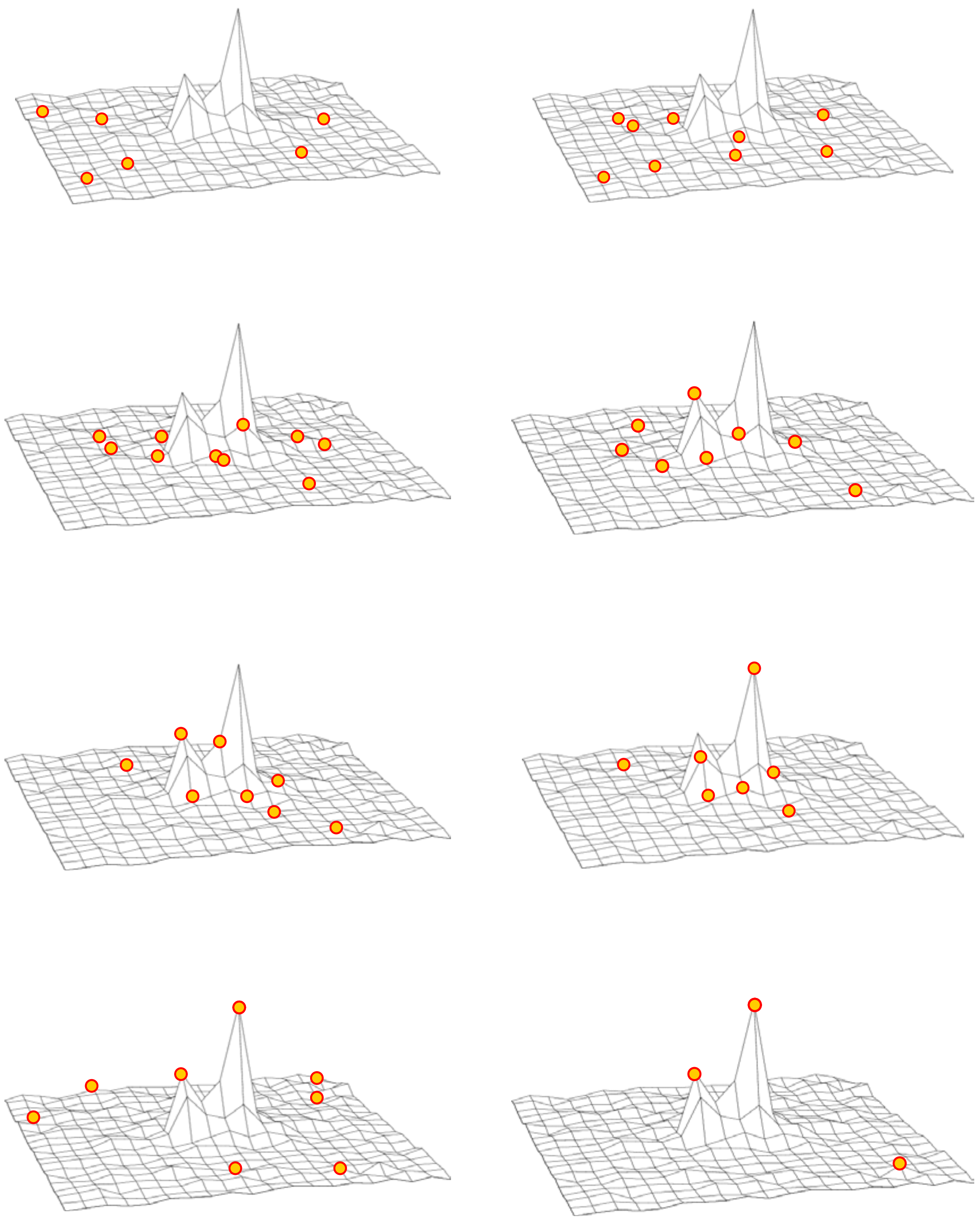


Figure 5 – In an example of GA operation over a two-dimensional data, some individuals are scattered over the area containing the local maximum. In time by reproduction and random crossover some of the individuals achieve higher fitness values and decimate the inferior population, identifying the highest points.

2.2. Electronic Multi-Agents

In the theory of Electronic Multi-Agents, agents are small pieces of program imbued with one or more objectives. The agents cooperate with each other in order achieve the specified objective. They have a set of rules or procedures, called methods, which helps them to perform special functionality and communications, in order to accomplish such objectives.

The theory is not well defined and there is still many controversial definitions about the main concepts that define an electronic agent. Despite many approaches can converge in a common end, while emphasizing different aspects, two main lines of research can be distinguished: the first focuses on the building of individual intelligences whose communication is organized, whereas the second imagines very simple entities whose co-ordination emerges in time without the agents being conscious of it. In fact, a huge number of different schools of MAS persist, all coming from different theoretical backgrounds. These include the American DAI school (Lesser, Gasser, Sycara), the Rational Agents branch (Rao and Georgeff, Shoham, Castelfranchi), the branch focusing on Speech Acts (Finin), on Petri nets (Estraillier), the Reactive Agents branch (Brooks, Steels, Drogoul, Ferber, Demazeau) and those focusing on learning (Weiss and Sen) [7]. These researches, although having different points of view, are very complementary, and each one has their own selection of applications.

The main applications of multi-agent systems at the moment are [7]:

Problem Solving: As an alternative to centralized problem solving, either because problems are themselves distributed, or because the distribution of problem solving between different agents reveals itself to be more efficient way to organize the problem solving. The reason for its broad application is neither: It can be flexible and allow failures in the system, or, in some cases, it is the only way to solve the problem.

Multi-Agent Simulation: Simulation is widely used to enhance knowledge in biology or in social science and MAS gives us the possibility to make artificial universes that are small laboratories for the testing of theories and experiments.

Construction of Synthetic Worlds: These artificial universes can be used to describe specific interaction mechanisms and analyze their impact at a global level in the system. The entities that are represented are usually called ANIMATS, since they are mainly inspired by animal behaviors (hunting, searching or gathering habits). The aim of this research is to have societies of agents that are very flexible and can adapt even in cases of individual failure. (For example, when robots are sent on an expedition and they are required to be very independent from the instructions they could receive.)

Collective Robotics: Defining the robots as MAS where each subsystem has a specific goal and deals with that goal only. Once all the small tasks are accomplished the big task is too. The MAS approaches can also be used in the co-ordination of different mobile robots in a common space.

Kinetic Program Design: MAS can also be seen as a very efficient modular way to program, especially in an Object Oriented Programming environment.

In the present application, the agents will be used to perform a distributed search effort in a multi-dimensional space. To each will be given a population with a random, localized gene-pool and the GA rules to alter and operate the individuals in this population. The agents have mobility, meaning agents can migrate to other computers as new processing power become available. They can travel and communicate among themselves trough the network or even the internet, depending on the selected topology for each execution, allowing the use of all available means to spread the load for the task.

The comprehensive foundations for the MAS theory can be found in [7], [8], [11] and [12]. Practical applications of GA can be found in [15], [17] and [19].

2.3. Object Oriented Programming

Object Oriented Programming or OOP is a programming doctrine that quickly spread among programmers worldwide, due to the simplicity and practicality in organizing complex, multi-modular codes into several separate objects resembling the real world.

Objects are the key to understanding object-oriented technology. Comparing to real-world objects, Objects in OOP share two characteristics: They all have state and behavior. For example, cats have states, such name, color, breed, hungry and behaviors such eating, sleeping, etc. Software objects are modeled after real-world objects in that they too have state and behavior. A software object maintains its state in one or more variables. A variable is an item of data named by an identifier. A software object implements its behavior with methods. A method is a function (subroutine) associated with an object. By this definition, an object is a software bundle of variables and related methods.

Real-world objects can be represented by using software objects [10]. It is also possible to use software objects to model abstract concepts. For example, an event is a common object used in window systems to represent user actions, such a mouse button or a key press.

A single object alone generally is not very useful. Instead, an object usually appears as a component of a larger program or application that contains many other objects. Through the interaction of these objects, programmers achieve higher-order functionality and more complex behavior. Software objects interact and communicate with each other by sending messages. Sometimes, the receiving object needs more information so that it knows exactly what to do. This information is passed along with the message as parameters.

Recent OOP technologies allow exchanging of messages to occur in distributed environments, using the network to transport the messages to remote objects in several interconnected computers [2] [3] [16]. Those technologies are essential for the current project.

2.4. Parallel Computing

One of the advantages of employing MAS is that, in a multi-threaded environment, you can distribute several agents among the available processors and systems in order to obtain direct speed-ups. Now, lets suppose we have available a network of computers that stays idle for most of the time, or by chance just underachieve their full processing potential with desktop applications, at least in a specific time period, let’s say (i.e. during the night or holydays). It is natural to assume that their idle resources can be allocated to help solving the problem in a fraction of time. By porting some code to the target computer, agents can span to the connected machines through the network as their processors becomes idle. A small deployed executable evaluates the communication between the client (where the agents are migrating to) and the server (the primary holder of resources).

The implementation of parallel programs can be tricky and complex. Different from linear algorithms, parallel systems may have a series of peculiarities in the debugging process, such access problems and violations to public memory, and often require to the programmer to implement a series of safeguard routines, such signalizations and access privileges. The implementation can be facilitated by following common directives clearly and concisely presented in [2] and [3].

Three concepts are most important when working with distributed computing [3]:

- Granularity: refers to population, and how the system processors are allocated. When a large number of threads execute in few processors, you have a high granularity. Granularity is important to balance the performance of individual tasks.

- Partitioning: refers to the segmentation of the search space. Several topologies can be used, but better results are obtained if the partitioning is linked to the function and the way individuals explore the space.

- Communication: In parallel computing, refers to the way neighboring partitions exchange information, the concept differs from agent communication, principle applied in the current work.

Under the current project, the program executing in the client computers is no more than a re-compilation of the server-side code, but with modified code in order to target the server when operating global resources, such environment variables and the solution database. Through these resources, the client program evaluates the necessary communications, get directions for the search and informing the server of current progress. In a client-server structure, the server computer must be always operational in order to the communication among remote agents to function properly. In contraposition, clients can break operation abruptly without compromising the overall system integrity.

In the agent communication approach, the partitions of the multi-variable space are not well defined. Frequently individuals of an agent’s population initially allocated to a partition invade the neighboring partition to explore the space. These individuals are not stopped, unless they hit one particular point where evaluation was already performed.

2.5. Database

Database as referred here consists of a list of tested solutions and respective evaluation results, used to store DNA (solutions or individuals in GA) representing points already visited by the algorithm, avoiding redundant, time-consuming computations by the simulation method. Despite quick access database systems are available, such Oracle, SQL and JDBC, those constitute a much over-dimensioned database solution than actually needed for the present problem, once concepts such normalization and cross reference are not applied in this approach. Instead, here, we rely on the OS file-system to store files containing the test information. Most recent file-systems, such NTFS and ext3 and RaiserFS provide support for metadata and the use advanced data structures to improve performance, reliability and disk space utilization plus additional extensions such as security access control lists and file system journaling. Everything that has anything to do with a file (file name, creation date, access permissions and even contents) is stored as metadata. This elegant, albeit abstract, approach allowed easy addition of file-system features for fast searches. Although complex to implement, this allows faster access times in most cases. A file system journal is used in order to guarantee the integrity of the file system itself (but not of each individual file). The present implementation for the database is described in detail under the section The Database in Chapter 3.

Chapter 3

Project Implementation

Overview

In this chapter, the implementation of the project is described in detail. The procedures described here where performed simultaneously, as hardware and software design were intrinsically linked and modified constantly, but the project are separated in eight distinct parts: the physical model (hardware); code representation, simulation, agents, database, partitioning, networking and parameter definition.

3.1. The Robotized Experimental Rowing Mechanism "Ro-bot"

For the application, a robotic model, fondly named "Ro-bot", able to produce Ro movements in three degrees of freedom was built and later incorporated into a catamaran style ship. Figure 6 shows the model used in the project.

The robotic arm consists of three actuators disposed in a perpendicular orientation for an easy translation in a three-dimensional coordinate system (see Figure 7). The elevation, horizontal displacement and Ro rotation can be operated individually by sending codes to each actuator [9].

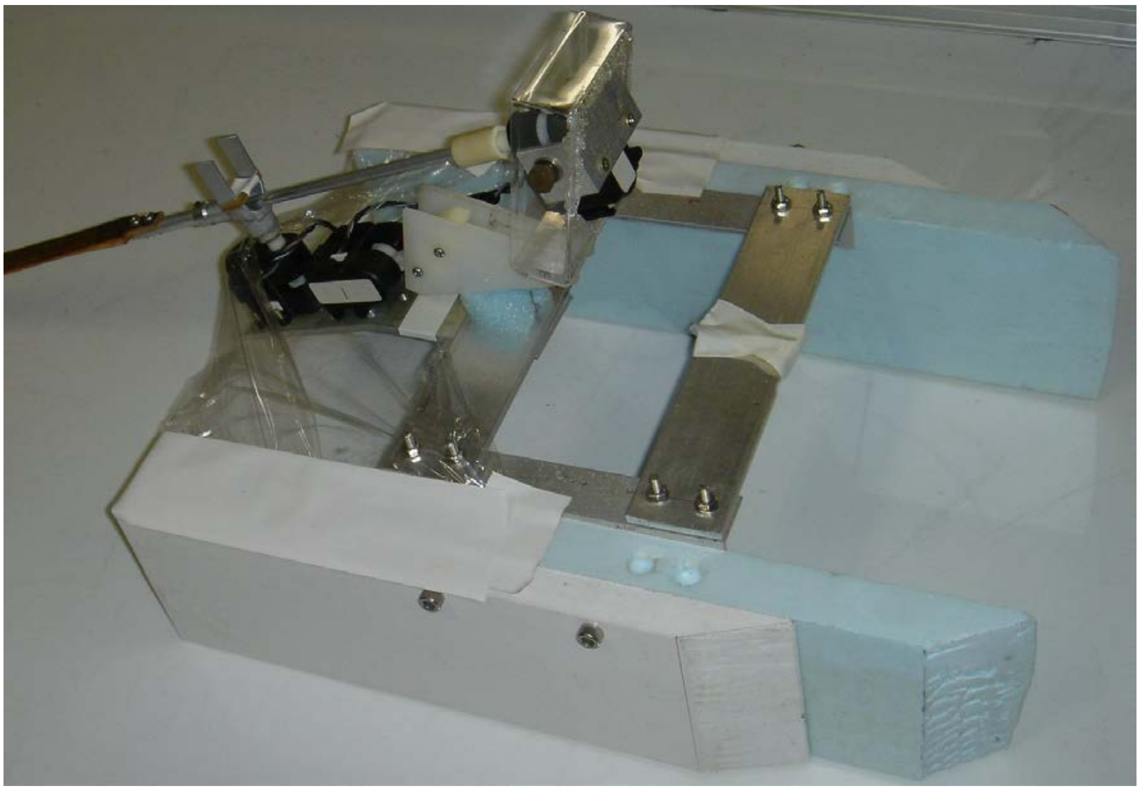


Figure 6 – The robotic rowing mechanism “Ro-bot” mounted in a Catamaran-style ship.

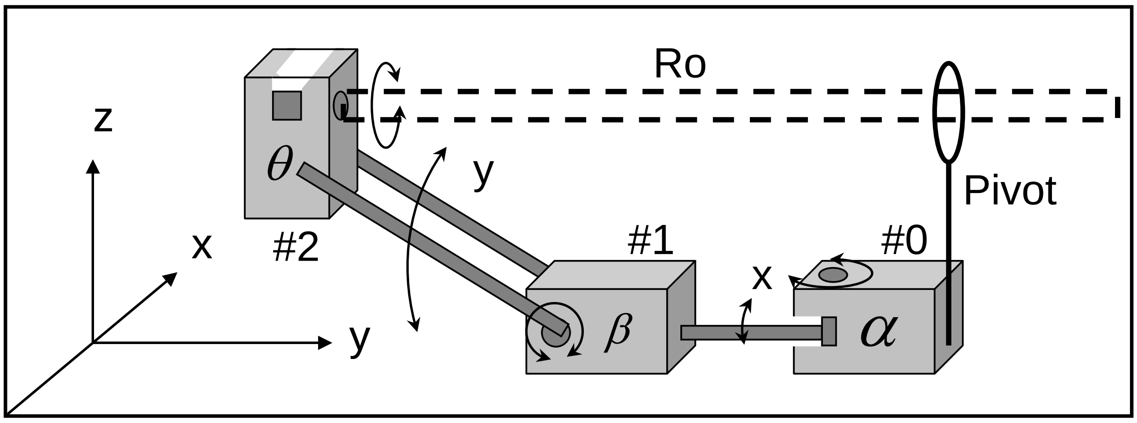


Figure 7 – Schematics of the rowing mechanism with the actual coordinate system adopted in the simulation.

In an early stage, the Ro-bot were mounted over a thrust block, used to measure disturbances caused by operation in the water and calibrate the simulation. In possession of such device and the semantic rules for its operation, the problem was to define suitable sets of control signals to activate it in a timely fashion.

3.2. Control Code Representation

The DNA parameters must be determined for the specific problem in study. The DNA must represent all the variables that affect the system. For the constructed Ro-bot, the actuators (step-motors) operate by receiving a four-byte serial code trough a common serial interface. The code is composed by:

1. Header: always FF;
2. COMMAND: motor address and operation;
3. POSITION: target position;
4. Checksum: calculated by the formula C = (CMD xor POS) and 7F

Each command then follows the structure:

Header (1)	COMMAND (2)	POSITION (3)	Checksum (4)

As an example, the following byte string resets the blade rotation horizontally:

FF	22	7F	5D

Figure 8 express these dependencies and respective ranges graphically.

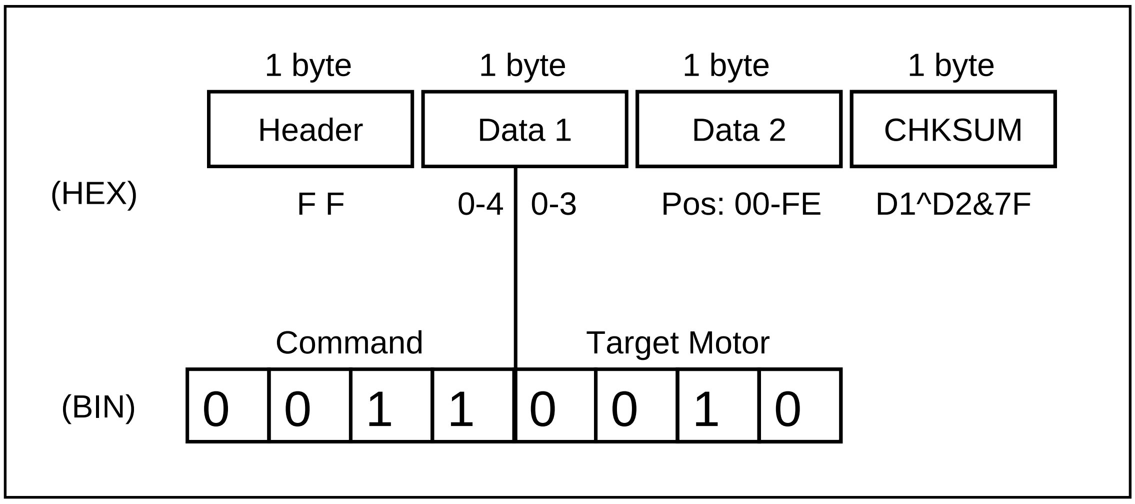


Figure 8 – Control data packet and its respective values.

Once those are already digital data, representation is unnecessary and the codes are directly assigned as DNA parameters. The header, as a constant, is excluded once it is not operated by the GA, simplifying the DNA. The same applies to Checksum, which can be later calculated by the control software. The COMMAND byte contains bits specifying the target actuator address and operation; however its inner workings are irrelevant for the GA, for the positioning command, it can assume the values 20, 21 and 22, referring to the actuator #0, #1 and #2 respectively. POSITION has particular limitations in range, for #0, #1 and #2, characteristic to the Ro-bot construction. This must be considered when operating this byte, once there is a risk of damaging the model when sending improper codes. One extra byte must be added in order to provide timing to the command queue; it has been named DELAY or TIME (short for time to wait). The DNA then consists of a series of sequential commands divided into chromosomes of three bytes, named respectively CMD, POS and DELAY or TIME. The sequence repeats in a loop, generating a unique signal that positions the actuators synchronously. A matrix is one possible representation of the data in the DNA array, making it easier to understand, but representation makes no difference whatsoever for the GA, which always treat the data as a byte array. Figure 9 shows the matrix representation and resulting positioning signals of a code based on the traditional human control.

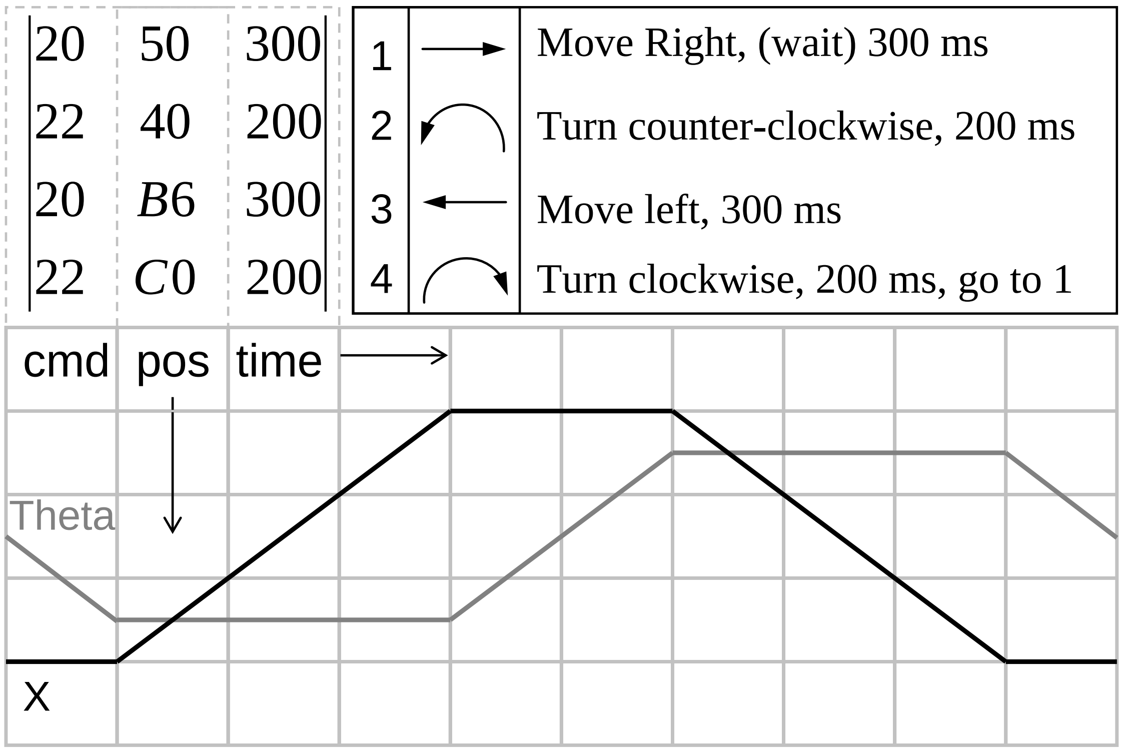

20 50 300
22 40 200
20 B6 300
22 C0 200
1 Move Right, (wait) 300 ms
2 Turn counter-clockwise, 200 ms
3 Move left, 300 ms
4 Turn clockwise, 200 ms, go to 1
cmd
pos
time
Theta
X

Figure 9 – Positioning signals for Ro generated by the execution of the DNA codes represented in the matrix, POS and CMD are represented in hexadecimal values, the time in milliseconds. The chromosomes correspond to the commands listed in the box.

Another representation was needed in order to manage database filenames and to be used in communications. A hash code provides an alternate representation of the DNA array, identifying individuals by short ASCII file-system compatible names. For this purpose, a simple concatenation algorithm was used to generate a unique hash for each individual.

3.3. The Simulation

The need for a consistent simulation of the mechanical model is fundamental for the proposed method to work. The simulation must be fine tuned to match physics and timing of the mechanical model in order to produce useful results that can be used in the model represented. The simulation must be able to evaluate the system for some predefined criteria, returning qualitative numeric values to be used by fitness functions and further classification of the solutions.

A simulation of the Ro-bot arm was implemented to test solutions. The DNA code is passed to a function that simulates the hydrodynamics around the swinging blade, changing the Ro orientation and rotation according to the positioning signals received. The function evaluates the amount of thrust produced by each Ro movement for a specified period. Figure 10 shows the simulation view port. The view port is disabled during GA evaluations to save processing power.

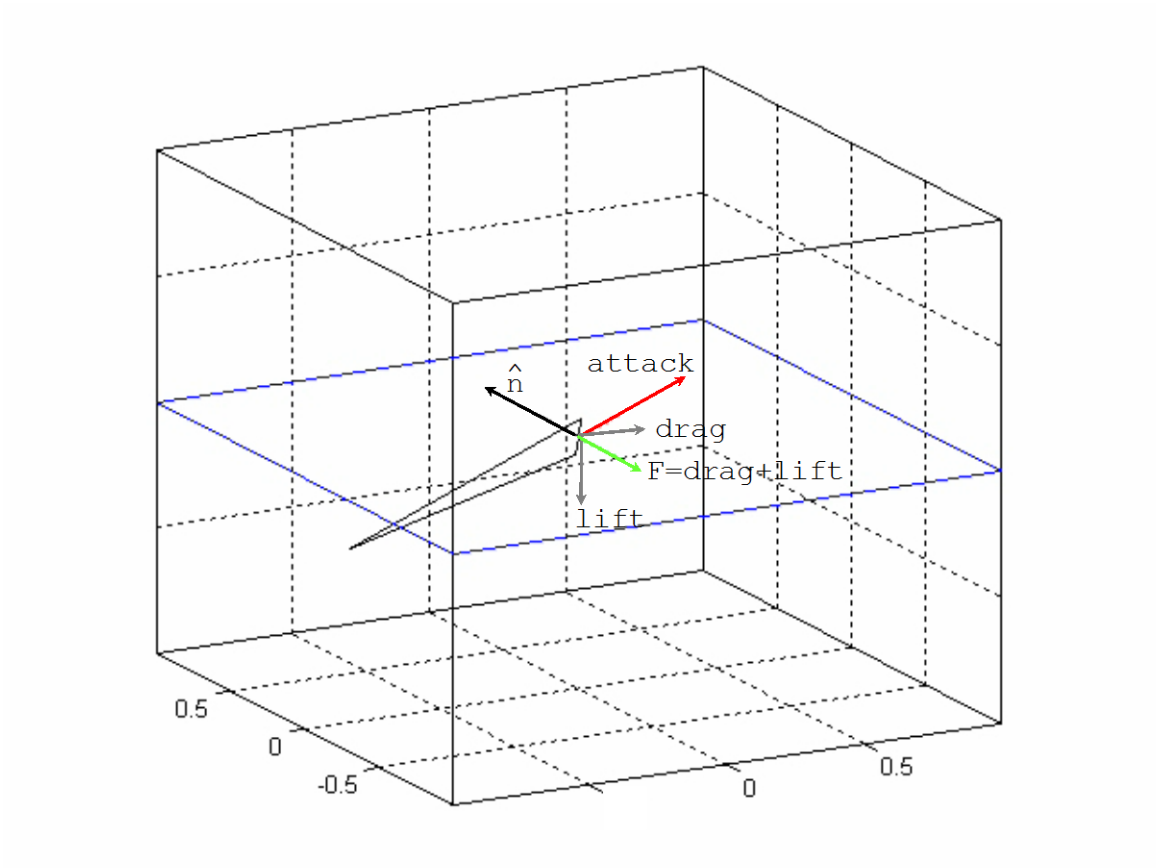


Figure 10 – Wire-frame view of the simulation shows the Ro model and resulting vectors: normal, attack, lift, drag and total impulse; calculated from the instant movement.

Some approximations were made for the simulation in order to speed up the calculations:

- Steady state: fluid speed remains unchanged. As considered during the calibration, the Ro-bot is considered to be fixed at the referential, with restrained movements, limited to the blade;

- Flat blade: The Ro shape was simplified to a flat blade, the wing effects were not considered into the simulation;

- Infinitesimal extrapolation: An infinitesimal surface area of the Ro is used for calculation and extrapolated to the whole surface.

The approximations allow faster calculations for GA experiments and benchmarking, however they introduce discrepancies from the real model. These discrepancies, nevertheless, affect uniformly all the tested solutions, not affecting the overall comparison among solutions for the steady case.

The first step is translating the command codes contained into the DNA to positioning commands for the actuators, using them to indicate actual targets angles for our virtual Ro blade.

Second, we perform simulated movements in time matching those of the actual model. The absolute positions were calibrated experimentally and transformations calculated to match the actual physical model. The obtained translations, from actuator positioning commands to angles in radians (as shown in Figure 7) were matched as follows:

$$\alpha = \frac{POS_0 - 127}{89.141} rads$$
$$\beta = -\frac{(POS_1 - 112) * \pi}{180} rads = -POS_1 * 0.7854 rads$$
$$\theta = \frac{POS_2 - 127}{84.667} rads$$

To position the blade, we then use the rotation transform in three dimensions to shift the virtual Ro in space, applying for each point the three transformation matrixes [14]:

$$\begin{bmatrix} Px_1 \\ Py_1 \\ Pz_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} Px_0 \\ Py_0 \\ Pz_0 \end{bmatrix} \text{Rotation over Y axis (Roll)}$$

$$\begin{bmatrix} Px_2 \\ Py_2 \\ Pz_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{bmatrix} \begin{bmatrix} Px_1 \\ Py_1 \\ Pz_1 \end{bmatrix} \text{Rotation over X axis (Pitch)}$$

$$\begin{bmatrix} Px \\ Py \\ Pz \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Px_2 \\ Py_2 \\ Pz_2 \end{bmatrix} \text{Rotation over the Z axis (Yaw)}$$

Considering then the disposition of the actuators and consequent dependencies for the transformations, to determine the final position for the vertices, we start the rotations with the Ro in the default initial position, as presented in Figure 7. Then we first perform the Y-axis rotation with the θ angle, changing the Ro inclination. Subsequently, the points can be repositioned in space by performing two individual rotations, first over the X-axis, with the β angle and after over the Z-Axis with α. This way, the operations follow the dependencies presented in the actuator assembly, where the hierarchy determines the resulting position for the vertices.

A simplified discrete version of the hydrodynamic force on blade, derived from the Bernoulli principle was used for numeric calculation. Assuming proportional vectors and eliminating constants, we obtain the summation of force for specific simulated time, in a thrust related action:

$$\vec{F} = \left(\vec{L} + \vec{D}\right) = \oint_{\partial\Omega} p\vec{n}d\partial\Omega$$

In the discrete form, the numerical calculation of force is then obtained by the sum over time of the resulting force for a specified simulated time.

$$\vec{F}t = \left(\vec{L}t + \vec{D}t\right) = \sum_{T}\sum_{\partial\Omega} p\vec{n}$$

The function returns a three dimensional vector containing the strength and direction of the average total force resulting from the full operation over the simulated time:

$$\vec{F}t = \sum_{T}\left(F_X, F_Y, F_Z\right) = \left(X, Y, Z\right)$$

The fitness is calculated by the dot product (projection), using a unitary vector in the evaluated direction:

$$Fitness = \sum_{T}\vec{F}\cdot\hat{d} = \vec{F}t\cdot\hat{d}$$

Note that the simulation returns a unique vector for each point in space associated to the received DNA. Considering that the simulated period is not changed, these values can be reused even if the fitness function is changed into the agent. The fitness is a scalar value, and changes according to the selection criteria.

Calibrating the simulation to match the real physics in the model is the most important, as well as the trickiest part of the experiment. The simulation failure to match operating aspects may cause unexpected behaviors outside the simulation and even render the solution unusable. For this reason, several experiments were conducted to evaluate actuators in operation conditions. Figure 11 shows the sensor configuration assembled to measure the actual model response and attributes.

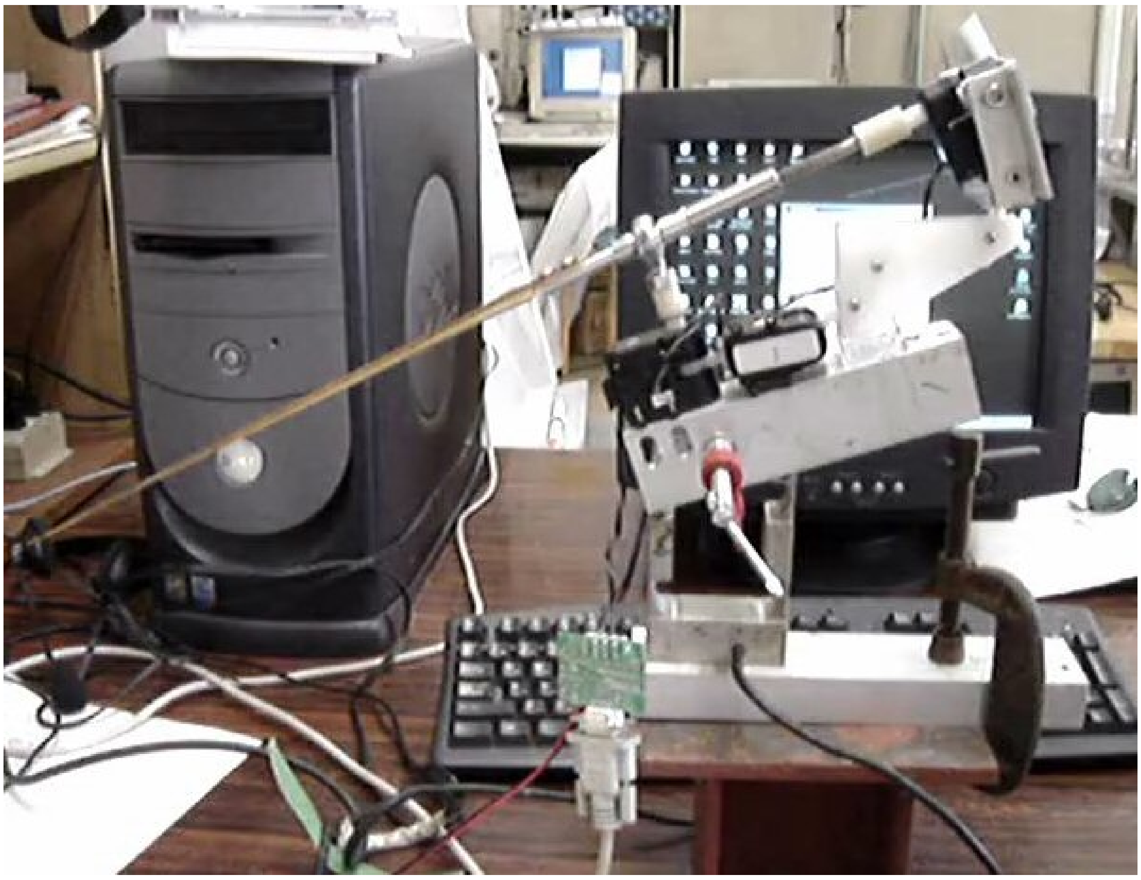


Figure 11 – Sensor assembly used to calibrate the simulation. The Ro-bot mechanism was first mounted over a thrust block, where the impulse produced in the Y-direction could be recorded.

Sensors were used to determine the amount of impulse generated by operating Ro inside a water basin. The measures were later compared to the data generated by the simulation, to verify consistency. Figure 12 shows the comparison between measured and simulated impulses.

After calibration, the simulation obtained frequency and amplitudes matching the real model sensory data, differing only from a scale factor, result of the approximations introduced in the simulation.

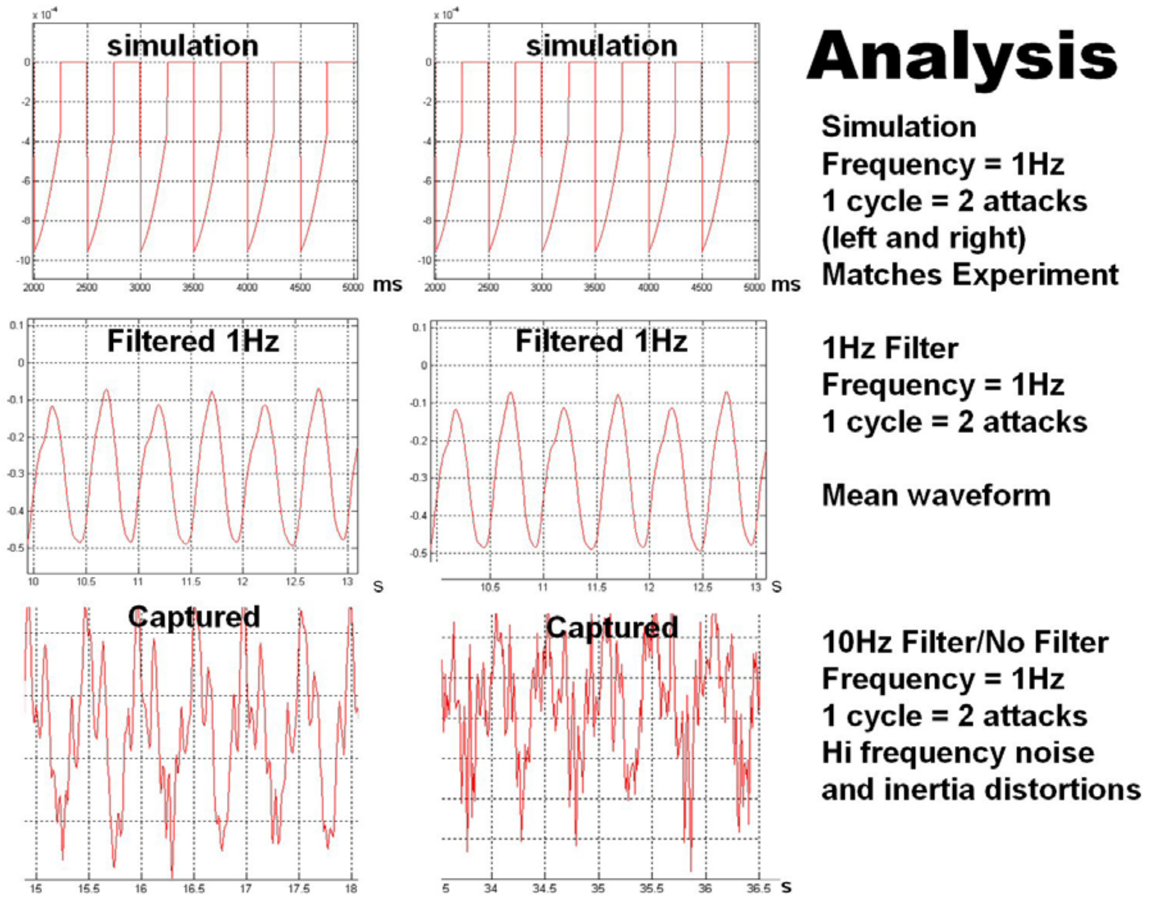


Figure 12 – Data obtained from sensors, filtered to remove the noise and the data generated by the simulation compared, the thrust generated forward (Y direction) matches in frequency and scale the simulated model.

3.4. Agents

The agents in this approach are instances of one same program. They execute simultaneously, both within the system and through the limits of the network in case of distributed computing topologies. They spread uniformly among the available processors locally and in all MAS enabled systems [11]. Figure 13 summarizes the main groups and respective modules (representing methods) involved in each single agent operation, as well as the connections between these modules.

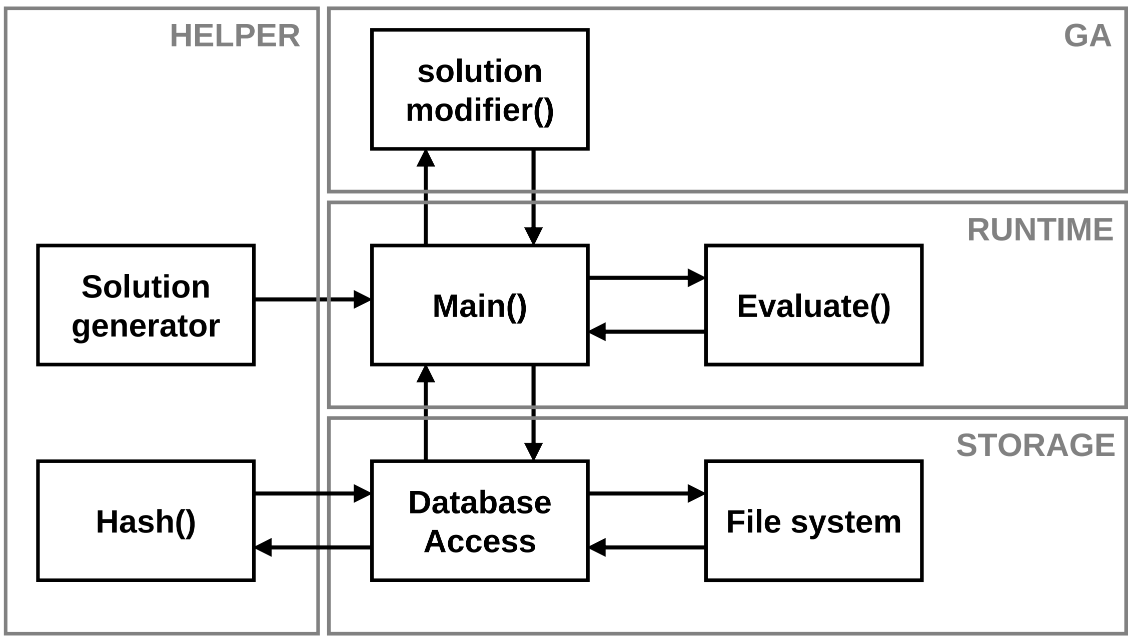


Figure 13 – Single agent diagram containing main groups of functions, identifying its respective links.

At first, a set of solutions is generated by the solution generator method that has the set of rules to create valid operating solutions randomly within allowed boundaries. Then the agent is locked into an infinite loop, which performs the evaluation for every solution, rank and finally performs the genetics according to provided rules, storing the progress in the FS Database.

Setting the local population to 1000 individuals (parameter selected for the current case), the following pseudo-code represents the operations performed by one single agent (the Matlab codes are available for reference in the Appendix):

```
1.  Number of solutions n=1000
2.  Randomly create    n solutions from Global_Template with global_variance
3.  Update Global_Template to for partition
4.  Maximum Fitness MAX=0
5.  Loop:
    For all n solutions (highly parallelizable)
    Consistency check
    If evaluation is in the Global_database,
            retrieve evaluation(n),
            else evaluation=evaluate(n)
    Fitness(n)=evaluation•direction (scalar)
    If Fitness>MAX,
            MAX=Fitness(n),
            Best_solution=solution
            Save solution to best_database
    Sort solutions by Fitness(n)
    Best 5%: time to live
    Worst 10%, kill
    50 amongst the best 25%, crossover by pairs
    Positions 100- 900, operate mutation
    Communicate progress to server
    Receive server directives
6.  Repeat loop
```

A preliminary consistency check before evaluation, avoids waste of simulation time with inconsistent solutions (invalid commands and POS out of range). A quick lookup on the database also checks if the evaluation was already performed for one point, avoiding redundant calls to the fitness function.

Finally, an agent may use an arbitrary number of operators to try in increasing its solutions fitness. These operators are not restricted to GA. The implementation of additional operators can benefit the convergence speed. Methods may vary depending on each programmer preference, ability and particular expertise. Some examples arise from, e.g., Neural Nets, Simulated Annealing and gradient climbing. For the current experiment, only Mutation and Crossover were used.

The number of agents and local populations must be scaled according to available computational resources. Each agent can be given an arbitrary number of individuals, a large population can make the calculation slow, and a small population may not produce significant improvements per generation. Selecting optimal values for agent numbers and respective populations comes only from experience. Granularity problems may arise from bad choices. To the actual experiment topology, a dual-core computer (server) connected to 4 single-core computers (clients), 2 to 4 agents per core with a 1000 local population produced satisfactory results. The analysis used to define the optimal numbers is later presented in Chapter 6 (see Figure 36).

The simulation (Evaluate function) is the most time-consuming task and the most requested method by every agent, in parallel computing terms, the bottleneck. Several ways to reduce calls to this function were implemented, such preliminary tests, lookups in the local and server database and communications to check if another agent is performing a search in the neighborhood. By this context, agent communications have not the purpose share results in order to improve individual populations, but to avoid redundant searches in the same region of the space. Once populations often revisit recent evaluated points, is important to keep track of recent points. This is done by adding some specific fields in the DNA. Another way to reduce the call to the fitness function is by implementing a solution database. The database requires a large amount of memory, and is not suitable for all problems. The use of a compressed hash code is a good method of saving storage memory.

3.5. The Database

One of the new proposals is to avoid redundant evaluations by implementing a database for tested individuals. In addition to each agent to keep on track of the recently performed evaluations, it stores the tested values into a database, associating each result to a unique hash. Once one GA randomly falls back in a tested point, it uses the stored value instead of calling the evaluation function. This method is limited by memory to the number of solutions currently stored by the agent, and by disk-storage capacity to the resolution of scanned space. It also requires fast search capabilities, recent 64-bit computing, together with improvements in the file-system finally allow the use of this resource to save processing time.

As alternative to the use of a database-engines, such as SQL or JDBC, which would increase the overall resource load, those values were stored straight into the file-system, under a special folder hierarchy, using the file-system itself to provide fast search capabilities. Making the database folder accessible trough the network, all agents, local or remote, can access its contents.

A unique hash is obtained by concatenating the hexadecimal DNA array to be used as filename. The evaluation return values are the only data recorded into it. Hash uniqueness is required, compression and hash algorithms also can be used for this purpose. To accelerate the search, the folder structure is determined by the first six values in the DNA (being never smaller than nine bytes). The gene database also keeps record of the best solutions found, with date and time, allowing a history analysis of the progress. The server organization is structured as follows:

```
DNA: 20 50 03 22 40 02 20 B6 03 22 C0 02

HASH: (0)3-50-(2)0+(0)2-40-(2)2+(0)3-B6-(2)0+2+C0+2=350024023B602C02

FILE: //server/tests/20/50/03/22/40/02/350024023B602C02.dat

Best solutions: //server/best/350024023B602C02.dat

Global variables: //server/data.db
```

All agent information and current progress is stored at the server once in every generation, not allowing data to be lost by crashes on server or client sides.

This approach creates an intricate folder structure and a large number of files. Disks are usually required to be formatted after being used for this purpose. For this reason alone a dedicated hard disk is recommended. LAN drives are cheap and adequate for the task, the disk-accesses are of an order of milliseconds (for new hard disks and file-systems), in comparison to a seconds-long time required to perform one simulation, it represents a time saving.

3.6. Multi-variable Space Partitioning

The task of distributing the load among computers starts with the decision of how to partition the problem space. The space can be divided in several ways, by different topologies, depending on the number of dimensions, treatment given, nature of operations, etc. To determine the best partitioning method, knowledge about the volume being partitioned is essential.

For the current problem, we can first analyze the structure of the present multi-parameter space by determining the possible values each member of the DNA can assume. Using for this purpose the matrix representation of the values, we can explode the possibilities for each line as follows:

CMD=20, 21, 22 (3)	POS=0-FF (255)	T=0-1s (10)

The maximum number of lines is 14, limited by the algorithm. In this case, we have an exponential explosion of the form:

$$possibilities = 3 * 255 * 10E14 = 4.02E46$$

The exponential explosion leads to a number never bigger than 4.02E46, the number of maximum possible solutions.

The available agents have to spread its population trough a part of this space, defined by the assigned partition, expressed by a central point and a variance, defining a uniform cubic distribution. Then the agent scatters randomly its initial population in a uniform distribution according to those parameters.

Once the GA cannot distinguish between the variables in the DNA, the easiest way to partition this space is by the initial variables in the DNA. In this case there are 3 main partitions in the first dimension. The second and third dimensions are divided in sections according to the estimated maximum number of agents available for the MAS. Additional parameters in the DNA can be used for the partitioning, but it may complicate partition visualization. This was the current solution adopted.

A better solution would be to link the partitioning to the way agents search the space, specifying segments in every dimension, i.e., divide every parameter in the DNA by equal segments. It would produce a large number of partitions and a highly localized agent population, consequently increasing the resolution of the search. But this high granularity solution would require additional processing power, not available in the circumstance.

3.7. Spreading Agents Trough Networked Computers

The agents executing in the client computers are no more than the same version of the server-side agents with modified global variables so they can target the server's address on the network. The server holds the global resources, environment variables and the solution database, necessary to clients to perform communications, get directions for the search and inform of the current progress. Figure 14 presents the actual architecture for network distributed processing.

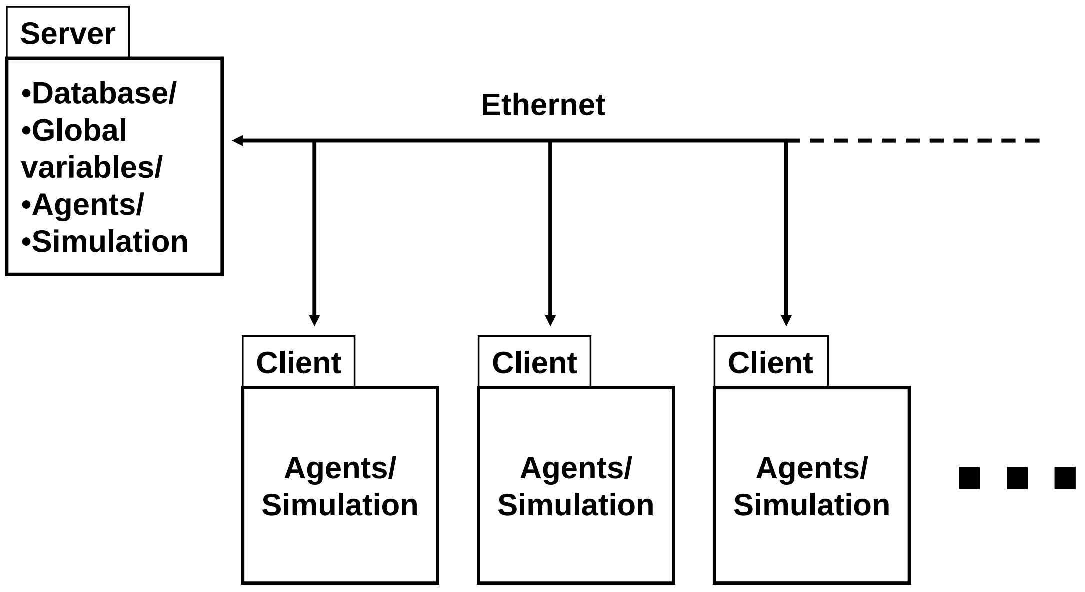


Figure 14 – Diagram of the distributed structure for the MAS. The server stores tested solutions and global variables while clients perform the computations.

To profit on the networked computers idle power, we set the agent launching application as screensaver in all clients. Once the idle time is elapsed, the application is started, initiating a pre-specified number of agents as specified, determined according to the local system resources [6].

The started agents get a random partition of the space to search on, defined by a point in space and a specific variance for each dimension. This localizes the search at the start, but individuals eventually travel to neighboring partitions over time. For trespassing individuals, they are not stopped unless they hit a point where evaluations were already performed by another agent. This condition is implemented in the database. If there is an evaluation for that particular point in the current database performed by another agent, the current individual is eliminated, avoiding redundant evaluations.

The communication relies heavily on the operational system, once all the agents have the same code and communicate trough global variables localized in the server, special care must be taken to avoid simultaneous accesses to the same file. Signaling and queuing procedures have been adopted to ensure that only one agent has access to one file at the time.

In cases where some computers are fully dedicated to the search effort, increasing the number of agents for a high granularity is recommended. Higher granularities will swarm the populations over the multi-dimensional space more efficiently, however, will cause the system to run unbearably slow.

3.8. Selected Parameters for MAS and GA

For the actual experiment, the following best working parameters were obtained:

Number of agents:

- Usually NP+1 (Number of Processors in the system plus one) to 5NP (five times the Number of Processors), depending on each computer performance;
- One single agent for shared systems, systems running critical processes and elder computers. Population per agent: 1000;

Population variance (space partitioning):

- 3 partitions in the first parameters (20, 21, 22);

- 20% for each following parameter (5 partitions per dimension);
- Total partitions = 3x5x5 = 75 partitions.
- Task priority was set to highest on the server and to idle on client computers.

Mutation rate:

- 90% probability for factor 0-1%;
- 9% probability for factor 1-10%;
- 1% probability for factor 10-100%.

Crossover:

- Performed for 25 random pairs among the 100 best, producing 2 children per couple.

Artificial selection rules every generation:

- Elite (top 5 highest ranked individuals) receive a time-to-live of 5 generations;
- Discard worst 100.

Stop rules:

- Stop if maximum fitness is unchanged for 3k generations;
- Stop if no new points are tested for 1k generations;
- After stop, randomize population in next partition and restart.

The actual parameters where achieved by successive adjustment. Different networks and problems may present better results with different settings. Theoretical extrapolations can be used to guess starting points, but most theories do not consider concurrent populations.

Chapter 4

Simulation Results

Overview

In this chapter, some of the results obtained in the simulations are introduced. The traditional mode, as well as other well know modes, are also presented for future comparison. Matrixes and diagrams are used to explain the mechanics of each particular rowing, as introduced in section 4.1. The mechanics of the new found modes in two and three degrees of freedom are presented in the proposed form.

4.1. Simulating Well Known Modes

As hard as it is to represent time-dependent events by still images, the selected representation consists of a top view with a fix reference (simulation scenario, with Ro-bot attached to the referential) and the blade can be seen swinging in the surface from above. The diagram shows Ro represented by a single slice (see Figure 15) in the air-water interface, used to show positions the blade assume in time (gray slices), black slices denotes points where commands are issued and are accompanied by arrows representing operations. The matrix representation and hash code are also presented, and can be used to visualize the rowing in three dimensions using the simulation's visualization engine or to control the model using the control the application [20].

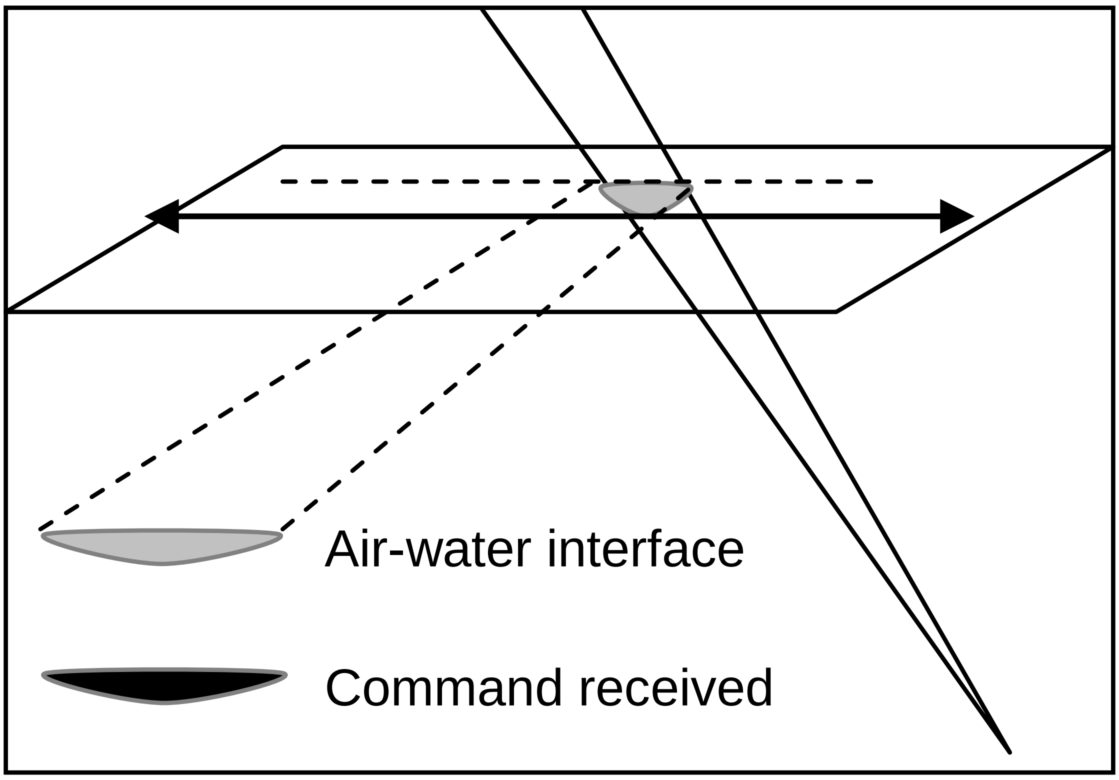


Figure 15 – Representation of a swinging blade by a still image, the blade assumes the positions marked by gray slices as it moves in the directions pointed by arrows, black slices marks positions where commands are issued.

4.1.1. Specification and Simulation of the Traditional Swing

We first analyzed the “classic” Ro swing reproduced as observed in the traditional humanistic control; it will be called C-2DF from here. The control codes were presented previously in Figure 9, in hexadecimal values and milliseconds, from now all the matrix values will be presented in pure decimal numbers: Figure 16 shows a box containing the commands represented by the chromosomes in the matrix, and a diagram of C-2DF rowing mode:

$$\text{C-2DF}=\begin{bmatrix} 32 & 80 & 300 \\ 34 & 64 & 200 \\ 32 & 182 & 300 \\ 34 & 192 & 200 \end{bmatrix} \qquad \text{Hash 350024023B602C02}$$

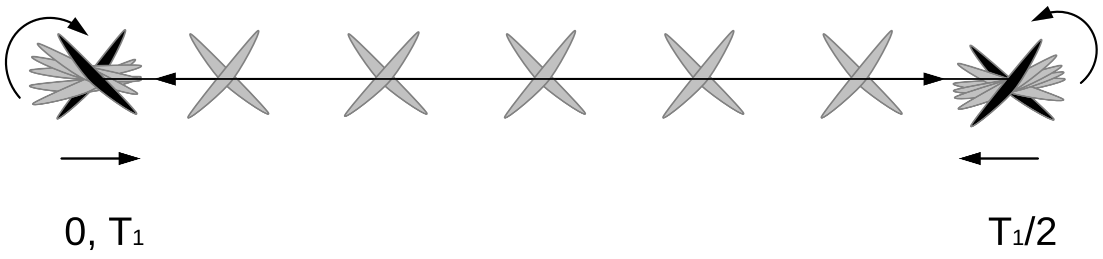


Figure 16 – Classic rowing in the proposed bi-dimensional graphic representation, according to the command sequence shown above in decimal values.

Observing the instant impulse on time produced in the evaluated direction we can better understand the implication of the fitness, the area under the curve for F vs. time. Figure 17 shows the plot of instant impulse vs. time.

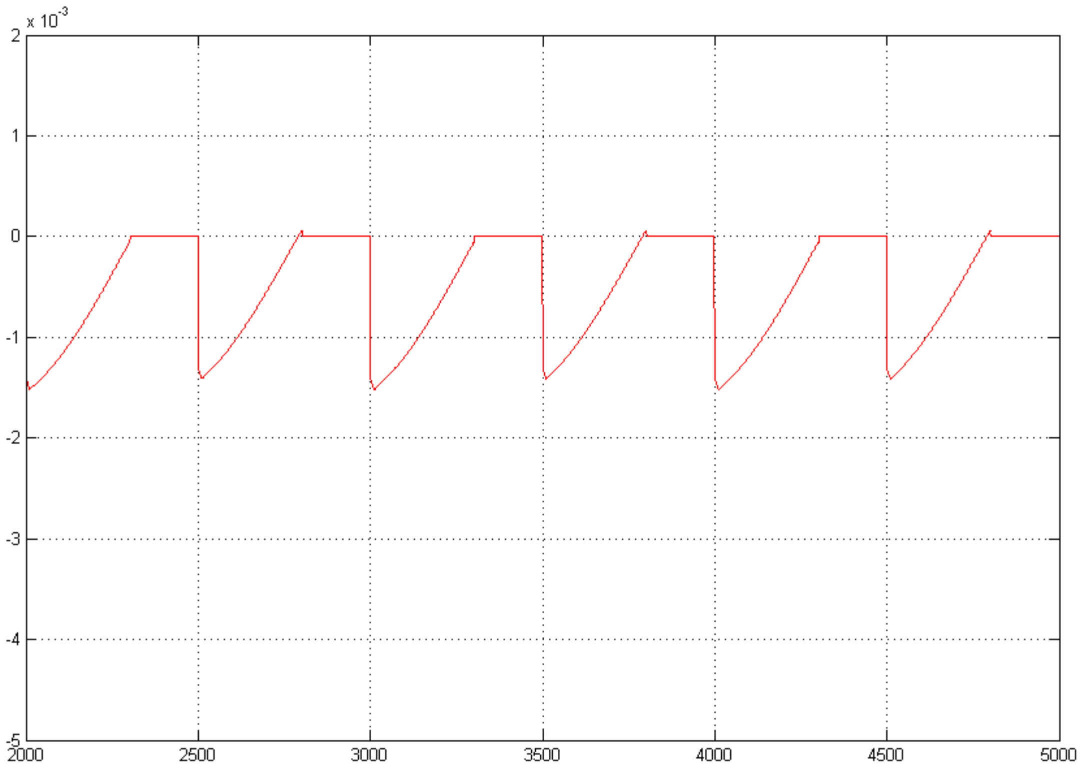


Figure 17 – Plot for the instant propulsion over time for C-2DF shows negative values as thrust direction is behind (-Y), the total fitness to be the integral over the area between zero and the curve.

4.1.2. Simulation of the known mode "Rotated Blade"

Another solution, known as the Rotated Blade (referred here as RB-2DF, see Figure 18), was manually feed into the program. The aim was look for an optimized version for this mode. For the simulation, a flat blade was considered. In the real Ro model a wing effect exists in only one direction, causing asymmetry in the resulting thrust and generating side-effects (stall). Any tentative to optimize this mode failed, leading to some of the other modes discovered by MAS, indicating that this mode is less efficient than other newly found swings.

$$\text{RB-2DF} = \begin{bmatrix} 33 & 140 & 0 \\ 32 & 80 & 300 \\ 34 & 64 & 200 \\ 32 & 182 & 300 \\ 34 & 192 & 200 \end{bmatrix} \quad \text{Hash 08C1350024023B602C02}$$

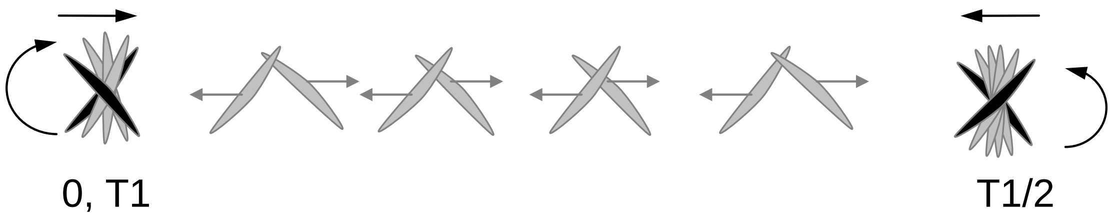


Figure 18 – Rotated blade (RB-2DF) swing mode, specified manually.

4.2. Optimal Rowing Mode Found in Two-degrees of Freedom

Ignoring the Ro-bot ability to perform movements in three degrees of freedom (3DF), at first we limited the movement in the MAS to two degrees of freedom (2DF) by disabling access to one of the actuators, in order to check if the system finds a similar answer to C-2DF.

4.2.1. GA Optimized Two-Degrees of Freedom rowing mode

The result was found in 9 hours, the MAS achieved a similar solution to C-2DF, but with altered response times and increased frequency. The increased frequency led consequently to a higher thrust over time. This mode will be referred as Optimized for 2DF or O-2DF. Figure 19 schematizes the mode.

$$\text{O-2DF}=\begin{bmatrix} 32 & 80 & 200 \\ 32 & 179 & 100 \\ 32 & 182 & 0 \\ 34 & 224 & 300 \\ 32 & 95 & 0 \\ 32 & 94 & 100 \\ 34 & 40 & 100 \\ 34 & 39 & 0 \end{bmatrix} \qquad \text{Hash 25001B300B603E0205F015E012820272}$$

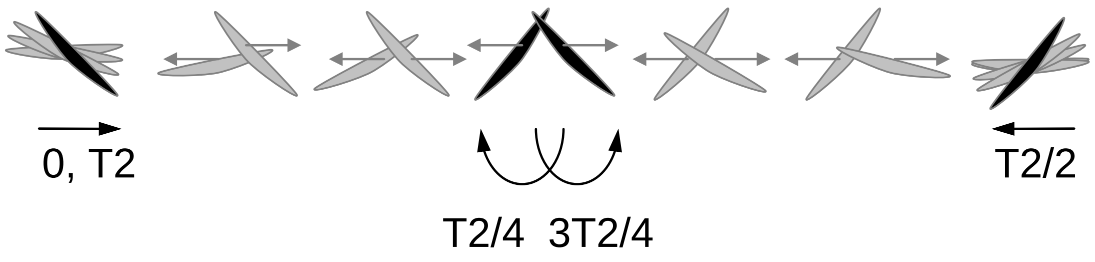


Figure 19 – GA optimized 2DF rowing (O-2DF).

Analyzing the O-2DF it can be observed that the rowing mode gains its frequency by reducing the time required to turn Ro in the corners. Instead, this mode turns Ro gradually just before in the region where less thrust is generated. More than provide a better performance, this rowing mode requires about the same energy for thrust as the traditional rowing.

The plot of instant impulse vs. time for this mode shows an increased area of thrust under the graph, as noticed in Figure 20.

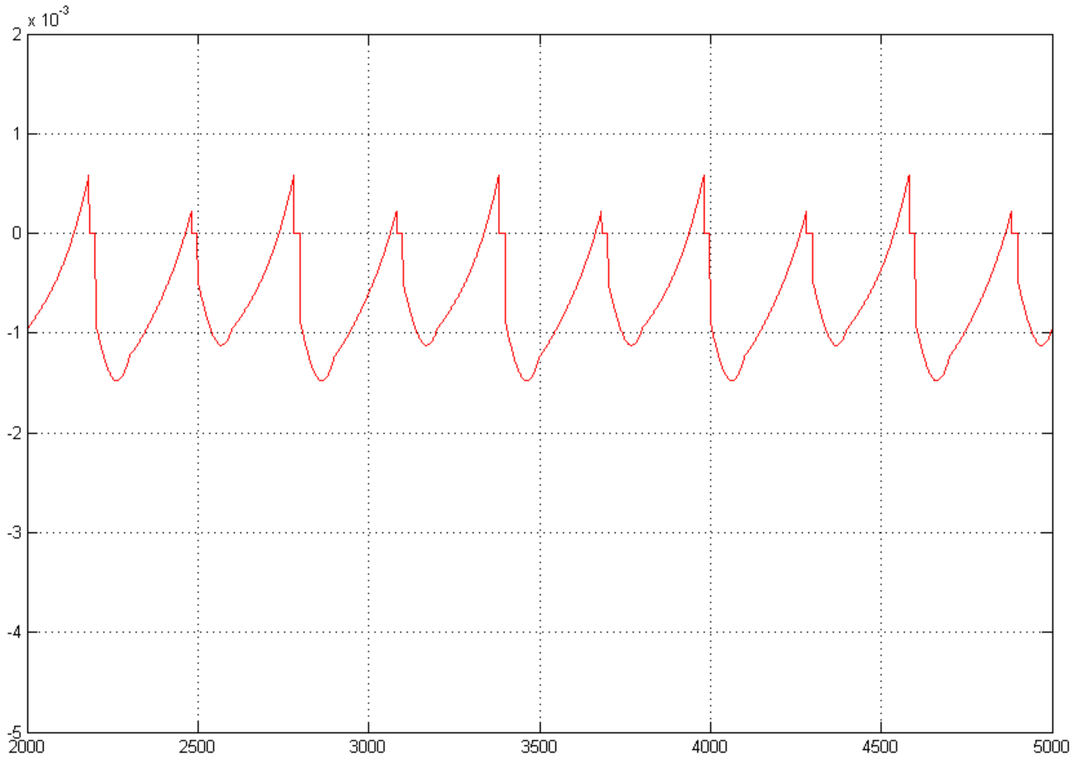


Figure 20 – Plot for the instant propulsion over time for O-2DF shows the total fitness to be result in a larger integral area between zero and the curve than O-2DF.

4.3. New Rowing Modes Found in Three-degrees of Freedom

The next step was to extend the movements to 3DF, allowing the use all actuators. Several new possibilities for operating Ro in this condition were found. Here we describe the most efficient and interesting among them, named respectively: The X-swing (X-3DF), the U-swing (U-3DF) and the M- swing (M-3DF).

4.3.1. The X-Swing rowing

$$\text{X-3DF}=\begin{bmatrix} 34 & 200 & 200 \\ 33 & 100 & 200 \\ 32 & 56 & 0 \\ 33 & 140 & 200 \\ 34 & 55 & 200 \\ 33 & 100 & 200 \\ 32 & 200 & 0 \\ 33 & 140 & 200 \end{bmatrix}$$

Hash 2C822641038028C1237226410C8028C1

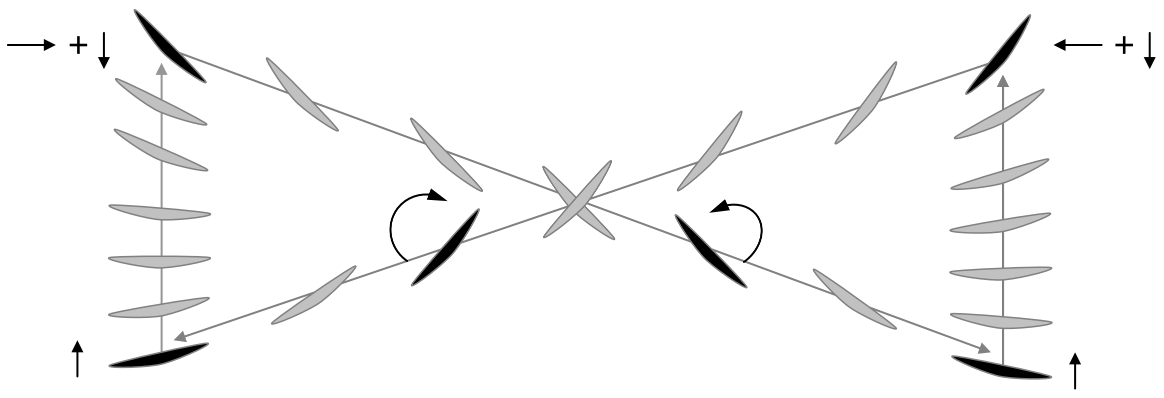


Figure 21 – Diagram for the X-3DF swing, the latest discovered and more efficient 3DF mode.

The most efficient new 3DF rowing found was named X swing, for the moving that resembles an X shape. This move combines lift and drag to increase overall output of thrust. Figure 21 has a diagram representing the X-3DF mode.

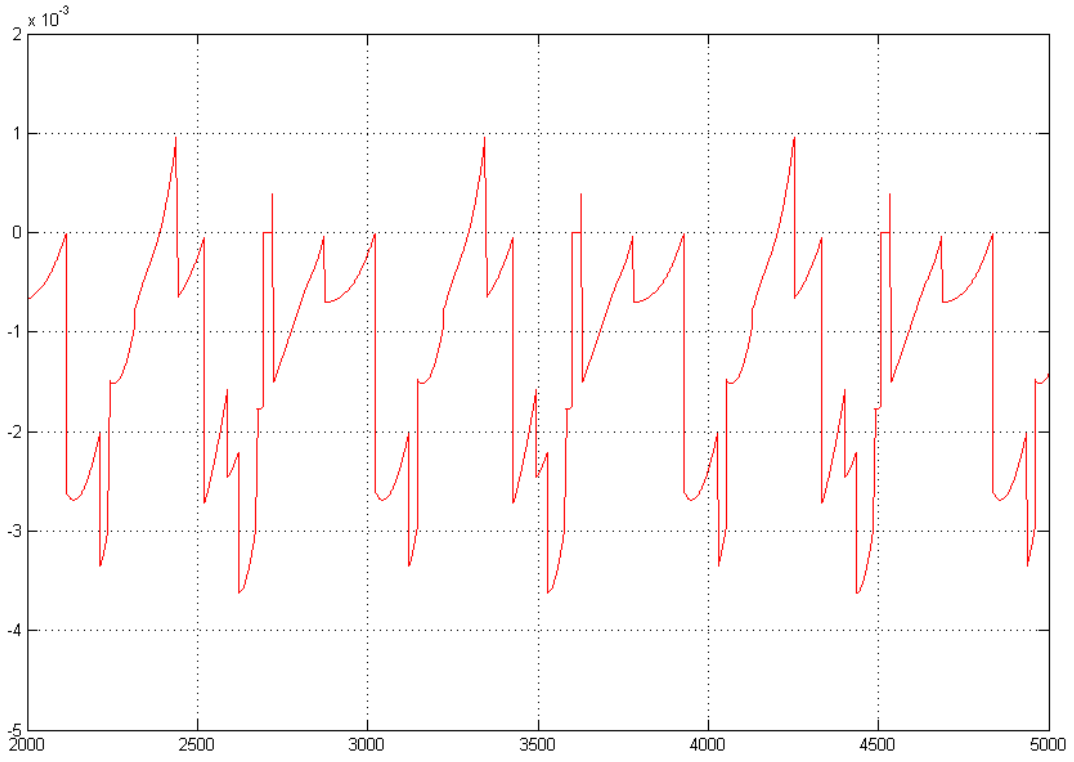


Figure 22 – Plot of instant impulse vs. time for this mode.

The modes in three degrees of freedom produces more work as consume more energy. The impulse graph presented in Figure 22 gives an idea of the generated thrust.

4.3.2. The M-Swing rowing

$$DNA=\begin{bmatrix} 32 & 80 & 0 \\ 33 & 100 & 100 \\ 33 & 127 & 100 \\ 34 & 64 & 200 \\ 32 & 182 & 0 \\ 33 & 100 & 100 \\ 33 & 127 & 100 \\ 34 & 192 & 200 \end{bmatrix}$$

Hash 0500164117F124020B60164117F12C02

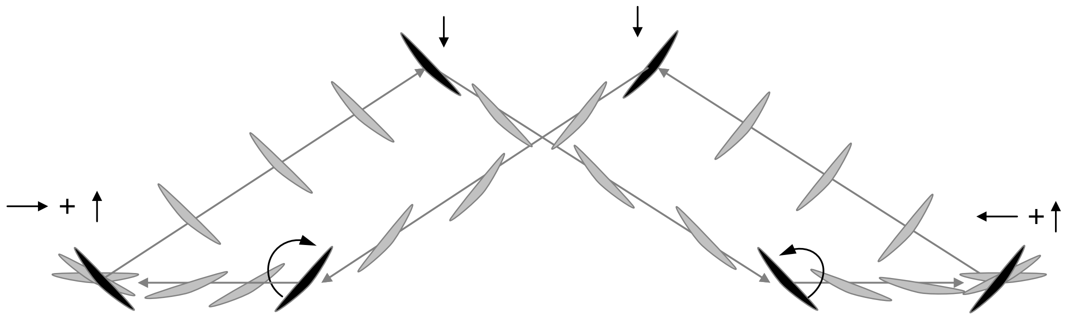


Figure 23 – Diagram of the M-3DF rowing, the first 3DF mode discovered.

The first 3D mode found, having inferior thrust compared to X-3DF and U-3DF, was named M swing for a similar a reason as in X-3DF. Both require much higher energy levels to perform the rowing operation when compared to 2DF modes. Figure 23 schematizes the M-3DF swing. Figure 24 shows the instant thrust vs. time.

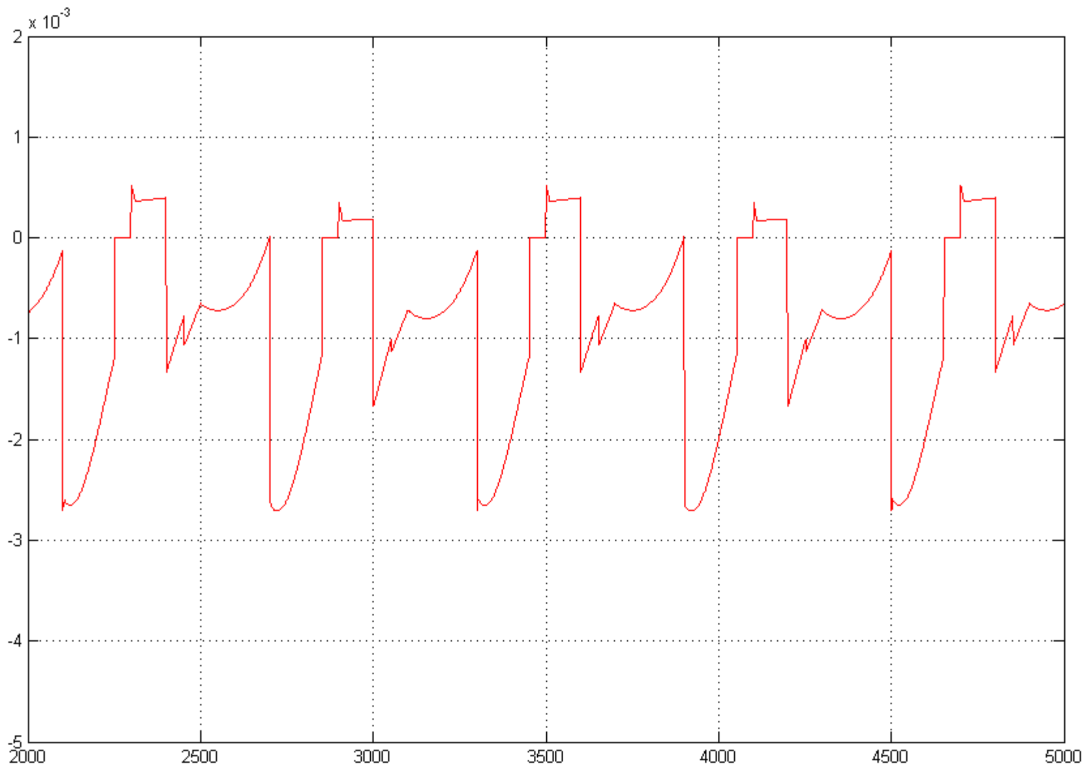


Figure 24 – Plot of instant impulse vs. time for this mode.

4.3.3. The U-Swing rowing

$$\text{DNA}=\begin{bmatrix} 32 & 182 & 100 \\ 34 & 255 & 200 \\ 33 & 100 & 200 \\ 33 & 140 & 100 \\ 32 & 80 & 100 \\ 34 & 7 & 200 \\ 33 & 100 & 200 \\ 33 & 140 & 100 \end{bmatrix}$$

Hash 1B602FF2264118C115002072264118C1

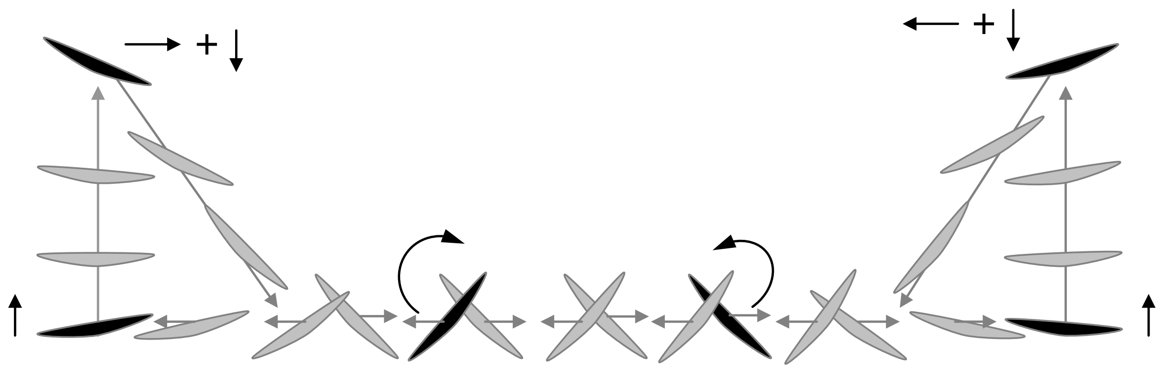


Figure 25 – Diagram of the U-3DF rowing, the second found 3DF rowing.

The third newly found 3DF rowing for three degrees of freedom. This rowing presents higher performance than the M-3DF and less performance than the X-3DF. Figure 26 shows the instant impulse vs. time.

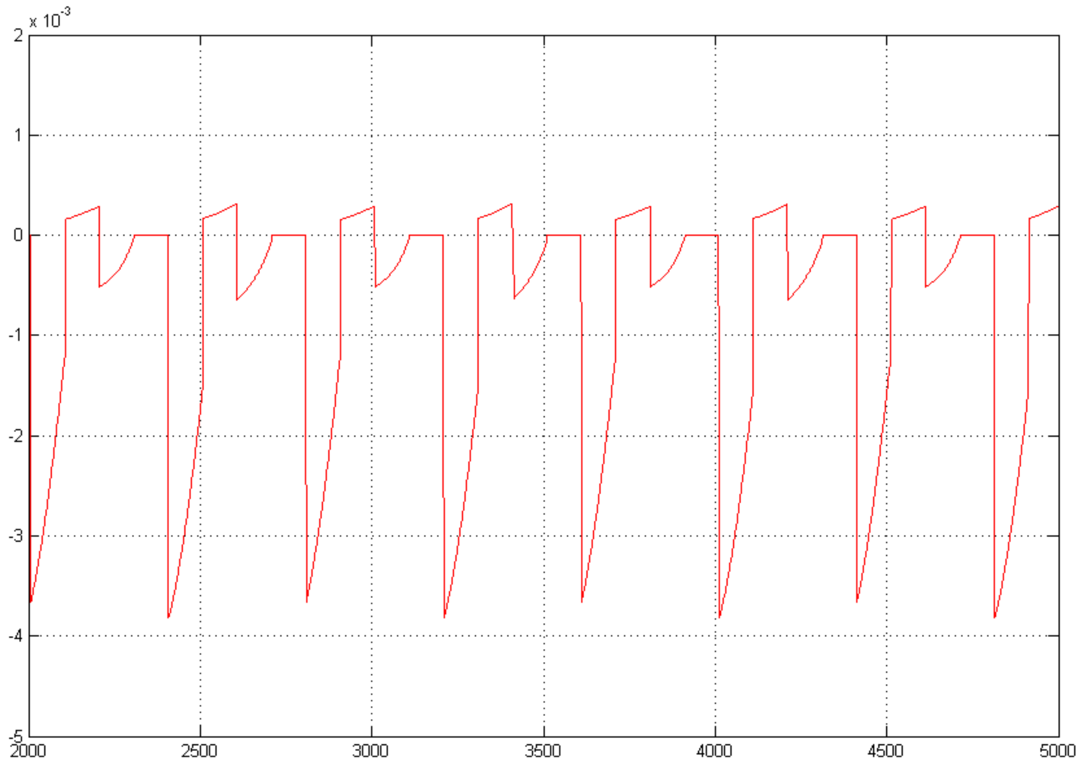


Figure 26 – Plot of instant impulse vs. time for this mode.

4.4. Multi-Directional Modes

For maneuvering purposes, additional modes were evaluated by changing the fitness function to left and right turn and backward rowing. Those were used mostly for positioning the model in experiments, as well as to prove the capability of the MAS to find solutions by different criteria only by changing the fitness function. The following list summarizes the DNA codes in decimal matrix and hash form for turning and maneuvering backwards:

$$\text{Backward}=\begin{bmatrix} 34 & 192 & 200 \\ 32 & 182 & 300 \\ 34 & 64 & 200 \\ 32 & 80 & 300 \end{bmatrix} \quad \text{Hash 2C023B6024023500}$$

$$\text{Right turn}=\begin{bmatrix} 33 & 80 & 100 \\ 32 & 80 & 200 \\ 34 & 1 & 200 \\ 32 & 182 & 200 \\ 34 & 140 & 200 \end{bmatrix} \quad \text{Hash 1501250020122B6028C2}$$

$$\text{Left turn}=\begin{bmatrix} 33 & 80 & 100 \\ 32 & 80 & 200 \\ 34 & 127 & 200 \\ 32 & 182 & 200 \\ 34 & 254 & 200 \end{bmatrix} \quad \text{Hash 1501250027F22B602FE2}$$

Chapter 5

Experimental Verification

Overview

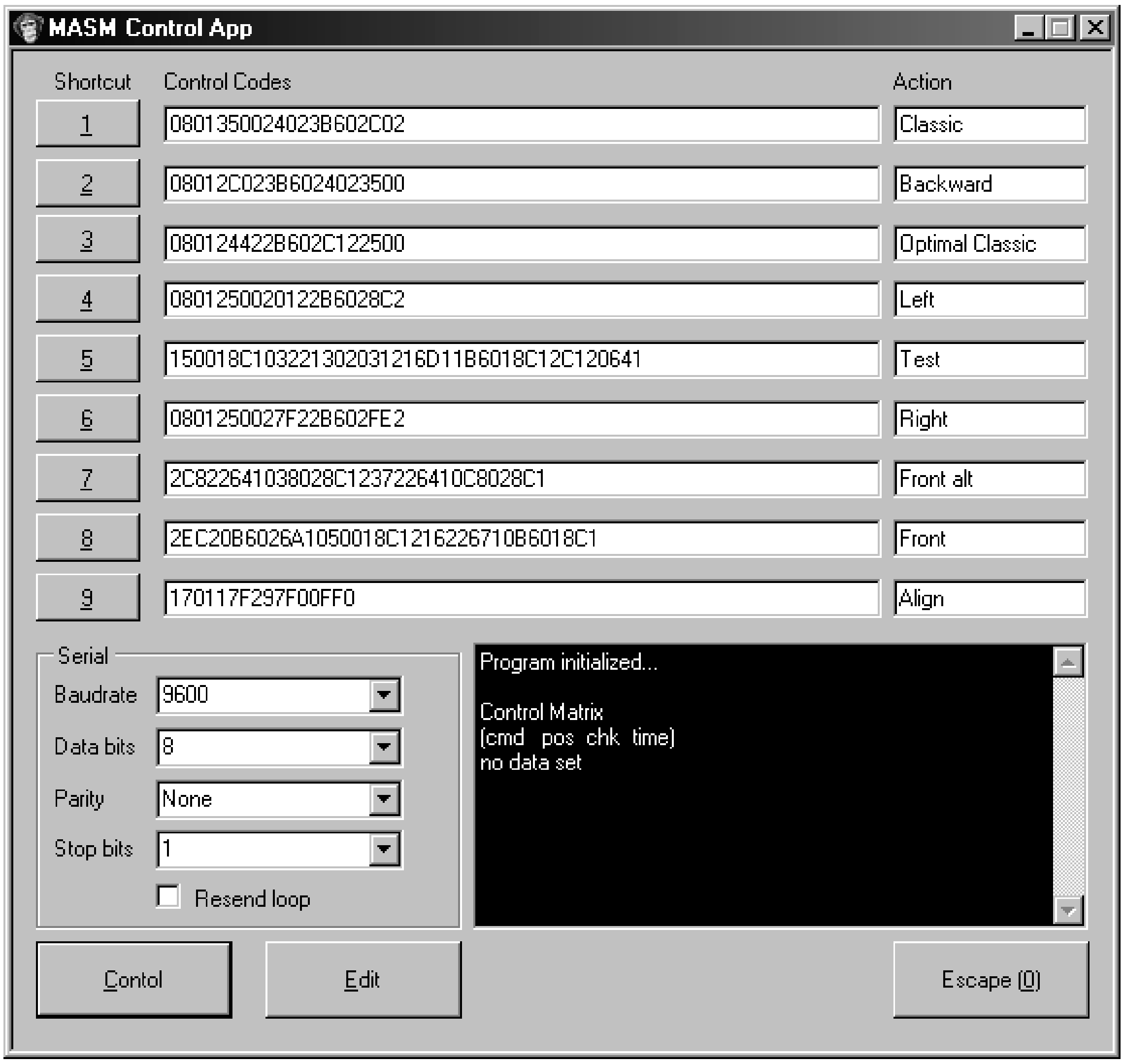
In this chapter experimental results are obtained by testing the codes autonomously found by the MAS in the actual physical model. The experiments were performed in a towing tank for a limited length in two different conditions. Each rowing had its time taken for a three meters course several times, under different circumstances.

5.1. Experiment Description

For the experiment, the codes are entered into a serial communications program, especially developed for sending the control codes to the actuators trough the serial interface. Figure 27 shows the interface of the control application. The interface was designed to operate the system using a numeric keypad, as the one shown in Figure 28.

The time benchmarking of experiments were performed in a towing tank for a three meters course under two different conditions, from rest (T1) and at a constant speed (T2), to be able to compare both maximum speed and acceleration. Each rowing mode had its course time taken five times and the average time was used for comparison.

The course length is 3 meters, for times T1 and T2. T2 has a 2 meters acceleration length before the start of benchmarking. The model size is about 40 cm in length by 30 cm in diameter, and 10 cm tall. The depth inside the water is about 7 cm. It is connected to the computer and power supply by a 3 meters wire. The Ro size is about 28 cm by 2 cm. Figure 29 summarizes the model dimensions and parts.



Shortcut	Control Codes	Action
1	0801350024023B602C02	Classic
2	08012C023B6024023500	Backward
3	080124422B602C122500	Optimal Classic
4	0801250020122B6028C2	Left
5	150018C103221302031216D11B6018C12C120641	Test
6	0801250027F22B602FE2	Right
7	2C822641038028C1237226410C8028C1	Front alt
8	2EC20B6026A1050018C1216226710B6018C1	Front
9	170117F297F00FF0	Align

Serial

Baudrate	9600
Data bits	8
Parity	None
Stop bits	1

☐ Resend loop

```
Program initialized...

Control Matrix
(cmd   pos  chk  time)
no data set
```

Figure 27 – Screenshot of the serial control application, the control codes are entered in the proper fields, optionally accompanied by its description. The mouse or the keyboard can be used to start sending the codes into a continuous loop.

Figure 28 – The program interface and its shortcuts were designed specifically to enable the full operation to be performed by keyboard or even simple portable numeric keypads connected to the system.

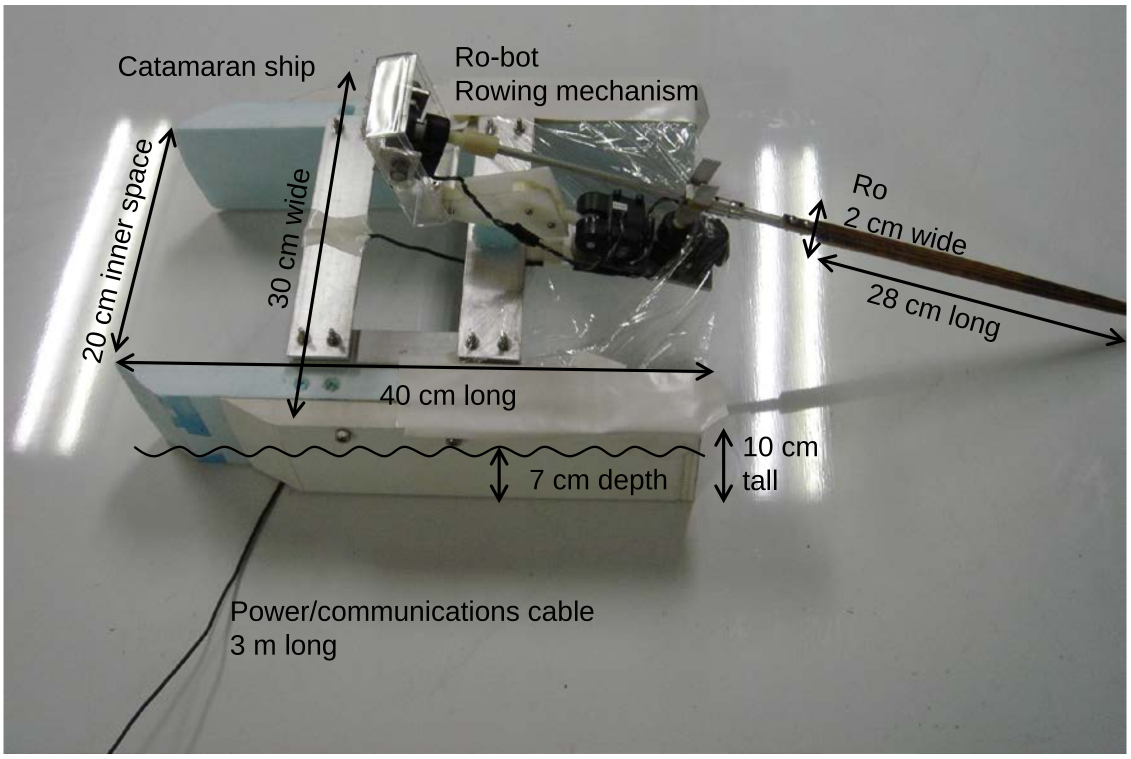


Figure 29 – Model dimensions and parts.

5.1.1. Cruise Time from Rest (T1)

The time was measured for the course starting from rest, meaning that at T=0, the Speed=0. This measure includes the time required for accelerate the model, and will allow the comparison of acceleration characteristics among rowing modes. Figure 30 describes the experiment for measuring T1.

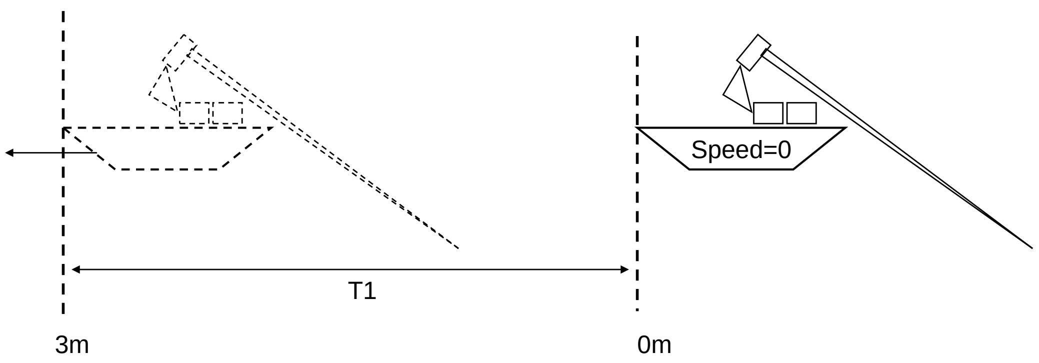


Figure 30 – Description of the experiment to measure T1. The timer starts with the model at rest at the staring point. It is activated simultaneously with the timer.

5.1.2. Cruise Time at Constant Speed (T2)

This measuring starts at a speed condition as the model is accelerated at some distance from the starting point of the course. This allows us to compare the rowing modes by maximum speed. Figure 31 explains the experiment to measure T2.

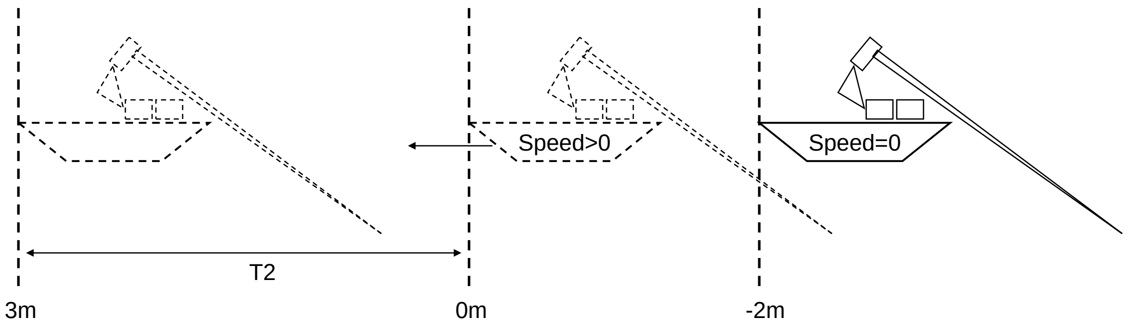


Figure 31 – Description of the experiment to measure T2. The model is accelerated before the initial position, the timer starts when the model crosses the starting point.

5.2. Experimental Results

Here the calculated fitness in the Y direction is compared to the times T1 and T2 obtained in the experiments, the comparison between the several modes. These results are analyzed and discussed in Chapter 7.

Table 1 – Simulation and experimental results comparison

MODE	F (K)	T1 (S)	T2 (S)
C-2DF	3.2	12.0	10.5
RB-2DF	2.9	11.5	10.2
O-2DF	4.6	10.5	9.6
X-3DF	6.6	11.4	10.5
M-3DF	5.4	12.0	11.0
U-3DF	6.2	11.2	10.6

Table 1 compares the fitness calculated from the evaluation in simulation to the course times for the two described conditions: T1, from rest and T2, at constant speed.

The correlation between the two timings can be better contemplated in the plots T1 and T2 versus fitness presented in Figure 32, Figure 33 and Figure 34, respectively.

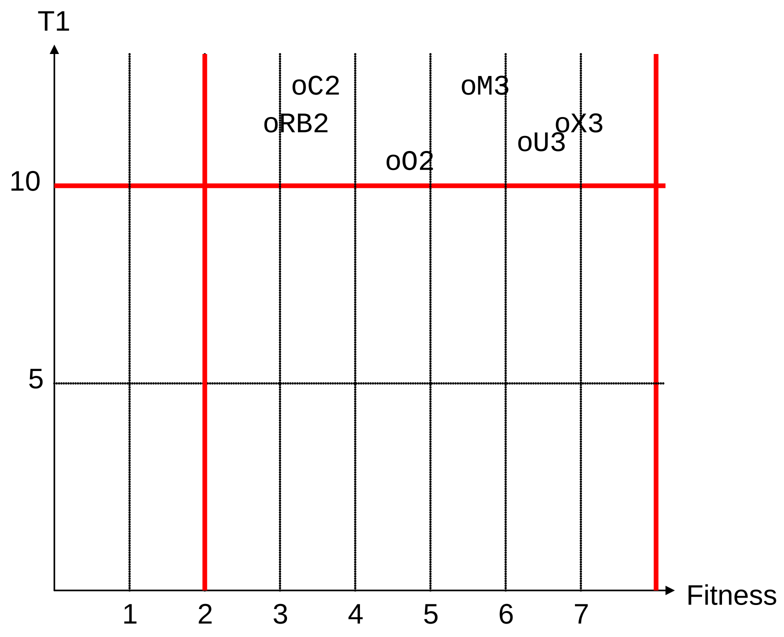


Figure 32 – Plot of the time measured T1 versus the fitness obtained for the control modes, the solutions are concentrated in a small area with close correlations.

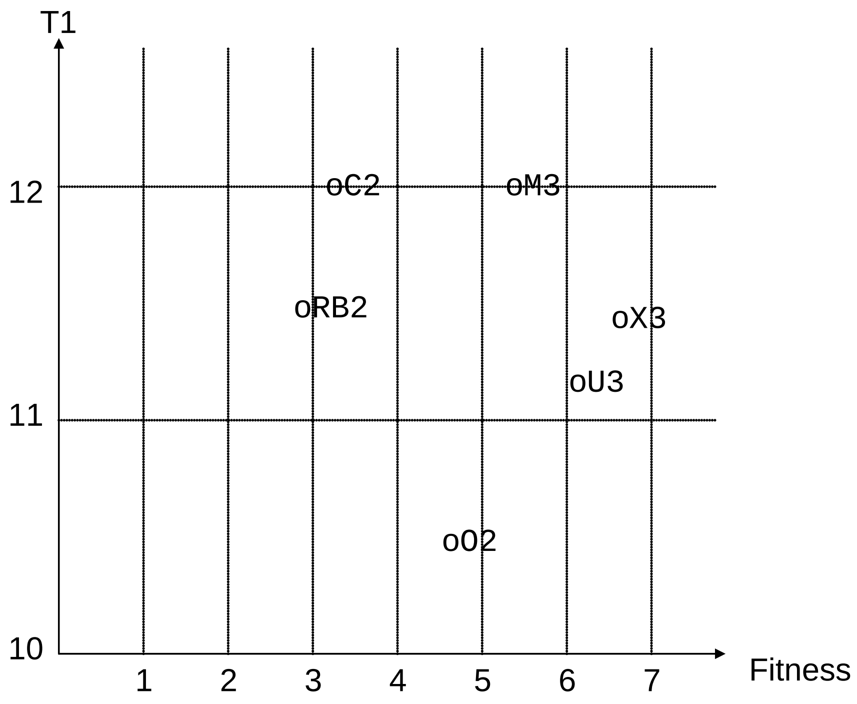


Figure 33 – Zoomed plot of the region where solutions are concentrated shows the correlation of points for 2DF and 3DF modes with acceleration included.

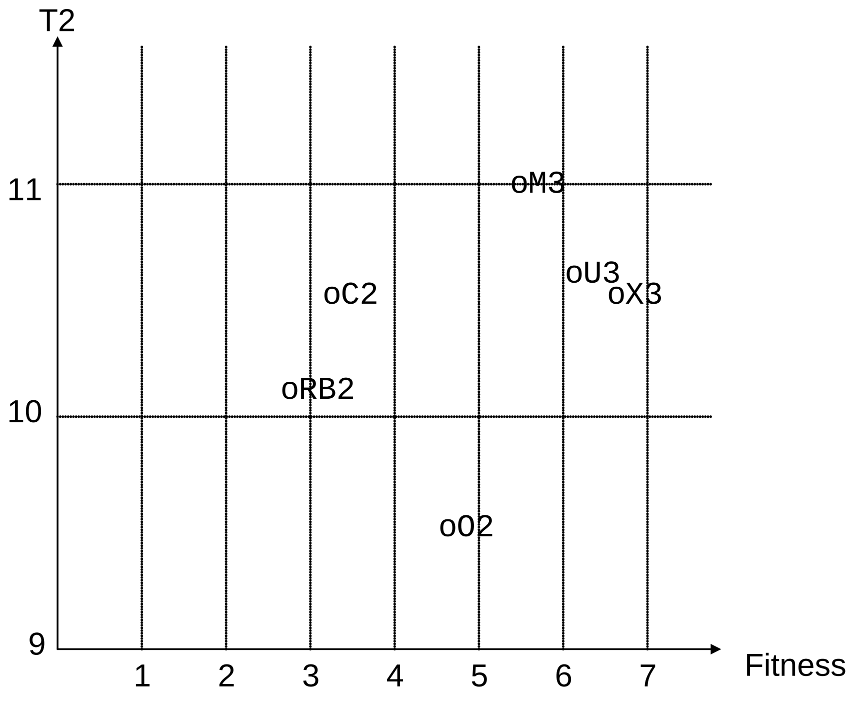


Figure 34 – Plot of T2 vs. fitness for the region where the solutions are concentrated shows the correlation of points for 2DF and 3DF modes without acceleration.

The solutions are concentrated in a small region closely correlated, but no linear distribution can be observed. The causes will be later discussed in Chapter 7, but for now is enough to say that the simulation considers the system to be fixed at the referential and in the actual experiment different rowing modes generated a soft of side effects, such tilt and swing of the whole set (ship and rowing mechanism) in addition to the increasing drag with speed for 3DF modes, causing the observed discrepancies between the simulated fitness and the measured timings.

Chapter 6

Computational Analysis

Overview

In this chapter we compare the results obtained in executing the MAS in different processing scenarios and settings by the computational perspective, pointing the particular performances and advantages obtained with each approach, according to the number of agents, processors and topology of network involved in the task.

6.1. The Single-Threaded Model

At first, in the single threaded experiment, a sub-optimal solution (10% of global maximum) was obtained after two days of continuous execution. The optimal control for each problem was found in intervals from 12-48 hours. In many occasions the experiment converged into a sub-optimal region of the space, never reaching out to the global optimum. To confirm the optimal values, was required to reset the MAS several times to random values. Further executions required less time, once they profit on evaluations stored into the database.

The progress, or system evolution, is obtained by observing the maximum fitness in the population over time. This gives the sense of how the solutions are progressing over the generations. Figure 35 shows the fitness over generations plot for the single thread model of GA.

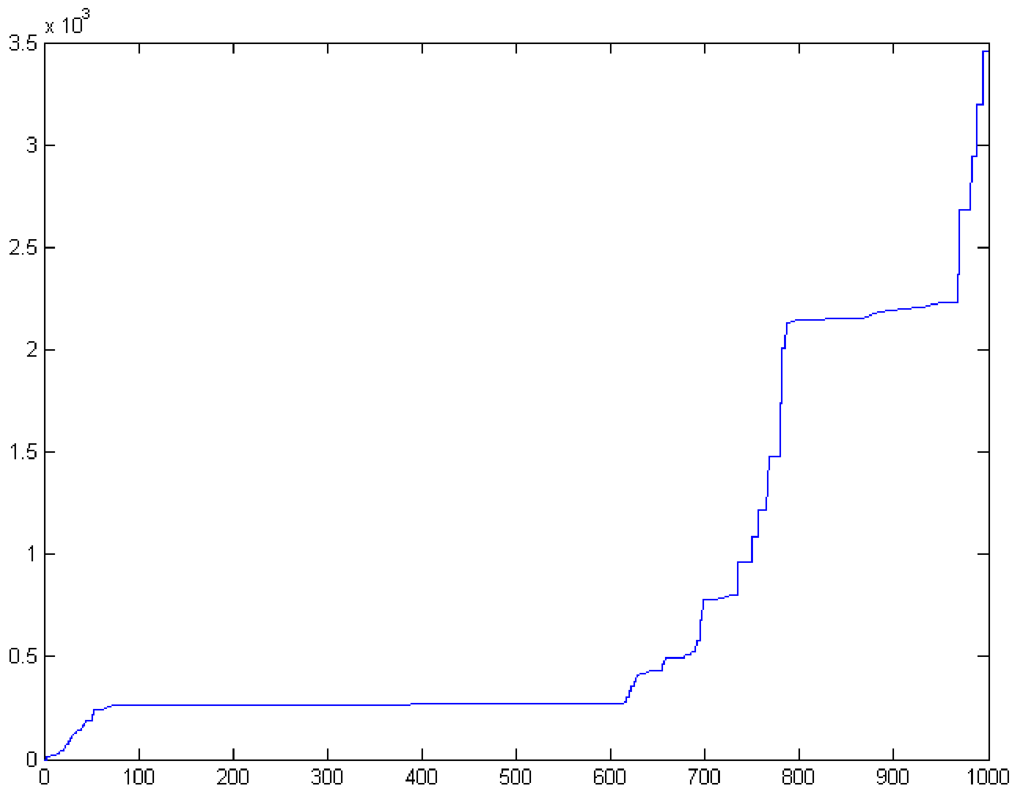


Figure 35 – Progress history of the maximum fitness in the population over the first 1000 generations for the single threaded GA model.

The final fitness in this graph is ~3.4k, but the experiment goes beyond 1000 generations. The total evaluation time for this example was 26h.

It is observed that the system many times stagnate for several generations, then suddenly, an evolutionary jump is observed. This happens in a fashion much similar to observed in the nature, where better fit individuals appear and later dominate the entire population. For GA, the size of the population and its variety often determines how frequently these evolutionary jumps happen.

6.2. The Multi-Threaded Model

The later agent-based experiments (multi-threaded) demonstrated the saying "two heads are better than one". For MAS running in one single, dual-core system (two processors), under the "divide and conquer" approach, initial sub-optimal solutions were found at a record of 15 minutes, and optimal values form 2 hours to 25 hours, depending on the configuration used. Concurrent populations not only decreased computation time for a limited number of agents, but the probability of getting stuck in local maximums was negligible for higher number of agents.

The MAS system was executed for several numbers of agents in the tested system, with two processors. The time was measured for the MAS to achieve fitness 10% close to the maximum known fitness. The plot of the results obtained is shown in Figure 36.

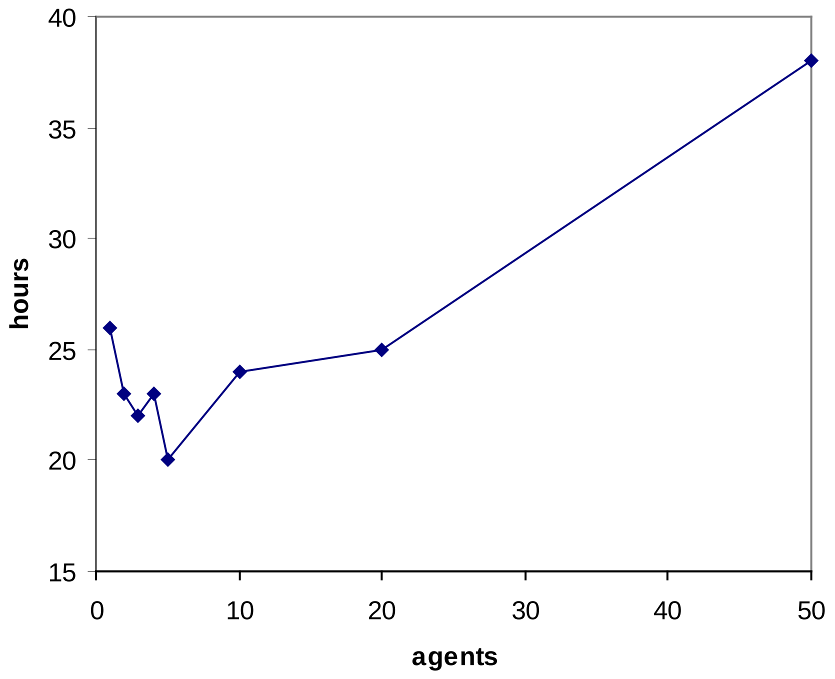


Figure 36 – Time required for reach a 10% near maximum fitness according to the number of agents allocated to the effort. The maximum speedup was obtained using around 5 to 8 agents in a dual core system.

It was observed that the MAS performed better as the number of agents is increased, reaching a maximum around 5-8 agents for this system. After that point, adding agents caused the system to run increasingly slow. As the resources, such memory and processing power, get scarcer and more programs are competing for these resources, executing a large number of agents simultaneously causes a total collapse of performance. It was observed that the number of agents must be associated with the resource usage, observing the limitations, especially for available memory, once disk swaps can cause the system to run at a very low performance.

The most efficient number of agents for each system depends of the system resources and the agent algorithm, or how much resources each single instance of agent allocates. Theoretically, for unlimited resources, the bigger the number of agents, better the algorithm converges (high granularity).

6.3. The Distributed Computing Model

In the last phase, we distribute the agents among all the available systems for comparison. The number of agents was dimensioned to obtain the maximum performance in each system.

In our current application, we have computers of different ages and performances. Agents executing in slow computers are at a disadvantage. Trying to keep the distribution uniform, we limit the number of agents by the available processing power. This way, we can have a similar agent/processor ratio by balancing the population by this criterion.

We measure the time for execution of the MAS individually in each system to obtain the performance calibration and determine the proper number of agents for that system. Later, we executed the MAS in a distributed fashion, with network communication trough the server, and measured the time to achieve a 10% near value of the maximum known fitness, as we add more systems to the task. Figure 37 summarizes the obtained times.

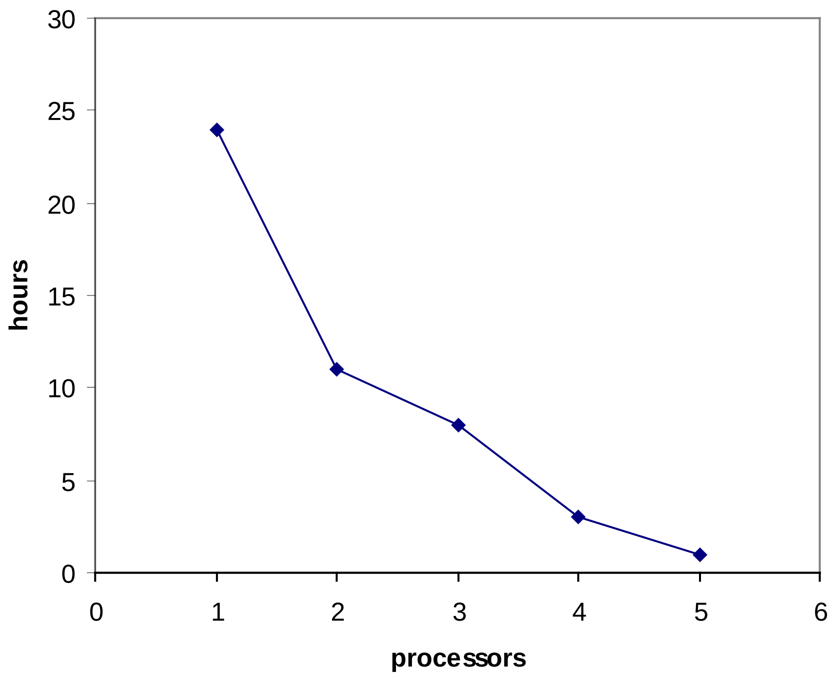


Figure 37 – Evaluation time vs. number of processors for 25 agents balanced uniformly among computers. The processor number is the total allocated in all the systems.

It was observed that the time decreased as more systems are added to the task. The limited number of systems available for the experiment gives an idea of the performance gain by utilizing a distributed MAS approach. The gain can be better analyzing by utilizing the Speedup, a common measure used to evaluate parallel algorithms [2] [3]. The Speedup is obtained by dividing the time required for execution in one single processor by the time required for execution in NP (number of processors).

$$SpeedUp = \frac{T(1P)}{T(NP)}$$

Figure 38 shows the plot for speedup versus number of processors.

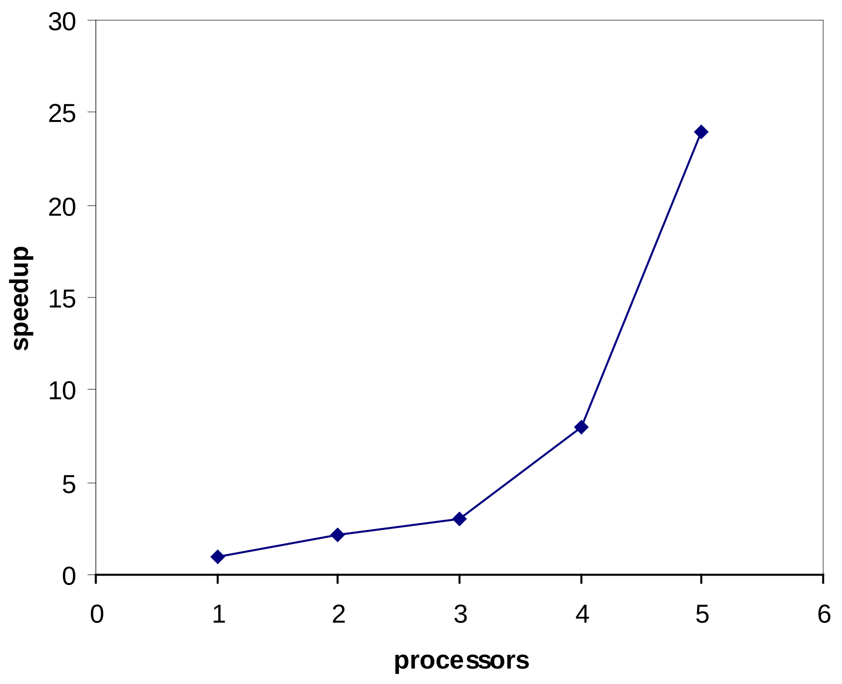
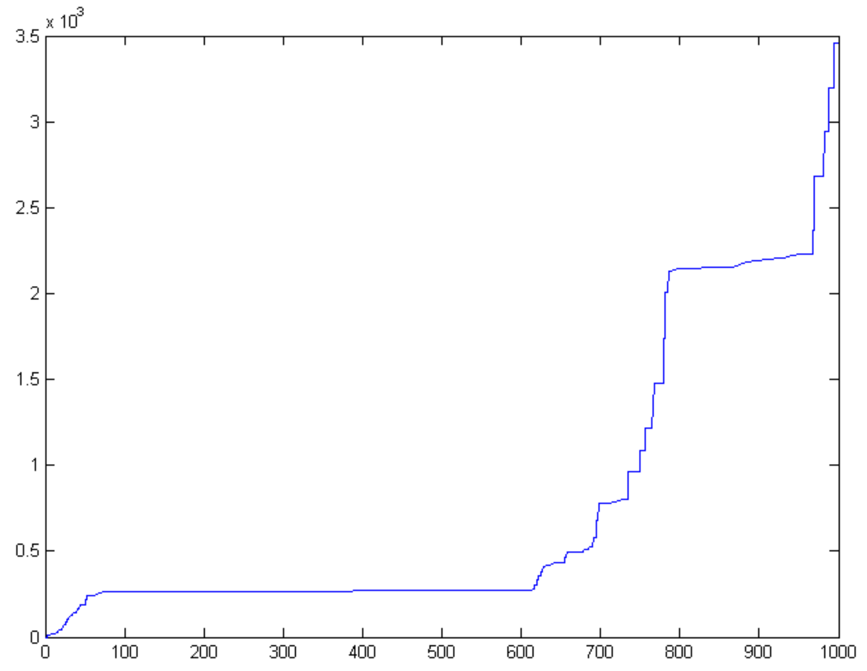


Figure 38 – Speedup: time required for execution in one processor by the time required for execution in NP. Figure shows a super-linear speedup [2].

People familiar with parallel algorithms knows that most problems present a less than linear speedups, other logarithmic speedups. But for some problems, the nature of the problem, when treated by parallel algorithms, generates what is called a super-linear speedup. This is what happens in this case. The nature of the problem, where new solutions depends intrinsically of previously found solutions, causes a larger number of agents to explore the space more efficiently, with a higher resolution for each partition, consequently achieving better solutions in less time.

6.4. Model Comparison

Analyzing the average time required for calculations in each case, it is observed that higher agent numbers are better, but a limitation exists, imposed by the available computational resources. By distributing agents among several systems, we obtained a super-linear speedup. The time is reduced usually to less than half for every added system, as could be observed in the presented experiments. Figure 39 compare the progress histories of the single-threaded and the distributed multi-agent model.



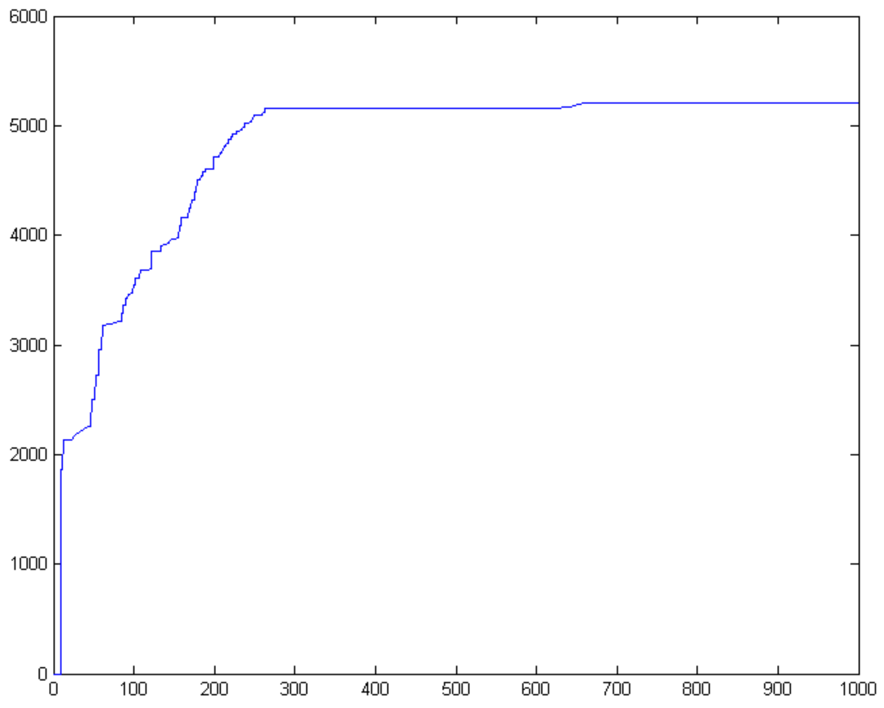


Figure 39 – Progress of maximum fitness over the first 1000 generations for the single threaded GA version and the distributed MAS model; the final fitness are 3.4k and 5.2k respectively; total evaluation time dropped from average 26h to an average 2h.

The following figures shows history plots of fitness for several experiments in different conditions, each described in the respective caption. The purpose of these plots is to spot the random nature of the Genetic process, which can radically vary for two consecutive experiments, even if restarted with unchanged parameters and conditions.

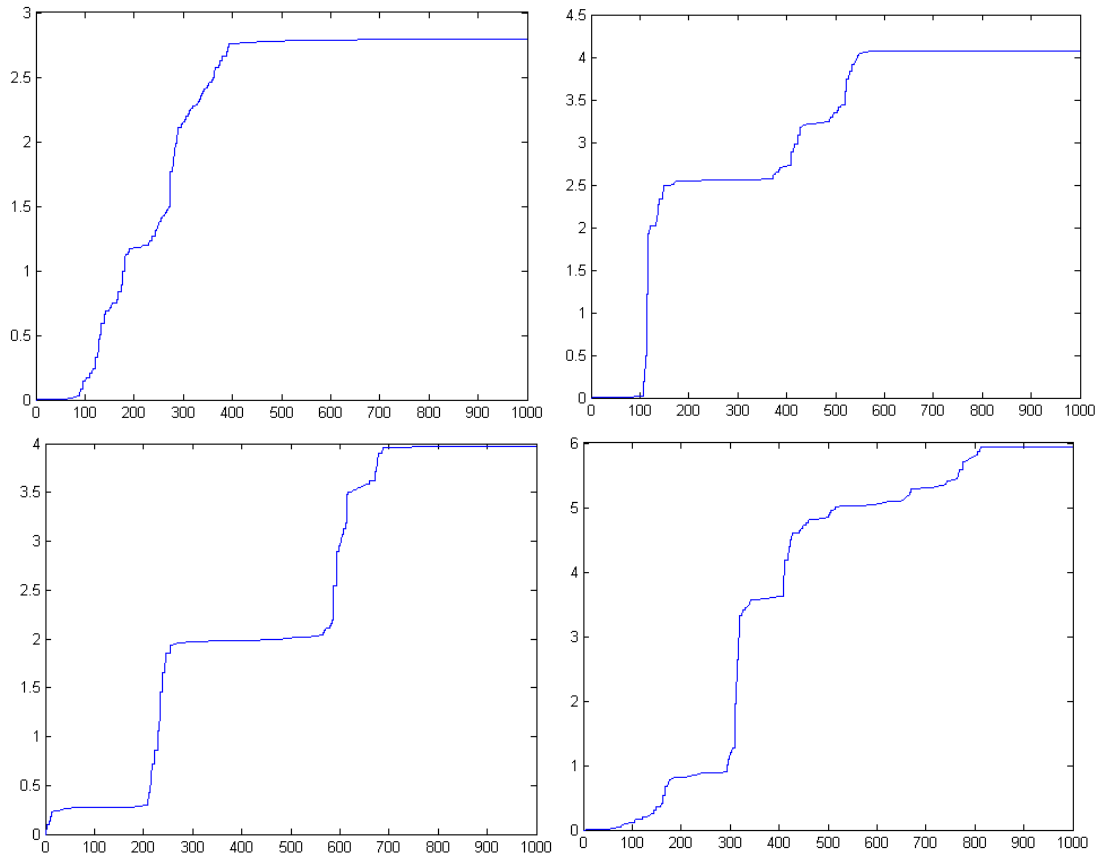


Figure 40 – Four alternative examples of the single threaded model showing cases of evolutionary jumps, points where a higher fitness is spotted, a completely random event.

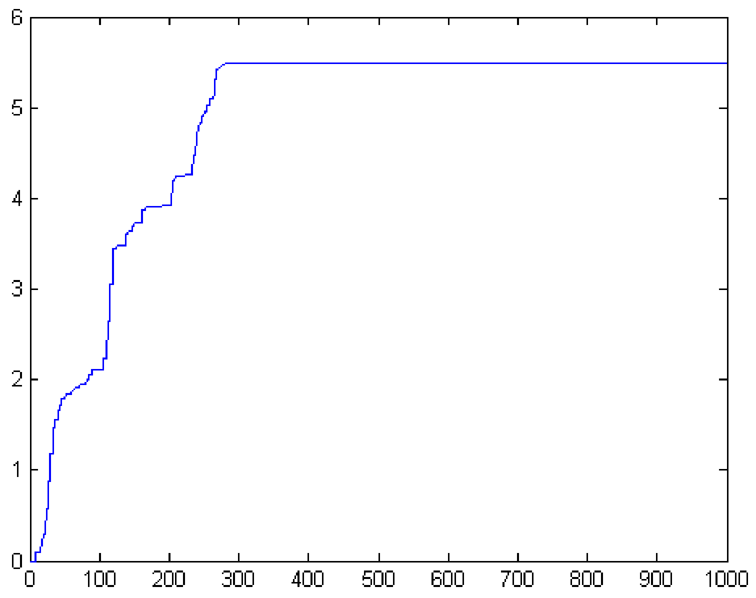


Figure 41 –Fitness history in the multi-threaded model considering only the first 1000 generations.

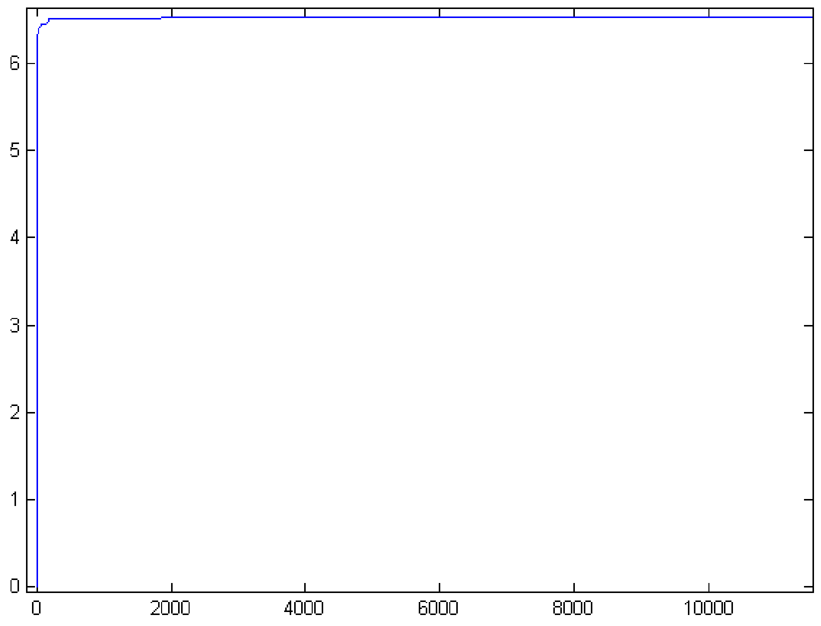


Figure 42 – Aspect of a full evaluation, in this example most of the progress is observed in the first 2000 generations, but the Agent only breaks operation after 3000 generations without any improvement. This case consumed 17h.

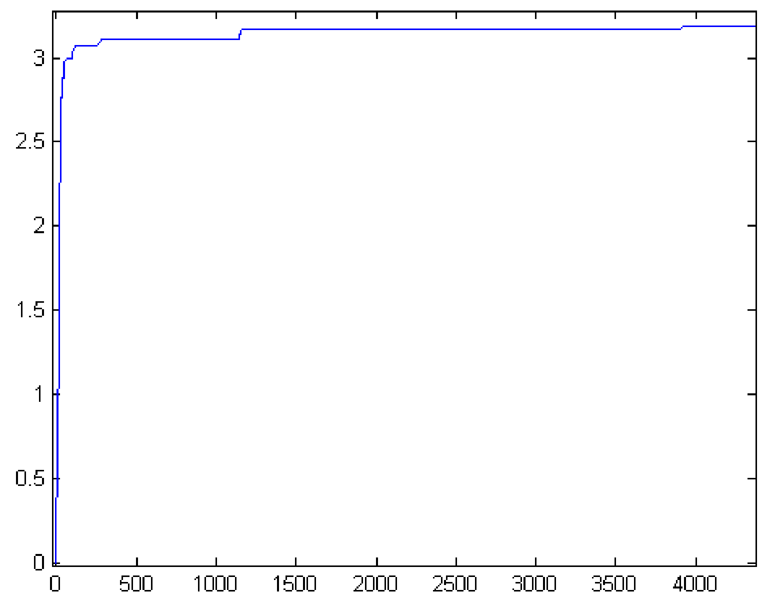


Figure 43 – When observed for a sufficiently large number of generations, the fitness history graph tends to look similar to the one presented in Figure 42 in most cases.

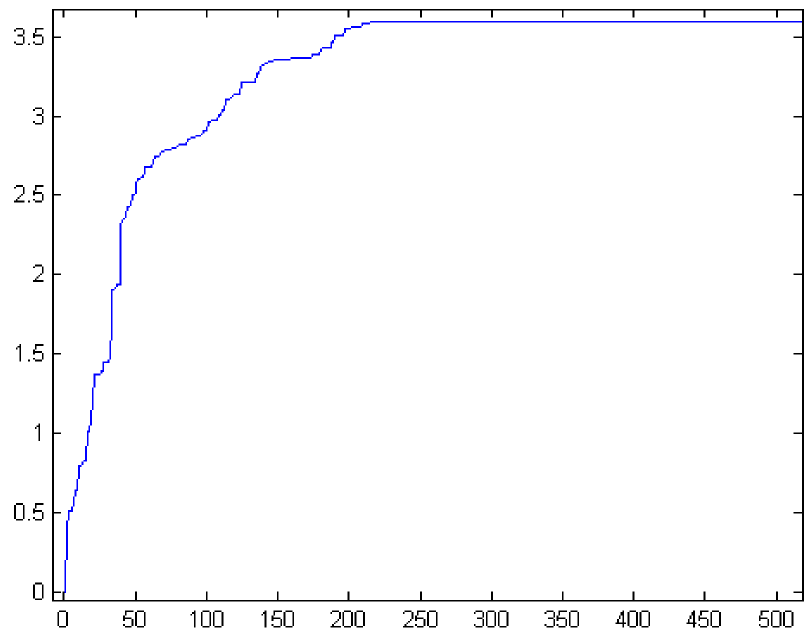


Figure 44 – Detail of the previous picture shows that a 10% near maximum value was achieved in the first 500 generations.

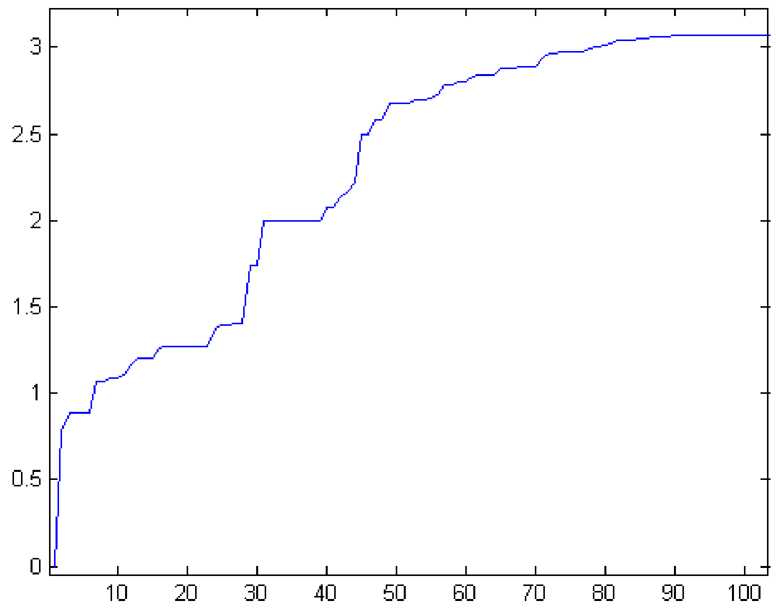


Figure 45 – Multi-threaded, multi-processed GA version history for the first 100 generations shows that the PGA approach provides a smoother progress.

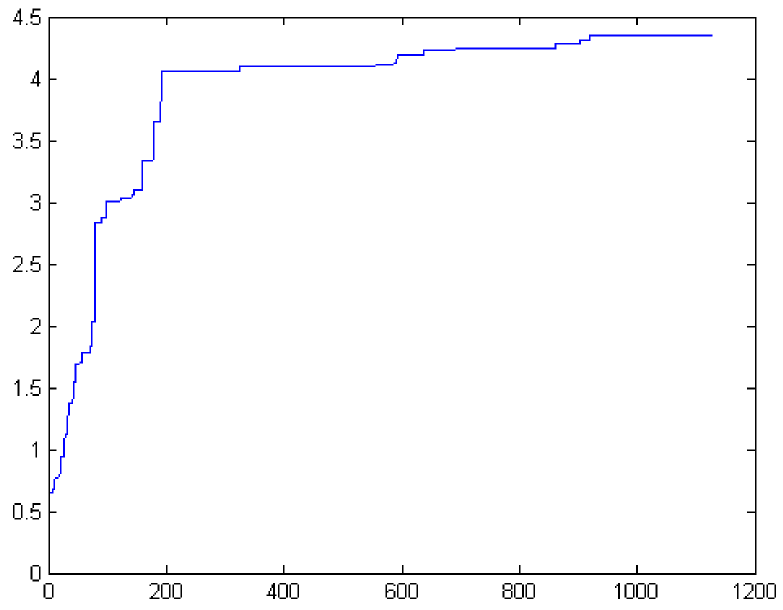


Figure 46 – A case study of PGA with a small population per agent (100 individuals) shows a crispy evolution in time, denoting that higher populations are preferable.

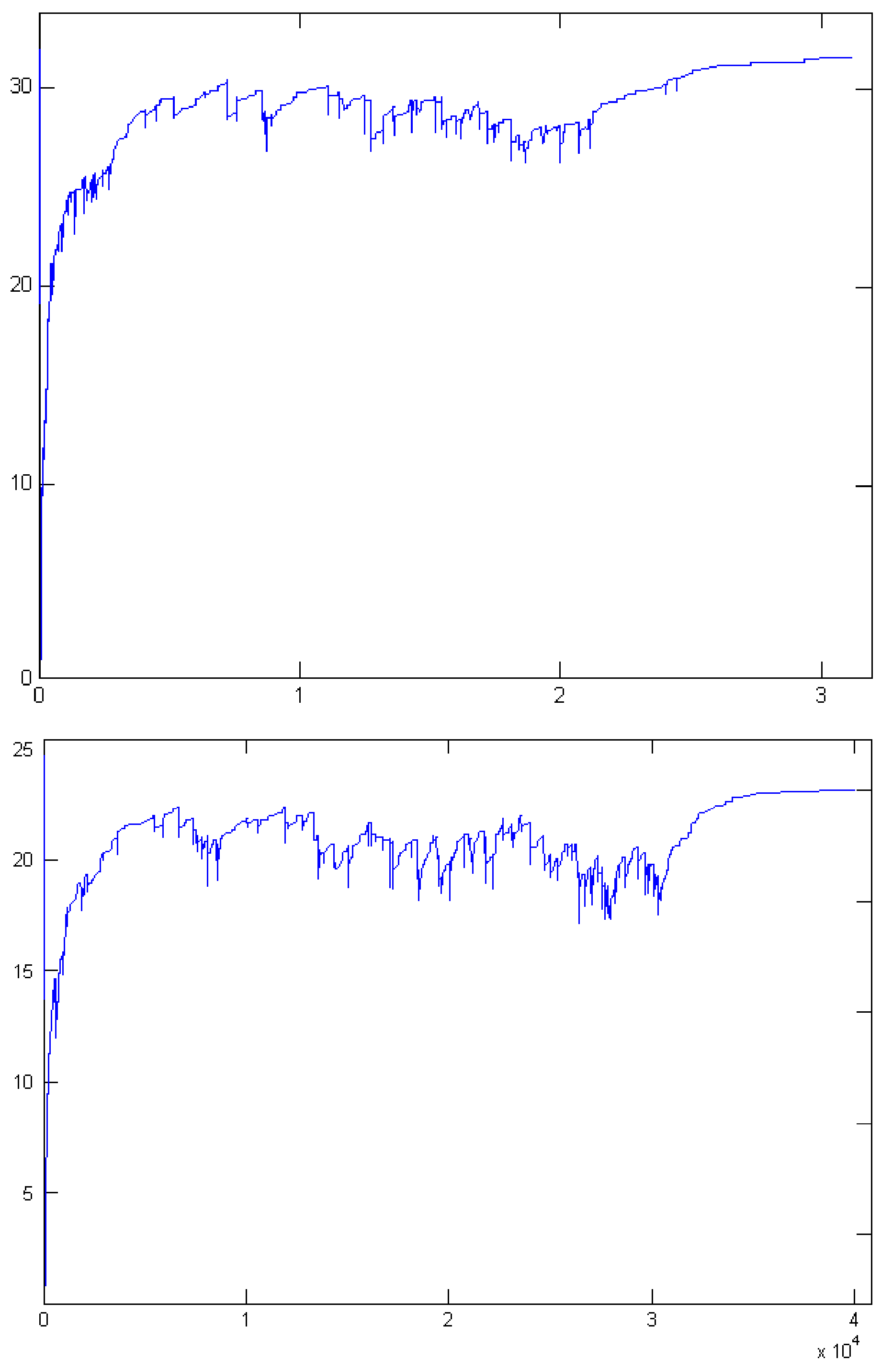


Figure 47 – Another way to determine the evolution of a system is by the average or the total fitness, the sum of fitness for the whole population of all agents or individual agents. To obtain the average fitness the value must be divided by the global population.

Chapter 7

Discussion and Conclusions

7.1. Discussion

In the 2DF experiments, the lowest resistance to the water flow is obtained in the RB-2DF, but this mode is still less efficient than the newly found O-2DF, which is able to achieve higher speeds. Nevertheless, RB-2DF revealed a superior efficiency over the traditional classic rowing C-2DF.

In the later 3DF (three degrees of freedom) experiments, it was noticed that the 3DF modes present a better acceleration, but an inferior top speed. The experimental results were not as optimistic as expected by the simulation analysis, considering the experimental Ro-bot mounted in a ship do not operate in the same conditions as expressed in the simulations for 3DF modes.

The simulation was based in the first built model of the Ro-bot experiment, where the Ro-bot was mounted over a thrust block, having its movements limited by the assembly at a fix position, having only the Ro blade as moveable part. When incorporated into the catamaran style ship, constructed later for additional comparisons, most of the modes generated a large deal of tilt, splash and other side effects over the whole system.

The discrepancy is also a result of some of the approximations made to implement a fast simulation, which considered a static fluid and just measure the impulse caused by the swing of the blade, with the ship mechanism being considered fixed into a referential. This approximation benefited the acceleration factor for the case when the fluid speed is low, explaining the high fitness obtained for simulated 3DF modes. The bottom-line is that under certain fluid motion conditions (higher speeds), moving the blade back and forward adds an increasing resistance to the water flow, increasing the drag and therefore reducing the maximum speed. The price paid for a fast simulation was that this behavior was not predicted. A full hydrodynamic simulation, exactly as considered in the water-borne experiment, would imply much slower simulations, resulting in exponentially long evaluation times. The limited project schedule and computational power required a fast model in order to provide benchmarking information and verify he efficiency of the method. The results obtained feature low consumption and hi-speed, as in O-2DF, and high acceleration, verified in X-3DF. However, a full featured simulation may spot even more efficient rowing modes for hi-speed conditions.

Yet, the new found results in 3DF could be proven to be as effective as the modes in 2DF, and even more efficient in some aspects, especially when compared with the Classic human performed rowing style.

7.2. Conclusion

The experiments demonstrated that the method can successfully locate and identify potential solution areas in the multi-variable space, potentially spotting the highest fitness solution for a specified control problem.

The method was able to identify new solutions for the proposed problem, in addition to the conventional, optimized solution as expressed by the evaluation function. Some of the newly found solutions clearly demonstrate to be non-intuitive, being hardly achieved by human specification methods. Other useful solutions needed for maneuvering (turning motions), could be easily identified by simply changing the fitness function to evaluate vector projections in different directions.

Using the actual model, the results could be tested and compared, allowing the improvement of the theoretical computer simulation, as well as verifying the feasibility of spotted potential new rowing modes.

The agent-oriented approach enabled the achievement of solutions in a reduced time frame, obtaining super-linear speedups for an increased number of agents and processors involved.

Once the method has been used successfully for the proposed problem, it is safe to assume that it can be extended to the treatment of more complex control system problems, such as navigation, collision avoidance or docking procedures.

It is fundamental to remark that the reliability of the described method mainly depends on the accuracy of the computer simulation used to evaluate the solutions. As noticed in the current experiment, small discrepancies can lead to unpredictable effects.

7.3. Suggestions for Future Works

The current work neglected some characteristics of ship motion in order to provide fast benchmarking. A full hydrodynamic model of the ship, considering speed, tilt and momentum, can produce more satisfactory results, and even reveal new rowing modes not yet discovered.

The physical model had a limitation imposed by a wired connection to the computer, used for serial communication and power supply. This limited the maximum length for experiments. A wireless solution for communications, as well as a built in power supply would remove this limitation. Unfortunately, it would also increase the overall weight of the model, decreasing speed and slowing acceleration.

The same method described can be adapted without much difficulty to simulate other maneuvering characteristics of Ro, as well as to solve other control problems in marine science. Here are some examples:

Determine an autonomous obstacle avoidance guideline for ships, with optimal response times and lowest fuel consumption.

Establish an autonomous docking system for computer controlled ships, using an input-output matrix relation.

Determine intelligent controls for up-to-date manually performed tasks in several aspects of ship operation.

Chapter 8

References

1. P. J. Bentley, (Editor), *Evolutionary Design by Computers*, Morgan Kauffmann (1999), ISBN: 155860605-X

2. A. Grama, G. Karypis, V. Kumar, A. Gupta, *An Introduction to Parallel Computing: Design and Analysis of Algorithms*, Addison Wesley (2003) Second Edition, ISBN: 0201648652

3. I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison Wesley (1995), ISBN: 0201575949, also available on http://www-unix.mcs.anl.gov/dbpp/, last access June/2006.

4. Mitchell, M., *An Introduction to Genetic Algorithms*, MIT Press (1997), ISBN: 0262133164

5. Michalewic, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag (1996), ISBN: 3540606769

6. Cantú-Paz, E., *Efficient and Accurate Parallel Genetic Algorithms*, Volume 1 of the Book Series on Genetic Algorithms and Evolutionary Computation, Kluwer Academic Publishers (2002), ISBN: 0792372212

7. Ferber, J., *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*, Harlow: Addison Wesley Longman (1999), ISBN 0201360489

8. Xie, M., *Cooperative Behavior Rule Acquisition for Multi-Agent Systems using a Genetic Algorithm*, On “Advances in Computer Science and Technology”, ACST 2006, ACTA Press (2006), ISBN: 0889865450

9. Killian, C., *Modern Control Technology - Components & Systems*, Thomson Delmar Learning, 3rd edition, ISBN: 1401858066

10. Yevick, D., *A First Course in Computational Physics and Object-Oriented Programming with C++*, Cambridge University Press (March 17, 2005), ISBN: 0521827787

11. Huntbach, M. M.; Graem A Ringwood, G. A., *Agent-Oriented Programming: From Prolog to Guarded Definite Clauses*, Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence. Vol. 1630, pp. 329-333, Springer (1999), ISBN: 3540666834

12. Hayes-Roth, B., *An Architecture for Adaptive Intelligent Systems*, Artificial Intelligence: Special Issue on Agents and Interactivity 72 (1995), pp. 327-365

13. Kaplan, D.; Glass, L., *Understanding Nonlinear Dynamics*, Springer (1995), ISBN: 0-387-94440-0

14. Kreyszig, E., *Advanced Engineering Mathematics*, John Wiley & Sons; 8th edition (January 1999), ISBN: 0-471-15496-2

15. Neves, R. P. O.; Netto, M. L., Evolutionary Search for Optimization of Fuzzy Logic Controllers, 1st International Conference on Fuzzy Systems and Knowledge Discovery, Volume I, on Hybrid Systems and Applications I (2002), ISBN: 9810475209, available at http://www.lsi.usp.br/~rponeves/work/fuzzy/FSKD'02 Evolutionary search for FLC.PDF , last access June/2006

16. Shoham, Y., Agent-Oriented Programming, Readings in Agents, edited by Huhns, M.N.; Singh, M.P., pp. 329-349 - Morgan & Kaufmann, S. Francisco (1998), available http://www.damas.ift.ulaval.ca/~coursMAS/Agent-Oriented-Programming.pdf, last access June 2006.

17. Neves, R. P. O. and Netto, M. L., *A Virtual Reality Framework for Artificial Life Simulations*, VII Symposium on Virtual Reality Proceedings, Sao Paulo, SP, pp. 217-227, ISBN 8576510065, available at http://cognitio.incubadora.fapesp.br/portal/producao/artigos%20eventos/SVR2004/2004.06_RPON_SRV-alive-VF.pdf , last access June/2006.

18. Various contributors, *Multi-Agent Systems*, http://www.multiagent.com/, last access June/2006.

19. Various contributors, *Genetic Algorithms Archive*, http://www.aic.nrl.navy.mil/galist/, last access June/2006.

20. Hirakawa, Y., *Sea and Air Control Systems Laboratory Home page*, http://www.seakeeping.shp.ynu.ac.jp/index-e.html .

Appendix – Source Codes

Overview

In this single appendix, the latest source codes used to generate the solutions are presented. The source codes presented here are the final versions, presenting some improvements over previous versions and removing some less used functions for simplicity. The math and subroutines were simplified and separated in order to make easier the code comprehension. The codes are Matlab sources, the C++ sources used to implement the executable versions were derived directly from the compiled versions of such codes using the Matlab compiler Matcomp. The alterations needed to provide distributed functionality are described as comments within the code.

Main Code

This code performs most of the agent functionality as well as the GA functionality.

```
function main()

% Main program for MASM (no args)

%% Initialization
mat=zeros(15,3,100);
% Enter partial results here
 % mat(:,:,1)=fil(hmat('0500164117F124020B60164117F12C02'));

beginwith=1; % and change the index for 1st randomly created

```

```
datapath='solutions/';  % Change for target network path
                        % e.g. '//IP/path/solutions/' for remote agents
tempdrive='T:/'; % Use ramdrive or temp partition if unavailable
disp(['datapath is "' datapath '"']);
disp(['tempdrive is "' tempdrive '"']);
if exist(datapath,'dir')~=7
   disp([datapath ' not found, creating folder structure...']);
   mkdir(datapath);
   mkdir([datapath 'high']);
end
if exist([datapath  'db/'],'dir')~=7
   mkdir([datapath 'db/']);
end
if exist([tempdrive 'temp/'],'dir')~=7
   mkdir([tempdrive 'temp/']);
end
if exist([tempdrive 'temp/masm/'],'dir')~=7
   mkdir([tempdrive 'temp/masm/']);
end
for i=0:9
   if exist([tempdrive 'temp/masm/' num2str(i) '/'],'dir')~=7
      mkdir([tempdrive 'temp/masm/' num2str(i) '/']);
   end
end
if exist([datapath 'data.mat'],'file')==0,
   disp([datapath 'data.mat not found, starting from scratch!']);
   index=zeros(size(mat,3),1);
   fm=0;
   hist=[0; fm]; % Time History
   disp('Creating entries...');
   % set i=1:size to discard the partial results
   % else, set i=next blank entry:size
   for i=beginwith:size(mat,3)
      mat(:,:,i)=fil(create());
   end
   disp([num2str(size(mat,3)) ' entries created!']);
```

```
else
    disp('Loading data.mat...');
    load([datapath 'data.mat'],'index','fm','mat','hist');
end
gen=0;

%% Main Loop
tic;
disp('Evaluating...');
while(1)
    tested=0;
    reused=0;
    gen=gen+1;
    for i=1:size(mat,3),
        submat=clip(mat(:,:,i));
        hs=hash(submat);
        if exist([tempdrive 'temp/masm/' hs(1) '/' hs '.mat'],'file')==0,
            f=evl(submat);
            ftm=-f(2);
            mat(15,3,i)=ftm; % single fitness for selection
            if (ftm>fm),
                fm=ftm;
                fid=fopen([datapath 'high/' hs],'w');
                fclose(fid);
                save([datapath 'data.mat'],'index','fm','mat','hist');
                save([datapath 'best.mat'],'submat');
                disp(['New best: (Ef=' num2str(fm) ')  ' hs ' Saved.']);
            end
            save([tempdrive 'temp/masm/' hs(1) '/' hs '.mat'],'ftm')
            % uncomment to enable db-track functionality
            %fid=fopen([datapath 'db/masm/' hs(1) '/' hs '.mat'],'w');
            %fclose(fid);
            tested=tested+1;
        else
            load([tempdrive 'temp/masm/' hs(1) '/' hs '.mat'],'ftm');
            mat(15,3,i)=ftm; % single fitness for selection
```

```
          reused=reused+1;
      end
   end
   % Sort index
   [ef index]=sortrows(reshape(mat(15,3,:),[size(mat,3),1]));
   ef(1:size(mat,3))=ef(size(mat,3):-1:1);
   index(1:size(mat,3))=index(size(mat,3):-1:1); % Decrescent

    Apply GA rules
   % disp('Performing genetics...');
   best=1;
   worst=size(mat,3); % first to go
   % Create some new
   while(worst>.95*size(mat,3))
      mat(:,:,index(worst))=fil(create());;
      worst=worst-1;
   end
   % Best: clone/mutate clone
   while(best<.01*size(mat,3))
     mat(:,:,index(worst))=fil(mutate(clip(mat(:,:,index(best)))));
      best=best+1;
      worst=worst-1;
   end
   % Time to live
   for i=1:5
      if mat(15,2,index(i))<1, mat(15,2,index(i))=6;
      elseif mat(15,2,index(i))==1,
         mat(:,:,index(i))=fil(mutate(clip(mat(:,:,index(i)))));
         disp(['Elite top ' num2str(i) '/5 died.']);
      elseif mat(15,2,index(i))>1,
mat(15,2,index(i))=mat(15,2,index(i))-1;
      end
   end
   % Crossover = 2*n offspring from n pairs (best quarter only)
   for i=1:.2*size(mat,3)
      a=clip(mat(:,:,index(floor(rand*size(mat,3)/4)+1)));
```

```
        b=clip(mat(:,:,index(floor(rand*size(mat,3)/4)+1)));
        [c d]=crossover(a,b);
        mat(:,:,index(worst))=fil(c);
        worst=worst-1;
        mat(:,:,index(worst))=fil(d);
        worst=worst-1;
    end
    % Mutate all the rest
    while(worst>5)
         mat(:,:,index(worst))=
             fil(mutate(clip(mat(:,:,index(worst)))));
        worst=worst-1;
    end
    hist=[hist; fm];
    save([datapath 'data.mat'],'index','fm','mat','hist');
    c=clock;
    disp(['[' num2str(gen) '-Gen] ' date ' ' num2str(c(4)) ':' ...
           num2str(round(c(5))) ', '...
           num2str(round(toc/60)) ' minutes total, '...
           num2str(tested) ' tested, '...
           num2str(reused) ' reused.']);
end
```

Evaluation function

This is called from inside the MAS code and performs the simulation of the Ro blade, evaluating the thrust related force for the specified time.

```
function evl(mat,cord)

% ef=evlg(mat,cord)
%  Evaluates the efficiency of command matrix mat

if nargin<1,
   mat=[ 32 85 300
        34 85 200
        32 169 300
        34 169 200 ]; % 350024023B602C02
end;

%% Simulation variables initialization
line=1;
wait=0;
time=10000;       % Specified time in milliseconds
lines=size(mat,1);
pos=[127 127 127];            % Initial position
dest=[127 127 127];           % Initial destination
step=[.30 .16 .45];           % Step in 1/1000 s

%% Evaluation variables initialization
ft=[0 0 0];                   % Total force applied

aph=(pos(1)-127)/89.141;      % Alpha angle
bt=-(pos(2)-92)/57.3;         % Beta angle
RX=-cos(bt)*sin(aph);
RY=cos(bt)*cos(aph);
RZ=sin(bt);
ref=[RX RY RZ]; % Reference vector
```

```
all=[0 0 0];
%% Main Loop
for t=1:time
    %% Enter command
    if wait<1
        motor=mat(line,1)-31;
        newpos=mat(line,2);
        wait=mat(line,3);
        dest(motor)=newpos;
        line=line+1;
        if line>lines, line=1;
        end
    end
    direction=((dest-1)>pos)-((dest+1)<pos);
    pos=pos+(step.*direction);
    wait=wait-1;

    %% Evaluation process
    aph=(pos(1)-127)/89.141;
    bt=-(pos(2)-92)/57.3;
    th=(pos(3)-127)/84.667;

    X=sin(th)*cos(aph)+cos(th)*sin(bt)*sin(aph);
    Y=sin(th)*sin(aph)-cos(th)*sin(bt)*cos(aph);
    Z=cos(th)*cos(bt);

    RX=-cos(bt)*sin(aph);
    RY=cos(bt)*cos(aph);
    RZ=sin(bt);

    aref=ref;
    normal=[X Y Z];
    n=-normal;
    ref=[RX RY RZ];
    attack=aref-ref;
    impulse=(dot(attack,n)/dot(n,n))*n;
    ft=ft+impulse;
    % Activate this comments for the motor positioning plot
    % projection (dot(a,b)/dot(b,b))*b
    % [sqrt(X^2+Y^2+Z^2) X Y Z aph bt th]

    %% Graph 2D
    %plot(t,pos(1),'r-','Markersize',3,'Erasemode','none');
    %plot(t,pos(2),'y-','Markersize',3,'Erasemode','none');
    %plot(t,pos(3),'b-','Markersize',3,'Erasemode','none');

    %% Gpaph 3D
    %ap=attack*100;
    %g=impulse*100;
    %plot3([g(1) 0 ap(1) 0 RX 0 X ],…
            [g(2) 0 ap(2) 0 RY 0 Y],…
            [g(3) 0 ap(3) 0 RZ 0 Z]);
    %axis([-1 1 -1 1 -1 1]);

    %Graph common
    all=[all; impulse];
    %grid on;
    %drawnow;
end
f=ft;

%% Graph
figure(1); clf;
subplot(3,1,1);
axis([1000 time min(all(:,1)) max(all(:,1))]); hold on; box;
plot(all(:,1)','r-');
grid on;
title('X');
subplot(3,1,2);
axis([1000 time min(all(:,2)) max(all(:,2))]); hold on; box;
plot(all(:,2)','r-');
grid on;
```

```
title('Y');
subplot(3,1,3);
axis([1000 time min(all(:,3)) max(all(:,3))]); hold on; box;
plot(all(:,3)','r-');
grid on;
title('Z');

%legend('Motor 0','Motor 1','Motor 2',3);
```

Visualization function

Use this function to visualize the movements in the generated solutions, passing as argument the matrix generated by the MAS main program or the hash code, together with *hmat('hash')*.

```
function vis(mat, mode, time)

% vis(hsh,mode,time,framerate)
%  Graphs moves
tic
if nargin<1,
    mat=[32    80   300
         34    64   200
         32   182   300
         34   192   200]; % QUICK
end;

if nargin<2,
    mode=1;
end
if nargin<3,
    time=6000;
end
```

```
%% Simulation variables initialization
line=1;
wait=0;
data=[0 0];

[lines cols]=size(mat);
pos=[127 127 127];            % Initial position
dest=[127 127 127];           % Initial destination
step=[.30 .16 .45];            % Step in 1/1000 s

%% Evaluation variables initialization
ft=[0 0 0];                   % Total force applied

aph=(pos(1)-127)/89.141;      % Alpha angle
bt=-(pos(2)-92)/57.3;         % Beta angle
RX=-cos(bt)*sin(aph);
RY=cos(bt)*cos(aph);
RZ=sin(bt);
ref=[RX RY RZ]; % Reference vector

%% Graph nitiation

fig = figure(1);
set(fig,...
  'Color',[1 1 1],...
  'InvertHardcopy','off',...
  'PaperUnits','points',...
  'PaperPosition',[1 311 640 480],...
  'PaperSize',[640 480],...
  'PaperType','<custom>');
clf;

if mode==0
    title('Motor positioning');
    set(gca,'Drawmode','Fast');
    axis([0 time 0 260]); hold on; box;
```

```
else
    title('Vectors');
    set(gca,'Drawmode','Fast');
    plot3([-1 1 1 -1 -1],[1 1 -1 -1 1],[0 0 0 0 0],'-b','Erasemode','none');
    axis([-1 1 -1 1 -1 1]);hold on; box; grid on;
    p1=plot3([0 0 0 0 0 0],[0 0 0 0 0 0],[0 0 0 0 0
0],'-k','Erasemode','normal');
    p2=plot3([1 0 0],[0 1 0],[0 1 0],'-r','Erasemode','normal');
    p3=plot3([0 1 0],[0 1 0],[0 1 0],'-g','Erasemode','normal');
    drawnow;
end
%% Main Loop
for t=1:time

    %% Enter command
    if wait<1
        motor=mat(line,1)-31;
        newpos=mat(line,2);
        wait=mat(line,3);
        dest(motor)=newpos;
        line=line+1;
        if line>lines, line=1;
        end
        % disp([motor newpos wait]);
        disp(pos)
    end
    direction=((dest-1)>pos)-((dest+1)<pos);
    pos=pos+(step.*direction);
    wait=wait-1;

    %% Evaluation process
    aph=(pos(1)-127)/89.141;
    bt=-(pos(2)-92)/57.3;
    th=(pos(3)-127)/84.667;

    X=sin(th)*cos(aph)+cos(th)*sin(bt)*sin(aph);
```

```
Y=sin(th)*sin(aph)-cos(th)*sin(bt)*cos(aph);
Z=cos(th)*cos(bt);

RX=-cos(bt)*sin(aph);
RY=cos(bt)*cos(aph);
RZ=sin(bt);

aref=ref;
normal=[X Y Z];
n=-normal;
ref=[RX RY RZ];
attack=aref-ref;
impulse=(dot(attack,n)/dot(n,n))*n;
ft=ft+impulse;

% projection (dot(a,b)/dot(b,b))*b
% [sqrt(X^2+Y^2+Z^2) X Y Z aph bt th]
if mode==0   %% Graph 2D
   plot(t,pos(1),'r-','Markersize',3,'Erasemode','none');
   plot(t,pos(2),'y-','Markersize',3,'Erasemode','none');
   plot(t,pos(3),'b-','Markersize',3,'Erasemode','none');
else   %% Gpaph 3D
   ap=attack*100;
   g=impulse*100;
   set(p1,'Xdata',[0 -Z/16 RX Z/16 0 X/4]);
   set(p1,'Ydata',[0 X/16 RY -X/16 0 Y/4]);
   set(p1,'Zdata',[0 X/16 RZ -X/16 0 Z/4]);

   set(p2,'Xdata',[0 ap(1) 0]);
   set(p2,'Ydata',[0 ap(2) 0]);
   set(p2,'Zdata',[0 ap(3) 0]);

   set(p3,'Xdata',[0 g(1) 0]);
   set(p3,'Ydata',[0 g(2) 0]);
   set(p3,'Zdata',[0 g(3) 0]);
end
```

```
    drawnow;
    data=[data;t impulse(2)];
end
disp(ft)
f=sqrt(ft(1)^2+ft(2)^2+ft(3)^2);
disp(f)
%% Graph offset
if mode==0
    legend('Motor 0','Motor 1','Motor 2',3);

end
```

Helper functions

These functions provide side functionality for other members of the package, as described in the first command for each function.

Standardize the matrix format to 14 lines

```
%% Adjust aspect
function c=fil(a)  % Fill zeros
c=a;
n=size(a,1);
if n>14, n=14;
end

c(15,:)=[n 0 0];
```

Remove zeroed lines from matrix

```
function c=clip(a)  % Remove zero lines

c=a(1:a(15,1),1:3);
```

Performs mutation

```
%% Mutational operator
```

```
function c=mutate(a)  % Mutate matrix argument
ProbSamllMut=1; % x100=%
ProbSwap=.15;    % 10%
ProbLargeMut=.05; % x100=%
r=rand; c=a;
if r<ProbLargeMut % Change whole chromosome
   i=round(rand*size(a,1))+1;
   % c(i,1)=floor(rand*3+32);     % COMMAND CHANGE
   c(i,2)=floor(rand*255);       % POSITION CHANGE
   c(i,3)=round(rand*9)*100;      % TIME CHANGE
   if r<ProbLargeMut/4
      if size(a,1)>3
         c=[ a(1:i-1,:); a(i+1:size(a,1),:)];   % CROP LINE i
      end
   end
elseif r<ProbSwap
   i=floor(rand*(size(a,1)-1))+1;
   temp=c(i,:);
   c(i,:)=c(i+1,:);
   c(i+1,:)=temp;
elseif r<ProbSamllMut % Small mutation
   i=floor(rand*size(a,1))+1;
   if r<.60
      while (c(i,2)==a(i,2))
         c(i,2)=c(i,2)+round(rand*4-2);
      end
   elseif r>=.60
      while (c(i,3)==a(i,3))
         c(i,3)=((c(i,3)/100)+round(rand*2-1))*100;
      end
   end
end

c=clamp(c);
```

Performs crossover

```
%% Crossover operator
function [c d]=crossover(a,b) % Crossover matrix arguments a and b into
c
sa=size(a,1); % size a
sb=size(b,1); % size b
pa=floor(rand*sa)+1;   % insert point
pb=floor(rand*sb)+1;   % extraction point
c=a;
d=b;
if pb<sa, c(pa,1:3)=b(pb,1:3);
end
if pa<sb, d(pb,1:3)=a(pa,1:3);

end
```

Create a random matrix within allowed boundaries

```
%% Create from scratch
function c=create()  % Create testmatrix c
lines=floor(rand*11+3);
for i=1:lines
    c(i,1)=floor(rand*3+32);
    c(i,2)=floor(rand*255);
    c(i,3)=round(rand*9)*100;
end

c=clamp(c);
```

Tests the command matrix code for consistency

```
%% Clamp values between allowed boundaries
function c=clamp(a)
min2=80;    max2=182; % 32, motor 0 (Left, Right bounds)
min3=100;   max3=140; % 33, motor 1 (Up, Down bounds)
min4=1;     max4=250; % 34, motor 2 (Theta bounds)
for i=1:size(a,1)
    % clamp time
    if a(i,3)<0, a(i,3)=0; end
```

aa

```
    if a(i,3)>900, a(i,3)=900; end
    % clamp motors
    if (a(i,1)<32), a(i,1)=32; end
    if (a(i,1)>34), a(i,1)=34; end
    if (a(i,1)==32),
        if a(i,2)<min2, a(i,2)=min2; end;
        if a(i,2)>max2, a(i,2)=max2; end;
    end
    if (a(i,1)==33),
        if a(i,2)<min3, a(i,2)=min3; end;
        if a(i,2)>max3, a(i,2)=max3; end;
    end
    if (a(i,1)==34),
        if a(i,2)<min4, a(i,2)=min4; end;
        if a(i,2)>max4, a(i,2)=max4; end;
    end
end

c=a;
```

Generates the hash code for one solution

```
function h=hash(mat)

% h=hast(mat)
%   h=solution of matrix mat

if nargin<1,
    mat=[ 32 hex2dec('50') 300;
          34 hex2dec('40') 200;
          32 hex2dec('B6') 300;
          34 hex2dec('C0') 200 ]; % 350024023B602C02
end;
name='';
for i=1:size(mat,1)
    pos=dec2hex(mat(i,2));
    if size(pos,2)<2 pos=['0' pos]; end
    name=[name num2str(mat(i,3)/100) pos dec2hex(mat(i,1)-32)];
    % delay position motor
end

h=name;
```

This retrieves the command matrix from a previously generated hash code.

```
function mat=hmat(name)

% h=hast(mat)
%   h=solution of matrix mat

if nargin<1,
    name='350024023B602C02';
end;

s=size(name,2);
c=s/4;
mat=zeros(c,3);
for i=1:c
    mat(i,:)=[str2num(name(4))+32 hex2dec(name(2:3))
str2num(name(1))*100];
    if i<c name=name(5:s); end
    s=s-4;

end
```

Use this function to generate a video of the simulator's swinging Ro.

```
function vid(mat,mode,time,avifilename,framerate,discard)

% vis(hsh,mode,time,framerate)
%  Generate a video of the simulation output
%  Use the matrix, Graph mode, Simulated time, AVI file-name,
%  AVI frame rate and how many frames to discard before start
%  as arguments
tic
```

```
if nargin<2,
   mode=1;
end
if nargin<3,
   time=5000;
end
if nargin<4,
   aviout=1;
   avifilename='video';
else
   aviout=1;
end
if nargin<5,
   framerate=5;
end
%% Simulation variables initialization
line=1;
wait=0;
data=[0 0];

[lines cols]=size(mat);
pos=[127 127 127];              % Initial position
dest=[127 127 127];             % Initial destination
step=[.333 .25 .6];             % Step in 1/1000 s
direction=[0 0 0];              % Motor direction

%% Evaluation variables initialization
ft=[0 0 0];                     % Total force applied

aph=(pos(1)-127)/89.141;        % Alpha angle
bt=-(pos(2)-92)/57.3;           % Beta angle
th=(pos(3)-127)/84.667;         % Theta angle

X=sin(th)*cos(aph)+cos(th)*sin(bt)*sin(aph);
Y=sin(th)*sin(aph)-cos(th)*sin(bt)*cos(aph);
Z=cos(th)*cos(bt);
```

```
RX=-cos(bt)*sin(aph);
RY=cos(bt)*cos(aph);
RZ=sin(bt);

normal=[X Y Z]; % Normal vector
ref=[RX RY RZ]; % Reference vector

%% Graph nitiation

fig = figure(1);
set(fig,...
  'Color',[1 1 1],...
  'InvertHardcopy','off',...
  'PaperUnits','points',...
  'PaperPosition',[1 311 640 480],...
  'PaperSize',[640 480],...
  'PaperType','<custom>');
clf;
if mode==0
    title('Motor positioning');
    set(gca,'Drawmode','Fast');
    axis([0 time 0 260]); hold on; box;
else
    title('Vectors');
    if aviout,
        set(fig,'DoubleBuffer','on');
    end

    set(gca,'Drawmode','Fast');
    plot3([-1 1 1 -1 -1],[1 1 -1 -1 1],[0 0 0 0 0],'-b','Erasemode','none');
    axis([-1 1 -1 1 -1 1]);hold on; box; grid on;
    p1=plot3([0 0 0 0 0 0],[0 0 0 0 0 0],[0 0 0 0 0
0],'-k','Erasemode','normal');
    p2=plot3([1 0 0],[0 1 0],[0 1 0],'-r','Erasemode','normal');
    p3=plot3([0 1 0],[0 1 0],[0 1 0],'-g','Erasemode','normal');
    drawnow;
end
%% Main Loop
user_entry = input('Press any key when ready:')
if aviout,
    aviobj = avifile(avifilename,'fps',25);
end
framecount=0;
discard=1500; % time to stabilize
for t=1:time

    %% Enter command
    if wait<1
        motor=mat(line,1)-31;
        newpos=mat(line,2);
        wait=mat(line,3);
        dest(motor)=newpos;
        line=line+1;
        if line>lines line=1;
        end
    end
    direction=((dest-1)>pos)-((dest+1)<pos);
    pos=pos+(step.*direction);
    wait=wait-1;

    %% Simulation process
    aph=(pos(1)-127)/89.141;
    bt=-(pos(2)-92)/57.3;
    th=(pos(3)-127)/84.667;

    X=sin(th)*cos(aph)+cos(th)*sin(bt)*sin(aph);
    Y=sin(th)*sin(aph)-cos(th)*sin(bt)*cos(aph);
    Z=cos(th)*cos(bt);

    RX=-cos(bt)*sin(aph);
    RY=cos(bt)*cos(aph);
```

```
RZ=sin(bt);

aref=ref;
normal=[X Y Z];
n=-normal;
ref=[RX RY RZ];
attack=aref-ref;
impulse=(dot(attack,n)/dot(n,n))*n;
ft=ft+impulse;

% projection (dot(a,b)/dot(b,b))*b
% [sqrt(X^2+Y^2+Z^2) X Y Z aph bt th]
if mode==0   %% Graph 2D
   plot(t,pos(1),'r-','Markersize',3,'Erasemode','none');
   plot(t,pos(2),'y-','Markersize',3,'Erasemode','none');
   plot(t,pos(3),'b-','Markersize',3,'Erasemode','none');
else   %% Gpaph 3D
   ap=attack*100;
   g=impulse*100;
   set(p1,'Xdata',[0 -Z/16 RX Z/16 0 X/4]);
   set(p1,'Ydata',[0 X/16 RY -X/16 0 Y/4]);
   set(p1,'Zdata',[0 X/16 RZ -X/16 0 Z/4]);

   set(p2,'Xdata',[0 ap(1) 0]);
   set(p2,'Ydata',[0 ap(2) 0]);
   set(p2,'Zdata',[0 ap(3) 0]);

   set(p3,'Xdata',[0 g(1) 0]);
   set(p3,'Ydata',[0 g(2) 0]);
   set(p3,'Zdata',[0 g(3) 0]);
end
drawnow;
framecount=framecount+1;
data=[data;t impulse(2)];
if framecount>framerate
   if aviout,
          if discard>0,
             discard=discard-framerate;
          else
             frame = getframe(gca);
             aviobj = addframe(aviobj,frame);
          end
       end
       framecount=0;
    end
end
if aviout,
   aviobj = close(aviobj);
end
ft
f=sqrt(ft(1)^2+ft(2)^2+ft(3)^2)
%% Graph offset

%legend('Motor 0','Motor 1','Motor 2',3);
```

Serial control application

This code in VB.net is the serial control application, used to interface the hash commands generated by the MAS to the robotic actuators with correct timing.

```
Imports System.ComponentModel
Imports System.Threading

Public Class FormControl

   Private backgroundController As
System.ComponentModel.BackgroundWorker
   Delegate Sub SetTextCallback(ByVal [text] As String)
```

```
    Public Sub New()
        ' This call is required by the Windows Form Designer.
        InitializeComponent()
        backgroundController = New BackgroundWorker()
        backgroundController.WorkerReportsProgress = False
        backgroundController.WorkerSupportsCancellation = True
        AddHandler backgroundController.DoWork, New
DoWorkEventHandler(AddressOf backgroundController_DoWork)
        AddHandler backgroundController.RunWorkerCompleted, New
RunWorkerCompletedEventHandler(AddressOf
backgroundController_RunWorkerCompleted)

        updateStatus(String.Empty)

    End Sub

    ' Report an error
    Private Sub reportError(ByVal e As Exception)
        updateStatus("Error!")
        MessageBox.Show("The following error occurred: " +
ControlChars.CrLf + e.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
    End Sub


    Private Sub reportError(ByVal message As String)
        updateStatus("Error!")
        MessageBox.Show("The following error occurred: " +
ControlChars.CrLf + message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error)
    End Sub

    Public Sub updateStatus(ByVal status As String)
        SetText(status)
    End Sub
    Private Sub SetText(ByVal msg As String)
```

```
        ' InvokeRequired required compares the thread ID of the
        ' calling thread to the thread ID of the creating thread.
        ' If these threads are different, it returns true.
        If Me.TexOut.InvokeRequired Then
            Dim d As New SetTextCallback(AddressOf SetText)
            Me.Invoke(d, New Object() {msg})
        Else
            Me.TexOut.AppendText(Chr(13) + Chr(10) & msg)
            If Me.TexOut.Text.Length > 5000 Then
                Me.TexOut.Text = Me.TexOut.Text.Substring(1000,
Me.TexOut.Text.Length - 1000)
            End If
        End If
    End Sub

    ' This executes in a separate thread
    Private Function sendControl(ByVal start As String, ByVal worker As
BackgroundWorker, ByVal e As DoWorkEventArgs) As Integer
        ' Open serial port
        Using com1 As IO.Ports.SerialPort = _
                My.Computer.Ports.OpenSerialPort("COM1")
            com1.BaudRate = 9600
            com1.DataBits = 8
            com1.Parity = IO.Ports.Parity.None
            com1.StopBits = IO.Ports.StopBits.One
            com1.DtrEnable = True
            Dim oEncoder As New System.Text.ASCIIEncoding
            Dim oEnc As System.Text.Encoding =
System.Text.ASCIIEncoding.GetEncoding(1252)
            com1.Encoding = System.Text.ASCIIEncoding.GetEncoding(1250)
            Me.SetText("COM1: (OK) " + com1.BaudRate.ToString + "  " +
com1.DataBits.ToString + "-" + com1.Parity.ToString + "-" +
com1.StopBits.ToString + " Open=" + com1.IsOpen.ToString)

            ' Prepare data
            Dim i As Integer = 0
```

```
          Dim header As Byte = &HFF
          Dim cmd As Byte = &H21
          Dim pos As Byte = &H8A
          Dim chk As Byte = &H2B
          Dim t As Integer = 0
          Dim Buffer1 As Byte() = {header, cmd, pos, chk}
          Me.SetText("Positioning motor.")
          com1.Write("SerialControl initialized, Positioning:")
          com1.Write(Buffer1, 0, 4)
          Me.SetText(header.ToString + "    " + cmd.ToString + "    " +
pos.ToString + "    " + chk.ToString + ", wait " + t.ToString + "ms")
          Me.SetText("---------------")
          While True
              ' Check for cancellation
              If worker.CancellationPending = True Then
                  e.Cancel = True
                  Exit While
              Else
                  ' Routine here
                  i = 0
                  While i + 4 <= start.Length
                      Integer.TryParse(start.Substring(i, 1).ToString,
t)

                      t = t * 100
                      i += 1
                      pos = CByte("&h" + start.Substring(i, 2).ToString)
                      i += 2
                      cmd = 32 + CByte(start.Substring(i, 1).ToString)
                      i += 1
                      If pos = 255 Then
                          Exit While
                      End If
                      chk = (cmd Xor pos) And 127
                      ' send data trought serial port 1
                      Dim Buffer2 As Byte() = {header, cmd, pos, chk}
                      Me.SetText(header.ToString + "    " + cmd.ToString +
"    " + pos.ToString + "    " + chk.ToString + ", wait " + t.ToString +
"ms")
                    com1.Write(Buffer2, 0, 4)
                    ' Wait for next command
                    System.Threading.Thread.Sleep(t)
                 End While
                 Me.SetText("---------------")
              End If
              If pos = 255 Then
                 Exit While
              End If
           End While
           ' MessageBox.Show(pos.ToString, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Information)
           ' Close serial port
           com1.Close()
        End Using
        Return 1
    End Function

    ' Thread start/finish
    Sub backgroundController_DoWork(ByVal sender As Object, ByVal e As
DoWorkEventArgs)
        Dim start As String = CStr(e.Argument).Trim
        e.Result = sendControl(start, CType(sender, BackgroundWorker), e)
    End Sub

    Sub backgroundController_RunWorkerCompleted(ByVal sender As Object,
ByVal e As RunWorkerCompletedEventArgs)
        If e.Cancelled Then
           updateStatus("Cancelled.")
        ElseIf e.Error IsNot Nothing Then
           reportError(e.Error)
        Else
           updateStatus("Done!")
        End If
        enableControls()
    End Sub

    ' Disable/re-enable the controls
    Private Sub diseblecontrols()
        disableAccess()
        But1.Enabled = False
        But2.Enabled = False
        But3.Enabled = False
        But4.Enabled = False
        But5.Enabled = False
        But6.Enabled = False
        But7.Enabled = False
        But8.Enabled = False
        But9.Enabled = False
        ButtonSave.Enabled = False
        ButtonLoad.Enabled = False

    End Sub
    Private Sub enableControls()
        But1.Enabled = True
        But2.Enabled = True
        But3.Enabled = True
        But4.Enabled = True
        But5.Enabled = True
        But6.Enabled = True
        But7.Enabled = True
        But8.Enabled = True
        But9.Enabled = True
        ButtonSave.Enabled = True
        ButtonLoad.Enabled = True
    End Sub
    Private Sub kick(ByVal start As String, ByVal name As String)
        If start = String.Empty Then
            reportError("No control string defined!")
        Else
```

```
            diseblecontrols()
            updateStatus("Sending '" + name + "' data...")
            backgroundController.RunWorkerAsync(start)
        End If
    End Sub

    ' Buttons functionality
    Private Sub But1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But1.Click
        kick(TextBox1.Text, Name1.Text)
    End Sub
    Private Sub But2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But2.Click
        kick(TextBox2.Text, Name2.Text)
    End Sub
    Private Sub But3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But3.Click
        kick(TextBox3.Text, Name3.Text)
    End Sub
    Private Sub But4_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But4.Click
        kick(TextBox4.Text, Name4.Text)
    End Sub
    Private Sub But5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But5.Click
        kick(TextBox5.Text, Name5.Text)
    End Sub
    Private Sub But6_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But6.Click
        kick(TextBox6.Text, Name6.Text)
    End Sub
    Private Sub But7_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But7.Click
        kick(TextBox7.Text, Name7.Text)
    End Sub
    Private Sub But8_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But8.Click
        kick(TextBox8.Text, Name8.Text)
    End Sub
    Private Sub But9_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles But9.Click
        kick(TextBox9.Text, Name9.Text)
    End Sub
    ' Save and load Config (not implemented)
    Private Sub ButtonSave_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ButtonSave.Click
        disableAccess()
    End Sub
    Private Sub disableAccess()
        ComboBox1.Enabled = False
        ComboBox2.Enabled = False
        ComboBox3.Enabled = False
        ComboBox4.Enabled = False
        CheckBox1.Enabled = False
        TextBox1.Enabled = False
        TextBox2.Enabled = False
        TextBox3.Enabled = False
        TextBox4.Enabled = False
        TextBox5.Enabled = False
        TextBox6.Enabled = False
        TextBox7.Enabled = False
        TextBox8.Enabled = False
        TextBox9.Enabled = False
        Name1.Enabled = False
        Name2.Enabled = False
        Name3.Enabled = False
        Name4.Enabled = False
        Name5.Enabled = False
        Name6.Enabled = False
        Name7.Enabled = False
        Name8.Enabled = False
        Name9.Enabled = False
        TexOut.Enabled = False
        updateStatus("Controls locked. Esc or keypad 0 to stop.")
        updateStatus("--------------------")
    End Sub

    Private Sub ButtonLoad_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ButtonLoad.Click
        ComboBox1.Enabled = True
        ComboBox2.Enabled = True
        ComboBox3.Enabled = True
        ComboBox4.Enabled = True
        CheckBox1.Enabled = True
        TextBox1.Enabled = True
        TextBox2.Enabled = True
        TextBox3.Enabled = True
        TextBox4.Enabled = True
        TextBox5.Enabled = True
        TextBox6.Enabled = True
        TextBox7.Enabled = True
        TextBox8.Enabled = True
        TextBox9.Enabled = True
        Name1.Enabled = True
        Name2.Enabled = True
        Name3.Enabled = True
        Name4.Enabled = True
        Name5.Enabled = True
        Name6.Enabled = True
        Name7.Enabled = True
        Name8.Enabled = True
        Name9.Enabled = True
        TexOut.Enabled = True
        updateStatus("--------------------")
        updateStatus("Controls Unlocked.")
    End Sub

    Private Sub ButtonCancel_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles ButtonCancel.Click
        If backgroundController.IsBusy Then
            updateStatus("Interrupting...")
            backgroundController.CancelAsync()
        End If
        updateStatus("Idle.")
    End Sub

End Class
```