Agent-Oriented Evolutionary Control Systems
– Determining the Optimal Control Codes for "Ro", the Oriental Oar –

Rogério NEVES* and Tsugukiyo HIRAYAMA**

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. Yet, this procedure generates human-oriented solutions, causing the defined controller to simulate human conducted control, rather than generate optimized, machine-oriented signals specific for the electro-mechanical system. Furthermore, many control systems may involve a huge number of inter-dependent variables, incompatible with the familiar four-dimension coordinate visualization, rendering the system incomprehensible for human abstraction and therefore unsuitable for conventional treatment. We present here an alternative, machine-oriented and autonomous control system training model, based on natural evolution, Multi-Agent Systems and Distributed Computing, which is able to profit on modern computer architectures and deal with Evolutionary Search issues in a reduced timeframe. Here we describe the method in practice, as we apply it to solve the "Ro" control problem, a simple one-oar robotic rowing system that allow us to compare the autonomous discovered method to the traditional, human conducted rowing method.

Keywords*: Ro, autonomous training, control, Multi-Agent Systems, Evolutionary Search, Genetic Algorithms*

1. Introduction

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, and its operation is simple and intuitive. On the other hand, reproducing this operation with robotic actuators is not commonplace, requiring an operator to describe the process in a comprehensible fashion to a programmer able to implement it into control software. In the traditional method, the programmer adjusts the set of control codes interactively, until the desired result is obtained, roughly matching the described procedure. As it just mimics the operator, the approach produces human-oriented signals, rather than optimal results specifically designed for the hardware in consideration.

To solve the Ro control problem, we propose the use of an autonomous training method, based on Genetic Algorithms (GA) [(1)], Multi-Agent Systems (MAS) [(2)] and computer simulation in a distributed framework [(3)], able to find optimal, machine-oriented solutions in reduced times, with additional advantages: It scans simultaneously several regions of the multi-variable space, spotting a variety of solutions; It can optimize the system by many criteria for different benefits, e.g. speed, acceleration, fuel consumption, friction, etc. The method is ideal to address problems for which the best solution is unknown and that often challenges human abstraction, such as ones involving multiple dimensions and coordinate systems as well as dynamical, inter-dependent variables.

The current problem is one example of what can be dealt by this approach, allowing the comparison of the autonomous and humanistic approaches. Once proven ship-worthy, the same guideline can be applied to solve more complex problems (e.g. navigation control, collision avoidance, and control systems for docking).

* Student Member: Faculty of Engineering Yokohama National University, 79-5 Tokiwadai, Hodogaya-ku, Yokohama 240-8501, neves@vento.shp.ynu.ac.jp

** Member: Faculty of Engineering of Yokohama National University, 79-5 Tokiwadai, Hodogaya-ku, Yokohama 240-8501, hirayama@ynu.ac.jp

1.1. Previous works

The concept of Evolutionary Search is quickly spreading throughout different areas of science, as the computer industry offers increasing computer power with decreasing costs, evolutionary approaches can be put in use for a wider range of problems [4].

Innumerous researches are in progress in order to use newly available multi-processed systems to deliver hi-quality solutions in reasonable times, Parallel Genetic Algorithms or PGA [5] is among them. Nevertheless the efforts have given many working solutions, faster and cheaper alternatives to PGA are yet being proposed.

1.2. Motivation

Computer systems with a large number of processors have high costs and short life spans. For problems requiring complex computations, such as the hydrodynamic simulation employed here, parallel implementations become mandatory. As alternative to expensive multi-processed systems, ordinary networks of cheap computers are widely available and often present in most research environments. The main motivation for this project was to be able to use these networks to improve our search capabilities.

2. Problem outline

The specification and build of an efficient control system can be separated in two parts: Hardware (the physical model, including actuators and mechanical parts) and software (control program, data to activate the actuators). The building of optimal hardware can be aided by evolutionary design, as described by Bentley [4], but such problem will not be addressed here. In the current research, hardware exists for which optimal control codes need to be defined. The referred codes represent signals to perform certain operation. In this paper, a new approach to define the referred signals autonomously is explained. The method provides several working results (sub-optimal solutions) in short term, in addition to the optimal control code for the system in long term. In this section the basic concepts of each discipline used to solve the problem are presented.

2.1. Working with Genetic Algorithms

GA has its roots on Darwin's theory of natural selection. Even though their mechanics are simple, they are complex non-linear algorithms controlled by many parameters, such as population, crossover and mutation rates, distribution trough space and selection. Default values are not defined, changing for each particular problem. Nevertheless, good practices are introduced by Cantú-Paz [5], especially when dealing with parallel implementations of GA.

In GA theory, DNA refers to the data structure [6], common to each and every individual in the gene-pool (collection of all individuals) describing its particular characteristics. In this project, DNA is a string of bytes representing the relevant aspects of the problem. The most popular mechanisms used to improve the gene-pool are employed: Mutation and Crossover. The gene-pool is started randomly, once boosting the process by adding human defined solutions is known to cause addictions into local optimums, as pointed by Mitchell [1]. In every generation, a small percentage of new individuals statistically improve, favoring the best according to selected fitness introduces an artificial-selection, similar to Darwin's law.

2.2. The MAS approach

In the theory of Electronic Multi-Agents, agents are small pieces of code and each one is imbued with one or more objectives. Concepts such as autonomy, mobility and cooperation are applied, as exemplified by Xie [2]. They have a set of rules and procedures, called methods, which performs its communications and other functionality, in order to accomplish the specified objectives. MAS offer an elegant and efficient approach to handle parallelization. The PGA theory, as described by Cantú-Paz [5], can be extended to an alternative MAS approach, where the load is dynamically spread to several computers.

In the actual implementation, to each agent is given a population of solutions with a random and localized gene-pool, as well as rules, derived from GA, to operate individuals. The agents have mobility, meaning they can migrate to other computers running MAS, trough the network or even the internet.

2.3. Divide and conquer strategy

Employing MAS has the advantage of dividing the population into small localized groups, therefore increasing the search resolution as well as spreading the load dynamically, as agents are able to migrate to new available clients, and uniformly on local and remote processors, resulting in direct speed-ups.

The granularity is used to balance performance, i.e.,

higher granularities are supposedly better, as a larger number of agents spread the load more uniformly and a bigger population per agent increases the local search resolution, but excessive numbers in any of these will increase load and reduce system performance. On the opposite, small populations may not produce significant improvements per generation.

Networked lab computers are used when idle. To enable a client to run MAS, an executable is deployed and set as screensaver. It remotely initiates agents and evaluates communication between server (resource holder) and clients. In the client-server structure, the server must be operational at all times for agents to function, but clients may break operation without compromising the overall integrity.

The communication relies on the file-system, once all agents have the same code and communicate trough global variables stored in the server, special care is taken to avoid simultaneous accesses to files. Signaling procedures were adopted to ensure only one agent has writing access to a file at once [7].

The partitioning is done by equally dividing all dimensions according to the expected granularity. Each agent receives a segment of the space to populate uniformly. Partitioning, granularity, signaling and communication, as well as other important concepts in Parallel Programming, were based on the guidelines provided by Foster [3].

3. Implementation procedures

The procedures were performed simultaneously, with hardware and software being constantly modified. Nevertheless, the project can be grouped in: the model (hardware); code representation, simulation, agents, database and parameter definition (software).

3.1. Physical model

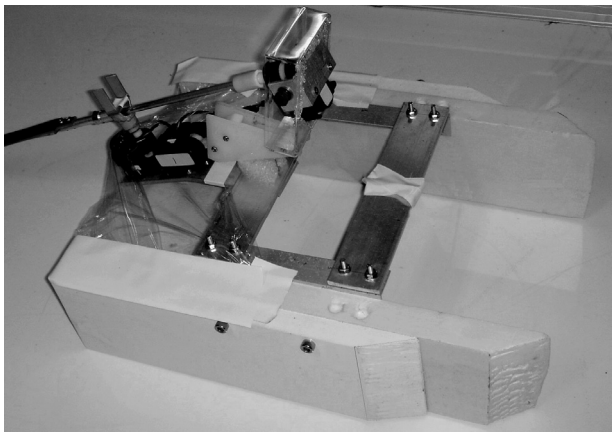

Fig.1 Ro-bot mounted in a Catamaran-style ship.

For the application, a robotic model, fondly named "Ro-bot", was built. The robotic arm consists of three actuators disposed perpendicularly for an easy translation into a three-dimension coordinate system. Ro pitch, yaw and rotation can be operated individually by sending codes to each actuator. Fig.1 shows Ro-bot incorporated into a catamaran style ship, Fig.2 schematizes the Ro-bot operation.

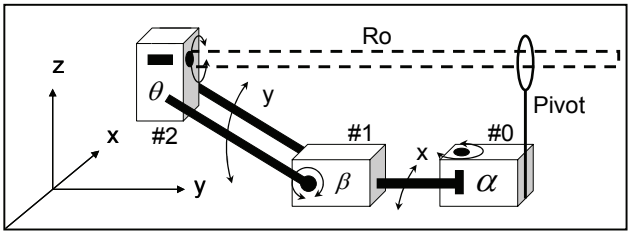

Fig.2 Schematic diagram of the rowing mechanism with the actual coordinate system adopted.

3.2. Code representation

To determine the DNA parameters, all the variables that affect the system must be identified. The actuators operate by receiving a 4-byte code trough a serial interface, composed by: Header, Command, Position and Checksum. The header is a constant and Checksum is a calculated byte, both are added later by the control software itself. The command byte contains bits specifying the target actuator address and operation; however it is irrelevant for the GA. Position has operating ranges that must be respected for each actuator, considering there is a risk of damaging the mechanism with improper codes. The DNA then consists of a series of sequential commands divided into chromosomes of three bytes, named respectively CMD (command), POS (position) and DLY (delay or time in milliseconds) to provide timing. The sequence repeats in a loop, generating a unique signal that positions the Ro synchronously.

Organizing the byte array in the DNA as a matrix makes it easier to understand, as suggested by Michalewicz [6]. Fig.3 shows the matrix and resulting positioning signals of a DNA based on the traditional human control.

Another representation was needed in order to manage database filenames and 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 code for each individual.

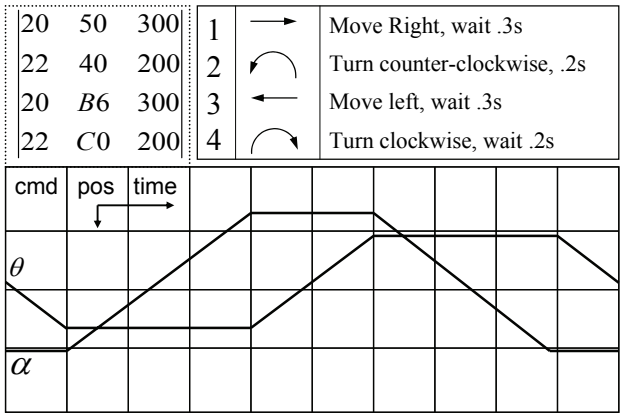

Fig.3 Positioning signals for Ro generated by the execution of the DNA codes represented in the matrix, the chromosomes correspond to the commands listed in the box.

3.3. Simulation modeling and calibration

A simulation of the Ro-bot arm was implemented to test the solutions. The DNA code is passed to a function that translates it into positioning signals, to orient the virtual Ro and simulate the hydrodynamics around its blade, evaluating the amount of thrust produced by one movement in a short period. The method returns a qualitative numeric vector usable by fitness functions for classification.

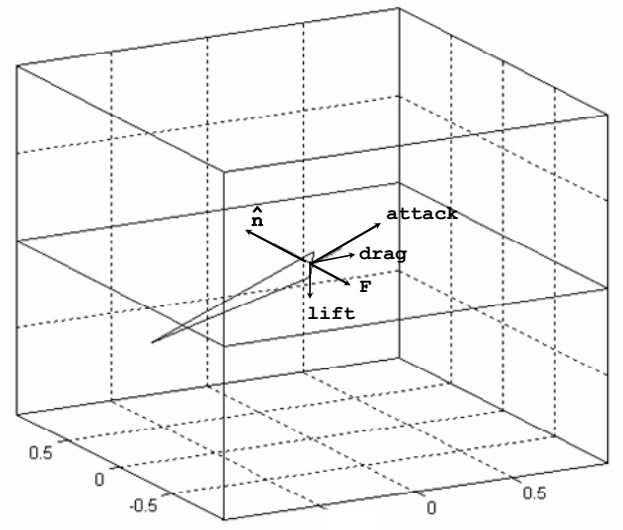

Fig.4 Wire-frame of the simulation shows the virtual Ro and resulting vectors calculated from the instant movement: normal, attack, lift, drag and total impulse (F).

The simulation had to be fine tuned to match physics and timing of the actual model in order to produce useful results. Fig.4 shows the simulation view port. The view port is disabled during MAS computations to reduce load.

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

- Steady state: fluid speed remains unchanged
- Ro-bot is fixed at the referential
- Flat blade: The Ro shape was simplified to a flat blade, ignoring the camber effects
- Infinitesimal extrapolation: An infinitesimal surface area of the Ro is used for calculation

The approximations allow faster calculations for MAS experiments and benchmarking, however they introduce discrepancies from the real model. These discrepancies, nevertheless, affect uniformly all the tested solutions, not compromising the overall comparison between solutions.

A simplified discrete version of the hydrodynamic force on blade 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:

$$\vec{F}t = \sum_0^T \vec{F} \qquad (1)$$

The function returns a three dimensional vector containing the strength and direction of the average total force resulting from the full operation on T. The fitness is calculated by the dot product (projection), using a unitary vector (*d*) in the evaluated direction:

$$Fitness = \sum_0^T \vec{F} \cdot \hat{d} = \vec{F}t \cdot \hat{d} \qquad (2)$$

Note that the simulation returns a unique vector (1) for each point in space represented by the DNA, for one same evaluation period T. These values can be reused even if the fitness function (2) is changed. The fitness may change according to the artificial selection criteria and direction.

Calibrating the simulation to match the real physics in the model is the most important and complicated 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 operating conditions. Fig.5 shows the sensor configuration assembled to measure the actual model response and attributes. The sensors were used to determine the amount of impulse generated by operating Ro inside a water basin. The signals recorded were later compared to the generated by the simulation, to verify consistency. Fig.6 shows the comparison between measured and simulated impulses.

Fig.5 Sensor assembly used for calibration.

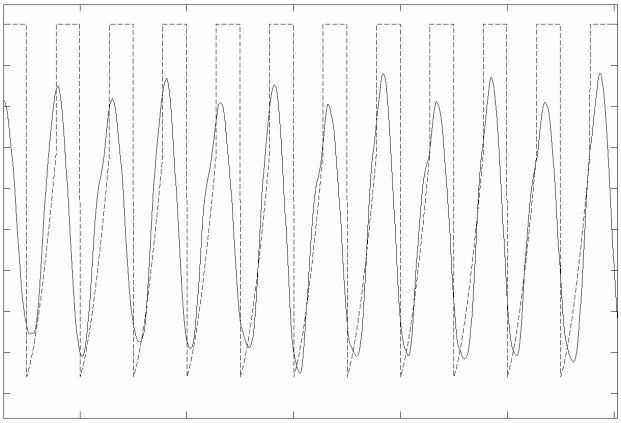

Fig.6 Filtered data obtained from sensor readings (solid) and the data generated in the simulation (dashed), matching can be observed in the overlapped images.

Once calibrated, 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.

3.4. Agents

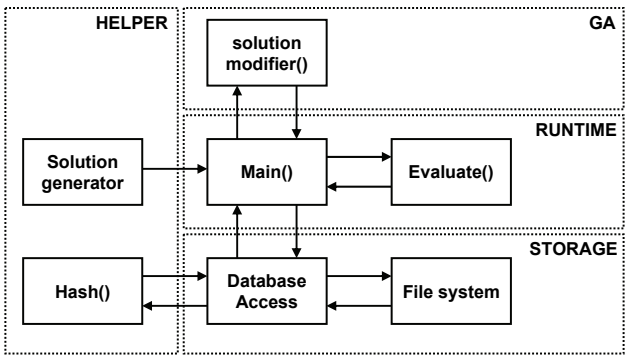

Fig.7 Diagram of main groups and methods for one agent, identifying the respective links.

The agents here are instances of one same program (other examples of implementation can be found in [8]). They execute simultaneously within the server and through the network, in all clients running MAS. Fig.7 summarizes the main groups and respective modules (or methods) involved in agents operation, as well as its links.

Setting the local population to 1000 individuals, the following pseudo-code represents the operations performed by one single agent:

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

Upon start, the 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 in the beginning, but individuals eventually travel to neighboring partitions over time.

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 evaluation function.

An agent may use an arbitrary number of methods to try in increasing its solutions fitness, not restricted to GA. Here only Crossover and Mutation are used, but the implementation of additional methods, such as Neural Nets, Simulated Annealing and gradient climbing, can benefit the convergence speed.

The simulation is the most time-consuming and requested method, in parallel computing terms, the bottleneck. Every agent calls to it a thousand times per

cycle. Several procedures to reduce simulation calls were implemented, such preliminary tests, lookups in the local and server database and communications to check if another agent is searching the neighborhood. By this context, agent communications aim to avoid redundant searches in the same region of the space. Once individuals often revisit recent evaluated points, it is important to keep track of recently visited points, by adding some specific fields in the DNA, preventing the GA from coming back to recent values. Another way to reduce the call to the fitness function is by implementing a solution database.

3.5. The database

One of the new proposals to avoid redundant evaluations is, in addition to each agent to keep on track of the recently visited points, implementing a database of evaluations. The database requires a large amount of storage memory, and is not suitable for all problems. It associates each evaluation result to a unique hash, representing one individual point. When values return to tested points, the stored result is used instead of performing a new evaluation. This method is limited by disk, meaning hi-resolution spaces may exceed the physical disk-storage capacity of conventional systems. It also requires fast search capabilities. Recent improvements in the file-system finally allow using this resource to save processing time. The vectors are stored into the server's file-system, in a folder hierarchy designed to provide faster searches. Making the folder accessible in the network, all agents on server and clients can access its contents.

A unique hash code to be used as filename is obtained by concatenating the hexadecimal DNA array. Hash uniqueness is required. Compression algorithms can be also employed for this purpose. The evaluation is the only data recorded into it. To accelerate the search, the folder structure is determined by the first six values in the DNA. The database also keeps a record of the best solutions with date and time in a different disk, allowing an analysis of the progress history. The server folder organization is as follows:

DNA: *AF 0A 1C 83 B7 C8 F9 1B 8E EA BC FF FA*
FILE: *server**tests**af**0a**1c**83**bf**c8**af0a1c83*...
Best solutions: *server**best*\...
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 when clients go offline. The agents executing in the clients have global variables targeting the server's address on the network, where the global variables and the solution database are stored.

This approach creates an intricate folder structure and a large number of files, often requiring disks to be formatted after being used as test database. Modern LAN drives are cheap and adequate for this task, the disk-accesses are of an order of a few milliseconds, saving considerable time when compared to a seconds-long wait required to perform one simulation.

3.6. Selected parameters for MAS and GA

The number of agents and local populations were scaled according to individual computational power. Selecting the best values for agent numbers and respective populations came from trial and error. For the current topology used for numerical experiments, a dual-core server connected to four single-core clients, the following best working parameters were achieved by successive adjustment. Different networks or problems may present better results with other settings. Theoretical extrapolations, as suggested by Cantú-Paz [5] can be used to guess starting points.

The number of agents was usually NP+1 (Number of Processors in the system plus one) to 5 times NP, depending on each computer performance, one single agent in shared systems, systems running critical processes and elder computers. Population per agent: 1000; Population variance (space partitioning): 20% for each chromosome. Task priority was set to high on the server and to idle on client computers.

The probability (P) for each mutation factor is: 0-1% rate, P=90%; 1-10% rate, P=9%; 10-100% rate, P=1%. Crossover was performed for 25 random pairs among the best 100, producing 2 children per couple.

For every generation, the artificial selection rules are: Elite (top 5 highest ranked individuals) receive a time-to-live of 5 generations; 100 worst are discarded.

Stop rules: Stop if MAX is unchanged for 3000 generations; Stop if no new points tested for 1000 generations; After stop, randomize population in next partition and restart.

4. Results

For the experiments, we set the fitness function to find maximum thrust in several directions (0, 45, 90, 180, 225, 270, 315 degrees). Executing the program in different scenarios, we analyze the results by three different aspects: computation time, simulated results

and water-borne verification.

4.1. Parallel computation analysis

In early single threaded numerical experiments, sub-optimal solutions were usually obtained after some hours of computation. The optimal control for each problem was found usually after 12-48 hours of computation. In many occasions the GA converged into a sub-optimal region of the space, never reaching out to the global optimum. To confirm the optimal values it was required to reset the GA several times. Further executions required less time, once they profit on database stored evaluations.

The later agent-based experiments (multi-threaded) demonstrated why "two heads are better than one". Initial sub-optimal solutions were found at a record of 15 min and optimal values form 2 hours. Concurrent populations not only decreased computation time, but the probability of getting stuck in local maximums was negligible for higher number of agents.

Analyzing the average time required for calculation, we obtained a super-linear speedup, less than a fraction of time for single-threaded models. The boost is a consequence of the random nature of GA, where improvements depends only on the previous progress achieved. Fig.8, Fig.9 and Fig.10 compare the single-threaded model, multi-threaded model and the distributed multi-agent models.

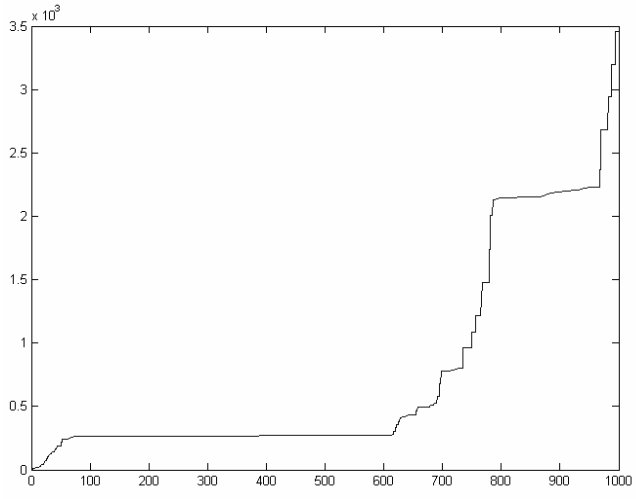

Fig.8 Progress of maximum fitness over the first 1000 generations for the single threaded GA version; maximum fitness after 1000 generation was 3400. The simulation continues and the total evaluation time was 26 hours.

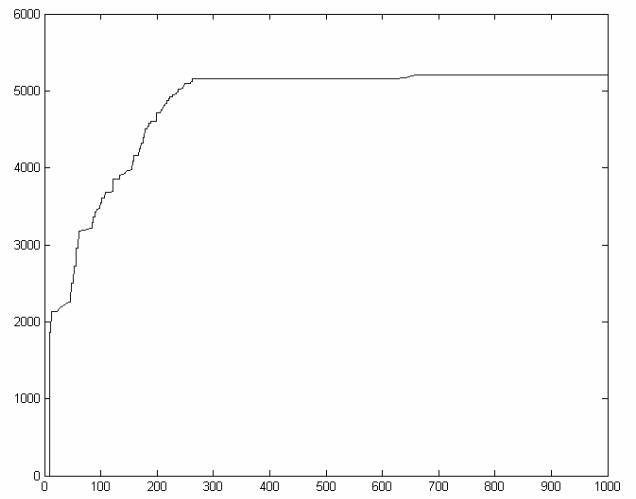

Fig.9 Progress of maximum fitness over the first 1000 generations in the multi-threaded MAS version with 5 agents and 2 processors; maximum fitness after 1000 generations was 5200 and total evaluation time 8 hours.

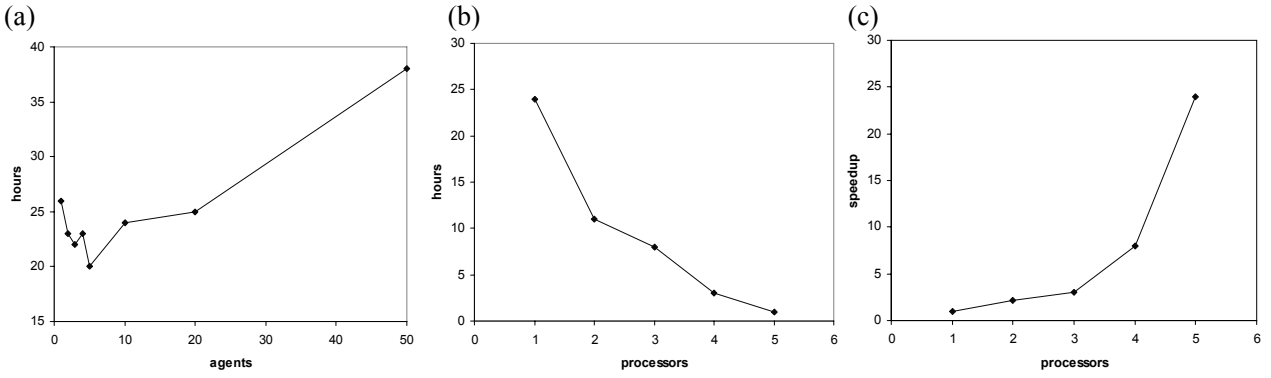

Fig.10 Average time required to achieve a near maximum fitness (10% tolerance): (a) Evaluation time vs. total number of agents for one processor (Number of Processors NP=1); (b) Evaluation time vs. NP for 25 agents distributed uniformly among computers (independent of each system's individual NP); (c) Speedup vs. NP, time required in one processor divided by time required for NP.

4.2. Tested rowing modes for the robotic "Ro"

As hard as it is to represent time-dependent events by still images, the representation consists of a top view with the reference fixed in the Ro-bot stand, as considered in the simulation scenario. The blade can be seen swinging in the surface from above the reference point. The diagram shows Ro represented as a single slice in the air-water interface, as presented in Fig.11. Gray slices are used to show positions the blade assume in time, black slices denotes points where commands are issued and are accompanied by arrows representing operations.

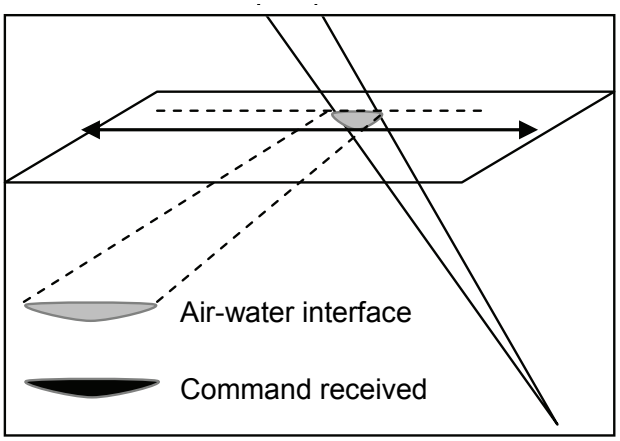

Fig.11 Rowing representation by still images, with slices marking positions and commands.

We first analyze the "classic" swinging for Ro as observed in the traditional humanistic control; it will be called C-2DF from here. Fig.12 shows a diagram of C-2DF rowing mode, obtained by the reproduction of codes previously presented in Fig.3.

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

The result was effective in 9 hours, as the MAS achieved a similar solution, but with altered timing resulting in an optimized code with increased signal frequency and consequently higher thrust over time. It will be referred as Optimized for 2DF or O-2DF. The frequency increase was obtained by reducing the time required to turn Ro in the corners. Instead, Ro is turned gradually in the region where less thrust is generated. Despite the higher performance, this rowing requires about the same energy for thrust as the traditional rowing. Fig.13 schematizes the mode.

Another solution was tested: The Rotated Blade, referred here as RB-2DF, is obtained by rotating the Ro by 90 degrees (see Fig.14). Searching for an optimized version for this mode, the GA usually led to some of the previously found results, indicating that this mode is less efficient than other previously found solutions. For the simulation, a flat blade was considered. In the real Ro model the wing effect exists in only one direction, causing side-effects, such as asymmetry in the resulting thrust.

Fig.12 Classic rowing (C-2DF) in the proposed graphic representation, the command sequence is shown in Fig.3.

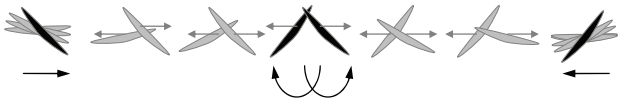

Fig.13 GA optimized 2DF rowing (O-2DF).

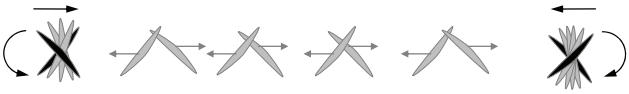

Fig.14 Rotated blade (RB-2DF).

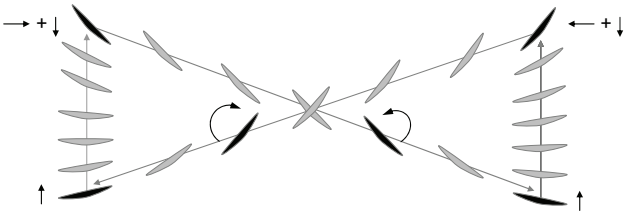

Fig.15 X-3DF diagram, the latest discovered mode.

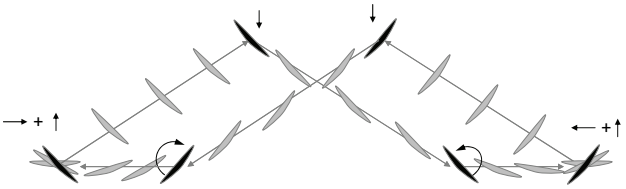

Fig.16 M-3DF rowing diagram, the first 3DF mode.

The next step was to allow operations in 3DF, using all actuators. Several new possibilities for operating Ro in this scenario were found, here we describe two among the most efficient: The X-motion blade swing (X-3DF) and the M-motion swing (M-3DF), being X-3DF the most efficient, named X swing for the moving that resembles an X shape; M-3DF was the first 3D mode found, producing less thrust if compared to X-3DF, was named M swing for the same reason. Both require higher energy levels to perform the rowing operation than 2DF modes. Fig.15 has a diagram representing the X-3DF, and Fig.16 schematizes the M-3DF swing.

Additional rowing modes were generated by changing the fitness function to several other directions, as left, right turn and backward rowing. Those were used in experiments for maneuvering and are not presented.

4.3. Verification of solutions

In this stage we tested the newly found modes in water-borne experiments performed in a small towing tank, with the Ro-bot mounted into a Catamaran ship for final verification (Fig.1). Each mode had its time taken for a 3-meter course under two conditions: Time from rest, with no speed (T1) and already in motion at constant speed (T2). The calculated fitness (F, in thousands of simulation points) and average of times measured (T1, T2 in seconds) are stated in Table 1.

Table 1 Experimental mode evaluation

MODE	F ($x10^3$)	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

It was noticed that the 3DF modes present a better acceleration, but an inferior top speed. The lowest resistance to the water flow is obtained in the RB-2DF, but this mode is still less fit than the newly found O-2DF, which is able to achieve higher speeds, nevertheless, RB-2DF revealed a superior speed compared to the traditional classic rowing.

The experimental results do not express the optimal results shown by the simulations. The discrepancy is a result of the approximations made to implement a fast simulation, which considered a static fluid and just measure the impulse caused by the swinging of Ro with the arm attached to the referential. Effects such as ship tilt and skid were not predicted in the simulation. Under certain fluid motion conditions and higher speeds, moving the blade back and forward also adds an increasing resistance to the water flow, increasing drag and therefore reducing the maximum speed.

A full ship hydrodynamic simulation expressing the exact conditions as in the water-borne experiments may spot even more efficient rowing modes for high speeds, but this imply much slower simulations, resulting in exponentially longer GA evaluations. The limited project schedule and computational power required a fast model in order to provide benchmarking information and to verify the efficiency of the method. Still, we obtained results featuring low consumption and high speed, as in O-2DF, and high acceleration, as verified in X-3DF.

5. Conclusions

In this paper, we try to find new solutions for the "Ro" control problem autonomously, introducing an agent-oriented evolutionary approach. The following results were obtained:

(1) The experiments demonstrated that the method can successfully locate and identify solution areas in the multi-variable space, eventually spotting the highest fitness solution for the specified control problem.

(2) The method was able to identify new solutions for the proposed problem, in addition to the conventional optimized solution as expressed by the fitness function.

(3) Some of the newly found solutions clearly demonstrate to be non-intuitive, being hardly achieved by the traditional human specification method.

(4) Other useful solutions for multi-directional maneuvering could be easily identified, by simply changing the fitness function to evaluate vector projections in the desired direction.

(5) Using the robotic 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.

(6) The agent-oriented approach achieved solutions in reduced time, obtaining super-linear speedups for an increased number of agents and processors involved.

(7) It was observed that the quality of obtained solutions depends intrinsically on the accuracy of the computer simulation used to evaluate the operating conditions. Slightly different scenarios than considered in the simulation, as the use of Ro-bot mounted in a ship for water-borne experiments, caused the solutions to achieve inferior performances in such alternate scenario than predicted by the simulations, as a result of inertial losses and other ship motion effects not considered, such as tilt, skid and deflection by operation of Ro.

Acknowledgement

The authors would like to thank Mr. Y. Hirakawa and Mr. T. Takayama for the expertise and support in the preparation of the physical model and measuring instruments, fundamental for carrying out experiments.

References

(1) Melanie MITCHELL: An Introduction to Genetic Algorithms, MIT Press, pp. 155-178, 1997

(2) Mengchun XIE: Cooperative Behavior Rule Acquisition for Multi-Agent Systems using a Genetic Algorithm, Proceedings of Advances in Computer Science and Technology, ACST'06, ACTA Press, 2006

(3) Ian FOSTER: Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering, Addison Wesley. Online http://www-unix.mcs.anl.gov/dbpp/, 1995

(4) P. J. BENTLEY (Editor): Evolutionary Design by Computers, Morgan Kauffmann, 1999

(5) Erick CANTÚ-PAZ: Efficient and Accurate Parallel Genetic Algorithms, Book Series on GA and Evolutionary Computation, Kluwer Academic Publishers, Volume 1, pp. 28-30, pp. 81-95, 2002

(6) Zbigniew MICHALEWICZ: Genetic Algorithms + Data Structures = Evolution Programs, Springer Verlag, pp. 188-193, 1996

(7) A. GRAMA, G. KARYPIS, V. KUMAR and A. GUPTA: An Introduction to Parallel Computing: Design and Analysis of Algorithms, 2nd Edition, Addison Wesley, 2003

(8) Various online contributors/references: http://www.multiagent.com/ ; GA Archives http://www.aic.nrl.navy.mil/galist/

Question & Answer

M. TSUGANE (Tokai University): Please tell the function of each agent in one experiment carried out. (three agents case).

R. NEVES (Yokohama National University): The three agents have similar functionality, (Fig.7) but perform searches in different regions. They receive a portion of the space to perform the search on, determined by a central start point and a variance, defined according to the total number of agents. This localizes the population, therefore improving the resolution of the search. All agents then work over its population, constantly evaluating and applying GA. They exchange information through the server, such as evaluations, progress and scanned areas, avoiding redundancies.