Optimization of Control Strategy Assignment. A Framework Structure for Multiple Mobile Robots

Fadi Issa*. Ioan Dumitrache**

Politehnica University of Bucharest,
Faculty of Automatic Control and Computers
** (e-mail: fadiissa82@yahoo.com)*
*** (e-mail: idumitrache@ics.pub.ro)*

Abstract: In this paper we present a concept of a framework structure that can be used in the study and implementation of multi mobile robot control strategy selection approaches. The control selection or assignment is important in situations of varying environments and constraints. The framework can be used for testing control strategies for single or multi mobile robots groups, and analyzing proposed approaches for mobile robot control in such a way that allows comparison among these different approaches and also allows the evolution of an optimal control strategy assignment to a group of different mobile robots doing different tasks.

Keywords: Framework, Genetic Algorithm, Artificial Neural Networks, Multi-Threading, Multi-Task Multi-Robot Groups.

1. INTRODUCTION

In the recent years, a significant growth of interest has occurred in investigating problems that involve multiple, rather than single mobile robots [Gerkey, 2003]. This was accompanied by technological evolutions of computers, sensors and actuators [Cote, 2004], leading to increased complexity of multi-robot systems. This complexity is reflected in larger team sizes, and greater heterogeneity of robots and tasks. This trend will continue and it is expected that larger and larger robot teams will be engaged in concurrent and diverse tasks over extended periods of time [Gerkey, 2003].

Development of mobile robots tries to move from specific purpose robots to multi-task general purpose robots that can also be used in different environments. A typical mobile robot can do different tasks (Fig.1.a), whether they are simple or not; each task can have different possible control strategies that implement it (Fig.1.b), and each one of these control strategies has advantages and disadvantages, making these control strategies suitable for some environments or conditions while not suitable for others. This raises a question: "Which control strategy should be used that most fits the current environments and constraints for carrying the requested task?"

In this paper we try to answer this question. Even for relatively simple multi mobile robot systems, this question is still important, and its importance grows with the complexity (in size and capability) of the system under study, since there will be too many research questions and/or variables at play.

One of the aspects of research that has raised from the advancement of multi-robots is *task allocation* [Brian, 2004], which became a key research issue in its own right. Our research goes a step further; to when a task can have several approaches of implementations (several control strategies), and the best control strategy must be chosen for each robot to execute its own task. Usually, these different implementations are provided either to suit different conditions and environments, or simply they are provided by different researchers (each one addressing the same task from a different point of view, and leading to advantages and disadvantages for each approach over the other).

A control strategy can be a small variation or modification of an existing one, or it can be a completely new one, but in order to have a case that applies in all circumstances, we will consider every control strategy as a new one in its own, regardless of its similarities with other strategies.

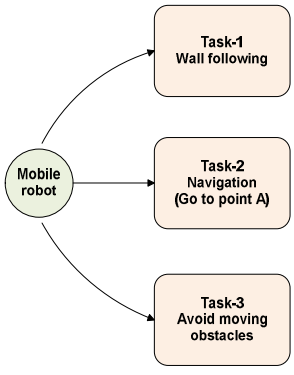

Fig. 1.a Mobile robots can do several tasks.

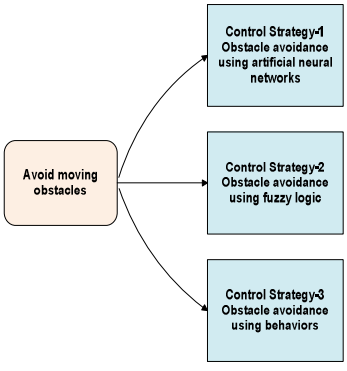

Fig.1.b A task can be implemented using several control strategies.

2. CONTROL STRATEGY SELECTION IN MOBILE ROBOT GROUPS

Current mobile robots can be used for multiple tasks; each task (according to its popularity and the amount of research put into it) can have several control approaches that implement it, and we call them "control strategies". An example of such a task with multiple control strategies is *obstacle avoidance*; it has tens of control approaches, and new ones keep coming (other tasks such as localization, mapping, planning, modelling, recognition, searching, tracking, interaction, cooperation, decision making, all have several implementation approaches on different robotic platforms [Cote, 2004]). A robot has to select the proper control strategy that implements its required task depending on the conditions and constrains it encounters, and on the performance required.

Therefore, when a task can be done in several ways, the best way has to be chosen intelligently; in order to do this, there should be provided a *metric* for each task implementation approach, which can be used to compare two candidate approaches for a task to select one of them. So, control strategies that implement the same task can differentiate themselves from other strategies in several areas (some are suitable for certain environments, some are faster to accomplish the same goal, some use less resources, some are more accurate). Each one of these areas has a performance measurement that allows comparing different strategies according to the same basis.

The areas from which we can obtain performance measurements can be either hardware related (such as the memory needed by each method, or the processing power needed, or the communication overhead and latency), while other areas are non-hardware related (such as the accuracy of the method, the time it takes to accomplish a predefined task, or any other measurement that is of interest to the user). In some measurements, the performance of one control strategy can be superior to the others, while for other measurements, this may not be true; this is because algorithms differ among themselves in their ability to scale up in terms of number of robots, number of targets, number of obstacles, and environment fitness. These performance measurements, besides being used for control strategy evaluation, can also be used for multi-robot groups performance evaluation.

When there are two approaches that address the same task, each one with its own strength points, researchers usually tend to design new approaches that can combine majority of both method's strength, sometimes with sacrifices in other areas (such as complexity of coding or higher resources usage). We think it will be easier that, instead of trying to make new control strategies that combine the strong points of other approaches, and can be applicable in several areas, (which also takes more time and effort to implement), it will be better to let the robot choose the control strategy according to the varying conditions. This way, for each condition or constraints, the robot chooses a strategy that most suits these constraints.

When environment changes, there is a need to change the control strategy; if a robot is working alone, then this will not be a problem, since it can choose what best suits it to do its required task in a way that gives the best performance (according to the preferred performance measurement). But once several robots act together, things start to become challenging. There is the possibility that one control approach for one robot can affect the performance of another robot, or can prevent it from doing its task. A simple example is shown in Fig.2 which illustrates such a case; there are three robots of different types: the first two robots (R1 and R2) are from a small fast type, they move continuously (maybe carrying objects) between point A to point B, while the third robot (R3) is a slow one that has to cross the line between A and B to reach point C. If we suppose that the robots from the first type do not have obstacle avoidance mechanism, and the slow robot cannot pass to point C with its current speed, then if it tries to go there it will collide with one of the first type robots. Using another control strategy for the first type of robots that uses slower speeds will solve this issue. This is a simple case to show what can happen if each robot acts selfishly just based on its interest, and how it can affect the performance of other robots, making unbalanced performance for the entire robots group.

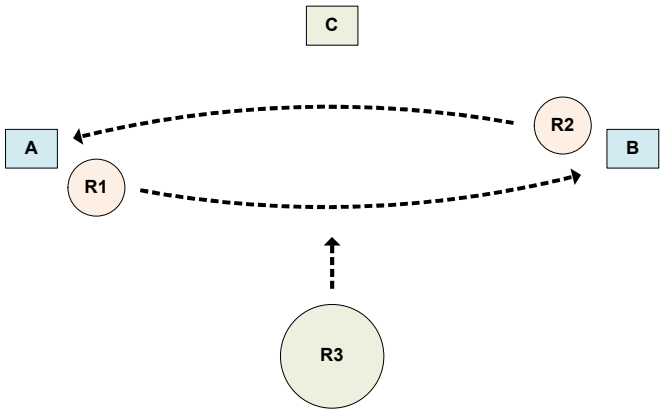

Fig.2 Robots R1 and R2 are moving between point A and point B with a speed that prevents robot R3 from going to point C.

In order to implement a system that solves the control strategies assignment problem, we will first have a look at the current robotics tools and frameworks. There are currently a great variety of programming environments and tools for robotics, and some of the most famous are:

- ***Player*** [Toby, 2005]***:*** device abstractions for robotic programming.
- ***CARMEN*** [Carmen 2009]: navigation toolkit.
- ***MIRO*** [Utz, 2002]: a middleware for mobile robot applications.
- ***MARIE*** [Marie, 2009]: a programming environment allowing multiple applications, programs and tools, to operate on one or multiple machines/ OS and work together on a mobile robot implementation.

Most of these initiatives are developed independently of the others, driven by specific applications and objectives in mind

[Cote, 2004]. However, they do not address the issue of control strategy assignment, and if we want to add this capability to one of these tools or frameworks without modifying their existing code structure, it will be a very difficult task. Besides, they do not offer means of comparisons between the different approaches for a task based on a performance measurement.

The current efforts to address the control strategy selection issue are small, ad hoc in nature, and relatively little has been said regarding it as a general issue in multi-task multi-mobile-robot systems. But this does not prevent us from investigating it. In fact, we believe that it will be more useful when this type of robotics systems grows more in capabilities and popularity, and developers contribute more to frameworks that address this issue.

We propose a framework that contains a new system for selecting control strategies for groups of heterogeneous mobile robots. Our system takes the whole group performance into account to balance the control strategy selection.

3. THE FRAMEWORK

Robotics research (like other types of work) is affected by the tools that are used; better tools make tasks easier to implement. The availability of flexible, reliable, and reusable tools for dealing with mobile robots is important to the research community. This set of tools is called *"robotics programming framework"*.

We propose a framework structure that can be used to search for an optimal solution that gives each robot the best control strategy that implements its required task, in a way that will result in the best overall performance for the whole group of mobile robots.

We set some goals when designing this framework, besides implementing the control strategy assignment system, the main goal of any framework is to reduce the time and complexity of development, In order to achieve this we tried to meet the following requirements:

- ***Reusable software:*** which suggests to loosen the coupling between software modules. This lead us to use object oriented programming language for implementation. The framework provides predefined object interfaces so that modules built on top of the framework have a common ground for interaction.

- ***Platform independence:*** Users should be able to freely choose an appropriate platform for running the applications, this is why we chose Java, which can run on any hosting operating system thanks to its virtual machine.

- ***Enhanced Scalability:*** The framework should be flexible and extensible. We looked at this scalability goal from both hardware and software perspectives.

From the *hardware perspective*, we built this framework in a way that takes benefit from current and expected advancements in computer hardware, especially Multi-Core Processors [Guz, 2009]; since we have a lot of components running together, it is good to develop the framework to make use of several processors, this will help increase the performance, and will decrease the time needed till reaching an optimum solution. In order to do this, the framework is designed to take benefit of several cores [Akhter, 2006], so we developed a multi threading approach [Multithreading, 2009] that is integrated into the core of the simulation, and it is based on asynchronous threads [Intel, 2007] that represent each component in the framework.

As for scalability from the *software perspective*, since the framework is positioned towards contributors to develop continuously, then a goal of the framework is to make their contributions as easy as possible in regards to the easiness of adding their work, and how it can affect or conflict with the existing work. We want the framework to be easy to improve and maintain, and allow adding new components without the need to modify existing code; this lead us to build the software components based on interfaces and abstract classes [Alistair, 2006]; The framework defines key object types and the syntax and semantics of interface-specific messages that can pass between them; this way we ensure that when a developer adds a new class representing any possible object (whether it is a robot, or a sensor, or a task or control strategy), if he implements that class according to the relevant interface, then this will assure easy addition without modifying existing components; this is very important; especially when the size of the framework grows in size.

- ***Flexibility:*** Given the rapid pace of development and change in the robotics community, any system that is to remain relevant for a significant period must be flexible. We use polymorphism [Budd, 2000] to ensure that future components (and current ones) can communicate with each other without prior knowledge of each other at design time.

The framework is composed of two main functionality modules (Fig.3):

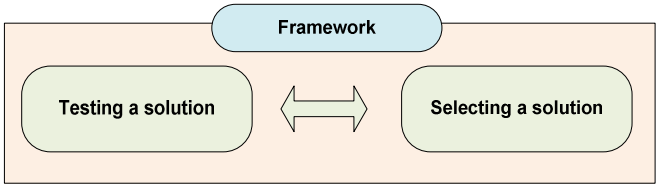

Fig.3 The two main functional modules of the framework.

- a testing system that accepts a possible solution as an input to test it and evaluate its performance;
- a system that generates and selects a solution according to some rules, and then sends this supposed solution to the testing system to verify it.

These two systems are connected together using a bidirectional connection; the selection system sends a candidate solution to the testing system, and the testing system sends back the results of that test.

Testing a possible solution can be done in two ways (Fig.4): either by applying the candidate solution on the real robots in the group under test (the optimal solution we are trying to find is a solution that contains the best control strategies that

has to be used by each robot in order to get the best overall performance), or by testing it using simulation. The simulation can have some advantages over the real world testing, since it can be done much faster by speeding up the simulation time. This can be very useful in situations where we have to test a lot of candidate solutions (as in Genetic Algorithms), where sometimes we have hundreds of candidate solutions that need to be tested (which is very time consuming if done in real world).

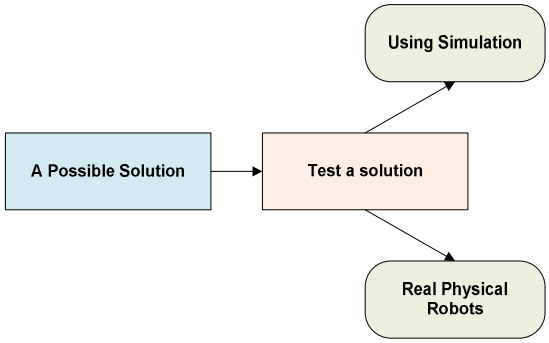

Fig.4 Testing can be in real world or by simulation.

The selection module is where most of the effort of our research is put. This module has the goal of finding the best possible control strategy assignment for all the robot in the group of mobile robots, and here we developed two selection (or searching) systems, one is based on genetic algorithms, and the other is based on a hybrid system between genetic algorithms and artificial neural networks (Fig.5). Each one has some advantages over the other; the genetic based selection system is more accurate than the hybrid one, while the hybrid system is faster in reaching the optimum solution. These selection modules run independently from each other, but if they are given enough time, they should reach the same solution. But sometimes an acceptable solution can be sufficient for some users for their specific needs; this is why the final choice of what solution to accept relies on the user preferences.

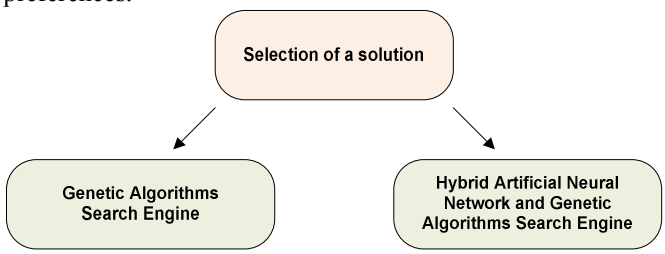

Fig.5 Two systems for selecting candidate solutions

The general outline of how this framework works is shown in (Fig.6): at first, the user specifies the performance measurements he wants it to be optimized for the group of mobile robots (e.g. he specifies that he wants to have minimum execution time, or minimum energy used, or minimum memory used), then the genetic algorithm system will start to evolve around this performance measurement. During this time, the results that are obtained from testing each chromosome in the G.A. population are converted to training samples, and are sent to the artificial neural network in the hybrid system for training purpose. Then, once the ANN is trained, the genetic algorithm component of the hybrid system starts to use the trained neural network to test is own solutions and evolve them to reach optimal solution.

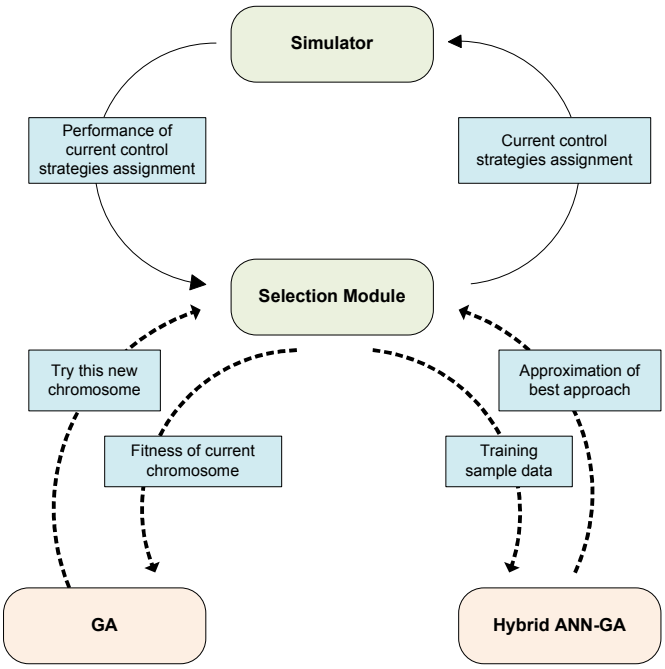

Fig.6 General outline of how the framework functions.

Our idea is to build a framework that develops by time, accumulating past implementations and their performances, and allowing to compare them on the same basis and measurements, so a researcher develops a new control strategy that implements a task, and then leaves the framework to decide when and under which condition to use it.

Next we will discuss each selection system in detail.

3.1 The GA System:

A genetic algorithm can be used to find a solution in much less time than other approaches for the same type of problems, although it might not find the best solution, but it can find a near perfect solution in a relatively short time. It relies on representing the problem elements in terms of genes, while a complete solution is represented as chromosome, and the set of possible solutions in a generation is the population.

A solution to the assignment problem has the following form (Fig.7*)*

We will use the following G.A. representation: each gene will represent a control strategy that is assigned to a robot in order to accomplish a task; the position of the gene in the chromosome represents the robot while the chromosome will represent the solution of the whole robot group control strategy assignment. It will assign a control strategy for each robot in the group of mobile robots, each one depending on

the task that it has to accomplish. The population is the set of different chromosomes (solutions for the problem); Our goal will be to reach the best possible chromosome by using crossover and mutation. The chromosome should, in some way, contain information about the solution it represents, which in our case is the set of control strategies assigned to the different robots in the group.

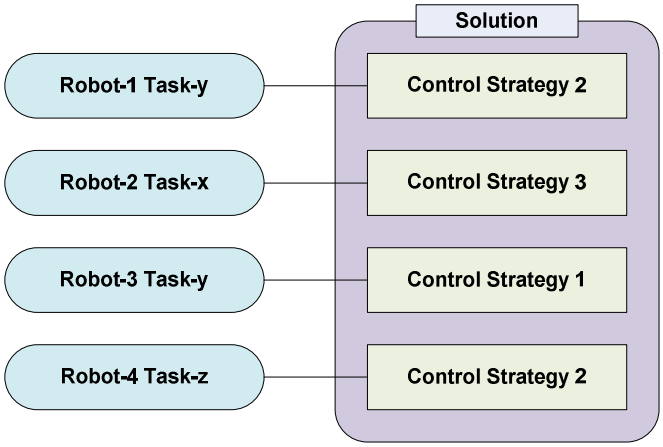

Fig. 7 The structure of a solution to the control assignment problem.

The stop condition for the genetic algorithm to cease creating new generations of populations is usually when it reaches a performance that is considered acceptable for the studied problem.

The crossover and mutation take place according to two different probability values: crossover probability and mutation probability, which can be set by the user. Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better, while mutation is made to prevent GA from falling into local extreme, but it should not occur very often, because then GA will become closer to random search.

The whole system operates as follows (Fig.8):

- the user selects the performance measurement he wants to be used by the fitness function of the genetic algorithm, and whether he wants to maximize this value or minimize it;
- the robots in the group send requests to the genetic algorithm system to provide them with the control strategies they have to use to achieve their tasks; at first, these control strategies will be randomly chosen, and for each chromosome in the population, the GA system will test it and evaluate its performance;
- after finishing testing the whole population, the performance of the best chromosome is evaluated to decide if that is an acceptable solution or not; if not, a new offspring of the population is created and then the new chromosomes are assigned to robots for testing, then wait for the new evaluation;
- this process repeats until a stop condition is met (whether it is a number of generations, or a performance that is considered acceptable).

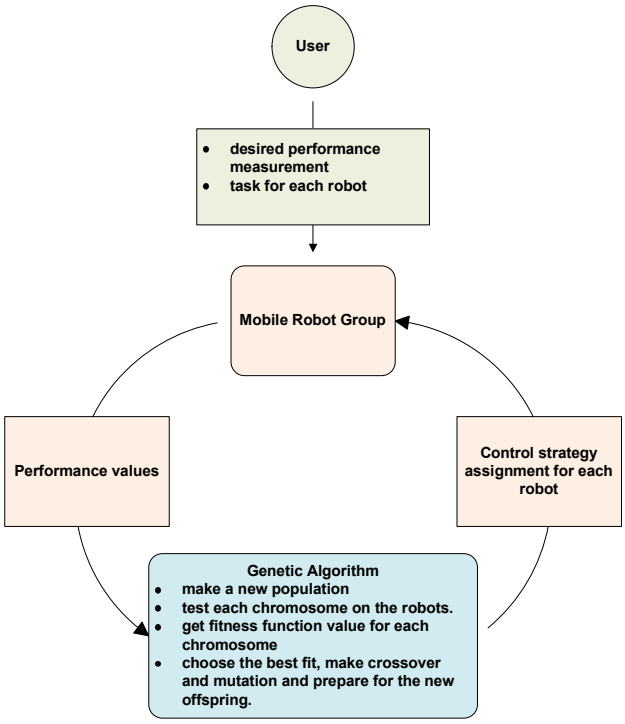

Fig. 8 The entire system operations

3.2 The Hybrid ANN-GA Component

In a dynamic environment, the conditions and requirements from robots can be changed without prior informing, and the robot can find itself puzzled on what control approach should it use. Our framework offers the tools that deal with this situation without the need to start the testing process of G.A. from the beginning to evolve a whole new population of chromosomes. This is accomplished by incorporating a hybrid intelligent system based on genetic algorithms and artificial neural networks that can guide the robot on what to choose (this choosing based on ANN happens a lot faster than waiting for the genetic algorithm to evolve).

Another benefit of this hybrid selection system is that it can be applied while the other G.A selection system is running, to help reaching an acceptable solution sooner than waiting for real testing (whether it is by simulation or real world tests). To achieve this, the artificial neural networks must be fed with trained data during the testing for G.A selection (or it can be collected from random tests); and by training the ANN, it should get an approximation of a function that represents the performance of the system according to the input factors (the used robots, the environment,..). And by relying on this proximity function, we can get immediately the expected performance of a group of mobile robots if we put the problem conditions as an input to this ANN.

Another use of this system is when the performance measurement that must be optimized is changed, even if all the conditions and robots remain the same; for the G.A selecting system it has to start from the beginning, while if the ANN was getting trained with the test data (there can be an ANN for each performance measurement), then according to the new performance measurement required, it just uses

the trained neural network to estimate the performance of possible solutions, Fig(9) shows how the training data is taken from the G.A selection system.

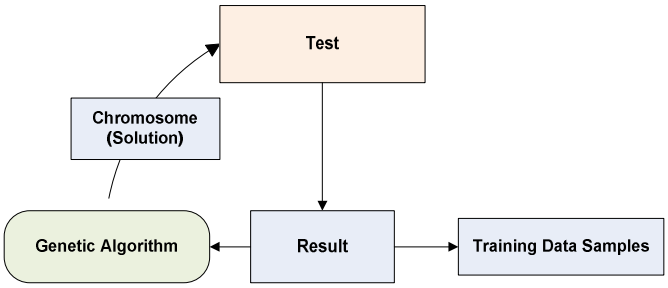

Fig.9 Test data from G.A selection operations are sent to the ANN as training data.

The training data samples can be either created and tested randomly, or collected from the testing results of the G.A selection system while it is evolving. In either case, the training samples can contain different performance measurements, not only the one that we are trying to optimize. Then, if we become interested in a different performance measurement, we can take benefit of current collected data, with no need to make new tests. After that, the training data will be sent to the ANN component of the hybrid system.

These training data samples are sent to the ANN as shown in Fig.10. The input layer of the ANN has a number of neurons equal to the number of factors affecting the performance of the mobile robot group (it can be just the control strategies used by each robot); the number of neurons in the hidden layer is set by the user (it can take several attempts till reaching a number of neurons that makes the ANN learn better), and the output layer of this ANN contains just one neuron (which represents the performance of the group of mobile robots according to the chosen performance measurement).

After the ANN is trained, the G.A. component of the hybrid system comes into action. This G.A component will start to evolve the inputs of the ANN in order to reach an optimum performance (which is the output of the ANN), and this is usually done very fast compared with the G.A selection system, since here there is no need for simulation in order to get a performance evaluation of the mobile robots group; instead, it relies on the ANN to get the estimated performance, and the larger the training data (more samples), the better the chances are that the ANN learned an approximation function that mimics the real world results (Fig.11).

It is worth mentioning that the training data are not bound to be collected from the G.A. selection system; instead, they can be collected from random testing. There is one more feature or benefit from using the ANN approach: it is when the conditions of the robots or environment changes slightly, the same trained ANN can still be applied, and there is no need to train a new ANN for this particular purpose

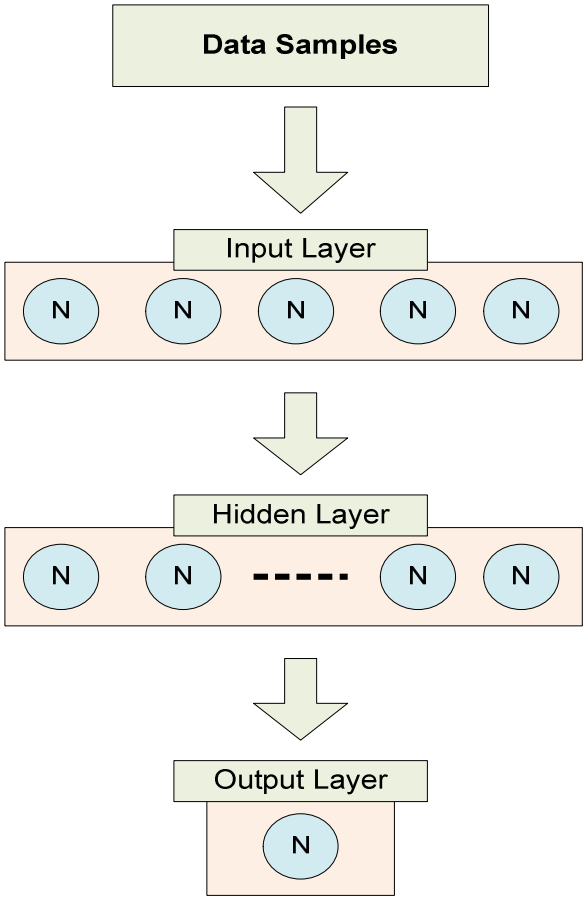

Fig.10 ANN training on the data samples sent from the GA selection system.

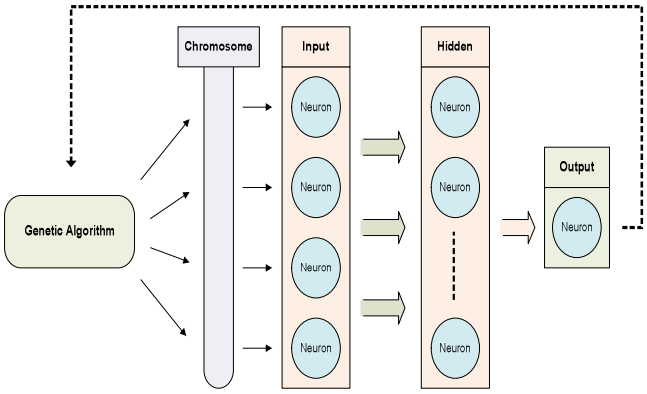

Fig.11 G.A. used to evolve the inputs of ANN to reach optimum output.

3.3 Detailed framework representation

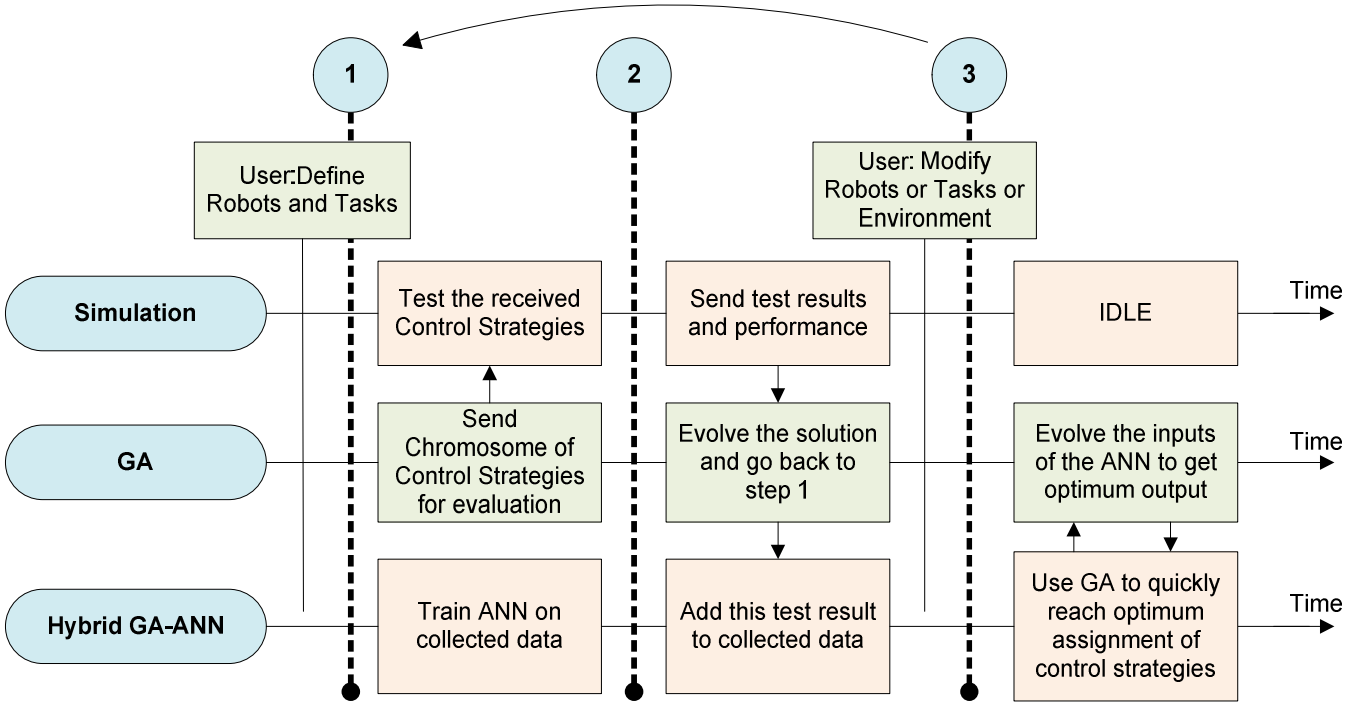

Fig.12 The entire framework in steps.

The selection module of the framework is the one responsible for optimizing the performance of the group of robots running different tasks, by assigning the proper control strategy to each robot so we get the best overall performance from the entire group of robots.

There are two scenarios in which this system can be used:

- the first one is when we do not know the performance attributes of the control strategies, and we need to test them in order to get these performance measurements. The G.A selection system was implemented for this scenario, and it can reach an acceptable solution that provides good performance in a relatively short time.

- the second scenario is when the system has been running for some time by now, and during that time it has been storing the performance data of each control strategy for each robot, and training an artificial neural network with this data. Therefore, when the environment varies, or a new condition is added, or a constraint changes, then we can either use the first approach of genetic algorithms and collect the performance information from the beginning, or we can rely on the collected training data till now and use the trained ANN to find the best combination of control strategies in a lot shorter time than when using the genetic algorithm approach.

The first approach is an accurate optimum search, which uses GA, while the other is an approximation on how the solution should be; its accuracy depends on the training data, its amount, and its variety.

When a user interacts with the simulation, he has the possibility to choose the approach that he likes for the task implementation, or leave this responsibility to the framework that can use its built-in selection engine to make this assignment procedure in a way that gets the best overall performance possible.

The whole situation is illustrated in (Fig.12), where it shows the interaction between the components of the framework, and how it differs according to time and user inputs.

The first step is when the user defines the conditions and constraints of the mobile robots group, such as the number of running robots and their type, the task that each one must carry, and the environment type, etc.., then the G.A starts to work by sending some candidate solutions to the simulator for testing. In step2, the simulator sends back the results of testing the candidate solution that it received previously from the GA selection system, so the GA system evolves and tests other solutions, while at the same time, it sends the test results which it received from the simulator to the ANN in the hybrid system as training data. These two steps (1 and 2) repeat until one of the systems reaches an acceptable solution, or some conditions changes, then it goes to step 3 where the hybrid system starts to work by using the same mechanism of the G.A. selection system, but instead of using the G.A to evolve the solution itself as was the case in the G.A selection system, it is used for evolving the inputs of the ANN to reach the optimum output.

3.4 Framework Usage

The users of the framework can be:

* ***Framework builders***: which do not work on any specific application but, instead, provide the infrastructure code to support applications. They build components models, like

new sensors, or modelling new types of mobile robots. They use the infrastructure of the framework to add these new components, by implementing the existing interfaces, or by modifying and providing new ones, (for example, they can add a new task for an existing type of mobile robots, and then they provide an interface that has to be implemented by researchers if they want to test a new control strategy to perform that task).

Control contributors: Usually, they use the existing predefined components and interfaces to implement their control approaches for the existing tasks.

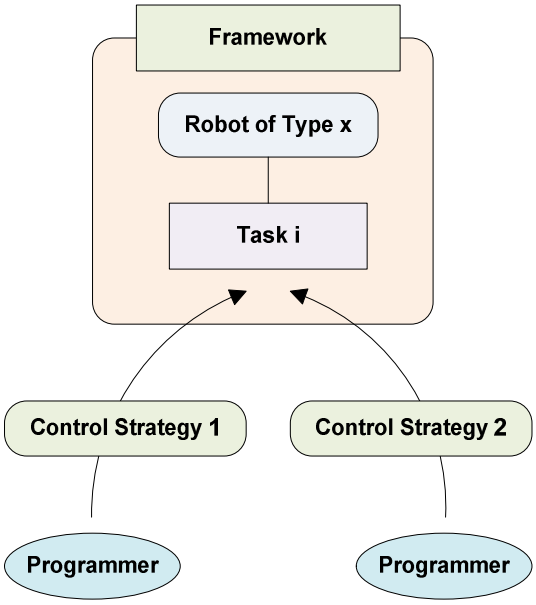

Fig.13 Adding control strategies to the framework.

Fig.13 shows the way in which two contributors add two different control strategies that implement the same task; all they have to do to ensure correct integration with the framework is to implement the task interface (defined by the framework builders) in their provided control strategy classes.

If the user of the framework is just testing existing control approaches, then our framework provides the following capabilities:

* gather a set of robots into a team or group.
* give tasks for each robot in the team.
* add or remove a robot from the group and give tasks individually.
* choose a performance measurement and let the framework select the proper control approach for each robot in order to optimize the performance measurement of the entire group of robots.

4. TESTS AND EVALUATION

Our first implementation of this framework was aimed as a proof of concept, so the concentration was on implementing the basic functionalities and being sure that they work as they are supposed to. Therefore, when we developed the simulator component of the framework, we did not model a lot of sensor types and a lot of robot models (actually the tests were done on models of Khepera2 and models of some custom built mobile robots). We used this simulator to test and evaluate the proposed solutions that are sent to it from either the genetic algorithm selection system, or the hybrid system.

We made tests to show the difference between each of these selection systems, and how they compare to each other. The test problem was that we have ten mobile robots, each one trying to navigate from one point to another, then return to the first point. Each one of these robots has 3 control strategies it can use to accomplish its goal, and each strategy uses different speed for movement.

We defined the goal of the whole group to be that all robots of the group reach the other point and return to the first one in the shortest possible time. So our performance measurement was time, and we wanted to minimize it.

For testing the GA based system, we set a population of 4 chromosomes in each generation, and the mutation rate was set to different values ranging from 10% to 30%. The results were the following: for a mutation rate of 10% it reached an optimum solution after evolving for 97 generations, while for 30% mutation rate, it reached the same solution after 44 generations. After this, we tried to set the population to 10 chromosomes, which delivered an optimum solution in fewer generations than for the smaller population used before, and it reached the optimum solution even faster when raising the mutation rate again from 10% to 30%. This means that for some problems, raising the mutation rate can fasten reaching the optimal solution. The fitness function that was used to evaluate the performance of a chromosome was the sum of times that each robot needed in order to cross the distance between the two points; the crossover operation used one point crossover and the position of this point was randomly chosen each time there was a crossover. We also used elitism of two chromosomes so that the best 2 results in a population are not eliminated when a new population is formed. We could know that the solution reached was optimum when we let the genetic algorithm evolve for additional hundreds of generations, while the best chromosome was still the same one we obtained before; so we knew it was the optimum solution. When we increased the mutation rate over 30%, it actually gave worse results, making the convergence for a solution to take more generations than with smaller mutation rate; this indicates that increasing the mutation rate does not necessarily increase the speed of reaching an optimal solution. Instead, it makes the search to be closer to random search. The experiments we did show that using genetic algorithms to evolve assigning of control strategies to robots can lead to a solution that is considered acceptable (compared to other approaches) in a relatively short time.

For testing the hybrid ANN-GA system, and for the same problem used in testing the GA system, we wanted to see how many training samples have to be collected from the GA system before the ANN starts to give good estimation of chromosomes performances. In this specific problem, it took the neural network to wait for the GA to test about 40 chromosomes and convert their results to training sample in order for the ANN to reach the optimum solution using the

hybrid method, which shows real benefit of the hybrid system over the G.A (the GA needed to test more than 160 chromosomes to reach the same solution). The only limitation of the hybrid approach is that we cannot be sure if its solution is really the optimum one or not, while in the GA method, we are more confident of its results because they are based on real testing (or simulation at least), while the hybrid method relies on how the ANN represented a function approximation of the chromosome fitness.

This proves our point that we can use this hybrid system to reach a solution better than the ones available (in the training samples), without necessarily being tested previously. This was done in a relatively short time compared to the GA alone. Our tests were for a simple problem, for more complex problems it could reach the optimum solution after hundreds of generations on a training data of tens of samples. In both cases, it could reach solutions that are not present in the training data, and which delivered better performance than all the previously tested training data.

In our tests we chose time as a performance measurement, but there can be other measurements. The concept is still the same, optimizing the control strategy assignment in order to get the optimum group performance.

5. CONCLUSION

Incorporating some parts of this research in other more established frameworks can add new capabilities to them, making it possible to test our selection systems on a wider range of mobile robots and so increase the benefit of the approaches presented here.

We presented a new approach regarding the design of multi mobile robot frameworks while having the performance of the entire group in mind. This was achieved by addressing the control strategies assignment issue. The underlying ideas of genetic algorithms and artificial neural networks were borrowed directly from artificial intelligence. Yet, they seem to be applicable to the framework implementation instead of just being used for special purpose and specific problems. The use of interfaces (in object oriented context) makes contributing to the framework an elegant procedure.

We hope that using this framework will advance the use of computational intelligence, and specifically G.A., in areas of mobile robotics with multiple strategies for multiple tasks implementations.

In the end, the true utility of the framework will be determined by future robot programmers who will use this framework and modify it. And we hope that control assignment will play a good role in future robotics research and get more attention for further investigation.

REFERENCES

Akhter, Shameem, Roberts, Jason, 2006, *"Multi-Core Programming Increasing Performance through Software Multi-threading"*, Intel Press.

Alistair Rooney, 2006, "More OO in Java—Interfaces and Abstract Classes", *"Foundations of Java for ABAP Programmers"*, Apress, Pages 57-60.

Brian P. Gerkey, Maja J, 2004, "A formal analysis and taxonomy of task allocation in multi-robot systems". *Intl. J. of Robotics Research,volume 23, pages :939-954.*

Budd, Timothy, 2000, "Understanding Object-Oriented Programming with Java", Addison Wesley.

Cote, C. Letourneau, D. Michaud, F. Valin, J.-M. Brosseau, Y. Raievsky, C. Lemay, M. Tran, V. 2004, "Code reusability tools for programming mobile robots", Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems, (IROS 2004). Volume: 2, On page(s): 1820-1825.

Gerkey, Brian P., and Matarić Maja J, 2003, "A Framework for Studying Multi-Robot Task Allocation", Conference Paper, *Multi-Robot Systems: From Swarms to Intelligent Automata*, Volume II. pages 15-26, the Netherlands.

Marie, retrieved august 10th,2009, from http://marie.sourceforge.net

Carmen, retrieved august 10th,2009, from http://carmen.sourceforge.net

Multithreading (computer hardware) Retrieved July 20th, 2009,from: http://en.wikipedia.org/wiki/Multithreading

Intel® Software Network, April 12, 2007, "Measuring performance on multi-core H.W."

Toby H. J. Collett and Bruce A. MacDonald, Brian Gerkey, 2005, "Player 2.0: Toward a Practical Robot Programming Framework", *Proc. of the Australasian Conf. on Robotics and Automation (ACRA)*.

Utz, H. Sablatnog, S. Enderle, S. Kraetzschmar, G. 2002, "Miro - middleware for mobile robot applications", *IEEE Transactions on Robotics and Automation*, Volume: 18, Issue: 4 ,pages: 493- 497.

Guz, Zvika, Bolotin, Evgeny, Keidar, Idit, (2009), "Many-Core vs. Many-Thread Machines: Stay Away From the Valley". *IEEE Computer Architecture Letters*, vol. 8, no. 1, pp. 25-28.