UBA-Sot : An Approach to Control and Team Strategy in Robot Soccer

Juan Miguel Santos, Hugo Daniel Scolnik, Ignacio Laplagne, Sergio Daicz, Flavio Scarpettini, Héctor Fassi, and Claudia Castelo

Abstract: In this study, we introduce the main ideas on the control and strategy used by the robot soccer team of the Universidad de Buenos Aires, UBA-Sot. The basis of our approach is to obtain a cooperative behavior, which emerges from homogeneous sets of individual behaviors. Except for the goalkeeper, the behavior set of each robot contains a small number of individual behaviors. Basically, the individual behaviors have the same core: to move from the initial toward the target coordinates. However, these individual behaviors differ because each one has a different precondition associated with it. Each precondition is the combination of a number of elementary ones. The aim of our approach is to answer the following questions: *How can the robot compute the preconditions in time*? *How are the control actions defined, which allow the robot to move from the initial toward the final coordinates*? The way we cope with these issues is, on the one hand, to use ball and robot predictors and, on the other hand, to use very fast planning. Our proposal is to use planning in such a way that the behavior obtained is closer to a reactive than a deliberative one. Simulations and experiments on real robots, based on this approach, have so far given encouraging results.

Keywords: Robot soccer, emergent cooperative behavior, planning by non-linear-optimization.

1. INTRODUCTION

Robot soccer provides an ideal environment for demonstrating the difference between deliberative and reactive behaviors. On the one hand, reactive behaviors are suitable for dynamic environments or tasks for which a rapid answer must be found (i.e. real time problems). On the other hand, deliberative behaviors can be used in static environments or for tasks without important time constraints. As a general criterion, reactive behavior implies that there is a mapping between the state and action spaces, while deliberative behavior implies that time is spent on planning. Using this criterion, robot soccer appears to be ill-conditioned for either of these approaches: firstly, because the state-action space is, in principle, infinite, and secondly, because the robots need to respond very rapidly.

To deal with an infinite or huge state-action space, we could try to cluster the space, and in this way to reduce the size of the mapping needed to implement reactive behaviors. However, it is not clear that a clustering technique would allow us to group states easily in a way that would allow us to assign a unique action to each cluster. From the control point of view, very small differences between states could lead to very different actions (i.e. a small change in the location of one robot could drastically change the optimum trajectory of another robot). Moreover, even if such a clustering were to exist, obtaining it might be as difficult as calculating the mapping corresponding to the reactive behavior.

Besides, the problem of the space cardinality affects the feasibility of the learning methods. In general, learning strategies are applied to reduced regions of the state-action space or with biases that allow us to cope with only a partial aspect of the problem (i.e. *how does a robot pass the ball to a teammate*? Or *how does a robot choose among a restricted number of actions in bounded scenes*? Etc.). Therefore, when reactive behaviors, learned in this way, are applied using the complete state-action space, their performance could be seriously affected.

On the other hand, planning allows the robots to take into account the complete state-action, at the cost of potentially excessive delays.

Another interesting aspect of the robot soccer problem is the collective behavior of the team. The

Manuscript received February 28, 2002; revised February 28, 2002; accepted June 24, 2002. The UBA-SoT project is supported in part by a grant from the Universidad de Buenos Aires, corresponding to Res. 1227/2001, and in part by Hewlett-Packard Argentina, Microsoft Argentina and Intel Argentina.

Juan Miguel Santos, Hugo Daniel Scolnik, Ignacio Laplagne, Sergio Daicz, Flavio Scarpettini, Héctor Fassi, and Claudia Castelo are with the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Ciudad Universitaria, Pabellón I, 1428 Cdad. de Buenos Aires, Argentina. (e-mail: jmsantos@dc.uba.ar, scolnik@fd.com.ar, ilapla@saluduno. com, sdaicz@dc.uba.ar, flavios@ciudad.com.ar, h_fassi @ciudad.com.ar, castelo@visa.com.ar).

desired situation is that the team behaves in cooperative way, that is, we want the individual behavior of each member to contribute to an increase in performance of the team over time. There is no unique way to obtain such cooperative behavior and each approach has different advantages and drawbacks.

In this paper, we introduce the approach used for the UBA-SoT (Universidad de Buenos Aires Soccer Team) to deal with the control and strategy problems.

From the strategy point of view, we followed the cooperative emergent approach. That is, we did not include hand-coded strategies involving two or more players during the game. Instead we put our efforts into endowing each player with a suitable set of individual behaviors, in order to achieve the cooperative behavior of the team. For example, we see a pass behavior as emerging from two individual behaviors, usually within two different robots, to pass the ball (for one robot) and to locate itself for receiving a pass (for the other robot, although it could perhaps be the same robot that passed the ball).

Thus, each robot needs to know when it must start each of its individual behaviors. In order to accomplish this, we associated preconditions with each behavior, which the robot must evaluate before acting.

From the control point of view, we used fast planning in such a way that the robot is able decide its action in a very short time, quite close to the response time of reactive behavior.

To complete the above approach, we must answer the following questions: 1) How can the robot compute preconditions in time? 2) How are the control actions defined, which allow the robot to move from the initial coordinates toward the final ones?

The answer to the first question is that we need fast prediction tools and models. Regarding the second question, we need fast action computing, based on the system state delivered by the vision system or by the simulation server.

In the remainder of the paper we describe the manner in which we answered these questions in our system. In Section 2, we provide details about the individual behaviors and their preconditions. In Section 3, we comment on the prediction methods used to facilitate the evaluation of the preconditions. In Section 4, we provide details about the optimization method which was selected, and how it is used in order to satisfy the time constraints.

2. COOPERATIVE BEHAVIOR EMERGING FROM A SET OF INDIVIDUAL BEHAVIORS

The basis of our approach is to enable cooperative behavior to emerge from homogeneous sets of individual and simple behaviors [3]. Except for the goalkeeper, the behavior set of each team robot has a small number of simple and individual behaviors (for example, ball passing, ball kicking, locating, ball intercepting). Since there is no centralized control, coordination among individual behaviors of different robots is carried out by a limited number of occasional messages.

Basically, the individual behaviors in each set have the same purpose: to move from the initial coordinates toward the target ones. The coordinates of a robot are the Cartesian location, the deviation angle and the wheel's velocities. However, there are two aspects that differentiate the behaviors. One aspect is the way in which the robot defines some components of the target coordinates. For example, depending on whether the robot decides to kick or pass the ball to another teammate, the final wheel velocities or the deviation angle are defined in different ways, although the robot must reach the same Cartesian location.

The second and main difference among individual behaviors is the conditions that must be satisfied for activating each of them. We name these conditions, *behavior preconditions*. Each behavior precondition can be formed by one or more *elemental preconditions*, such as: Is this robot the best placed to kick the ball toward the goal? Or, is this robot the best placed to pass the ball to a given teammate? Or, is this robot the best placed to intercept the ball trajectory?, etc.

Thus, the behavior of the team depends on the definition of the behavior set for each robot and a limited number of occasional messages among robots. However, the activation of each individual behavior depends on the behavior-precondition associated with it, and therefore the team behavior will depend on the set of elemental preconditions and how they are combined to form each behavior precondition.

Unfortunately, there is no standard method of knowing what the individual behaviors should be or of determining the behavior preconditions associated with each one. This is a process which must be made by-hand based on the knowledge that a designer has about the task to be accomplished. We identified seven candidates for individual behavior: *Clear the ball*, *Move to re-locate in defense*, *Receive and shoot the ball (robot j from robot i)*, *Kick toward the goal*, *Pass the ball (from robot i to robot j)*, *Move in order to receive the ball (robot j from robot i)* and *Move in order to re-locate within the football pitch*. In the same way, we defined a set of ten elemental preconditions and their combinations, in order to establish the behavior preconditions. For example, before a robot can decide to *Clear the ball*, it must evaluate four elementary preconditions:

1) Are there any free paths between the ball and our goal at time interval [t1,t2]?

2) Is the ball in our half of the field?

3) Is robot *i* the best placed to intercept the ball?

4) Is there any adversary robot which has a chance of kicking the ball into our goal?

That is, if the system state satisfies these behavior preconditions, then the robot can initiate the *Clear the ball* behavior. Elementary preconditions 1, 2, and 4 allow the robot to know if its team is at risk of a rival shooting toward the goal. The third one requires the robot to evaluate whether it is the best placed for moving toward the ball and clearing it, compared with the rest of the team.

A complete list of the elementary preconditions that we defined is:

EP1) Are there any free paths between the ball and the adversary goal at the time interval $[t_1,t_2]$?

EP2) Is robot i the best placed for reaching the ball and kicking it toward the goal at some instant in the interval $[t_1,t_2]$?

EP3) Are there any free paths between the ball and our goal at the time interval $[t_1,t_2]$?

EP4) Is the ball in our half of the field at time t?

EP5) Are there any messages from other teammates, wanting to receive the ball?

EP6) Is robot i the best placed for reaching the ball and passing it to robot j at some instant in the interval $[t_1,t_2]$?

EP7) Is robot j the best placed to receive a pass at some instant in the interval $[t_1,t_2]$?

EP8) Is robot i the best placed to intercept the ball at some instant in the interval $[t_1,t_2]$?

EP9) Are there any free paths between robot i and robot j at time interval $[t_1,t_2]$?

EP10) Are there any adversary robots which have a chance of kicking the ball toward our goal at the time interval $[t_1,t_2]$?

Then, we associated these elementary preconditions by means of logical operators to define the behavior preconditions:

Clear the ball
PC: EP3 and EP4 and EP8 and EP10
Move in order to re-locate in defense
PD: EP3 and EP4 and not EP8
Receive and shoot the ball (robot j from robot i)
PR: EP5 and EP1 and not EP4
Kick toward the goal
PK: EP1 and EP2 and not EP4
Pass the ball (from robot i to robot j)
PP: EP6 and EP7 and EP9
Move in order to receive the ball (robot j from robot i)
PM: EP5 and PP (from robot j to robot k)
Move in order to re-locate
PL: none

The order in the list indicates the priority of each individual behavior. This does not imply a sequential evaluation.

3. PREDICTORS

In order to implement these preconditions we had to use some tools based mainly on prediction. For example, to solve the elementary precondition, *"Are there any free paths between the ball and our goal at time interval* (t_1,t_2)*?"*, it is necessary to have a predictor of the ball's location and of the locations of the adversary robots.

The camera provides snapshots of the ball and the adversary-robots' coordinates $(x(t_i), y(t_i))$, i: 0, 1, ... and so the problem is to be able to use this data, which is affected by noise, to predict their future positions.

A number of algorithms have been employed in different fields for smoothing noisy data, using least-squares and functions such as polynomials, normal splines, B-splines, negative exponentials, minimum distance neighbors, cross correlation, outliers detection, wavelets, etc [4]. In the particular case of robot soccer, some additional factors must be taken into account (i.e., sudden changes in the ball's trajectory due to impacts against obstacles such as the robots themselves or the walls surrounding the football field).

In order to take this particular problem into account, a special algorithm was developed, which combines different techniques for providing reliable estimations of the ball's coordinates. Essentially, we use a fast implementation of the cross validation method, plus an analysis of the time series for deciding which data segments should be used for the prediction. The choice of algorithms to include was based on reliability and speed considerations, since this application requires real time calculations.

On the other hand, each robot needs to have a predictor of its teammates' positions (since at time t, each robot on the team only knows what the velocities of his other teammates are). The model used to predict the coordinates of the other members of the team was constructed using the kinematics equation:

$$x = x_0 + \Delta x,$$
$$y = y_0 + \Delta y,$$
$$\alpha = \alpha_0 - va.\Delta t,$$

where if $v_{left} \neq v_{right}$

$$\Delta x = \frac{vl}{va} * (\sin(\alpha_0 + va.\Delta t) - \sin(\alpha_0)) \quad \text{and}$$
$$\Delta y = \frac{vl}{va}(-\cos(\alpha_0 + va.\Delta t) - \cos(\alpha_0)),$$

or if $v_{left} = v_{right}$

$$\Delta x = vl.\cos(\alpha_0).\Delta t \quad \text{and}$$
$$\Delta y = vl * \sin(\alpha_0)\Delta t,$$

and

$$va = \frac{v_{left} - v_{right}}{L},$$

$$vl = \frac{v_{left} + v_{right}}{2},$$

where L is the distance between the wheels.

4. THE CONTROL PROBLEM

4.1. Introduction

In order for the planning to be very fast, we defined a cost function which has to be minimized using a non-linear optimization technique. It can be formulated within the framework of control theory. More specifically, this is a discrete time control problem, and therefore is equivalent to a nonlinear programming problem. There exists a huge variety of algorithms according to the kind of functions to be minimized, the availability of partial derivatives and eventually of Hessians, the number of variables, the memory requirements, and the necessity of having close to real-time solutions [5]. Constrained problems can be dealt with using interior point techniques, penalty functions (exact or not), Lagrangian techniques, etc. Their main feature is that if the value function is zero for the optimal values of the parameters, then the corresponding decision is feasible.

In order to solve this problem, we use a version of Powell's derivative-free minimization method that allows the robot to make a decision about the action to carry out [2].

The key to defining such a function is to consider a small number of *intermediate points* along the trajectory. At each intermediate point, the robot must decide what its wheel velocity values should be and for how long it should maintain these values. Between two intermediate points, the robot will describe an arc, and thus it will be easy to predict what the robot location will be at any given time or, for a given point, to predict what the delay to reach it will be.

Additionally, the cost function allows us to insert constraints, in order to avoid collisions, robot locations which would be outside of the football field, acceleration and velocity overshooting.

For each intermediate point, the number of variables included in the cost function increases by two or three, depending on whether the intervals – defined as the elapsed time from one intermediate point to another -, are fixed or variable.

In this way, an elemental precondition that says, "is this robot the best placed to kick the ball towards the goal?", will be translated into a computation of the minimum trajectory duration required by each team member to reach the target with the desired angle and wheel velocities.

However, since robot soccer environment is dynamic, the planned trajectories could rapidly become obsolete. For this reason, we need a fast planning method; that is, we need a very fast minimization technique.

Additionally, to have a fast planning method allows us to know what the wheel velocities should be, in order to navigate along the chosen trajectory. That is, the optimal values of the variables are precisely the wheel velocities and elapsed time of each arc along the trajectory. If the robot gets the velocities for the first arc, it will be able to react in order to follow the planned trajectory. In fact, the most likely situation is the one in which the robot decides to use data corresponding only to the first arc, discarding the remaining ones due to the environmental changes.

Although the nature of the behaviors is deliberative, if the robot can compute the appropriate action to take in a negligible amount of time, the resulting behavior will have the appearance of being reactive.

4.2. The cost function

The robot must reach the target coordinates which are defined by the tuple $<x_1, y_1, \alpha_1, v_1^{left}, v_1^{right}>$ where x_1, y_1 is the Cartesian location, α_1 is the deviation angle, and v_1^{left} and v_1^{right} are the left and right velocities, respectively. Analogously, the tuple $<x_0, y_0, \alpha_0, v_0^{left}, v_0^{right}>$ represents the current coordinates of the robot. We could formulate a simple cost function for minimizing and thus extracting the appropriate velocities which would allow the robot to move from the current to the target coordinates (those corresponding to the minimum of the function). For example, we could take some distance measurement (i.e. Euclidean) between both points and straightforwardly define such a cost function. However, the more probable situation is that the robot will not be able to reach the target by setting its velocities only once.

To follow the correct trajectory from the current to the target coordinates, the robot must know the velocity values (for the left and right wheels) at each point over time. However, it is possible that the robot will not need to change its velocities at each point along its trajectory. If we consider that for each velocity pair, the trajectory describes an arc from an initial intermediate point to a final intermediate point, then the complete trajectory will be defined by a set of arcs from $< x_0, y_0, \alpha_0, v_0^{left}, v_0^{right} >$ to $< x_1, y_1, \alpha_1, v_1^{left}, v_1^{right} >$ Therefore, there are, at present, two questions which need to be answered: 1) How many arcs does the robot need to reach the target?, and 2) What is the length of each arc?

Before answering these questions, we will formulate the problem in a more precise way. Between two intermediate points there exists more than one trajec-

tory which constitutes a solution to the problem, however, not all of these trajectories are feasible. That is, the robot must not only reach the target, but it must also avoid colliding against the walls surrounding the football field or against other robots. Obviously, this fact could affect the number of arcs (or the number of decisions to be made) along the trajectory. To reflect these constraints, we have to adequately define the cost function, taking into account not only the distance between the target and the current coordinates, but also the potential wall and robot collisions. However, it is necessary to incorporate some additional constraints into the analysis. On the one hand, we have to limit the maximum and minimum velocities of the robots, in order to avoid solutions which do not have any physical meaning. Therefore, the cost function will have to have complete information about the system state. On the other hand, we defined a constraint to avoid sudden changes in the velocities corresponding to neighboring arcs. That is, we want the velocity changes (i.e. the estimation of the acceleration) not to exceed a maximum value, which would otherwise produce undesirable effects, such as wheel slipping.

Regarding the length of each trajectory, we added an additional variable to the cost function to deal with this matter. Briefly, the non-linear optimization method searches for a common arc interval, which satisfies the existence of a minimum value for the function being evaluated. Clearly, there are at least two alternatives which would allow us to to cope with the arc elapsed time. The first one is to define, in advance, a fixed time interval. In this case, we would be fixing the number of arcs in the solution. The second one is to define an additional variable for each arc in the trajectory, however, in this case, we are increasing the number of variables involved in the problem.

Again, we had to add constraints to avoid the optimization method's giving solutions with non-acceptable time intervals. If the time interval for each arc is shorter than the control cycle, the robot cannot change its velocities in time. If the time interval is too long, the solution is meaningless in the context of the game.

A basic expression of the cost function is

$$Cost = \sum_{j=1}^{NIP} (r\_robot\_collision_j + r\_ball\_collision_j + \\ + r\_acceleration_j + r\_wall\_collision_j),$$

where j indexes the arcs along the trajectory and *NIP* defines the number of intermediate points.

The $r\_robot\_collision_j$ term corresponds to the cost of colliding with one or more robots along the trajectory depicted by the arc j. Once the velocities of the arc and the time interval are defined, then the partial trajectory j for the robot can be included in the cost function, and the cost component is computed following geometrical criteria. A similar situation happens for the $r\_ball\_collision_j$ and $r\_wall\_collision_j$ terms in the sum.

The $r\_acceleration_j$ term checks, at each intermediate point, whether the change in the velocities exceeds a bound defined in advance. If this were to be the case, an additional cost is added to the function value.

Finally, constraints on the time intervals and velocities of the robots are added using a transformation (a bounded function with first and second derivatives in its entire domain) between the real space and a region in which the values have physical meaning.

4.3. The number of intermediate points

Perhaps the main difficulty encountered when using optimization methods (OM) to find trajectories is having to set the required number of intermediate points (*NIP*). *NIP* estimation is not easy to perform and we had to consider the use of Artificial Neural Networks (ANN) to satisfactorily handle most of the cases.

Although the elapsed time used by the OM is really too short, it strongly depends on the number of variables included in the cost function and therefore, on the *NIP* value being used. However, as the problem is a real time one, the robot would need to decide which action to perform before the vision system refreshes the system state. In this way, the maximum *NIP* to be used is constrained by the speed of the processor in the computer platform. The problem is that the optimization algorithm cloud not generate a solution for the *NIP*s which is lower or equal to the maximum *NIP*. In practice, this does not represent a major problem, since those solutions, for which the value of the cost function is not equal to 0, tend eventually to approach a solution through the repeated calls to the OM over time. That is, each time the vision system refreshes the system state, a new call to the OM is made, but for a new system state, which corresponds to a cost function with a lower value than that of the previous one.

On the other hand, the repetition of the OM calls allows the robots to search for solutions which take into account the changes occurring within the system. We call this the heuristic, incremental optimization method.

Unfortunately, the value of *NIP* is unknown a priori. That is, there is no analytical way of knowing, in advance, whether the optimization method will lead to a solution (with a null cost value) for a given *NIP* value. We tackled this problem using two alternative methods. The first method is simple and exact, and consists of testing the optimization method with incremental values of *NIP*. The second one involves estimating the *NIP* value.

To implement the first alternative, we must take care with the elapsed time necessary to carry out each

call to the optimization algorithm. For this reason, we implemented these calls in parallel using a multithread technique on a dual processor platform. This is possible because each call is independent of the other ones. In this way, if there exists a solution with a small *NIP*, the remained threads are cancelled as soon as the solution is obtained.

The second alternative involves estimating the NIP for a given problem (i.e. system state and target coordinates) using ANN. To train the ANN we defined 4 training sets of 5000 examples each (noted ANN1, ..., ANN4). The input components represent the system state and target coordinates. Each training set has a different output component. For the first training set, the output value was 1.0, if the *NIP* used to call to the OM (and it found a solution with null cost) was 1. For the second training set, the output was 1.0, if the corresponding *NIP* was 2. The third training set was constructed in a similar manner. For the fourth training set, the output was 1.0, if the corresponding *NIP* was 4 or more. That is, we used ANN as an oracle.

To obtain the input part of each example, we set each one of the system variables randomly following a uniform distribution.

To obtain the output part, we call the optimization method incrementally, until the correct *NIP* (i.e. the one that allows the optimization method to find a null value for the cost function) is obtained.

The ANN used was a multilayer perceptron with only one hidden layer and an output neuron [1]. Both desired outputs and inputs were scaled to the range [0.1].

Once the ANN was trained, we tested the networks with independent testing sets of 1000 examples each (created in a similar manner to the training set). We obtained a generalization percentage of at least 75 %. That is, we obtained the correct *NIP* for at least 750 examples from a set of 1000 examples.

During a game, when the robot needs to call the optimization method, it asks each ANN to forecast the correct *NIP*. That is, it asks ANN1 if the correct NIP is 1, ANN2 if the correct NIP is 2, and so on. Ideally, only one ANN produces an output of 1.0, while the remaining ones give 0.0. In this case, the robot calls the optimization algorithm with an *NIP* equal to the ANN index whose output was 1.0. In the case where all of the networks give an output of 0.0, the robot will use an *NIP* value equal to 4, and in the case where more than one network gives an output of 1.0, the *NIP* used by the robot will be the maximum value of the *ANN* index whose output was 1.0 (if more than one network has an output value not equal to zero, we choose the *NIP* that favors the existence of a solution with minimum cost).

5. CONCLUSIONS

In this study, we introduced our approach to dealing with the robot soccer problem for a team of 5 mobile robots. We identified and depicted a set of 7 individual behaviors, which define the activity of each robot during the game. The exception was the goalkeeper, which has a different definition. The basis of our approach is to obtain a cooperative behavior, which emerges from homogeneous sets of individual behaviors that are activated when particular preconditions are satisfied. We also presented a set of 10 elemental preconditions and described how these are combined to obtain each behavior precondition.

To evaluate each precondition we used predictors. These predictors are based on models or statistical and algorithmic methods. Additionally, we used a nonlinear optimization method, which minimizes a cost function in order to find the optimum trajectory during the game. Since robot soccer is a real time problem, we had to use heuristic and algorithm parallelization techniques, in order for the optimization technique to be able to provide a solution in time. The heuristic part was based on Artificial Neural Networks that were used as an oracle, in order to set a parameter value used in the optimization procedure. Although our method produces a trajectory plan for each control cycle, the dynamic characteristics of robot soccer result in only the first step of each plan being used. This results in behaviors which appear more reactive than deliberative.

Much work remains to be done. Basically, our next step will be based on the intensive application of learning that, as was mentioned in the introduction, will likely lead to a myriad of research challenges.

REFERENCES

[1] D. E. Rumelhart and J. L. McClelland, *Parallel Distributed Processing,* vol. 1, MIT Press, Cambridge, 1986.

[2] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *Computer. Jounal,* vol. 7, pp. 155, 1964.

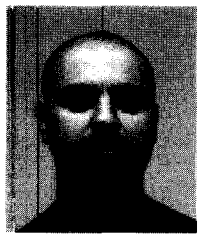
[3] P. Maes, and R. A. Brooks, "Learning to coordinate behaviors," *AAAI,* Boston, MA, pp. 796-802, 1990.

[4] W. Li, D. Naik and J. Swetits, "A data soothing tchnique for pecewise convex/concave curves," *SIAM Journal on Scientific Computing*, vol. 17, Number 2, pp. 517-537, 1996.

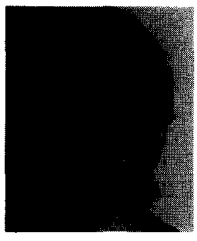
[5] M. H. Wright, "Direct search methods: once scorned now respectable," *Proc. of the 1995 Dundee Biennial Conference in Numerical Analysis*, E. D. F. Griffiths and G. A. Watson, Addison Wesley Longman, pp. 191-208, 1996.

Santos, Juan Miguel is an Assistant Professor at the Departamento de Computación, Facultad de Ciencias Exactas (FCEN) y Naturales of the Universidad de Buenos Aires (UBA). He obtained his Ph.D. degree in Computer Science in 1999, from FCEN-UBA and (DIAM-IUSPIM)-Universite de Aix-Marseille III, France.

Scolnik, Hugo Daniel is a Full Professor at the Departamento de Computación, Facultad de Ciencias Exactas y Naturales of the Universidad de Buenos Aires. He received his Ph.D. degree in Mathematics in 1970, from the University of Zurich, Switzerland.



Daicz, Sergio is a Teaching Assistant at the Departamento de Computación, Facultad de Ciencias Exactas y Naturales of the Universidad de Buenos Aires. He graduated in 2000, obtaining the degree of Licenciado en Ciencias de la Computación (M.Sc) from FCEN-UBA.



Laplagne, Ignacio is a Teaching Assistant at the Departamento de Computación, Facultad de Ciencias Exactas y Naturales of the Universidad de Buenos Aires. He graduated in 2002, obtaining the degree of Licenciado en Ciencias de la Computación (M.Sc) from FCEN-UBA.

Scarpettini, Flavio was graduated in 2002, obtaining the degree of Licenciado en Ciencias de la Computación (M.S.) at the Departamento de Computación (FCEN-UBA). He is member of the Computational Intelligence applied to Cooperative Robotics Group.

Fassi, Héctor was graduated in 2002, obtaining the degree of Licenciado en Ciencias de la Computación (M.S.) at the Departamento de Computación (FCEN-UBA). He is a member of the Computational Intelligence applied to Cooperative Robotics Group.

Castelo, Claudia was graduated in 2002, obtaining the degree of Licenciada en Ciencias de la Computación (M.S.) at the Departamento de Computación (FCEN-UBA).