# A Mobile Robot Behavior Based Navigation Architecture using a Linear Graph of Passages as Landmarks for Path Definition

LUBNEN NAME MOUSSI and MARCONI KOLM MADRID
DSCE – FEEC
UNICAMP
Av Albert Einstein, 400 – Cidade Universitária Zeferino Vaz, Campinas, SP
BRAZIL
lubnen_moussi@hotmail.com marconimadrid@uol.com.br

*Abstract:* - In this work the authors present a mobile robot architecture design for real time autonomous navigation considering minimum requirements of hardware and computation. These requirements determine the use of simple sensors and minimum representation of the environment. The approach chosen was Behavior-Based for the design of the architecture and the behaviors. It is implemented under simulation of a differential wheels robot inside a 2D environment with a layout very close to a real situation for laboratories, offices and classrooms. The solution takes into consideration local and long run navigation. A first approach utilizes only reactive behaviors, solving very well local navigation. It also gives a partial solution to long run navigation in a simple environment, with only two rooms, doing it without path definition. A complete solution for an environment with many rooms is developed adding to the previous approach more behaviors that will take care of path definition and control. A linear graph, having the passages as landmarks, structures the environment representation and is the basis for the algorithm of path definition that gives an efficient solution. The architecture has not a central planner and controller as in traditional deliberative architectures; planning and control arises from the independent parallel functioning of all the behaviors. The results demonstrate that the design achieved its objectives and some important points to improve it are shown.

*Keywords*: - autonomous navigation, mobile robotics, behavior-based robotics, emergent behaviors, path definition and control, target seek.

## 1 Introduction

Considering a mobile robot, the classical solution to provide it with the capability to perform autonomous navigation involves detailed representation of the environment and powerful computation. The good point of this approach is the precision that it gives to all the actions of the robot, from setting the speed and obstacles avoidance to path definition and control. But, it requires expensive navigation hardware and computation, and real time responses are difficult to obtain in a changing environment. Not always it is of relevance to have a precise solution, and there are situations were real time performance might be more important. That is the case with an autonomous robot that has to run in an environment where the position of the obstacles is changing, and with human beings inside of it. In this case, precision is not the most important point. Of course, if the task is to find the target and to peek it up and hold it, the navigation to get near the target is not required to use the optimum and precise path, but the mechanism to peek and hold the target requires it. This work deals with navigation, that is, the architecture design does not require the solution defined to be the best one, but a good and feasible one.

In this work it is shown the implementation of a differential wheels robot efficient intelligent architecture for performing real time autonomous navigation in a simulated two dimensional environment for labs, offices and classrooms.

It uses Behavior-Based Robotics (BBR) and the solution is presented in two steps. The first is a simple architecture based only on reactive behaviors, without utilizing representation, and that solves navigation in a simple environment. The second gives solution to a more complex environment, utilizing the prior one for local and immediate requirements and adding new behaviors to allow long run path definition and control. A linear graph, having the passages as landmarks, structures the environment representation and is the basis for the algorithm of path definition.

The results demonstrate that the design achieved its objective. Some important points under consideration to improve it are shown.

Section 2, Architectures for Autonomous Navigation, is a very brief introduction to Behavior-Based Robotics directed to the reader not well familiarized with it and also presents references for them to get deeper into this subject. In Section 3, Choosing the Architecture, the authors show why they elected Behavior-Based Robotics as the architecture for their project of autonomous navigation and why, besides their previous experience with Evolutionary Robotics, they decided to implement the behaviors following the Behavior-Based approach. The Simple Architecture is presented in Section 4, where it is shown how local navigation is solved by it, the details of the behaviors that make the architecture and the result of its functioning. Section 5, The Simple Architecture in a Complex Environment, explains how the oscillations are originated giving some examples, shows also that the solution given by this architecture in a complex environment is quite often obtained after dangerous oscillations and, finally, how it performs well environment exploration. The Complete Architecture design is detailed in Section 6 with an overview of how it was obtained, the implementation of the linear graph of passages, the algorithm of path definition and the results. Conclusions and perspectives are given in Section 7.

## 2 Architectures for Autonomous Navigation

The classical approach to mobile robot autonomous navigation requires detailed representation and actualization of the environment with the objects inside. Usually, the layout of the environment is memorized previously and the objects are frequently detected and updated in the representation. Also, a deliberative or planer based centralized architecture running very sophisticated computer power consuming algorithms is in charge of finding the right path to the target. There are other alternatives. The state of the art architectures [2], [10] to solve autonomous navigation can take, in a general and simplified overview, three distinct alternatives: deliberative, reactive and hybrid. Behavior-Based Robotics (BBR) is located somewhere in between Reactive and Deliberative architectures.

### 2.1 Deliberative or Planner-Based Architecture

Deliberative or planner-based architectures have a centralized nature and are highly dependent on internal representation. Sensor data are fed and analyzed, in each step, to determine the proper action, being it to deviate immediately from an obstacle or to proceed according to a re-planned path to a long run goal. It gives precise definition for the actions, but the process takes expressive time and will usually work only in very well controlled environments.

### 2.2 Reactive Architecture

A Reactive Architecture is characterized by linking tightly sensing to action and, because of that, being able to give quick responses. It achieves real-time performance, in contrast with the Deliberative Architecture difficulties to accomplish it. But, in the other hand, it only solves immediate purposes of the robot, like object avoidance. Its quick response is due mainly to the non use of environmental modeling, but instead, using the environment itself as the model, that is, the robot senses and acts upon the environment directly. This architecture normally does not use state and bases its functioning on a mapping between stimuli and appropriate responses. It can be noticed also that the reactive architecture uses little reasoning, while deliberative needs a great amount of it.

### 2.3 Hybrid Architecture

First researches in artificial intelligence were mostly deliberative. More recently, in the last decade of last century, BBR, using reactive behaviors, started solving many of their unsolved problems, related mainly to safe locomotion, environment exploration and target seek. One way to take advantage of these two separate and quite opposite techniques is by a hybrid architecture, where a reactive architecture takes care of short run requirements and medium and long run planning are left to a deliberative architecture. This architecture requires usually three layers: Reactive, Deliberative and Intermediate or Supervisory layer, which will take care of conflicts and integration.

### 2.4 Behavior-Based Robotics

BBR is located somewhere in between Reactive and Deliberative architectures. Besides the basic behaviors being mostly reactive, BBR can also use state and internal representation, being able to deal with immediate, medium and long run goals.

A behavior can be viewed as a procedure or law that performs an action given a condition. For instance, given an object in the near frontal vicinity to the left and none to the right, turn right is an action taken by a behavior of obstacle avoidance.

Behaviors are implemented through hardware ore software. With hardware, they are distributed and do not require synchronous functioning. With software they are easily defined using decision structures like *if <condition> then <action>;* for instance, *if <front left sensor sees an obstacle in a non accepted proximity and the front right sensor is free> then <turn*

*right*>.

BBR has a decentralized nature, it is a distributed architecture composed by behaviors that work in parallel, that is, each behavior is fed with sensor data and all the behaviors are processed in each cycle. A behavior can also receive input from other behaviors. Examples of basic behavior for a mobile robot are to avoid obstacles, to follow walls and to seek a target. All of them are fed with sensor data and processed in parallel. Note that they could give conflicting results, so there must be a conflicting disambiguation mechanism for solving it. This mechanism, depending on the case, can be a hierarchical definition or some combination of their results.

Behaviors in BBR are designed and implemented incrementally. For example, someone design the behavior 'deviate' to accomplish obstacle deviation. If it is not the first one, he introduces it in the system and implements the procedure to disambiguate possible conflicts with the prior ones. Doing so it is possible to develop and test each behavior to get its right design, independently of finishing the entire project to get it. A new behavior introduced can take advantage of the previous ones. For instance, introducing a behavior for wall following, it can stand without dependencies and take advantage of a previous behavior for obstacle avoidance working in parallel.

The design follows a bottom-up procedure resembling biological evolution in its incremental refinements. More complex behaviors are usually designed indirectly, obtained as an emergent result of the combination of some of the simpler ones. In some cases the emergency is predictable, in some not. In case the emergency is not welcomed, it is required redesign to solve it.

The project of an intelligent control using behaviors is done, most of the time, in a trial and error basis. This affirmation is valid when designing the behavior and also when adjusting its parameters.

Usually, a simple and basic behavior is reactive. A number of attitudes can be solved by reactive behaviors, as can be seen in nature and robotics [4], [5]. But a behavior can have state and environment representation can be implemented by behaviors. In [10] representation is accomplished in a distributed manner, in a network of behaviors. There is a strength performed by BBR researchers to develop more complex behaviors, even social [11], utilizing a combination of base behaviors and learning that gives rise, or emergence, to more sophisticated ones.

By now, what is certain is that BBR is becoming common sense for immediate attitudes of the robot, and even for some medium and long run tasks like target seek. For more complex problems the way to take is highly dependent on its nature and on the researcher's preferences, there are not yet general rules to decide which way to take, purely BBR or hybrid [1], [2].

The following references are for the reader interested in getting closer to BBR. Arkin [1] gives a comprehensive view of Behavior-Based Robotics. Applied perspectives with relevant points related to design are found in Mataric [8], [10], and [11]. *Autonomous Robots* [2] dedicates a chapter to architectures, including BBR. *Robot Programming* [7] is a simple practical guide. *Flesh and Machines* [6] gives a historical vision of artificial intelligence and robotics in general.

# 3 Choosing the Architecture

## 3.1 Choosing the architecture

Real time requirements, and minimum navigation hardware and computation determine that the Deliberative Architecture is not a good choice. A Hybrid Architecture might be a choice, but the authors decided to try a complete BBR Architecture. This decision was reinforced by the work developed by Mataric in [8], [9] that offered solid evidences of an easier design and turned to be a sound inspiration for their work.

Designing the architecture under Behavior-Based Robotics requires the identification of the actions that the robot has to perform to accomplish its task and configure them into behaviors.

## 3.2 Implementing the behaviors

After defining a behavior, its implementation under the Behavior-Based approach can be done by software using mainly *if* ….. *then* …. constructions. Another way for implementing the behavior is utilizing evolutionary robotics. In this way, previous work of the authors in evolutionary robotic for obstacle avoidance [17] [18] [13] utilizing classifiers systems [3] led them to consider it for autonomous navigation. However, they knew by experience that it would be a hard work and so, prior of starting in that direction, they decided to look for related work in the literature, what confirmed their expectation. For instance, difficulties are shown in Mataric [12], and in Nolfi and Floreano [19] the solutions are presented for very simplified environments with evident great effort in design.

Therefore, the authors decided to design the architecture and to develop the behaviors using the Behavior-Based approach, and found out that it was rather simpler for them.

### 3.3 Solution in two steps

It looks reasonable to organize the work of designing the architecture dividing it in smaller parts. There are three evident indications to suggest the division of the design in steps: reactive behaviors versus non reactive behaviors, no environment representation versus minimum environment representation, and local navigation versus long run navigation. These indications direct the design almost naturally to two steps. First, one architecture with only reactive behaviors, what also means not representing the environment, and with the task of solving local navigation. Second, another architecture would take advantage of the prior one and add new behaviors for minimum environment representation and long run path definition.

However, there are other ways to look to the indications, one of them being to consider one architecture with only reactive behaviors and see how far it can solve our task, adding, afterwards, non reactive behaviors do deal with the unsolved part. The authors decided to try this alternative and the result was that it was possible to solve local navigation with it and to go a little further solving partially long run navigation.

Therefore, the first step is an architecture built with reactive behaviors, without representing the environment, which solves local navigation and, partially, long run navigation. This architecture is named in this work The Simple Architecture.

The second step is an architecture built over the Simple Architecture to enable solution to long run navigation in a more complex environment. Thus, it utilizes the prior one for local and immediate requirements and adds new behaviors to allow long run path definition and control.

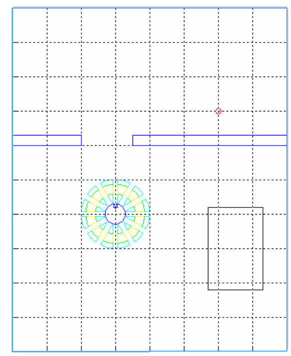
## 4 The Simple Architecture

This architecture uses only reactive behaviors, without memory and environment representation and was first presented in Lubnen and Madrid [14]. It is designed to solve navigation in a simple environment and to give a partial solution to it in a more complex environment.

### 4.1 The robot and the environment

Figure 1 shows a differential wheels robot, a blue circle. It has a circular shape, having 60 cm of diameter, with a ring of 12 proximity sensors uniformly distributed, and a bearing sensor. In the figure, the robot is surrounded by a drawing of its proximity sensors zones and its heading is shown by the small blue circle. The environment is 8 x 10 m, with two rooms and one of them has a table. The target is represented by a small red circle.

The data of the proximity sensors are classified in four zones: *danger*, *safe*, *edge* and *out*. The danger zone corresponds to proximities that the robot has to avoid. The safe zone is used to align to the surface of an object. The robot avoids objects in its forward direction edge zone. Out corresponds to the inexistence of near object.



Grid 1 x 1 m only for measurements

**Fig. 1:** Robot and Environment

### 4.2 An overview of the solution

Looking for basic behaviors it is worth to observe that in nature and robotics a basic behavior is reactive [4], [5]. The Simple Architecture utilizes only reactive behaviors and considers the environment itself as its representation. Consequently, there is not the burden of environment representation as in the classical approach.

For defining the behaviors to build the Simple Architecture we have to understand what are the required actions that the robot has to perform to accomplish its objective. The robot has only proximity and target direction sensors, so, through sensing the environment, it can perform actions of running or stopping, obstacle deviation and target seek. But, even in local search, these actions do not guarantee that the robot reaches the target because any obstacle in its path will create a great difficulty, many times unsolvable.

To solve it we can imagine that the robot, instead of deviating from the obstacle in its path to the target, aligns with it and follows its contour until finding a free heading to the target. That is, defining a behavior to align with the contour of the obstacle, or, in general, a behavior to align with the surface of an object when getting close to it.

To deviate, to align and target seek are quite often conflicting actions that require a manner to deal with that will be shown in the following sub-section.

When we provide a robot with a behavior of aligning with objects near to it actually we are giving something else. This attitude leads the robot to perform wall following, what permits it to reach a passage and to get inside the other room. It means that a robot inside one environment with many rooms and passages is able, with behaviors to deviate and align, to explore it with safety. It also means that we are solving local target seek and, at some extent, long run target seek.

Looking to this fact in terms of dynamic systems, the solution of the non-linearity caused by an obstacle in the path of the robot to the target given by the behavior align gives also solution to the nonlinearity caused by a wall in the path of the robot to the target in another room. That is, the solution to local nonlinearity of obstacles gives also solution to nonlinearity of walls.

It has to be noted that the solution of the conflict raised by deviate, align and target seek when dealing with the nonlinearity can generate oscillations. These oscillations are increased as the complexity of the system is increased, so, it is expected that local navigation can be well solved by this architecture and long run navigation will be very dependent on the layout of the environment.

Hence, we have an architecture that solves local navigation and is able to perform environment exploration with safety and that solves also, at some extent, long run navigation.

Let us see in more detail through an example the effect of nonlinearity in the case of long run navigation. In Figure 2, we see the robot in one room and the target in the other room, without a clean straightforward path linking the robot to the target. The robot does not know where it and the target are. Its proximity sensors just show the obstacles. The bearing sensor tells the target direction. There is no information to compute a path to the target. The robot has to find a way to go to the passage, pass through it, get inside the other room and run straight to the target. With the behavior align implemented, when the robot is running in the direction of the target and reaches the wall it has the chance to align with it, to achieve the passage, to enter inside the other room and to solve target seek.

However, this architecture gives only partial solution to long run navigation. Consider again the robot in Figure 2 running in the direction of the target and that when reaching the wall it turns to the left to start the alignment. It will eventually align with all the walls in that room, running counterclockwise, until it reaches the passage. So, it becomes clear that depending on the complexity of the layout the robot can take a very long path to reach the target and even be unable to reach it.

Besides being partial, the long run solution with the Simple Architecture is very surprising. Just remember that it is obtained with only reactive behaviors, what means that there is no map to find the target.

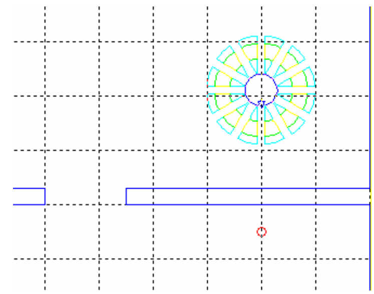


**Fig. 2:** Non-Linearity

Until now we saw the actions of deviating, aligning and target seek directing the robot to solve its mission. These actions define the behaviors: Deviate, Align and Seek. But, it is also required the actions to start the robot in a forward direction and to stop it that will be done by the behavior Forward. Actually, Forward have more actions to perform that help the robot to run safely in the environment. The details of the functions performed by each behavior and the solution of the conflicts generated by their parallel functioning are shown in the next sub-section.

### 4.3 The architecture

The behaviors used are reactive and similar to the basic ones utilized by Mataric [8], but they are not equal. The architecture is shown in Figure 3.

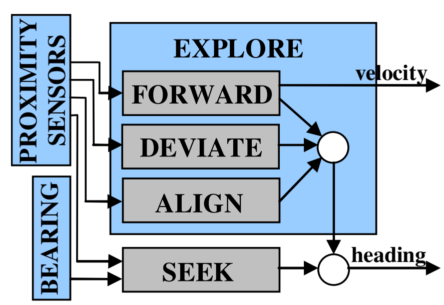



**Fig. 3:** Architecture

All behaviors are processed in each cycle. Hierarchical attributes and exclusive conditions are defined to solve conflicts. The data of the proximity sensor classified in danger, safe, edge and out is delivered to all the behaviors. Only Seek gets the output of the Bearing sensor. Only the behavior Forward sets the velocity.

The details of the behaviors actions and the conditions in which they happen as well the priorities defined to solve conflicts are shown in the following sub-sections.

#### 4.3.1 Forward

The behavior Forward is in charge of setting the robot to run when it is in a free direction and to stop it when it is in danger to collide with an obstacle. The decision to stop the robot is taken when any of the proximity sensors shows an object in the danger region and any front sensor shows an object below the edge region. And, once it stops the robot, this behavior makes it to turn around itself until it finds a direction free of obstacles to go. The result of this occurrence is a sharp deviation. Failing to find a free direction to go, the robot is moved backwards for a while in an arbitrary safe direction.

Forward is the only behavior that sets the velocity. It is the principal behavior responsible for security; it protects the robot and the environment and that is why it receives the maximum priority.

#### 4.3.2 Deviate

Deviate is inserted incrementally over Forward and works in parallel with it. It avoids obstacle entering the front edge region by changing the heading of the robot. If the obstacle is in the left side the robot turns to the right and vice versa. If the object is straight in the front of the robot it turns randomly left or right. The result of the action of Deviate is smoother turnings then those made by Forward.

This behavior is also responsible for the robot security and diminishes the need of the use of Forward. Deviate has lower priority then Forward. That is, if the results of the Proximity Sensors are such that Forward and Deviate are able to give output, what means that they are activated, the Simple Architecture will accept the output of Forward and ignore the output of Deviate.

#### 4.3.3 Align

Align is inserted over, runs in parallel and has low priority then the above ones. Its goal is to maintain the robot in the proximity of the object to which it gets near. It tries to maintain the object in the safe zone. If any of the lateral sensors detect an object in the safe zone, this behavior is activated and its action brings the robot to the vicinity of the object that it is getting near to or escaping from. When the robot reaches the edge of a wall that it is following, like in a passage, it turns around it, getting inside the other room. Align is responsible for solving nonlinearity produced by obstacles and walls.

This behavior relies on Forward and Deviate to maintain the safety. It has lower priority then both of them. That means that its output will be accepted by the Simple Architecture if either Forward or Deviate is not activated.

#### 4.3.4 Explore

The behaviors above, Forward, Deviate and Align give rise to the behavior Explore that emerges from their parallel execution. It happens because the behavior Align allows the robot do keep wandering, and so, to explore the environment. At the same time, Forward and Deviate guarantees that the exploration is done under safety.

#### 4.3.5 Seek

Seek is added to the architecture over the other behaviors and runs in parallel with them. The robot senses the bearing of the target and determines the heading to go straight there. Seek checks if the robot is free of obstacle in the direction of the target. If it is, Seek overwrites the heading given by Explore.

### 4.4 Results

The simulation for obtaining the results is run in the environment of Figure 1. The robot has an axis of 60 cm of diameter. The behaviors Forward, Deviate and Align, utilize 30 degrees for each deviation. The following parameters had to be adjusted to get the results and the adjustments were done by trial and error:

- Velocity = 20 cm/s, is the robot speed set by Forward.
- Iteration interval = 1.2 s, is the time interval allowed for each iteration of the simulation. It has to be sufficient for the time required for all sensors readings and outputs, low level control, and also for computation.
- Maximum distance to detect an object = 70 cm. It relates to the end of the edge region. Adjusting this value, automatically adjusts the dimensions of the safe and danger regions.
- Bearing Sensor aperture = 90 degrees in the heading direction. This aperture can be understood as the field of view of the robot, that is, in which directions it can see the target.

In Figure 4 the Forward behavior is working alone and the robot trajectory shows sharp changes in orientation when an obstacle gets inside the danger region and the front sensors shows an object below the edge region. The robot stops and keeps changing its orientation until the front proximity sensors are free, when it runs again. One point to observe is that the robot stays most of the time inside one of the rooms.

Figure 5 shows the effect of Forward and Deviate working together. Most of the changes in orientation are smoother then in Figure 4. There are still some sharp deviations due to the action of Forward. There is not yet evidence of exploring efficiently the entire environment.

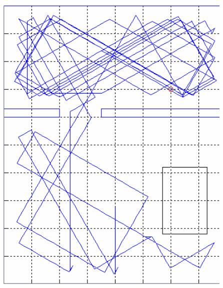


**Fig. 4:** Forward

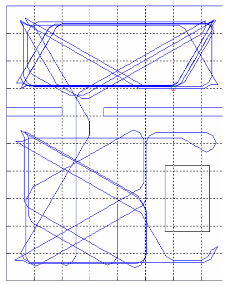


**Fig. 5:** Forward and Deviate

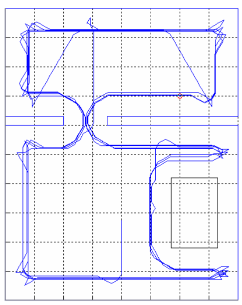
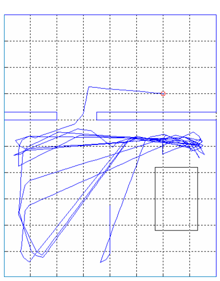


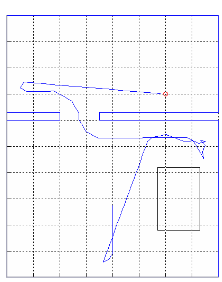
**Fig. 6:** Forward, Deviate and Align - Safe Exploration

Figure 6 shows the robot exploring safely the environment under the work of Explore that emerges from the parallel functioning o Forward, Deviate and Align. It is noticeable the effect in aligning with the objects of Align, the smooth changes in orientation of Deviate and some stops and sharp changes in orientation of Forward. And it is clear that the robot wanders throughout the entire environment.



Target is reached after 495 iterations

**Fig. 7:** Target Seek without Align



Target is reached after 108 iterations

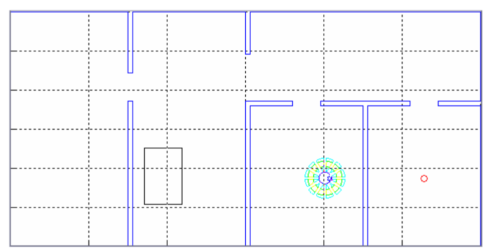
**Fig. 8:** Target Seek

Figures 7 and 8 show the Simple BBR Architecture performing Target Seek. In Figure 7 Align is turned off, and the robot reaches the target after 495 iterations, seeming almost left to chance. In Figure 8, Align is turned on and it shows its relevance in the solution of the nonlinearity caused by the wall. The robot achieves the target after 108 iterations and does not go straight to the target; however, it goes under a good and safe path.

Therefore, it becomes clear that the Simple Architecture achieves its objective of solving local navigation. It also gives solution to long run navigation in a simple environment, but, as expected, it has to be improved. This improvement is possible with the Complete Architecture.

## 5 The Simple Architecture in a Complex Environment

It is relevant to verify the Simple Architecture working in a more complex environment to understand better the kind of difficulties that appears and how far it can help to solve them. In Figure 9 the environment is 20 x 10 meters and shows the robot inside a room separated by a wall without a passage to the other room were the target is located.



Grid stands for measurements

**Fig. 9:** Robot and environment

This environment is more realistic then the one in Figure 1, it has more rooms and the passages are narrower. The parameters had to be adjusted to enable the architecture to perform environment exploration and were set to the following values for the simulations in this section:

- Velocity = 15 cm/s
- Robot axis = 50 cm (as the passages are narrower it is better to use a smaller robot)
- Iteration interval = 0.6 s
- Maximum distance to detect an object = 60 cm

The aperture will be set to different values in this section.

## 5.1 Oscillations

Oscillations occur due the conflict between the behaviors Explore and Seek and are also dependent on the aperture of the Bearing sensor. The conflict arises because Explore drives the robot making it align or deviate, while Seek attracts it to the target.

## 5.2 Maximum aperture

Figure 10 shows one example using 360 degrees of aperture for the Bearing sensor. With this value the target detection will happen independently of the robot orientation. The robot starts running in the direction of the target until it reaches the wall. Then, Deviate makes a deviation to the right. In the next cycle, Align enters in operation aligning the robot with the wall and, while it is aligned with it, there is not a free way in the target direction. When the robot reaches the corner, Deviate makes a right turn and the robot moves a little bit further leaving back the wall it was aligned with. So, it has a free path almost backwards in the direction of the target, and Seek is activated. The robot moves in the direction of the target, reaches the wall, and aligns with it in the opposite direction it did before. When the robot reaches the other corner it will enter a similar process. In the example in Figure 10, the upper corner provides more visual details of what happens when the robot reaches the corner, makes a left turn, runs further a little bit, finds a free way to the target, moves in the direction of the target and aligns with the wall.

In this example the robot cannot escape from the oscillation. Total aperture should not be used because it gives a high probability of oscillation of this nature.

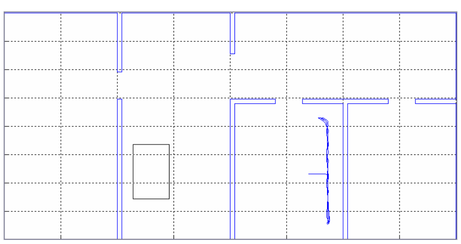


**Fig. 10:** Oscillations with 360 degrees for the aperture of the Bearing sensor.

## 5.3 Minimum aperture

Figure 11 shows one example with low aperture where it is set to 30 degrees. This aperture means that the robot senses the target within 15 degrees to its left side and 15 degrees to its right side.

The attraction becomes very much dependent on the robot orientation. When the simulation starts the robot runs in the direction of the target until it reaches the wall. Then, Deviate makes a right turn and after this point, in the entire trajectory shown in the figure, the robot cannot detect the target.

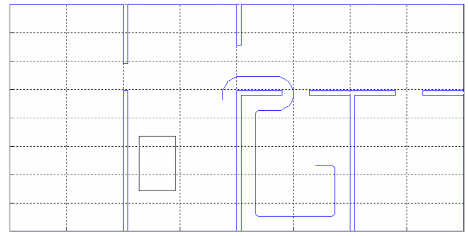


**Fig. 11:** Trajectory with 30 degrees for the aperture of the Bearing sensor.

The robot may be entering a low frequency oscillation, returning back again after a long run due mainly to environment exploration. Eventually, with low probability, the robot will achieve the target.

A minimum value might not be a good choice for the aperture because it provides very low possibility of attraction by the target.

## 5.4 Medium aperture

The example in Figure 12 has the aperture set to 180 degrees. What happens is that the robot is not oscillating as with the maximum aperture and does not ignores almost completely the target as in the minimum. The robot is going away from the target, and as the attraction is higher it might have more probability of returning and finding the target. But, it is not guaranteed. Medium aperture, or a value near it, looks more appropriate to help in solving the nonlinearity of a complex environment.

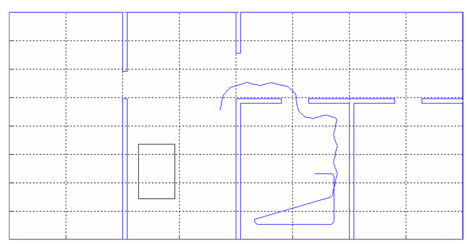


**Fig. 12:** Trajectory with 180 degrees for the aperture of the Bearing sensor.

## 5.5 Target seek

In Figure 13 the robot starts in the first room in the right side. The parameters are the same as used in the previous examples. The aperture is set to 180 degrees. The Simple Architecture is not designed to solve target seek for this environment. The idea here is to see what happens letting the simulation to run. The graph in the figure shows the distance to the target, putting in evidence the oscillations. The robot is almost kept imprisoned in the room near the room were the target is. But, it escapes and finds the way to the target.

It is clear the effect of the nonlinearity of walls causing oscillations and great difficulty or even invalidating target seek.

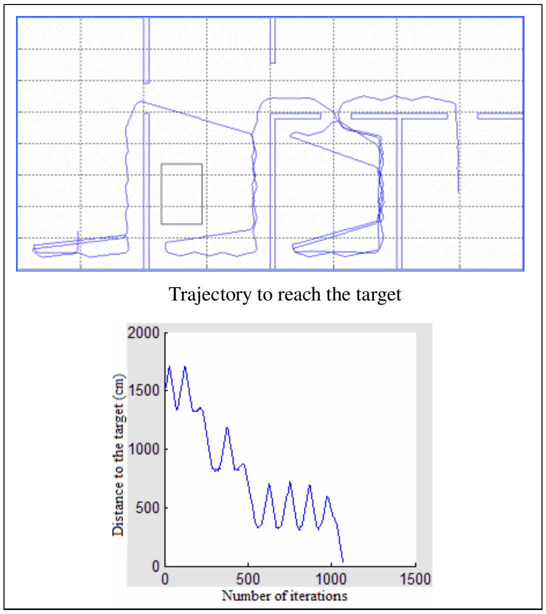



**Fig.13:** Result for the Simple Architecture

### 5.6 Environment exploration

The Simple Architecture is expected to perform environment exploration with safety, and does it well, as can be seeing in Figure 14.

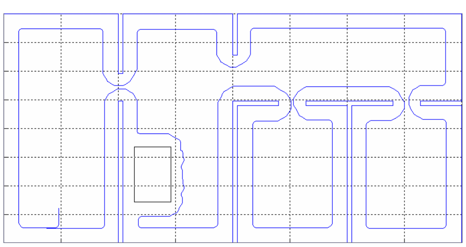
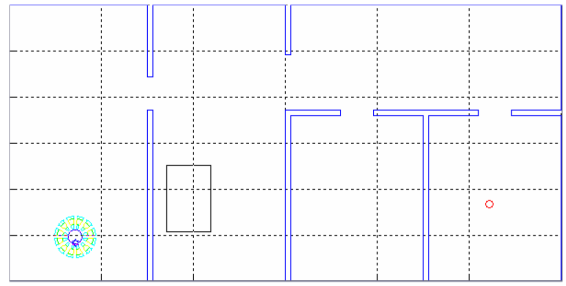


**Fig. 14:** Complex environment exploration

## 6 The Complete Architecture

This architecture is designed to give an efficient solution to autonomous navigation performing real time target seek in a simulated 2D environment with many rooms and passages connecting them. It utilizes only reactive behaviors for local navigation, as done previously, and adds new behaviors to take care of identifying where the robot is and to define the path [16]. The environment in Figure 15 is 20 x 10 meters. The robot is in the first room in the left, facing the wall near to it. There is a table in the second room, and the target is in the last room.

The robot characteristics are similar to the one used for the simpler environment. Now, it utilizes sensors for getting its position and orientation, and does not use the bearing sensor anymore.



Grid stands for measurements

**Fig. 15:** Robot and environment

### 6.1 Overview of the solution

The first ideas of the authors to solve efficiently autonomous navigation were presented in Moussi and Madrid [15]. There are two points to solve to guarantee a more efficient target seek. One is that the nature of the design used here and in most of the approaches based on intelligent control does not lead to an optimum solution. However, it is better to have the possibility to take a shorter way to the target, then the one shown in Figure 8. The other is the possible dangerous increase in oscillation, even invalidating reaching the target, due to the conflicts caused by Seek with Explore**,** mainly when the difficulties of the environment scales up, like in Figure 15.

One way to solve them is using landmarks that the robot can utilize for its navigation. The landmarks have to be of easy identification by the robot and conveniently placed to facilitate the path definition. There are a number of alternatives, but, passages solve the non linearity given by walls between the robot and the target. Using passages as intermediate goals, and building a path as a sequence of goals to reach the real target, there is an evident minimization of oscillations caused by the walls. In Figure 8, instead of taking a straight line to the target that resulted in augmenting the trajectory extension, the robot, using this method, can take an intermediate goal to the passage and go from there straight to the target.

Utilizing the passages as landmarks solves the two points mentioned before: it's possible to take a shorter way to the target and it decreases the probability of oscillations.

Therefore, the path is constructed as a sequence of sub goals of passages. To find this path it is constructed a linear graph having the rooms as nodes and the passages linking them. An algorithm running over this graph determines the solution.

The passages as local sub goals solve properly the non linearity given by the walls, enabling the local architecture to guide the robot to the target.

### 6.2 The Landmarks and the Path Definition Algorithm

The use of passages as landmarks facilitates the achievement of the requirement of minimum representation because it permits the construction of a linear graph with all the required information for navigation. The graph having the locals as nodes and the passages linking them contains the logic to find the path from where the robot is to the target. And also, the architecture provides a memory to store some information for each passage: the coordinates of its extremities, its angle with the axis of coordinates (angles greater or equal zero and lesser then 180 degrees), and the rooms it connects to its left and right, as in Table 1.

**Table 1:** Passages Information

conventions for Li and Pj depicted in Figure 16
px1,py1 and px2,py2 are coordinates of the extremities of the passages
coordinates in cm, angles in degrees

| Passage | px1 | py1 | px2 | py2 | angle | local to the left | local to the right |
|---|---|---|---|---|---|---|---|
| P1 | 510 | 620 | 510 | 740 | 90 | L1 | L2 |
| P2 | 1010 | 620 | 1010 | 820 | 90 | L2 | L3 |
| P3 | 1200 | 610 | 1320 | 610 | 0 | L3 | L4 |
| P4 | 1700 | 610 | 1820 | 610 | 0 | L3 | L5 |

**Table 2:** Rooms and its Passages

conventions for Li and Pj depicted in Figure 16
1 stands for the existence of the passage

| | P1 | P2 | P3 | P4 |
|---|---|---|---|---|
| L1 | 1 | 0 | 0 | 0 |
| L2 | 1 | 1 | 0 | 0 |
| L3 | 0 | 1 | 1 | 1 |
| L4 | 0 | 0 | 1 | 0 |
| L5 | 0 | 0 | 0 | 1 |

Additionally, it is also stored the information of what are the passages in each room, that are seeing in Table 2. All these information are permanent for a given environment, they represent fixed occurrences on its layout, and are only changed if it happens changes on it.

In the graph in Figure 16, rooms are nodes and arcs are passages. The algorithm to find the best topological path from the robot position to the target starts spreading a message from the node where the robot is with its identification to the nodes connected directly to it. The node receiving the message concatenates to it the information of the identification of the passage it came through plus its local identification and spreads forward this message to its neighboring nodes. A node can receive more then one message at the same time; in this case it will treat each of the messages independently. Messages that return to one local where it was before are eliminated. Therefore, when a message achieves the local where the target is, it contains the best topological path, with the information of all the passages to get there. When more then one message achieves the target at the same time, all of then are equal in terms of topology, so any one can be chosen. A simple modification can make the same algorithm find the ‘good’ shortest path.

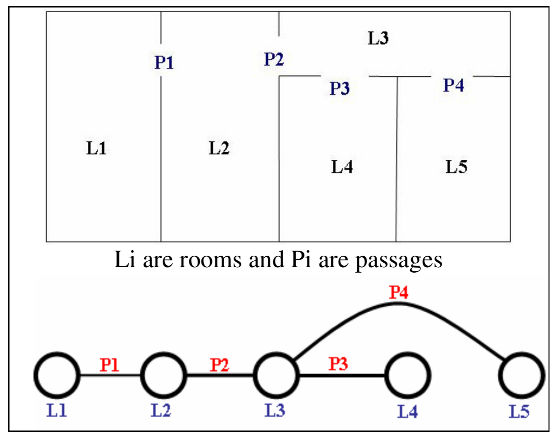
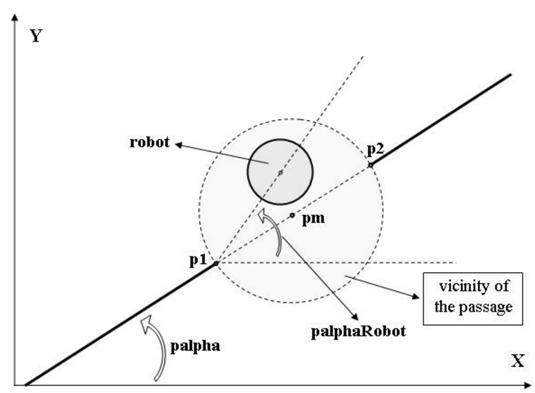



**Fig. 16:** The linear graph utilized




palphaRobot > palpha implies that the robot is in the left side of the passage

**Fig. 17:** The robot and the passage

When seeking a passage, the robot uses the coordinates of the center of the passage as its goal. Using the information in Table 1, it can tell whether it is in the left or right side of this passage. Also, it computes the vicinity of the passage (see Figure 17: Vicinity is a circular area about the center of a passage, having the passage size as diameter) and determines where it is, inside or outside. If outside, it will know when it gets inside, and, if inside, it will know when it gets outside. If it is inside and gets outside it just got into a room that might be the next one or the same room it was before. An obstacle near the passage has a chance to determine the robot to go back. So, it verifies where it is using **palphaRobot > palpha** (see Figure 17). A true result means that it is in the left side of the passage, and a false result means that it is in the right side. The knowledge of in which side the robot is and Table 1 defines the local where it is. When the robot enters a new room, it takes the next passage as its new goal.

A reason for the use of the vicinity is to avoid successive and unnecessary computation due to sudden changes of local as result of obstacles near the other side of the passage. The vicinity gives also a tolerance for probable sensors reading errors. These errors result in a virtual displacement of **pm**, the passage center point, and, if the displacement is somehow minor then the vicinity radius, the vicinity compensates for that. Without the vicinity, de displacement might cause the robot to determine that it crossed the passage that it had as intermediate goal before really doing that. Thus, the robot behaves as having achieved its current goal and defines the next goal in the other room as its current goal. It might create the situation of the robot being in one room having its target in the other room, a source of non linearity.

## 6.3 The architecture

The final architecture is in Figure 18. It is build over the behaviors of the Simple Architecture. There is a modification in the behavior Seek and now it is named Target Seek. There is not a Bearing sensor, Target Seek receives the coordinates of the target from Path Definition and calculates its direction. It, like Seek, will accept orientations of the target only if they fall inside its aperture. The Robot Position Sensor gives the position and orientation of the robot. The coordinates and room of the target are inserted through the User Interface.

There are three new behaviors added to the new architecture. They where designed, added and tested incrementally over the prior behaviors, and all the behaviors in the architecture run in parallel.

### 6.3.1 Passage Vicinity

This behavior receives data from the Robot Position Sensor, and current goal coordinates with its related information. It computes the vicinity and verifies whether the robot is inside or outside the vicinity. Passage Vicinity utilizes state memorizing where the robot is: inside or outside the vicinity. So, the behavior can tell when the robot 'is in', 'is out' or 'just got out' and delivers it to its output.

### 6.3.2 Local

The behavior Local is in charge of knowing in which room the robot is. Once it receives a 'just got out' from the Passage Vicinity behavior it uses information from the Robot Position Sensor and of the current goal to determine the result of the comparison of **palpha** with **palphaRobot**, being able to tell whether the robot crossed the passage and is in a new, the next one, room. This information is delivered to its output.

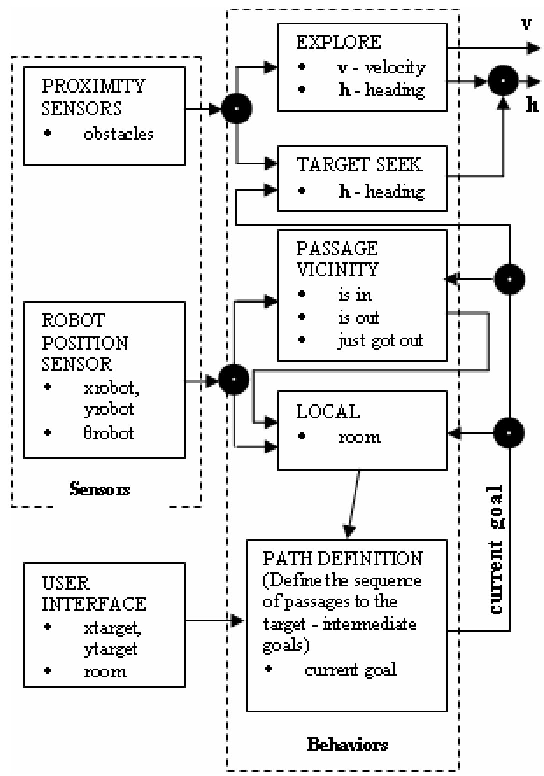



**Fig. 18:** The Complete Architecture

### 6.3.3 Path Definition

This behavior executes the Path Definition Algorithm, is responsible for determining the path with passages as sub goals to the target, and delivers the position and identification of the current goal. According to the input from Local, this behavior outputs the same goal or the next one.

This architecture defines a macro plan to get its work done, and this plan is controlled. At the same time it takes care of local events, the micro part of the macro plan. The local operation does not follow a detailed micro plan; it just takes the current goal and tries to solve the eventual local occurrences, step by step, as they happen to be.

Therefore, the macro and micro plan are performed by the Complete Architecture based only on behaviors. Planning and control comes from the integrated and harmonic functioning of all the system and it even can be understood as an emergent behavior that arises from it.

### 6.4 Results

The localization of the robot and the target in the environment are given in Figure 15. The absence of more obstacles, except for one table, is intentional to make evident the effect of the nonlinearity caused by walls. Anyway, the obstacles are local and can be well solved by the previous architecture. The passages are narrower compared with the passage in the simple environment, thus, the robot has now 50 cm of diameter. The parameters were adjusted empirically and their values are:

- Velocity = 15 cm/s
- Iteration interval = 0.6 s
- Maximum distance to detect an object = 60 cm
- Aperture for Target Seek to detect the goal = 180 degrees

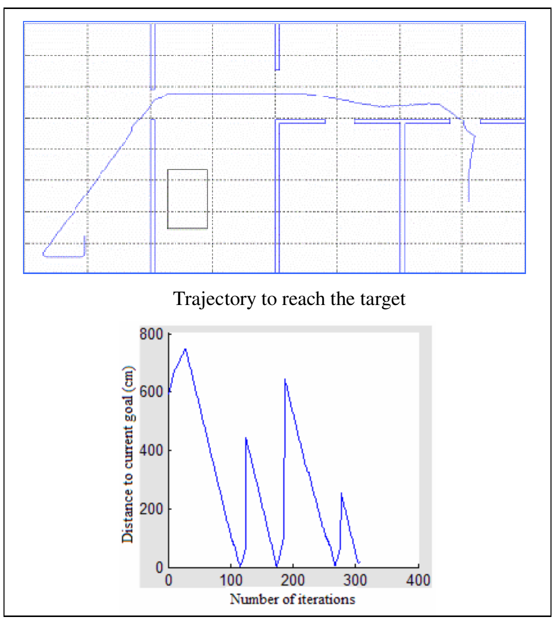



**Fig. 19:** Result for the Complete Architecture

In Figure 19 we see the Complete Architecture working. There is almost no oscillation. The graph shows distance to current goal. After the robot achieves a current goal the graph shows the distance augmenting for a while, until it gets out of the vicinity and takes the next goal. At this point occurs an abrupt increase in the distance.

Clearly, the new architecture achieved its objective with efficiency; the robot goes almost straight to the target, with minimum oscillation.

It has to be observed that the robot does not go directly to the first passage, its first sub goal. The reason is that this goal is not being detected by Target Seek. Once it is detected, the robot runs straight to it. Remembering that there is not the expectation to obtain an optimal solution, but a safe and good one, it is right to say that the solution was efficient. Anyway, there is solution to it, shown in the next section.

## 7 Conclusion and Perspectives

The objective of the authors was achieved. The architecture enables the mobile robot to perform real time autonomous navigation with efficiency in a simulated two dimensional environment.

There are still many points to be examined and probably implemented that are shown in the following lines.

- The trajectory of the robot to achieve its first sub goal, the first passage in Figure 19, is not straight. It happens because the passage is not in the field of view of the robot, that is, it cannot be detected by Target Seek. The first idea to solve it is augmenting the aperture of Target Seek, but it also can augment oscillations. A better way to solve it is to augment the aperture to 360 degree when the robot initiates a seek to a sub goal or to the target. In this moment the robot can detect the sub goal or target and, if this direction is free, to take it. In the next cycle the robot assumes again the aperture it was using.
- In Figure 19, the robot takes roughly 3 minutes to run 27 meters, at 15 cm/s. It can go faster augmenting its speed when there are not obstacles far away from the *edge* zone; what requires sensing beyond the *edge* zone and deciding when to increase and to decrease the speed.
- To take advantage of higher speeds it is relevant that the robot stays far from the walls when it is seeking a current goal. It is clear in Figure 19 that the robot, when in the direction of the last passage, travels near the wall, because the trajectory to the passage being the sub goal puts it there. A way to solve it is changing the position of the sub goal, making a displacement to move it to a point inside the room where the robot is, but still within the real vicinity. The robot takes the real goal again once it enters the real vicinity.
- There are also some points that relate to technology. This work does not address the problems of determining the position and attitude of the robot, whether or not it is sufficient to utilize microchips with accelerometers and gyroscopes, or to use shaft encoders, compasses, and GPS, or even a combination of them. The approach of utilizing the environment as its representation for the local navigation with the constant sensing and acting

does not require a so precise metric. But, it is relevant to know in detail what the implications are, mainly when dealing with the real world. Still in simulation, it is possible to get a good understanding of the sensibility to sensors distortions adding appropriate errors in the measurements, and verifying how it affects the results.

- The idea is to place the robot in real world with real obstacles and people, thus, it is relevant to examine its flexibility to deal with moving objects and people. Up to now, the design permits the robot to deviate from stationary or very slowly moving objects, which is expected for the type of environment it will inhabit. But it is not true with people that even will be interested, just for fun, in creating problems for the robot. Therefore, it is required to measure the speed of the incoming objects, and to take the right decisions and actions, such as decreasing speed, a high rate of deviation, and running backwards.
- The flexibility mentioned above requires also quick responses that are intimately linked to the iteration time. The constraint for the minimum time of the iteration is basically computation time and response time of the hardware. Computation power for the Complete Architecture is not critical; the problem might be with the response time of the hardware. It will depend mainly on things like the cycle of activation of the sensors, the torque of the motors, and low level control, all dependent on technology, which means cost.
- The behavior Explore drives the robot following some logic. In most of the passages it will make a rapid 180 degrees turn that can be considered as an indicator of a passage. Most of the 90 degrees turns that it makes correspond to the inside corners of a room. But, in the case of an outside corner, the turn is 90 degrees also, and is a passage. What differentiates them is that the internal turn is mostly a deviation, guided by the behaviors Forward and / or Deviate, and the external is mostly an alignment, guided by Align. Therefore, once the robot gets an indication of a passage it can perform a procedure that will detect it, probably with the help of another type of sensor. And so, this passage could be appended to the ones the robot already has memorized, and actualize the linear graph.

*References*


[1] Arkin, *Ronald* C., *Behavior-Based Robotics*. Intelligent *Robots* and Autonomous Agents Series, The MIT Press, May, 1998.

[2] Bekey, George A., *Autonomous Robots: From Biological Inspiration to Implementation and Control*. Intelligent Robots and Autonomous Agents Series, The MIT Press, Jun, 2005.

[3] Booker, L. B., Goldberg, D. E. and Holland, John H., 1989. *Classifier Systems and Genetic Algorithms*. Artificial Intelligence, 40 pp 235-282, 1989.

[4] Brooks, Rodney A., Intelligence Without Representation. *Artificial Intelligence Journal* (47), pp. 139–159., 1991.

[5] Brooks, Rodney A., Intelligence Without Reason. *Proc 12th Int. Joint Conf on A Intelligence*, August, 1991.

[6] Brooks, Rodney A., *Flesh and Machines: How Robots Will Change Us.* Pantheon Books, Feb, 2002.

[7] Jones, Josef L., *Robot Programming - A Practical Guide to Behavior-Based Robotics.* McGraw-Hill, New York, NY, USA, 2004.

[8] Mataric, Maja J., *A Distributed Model for Mobile robot Environment-Learning and Navigation.* Master of Science Thesis in Electral Engineering and Computer Science, Technical Report AI-TR-1228, MIT AI Lab, May, 1990.

[9] Mataric, Maja J., *Interaction and Intelligent Behavior.* Doctorate Thesis in Electral Engineering and Computer Science, Technical Report AI-TR-1495, MIT AI Lab, May, 1994.

[10] Mataric, Maja J., Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior. *J. of Exp. and Theoretical A. I., special issue on Software Architectures for Physical Agents*, 9(2-3), pp. 323-336, H. Hexmoor, I. Horswill, and D. Kortenkamp, eds., 1997.

[11] Mataric, Maja J., Behavior-Based Robotics as a Tool for Synthesis of Artificial Behavior and Analysis of Natural Behavior. *Trends in Cognitive Science*, pp. 82-87, Mar, 1998.

[12] Mataric, Maja J. and Cliff, Dave, Challenges In Evolving Controllers for Physical Robots *in Evolutional Robotics. special issue of Robotics and Autonomous Systems*, 19(1), pp. 67-83, Oct, 1996.

[13] Moussi, Lubnen N., *Aplicações de Sistemas Classificadores para Robótica Autônoma Móvel com Aprendizado*. Master of Science Dissertation in Electrical Engineering, UNICAMP, Nov, 2002.

[14] Moussi, Lubnen N. and Madrid, M. K., Simple Target Seek Based on Behavior, *proc 6th WSEAS International Conference on Signal Processing, Robotics and Automation (ISPRA '07)*, Corfu, Greece, February 16-19, 2007

[15] Moussi, Lubnen N. and Madrid, M. K., *Solving Target Seek with a Behavior-Based Architecture.*

WSEAS Transactions On Systems And Control, Issue 2, Volume 2, pp. 176-184, ISSN 1991-8763, February 2007.

[16] Moussi, Lubnen N. and Madrid, M. K., Behavior Based Autonomous Navigation Using Passages as Landmarks for Path Definition, *proc 12th WSEAS International Conference on Automatic Control, Modelling & Simulation ( ACMOS'10), pp. 131-137 in New Aspects of Automatic Control, Modelling & Simulation, EUROPMENT Press ISBN 978-954-92600-1-4,* Catania, Sicily, Italy, May, 2010

[17] Moussi, L. N., Gudwin, R. R., Von Zuben, F. J. and Madrid, M. K, *Neural networks in classifier systems (NNCS): An application to autonomous navigation*, in V.V. Kluev & N.E. Mastorakis (eds.) Advances in Signal Processing, Robotics and Communications, Electrical and Computer Engineering Series, WSES Press, pp. 256-262, 2001.

[18] Moussi, L. N., Von Zuben, F. J., Gudwin, R. R. and Madrid, M. K, A Simulator using Classifier Systems with Neural Networks for Autonomous Robot Navigation, *proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'2002), vol. 1, pp. 501-506, in the 2002 IEEE World Congress on Computational Intelligence (WCCI'2002)*, Honolulu, Hawaii, 12-17 maio, 2002.

[19] Nolfi, S. and Floreano, D., *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*, Intelligent Robots and Auton. Agents Series, The MIT Press, Nov, 2000.