Behavior Based Autonomous Navigation Using 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: - This work presents a solution to a mobile robot autonomous navigation utilizing minimum representation of the environment. It is done 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 and utilizes only behavior-based architecture. 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 that 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 approach used does not show a central planner and controller as in traditional deliberative architectures; planning and control arises from the independent parallel functioning of all the behaviors.

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

1 Introduction
This work is about the implementation of a differential wheels robot efficient intelligent architecture to perform autonomous navigation in a simulated two dimensional environment for labs, offices and classrooms. It uses behavior-based architecture considering reactive behaviors for local navigation and behaviors with minimum internal representation through a linear graph of passages for long run path definition.

Section 2 presents how the architecture was chosen and designed. Section 3 shows the Simple Architecture intended for local navigation and target seek. Section 4 is about the Complete Architecture that adds behaviors for path definition and control to the previous one. Conclusions and perspectives are in Section 5.

2 Solving Autonomous Navigation
Previous experience in evolutionary robotic for obstacle avoidance [13] utilizing classifiers systems [3] led the authors to consider it as a possible starting point for their project of autonomous navigation presented here. They knew that it would be a hard work. Looking for related literature they confirmed their expectation, as can be seeing, for example, in Mataric [12] and in Nolfì and Floreano [16] that makes clear the great effort expended in design. Therefore, the authors had to find out other possibilities and got to Behavior-Based Robotics, where the research of Mataric [8], [9] offered solid inspiration.

The classical approach to a 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 inserted 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.

This work utilizes Behavior-Based Robotics and divides the approach into two steps, named here Simple and Complete Architectures.

In nature and robotics a basic behavior is reactive [4], [5]. The Simple Architecture utilizes only reac-
tive behaviors considering the environment itself as its representation. Consequently, there is not the burden of environment representation as in the classical approach. Through sensing the environment, the robot can perform actions of running or stopping, deviating and aligning. These actions implemented with basic behaviors gives rise to the emergence of a behavior to explore the environment. Additionally, the Simple Architecture takes care of target seek because the path to the target, defined when running under the Complete Architecture, is divided into sub goals and each one of them is a local landmark. Another way to look to this architecture is to know that it was first developed to verify whether its emerged behavior of environment exploration allied to target seek can give solution to the non linearity caused by walls.

The Complete Architecture is developed to provide efficient target seek when the target is not local and is constructed over the Simple Architecture adding new behaviors for path definition and control. It utilizes a very simplified environment representation, a linear graph, having the passages as landmarks, that structures the environment representation and is the basis for the algorithm of path definition that gives an efficient solution. The path to the target is defined as a sequence of passages from the actual robot position to the local where the target is. 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.


## 3 The Simple Architecture

This architecture uses only reactive behaviors, without memory and environment representation. A more detailed view of what is presented in this section is found in Lubnen and Madrid [14].

Figure 1 shows a mobile robot (a blue circle) with differential wheels. 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.

### 3.1 The Behaviors

The behaviors used are reactive and similar to the basic ones utilized by Mataric [8], but are not equal. Hierarchical attributes and exclusive conditions are defined to solve conflicts. The architecture is shown in Figure 2.

The behavior **Forward** puts the robot to run when it is safe and stops it if it is in danger. If any proximity sensor shows an object in the danger region and the front sensors show an object in the safe and / or danger region, the robot is stopped and tries to find a safe direction to go. Failing to find, it moves backwards for a while in an arbitrary safe direction.

**Deviating** is build incrementally over, and has low priority then the above one. It avoids objects in its front edge region, and diminishes the use of **Forward**.

**Align** is built incrementally over and has low
priority then the above ones. Its goal is to maintain the robot in the proximity, in the safe region, of the object to which it gets near. 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.

The behaviors above utilize 30 degrees for each deviation. They give rise to a higher order behavior that emerges from their parallel execution:

**Explore:** This behavior emerges as the result of the actuation of *Forward, Deviate* and *Align*. With it, the robot has the condition to run safely, due to *Forward* and *Deviate*. Additionally, it gives the possibility to the robot to keep wandering in the entire environment, due to *Align*.

**Seek:** It is built incrementally over the other ones. The robot senses the bearing of the target and determines the heading to go straight there. *Seek* checks whether it's safe to do it or not, verifying if the proximity sensors in that direction are free in the danger, safe and edge regions. If it's safe, *Seek* will overwrite the heading given by *Explore*.

### 3.2 Results

In Figure 3 the *Forward* behavior is working alone showing sharp changes in orientation when an obstacle gets inside the safe region of the front proximity sensors. The robot stops and keeps changing its orientation until the front proximity sensors are free, when it runs again. Another important point is that the robot stays most of the time inside one of the rooms.

Figure 4 shows the effect of incrementally adding *Deviate*. The changes in orientation become smoother, done in 30 degrees increments. There are still some sharp deviations due to the action of *Forward*. There is not yet evidence of exploring efficiently the entire environment.

Figure 5 shows the robot exploring safely the environment under the work of *Explore* that emerges from the parallel functioning of *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.

![Fig. 5: Forward, Deviate and Align - Safe Exploration](image)

Figures 6 and 7 show the Simple BBR Architecture performing Target Seek. In Figure 6 *Align* is turned off, and the robot reaches the target after 495 iterations, seeming almost left to chance. Figure 7 shows the relevance of *Align*, now it is turned on, and the robot achieves the target after 108 iterations. The robot does not go straight to the target; however, it does that under a good and safe path. But, there is a better solution. It is shown in the next section.

### 4 The Complete Architecture

This architecture is an efficient solution to autonomous navigation performing target seek in a simulated 2D environment with many rooms and passages connecting them. It utilizes only reactive BBR for local navigation, as done previously, and adds incrementally new behaviors to take care of identifying where the robot is and to define the path.
The environment in Figure 8 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.

4.1 Overview of the Solution

The first ideas of the authors to solve efficiently autonomous navigation were presented in Lubnen 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 7. 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 8.

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 caused 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 7, 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.

4.2 The Landmarks and the Path Definition

The architecture used 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. 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.

<table>
<thead>
<tr>
<th>Table 1: Passages Information</th>
<th>px1, py1 and px2, py2 are coordinates of the extremities of the passages, coordinates in cm, angles in degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passage</td>
<td>px1</td>
</tr>
<tr>
<td>P1</td>
<td>510</td>
</tr>
<tr>
<td>P2</td>
<td>1010</td>
</tr>
<tr>
<td>P3</td>
<td>1200</td>
</tr>
<tr>
<td>P4</td>
<td>1700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Rooms and its Passages</th>
<th>conventions for Li and Pj depicted in Figure 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 stands for the existence of the passage</td>
<td></td>
</tr>
<tr>
<td>L1</td>
<td>P1</td>
</tr>
<tr>
<td>L2</td>
<td>1</td>
</tr>
<tr>
<td>L3</td>
<td>0</td>
</tr>
<tr>
<td>L4</td>
<td>0</td>
</tr>
<tr>
<td>L5</td>
<td>0</td>
</tr>
</tbody>
</table>

In the graph in Figure 9, the rooms are nodes with the passages linking them. 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 than 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 is the best topological one, and it will have information of all the passages to get there. When more than 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.

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 10) 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 $\text{palphaRobot} > \text{palpha}$ (see Figure 10). 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 to 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 $\text{pm}$, 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.

4.3 The Architecture

The final architecture is in Figure 11. It is build over the behaviors of the prior one. Instead of the bearing sensor, **Target Seek** calculates the direction of the goal. 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.

**Passage Vicinity:** This behavior gets 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. It utilizes state, it memorizes where the robot is inside or outside the vicinity. So, it can tell when the robot is in', 'is out' or 'just got out' and delivers it to its output.

**Local:** 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 \( p_{\alpha} \) with \( p_{\alpha_{\text{Robot}}} \), 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.

**Path Definition:** This behavior 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. This architecture is, somehow, a mix, or a hybrid of BBR and Deliberative. But, Deliberative has a central, higher hierarchical position, utilizes detailed maps, actualizes them, and plans and controls the macro and the micro with detail. And that is not what happens here. Actually, the plan 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.

### 4.4 Results

The localization of the robot and the target in the environment are given in Figure 8. The absence of more obstacles, except for one table, is intentional to make evident the effect of the non linearity 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. These variables were adjusted empirically: velocity, size of the proximity sensors zones and the time interval of the iterations. The robot runs at 15 cm/s.

Figure 12 shows the result obtained with the Simple Architecture. As previewed, it is a difficult job for that architecture. The result is very instructive; it shows clearly the effect of the non linearity that is augmented with the increase of the number of rooms, and, consequently, the number of walls between the robot and the target.

Figure 13 shows 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 is verified 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.

5 Conclusions and Perspectives
The objective of the authors was achieved. The architecture enables the mobile robot to perform autonomous navigation with efficiency in a simulated two dimensional environment. Now, there are many points to be examined, some of them through simulation. The authors are working to solve the ones that are mentioned below.

The robot might go faster when there are not obstacles beyond the out zone.

Make a displacement in the coordinates of the center of the passage, to maintain the robot away from the wall of its passage being a goal.

Provide the robot with the possibility of being more flexible to deal with moving objects, mainly people.

Find a way to decrease the iteration time, it helps to make the robot move faster with safety near objects.

Enable the robot to detect landmarks, thus, appending them or even constructing the entire internal graph by its own. The behavior Explore follows some logic that suggests a possibility to do it.

Verify the sensitivity to sensors distortions adding appropriate errors in the readings.

Acknowledgements:
Lubnen Name Moussi acknowledges his partial support for this work from UNICAMP – Universidade Estadual de Campina, CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, and UNIFEOB - Centro Universitário da Fundação de Ensino Octávio Bastos.

References