MULTI-SENSOR NAVIGATION WITHOUT AN A PRIORI MAP

Alberto Vale, João Simões, José Machado, Pedro Lima

Instituto Superior Técnico – Instituto de Sistemas e Robótica
Av. Rovisco Pais 1, 1049-001 Lisboa – Portugal
e-mail: vale@isr.ist.utl.pt, j.m.simoes@teleweb.pt, zem@isr.ist.utl.pt, pal@isr.ist.utl.pt

Abstract: This article describes a navigation algorithm for a mobile robot, endowed with odometry and a range sensor, moving in a structured indoor environment, without previous knowledge of its map. The objective is to discover the exit and obtain the shortest path from the initial point. The navigation algorithm adapts to changes in the environment. Results obtained with a mobile robot, endowed with a laser scanner, are presented.

1. INTRODUCTION

In an industrial environment, the presence of autonomous vehicles that perform repetitive tasks is becoming more predominant. Due to the vast physical area, and its complexity, where these vehicles operate, their adaptation to the environment without human intervention is fundamental. In this process, the vehicle needs to acquire the indispensable information required for time and spatial task management. In this work, where the vehicle is a mobile platform, a navigation algorithm is developed to carry out a mission within an indoors office-like environment.

In the literature, two ways for mapping indoor environments can be found: grid-based and topological. The first leads to complex planning and problem solving in large-scale indoor environments, while the topological maps can handle more efficiently the sensor errors [4][5].

This article describes a navigation algorithm for a mobile robot, endowed with odometry and a range sensor, moving in a structured indoor environment – a maze – without previous knowledge of its map. The objective is to discover the exit and obtain the shortest path from the initial point. The navigation algorithm adapts to changes in the environment. We have chosen a topological map composed of two building blocks: crossings and corridors, both delimited by walls. Along its mission, the vehicle must build automatically a topological map, based on which it can locate itself and optimize the path to execute the tasks for which it was designed. This map construction must consider the possible enlargement of the original physical area and the access to the map must be fast, in order to efficiently plan a trajectory.

The paper is organized as follows: Section 2 presents an overview of the work presented in this paper. Section 3 describes the data parameterization for the two levels of the navigation process, which is detailed in Section 4. Experimental results are presented in Section 5. Section 6 concludes the paper and presents directions for further development of this work.

Figure 1. Nomad Super Scout II platform with a Sick Laser on top.

2. OVERVIEW

The navigation algorithm uses the sensors data to acquire environment information. After data processing and its parameterization, the information available is used for two navigation levels: low level and high level. The first controls motor wheels, and the second builds and uses the topological map.

The maze is a group of corridors and crossings that will constitute the map. A crossing consists of a location, where the vehicle can change its trajectory. A corridor is the connection between two crossings, always maintaining the same direction. The delimitation is made by walls, with which the vehicle cannot collide. These are perpendicular among themselves.

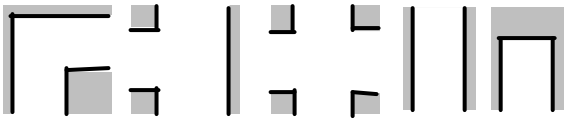

Figure 2. The combination of these five topological patterns (possibly rotated and/or enlarged or reduced) defines a maze.

The first 3 basic patterns on the left of Fig. 2, exemplify crossings in a maze. The fourth patterns represents a corridor. The last one is a special case: a corridor without exit, i.e., a dead end, which is identified as a crossing, where the only hypothesis is to turn backwards. However, the corridors have different widths and different lengths, which implies crossings with different dimensions.

The maze is represented topologically by means of a graph, where the crossings and the corridors correspond to the nodes and the arcs, respectively. Each node contains information on the connected arcs, the number of times that each one was visited and the dimensions of the crossing. The dimensions of the crossings are obtained using the dimensions of the previous and following corridors and the odometry data that is calibrated along the navigation.

The algorithm was tested on a mobile platform Nomad Super Scout II endowed with a Sick Laser, represented in Fig. 1. The vehicle's perception of the environment is obtained through evaluation of the Sick Laser's measurements and odometry [1], as mentioned. The Sick Laser measurements are represented in the (x,y) coordinate system (CS), coupled to the vehicle (Fig. 3). The odometry measurements are relative to the world CS, (x_w,y_w), coincident with the CS coupled to the vehicle for the initial posture.

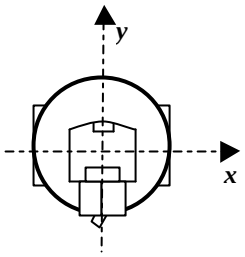

Figure 3. Vehicle CS for a Nomadic Super Scout II (top view).

The information retrieved by the Sick Laser consists of N=360 measurements in polar coordinates, (r_i,q_i) $i=1,...,N$, where r_i represents the value of the distance to the obstacle according to the direction q_i that ranges between 0º and 180º, with an increment of 0.5º. The odometry, based in the integration of the angles of each wheel, indicates the pose of the vehicle in the world CS. This information obtained from the sensors is then used to navigate through the maze.

3. DATA PARAMETERIZATION

As there is no initial knowledge of the maze where the vehicle will navigate, it is fundamental to select trajectories in order to gather the largest possible amount of information so that it can elaborate the topological representation of the maze. The mobile platform must avoid obstacles and maintain a smooth and centered trajectory both in the corridors and the crossings.

While the exit of the maze has not been reached, the priority options are those that allow discovering new nodes in the map which, regarding the maze, means new corridors and new crossings. When the objective is reached, to return to the starting point, the shortest path in the computed map is selected, which corresponds to the optimal path.

It is useful to point out that, for new paths in the maze, the map is enlarged, as much as the need of reaching new objectives (new points to reach). There are several methods of identifying that the objective was reached, e.g., by the odometry values, or by the recognition of a pre-defined geometric object with the Sick Laser, or by the visualization of a pre-defined image through a video camera, among others. The several steps taken along the navigation process are illustrated in Fig. 4.

After the acquisition of the Sick Laser's data, (r_i,q_i), the information is processed by filtering the points that are distant from the center of the vehicle of a dynamic value, d, proportional to the corridor's width, w. There are several corridors in the maze, with different width values, w. However it is assumed that each corridor has an approximately constant width, which w is determined by (4.5). This fact is due to the diversity of the dimensions of the corridors that constitute the maze. Because of the diffraction limitations of the laser, mainly in corners, it is necessary to apply median filtering. After median filtering, the existing related groups in the data are identified. By related group it is meant a group of points that are not further away from the previous value than a threshold value.

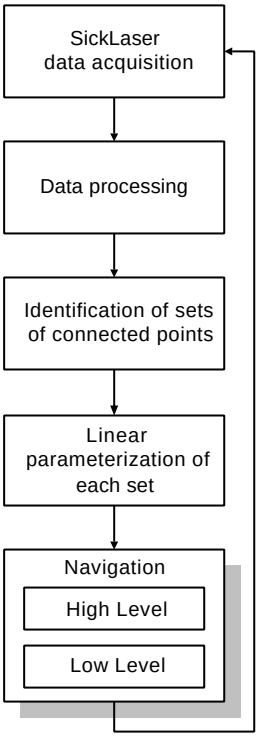

Figure 4. Schematic of the navigation process.

The (r_i,q_i) are compared with the previous values, because the points are sorted in increasing order according to the angle. At the end of each iteration, each point is identified as (r_i,q_i,c), $i=1,...,N$, where r_i,q_i corresponds to the polar coordinates of point i, that belongs to group c. The number of groups changes between iterations. The r_{th} and q_{th} are the threshold values.

As the environment is well structured, composed by corridors and crossings as described previously, each group can be parameterized with one or more straight lines.

$$e^2 = E\left[(y-\hat{y})^2\right] \tag{3.1}$$

The error of a given group with respect to a linear parameterization $\hat{y} = a \cdot x + b$ is minimized by determining a and b that minimize the cost function (3.1), where (x,y) are Cartesian coordinates and $\hat{y}$ is an estimation of y.

$$\min_{a,b} e^2 \Rightarrow \begin{cases} a = \dfrac{E[xy]-E[x]E[y]}{E[x^2]-E^2[x]} \\ b = \dfrac{E[x^2]E[y]-E[xy]E[x]}{E[x^2]-E^2[x]} \end{cases} \tag{3.2}$$

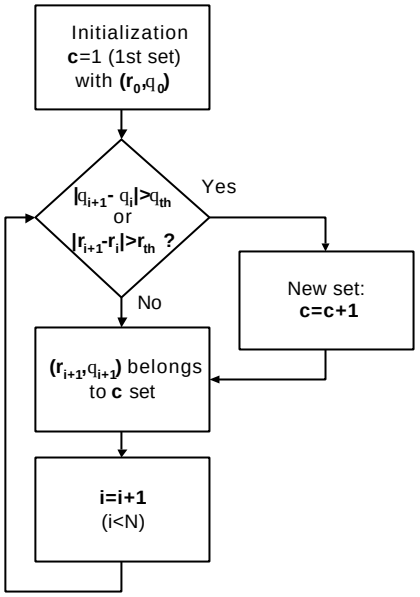

Figure 5. Sets constructing.

Due to the positioning of the maze walls and the chosen CS, the algorithm expects to detect almost horizontal straight lines, where the absolute value of b is very high, and vertical straight lines, where the slope, a, reaches high values. Due to this situation, straight lines are defined (j,b), which satisfy the parametric equation (3.3), where j is the angle between the straight line and the y-axis.

$$x \cdot \cos(j) + y \cdot \sin(j) = b \tag{3.3}$$

However, it is not guaranteed that the parameterization of each group is achieved with a single straight line [3], in order to assure that the error value is less than a threshold value. If this happens, the maximum possible number of consecutive points is parameterized with only one straight line (until the error value is less than a threshold). The same procedure is followed for the remaining points of the same group. The number of straight lines is increased until the group is completely parameterized (see Fig. 6). If each straight line is valid for only two points of this set, which means too many straight lines, then this iteration is ignored because the measures were inaccurate. Besides the values (j,b), other parameters are necessary to define the straight line, r_k, the limits between which the straight line is valid, x_{inf}, y_{inf}, x_{sup} and y_{sup}, and the set c it belongs to:

$$r_k = \left(j, b, x_{\inf}, y_{\inf}, x_{\sup}, y_{\sup}, c\right) \tag{3.4}$$

At each iteration there exists a variable number of groups and all this information is parameterized by seven parameters times the number of straight lines.

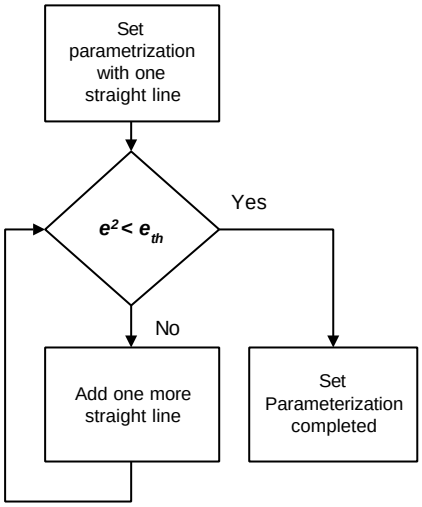

Figure 6. Linear parameterization of each set.

4. NAVIGATION

Data parameterization is used for navigation. It is necessary to elaborate trajectories and to build the map of the maze. There are two navigation levels: low level and high level. Low level navigation corresponds to the determination of the pose *(x,y,q)*, in order to accomplish trajectories that are equidistant to the maze walls, with the desired orientation. High level navigation consists of the elaboration of the graph map, as a topological maze representation. It also optimizes the path with all the information obtained so far, in order to reach the intended objective.

4.1 Low level navigation

Based on the previously computed parameterization, it is possible to calculate the angular error, q_e, that corresponds to the angle between the *y*-axis of the referential coupled to the vehicle and the corridor's walls, as well as the error distance, d_e, between the center of the vehicle and the middle of corridor, measured perpendicular to the walls. The value *w* defines the corridor with, measured perpendicular to the walls. As shown in the Fig. 7, the controller uses those values to calculate the velocity reference that will be applied to the wheel velocity controllers.

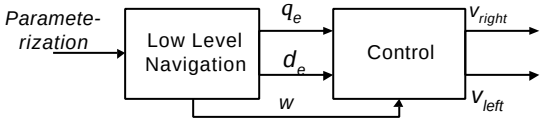

Figure 7. Schematic of parameterization vs. control.

Based on the CS coupled to the vehicle and due to the maze structured environment, the resulting straight lines of the parameterization could only be vertical or horizontal. Knowing the slope of the straight lines (defined by j, see (3.3)) it is possible to calculate the orientation error of the vehicle. Thus, straight lines that have $|j|<45^o$, are considered vertical straight lines, otherwise, they are horizontal.

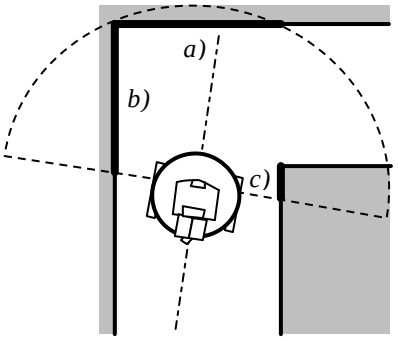

Figure 8. Typical situation, where the visualization range shows two sets: the first (on the left of the platform) with straight lines *a* and *b*, and the second set (on the right) with a single straight line *c* as a result of parameterization.

As a result of the parameterization we have two groups, the first composed of straight lines *a* and *b*, and the second with a single straight line *c*, see Fig. 8. While the straight lines *b* and *c* are considered vertical, *a* is horizontal. In Fig. 8, the effect of the occlusion in the group to the right of the vehicle is noticeable.

To calculate the orientation error of the vehicle, q_e, the weighted average of the slope of the vertical straight lines and the slope of the horizontal straight lines is used. Regarding vertical straight lines:

$$q^v_{mean} = \frac{\sum_i^{R_V} N_i j_i}{\sum_i^{R_V} N_i + \sum_j^{R_H} M_j} \qquad (4.1)$$

where R_v corresponds to the number of vertical straight lines of all groups, N_i, the number of points that constitutes the straight line *i* (obtained using the variables x_{inf}, y_{inf}, x_{sup} and y_{sup}, of (3.4)) and j_i, the slope of vertical straight line *i*. A similar procedure is applied to the horizontal straight lines. However, we must subtract 90º to the slope of the straight lines, because these are considered horizontal:

$$q^h_{mean} = \frac{\sum_i^{R_H} M_i (j_i - 90^o)}{\sum_i^{R_V} N_i + \sum_j^{R_H} M_j} \qquad (4.2)$$

The value of orientation error, q_e, is calculated by means of the weighted sum of q^v_{mean} and q^h_{mean}:

$$q_e = q^{v}_{mean} + q^{h}_{mean} \tag{4.3}$$

The distance error between the vehicle and the corridor's center, d_e, is the result of the sum of the distances, between the center of the referential coupled to the vehicle and the intersection of the vertical straight lines with the *x*-axis:

$$d_e = \frac{1/2}{R_{V,\,b>0}} \cdot \sum_{\substack{i \\ b_i>0}}^{R_V} \frac{b_i}{\cos(j_i)} + \frac{1/2}{R_{V,\,b<0}} \cdot \sum_{\substack{i \\ b_i<0}}^{R_V} \frac{b_i}{\cos(j_i)} \tag{4.4}$$

The parameter $R_{V,\beta>0}$ indicates the number of vertical straight lines, where b_i is positive, while $R_{V,\,b<0}$ indicates the straight lines number with negative b_i. Fig. 9 illustrates the orientation and distance errors with respect to the corridor's walls. Although the examples present the errors separately, they always exist together.

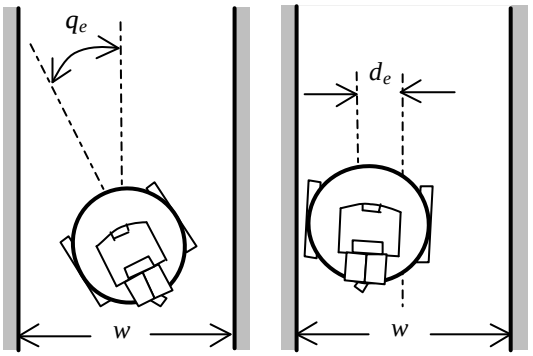

Figure 9. Example of orientation error (left) and distance error (right).

The calculation of the corridor's width is obtained using the straight lines that are considered vertical and the intersection of these with the *x*-axis of the CS coupled to the vehicle:

$$w = \frac{1}{R_{V,\,b>0}} \cdot \sum_{\substack{i \\ b_i>0}}^{R_V} \frac{b_i}{\cos(j_i)} - \frac{1}{R_{V,\,b<0}} \cdot \sum_{\substack{i \\ b_i<0}}^{R_V} \frac{b_i}{\cos(j_i)} \tag{4.5}$$

The controller is designed based on functions that depend on parameters d_e and q_e: $f(d_e)$ and $g(q_e)$ [2]. The maximum value of orientation error, is 45º (due to the definition of vertical straight lines; see Section 4.1), while $d_{e_{\max}}$ corresponds to half of the corridor's width, *w/2*, minus the half diameter of the vehicle. When the absolute value of d_e approaches the limit, it is necessary to correct that value, whatever the value of q_e will be. Function $g(d_e)$, defined by (4.6) is used to scale $g(q_e)$:

$$g(d_e) = \left(1 - \frac{|d_e|}{d_{e_{\max}}}\right) \tag{4.6}$$

Functions $f(d_e)$ and $g(q_e)$ are proportional to d_e and q_e respectively. In the crossings the function $f(d_e)$ is not used, because d_e cannot be calculated (4.4). When the platform enters the crossing, in the first iterations, the q_e signal is changed and its value is increased by a positive or negative offset, to make the controller unstable. The proportional gain constants for the controller were obtained experimentally.

Odometry calibration is essential for platform navigation, because the center of the crossings is obtained by odometry. The values are calibrated using the following equations:

$$\begin{aligned} x_i &= x_{i-1} \pm \left\|(x_i, y_i) - (x_{i-1}, y_{i-1})\right\| \cdot \sin(q_e) \\ y_i &= c_{corridor} \pm d_e \end{aligned} \tag{4.7}$$

where *i* and *i-1* are consecutive iterations and $c_{corridor}$ corresponds to the entrance of the previous crossing, given by the odometry, plus *w/2*. The signal ± corresponds to "+" for 0º orientation and "–" for 180º orientation. For other orientation cases, the variables *x* and *y* are exchanged in (4.7).

4.2 High level navigation

At this point, besides parameterization, we also know error values d_e and q_e. In this navigation level it is necessary to identify if the vehicle is either located in a crossing or in a corridor. It is also necessary to build the topological map and analyze all the information in order to make the right decision on the path to take.

If the positioning and the limits between which the straight lines are valid, that will allow either corridor or crossing identification. If the vehicle is located in a corridor, with error values below the defined limits, it is expected to perceive two groups, each with a straight line. Only the visible area allowed by the algorithm limits each one of those straight lines. When the upper limit of the straight lines begins to be inferior to the value imposed by the area allowed by the algorithm, the credibility of a crossing begins to increase. This credibility factor releases the algorithm of unsuccessful iterations. When the number of points of each straight line is less than a threshold value, it is considered that the vehicle is located in a crossing. The number of groups is analyzed, as well as the number of straight lines at each one, to evaluate the possible exits of the crossing. In the example shown in Fig. 8, the vehicle is about to enter a crossing, constituted by two groups. The first is defined by straight lines *a* and *b*, and due to the continuity between the straight lines and the positioning, indicates the inexistence of any exit for the left or in front. The second group displays a greater separation with respect to the previous

group, which means an exit for the right. The algorithm just considers an exit, when the distance between two groups is more than the diameter of the vehicle.

Nodes and arcs constitute the topological map as it was already referred in section 2. A node is associated to each crossing, identified as node k, containing the following information: the physical dimensions of the crossing contained in the vector $\bar{v}$, the number of times that each arc was visited $\bar{t}$ and vector $\bar{b}$ with the possible directions that the vehicle can take:

$$n_k = (\bar{v}_k, \bar{t}_k, \bar{b}_k) \quad k = 1,2,3,.... \tag{4.8}$$

The physical dimensions of the crossing $\bar{v}$, are defined relative to its center, width and length. Vector $\bar{b}$ identifies the arcs of each node. In positions *0, 1, 2* and *3* of this vector we define the arcs for the left, front, right and back respectively in relation to the world CS (coincident with the referential coupled to the vehicle at initialization). Whenever a node is created, the existent arcs in that crossing are identified and the respective positions of vector $\bar{b}$ are initialized to *0* (zero). The remaining positions are initialized with value –1 (or NULL). Knowing the branch that links to the previous node, the corresponding position of vector $\bar{b}$ is refreshed with the number of that node.

The graph is built based on this procedure. Once located in the crossing, before a node is created, the position of the vehicle is verified in the world CS and the existence of the node is sought in the graph map. In case it doesn't exist, this fact leads to the creation of the node, just as it was explained in the previous paragraph. Afterwards, based in the existent arcs in that node, a decision must be made. If the path to the objective is still not known, it prevails the mission "to conquer", that is, to search for crossings still not visited, or that have been traveled the smallest possible number of times. This is how an enlargement of the map is achieved.

$$out(n_k) = \min_{t_i}(branch(n_k)) \tag{4.9}$$

While the objective has not been reached, the process repeats itself iteratively. Once its mission is accomplished, the vehicle will have to return to the starting point, or otherwise, it will be assigned a new objective. If, through the built map, it succeeds to reach the objective, it will optimize the path according to the traveled distance. The *A** algorithm is used in the search of the optimal path between two nodes in the built map.

5. EXPERIMENTAL RESULTS

The vehicle used is a Nomad Super Scout II (see Fig. 1) a non-holonomic mobile unit with two motor-driven wheels. It includes a Pentium 233 MHz with 64 MB and odometry sensors. It also incorporates on the top of the vehicle a Sick Laser with a 180º scan angle (0.5º precision) and communicates with the computer through the serial port, using the RS232 communication protocol, at a speed of 38.4 kbps.

The low level navigation was initially tested in a corridor (see Fig. 10). As shown in Fig. 9, in the first test the platform begins with an orientation error, while in the second, with a distance error. In this case, the platform's orientation is parallel to the walls. The length of the corridor is about 1.5 meters. The platform corrects the distance error faster, which is also important to correct the orientation error. This is due to the distance error created when the platform moves, trying to correct the orientation.

The real situation of Fig. 8 is shown in Fig. 11. The platform changes its direction smoothly in the crossing. The small oscillation is due to errors in the laser measurements. However it is not perceptible on the platform motion, due to the reduced vehicle acceleration.

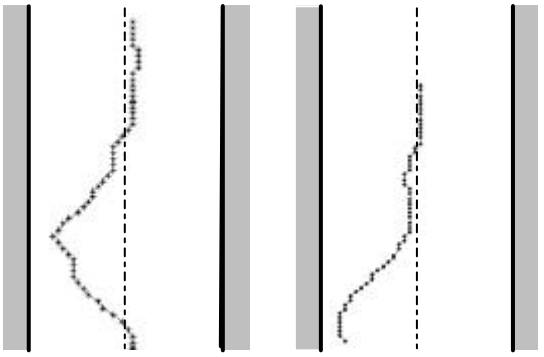

Figure 10. Example of navigation through a corridor starting with orientation error (left) and distance error (right).

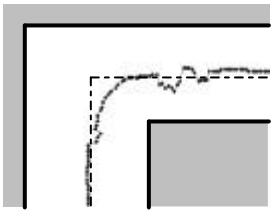

Figure 11. Navigation through a crossing.

After these low level navigation tests, a maze was built as shown in Fig. 12 and Fig. 13. The maze dimensions are 7 × 4 meters. There is no exit in the maze to visit all the corridors and crossings and to test the algorithm when a crossing is often visited. In

the first trial, Fig. 12, the odometry correction was disabled and, at each crossing, the direction was randomly chosen. For almost 4 laps, the increase of odometry error is visible. After a few more laps, the algorithm couldn't identify all the crossings. Due to the odometry error, in Fig. 12 it seems that the platform crashes against the wall in some corridors (in the middle of the maze and near the dead-end).

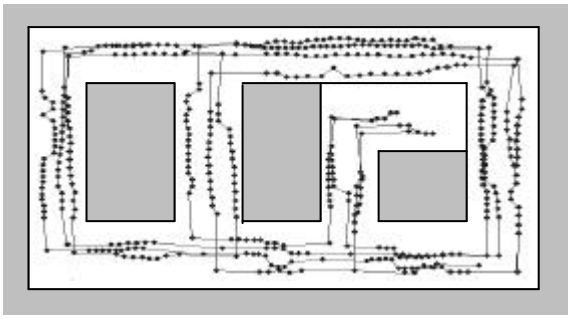

Figure 12. Navigation in a maze without odometry correction.

In the last test, shown in Fig. 13, the odometry correction was enabled. In each crossing the algorithm chooses the direction not so often visited (in order to discover more paths). The platform started its path in the right side of the maze. The right half of the maze is visited more times because of the deadlock, which is solved perfectly. When this side of the maze is completely known, the left half is discovered with one path. This test lasted almost 5 minutes to discover the entire map. If there were an exit, it would use less time.

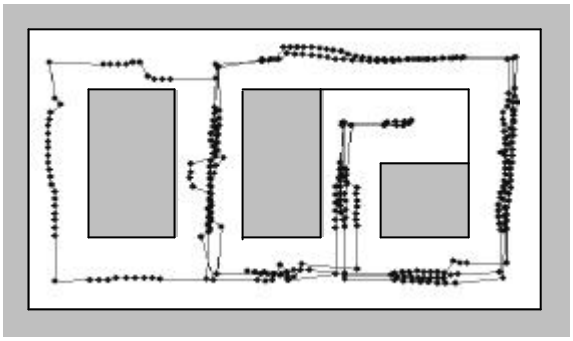

Figure 13. Navigation in a maze with odometry correction.

6. CONCLUSIONS

This paper demonstrated navigation in an unknown environment without a previous knowledge of a map. The method requires that the environment is expressed as a combination of the five patterns defined in 2. Still the fact remains that the navigation algorithm works with not parallel walls. The vehicle always navigates in the middle of the corridor using the parameterization algorithm in the limited observation area. For transient obstacles, as someone walking along the corridor, the navigation algorithm avoids the collision and stops. If the obstacle persists, the vehicle turns back along the corridor without modifying the map.

Further developments could integrate vision sensing to add more information about each crossing. Vision could be used to replace the odometry. To optimize the vehicle performance, a velocity controller could be developed to increase the translation velocity proportionally to the corridors width.

7. REFERENCES

[1] J. Borenstein, H. Everett, L. Feng, (1996). *Where am I ? Sensors and Methods for Mobile Robot Positioning*.

[2] R. Carelli, S. Humberto, V. Mut, (1999). *Algorithms for Stable Control of Mobile Robots with Obstacle Avoidance.* In Latin American Applied Research. Pages 191-196

[3] J. Mota, M. Ribeiro. *Localization of a Mobile Robot Using a Laser Scanner on Reconstructed 3D Models*. In Proceedings of 3rd Portuguese Conference on Automatic Control. September 1998

[4] J. Leonard, H. Durrant-White, I. Cox. *Dynamic Map Building for an Autonomous Mobile Robot*. In Proceedings of the IEEE International Workshop on Intelligence Robots and Systems. July 1990

[5] S. Thrun, A. Bücken. *Learning Maps for Indoor Mobile Robot Navigation*. Artificial Intelligence. 1998.