An Exploration and Navigation Approach For Indoor Mobile Robots Considering Sensor's Perceptual Limitations

Leonardo Romero, Eduardo Morales and Enrique Sucar

ITESM, Campus Morelos, Temixco, Morelos, 62589, Mexico
lromero@zeus.ccu.umich.mx, {emorales,esucar}@campus.mor.itesm.mx

Abstract

A new method for exploring and navigating autonomously in indoor environments is described. This method merges a local strategy, similar to wall following to keep the robot close to obstacles, within a global search frame, based on a dynamic programming algorithm. We introduce the concept of travel space as a way to map costs to grid cells based on distances to obstacles. This hybrid approach takes advantages of local strategies that consider perceptual limitations of sensors without losing the completeness of a global search. This exploration and navigation method is tested using a simulated and a real mobile robot with promising results.

1 Introduction

This paper introduces an exploration and navigation approach for an indoor mobile robot. The approach considers information from its sensors, to learn and use a *Probabilistic Grid-based Map* (PGM) [1] of an environment. A PGM is a two dimensional map where the environment is divided in square regions or cells of the same size that have occupancy probabilities associated to them.

The two principal components for building a PGM are exploration and position estimation. Research on exploration strategies has developed two general approaches: *reactive* and *model based*. By far the most widely–used exploration strategy in reactive robotics is *wall following*. Model based strategies vary with the type of model being used, but they are based on the same underlying idea: *go to the least–explored region* [4]. There are also several successful localization methods that can estimate the robot's position using its sensors [2]. However, most localization methods fail when the sensors of the robot are beyond its perceptual capability [7] (i.e. the robot is too far from obstacles). In [7], a *coastal navigation* method similar to our navigation approach is described, once the PGM has been built. Each cell in the map contains a notion of information content available at this point in the map, which corresponds to the ability of the robot to localize itself. The information content is based on the concept of entropy and some assumptions are considered to reduce the complexity of the method. Our approach generates a similar form of coastal navigation with a simpler and more efficient method. The incremental algorithm developed in this paper also fits the time requirements for the exploration task. In another related work [8], the probability of occupancy of the cell is used as a cost associated to the cells. The motion policy, given by a dynamic programming technique called *value iteration*, needs to be postprocessed in order to keep the robot near to the center of narrow passages (but they do not describe how to do that). In our approach there is no need to modify the policy given by the value iteration algorithm. The local guides, in the form of costs, are inside the value iteration algorithm, so the policy is optimal given the costs associated to cells and the cost to move to an adjacent cell.

This paper introduces a novel approach to explore a static indoor environment. The idea is to reach the nearest unexplored grid cell minimizing the travel cost. The travel cost takes into account the perceptual limitations of the sensors and tries to maintain a fixed distance to obstacles while the robot is moving (*wall following*). The concept of *travel space* is introduced to assign costs to grid cells. The motion policy of the robot is computed using a dynamic programing algorithm that includes the costs associated to the travel space. This approach merges local or *reactive* strategies with a global or *model based* strategy. The travel space is used to obtain an efficient navigation algorithm based on the same dynamic programming algorithm. The idea is to reduce the number of free

cells to be processed. A *roadmap* [3] is built upon the travel space, where only the cells of the roadmap are included in the dynamic programming algorithm for navigation.

The remainder of the paper is organized as follows. Section 2 describes the proposed exploration approach. Section 3 describes the navigation method, once the map has been built. Section 4 presents experimental results using a mobile robot simulator and a real mobile robot. The experiments are performed using a system build upon the ideas for sensor data fusion and position tracking given in [6]. Finally, section 5 is devoted to the conclusions and future work.

2 Exploration

The PGM building process does the following general steps:

1. Process the readings taken by all the sensors and update the probability of occupancy of the cells in the PGM (Sensor Fusion Step).
2. Update the travel space accordingly with the changes in the PGM (see Section 2.1).
3. Choose the next movement using value iteration (see Section 2.2). If the movement is not valid then the map is complete.
4. Execute the movement.
5. Get readings from the sensors and correct odometric error (Position Tracking Step).
6. Go to the first step.

The general idea for exploration is to move the robot on a minimum–cost path to the nearest unexplored grid cell [8]. The minimum–cost path is computed using *value iteration*, a popular dynamic programming algorithm. In [8] the cost for traversing a grid cell is determined by its occupancy value, while in [5] the cost is determined by the distance between cells (see chapter 8 in [4]). This paper proposes an approach that combines local search strategies within a modified version of value iteration described in [5]. When the robot starts to build a map, all the cells have the same probability of occupancy $P(O) = 0.5$. A cell is considered *unexplored* when its occupancy probability is in the interval (close to 0.5) defined by two constants $[Pe_{min}, Pe_{max}]$ $(Pe_{min} < 0.5 < Pe_{max})$ and *explored* otherwise. In an alternative approach [8], a cell is considered explored when it has been updated at least once. That approach, however, does not work well when there are specular surfaces (i.e., using ultrasonic range sensors) in the environment, since multiple measurements are normally required to get reliable estimates for the probability of occupancy of a cell.

Cells are defined as *free* or *occupied*. A cell is considered occupied when its $P(O)$ reaches a threshold value Po_{max} and continues to be occupied while its $P(O)$ does not fall below a threshold value Po_{min} (where $Po_{min} < Po_{max}$). It is considered *free* in other case. This mechanism prevents changes in the state of occupancy of a cell by small probability changes. We assume that $Pe_{max} < Po_{min}$, so an unexplored cell is also a free cell. In this way, the PGM becomes a binary map when cells are classified as occupied or free. This binary map will be called *occupied–free* map.

In this work, a cylindrical (circular base) robot was used, so the configuration space (c–space) [3] can be computed by growing the occupied cells by the radius of the robot. In fact, the c-space is extended to form a *travel space*. The idea behind the travel space is to define a way to control the exploration by a kind of *wall following strategy*. Wall following is a local method that has been used to navigate robots in indoor environments, but unfortunately it can easily get trapped in loops [4]. The travel space together with a dynamic programming technique has the advantages of both, local and global strategies: robustness and completeness.

2.1 Travel Space

Consider the travel space due to a single real occupied cell in the occupied–free map (see Figure 1). The travel space splits the cells of the occupied–free map in four categories:

1. *Occupied cells.* These cells are inside the circle given by the radius of the robot (as in the c–space) with center in the real occupied cell. After this expansion, the robot is considered as a single cell.
2. *Warning cells.* Cells close to an occupied cell. Let D_w be the maximum distance between a cell of this type and its closest real occupied cell. These cells are called *warning cells* because their purpose is to warn the robot about its closeness to an obstacle. The value of D_w takes into account the perceptual limitations of the sensors.
3. *Travel cells.* Cells close to a warning cell. Let D_t be the maximum distance between a cell of this type and its closest real occupied cell. These cells

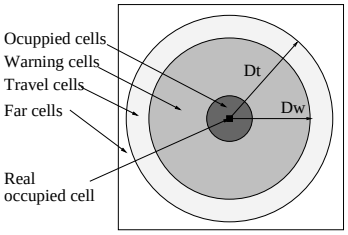

Figure 1: Travel space due to a single occupied cell

are called *travel cells* because their purpose is to suggest to the robot a path to follow.

4. *Far cells.* Any free cell (in the occupied–free space) that is not a warning or a travel cell.

In order to assign a higher cost to warning cells closer to obstacles, each warning cell must record, besides its type, the distance to the nearest occupied cell d_{min}. In this work, a linear function is used to get the cost of a warning cell depending on the distance to the nearest occupied cell. For travel and far cells it is enough to record the cell's type.

The travel space can be computed incrementally after each change of state of a cell in the occupied–free map while the robot is exploring the environment. Here is the algorithm:

1. Initialization. Let all free cells in the travel space be of type *far*.
2. If there is a change from free to occupied cell, do:
   - Grow the occupied cell by a radius of the robot. Assign the type *occupied* to the cells in the circle.
   - Using a larger circle, compute the warning cells (see Figure 1). For each of these cells, let d be the distance from the cell to the center of the circle, and t_{old} be the type of cell previously assigned to the cell. If $(t_{old} \in \{travel, far\})$ or $((t_{old} = warning)$ and $(d < d_{min}))$ then assign the type *warning* and distance d to the cell.
   - Using a larger circle, compute the travel cells (see Figure 1). For each of these cells, let t_{old} be the type of cell previously assigned to the cell. If $t_{old} = far$ then assign the type *travel* to the cell.
3. If there is a change from occupied to free cell, do:
   - Using a circle of radius D_t equal to the outer circle used to compute the travel cells, assign the type *far* to all the cells under the circle. In other words, it cleans the effect of the previous occupied cell.
   - Consider a circle of radius $2D_t$. For each occupied cell inside this circle in the occupied–free map, repeat step 2. This step redoes the effect of the occupied cells in its neighborhood.
4. Repeat steps 2 or 3 until the map building process ends.

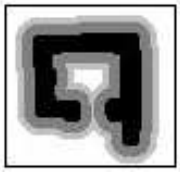

Figure 2: A travel space due to multiple occupied cells. From darker to lighter: occupied cells (black), warning cells (dark gray), travel cells (light gray), and far cells (white)

Notice that the process to update a transition from an *occupied* to a *free* cell is much more expensive than the change from *free* to *occupied*. Fortunately, most changes are from *free* to *occupied* during map building. An example of a travel space due to multiple occupied cells is shown in Figure 2.

2.2 Global Search

A policy to move to the unexplored cells following minimum–cost paths is computed using the travel space and a modified version of value iteration. The algorithm uses two variables, V and M, associated to each cell. $V(x, y)$ denotes the travel cost from cell (x,y) to the nearest unexplored cell. $M(x, y)$ represents the optimal movement to choose, given that the robot is in that cell. We consider 8 possible movements of the robot, one per cell in its vicinity. If $M(x, y) = (dx, dy)$, where dx is the change in x and dy is the change in y, the set of valid movements is $M_v = \{(1, 0),(1, 1),(1, 0),(-1, 1),(-1, 0),(-1, -1),(0, -1),(1, -1)\}$. The idea is to associate costs to cells depending on its type. If warning cells and far cells have costs higher than travel cells, then a wall following strategy for exploration is taken into account.

For simplicity, cells of type $warning$, $travel$ or far, will be call *free* cells in the travel space. Using the variables M and V, the algorithm has two steps:

1. Initialization. Unexplored cells (x, y) that are free in the travel space, are initialized with $V(x, y) = 0$, while explored cells that are free in the travel space are initialized with $V(x, y) = \infty$. All the free cells in the travel space are initialized with an undefined value to M.

2. Update. Let (x_r, y_r) be the position before the last movement of the robot. For all the explored free cells $(x, y) \neq (x_r, y_r)$ in c–space do:

   $V(x, y) \leftarrow \quad min_{(dx,dy)\in M_v}\{V(x + dx, y + dy) + Cost((x, y), (dx, dy)\}$

   $M(x, y) \leftarrow \quad arg\_min_{(dx,dy)\in M_v}\{V(x + dx, y + dy) + Cost((x, y), (dx, dy)\}$

where $Cost((x, y), (dx, dy))$ measures the cost of moving from the cell (x, y) to the cell $(x+dx, y+dy)$. This function punishes changes in direction and takes the value $C(x+dx, y+dy)+Dist((x, y), (x+dx, y+dy))+Kp_c$. Where $Dist(p_1, p_2)$ is the distance between cells p_1 and p_2 (1 or $\sqrt{2}$), and Kp_c represents the cost of the rotation of the robot to reach the next cell. $C(x, y)$ represents the cost associated with cell (x, y) in the travel space, based on its type. This assignment that punishes direction changes of the robot makes sense if we consider that rotation changes are a major source of uncertainty for the position of the robot.

The update rule is iterated, and when the values $(V(x, y),\ M(x, y))$ converge, the robot executes the movement indicated by M.

Exploration ends when $V = \infty$ for the cell where the robot is placed, which means that there is no way to reach an unexplored cell. Each time the value iteration algorithm is called, only the V values around the robot are initialized, in a similar way to the *bounding box* described in [8].

3 Navigation

Once the map is complete, the same algorithm used for exploration can be used for navigation. In this case, the goal cell takes the place of the unique unexplored cell (a zero value for variable V). In contrast to the exploration phase, during navigation there is no update of the PGM or the travel space.

A significant reduction in the number of free cells to be updated in the value iteration algorithm can be achieved using the travel space associated to the map built. The key idea is to consider travel cells as a kind of roadmap (as defined in [3]), a net of roads that the robot will use most of the time to go from one place to another. Some issues must be solved in order to build the roadmap and use it for navigation. First, how to find the cells of the roadmap where there are no travel cells in the travel space. In narrow passages there are only warning cells and the environment can have isolated sets of connected travel cells (see Fig. 3(a)). Second, how to consider the uncertainty in the position of the robot during navigation. Finally, how to handle cases where the initial and goal position are not within the roadmap.

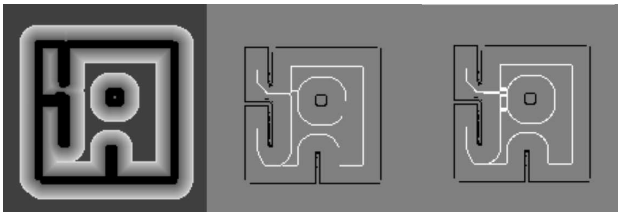

Figure 3: Building a roadmap. From left to right: (a) Travel space. (b) Partial roadmap. (c) Full roadmap

The following method solves the first problem. Given the distance D_t (the distance between a travel cell and its closest occupied cell) and a cell G of the roadmap (i.e. a travel cell), we can apply the value iteration algorithm to get a policy of movements to reach cell G. Using this policy, from each warning and travel cell there is a path of cells to the cell G. All the cells of these paths except the first cells (considering the length D_t) form the roadmap. Figure 3 (b) shows the roadmap for the travel space of Fig. 3 (a) using this method. This is a partial roadmap because there are cells missing for each *loop* of the full roadmap. Considering as the goal cell G each of the end cells of the partial roadmap a full roadmap can be built with an OR operation of the partial roadmaps (see Fig. 3 (c)).

The second problem can be solved if the cells in the roadmap grow in a similar form to the occupied cells in the travel space. In this case, the robot radius is the maximum uncertainty of the robot position.

The last problem can be handled if for each free cell that is not in the roadmap, it is computed the distance d_{min} to the nearest cell in the roadmap (this can be estimated using again the value iteration algorithm without costs for direction changes and type of cells, considering the cells in the roadmap as the unexplored cells). For a given cell, the cells in a circle of radious d_{min} connect the given cell to the roadmap. This approach solves efficiently the case of very close initial and final positions.

4 Experimental Results

This section presents the results obtained using a mobile robot simulator and a real mobile robot. The mobile robots have odometer, ultrasonic and laser range sensors (implemented with laser pointers and a camera). The implemented system, used to test the exploration and navigation approach, follows the ideas described in [6] for sensor data fusion and position tracking.

Figure 4 (a) shows the PGM built by the simulated robot without using the travel space (i.e. assigning null costs to warning and travel cells). The grid cells are 10x10cm^2 and the map is 10x10m^2. The simulator introduces an uniform random error on displacements of $\pm 10\%$ and a uniform random orientation error of about ± 7 degrees, per movement. The lighter trace on the map is given by the odometer and it shows the path followed by the robot. Note that sometimes the robot gets very close to obstacles. Figure 4 (b) shows the map built using the travel space and Fig. 4 (c) shows the map built using a Super Scout Mobile Robot within an office environment with desks, chairs, bookshelfs, etc. (note the effect of a glass door in the lower right corner). In these cases, warning cells have costs in the interval $[13, 1]$ (a linear function was used to estimate costs depending on the distance from the cell to the nearest occupied cell), travel cells have a cost of 0.001 and far cells have a cost of 6. The warning cells form a layer of 100 cm. and the travel cells form a layer of 20 cm. These cost values implement the wall following strategy to explore the environment, as can be observed in the map. Also the robot does not get too close to obstacles even in narrow passages. Instead, in narrow passages (where there are only warning cells) the robot tends to maximize the clearance between the robot and the obstacles. The map of the Fig. 4 (b) is also more accurate than the map built without using the travel space (Fig. 4(a)). This is because the travel space approach tends to move the robot to positions where the sensor readings are more reliable and hence the position tracking algorithm gives better estimations.

Some experiments were performed to evaluate the changes due to Kp_c, the cost of making orientation changes in the robot movements, during the exploration phase using the simulator. In these experiments we assign a cost of Kp_c for rotation of 45 degrees, $2Kp_c$ for 90 degrees, and so on. Table 1 shows the results, considering the length d (in cm.) of the path followed by the robot, the total number of movements made by the robot (n), the amount (θ) of orientation changes (in 45 degrees units) made by the robot, and the ratio (θ/n). These results suggest that higher Kp_c values tend to decrease the number of movements that change the orientation of the robot. The effect of Kp_c is analog to the effect of *inertial mass*: it tends to keep the orientation of the robot unchanged.

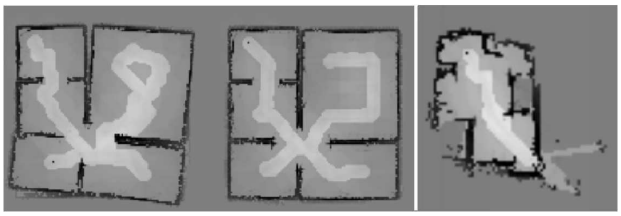

Figure 4: PGMs. White areas represent cells with occupancy probabilities near to 0. From left to right: (a) Using the simulator without the travel space. (b) Using the simulator and the travel space. (c) Using the real mobile robot and the travel space

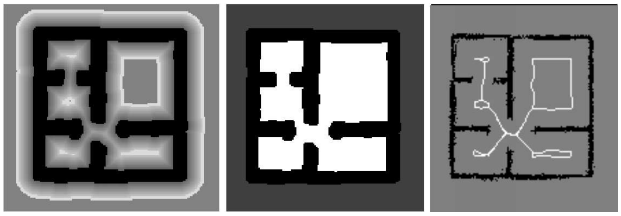

Figure 5: Roadmap in the simulated case. From left to right: (a) Travel space associated to map of Figure 4 (b). (b) Free cells extracted from (a). (c) Roadmap build upon the travel space shown in (a)

Kp_c	d	n	θ	θ/n
0	3352	284	162	0.5704
1	3640	310	163	0.5258
2	3596	304	155	0.5098
3	3536	306	145	0.4738

Table 1: Some experimental results for different costs of orientation changes (Kp_c)

The travel space associated to the map of Figure 4 (b) is shown in Figure 5(a). Figure 5 (b) shows the free cells where the robot can move for the map shown in Fig. 4 (b), and Figure 5 (c) shows the roadmap extracted from the travel space, using the ideas described before. In this roadmap there is a significant reduction from 6386 cells in the free cells set, to 371 cells. Figure 6 shows the travel space and the roadmap for the map built using the real mobile robot (Fig. 4 (c)).

Figure 7 shows three steps of the navigation

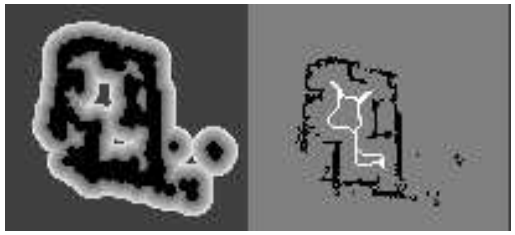

Figure 6: Roadmaps in the real case using a warning layer width of 50 cm. From left to right: (*a*) Travel space associated to map of Fig. 4(c). (*b*) Roadmap built upon the travel space shown in (a)

Figure 7: Using the roadmap for navigation. From left to right: (*a*) Roadmap with an uncertainty of 20 cm. in the position of the robot. (*b*) The cells of the roadmap and the cells that connect the initial and final cells. (*c*) Values of V given by the value iteration algorithm (dark pixels denote high values)

method. Figure 7(a) shows the roadmap (simulated case) built with an uncertainty on the robot position of 20 cm. Figure 7 (b) shows the circles that connect the initial and goal cells to the roadmap (left and right circles respectively) and Figure 7 (c) shows motion policy to reach the goal cell given by the value iteration algorithm. Darker pixels denote higher values of V that represent higher accumulated costs to reach the goal cell. Note that the goal cell has a null cost and it is represented by a white pixel.

5 Conclusions

A new approach for a mobile robot to explore and navigate in an indoor environment that combines local control (via cost associated to cells in the *travel space*) with a global exploration strategy (using a dynamic programming technique) has been described.

As the experimental results confirm, the exploration follows a kind of *wall following* technique to reduce uncertainty in terms of localization, as well as to guide the robot through narrow passages, maximizing the distance between the robot and the obstacles. This combination of local and global strategies takes the advantages of both: robustness of local strategies and completeness of global strategies. In addition, a heuristic to minimize the number of orientation changes, trying to minimize the accumulated odometric error, is also introduced.

A preprocessing of the travel space that results in a *roadmap* is used for navigation purposes. This step reduces the number of cells to be updated in the value iteration algorithm, and significantly reduces the time to compute the motion policy for the robot.

In the future, we plan to consider different uncertainties for different positions of the robot (bigger uncertainties for positions farther from obstacles) and to extend the navigation approach to dynamic environments.

References

[1] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *IEEE Computer*, 22(6):46–57, 1989.

[2] J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proc. International Conference on Intelligent Robots and Systems (IROS'98)*, 1998.

[3] J-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[4] D. Lee. *The Map–Building and Exploration of a Simple Sonar–Equipped Robot.* Cambridge University Press, 1996.

[5] P. J. McKerrow. *Introduction to Robotics.* Addison-Wesley, 1991.

[6] L. Romero, E. Morales, and E. Sucar. Learning probabilistic grid–based maps for indoor mobile robots using ultrasonic and laser range sensors. In O. Cairo, E. Sucar, and F.J. Cantu, editors, *MICAI2000, LNAI.* Springer–Verlag Berlin, 2000.

[7] Nicholas Roy, Wolfram Burgard, Dieter Fox, and Sebastian Thrun. Coastal navigation – mobile robot navigation with uncertainty in dynamic environments. In *Proc. IEEE Conf. Robotics and Automation (ICRA)*, May 1999.

[8] S. Thrun. Learning maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.