Autonomous Robots, 1(2), 1995, 149–165

Robot Navigation in a Known Environment with Unknown Moving Obstacles *

Steven Ratering and Maria Gini
Department of Computer Science, University of Minnesota, Minneapolis, MN 55455
gini@cs.umn.edu

Abstract

We propose a new type of artificial potential field, that we call hybrid potential field, to navigate a robot in situations in which the environment is known except for unknown and possibly moving obstacles. We show how to compute hybrid potential fields in real time and use them to control the motions of a real robot. Our method is tested on both a real robot and a simulated one. We present a feature matching approach for position error correction that we have validated experimentally with our mobile robot. We show extensive simulation results with up to 50 randomly moving obstacles.

1 Introduction

To be useful in the real world, robots need to move safely in unstructured environments and achieve their given goals despite unexpected changes in their surroundings. The environments of real robots are rarely predictable or perfectly known so it does not make sense to make precise plans before moving.

The robot navigation problem can be decomposed into the two problems of:

Getting to the goal. This is a global problem because short paths to the goal generally cannot be found using only local information. The topology of the space is important in finding good routes to the goal.

Avoiding obstacles. This can often be solved using only local information, but for an unpredictable environment it cannot be solved in advance because the robot needs to sense the obstacles before it can be expected to avoid them.

Some have solved the navigation problem by solving these two sub-problems one after the other. A path is first found from the robot's initial position to the goal and then the robot approximates this path as it avoids obstacles. This method is restrictive in that the robot is required to stay fairly close to or perhaps on a given path. This would not work well if the path goes through a passageway which turns out to be blocked by an unforeseen obstacle. Solutions that are only local or reactive [6] can lead the robot into local minima traps. Solutions that assume a priori knowledge of the paths of the obstacles (e.g. [10, 11, 15]), or that select a path using only information on stationary obstacles and determine the speed of the robot while following the path (e.g. [13]), or solutions that require the robot to stay within some distance from its assigned path while avoiding unknown moving obstacles (e.g. [12]), are not always sufficiently flexible to deal with situations in which an obstacle blocks a path to the goal.

In this paper we propose a very flexible solution using a common tool, the artificial potential field, in a new form that we call a *hybrid artificial potential field.* A hybrid artificial potential field is obtained by combining two different kinds of artificial potential fields, a global discontinuous potential field and a local continuous potential field.

The global potential field covers the whole floorplan and captures the static floorplan information. Since it includes only information about static objects, it can be computed "a priori" when given the goal. The

*This work was funded in part by the NSF under grant NSF/CDA-9022509, and by the AT&T Foundation.

purpose of this global field is to "pull" the robot in a direction that will lead it to the goal. This direction may not be directly toward the goal if there are known obstacles in the way.

The local field captures local information obtained by sensors and covers only the area around the robot. The purpose of this local field is to "push" the robot away from obstacles that are on its path. This field has to be computed repeatedly as the robot moves in the workspace.

We use a grid-based representation to compute these artificial potential fields and we combine them by addition. Grid-based representations have been used successfully by many for robot navigation. Examples include [3, 23, 29] just to name a few. The grid representation allows for easy integration of sensory information, as proposed by Moravec [24]. Zhang and Webber [30] have used occupancy grids and the Hough transform to handle moving objects, but their method is too time consuming and works only with a few velocities and slow objects.

Since dead-reckoning is never perfect in robots, we have developed a simple but effective localization method that keeps the positional error of the robot low.

Our approach combines the advantages of reactive methods with the ability to avoid local minima that is given by global information. The global field finds the best solution possible using the information available at the time it is computed. The local field makes the robot reactive. Although local minima may appear when the two fields are combined, these minima tend to be shallow and thus easier to escape than the local minima traditionally found when using only local methods.

The need for the global potential field increases the computational complexity of our method, since the complexity depends on the number of grid elements. The fineness of the grid is a tradeoff between reducing computational costs and achieving maneuverability in tight quarters. Hierarchical grid representations, such as those proposed by Moravec in [24], could be used when navigating in large spaces. Faster parallel algorithms could be employed for the wavefront propagation algorithm used to compute the global potential field.

The ability to avoid many fast unforeseen obstacles is a unique feature of our method. Our method is ideally suited for indoor environments, like a large crowed building, but it could be extended to outdoor navigation. Payton's gradient fields [25], that are similar to our global potential field, have been successfully used for navigation in large scale space and our method could easily be expanded to allow for uneven terrain.

Our solution works well in cases with many obstacles that move in unpredictable paths and other stationary obstacles that block some of the paths to the goal, but our method does not guarantee success, i.e., the robot may not reach the goal, and if it does, it may not avoid all collisions. A sufficiently fast obstacle could run into the robot, no matter what the algorithm is. When obstacle movements are not known and not predictable, even if their velocity is bounded, no algorithm can guarantee the robot will reach the goal. A slow obstacle may block the only doorway to the goal. Even if there exists infinitely many obstacle-free paths through space and time from the initial position to the goal position and these paths do not require velocities or accelerations greater than the robot's capacity, still no algorithm can guarantee success in an unpredictable environment.

Consider, as an example, the class of situations where the robot is in one room and the goal is in an adjacent room. Two doors are the only connections between the rooms. At time 0.0 seconds one of the doors will open, at time 1.0 seconds that door will close. At time 2.0 seconds one of the doors will open, at time 3.0 seconds that door will close, and so on. Suppose it takes a full second for the robot to pass through a door if the robot is next to the door when it opens. It takes some finite amount of time for the robot to move from next to one door to next to the other. Given any robot navigation algorithm and any arbitrarily long amount of time, there always exists at least one situation of the class just described where the robot will not be able to get to the goal without running into a door.

2 Artificial Potential Fields

Artificial potential fields have been used extensively to solve robot navigation problems by determining the positions the robot should move through or by controlling the forces that move the robot [19]. Khatib [16] pioneered the use of potential fields in a force control context. His robot was directed to move as if the goal were generating an attractive force and the obstacles were generating repulsive forces. Each of these forces could be represented by a surface in three-dimensional space in which the negative of the gradient points

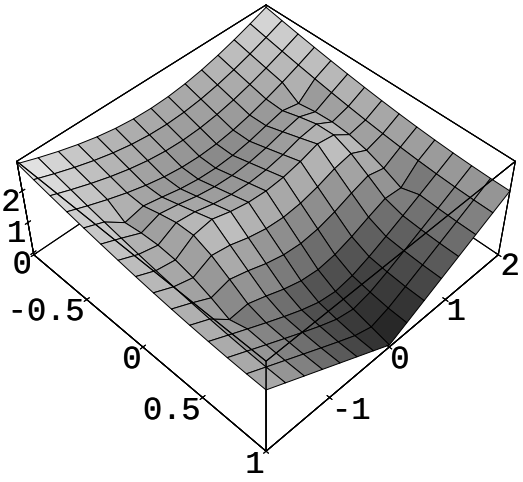


Figure 1: A wall generates a local minimum in a continuous potential field. The robot is in the middle of the upper left edge, the goal in the middle of the lower right edge, and there is a wall between the robot and the goal. The grid on the surface is shown for convenience of visualization.

in the direction of the force. The attractive force is represented by a valley and the repulsive forces are represented by hills. The robot rolls downhill until it reaches a minimum point and then it stops.

Brooks [6] has popularized the use of potential fields for navigation. In his approach, the artificial force is used for position control and not force control. The range information returned by sensors is transformed into an artificial repulsive force and added to other artificial forces acting on the robot. The main advantage is that it is easy to compute the artificial force acting on the robot without having to precompute the whole potential field [1].

The primary problem is that there may be minima other than the goal so the robot may roll to some spot other than the goal and stop, as shown in Figure 1. The robot is in the middle of the upper left edge, the goal in the middle of the lower right edge, and there is a wall between the robot and the goal. Walls between the robot and the goal will typically generate local minima.

There have been many attempts to get around this problem. Koren and Borenstein [20] analyze the problems inherent in the use of potential fields and describe a method called Virtual Force Field that allows fast motions among unknown obstacles. Arkin [2] has proposed escape and dodging behaviors based on potential fields to allow navigation with some moving obstacles. Kosla and Volpe [17] use superquadratic functions to surround obstacles, to eliminate many local minima. Connolly [8] uses Laplace's equation to build a potential field without local minima and uses relaxation over a grid to find numerical solutions. Koditschek [18] showed that no analytic potential field function will direct a point robot in two dimensions with stationary obstacles of arbitrary shape to goal points while avoiding all obstacles.

Slack [28] has proposed Navigation Templates (or NaTs). A NaT plan for navigation requires that the robot surroundings be described as a set of convex obstacles, each with an assigned spin (clockwise or counter-clockwise). The spin is used to give the direction the robot should pass the obstacle. The method is strictly local, so the robot can get trapped, but it handles unknown moving objects. Results with a real robot navigating in an outdoor parking lot at a maximum speed of 20 cm/sec have been reported [4]. No indication was given on the speed of the obstacles.

Barraquand and Latombe [3] pioneered the use in robotics of a different type of potential field, that they call "numeric". This potential field is computed in a discretized workspace and is used as a heuristic function by a search algorithm. The space is represented as a grid of free and occupied cells and the goal cell is given the value zero. The neighbors to the goal (that are free) are given the value one, their neighbors (that are free and unassigned) are given the value two, and so on. Eventually, each free cell will be assigned the city block distance of the shortest obstacle-free path from that cell to the goal. We call this a discontinuous

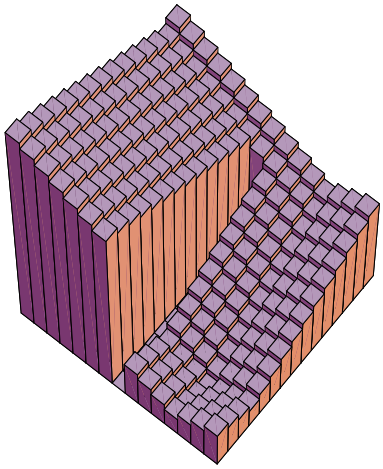


Figure 2: A wall generates a cliff in a discontinuous potential field. The value of the field has been set to 0 where there is a wall for ease of visualization.

potential field because the potential field is undefined for the occupied cells and there may be "cliffs", sharp discontinuities, generated by walls as shown in Figure 2. This potential field has no minima other than at the goal.

The algorithm for computing this potential field is based on one of the steps of Lee's routing algorithm [22], a popular method for routing wires in VLSI circuits. Many other authors have used it in robotics, mainly for navigation in 2D environments (see, for instance [29], who attributes the introduction of this method to the robotics community to [14]). Zelinsky and Yuta [29] combine this type of numeric potential field with a local map constructed from sensory data. Obstacles on the planned robot path are avoided by local replanning. The method assumes the obstacles are fixed.

Perhaps the approach most similar to ours is the method of gradient fields proposed by Payton [25]. Gradient fields are similar to Barraquand's potential fields, but more general since grid elements can have a different traversal cost that is obtained from a digital terrain map. Payton allows for replanning the path when new information comes in through sensors, but he assumes the obstacles are fixed. This allows for faster recomputation of the gradient fields, since only the part affected by the new information has to be recomputed. Payton suggested combining gradient fields with potential fields computed locally using sensor information (that would correspond to our local potential fields).

3 Robot

Before explaining how these artificial potential fields are constructed and used for navigation, we need to define more precisely our assumptions and describe our experimental setup.

We assume the robot is given one goal at a time, and the goal is expressed as a position for the robot to reach. We assume the environment contains known obstacles such as walls and large pieces of furniture, but it may also contain unknown obstacles such as people, other robots, or moved pieces of furniture. The obstacles are unknown until they are detected by the robot sensors, and their movements are unpredictable.

We make no assumptions about the movement of the obstacles because movement of people is often unpredictable. The obstacles are allowed to move rapidly, even faster than the maximum speed of the robot. The obstacles do not make any attempt to avoid running into the robot, so it is always the robot that has to move out of the way to avoid them.

Both our real and simulated robots can move forward and backward and turn while moving, but they cannot move sideways.

The sensors we use are ultrasonic rangefinders. Since we need to control the robot in real time, we do not perform much processing on the sensor data. This implies that the information obtained from the sensors is, at times, incorrect and it is also not sufficient to identify the nature of the objects. This make it impossible to do more sophisticated predictions of obstacle movements, but it takes little processing time.

The real robot we used for our experiments is a TRC Labmate, named Eric the Red. The robot size is 78 cm x 78 cm. Its maximum linear speed is 1 m/sec, and its maximum angular speed is 128 degree/sec.

Eric is controlled by a Sun SPARC 4/330 via two RS-232 serial ports. One port is used for communication with the mobility subsystem, the other for the proximity subsystem. The robot has 24 Polaroid ultrasonic sensors mounted in a 28 cm diameter ring about 60 cm above the floor. The sonars are mounted at intervals of 15° around this ring oriented along the radii through them. The sonars have a range of 15 cm to 10 m.

Eric runs in different modes. The most flexible mode allows repeatedly sending jog commands, each specifying the forward velocity (in mm/sec) and the turning velocity (in degrees/sec). With a turning velocity of 0 the robot moves on a straight-line path, otherwise the robot will move approximately on a circle whose radius is determined by the ratio of forward and turning velocities.

4 Navigation Using Hybrid Potential Fields

From our discussion before, it should be clear that the problem with the continuous version of artificial potential fields is the presence of local minima. A discontinuous artificial potential field does not have this problem, but it does not handle moving obstacles.

The method we present uses a hybrid between those two artificial potential fields. We compute a discontinuous potential field as a function of the floorplan and a continuous potential field as a function of the sensor readings. Since the floorplan does not change much, but the sensor readings change frequently, we refer to these two potential fields as static and dynamic.

4.1 Static potential field.

We first build the *static potential field* based on the floorplan. The algorithm to compute it requires the floorplan to be represented by a grid of occupied and free cells. All cells that are fully or partially occupied are labeled occupied. Then we expand the obstacles by the radius of the robot so that if the center of the robot is in a free cell the rest of it will also be in free cells. Each free cell is assigned the city block distance of the shortest obstacle-free path from that cell to the goal, so this potential field has no minima other than at the goal. We assume the floorplan is bounded by obstacles or walls.

The static potential field will "pull" the robot toward the goal (perhaps not directly) from any free location in the workspace. The value of the potential field in any cell is the shortest city block distance (in cells) to the goal.

The size of the array we use to represent the static potential field is 30 by 25 for our real robot and 100 by 100 for our simulated robot. For the real robot, each cell corresponds to 1 square foot in the real world (or 30.48 cm^2). In the simulation each cell corresponds to 25 cm^2. The size of the real robot is 78 cm^2, the simulated is 100 cm^2,

The algorithm for computing the static potential field in shown below.

Build Static Potential Field:

- *Step 1.* Initialize each element of the array SPF that is not inside an obstacle to $Undefined$. Create the queue used for computation.
- *Step 2.* Insert in the queue the x and y coordinates of the goal and its potential value 0.
- *Step 3.* While the queue is not empty, do:
  - Remove three elements from the front of the queue. Call them x, y, and n.
  - If the value of $SPF[x, y]$ is $Undefined$ then

- Assign n as value of $SPF[x, y]$.
- For each of the four neighbors of the cell of coordinates x and y that is not inside an obstacle and that has an $Undefined$ potential value, do:
  - insert into the queue the x and y coordinates of that cell and its potential value $n + 1$.

The time needed to compute the static potential field is proportional to the size of the workspace. For our real robot this takes 0.03 seconds, but for the simulated robot, since the grid has more elements, it takes almost 1 second. The static field is computed at the beginning and then recomputed only when stable unknown obstacles prevent the robot from reaching the goal, as described later in Section 5.

4.2 Dynamic potential field

The *dynamic potential field* is constructed using sensor readings from the sonar sensors. Since this potential field is continuously updated its size is much smaller than the floorplan. Each sensed obstacle generates a "hill" in the dynamic potential field. These hills "push" the robot away from all sensed obstacles, both stationary and moving. Since these hills are built as the obstacles are sensed, this method works with obstacles that move in unknown, unpredictable paths as well as other unknown fixed obstacles.

We represent this potential field as a discrete grid where the cells have the same size as the cells of the static potential field so that the two fields can easily be added together. This field is centered on the robot's current position. The size we use is 7 by 7 cells for the real robot (that corresponds in the real world to an area of approximatively 210 cm^2) and 9 by 9 cells for the simulation (that corresponds to 225 cm^2). A smaller size will reduce the computation but make the robot more "blind" and increase the chances of collision.

The procedure to build the dynamic potential field examines each cell c near the robot. If there is an obstacle in c, then a hill is built in the dynamic potential field centered at c. The radius of the hill depends on its extent, and its height depends on the extent and slope. The extent is the number of cells the hill extends beyond the expanded obstacles, the slope gives the difference in height between two adjacent cells. The height of the hill is the value of the potential field.

The height of hills inside obstacles or within the robot radius is essentially infinite. The height of hills at other points is computed using two parameters, the extent and the slope. A point obstacle centered at [0,0] generates at point p a dynamic potential field given by:

$$DPF(p) = max[(extent - \|p\|) * slope, 0] \quad (1)$$

The default extent we use is 8 cells and the slope is 3. Slope and extent can be modified by the user, depending on features of the environment and depending on how far away the robot should stay from obstacles. The extent is dynamically reduced if the robot can't get through a narrow passageway, as described later in Section 5.

Hills are combined taking the cell by cell maximum, as shown in the algorithm described here below.

Build Dynamic Potential Field:

- *Step 1* Initialize each element of the array DPF to 0. The dimension of the array is usually 7 by 7.
- *Step 2* Compute the portion of the floorplan that could contain relevant obstacles. An obstacle is considered relevant if the hill it generates overlaps the dynamic potential field. The portion of the floorplan depends on the size of the robot, the size of the dynamic potential field, and the maximum extent of hills.
- *Step 3* For each cell in the portion of the floorplan computed above, do:
  - if there is an obstacle at that cell then compute the extent of the hill generated by that obstacle. To find if there is an obstacle we use the current obstacles grid, that we describe later in Section 6. The computation of the extent is described in Section 5.
  - for each cell within the dynamic potential field, do:

- if it is within the robot footprint, then set the value of the potential field to be infinite.
- else, compute the value of the potential field using equation (1).
- if the new value is larger than the current value of DPF for the cell, then assign the new value as the value of the dynamic potential field for the cell.

5 Dealing with Local Minima

Unfortunately, when the static and dynamic potential fields are combined by adding them element by element, the result may have local minima. The local minima are caused by:

1. false sensor readings or transient obstacles,
2. hills that are wider than they need to be, and
3. sensed obstacles that are not in the floorplan.

In general, the minima are shallow, so it is not too difficult to escape them. We have developed the following strategies to deal with local minima.

False sensor readings or transient obstacles. If a local minimum is caused by a false sensor reading or by an obstacle that will get out of the way soon, then the robot should simply wait until its current location is no longer a minimum. Since we do not know if a reading is false, the robot will have to wait, even if there is no obstacle.

Hill extent too wide. If the minimum is caused by a hill that is too wide, the hill's extent should be reduced. This happens, for instance, near a doorway or a narrow passage. In general the extent is kept large for safety reasons, but a large extent might make it impossible for the robot to go through doorways or narrow passages. When the robot is stuck at a location and the environment appears stable, the extent is reduced. The default extent we use is 8 cells. The details of how the extent is reduced are different for the real and simulated robots. For the real robot, the extent is cut in half if the obstacle in the cell at the center of the hill appears stable (see Section 6) or is near a known obstacle. The extent is cut in half again if, in addition, the robot is stuck in one spot. For purposes of calculating the extent, we consider the real robot to be stuck if a weighted average of the forward velocity commands (see Section 8) is less than 5 cm/sec. Half of the weight for this average is for the most recent command, one-fourth of the weight is for the command of the previous control loop, one-eighth is for the iteration before that, and so on. For the simulated robot, the extent is cut in half after 10 iterations of the control loop if the robot hasn't moved more than 250 cm and the obstacle in the cell at the center of the hill either appears stable or is near a known stationary obstacle. The extent is again cut in half after 10 more iterations if the robot still hasn't moved more than 250 cm from where it was 20 iterations ago and the obstacle still appears stable. We consider the simulated robot to be stuck if it has not moved more than 250 cm in 30 iterations of its control loop. By modifying the parameters that specify how long the robot will wait in a local minimum, one can modify how "patient" the robot is.

Sensed obstacles not in the floorplan. If the minimum is caused by an obstacle that continues to be in the way, the robot should find another path to the goal. We force the robot to find a new path by recomputing the static potential field. When the static potential field is recomputed, we add to the floorplan obstacles that appear stable, and we remove obstacles that were previously added but have since moved out of their previous location. It may be the case that the obstacles added to the floorplan block every path to the goal and no new paths are opened up by removing other obstacles. In that case, we go back to the original floorplan and original static potential field and keep trying. This way, if all paths to the goal are blocked for some time, and then one or more open up, the robot still has a good chance of reaching the goal. If this "persistent" attitude is not desired the algorithm can be modified so that the robot gives up sooner.

6 Stable Obstacles

To deal with local minima we need to distinguish between stable and moving obstacles for two reasons. The first reason is that after the robot is stuck for some time we want to reduce the extent of the hills of the stable obstacles but not of the moving obstacles. The second reason is that when the static potential field is recomputed, the stable obstacles are added to the floorplan, but the moving obstacles are not.

Since the sensors on the robot do not allow us to recognize the obstacles and to track them while they move, we use a much simpler approach based on the use of a *histogram grid*, as proposed in [5], and a *current obstacle grid*. Both grids extend throughout the floorplan. The histogram grid is a trace over time of what has happened in the robot surroundings, with the past fading as times goes by, while the current obstacles grid is a snapshot of the current situation.

A *histogram grid* is initialized to zero, and for each sensor reading, one cell is incremented by three (not beyond some maximum), and the cells between the robot and that incremented cell are decremented by one (not below zero). The cell that is incremented is at the distance given by the sensor reading and the angle given by the direction the sensor was pointing when the reading was taken. Large values in the histogram grid correspond to stable obstacles; small values correspond to moving obstacles, or obstacles that have not been observed much, or previous positions of obstacles that have since moved.

A *current obstacles grid* is similar to a histogram grid, but the possible values of a cell are simply one for occupied and zero for free. The current obstacles grid is initialized to zero, and brought up to date each iteration of the control loop. For each sensor reading, we set to one the cells on an arc with center at the robot, and set to zero the cells between the robot and the cells set to one. The arc is determined by the distance given by the sensor reading; the range of angles is determined by the direction the sensor was pointing when the reading was taken and by the angular range of the sensor. A more sophisticated sensor model could be used (see, for instance [21, 24]), but we have found out that this simple model is sufficient for our environment and fast to compute.

The current obstacles grid is used when building the dynamic potential field to determine where the obstacles are. For each obstacle that appears in the current obstacles grid, and that is in the portion of the floorplan covered by the dynamic potential field, a hill is placed in the dynamic potential field. The histogram grid is used to distinguish between stable and moving obstacles when computing the extent of hills.

This method is more responsive to changes in the environment than using only the histogram grid because an obstacle that moves out of a given cell has its image immediately removed from the current obstacles grid but remains awhile in the histogram grid.

When recomputing the static potential field we use only the histogram grid to determine which obstacles should be added to the floorplan. This way moving obstacles that stay in one general area and stationary objects that are missed in one iteration due to sensor error will be added to the floorplan. An example of the histogram grid and the current obstacles grid is shown in Figure 3.

In our experiments we considered a cell in the grid to contain a stable obstacle if the cell or any of its neighbors has a value greater than 8 in the histogram grid or if the cell or any of its neighbors is in a floorplan obstacle. The number of neighbors we consider is 8 for close objects and 24 for objects that are farther from the robot. This is because the sensors have a cone of sensitivity of 15° and the objects farther away have a larger uncertainty on their absolute position.

If an obstacle doesn't move for a few iterations, the robot will venture closer to it (if a path to the goal goes past the obstacle). The robot is more wary of obstacles that have moved recently and obstacles that haven't been seen before.

7 Determining Hills Extent

By default, we want the hill extent to be large so that the robot keeps a safe distance from all obstacles. However, if the extent is too large the robot will never be able to make it through narrow passageways.

Simply using small hills for stable obstacles and large hills on other obstacles does not always give good performance. Suppose, for instance, the robot is in a long corridor that turns sharply to the right, and the goal is to the right after the turn. The hills along the walls of the corridor will be of different sizes because

the portion of the wall near the robot is sensed more times than the portion down the hallway. Since the portion of the wall near the robot is sensed more times, the corresponding histogram grid values are large and the wall appears stable. The portion of the wall farther away is not considered stable because it hasn't been sensed enough times. The robot ends up repeatedly being in local minima that disappear as the robot takes more sensor readings and the extent of the hills decreases.

If small hills are used for obstacles that are in the floorplan, the robot will travel closer to the wall and have a smoother ride. However, as soon as an unexpected obstacle is added it generates a large hill and the robot will get stuck awhile in the local minimum.

The best compromise we have found to obtain smooth paths and to deal with local minima requires adjusting dynamically the size of the hills. As we have already described, we start with large hills and reduce them after the robot is stuck if obstacles appear stable. The extent of the hills we use goes from 8 cells (= 210 cm) down to 2 cells (= 60 cm).

8 Potential Fields to Robot Commands

The discrete potential field is not suitable for the direct control of the robot along smooth paths. We obtain a smooth path by examining the potential values in a small area around the robot (we use a 7 by 7 grid) and by determining the radius of curvature of all the paths that end in the cell with the smallest potential value, as shown in Figure 4.

In general there may be more than one cell in the grid with the same minimal potential value. For each of these cells we compute the turning radius necessary to move the robot from its current position to the cell where the minimum of the potential field is achieved. Any arc that will go through obstacles is discarded. If there are two or more cells with the same potential value near the robot, the one that is closest to the goal is selected.

The robot moves continuously along the arc computed in this way. This implies that the robot does not always move from a cell to the adjacent cell with minimal potential field value, rather, it moves toward the cell in its 7 by 7 neighborhood with minimal potential field value. A new arc is computed for each iteration of the control loop. The velocity of the robot along the arc is determined as follows.

Once an arc has been selected, the radius of the arc determines the ratio between the forward and turning velocity of the robot. Given a turning radius of r mm, the relationship between the forward velocity $forward$ (in mm/sec) and the turning velocity $turn$ (in degrees/sec) is:

$$turn = \frac{forward}{r} * \frac{180}{\pi} \tag{2}$$

We tentatively set the forward velocity equal to the minimum of the robot's maximum velocity $forward_{max}$ and the velocity required to move the robot to the desired position in one control loop iteration $forward_{goal}$:

$$forward = min(forward_{max}, forward_{goal})$$

We then use Eq. 2 to set the turning velocity $turn_{goal}$. If this results in a turning velocity larger than the maximum $turn_{max}$, we set the turning velocity to the maximum:

$$turn = min(turn_{max}, turn_{goal})$$

and reduce the forward velocity proportionately using Eq. 2.

9 Error Correction

Real robots lose track of where they are due to wheels that slip and slide. Our final contribution is in finding a fast method to adjust the robot's configuration variables so that the robot will not lose track of where it is.

Our robot Eric keeps track of its current position and orientation in three registers, x, y, and θ. The set of these three parameters describes a *configuration* of the robot. These parameters are automatically updated

as the wheels turn. However, the wheels slip and slide a little bit so the values in these registers slowly drift away from the robot's true position and orientation. To compensate for this error, we have devised a method for adjusting these parameters by comparing the real sensor readings with the sensor readings we expect, given the floorplan.

Given a configuration, we search for obstacles in the known floorplan and generate a list of expected sensor readings. We then compare this expected list of readings with the true sensor readings.

We count how many of the sensor readings are good, that is, are within some tolerance T of where one would expect them to be if the robot is where it thinks it is and there are no obstacles other than the ones in the floorplan. Then we count how many matches we would have if the robot were at a slightly different configuration. If we find that the number of matches would be higher at another configuration we update the robot configuration.

In our solution we take advantage of the fact that we can assume the error to be small, since we perform this comparison often.

To be more precise, the robot calculates, using the floorplan information, for each sensor i the expected sensor reading $expected_i$ for each of three different robot configurations that differ only in one of the three coordinates, x, y, and θ by a small amount Δx, Δy, and $\Delta\theta$. Then it counts how many matches there are between the expected sensor readings $expected_i$ and the real sensor readings $sensor_i$.

The number of matches $m(x, y, \theta)$ for the configuration (x, y, θ) is computed as follows:

$$m(x, y, \theta) = \sum_{i=0}^{23} c_i$$

where

$$c_i = \begin{cases} 1 & if\ |sensor_i - expected_i| \leq T \\ 0 & otherwise \end{cases}$$

If the current robot configuration yields a greater count of sensor matches, the configuration is considered correct and is not modified. Otherwise, either x, y, or θ is adjusted based on the number of sensor matches found for each of the configurations examined.

The new value for that coordinate is a weighted average of the three values considered for that coordinate. A weight of zero is used if the number of matches for the corresponding configuration is less than the number of matches for the original configuration (x, y, θ). Otherwise the weight is the square of the number of matches.

The weight $w_r(x, y, \theta)$ for a movement of Δx to the right of the current configuration is computed as follows:

$$w_r(x, y, \theta) = \begin{cases} m(x + \Delta x, y, \theta)^2 \\ \quad if\ m(x + \Delta x, y, \theta) > m(x, y, \theta) \\ 0 \\ \quad otherwise \end{cases}$$

The weight $w_l(x, y, \theta)$ for a movement of Δx to the left is computed similarly:

$$w_l(x, y, \theta) = \begin{cases} m(x - \Delta x, y, \theta)^2 \\ \quad if\ m(x - \Delta x, y, \theta) > m(x, y, \theta) \\ 0 \\ \quad otherwise \end{cases}$$

The correction Δx is then computed:

$$\Delta x = \frac{w_r - w_l}{w_r + w_l + m(x, y, \theta)^2}$$

The corrections Δy and $\Delta\theta$ are computed in a similar way for the y and θ coordinates.

We consider only three configurations per control loop iteration. Counting the number of sensor matches for a configuration takes almost 0.1 seconds, and we don't want to significantly slow down the control loop

because that would result in delayed reactions to obstacles. In one control loop iteration we consider the three configurations, $(x - \Delta x, y, \theta), (x, y, \theta)$, and $(x + \Delta x, y, \theta)$. In the next iteration we consider $(x, y - \Delta y, \theta), (x, y, \theta)$, and $(x, y + \Delta y, \theta)$. In the iteration after that we consider $(x, y, \theta - \Delta\theta), (x, y, \theta)$, and $(x, y, \theta + \Delta\theta)$, and we repeat this sequence every three iterations of the control loop.

Counting sensor matches for several configurations works better than a least squares approach when there are unknown obstacles. A least squares approach would put a lot of weight on sensor readings that greatly differ from the expected sensor readings. In contrast, counting the number of matches usually puts no weight on these readings that are probably due to unknown obstacles. If a true sensor reading is far from the expected sensor reading for a particular configuration, that reading will usually still be far from matching if the configuration is modified by some small value.

If no small change in x, y, or θ generates more matches, x, y, and θ are not updated. Thus if there are lots of unexpected obstacles there won't be much adjustment (for good or bad) because the walls won't be seen.

In our experiment we used a value of Δ equal to 30.48 cm (1 foot) for the x and y coordinate, and equal to 5° for the θ coordinate. The threshold for deciding if two sensor readings match or not is also 30.48 cm.

More sophisticated methods for localization have been proposed (as, for instance, in [9], [27]), but we have found that our error correction approach works well and is fast to compute. Drumheller's [9] method takes much longer to acquire data and process them and still does not achieve a much better localization (the localization reported is correct within one foot in any direction and 5° in orientation). No timing and precision of results is given in [27]. Our localization is within 15 cm, even when the initial position is incorrect. More details on our experimental results are given later in Section 11.

10 Control Loop

The control loop usually includes just the following steps: read the sensors, adjust the position and orientation, update the current obstacles and histogram grids, build the dynamic potential field, calculate the control parameters for the robot, and send the command to the robot. The time for this control loop is not a function of the number of obstacles nor of the size of the workspace.

With our current robot and computer, reading the sensors takes approximately 0.27 seconds. Adjusting the robot configuration takes between 0.20 and 0.30 seconds, depending on the size of the area seen by the sensors. Updating the current obstacles and histogram grids also take time proportional to the area seen by the sensors. This usually takes between 0.01 and 0.04 seconds. Building the dynamic potential field takes $O(kn^2)$, where n is the size of the dynamic potential field and k the number of hills (at most n^2). This usually takes between 0.03 and 0.07 seconds. Calculating the parameters for the robot movement takes $O(n^2)$, and usually is less than 0.01 seconds. Sending the commands to the robot takes between 0.04 and 0.07 seconds. The entire loop usually takes between 0.6 and 0.8 seconds.

When the robot gets stuck, the static potential field might have to be recomputed. This takes time proportional to the size of the workspace. For our real robot this takes 0.03 seconds, but for the simulated robot, since the grid has more elements, it takes almost 1 second. This computation could be easily performed on a second processor, so keeping the robot reactive to the environment even during the recomputation.

11 Results from the Real Robot

Our robot Eric is confined to a lab, connected to its controlling computer by an umbilical cord. The lab is approximatively 25 by 30 square feet, and contains a number of tables, cabinets, chairs, and boxes. The layout of the furniture in the Lab has been changed a number of times, as reflected by the figures showing Eric at work. The cabinets doors and whiteboards tend to reflect the ultrasonic beams, as discussed in [7]. Despite the problem of reflections (that create the illusion of a larger space), our method has been used successfully over a long period of time. We use it whenever we have visitors in the lab, because the method works well with any number of moving objects, and thus allows our visitors to move freely in the lab while the robot reaches its goal. Eric never had any trouble in navigating even in a crowed room. We have not tried, for safety reason, to have objects running into the robot, as we do in our simulation.

The floorplan of the Lab is given in Figure 5. Also shown in Figure 5 is a path Eric took starting from near the upper right corner of the lab. Eric soon got stuck in a local minimum behind two boxes and then it added the obstacles to its floorplan and recomputed the static potential field and then moved around the boxes.

We repeated the experiment ten times, and Eric took similar paths (same starting and ending positions) and on the average it ended up about 15 cm from where it "thought" it was.

Figure 6 shows the path Eric took when moving boxes and a moving person continued to block Eric's path for three minutes before getting out of the way. After ten similar paths, it again was off by about 15 cm. For the trials mentioned above, Eric regularly adjusted its position and orientation by comparing the sensor readings to the floorplan. This adjustment was usually unnecessary for the first case, but for the second case with the "determined" obstacles, when there was no adjustment, Eric was off by more than 90 cm on the average.

Our adjustment algorithm not only keeps Eric on track, but it can also get Eric back on track if the initial configuration it is given is incorrect. We ran a number of trials with the initial orientation off by $5°, 10°, 15°, 20°$, and $25°$. Each trial was repeated three times. Eric stopped about 15 cm from where it "thought" it stopped on the average when the goal was about 450 cm from the starting position. With perfect dead reckoning and no correction based on sensor readings, Eric would have been off almost half a meter and almost 2 m for the initial errors of $5°$ and $25°$, respectively. For safety reasons, we limited the maximum speed of the robot in the experiments to 25 cm/sec.

A view of the lab during another experiment is shown in Figure 7. The figure is a screen dump of the display window. The black part of the figure indicates the fixed furniture in the room. The empty area shown represents the part of the lab where currently Eric can move. The rest of the space is occupied by fixed furniture. In this run there are two persons close to the initial position of Eric, and an unknown box sitting in the corner of the room.

Figure 8 shows the display window at a later point after Eric has reached its goal. Since there were obstacles near the goal, Eric had to wait awhile and then back up to avoid getting too close to them. Because of the limited free space, the hill extent has been set to 50 cm. During Eric motions the display window is continuously refreshed to show sensor data and the path taken by Eric.

12 Results from the Simulated Robot

To evaluate the performance of our approach we have performed a number of simulation experiments. This has allowed us to study in a systematic way the effect of the parameters we use, and to experiment with a large number of moving obstacles. In our simulation studies, the size of the grid is 100 by 100 cells, and each cell is 25 cm^2. The size of the robot is 4 by 4 cells.

In our simulation, we use a very simple modeling of the ultrasonic sensors. Each sensor beam is modeled as a cone of $15°$, with the tip of the cone at the sensor position, and a range of 4.5 m. We simulate 24 sensors, covering all the $360°$ around the robot. Each sensor returns the distance to the closest object, either stationary or moving, within its cone. The discretization errors introduced by the grid representation, and the random movement of the obstacles make a more accurate modeling of the sensors not particularly useful.

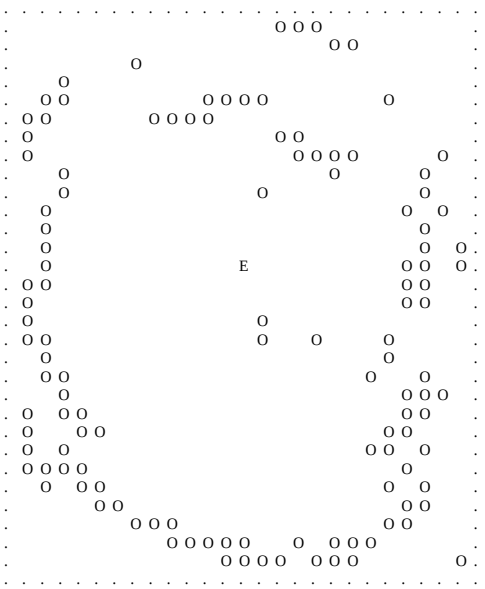
The purpose of our simulation studies is to examine the behavior of the algorithm with a large number of obstacles, not to evaluate the ability of the sensors to detect the obstacles. Since the obstacles move on random paths and no attempt is made to predict their future movements, a more accurate modeling of the sensors would produce roughly the same results as different movements of the obstacles. The odometry errors that affect real robots are also not simulated, for similar reasons. With multiple obstacles continuously moving around, the robot ends up seeing the walls of the rooms only rarely.

First, we have studied scenarios with one obstacle moving in a straight line with constant velocity and we have determined how fast the obstacle could move and still be avoided by the robot. To simplify the computation we computed the obstacle's speed to the nearest multiple of 10 cm/sec. In this set of experiments we have also changed the slope of the hills, their extent, and the direction of movement of the obstacle. A summary of the results is shown in Table 1.

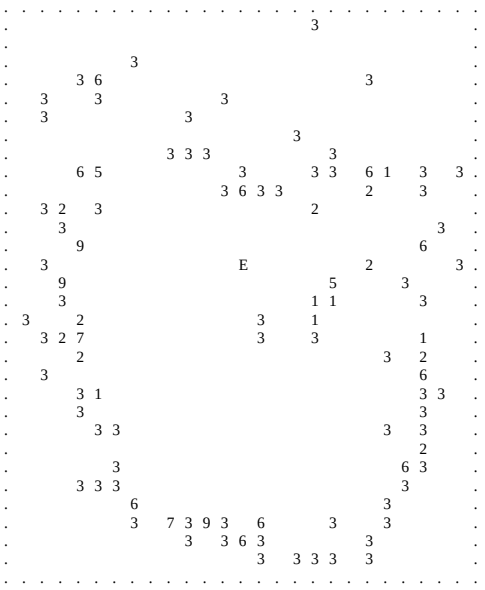
We have found that in the worst case it is possible for the obstacle to collide with the robot even if the obstacle moves slightly slower than the robot. In the best case the obstacle can move at twice the speed of

the robot and still be avoided. Increasing the extent does not always improve the situation, as the robot becomes too reactive to obstacles far away.

Examples of paths taken by the robot in two very similar scenarios are shown in Figure 9 and Figure 10.



Current Obstacles Grid



Histogram Grid

Figure 3: Histogram grid and current obstacles grid (taken during a run of the real robot).

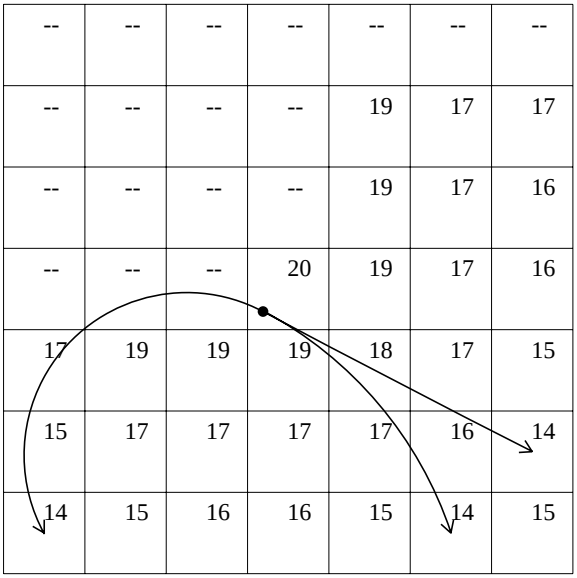


Figure 4: Possible paths to minima. The path that goes through an obstacle (shown in the figure by --) is discarded. Of the remaining paths, the path that takes the robot closer to the goal is selected.

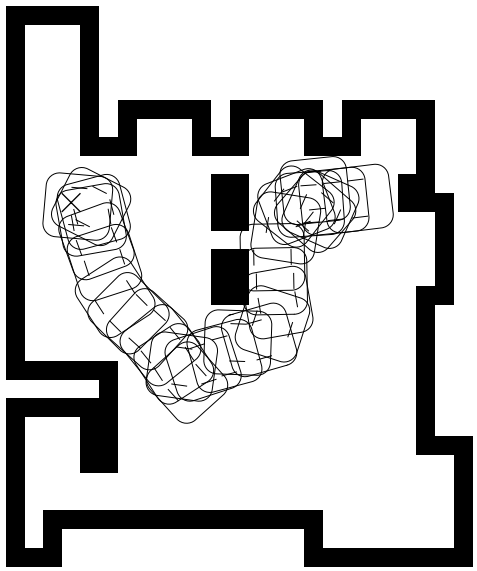


Figure 5: A path Eric took around some boxes

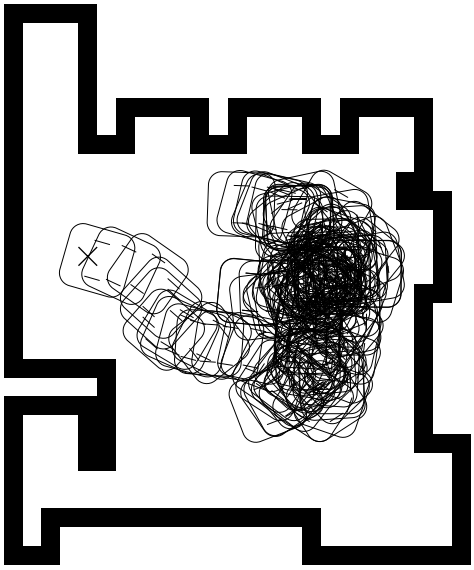


Figure 6: Eric's path when obstacles persistently got in the way

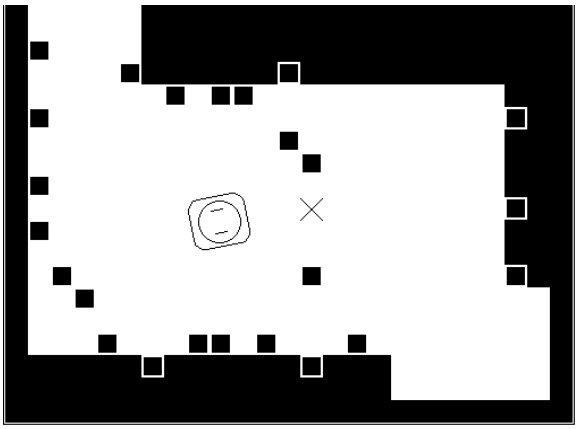


Figure 7: Eric senses the environment before starting. The goal is shown by a cross. There are two people close to its initial position and a box in the lower left corner. The figure is a screen dump obtained while Eric was moving. The black part of the figure indicates some fixed cabinets and desks. The empty area shown represents the part of the lab where Eric is free to move. The rest of the space is occupied. Each square shows the the location of a sonar hit.

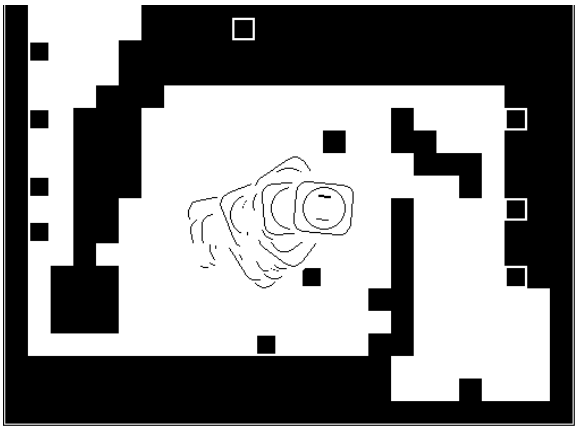


Figure 8: Eric reached its destination after a number of moves to avoid a person close to it. The black areas are either known obstacles or areas that have been seen as occupied by obstacles for more than 1 iteration of the control loop. A person has been walking on the right side of the room and the box has been moved by the other person from the lower left corner along the wall on the left.

hill slope	obstacle angle	hill extent (in cells) 6	8	10
2	45	50	50	50
	90	40	40	50
	135	60	80	100
3	45	50	50	60
	90	40	50	60
	135	60	70	80
4	45	40	40	50
	90	40	60	60
	135	60	60	80
5	45	40	40	40
	90	40	70	60
	135	70	60	50

Table 1: Fastest collision-free speeds for an obstacle (in cm/sec). The robot is moving at 50 cm/sec. The obstacle moves on a straight line at a constant velocity. To simplify the computation we compute the obstacle's speed to the nearest multiple of 10 cm/sec. Hill slope is the difference in height between two adjacent cells. Hill extent is the number of cells the hill extends beyond the expanded obstacles. Hill extent is measured in cells. Each cell is 25 cm wide. The size of the room is 100 by 100 cells.

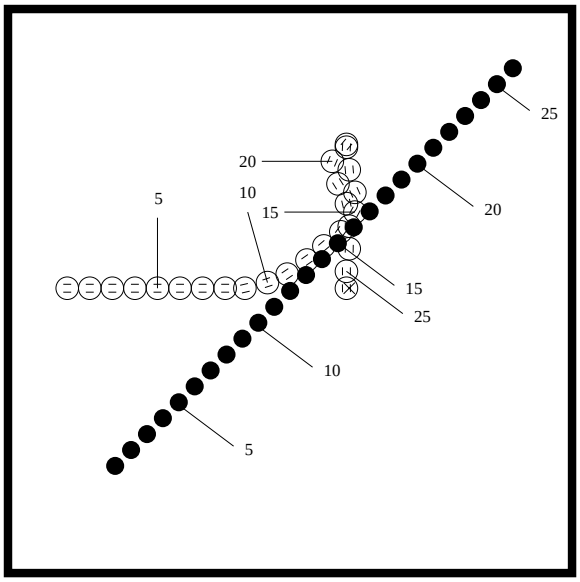


Figure 9: A path taken by the simulated robot with an obstacle coming at 45 °.

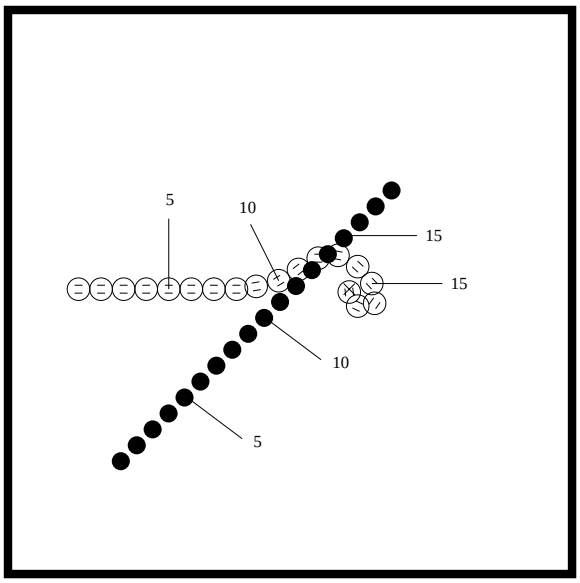


Figure 10: A different path taken by the simulated robot with an obstacle

obstacle speed (cm/sec)	no. of obstacles	average path time	st.dev. path time	no. of safe runs	no. of initial collisions	average no. of collisions	st.dev. no. of collisions
10	10	79.07	13.40	99	1	0.02	0.20
10	20	93.92	20.93	95	4	0.06	0.28
10	30	110.19	27.35	98	2	0.02	0.14
10	40	126.23	34.25	92	8	0.09	0.32
10	50	135.06	41.06	82	16	0.25	0.61
30	10	80.75	11.98	99	0	0.01	0.10
30	20	96.17	23.43	95	4	0.05	0.22
30	30	110.95	28.18	89	4	0.18	0.59
30	40	116.94	28.91	80	9	0.46	1.11
30	50	125.46	32.30	72	8	0.59	1.18
50	10	85.10	18.56	92	1	0.46	2.41
50	20	97.56	19.37	75	4	0.74	1.56
50	30	111.48	23.09	63	4	1.44	3.16
50	40	123.48	29.11	37	12	2.66	3.60
50	50	127.72	29.49	32	9	3.22	3.61

Table 2: One room statistics. The robot speed is 50 cm/sec. Each cell is 25 by 25 cm. The size of the room is 100 by 100 cells. The robot is 4 by 4 cells, the doors are 7 cells wide. The initial extent of the hills is 8 cells and gets reduced to 2 cells.

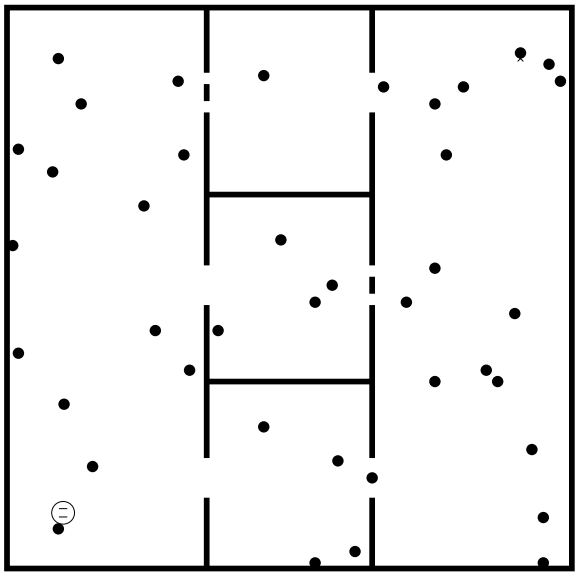


Figure 11: An initial configuration with 40 unknown moving obstacles. Two stable unknown obstacles have been added to block two of the doorways.

We studied situations in which the obstacles operate in coordination. When there are two obstacles, they could squeeze the robot between themselves and collide with somewhat slower speeds. When there are three or four obstacles, they could surround the robot at very slow speeds, and the potential field hills would prevent the robot from escaping, and then the obstacles could close in and crush the robot. More details are given in [26].

Finally, to test our method of navigation in environments with many moving obstacles we generated many scenarios with between 10 and 50 randomly moving obstacles. Each obstacle started in a random cell of the floorplan and moved in a random path consisting of short straight-line segments. The different obstacles moved in many different directions and changed directions often. Obstacles were circles with diameters of 20 cm. To simplify the simulation we allow obstacles to move across walls and to collide among themselves.

Each line of the tables 2 and 12 was derived from running 100 random scenarios. The tables give the obstacle speed in cm/sec (the robot's maximum speed is 50 cm/sec), the number of obstacles, the average times and standard deviation to reach the goal, the number of the 100 scenarios where the robot reached the goal without any collisions, the number of initial collisions, the average number of collisions and its standard deviation per scenario. The average number of collisions includes the initial collisions.

Table 2 was obtained using a floorplan of a single large room bounded by four walls and Table 12 was obtained using a five room floorplan with two blocked doorways. as shown in Figure 12. An initial configuration for the five room example with 40 obstacles is shown in Figure 11.

Some of the collisions occurred before the robot had a chance to move because some of the initial random configurations placed an obstacle on top of the robot. These are shown in the tables in the column with the number of initial collisions.

Slow obstacles (10 cm/sec) even when in large numbers do not pose a significant threat to the robot. The average number of collisions, even with 50 obstacles in a room of 25 m^2, is quite low at 0.25. As the obstacle speed increases, the number of safe runs decreases. With obstacle speeds at 30 cm/sec and 50 obstacles in the same space, 20% of the runs had at least one collision. When obstacles move as fast as the robot, the robot still has a very good chance of not colliding with any of them if the obstacles are not too many, but as the number of obstacles increases, the ability of the robot to avoid them decreases. This should not come as a surprise, considering that the obstacles are totally dumb and unpredictable, and make no effort whatsoever to avoid the robot.

Increasing the number of obstacles generally increased the time needed to reach the goal, because the robot had to take longer detours. Increasing the number of obstacles or increasing the speed of the obstacles

obstacle speed (cm/sec)	no. of obstacles	average path time	st.dev. path time	no. of safe runs	no. of initial collisions	average no. of collisions	st.dev. no. of collisions
10	10	413.99	89.30	98	1	0.03	0.22
10	20	461.98	110.54	91	4	0.14	0.49
10	30	505.54	122.94	92	2	0.08	0.27
10	40	543.53	142.93	73	8	0.51	1.47
10	50	553.34	159.35	63	16	0.56	0.93
30	10	436.50	79.89	87	0	0.33	1.03
30	20	479.21	133.73	61	4	0.96	1.92
30	30	467.59	96.08	48	4	1.61	2.35
30	40	503.21	116.93	21	9	3.13	3.03
30	50	494.46	112.68	11	8	5.25	3.86
50	10	424.35	62.07	43	1	1.83	2.53
50	20	465.34	93.47	13	4	4.64	4.31
50	30	506.38	101.76	4	4	8.44	5.60
50	40	492.43	127.58	1	12	13.95	8.31
50	50	502.46	130.56	0	9	21.50	11.09

Table 3: Five room statistics. The robot speed is 50 cm/sec. Each cell is 25 by 25 cm. The size of the floor is 100 by 100 cells. The robot is 4 by 4 cells, the doors are 7 cells wide. In addition to the moving obstacles there are two stable unknown obstacles blocking two of the doorways.

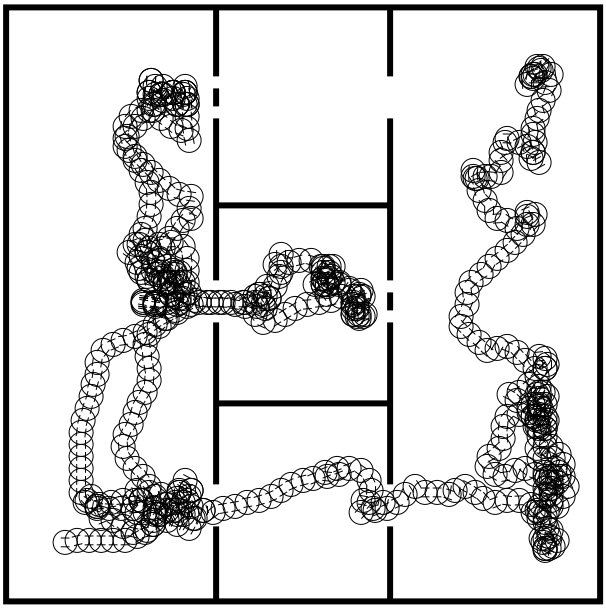


Figure 12: The path taken amidst 40 moving obstacles (not shown)

decreased the number of crash-free runs and increased the average number of crashes per run.

There were many more collisions in the five room example with blocked doorways. Most of these collisions were by doorways or walls or in the small rooms. The robot is at a disadvantage in small rooms because the space for maneuvering is limited. Since the obstacles are allowed to move across walls, the overall behavior is as if obstacles are created and consumed by the walls. The walls act not only as a barrier to the robot motion, but also to the robot ability to see. The robot is unable to see obstacles before they emerge from walls and so often does not have enough time to move away. This explains the very low success rate with fast moving obstacles shown in Table 12.

The average time to reach the goal is larger in the five room scenarios because the robot has to change path more often to avoid the obstacles. The performance of the robot degrades gracefully and becomes poor only in very difficult situations.

Figure 12 shows the path the robot took in one scenario with unknown obstacles blocking two doorways and 40 obstacles (not shown) moving on random trajectories.

13 Conclusions

We have presented a new method to allow a robot to navigate safely in a known environment with unpredictable moving obstacles. The method requires a floorplan, but does not make any assumption on the number or the motion of the obstacles. We have performed a large number of experiments both in simulation and with a moving robot equipped with ultrasonic sensors. Our experiments show that the method performs very well even when there are many obstacles moving in unpredictable paths. Its performance degrades gracefully as the speed and number of obstacles increase.

References

[1] R. C. Arkin. Motor schema-based robot navigation. *Int'l Journal of Robotics Research*, 8(4):92–112, August 1989.

[2] R. C. Arkin. Active avoidance: escape and dodging behaviors for reactive control. *Int'l Journal of Pattern Recognition and Artificial Intelligence*, February 1993.

[3] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Int'l Journal of Robotics Research*, 10(6):628–649, 1991.

[4] R. P. Bonasso, H. J. Antonisse, and M. G. Slack. A reactive robot system for find and fetch tasks in an outdoor environment. In *Proc. Nat'l Conf. on Artificial Intelligence*, pages 801–808, 1992.

[5] J. Borenstein and Y. Koren. The vector field histogram – fast obstacle avoidance for mobile robots. *IEEE Trans. on Robotics and Automation*, 7(3), June 1991.

[6] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[7] J. Budenske and M. Gini. Why is it so difficult for a robot to pass through a doorway using ultrasonic sensors? In *IEEE Int'l Conf. on Robotics and Automation*, pages 3124–3129, 1994.

[8] C. I. Connolly, J. B. Burns, and R. Weiss. Path planning using Laplace's Equation. In *IEEE Int'l Conf. on Robotics and Automation*, pages 2102–2106, May 1990.

[9] M. Drumheller. Mobile robot localization using sonar. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-9:325–332, March 1987.

[10] M. Erdmann and T. Lozano-Perez. On multiple moving obstacles. *Algorithmica*, 2(4):477–521, 1987.

[11] Kikuo Fujimura. *Motion Planning in Dynamic Environments.* Springer Verlag, Tokyo, 1991.

[12] J. Gil de Lamadrid and M. Gini. Path tracking through uncharted moving obstacles. *IEEE Trans. on Systems, Man, and Cybernetics*, 20(6):1408–1422, 1990.

[13] N. C. Griswold and J. Eem. Control for mobile robots in the presence of moving objects. *IEEE Trans. on Robotics and Automation*, 6(2), April 1990.

[14] R. A. Jarvis and J. C. Byrne. Robot navigation: touching, seeing, and knowing. In *Proc. 1st Australian Conference in Artificial Intelligence*, November 1986.

[15] K. Kant and S. W. Zucker. Planning collision free trajectories in time varying environments: a two-level hierarchy. In *IEEE Int'l Conf. on Robotics and Automation*, pages 1644–1649, 1988.

[16] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int'l Journal of Robotics Research*, 5(1):90–98, 1986.

[17] P. Khosla and R. Volpe. Superquadric artificial potentials for obstacle avoidance and approach. In *IEEE Int'l Conf. on Robotics and Automation*, pages 1778–1784, 1988.

[18] D. E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *IEEE Int'l Conf. on Robotics and Automation*, pages 1–6, 1987.

[19] D. E. Koditschek. Robot planning and control via potential functions. In *The Robotics Review*, pages 349–367. The MIT Press, 1989.

[20] Y. Koren and J. Borenstein. Potential fiels methods and their inherent limitations for mobile robot navigation. In *IEEE Int'l Conf. on Robotics and Automation*, pages 1398–1404, 1991.

[21] Roman Kuc and M. W. Siegel. Physically based simulation model for acoustic sensor robot navigation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 9(6):766–778, 1987.

[22] C. Y. Lee. An algorithm for path connections and its applications. *IRE Trans. Eletronic Computers*, September 1961.

[23] Jang Gyn Lee and Hakyoung Chung. Global path planning for mobile robot with grid-type world model. *Robotics and Computer Integrated Manufacturing*, 11(1):13–21, 1994.

[24] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[25] David W. Payton. Internalized plans: a representation for action resources. *Robotics and Autonomous Systems*, 6(2):89–103, 1990.

[26] S. Ratering. *Global Directed Robot Navigation in a Known Indoor Environment with Unknown Stationary and Moving Obstacles.* PhD thesis, University of Minnesota, 1992.

[27] B. Schiele and J. L. Crowley. A comparison of position estimation techniques using occupancy grids. *Robotics and Autonomous Systems*, 12(3-4):167–171, April 1994.

[28] Mark Slack. Navigation templates: mediating qualitative guidance and quantitative control in mobile robots. *IEEE Trans. on Systems, Man, and Cybernetics*, 23(2):452–466, March/April 1993.

[29] Alexander Zelinsky and Shin'ichi Yuta. Reactive planning for mobile robots using numeric potential fields. In *Proceedings 3rd International Conference on Intelligent Autonomous Systems (IAS-3)*, pages 84–93, Pittsburgh, PA, 1993.

[30] Y. Zhang and R. E. Webber. On combining the Hough transform and occupancy grid methods for detection of moving objects. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 2155–2160, 1992.