Path Planning for the Cye Personal Robot

Parag H. Batavia Illah Nourbakhsh

Carnegie Mellon University
Robotics Institute
Pittsburgh, PA 15212
[parag/illah]@ri.cmu.edu

Abstract

In this paper, we describe the Cye personal robot, concentrating on its path planning and navigation. We start out with a brief description of the general problem, followed by more detail on the Cye robot. After that, we concentrate on the path planner, which uses a global potential field approach, combined with a novel optimization criterion. This ensures that the robot maintains adequate distance from obstacles, while finding an optimal path which incorporates partial terrain knowledge. This system has been in use for over five months, with approximately 200 installations. As we will empirically show, it is robust and can handle dynamic obstacles and imperfect maps.

1. Introduction

Robust mobile robot navigation in real world environments is a long standing problem in robotics. In this paper, we describe research in navigation and path generation using the Probotics Cye robot. Cye is a 2-wheeled differential drive robot, whose primary mode of navigation is ded-reckoning. The only sensors are wheel encoders - there are no other active or passive sensing modes. The accuracy of the ded-reckoning is sufficient for indoor navigation, through the use of carefully designed wheels and a set of known calibration surfaces, or "checkpoints". An (inaccurate, incomplete) map of the world is interactively constructed using Cye, in which known freespace, known obstacle areas, and unexplored areas are all marked. Path generation in such an environment has the following challenges:

- The robot must maintain adequate distance from walls and other obstacles.
- There is uncertainty associated with the creation of the map, and there can be dynamic obstacles.
- The robot should ideally use explored areas, but should be willing to traverse unexplored areas if doing so results in a significant savings in path length.
- Path generation time must be short, as this is a robot which is designed for both research and consumer use.
- The use of check points must be incorporated into the path planner, to minimize ded-reckoning error.

While other planners optimize for path length [8] in the presence of obstacles, or concentrate on real-time obstacle avoidance [2], we use an optimization criterion first introduced in [7] but also used in [6] (for the case of polygonal regions of varying terrain), which optimizes the integral of path cost, where the path cost is influenced by *both* distance to obstacles and terrain traversability.

Therefore, it is possible for the lowest cost path to cut through an unexplored area of the world, if doing so would create a significantly shorter path. In this manner, the traversability of various areas can be set, and the planner will generate an optimal path using that information.

2. The Cye Personal Robot

The Cye robot is designed to be used in a home, office, or research setting. As sold, it is available with a Hoover mini-upright vacuum cleaner attachment and is capable of vacuuming autonomously. A wagon is also available, which allows Cye to carry things such as coffee, dishes, documents, etc. Since it is capable of pulling over 30lb, Cye can transport relatively large items.

Cye is a two-wheeled, differential drive robot with non-holonomic constraints, and is shown in Figure 1. There is no caster wheel; instead, Cye is balanced on its two main wheels, by placing the CG below the wheel axel, to reduce drag and slippage. The reason for this is Cye's only means of navigation is ded-reckoning. Each wheel has an encoder on it, and the on-board micro controller uses this encoder to determine odometry and thereby maintain the robot's position in the world -- namely, (x, y, θ).

Ded-reckoning's main advantage is that it is simple to compute, and does not rely on expensive sensors or off-board beacons. The disadvantage is that position estimation error accumulates over time. In Cye's case, for instance, after travelling 50 feet, the positional error can be as large as six inches. This is because a small

Figure 1: The Cye Personal Robot

error in heading can cause a large error in position. For example, after travelling fifty feet, a one degree heading error leads to a positional error of about 10". The error growth rate depends on floor type and path geometry. Path geometry plays a role because rotations cause heading error to increase more rapidly than straight-line travel.

To maintain tracking accuracy, Cye makes use of calibration surfaces, called "checkpoints". A checkpoint is a known surface, usually a wall or a corner, which the robot bumps into to re-calibrate itself. This assumes that the tracking error hasn't grown too large. That is, Cye has to be able to get relatively close to a checkpoint, and know that it is close, so that it can bump up against the surface and reset its error.

Cye has no active sensors. It detects obstacles by monitoring wheel velocity. When Cye bumps into an obstacle, its wheel velocity drops, and this is sensed by the on-board micro controller. While this means that Cye has to physically bump into something to detect it, it resulted in a cost-savings and a simpler design. As Cye was designed to be sold at scales of thousands of units per year, this was an important consideration.

As mentioned above, Cye only has a micro controller on-board. All higher level functions are carried out by a remote PC. Cye and the remote PC communicate via a 19.2Kb radio link. Cye uses this link to send packets to the PC, containing position updates and the state of the obstacle detection system, and to receive via-point commands from the PC.

When setting Cye up in a new environment, the user has to create a map, which is shown in Figure 2. This map is created by dragging an icon of the robot around on the screen using a mouse. The physical robot tracks this movement, and the operator can teach it which areas are free and which are occupied. An obstacle can be mapped by driving Cye into it, which results in the creation of an obstacle area on the map. Basic drawing tools are included, allowing the user to specify areas as free, occupied, or uncertain.

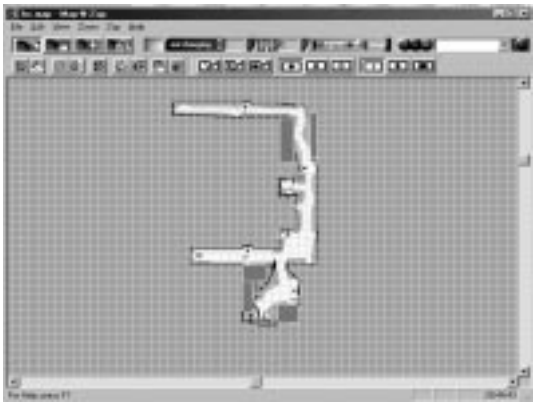

Figure 2: Map-N-Zap Screenshot. White areas are freespace, grey areas are uncertain, and black areas are obstacles.

This screen shot helps to introduce Cye's internal representation of the world. Cye uses an evidence-grid approach [4], in which the world is discredited into small grid cells, and the contents of each grid cell are a measure of the occupancy. At the extremes, a cell can have a '0', which means that it is free space with 100% certainty, or '255', which means that the space is occupied, with 100% certainty. Values in-between reflect the certainty of the presence of an obstacle. In Figure 2, white areas represent free space, grey areas represent unexplored (or uncertain) space, and black areas represent obstacles.

3. Path Planner

The underlying structure of the planner incorporates a grid based potential field. We assume that the robot is a holonomic platform. Although Cye is non-holonomic, there is low-level motion control on-board which transforms an arbitrary path to a set of wheel velocities approximating that path. This field is constructed in two stages. In the first stage, each cell contains a value denoting the traversability of that point. To do this, a field describing the distance of any given world point to the nearest obstacle is created using a grassfire approach. In this field, unexplored areas are marked with a predetermined "pseudo-distance". This distance value biases paths around unexplored areas.

In the second stage, a potential field is created in which each world point is the distance to the goal, non-linearly weighted by the corresponding value in the traversability grid. This field is generated using a wave-

front expansion from the robot's goal point. This field is used to create a path from start to goal which minimizes the path length plus the cost of traversability.

3.1. Traversability Field

The traversability field is a grid in which each cell corresponds to a point in the map, based on its coordinates. Each cell has a related value which is a measure of traversability. It is important to note that here, two factors influence traversability:

- The distance to the nearest obstacle
- Terrain type

Therefore, an area can have low traversability because it is unexplored, or also because it is close to an obstacle.

A Grassfire transform is used to create the traversability field. This is one method to create a voronoi diagram, for cell-based path planning methods [1]. These planners maximize distance from obstacles, but do not necessarily optimize for path length, sacrificing path length for safety.

The transform operates as follows. First, the grid is initialized with '1's in all grid locations which correspond to map locations which are occupied. In this formulation, lower values indicate lower values of traversability. A '1' means that the cell is completely occupied, and therefore is not traversable. Higher values mean that it is possible to traverse the area.

Since the map contains a priori information about the traversability of non-obstacle areas, the field is initialized with this information as well. In this case, it means pre-assigning values in the traversability grid to each cell which corresponds to a map location which is unexplored. Although the map contains certainty information, this information is not used when seeding the traversability grid. Instead, thresholds are used to mark areas as "unexplored". These areas are seeded with a value, adding a bias to unexplored areas, which, as we will show, influences the paths which are generated.

After the initial seeding is done, the field is grown using the algorithm shown in Figure 3.

The first two plots in Figure 4 show this process. The top plot shows a sample world, consisting of a start point, an end point, one obstacle, and one unexplored area. The middle plot shows the results of a grassfire transform applied to the sample world. The italicized values show the initial seeding. Note that in a standard grassfire transform, the value of each cell denotes the distance to the nearest obstacle. In this formulation, however, the contents of each cell is a measure of traversability, due to the inclusion of the unexplored terrain. The bottom plot shows the potential field which is generated using the traversability field. The method for generating the potential field is described next.

```
Set growth_cntr = 1
Iterate over all cells, c(i, j)
 progress = 0
 if c(i, j) == growth_cntr
  progress = 1
  Iterate over the 8-neighbors of c(i, j)
   n(k, l) = neighbor
   n(k, l) = MIN(growth_cntr+1, n(k, l))
 if progress == 0 then exit
 Increment growth_cntr
end
```

Figure 3: Traversability Grid Algorithm

3.2. Potential Field

The final potential field is created using a combination of a wavefront transform [REF] and the traversability grid. The transform starts at the goal, and progresses until the start point is encountered. In the standard wavefront algorithm, the cell representing the goal point is seeded with a '1', and all the unoccupied neighbors (we use the 4-neighbors in this case) are seeded with '2', and all the unoccupied neighbors of '2' are seeded with '3', etc., until the start point is reached. At this point, a potential field is created. The planner then starts at the start point, and takes steps in the direction of decreasing cell value, until the goal is reached

This method generates a path that is optimal in length, but tends to hug the sides of obstacles. This problem can be alleviated by "growing" the obstacles by a certain amount, guaranteeing a minimum distance. However, this is unsatisfying, as occasionally, to guarantee completion, it may be necessary to take a risk and get very close to an obstacle.

Similarly, the basic wavefront approach cannot handle varying terrain types or any uncertainty in the state of the world. One method for getting around this is to tag all uncertain or unexplored areas as obstacles, and avoid them entirely. Again, this is unsatisfying, as it is worthwhile to explore new areas, if doing so could result in a significant savings in path cost.

Using the traversability grid allows the planner to use information on how navigable a region is while generating the path. Unlike a standard wavefront algorithm, where the optimality criteria is path length, the optimality criteria of our planner is a combination of path length and traversability. The algorithm for calculating the potential field is shown in Figure 5.

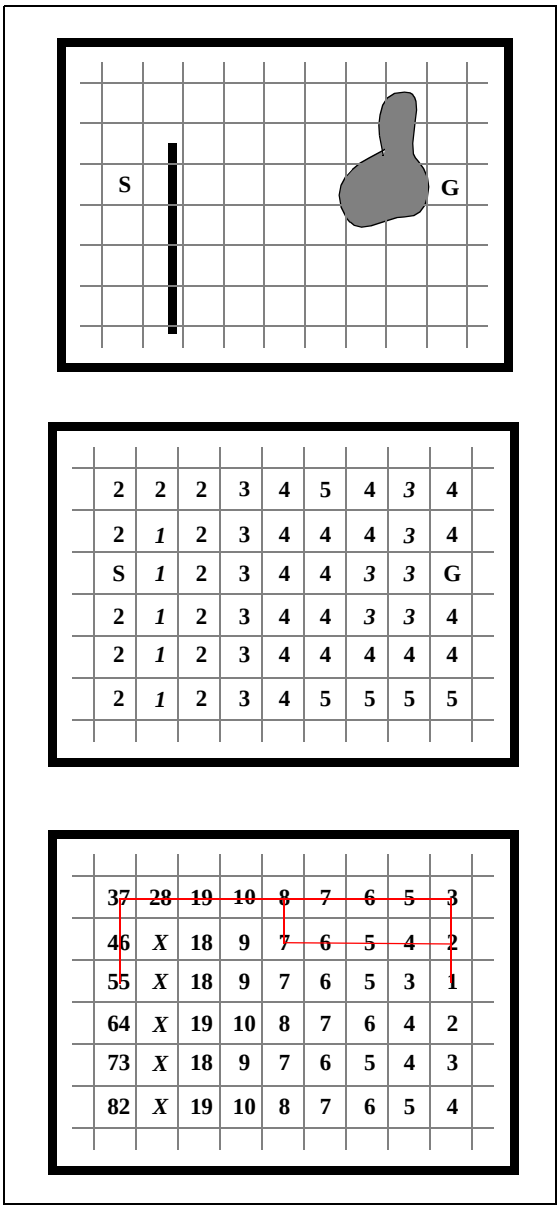

Figure 4: The Cye Path Planning Algorithm. The top image is a sample world with an obstacle on the left and an unexplored area on the right. The middle grid shows the traversability grid, and the bottom grid shows the final potential field.

Note the use of the "MinTraversability" parameter. This states that if the traversability of a cell is below a certain value, then we bias the corresponding potential field value by a non-linear factor of the traversability. Practically, this has the effect of setting a minimum traversability (or distance from obstacle) requirement. When going down hallways, this causes the robot to run down the center of the hall. It takes a significant savings in path cost to overcome the bias of this factor.

```
Set growth_cntr = 1
Iterate over all cells, c(i, j)
 progress = 0
 if c(i, j) == growth_cntr
  progress = 1
  Iterate over the 4-neighbors of c(i, j)
   n(k, l) = neighbor
   T = traversability(k, l)
   if T < MinTraversability
     n(k, l) = growth_cntr +
       (MinTraversability - T)^3 + 1
   else
     n(k, l) = growth_cntr + 1
 if progress == 0 then exit
end
```

Figure 5: Potential Field Algorithm.

The non-linearity is important. It can be shown that if a linear scaling is used, then the path only asymptotically converges to the appropriate distance from obstacles. I.e., when entering a hallway, the robot will cut the corner slightly to save path length, but then will only asymptotically approach the center of the hallway. Using a non-linear scaling guarantees that the robot will quickly converge to the center of the hall. The planner is fairly insensitive to the non-linear factor. Extreme values only slightly affect how far the robot travels before centering itself.

The result of this algorithm is shown in the bottom figure of Figure 4. To create a path from this grid, we begin at the start point, and walk "down" the field, moving in the direction which results in the largest decrease in cell number. This path is marked on the bottom figure. At the point where the field value is '8', there are two possible decisions -- either right or down -- resulting in two different, but equivalent paths. Both paths have length 12, and both paths pass through one unexplored cell. An alternative would be to detour around the unexplored region entirely, which is what a standard wavefront algorithm would do. This would result in a longer path length of 16. The optimality criterion we use trades off traversability for path length, and therefore was willing to enter an unexplored area because it led to a shorter path.

3.3. Sub-goal Planning

Since Cye navigates using ded-reckoning, having a working path planner is not enough to guarantee that it can always reach its goal. As mentioned in Section 2, Cye uses checkpoints, or known calibration surfaces, to register itself with its map. A full path planning system

should make use of these check points, so that it can traverse longer distances, while keeping positioning error low.

The algorithm the planner uses to incorporate check points is fairly straightforward, and basically involves searching over the available checkpoints, and generating intermediate paths to them. The first step is to generate a path from the start to the goal. After this, a wavefront expansion is done from the entire path, keeping track of all checkpoints that are encountered. This gives a measure of the distance from the path to each checkpoint. We filter on this distance, to prevent using checkpoints which are distant from the original path, as distant checkpoints provide a diminishing return on overall positional accuracy.

Next, the checkpoint closest to the start of the path is found, and a new path is generated from Cye's current location to the checkpoint. Following this, a new path is generated from the checkpoint to the end goal. Then, the next checkpoint is located, and planned to. If the nearest checkpoint is actually the end goal, then we are done, and Cye moves to the final goal. If it is not, we repeat the process until the final goal is found. Although this algorithm involves additional plan generation, the overall computation time of the planner is low, so this does not add much overhead, but it greatly enhances the performance.

4. Results

This path planner has been available for about five months, and is an integral part of Cye's operation. There are about 200 installations of the planner, based on the number of people who have the latest version of Map-N-Zap. Assuming a mean usage time of 2.5 months, and a weekly usage of 300' per user (probably an underestimate), Cye (and by extension, the path planner) has autonomously travelled over 110 miles.

Since the planner calculates a global field, it is not susceptible to local minima. Figure 6 shows an example of this. The figure shows an obstacle configuration which traditionally causes problems for potential field based planners which operate locally. The planner properly avoids the "U"-shaped obstacle, and routes around it instead.

Figure 7 shows the advantage of having a planner which accounts for terrain type. The top figure shows an example of this. White areas are explored, and grey areas are unexplored. In the top case, the cost of going around the unexplored area is less than the cost of going through it. However, as the bottom figure shows, if the detour is too large, then cutting through the unexplored area becomes the better option.

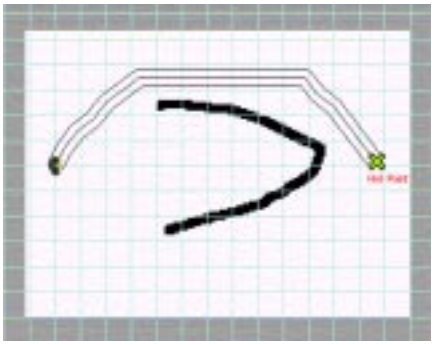

Figure 6: Avoiding Local Minima

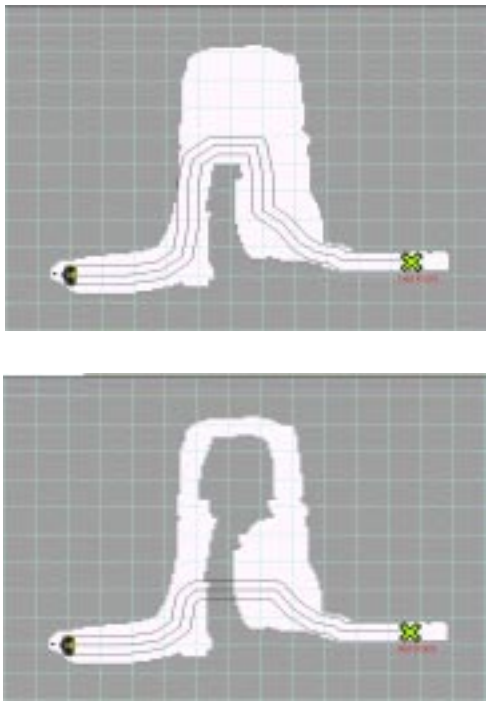

Figure 7: Cye opportunistically using unexplored areas to shorten path length.

While the planner operates well in "simulated" mode as the above results show, the real test is when it is coupled with the physical robot. To put the planner through a significant real world test, we performed the following experiment.

Figure 8 shows a map representing an area including an office, along with the adjoining corridors. The traversable area of this map is about 450 sq. ft. There is a fair amount of traffic through this area.

We randomly generated 25 goal points spaced on the free areas of the map. The only constraints were that no two goal points lie within 2 feet of each other, and no goal point can be closer than 6" to an obstacle. After that, a random ordering of the goal points was gener-

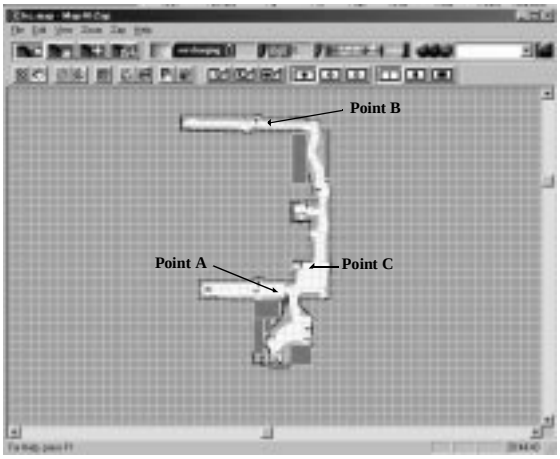

Figure 8: Real-World Test Area

ated, and Cye navigated through this series of goals. We did four runs. Each run started at the home base, went through all 25 goal points, and then returned to the home base, resulting in 26 paths per run. The total number of paths generated was 104, over the four runs. Note that this number indicates paths from start to goal - it does *not* include paths generated during checkins. Therefore, the actual number of paths generated is much higher. The first two runs did not have any goal points placed in the office area. The last two runs included goal points in the office area, which meant that Cye had to accurately navigate a narrow doorway a number of times. Table 1 shows the results.

Run	Total Length (ft.)	Mean Length (ft.)	Min. Length (ft.)	Max Length (ft.)	Num. Failure / (%)
1	840	32	3	79	1 / 4%
2	833	32	3	80	0 / 0%
3	976	38	4	105	1 / 4%
4	1098	42	4	92	1 / 4%
Total	3748	36	3	105	3 / 3%

Table 1: Real-World Results

Over the four runs, Cye ran approximately 0.68 miles (1.1km) autonomously. There were six collisions which did not require human intervention. In these cases, Cye was able to plan around the unexpected obstacle, and continue. There were three failures which required human intervention. In these cases, the intervention was limited to repositioning Cye on the map manually. We never had to physically move Cye. The locations of the failures are marked on the map in Figure 8. The three failures were:

- An individual repeatedly blocked Cye's path in a hallway, until it was unable to find a free path through the hallway. (Point A)
- An unknown failure happened here. (Point B)
- Cye's ded-reckoning error accumulated to the point where it couldn't find the checkpoint it was expecting, and so because lost (Point C)

Overall, these results are encouraging. They show that it is possible to reliably navigate in a real world environment. This is a necessary requirement for personal robots.

5. Conclusion

We have developed a path planner which uses a novel optimization criterion. This criterion allows a robot to find short paths, while maintaining adequate distance from obstacles, along with handling unexplored areas or areas of varying terrain in a principled manner.

This planner has been used for months with over 200 real world installations. Our experiments show that the planner is robust, handles dynamic obstacles, and can deal with an uncertain map.

6. References

[1] J.F. Canny and M.C. Lin, "An Opportunistic Global Path Planner," *Algorithmica*, 10:102-120, 1993

[2] O. Khatib, "Real-time Obstacle Avoidance for Manipulators and Mobile Robots," *International Journal of Robotics Research*, 5(1):90-99, 1986

[3] D. Koditscheck, "Exact Robot Navigation by Means of Potential Functions: Some Topological Considerations," *Proceedings of the IEEE International Conference on Robotics and Automation*, May, 1987

[4] M.C. Martin and H. Moravec, "Robot Evidence Grids," CMU RI Tech Report CMU-RI-TR-96-06, March, 1996

[5] J.S.B. Mitchel, D.W. Payton and D.M. Keirsey, "Planning and Reasoning for Autonomous Vehicle Control," *International Journal of Intelligent Systems*, 11:129-198

[6] N.C. Rowe and R.S. Alexander, "Finding Optimal-Path Maps for Path Planning Across Weighted Regions," *International Journal of Robotics Research*, 19(2), 83-95, 2000

[7] C. Thorpe, "Path Relaxation: Path Planning for a Mobile Robot," *Proceedings of the National Conference on Artificial Intelligence, aaai-84*, August, 1984

[8] J.S. Zelek, "Dynamic Issues for Mobile Robot Real-Time Discovery and Path Planning," *Proceedings of the IEEE CIRA,"* November, 1999.