Advanced Robot Control

Midterm Exam

Andrew Ferguson

In Chapters 2 through 7, Choset has presented a number of different approaches to path planning. Explain in your own words the progression from Chapter 2 to 7. Include brief (synopsis) explanations of each approach (i.e., the main topic of each chapter).

Chapter 2 covers the two bug algorithms, Bug 1 and Bug 2. The Bug 1 algorithm is perhaps one of the most basic of navigating algorithms. The robot begins at the start and proceeds toward the goal until an obstacle is encountered. Once an obstacle is encountered, the robot will completely circumnavigate the obstacle before proceeding toward from the point on the perimeter that has the shortest distance to the goal.

The Bug 2 algorithm is very similar to the Bug 1 algorithm. The primary difference from Bug 1 is that an imaginary line, the M-line, is draw between the start and goal. When the robot encounters an obstacle, it will circumnavigate the obstacle until it reaches the M-line. Once it reaches the M-line, it will start moving toward the goal again; it does not completely circumnavigate the obstacle.

The primary benefits of Bug 1 and Bug 2 is that very little information is needed: a starting point, goal, and a series of touch sensors to determine when the robot runs into an obstacle.

A more advanced version of Bug 2 is the Tangent Bug. Tangent Bug utilizes a sensor of some range from zero to infinity to detect obstacles. When an obstacle is detected, the robot will start moving around the obstacle. The robot will continue its motion-to-goal routine as soon as it has cleared the obstacle.

In order to describe more complicated path planners, we need to be able to specify the position of the robot and the space it occupies. This is known as the configuration space and is the primary topic of chapter two. We used the new idea of configuration space to develop a set of equations that allow us describe the position of multi-joint planar arms. We extended the equations we used to define the movement of planar arms to define points in both R^2 and R^3. These equations allow us to define the position and orientation of a single point, as well as one point relative to another.

With our new ability to describe robots in a workspace, we can now begin working with more complicate path planning techniques. Path planning is quite different from the previous navigation algorithms we discussed, such as Bug 1, because path planning requires foreknowledge of the obstacles before the robot begins movement. Chapter four starts with a brief discussion on gradients. Gradients are a well studied mathematical function in undergraduate engineering university, which makes them a great introduction for more complex path planning methods. Gradients work by changing the repulsive potential as a robot gets closer to an obstacle. The start point is given a medium gradient potential, the end point is given a low gradient potential, and the obstacles are given high gradient potentials. All the robot needs to do is “roll down the hill.”

The Brushfire Algorithm is a discrete version of the aforementioned gradient algorithm and involves the use of a grid to determine the potential of cells. A variation on the Brushfire Algorithm is the Wave-Front Planner.

The Wave-Front Planner uses a grid, just like the Brushfire Algorithm, and assigns a “1” to each cell that has an obstacle (or part of an obstacle). The start point is labeled “2” and the “wave” propagates from that point. Each adjacent cell, if empty, is given an incrementally higher number until all the cells have a number. If the goal has a number in its cell, the goal is reachable in that many moves minus one. Adjacency can be computed using either four-point connectivity (where only the north, south, east, and west cells are considered adjacent) or eight-point connectivity (where north, south, east, west, and the diagonal cells are considered adjacent).

Sphere space and Star space are essentially extensions of the generalized potential path planning method with one critical difference: they only have one local minimum. Having one local minimum is a desirable quality as it greatly aids in path planning and avoiding dead ends.

Chapter five introduces roadmaps, which is effectively a way to break up the workspace into distinct stages or paths. The Visibility Graph plots straight line paths between the start, goal, and all the vertices of the all the objects in the workspace. The paths may cross over each other; however, they may not intersect an object. Once constructed, the visibility graph can be searched for the shortest path from the start to the goal.

The Generalized Voronoi Diagram extends the idea of the distinct paths by drawing equidistant “roads” between obstacles and the boundaries of the workspace. A similar technique to the GVD is the Silhouette Method, which works by determining the critical points of obstacles within the workspace. A tangent line extends from each critical point, and around the workspace edge. The path from the start to the goal “snaps” to these paths.

Trapezoidal Decomposition, which is the start of chapter six, extends the concept of the Silhouette Method. However, the obstacles (and workspace) must be represented as trapezoids. A vertical extension marks each vertex, separating the workspace into cells. The cells are mapped from the start to goal using an adjacency graph. The path planner plans a path by connecting the midpoints of the vertical extensions. Morse Cell and Boustrophedon Decomposition follow the same concept as Trapezoidal Decomposition. However, the goal in Morse Cell and Boustrophedon is path coverage rather than path planning.

Path coverage, which seeks to optimize the coverage of the workspace, leads into visibility-based decompositions for pursuit and evasion, which deals with how a pursuer attempts to capture prey and how the prey attempts to evade the predator. Specifically discussing how to clear rooms to make them “uncontaminated” and how to determine the number of predators that are required to check a system based on the number of edges in the workspace.

As this point, the path planning algorithms are becoming fairly complex and computationally intensive. So far, our examples have been using small workspaces with only a handful of obstacles. As the workspace size and number of obstacles increase, the time required to find a solution also increases, usually exponentially. Sampling-based Algorithms work by using a best-guess and check method. A path is calculated and then checked to see if it collides with any obstacles. Specific types of planners include Probabilistic Roadmaps (PRM), Randomized Path Planner (RPP), Expanisve-Spaes Trees (EST), Rapidly-

exploring Random Trees (RRT), and Single-query, Bi-directional, Lazy collision-checking (SBL). In all examples of sample-based path planning, the chances of finding a successfully path approach zero as the number of samples increase.

Pick your favorite path planning approach and explain why you prefer it. Give an example of how it works. Identify its strengths and weaknesses, compared to other methods. Describe how you might apply it.

My favorite path planning approach is the visibility graph. I enjoy the visibility graph because it is one the primary methods I use to when performing my own human version of "path planning." Last summer, I worked in a cubicle office space. There were two doors leading into the office space and the door to the restroom was more or less situated between the two doors. I was constantly trying to figure out which door required the least amount of steps. Although I didn't know it at the time, I was basically creating a visibility graph and attempting to determine the shortest path, all in my head.

The Visibility Graph works by plotting straight line paths between the start, goal, and all the vertices of the all the objects in the workspace. The paths may cross over each other; however they may not intersect an object. There are several methods to actually calculate the visibility graph; however the most common method is the rotational plane sweep algorithm. Once constructed, the visibility graph can be searched for the shortest path from the start to the goal. Searching for a path usually requires weighting each segment of the line. Typically, the weight would just be its distance. However, other factors such as quality of terrain or amount of energy needed could also be taken into account.

One of the most obvious weaknesses of the visibility graph is that its complexity greatly increases as the number of objects increases. Additionally, many of the edges in the visibility graph are unneeded, such as when a vertex occurs in a convex area. There are methods to remove these additional edges; however, they will require additional computation efforts. If a workspace is expected to have many objects with convex vertices, it would probably be more efficient to use trapezoidal decomposition.

One of the benefits of the visibility graph, especially when compared to trapezoidal decomposition, is that the path planner does not care about the boundaries of the workspace. This could be especially helpful in situations where the edges of the workspace are complex. And let's face it, looking at the visibility graph is pretty cool in and of itself.

Reflecting on the material covered in Chapters 2 through 7, what would you say is missing from the path planning discussed so far? How effective are the techniques discussed?

I think the biggest method missing is SLAM (simultaneous localization and mapping). So far, we have been able to navigate in a workspace that we know either nothing about or everything about. However, we do not yet have the tools to learn about our environment and make better decisions based on that new data.

I would also enjoy discussing path planning in a fully three-dimensional system, such as might be used for a spacecraft or satellite.

The techniques that we have discussed so far seem to be very effective. Each method has its strengths and weaknesses; however, taken as a whole, the path planners seem like they can handle any reasonably path planning problem.

One thing that has not been discussed is the efficiency of path planners. Ten years ago, this probably would have been a problem. However, current computers (even desktop computers) are powerful enough to quickly compute a path (or lack of path) for any reasonable workspace. Though, what defines a "reasonable workspace?" As path planning is applied to a more varied array of situations, such as biological devices, one can anticipate that the complexities of the paths will also increase. As long as computational power continues to increase, this won't be a problem. However, many industry analysts believe that we are quickly reaching the maximum transistor density for CPUs. Are the currently path planning techniques efficient enough, or do we need to investigate more efficient ways to determine paths (perhaps parallel processing)?

> ***Go out on the web and find a robot system, real or vitual, that uses one of these techniques. Describe the robot, the implementation of the technique, and both its benefits and drawbacks (i.e., what's good about the system and what needs work).***

The Cye Personal Robot is a commercially available, two-wheeled, autonomous robot. Cye's center of gravity is placed below the wheel axis, which allows it to be inherently stable and does not require the use of accelerometers or gyroscopes to maintain balance. In fact, the only sensors that Cye uses are wheel encoders. By using data gathered from the encoders, Cye can determine how far it's traveled and if it's hit an obstacle. Cye is powered by a 16-bit 16Mhz High Speed I/O Microprocessor (Long n.d.), similar to that of a HC9S12. Additionally, Cye has a radio uplink to a remote PC to perform higher level functions, such as allowing a user to input a map.

Cye uses a combination of techniques to path plan, although the "underlying structure of the planner incorporates a grid based potential field." (Batavia and Illah 2000) Cye also uses wave-front path planning to determine a path that minimizes path length (while also factoring in traversability).

To begin, the start and goal are placed on the traversability field, which is a grid. Rather than mark a cell as obstructed or not, a traversability field allows each cell to have a range of values that go from completely occupied (and therefore completely untraversable) to unexplored (which could be traversable or not) to known free space (which is guaranteed to be traversable).

```
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 1: Traversability Grid Algorithm

Known obstacles are inserted and the field is grown using a Grassfire transform (see Figure 1), a type of Voronoi Diagram that is used for cell-based path planning methods. One thing that is important to note in this version of the grassfire transform is that the value of each cell indicates the level of *traversability*, not *distance* to the nearest obstacle (as is the case in a standard grassfire transform).

From the traversability field, a potential field is generated using a wave-front path planner with four-point connectivity. As Batavia and Illah write:

> 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.

As it turns out, using the traversability field as the input to the wave-front path planner helps to overcome the aforementioned issues and, as mentioned before, a path is created based on traversability, rather than just shortest distance. As shown in Figure 2, if a distance is long enough, a route can be plotted *through* an unknown territory (right image) rather than *around* it (left image).

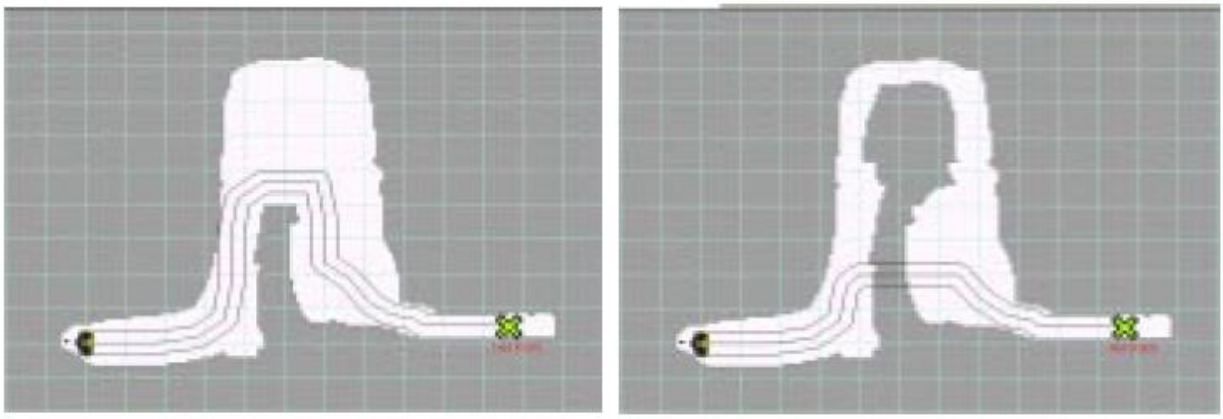

Figure 2: Cye opportunistically using unexplored areas to shorten path length

Finally, the global path planning algorithm determines local check points. Cye will navigate to these known locations to help eliminate positioning error. As Batavia and Illah write:

> [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.

This three stage path planning method has proven to be very robust. In analyzing the path planner, Batavia and Illah only had three failures that required human intervention over a distance of 1100 meters. Additionally, Batavia and Illah reported that, "[there] were six collisions which did not require human intervention. In these cases, Cye was able to plan around the unexpected obstacle, and continue."

It's apparent that the methods Cye uses to path plan are mostly complete. The biggest issue is probably the lack of active sensors. Currently, the only way Cye knows about obstacles is through human input (via a mapping tool on the computer) or bumping into them. Cye would greatly benefit from either IR or sonic distance sensors that could be used to determine the traversability of unknown regions before actually attempting to move through the region. Another area of issue is the reliability of the shaft encoders. Currently, Cye needs to dead-reckon from checkpoint to checkpoint. Increased resolution of the shaft encoders, as well as information about the power consumption of the motors, could be used to decrease the dead-reckoning error and increase the distance between checkpoints.

Works Cited

Batavia, Parag H., and Nourbakhsh Illah. "Path Planning for the Cye Personal Robot." 2000. http://www.cs.cmu.edu/~illah/PAPERS/cye.pdf (accessed March 24, 2009).

Long, Dan. *Hereee are Long++'s inside peek at Cye Pictures...* http://sunsmoke.org/~ldang/long/bot/cye/index.html (accessed March 25, 2009).