Rapidly-Expanding Random Trees and their Application to Botball
Wesley Myers
wmyers@andrew.cmu.edu

Rapidly-Expanding Random Trees and their Application to Botball

Introduction

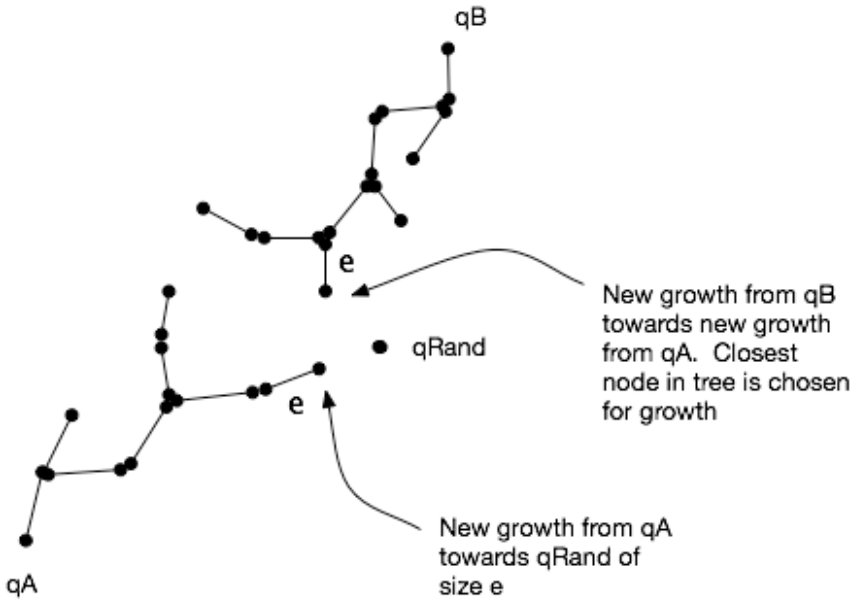
This paper discusses our exploration into the development of path planning in Botball. Here we tackle this problem using Rapidly-Expanding Random Trees (RRTs) [1]. With the robot being able to keep track of its position, path planning can be quite useful to move around the board.

Motivation

In Botball there are times in which the robot needs to get from one point to another on the game board. However, sometimes there is not always a good way to know how to do this navigation unless it is preplanned. In robotics navigation there are many ways to do path planning. A robot could use simple waypoint navigation to move around, or use a Voronoi path navigation system for example. Each method has its own tradeoffs and benefits. We decided to use Rapidly-Expanding Random Trees (RRTs) to solve the robot navigation problem. The benefit of RRTs is that a robot can quickly generate a safe path through the workspace of the game board. This is best used in conjunction with position tracking code.

How a RRT Works

A RRT works as follows. Given an initial configuration and a final configuration, the algorithm finds a path that guides the robot between the two configurations. Nodes in the tree are represented by a configuration and a parent node (think of this a tree that is represented by a Linked List with multiple children). The initial configuration has no parent since it is the first node, the next node's parent would be the initial node, etc. Let us call the initial configuration qA and the final configuration qB. First a random configuration is chosen somewhere in the configuration space. If that configuration does not collide with any obstacles, a "branch" of size epsilon is grown towards the new configuration. The other tree then finds the node closest to the newly grown configuration of the first tree and grows a new node towards the first tree's new node by size epsilon (represented by e in the below diagram). The two roles then swap for the next iteration. The trees keep on growing towards each other until they "touch." Once this occurs, a path has been found. The diagram below illustrates how this works.

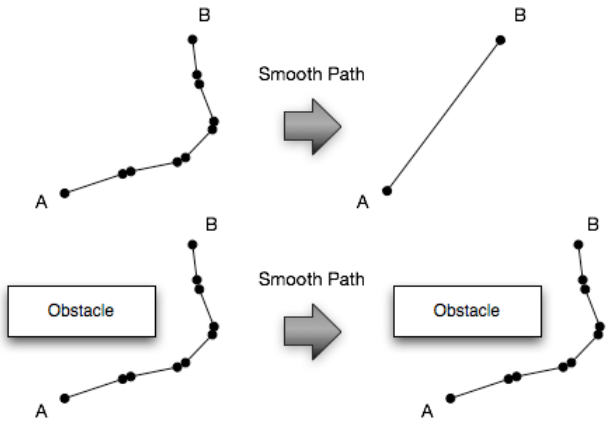


Path Extraction

At this point, the RRT has declared that a path exists; so next a path must be extracted. Extracting a path can best be done by backtracking through the tree's nodes. Backtracking stops when the node doesn't have a parent (initial or final configuration). These two separate paths are then concatenated and the result is the path from qA to qB. See Diagrams section for illustrations.

Path Smoothing

At this point there is a path, however the path is not very smooth. What is desired is a path with as few nodes as possible. There are many algorithms that can achieve this. One method is to choose two random nodes on the path. If there are no collisions between the two configurations and any obstacles, then delete all the nodes between the two nodes. Repeat this process a number of times until satisfied. The figure to the right illustrates this idea. If you have nodes A and B, and there is no collision between the nodes, then you can smooth the bath by removing the nodes in the middle. If there is an obstacle, then the path cannot be smoothed with those two nodes as illustrated in the bottom of the diagram.



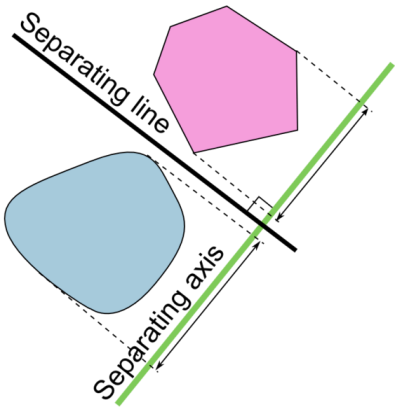
Collision Detection

In order to properly execute an RRT, some form of collision detection must be present. This can be done in multiple ways. Here we will discuss two different ways to do this.

The first step is to have a representation of the world. The world is the robot and its surroundings. The simplest way to represent the robot is by the form of a square or a rectangle. "Padding" to the exterior of the robot ensures that the robot does not come close to walls and obstacles. The RRT will search for a path, and sometimes that path will go along the edge of an obstacle. Since navigation is never perfect, the more space between an object and the robot, the better. Next, a representation of the world bounds is needed. This can best be done using points to form a closed polygon. This yields the possibility of now forming lines to represent the robot's world. Also represent any objects within the world as a polygon (points). These polygons are placed into an array for easy access.

There are several options as to how detect a collision between a robot and an object. The simplest would be to see if the rectangle that the robot forms intersects with any of the lines from the polygons that represent objects.

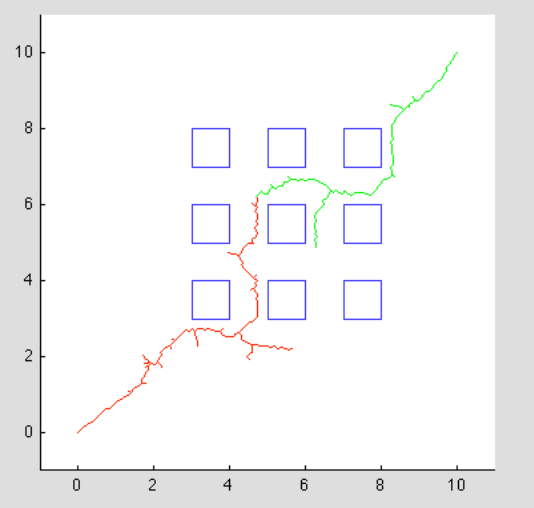
The other, more complicated, method is the Separating Axis Theorem. The theorem states that for objects lying in a plane, given two convex shapes, there exists a line onto which their projections will be separate if and only if they are not intersecting. The separating axis is the line for which the objects have disjoint projections. The separating line is perpendicular to the separating axis and is in between the two objects. This line cannot be drawn if the objects overlap. The figure to the right illustrates this concept. This algorithm is much more complicated, but is much more reliable as compared to the basic one previously discussed.



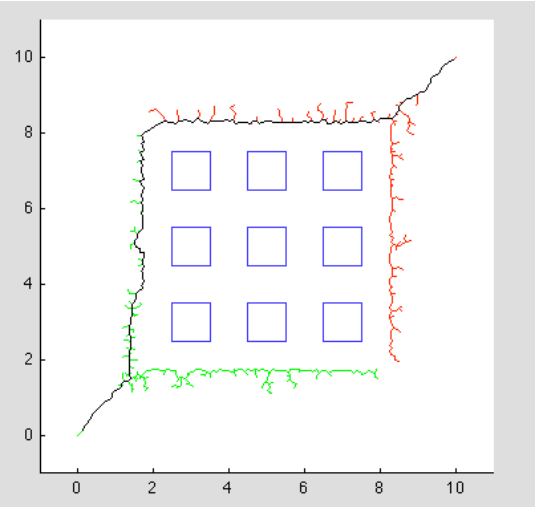
Diagrams

In this section, we discuss a few results that were generated in matlab for initial testing.

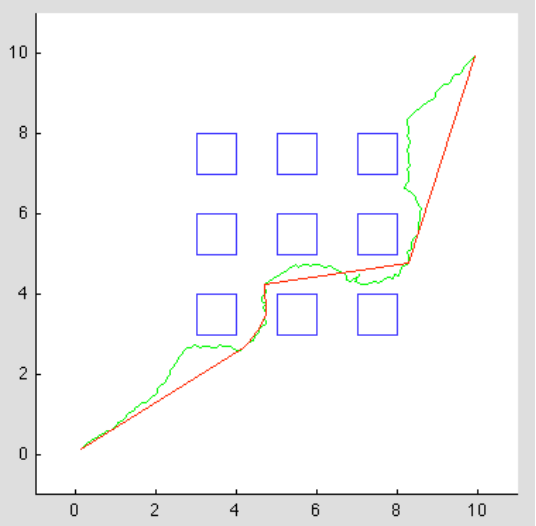
This first diagram shows a path starting at configuration (0, 0, pi/4) and ending at (10, 10, pi/4). What is shown here are the two trees navigating through a complicated workspace. The robot in this case is small enough to navigate between the blocks. Here one can see that the trees seem to weave their way through the world until they touch.



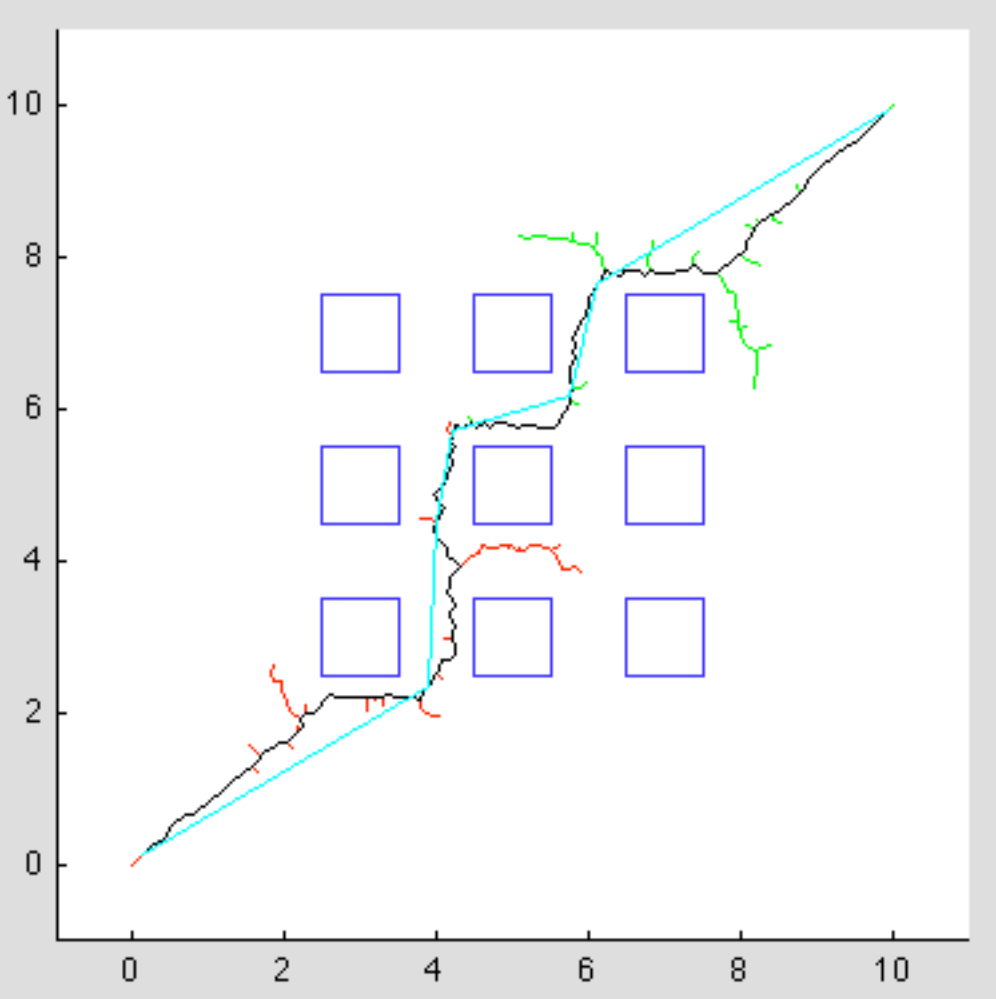
This diagram shows what happens when the robot is simply too large to fit between the obstacles. What is important to note here is that two paths could have occurred. One can see the obvious path that was created. The other one that could have been created is where the green and red branches came close to each other in the bottom right of the diagram. Either of these two paths could have occurred. The randomness of this algorithm yields a random path.



This diagram illustrates path smoothing. The tree gives a very jagged path from the starting configuration to the ending configuration. By smoothing the path, the robot can rotate in place, drive, rotate, drive, etc. The green line is the extracted path and the red line is the smoothed path.



This diagram illustrates all the components. The green and red lines are the two separate trees grown from the starting and ending configurations. The black line is the extracted path from the two trees. The cyan line is the smoothed path.



Results

We were able to achieve path planning with the robot in the world. We decided not to implement this in the KIPR Open. However this could be of great use in future years. This can give the robot more autonomy when it comes to interacting with the world. In the KIPR Open, robots are able to communicate with each other, so thus this can give teams a great advantage. If they are able to maintain an awareness of the team's robot's positions on the table, then this algorithm can be used by the robots to avoid each other.

References

[1] Kuffner, J. J. Jr., and LaValle, S. M. 2000. RRT-Connect: An efficient approach to single-query path planning. *Proc. IEEE International Conference on Robotics and Automation (ICRA-2000).*