Caelan Garrett
6.S078 Assignment 3
10/16/13

Description

I created separate classes for the c-space, RRT, and robot. The c-space class has methods for sampling the c-space for the robot, using linear interpolation to sample paths between two configurations, testing valid configurations, and moving in rotational configurations with wrap-around.

The RRT class maintains the search tree and computes nearest neighbors. Unfortunately, I was not able to install scipy.spatial because of some Window's errors (I know…) involving a conflicting older version. So the RRT class exhaustively searches the tree to find the nearest neighbor. The RRT planning algorithm selects points on the vector to the sampled configuration and adds them to the tree until it hits an obstacle. The algorithm samples 10 evenly spaced slices per vector and can be easily toggled. For Bi-directional RRT, it always tries to connect the last point added to the other tree (even if the sampled configuration is not reachable) which seems to work well from my experiments. I also alternate which tree to sample based on their sizes to ensure even growth as seen in the Howie Choset slides. I run the RRT algorithms for 1000 iterations. For the size of my experimental worlds if the algorithm did not find a path in 1000 iterations, it was very likely one existed. It often found solutions significantly faster, especially when using the Bi-directional RRT.

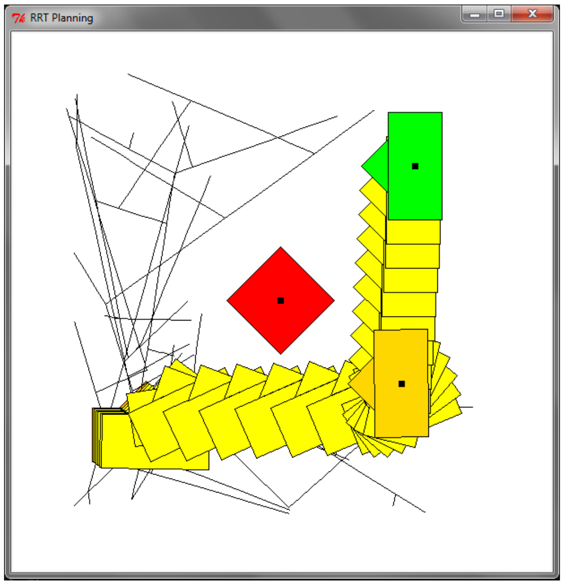
I created separate classes for the chain robot and polygonal robot. Each class has two primary methods. The first generates the polygons for the robot under that configuration for collision tests. The second is a distance metric using the robot's geometry. By separating the c-space code from the robot class, I can easily define a robot with different degrees of freedom later by creating a new class with just these two methods. The polygonal robot uses a metric that is the sum of the Euclidean distance with the rotation multiplied by the robot radius. The chain robot uses a different metric based off of the max radial movement for a point on the chains. The chain robot is defined to have a configuration of (0.0, …, 0.0) when it is entirely facing right (0 radians).

I use direct, random shortcutting to smooth RRT paths. It samples random steps on the path and tries to create a shortcut by a direct movement. It does this 100 times so it is effective at smoothing. The smoothed paths seemed comparable to paths created by visibility graphs in that they were reasonably efficient along similar trajectories, but may have had some unnecessary translations and rotations. Additionally, Bi-directional paths were overall worse, before and after smoothing. But this was a small price to pay for the speed-up.

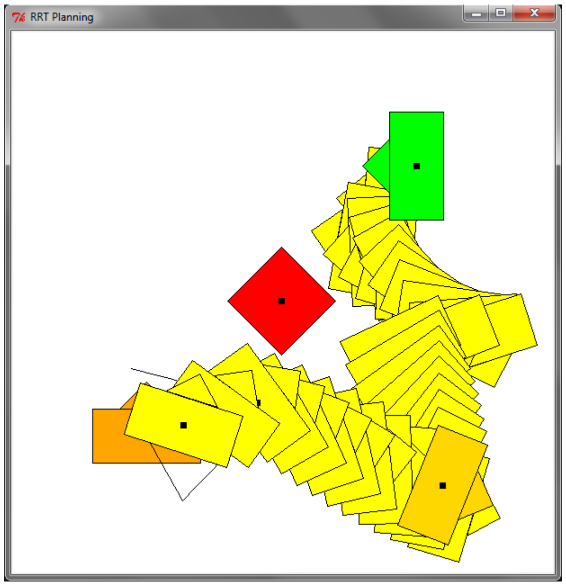
The starting configuration of the robot is orange. The obstacles are red. Reference points are drawn on the Objects. The goal configuration is green. Steps on the raw path are drawn in yellow. Steps on the smoothed path are gold instead. For the polygonal robot, I display the RRT projected from the full configuration space onto just the Cartesian subspace. The trees are represented by black line segments.

Polygonal Robot moving in (x, y, theta) where theta is fully rotational

Standard RRT



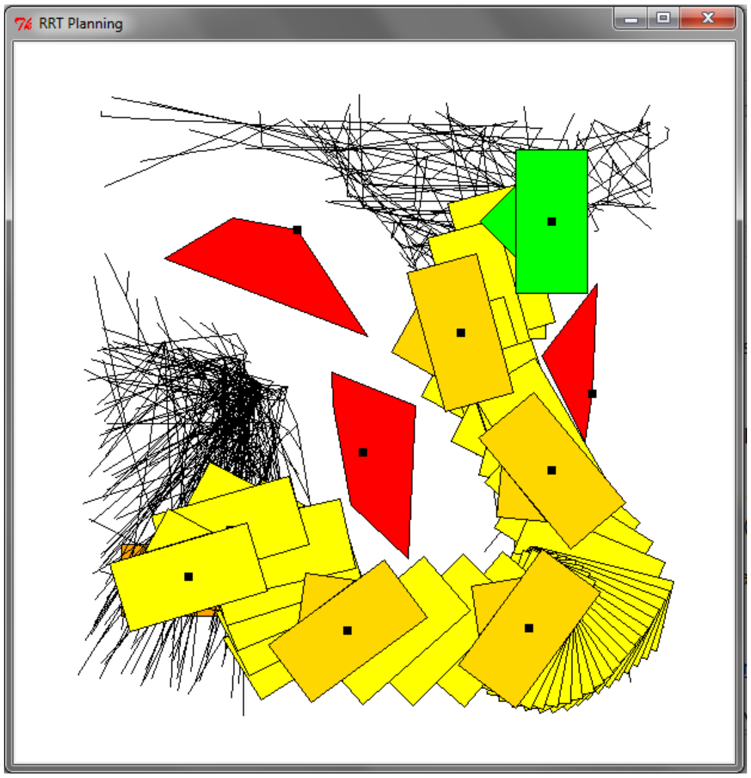
Bi-directional RRT



Algorithm	Run time (s)	Movements	Path length	Smoothed Movements	Smoothed Path Length
Standard RRT	0.631	27	23.047	3	21.220
Bi-directional RRT	0.083	34	41.498	2	33.523

Polygonal Robot moving in (x, y, theta) where theta is fully rotational. This is a harder world so I just ran Bi-directional RRT.

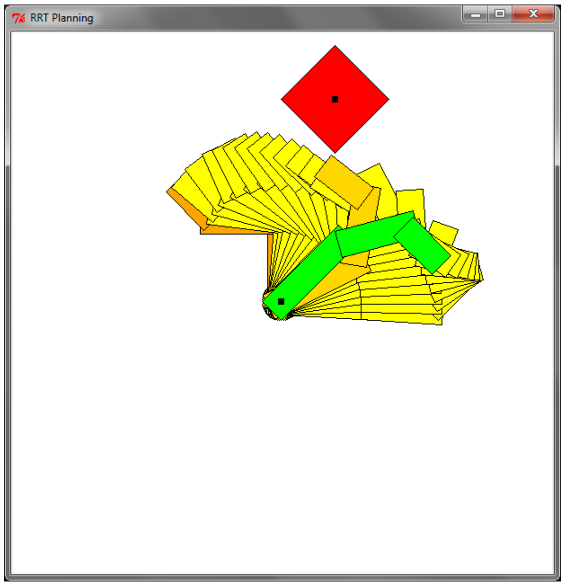
Bi-directional RRT



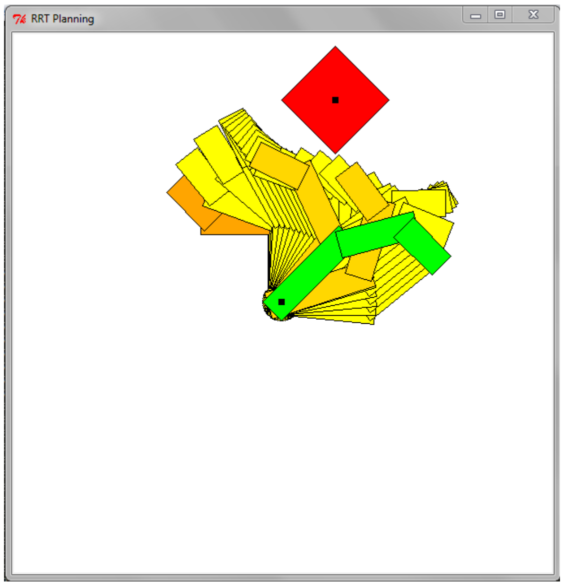
Algorithm	Run time (s)	Movements	Path length	Smoothed Movements	Smoothed Path Length
Bi-directional RRT	11.638	42	36.722	5	29.301

Chain Robot moving in (theta_1, theta_2, theta_3) where each theta is constrained between (-pi/2 and pi/2)

Standard RRT

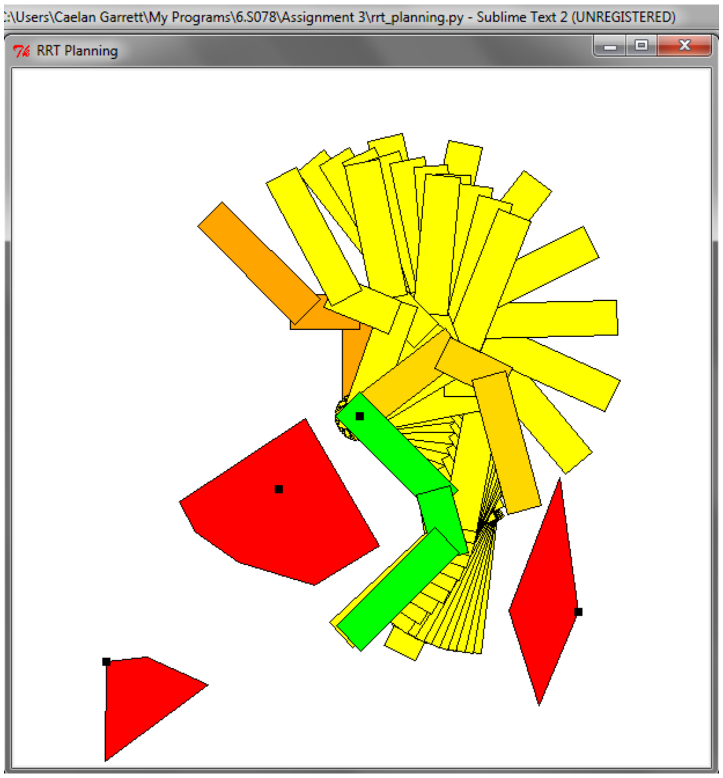


Bi-directional RRT



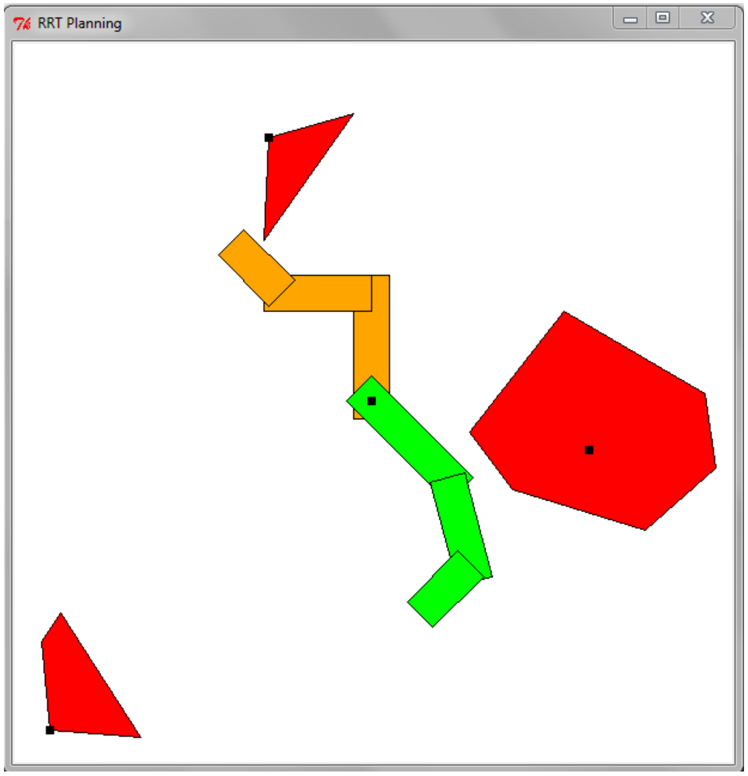
Algorithm	Run time (s)	Movements	Path length	Smoothed Movements	Smoothed Path Length
Standard RRT	1.557	26	12.245	2	8.872
Bi-directional RRT	0.143	32	14.217	3	9.918

Chain Robot moving in (theta_1, theta_2, theta_3) where each theta is constrained between (-pi/2 and pi/2). This is a harder world so I just ran Bi-directional RRT.


:\Users\Caelan Garrett\My Programs\6.S078\Assignment 3\rrt_planning.py - Sublime Text 2 (UNREGISTERED)
RRT Planning

Algorithm	Run time (s)	Movements	Path length	Smoothed Movements	Smoothed Path Length
Bi-directional RRT	3.961	43	21.799	2	11.331

Chain Robot moving in (theta_1, theta_2, theta_3) where each theta is constrained between (-pi/2 and pi/2). This is example of the Bi-directional RRT algorithm running for 34.531 seconds and failing to find a path due to the angle constraints.


RRT Planning