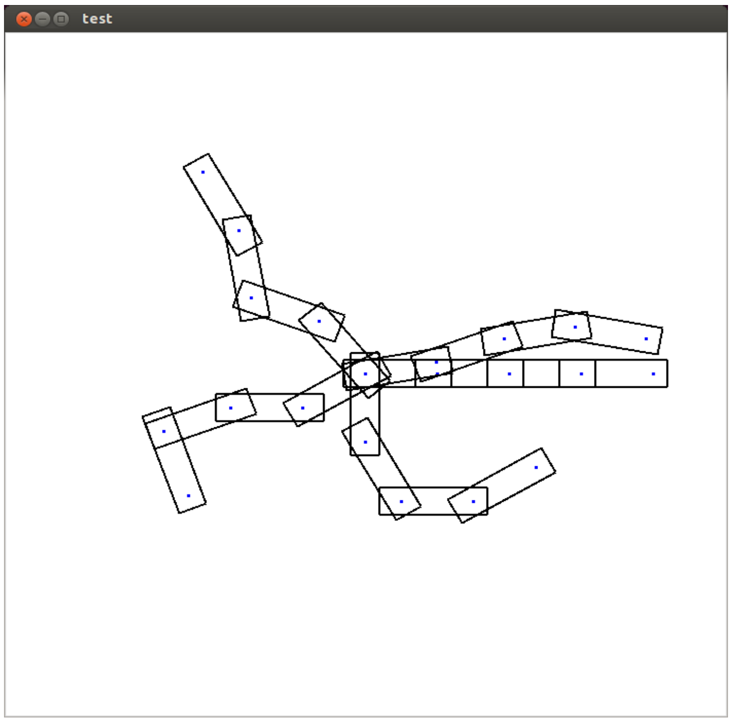
Jorge Perez

Problem set #3
6.S078
Planning Algorithms

<u>Part 0.</u> *Write the collision-checking interface for the two robots. You should have all the codefor the polygonal case already. For the n-link robot, define it as a list of convex polygons (one for each link), collision checking is then simply placing each link polygon given a conguration (a vector of joint-angles) and then doing polygonal collision checking. Dene the range of legal joint angles for each joint to be less that the full 2 range*

In part 0, we want to write the collision black box that the RRT is going to use. There collision checking for the motion of polygons in 2D was already done in Assignment 2. Thus, I will talk briefly about the implementation for the N-link robot.

The N-link polygon is defined in the code as a list of links that are mated by a pin(can only rotate around the pin). Links are polygons that have two reference points, which indicates the two points where they can be mated to another link. The first reference point is where the link is mated to the parent link and the second reference point is where the link is mated to the child link.



N-link Robot in different positions

Part 1. . *Implement the one-directional RRT, as described in LaValle. You can use a “brute force" nearest neighbor method or (if you can install scipy) look at the k-d tree implementation in scipy.spatial). You will need to have a \goal bias" (sample the goal 5-10% of the time). Run experiments for both robots.*

Metric:The metric I used was the euclidean norm for both the polygon robot and for the n-link robot.

Finding Nearest Neighbor: I used the brute force approach to finding the nearest neighbor. The code starts at the root and check all the nodes down the tree and returns the smallest distance according to the distance metric.

Collision Box: The collision box takes as parameters a list of obstacles and the robot. It has only one method “collides” which takes a robot configuration and checks for collisions. To check for collisions it puts the robot in the given configuration and checks for collisions with the list of obstacles.

Goal Bias: The algorithm samples the goal approximately 10% of the time

Node Spacing: I add a node every 5 legal steps.

Dimensions:

Polygon Robot:

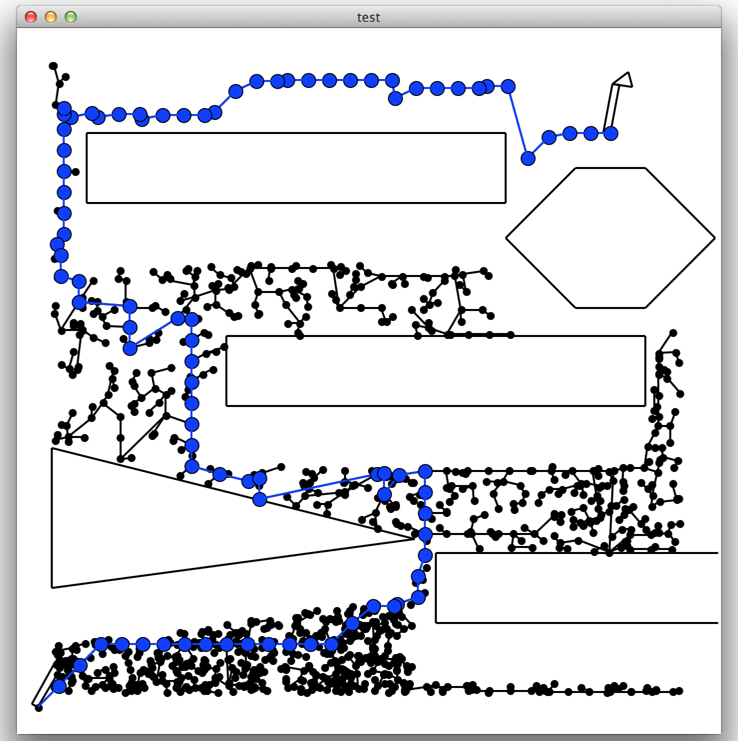
Dimension	Min	Max	Step size
x	5	95	0.5
y	5	95	0.5
ϴ	-90	90	1

N-Link Robot:

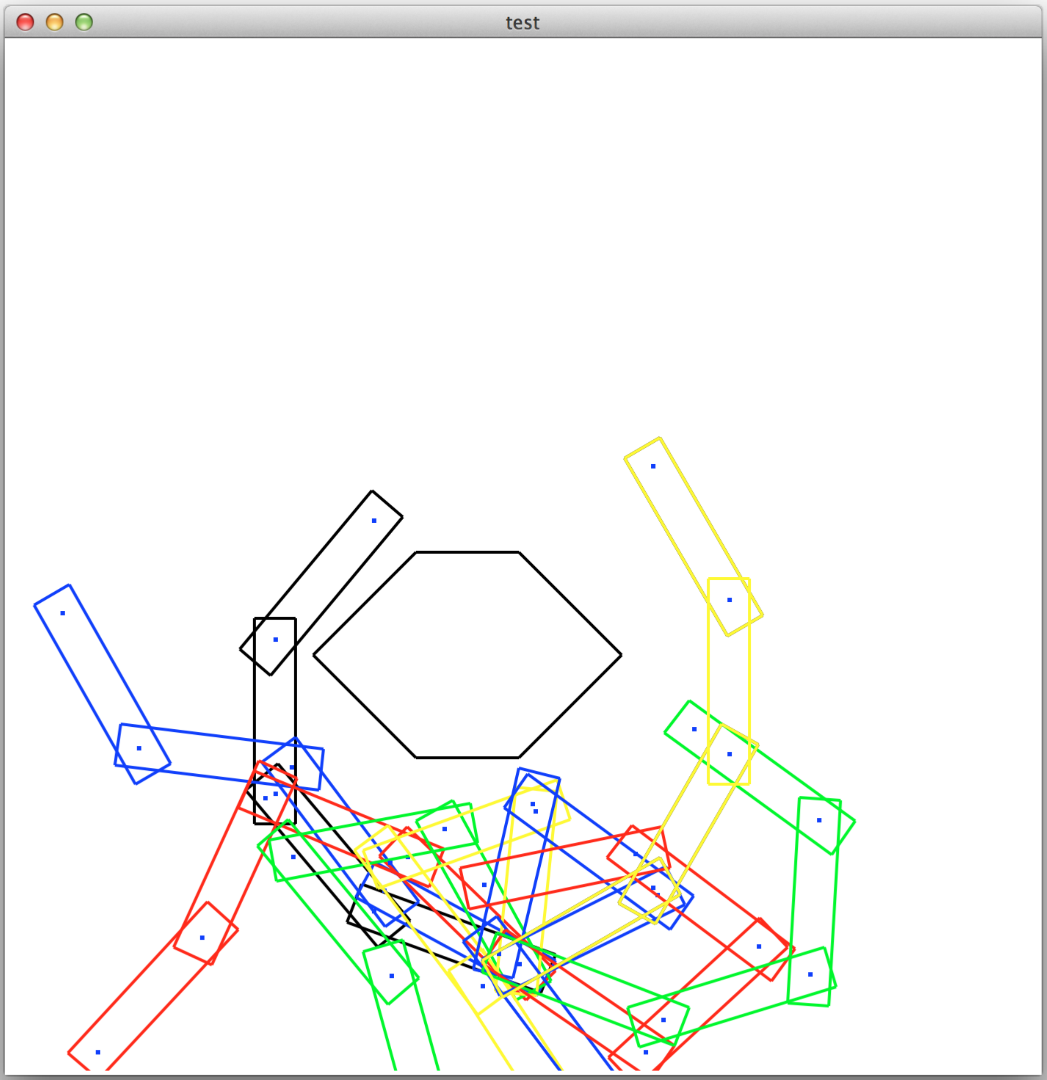
Dimension	Min	Max	Step size
ϴ1	-120	160	1
ϴ2	-120	120	1
ϴ3	-120	120	1
ϴ4	-120	120	1

Note: the links are assumed to the in different planes so they can overlap in the 2D planning.

Run for the Polygon Robot:



Note: In this run the robot was not allowed to rotate.
Run for the N-Link Robot:



Note: The start is the yellow configuration to the right and the goal is the black configuration to the left. The order of color is yellow, green, red, blue then loops. Part of a configuration might be blocked by a later configuration that was drawn on top. For the sake of clarity, not all of the steps in the plan were drawn.

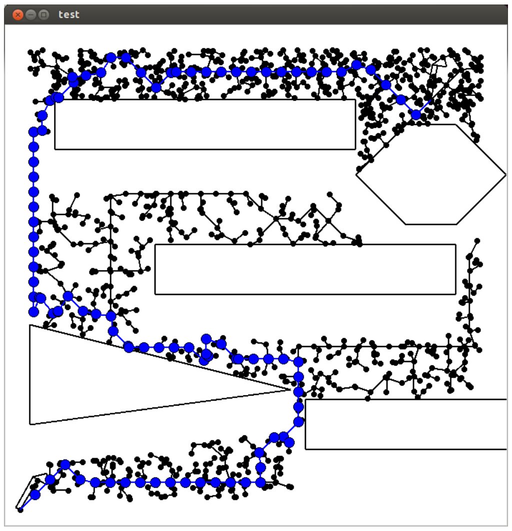
<u>Part 2.</u> *Implement the bi-directional RRT (be careful putting the paths from each tree together). Run experiments for both robots. Compare performance with the one-directional RRT*

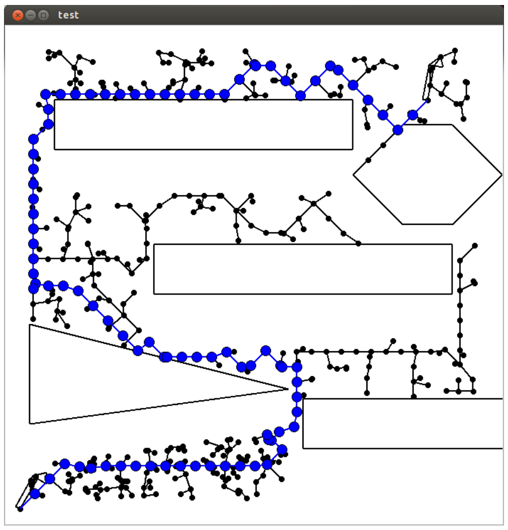
There were two main ideas needed to transition from the single direction RRT to the bidirectional RRT.

Changing trees: The criteria was that the smaller tree would be expanded.

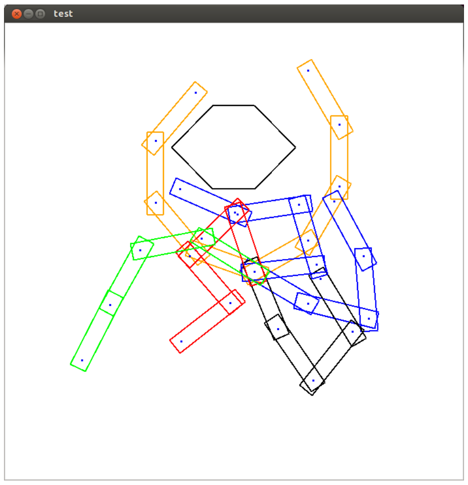
Connecting the trees: My implementation is very similar to the one that the book has. Every time new nodes are added to a tree, the last node added in the chain is checked for connection to the other tree.

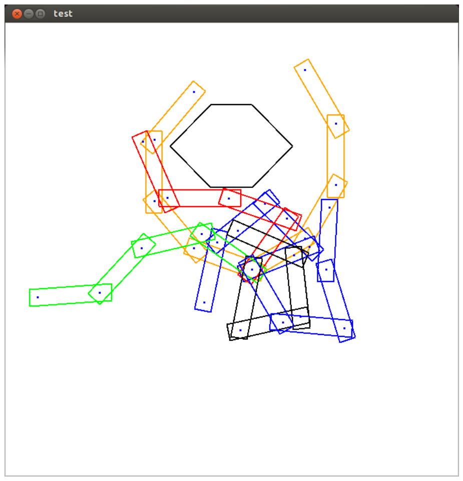
Runs for Polygon Robot with Bi-Directional RRT:





Runs for N-Link Robot With Bi-Directional RRT:





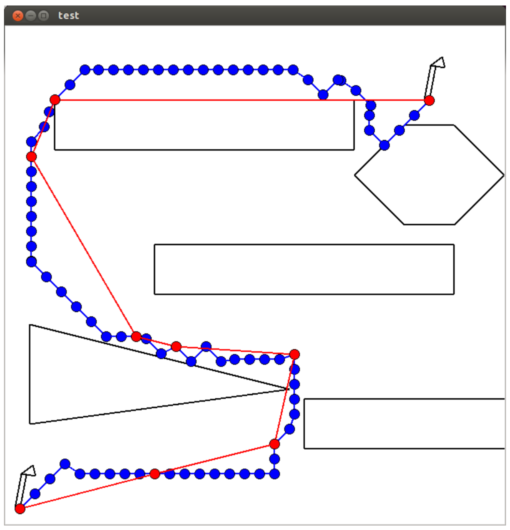
Note: The Black is the configuration that is connected to both trees. The sequence of colors start at the black followed by blue, red, green. The oranged configurations are the goal(right) and the start(left).

Comparison to RRT vs bi-directional RRT(nodes expanded):

Run	RRT (polygon robot)	bi-directional RRT (polygon robot)	RRT (n-link robot)	bi-directional RRT (n-link robot)
1	5346	607	1881	412
2	6699	364	5170	977
3	1980	1105	3520	600
4	1584	621	1606	1344
5	3190	498	4103	192
6	1969	4018	1476	283
7	715	3765	5214	270
8	792	704	1081	443
9	2134	808	4554	1062
10	1089	2600	3487	593
average	2,549.8	1,509	2,909	617.6

In both cases the bi-directional RRT need less sample points on average to get the path.

Part 3. *Implement “random short-cut" smoothing for the paths, illustrate the effect (and the cost). Compare to paths from the visibility graph method for the polygonal robot case. Make sure you have experiments where there are two ways to reach a goal (going to the left and right of one or more obstacles).*



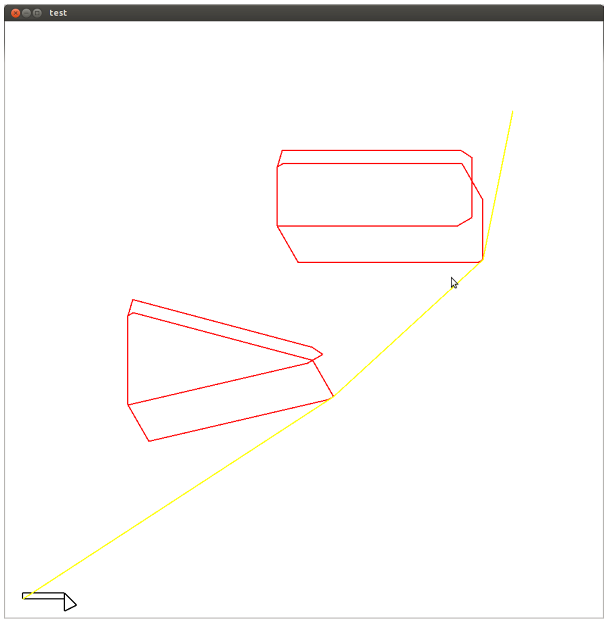
100 iterations of shortcutting

Comparison to RRT vs bi-directional RRT vs shortcut(number of collision checks for polygon robot):

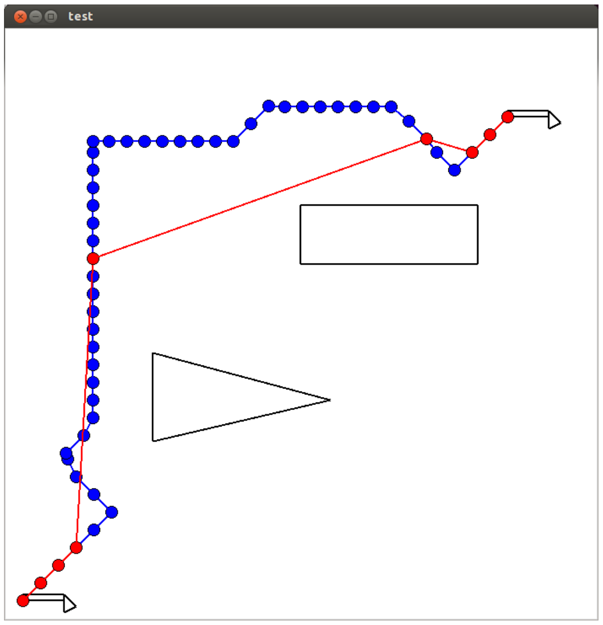
Run	RRT	bi-directional RRT	random shorcut(30 iterations trimming bi-directional RRT path)
1	5077	2008	677
2	3317	11298	638
3	4827	5005	383
4	5908	9183	600
5	3712	4564	464
6	10260	7004	816
7	4194	5424	445
8	2492	5036	685
9	5556	6629	461
10	4952	9802	494

average	5026	6595	566

The cost of shortcutting the path varies with the amount of iterations. Thirty iterations enhanced the path but could be better. The shortcut algorithm performed less collisions checks on average by a magnitude of 10 than the cost for the tree.



Path obtained with Visibility Graph



Path Obtained with bi-directional RRT and random shortcut(20 iterations)