Multi-Agent Bi-directional RRT

Robot Motion Planning :: ITCS:8152, Spring 2018

Sayantan Datta
sdatta3@uncc.edu

Introduction

My project works with making an RRT framework[1] for multiple robot moving in a formation. The motivation comes from a formation of soldiers guarding a Packed Trebuchet, while it slowly moves in the workspace. As the soldiers would move close to the trebuchet while going through a narrow path, and then spread out otherwise, this project tries to emulate the the convex hull path planning of the entire configuration of the robot.

I have implemented till the bidirectional RRT of the changing convex hull in the workspace. The visualization is still limited to single robot bidirectional RRT.

Implementation

I implemented this project in Klampt, where I completed my code in stages. Implementation details and abstractions are all mentioned in these stages

Stage 1: Implement a basic bi-directional RRT planner for a single non-holonomic robot

I used a Turtlebot to model the Trebuchet. I used a Turtlebot[2] to make the trajectory from one start point to the goal point. The turtlebot was constrained to move in either of the two steps:

a. Make a point rotation
b. Go straight ahead

Along with this point turn based system, I made another constraint that the robot can only make $\pi/18$ orientation steps while going from one step to another. I relaxed this constraint when the two paths of the bidirectional RRT meets, just to make the RRT converge faster.

The RRT has been implemented using an edge list representation of the graph. As for most of our test cases, the number of nodes in the RRT path did not exceed 150, the processing time was pretty low, within 0.5seconds in most cases, with outliers at around 1 second. As I did not implement a KD tree, graph search was expensive, being O(n) time complexity.

Once the path is complete, the discrete path is iterated upon, for each pose in the RRT path, a small value of angular velocity is added till the orientation of the robot falls within a small threshold of the required orientation ($0.005 * \pi$). Once the orientation is achieved, a small velocity is applied on the robot, till the robot achieves the position for the next pose as generated by the RRT algorithm. This too has an check for eplison threshold, after which we assume that the robot has reached the next pose, and the next control statements are applied.

Although the code mostly works well, sometimes while making rotations, the angular velocity being more fast than it needs to be, does not fall in the small bracket of acceptable orientation for the next pose. In such cases, the robot takes a longer time to rotate back and forth till it achieves the exact orientation. This is common especially for the edge connection the last pose of the path generated from start and goal poses respectively.

This part of the code works well, and is available at:
https://github.com/Kenzo450D/TrebuchetPlanning/tree/a37892035eb278e193e372c715102a8cbdaacd43

Stage 2: Implement the RRT with a varying robot size
In this stage of the implementation, I started using the idea of implementing a convex hull approximating my formation of the robot. So I had a bunch of parameters to play with: the size of the robot, and when do I change from one formation to another. To keep things simple, I used two formations: one large and one small. This time, I also kobuki robot base instead of the whole turtlebot. I made the large configuration by default, and changed to the smaller configuration only during an obstacle.

The previous bidirectional RRT algorithm was modified by the following steps:

1. Select a random point on the workspace and extend the tree towards this random point. This extended node is η
2. Check if η has is in collision with the environment.
    a. If η is in a collision configuration, change the robot to the tighter configuration, and go to Step 2. If there is no tigher configuration available, report η as not possible.
    b. If η is not in collision with the environment, report η as possible with the current configuration of the robot used
3. Check if η is within the neighborhood of η', where η' is the new node extended from the goal pose. If yes, then join these two vertices with an edge and exit, else go to Step 1.

My convex Hull -bidirectional RRT planner works well, it shows the discrete steps of the RRT path along with the formation of the robot which allows for that particular pose on the path. But I could not illustrate it on Klampt.

Implementation of which is found here:
https://github.com/Kenzo450D/TrebuchetPlanning

Results

The following screenshots illustrate the RRT planner for the single robot system, running on Klampt:

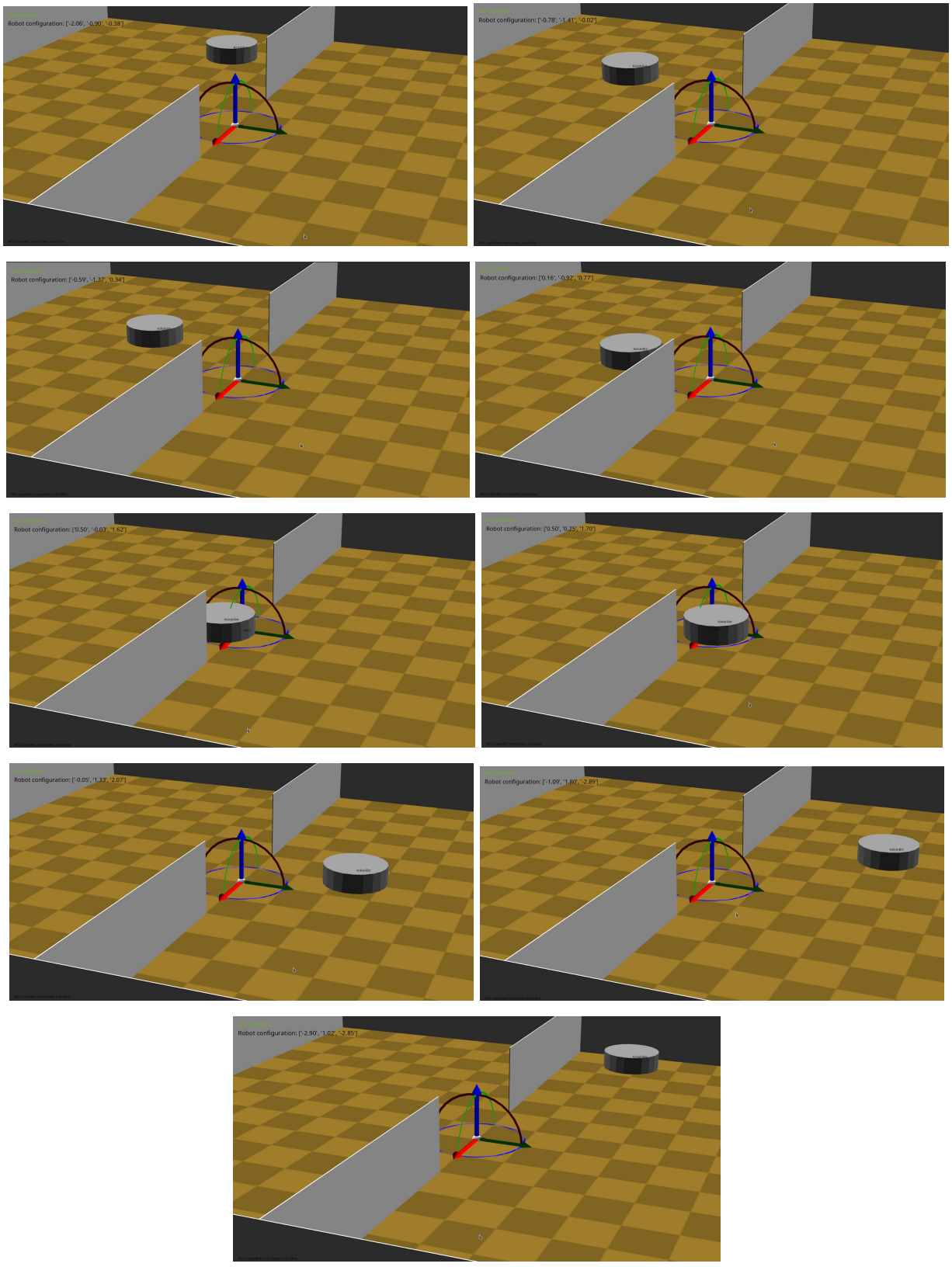

Figure 1: Poses of the robot during Bidirectional RRT Visualization of a single robot

The output of the bidirectional RRT for a sample workspace shows the configuration to use and the position and orientation the robot should be in:

```
kinSim :: Robot : 0 Config:  32  ::  0.219 -2.163 1.134
kinSim :: Robot : 0 Config:  33  ::  0.262 -2.072 1.309
kinSim :: Robot : 0 Config:  34  ::  0.288 -1.976 1.483
kinSim :: Robot : 0 Config:  35  ::  0.296 -1.876 1.658
kinSim :: Robot : 0 Config:  36  ::  0.288 -1.776 1.833
kinSim :: Robot : 0 Config:  37  ::  0.262 -1.68 1.658
kinSim :: Robot : 0 Config:  38  ::  0.253 -1.58 1.833
kinSim :: Robot : 0 Config:  39  ::  0.227 -1.484 1.658
kinSim :: Robot : 0 Config:  40  ::  0.218 -1.384 1.483
kinSim :: Robot : 0 Config:  41  ::  0.227 -1.284 1.309
kinSim :: Robot : 1 Config:  42  ::  0.253 -1.188 1.483
kinSim :: Robot : 1 Config:  43  ::  0.262 -1.088 1.658
kinSim :: Robot : 1 Config:  44  ::  0.253 -0.988 1.833
kinSim :: Robot : 1 Config:  45  ::  0.227 -0.892 1.726
kinSim :: Robot : 1 Config:  46  ::  0.212 -0.793 1.551
kinSim :: Robot : 1 Config:  47  ::  0.214 -0.693 1.726
kinSim :: Robot : 1 Config:  48  ::  0.198 -0.594 1.551
kinSim :: Robot : 1 Config:  49  ::  0.2 -0.494 1.377
kinSim :: Robot : 1 Config:  50  ::  0.219 -0.396 1.551
kinSim :: Robot : 1 Config:  51  ::  0.221 -0.296 1.518
kinSim :: Robot : 1 Config:  52  ::  0.227 -0.196 1.343
kinSim :: Robot : 1 Config:  53  ::  0.249 -0.099 1.508
kinSim :: Robot : 1 Config:  54  ::  0.256 0.001 1.333
kinSim :: Robot : 1 Config:  55  ::  0.279 0.098 1.508
```

Figure 2: Output information of Bi-directional RRT using a convex hull

References

1. LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." (1998)
2. Garage, Willow. "Turtlebot." *Website: http://turtlebot. com/last visited* (2011): 11-25.