CS 5335, Fall 2017
Project Report

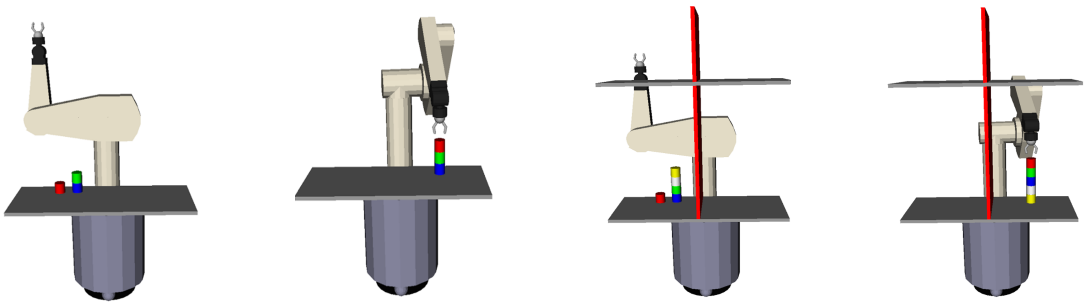
Shashwat Sanghavi
sanghavi.s@husky.neu.edu

Harshil Vasani
vasani.h@husky.neu.edu

December 2017

1 Introduction & Problem Statement

The problem of finding an object in space and placing it at a particular location is a very well known issue in the field of robotics. The process of moving an object from one place to another uses the concepts of kinematics, motion planning, obstacle avoidance and many more. This report explains an algorithm to solve a famous block world problem using a Puma robot arm. In the block world problem, the robot knows the current positions of the blocks in the space. The robot is required to move blocks to stack them up at particular position in particular order. For example, the blocks are placed on the table as shown in the Figure 1(a). The job of the robot is to place them at some other location in the desired output stack order (here Top: Red, Middle: Green, Base: Blue) as shown in Figure 1 (b). The time complexity of the problem increases with increase in number of blocks and obstacles as shown in Figure 1 (c) (d). The proposed algorithms works for N number of blocks.



(a) Initial condition (b) Final condition (c) Initial condition (d) Final condition

Figure 1: Problem Introduction
(a) and (b) are images for block world problem with 3 blocks
(c) and (d) are images for block world problem with 5 blocks and obstacles

The proposed algorithm is used to find the correct order for moving blocks to get the desired output stack order. Moreover, the algorithm uses bidirectional RRT to find an obstacle free path to move each block to a particular location. In the following sections, the report will explain both the algorithms in detail. It will also compare two motion planning algorithms RRT and Bidirectional RRT.

2 The Approach

2.1 Block ordering algorithm

2.1.1 Assumptions

The algorithm tries to solve the problem with following assumptions.

1. Initially blocks are placed in any of these:
   - All the blocks are lying at different location with none stacked above the other.
   - Blocks are stacked in a single column.
   - Blocks are stacked in two different colums.
2. Each block is a cylinder with radius r and height h. For the implementation r and h are assumed to be 0. px and 0.06 px respectively.
3. The initial positions of the blocks are such that the robot can pick the object without coliding with table or wall element.

2.1.2 Stack ordering and block moving algorithm

The Algorithm-1 contains a pseudo code for moving the blocks to obtain a desired stack order. Here the desired output order needs to be provided. The Algorithm-2 will identify the current configuration of the blocks and it will determine the order in which they are placed on the table. The line 7 of Algorithm-1 checks whether all the blocks are within two stacks or not? If they are not placed in two stacks, then according to the assumption, they are all placed on the table at different locations. In that case, the algorithm moves each block to the destination location as per the desired output stack order. If the blocks are arranged in at most two stacks, then as the 15th line of Algorithm-1 suggests, it checks whether the top blocks from any of the two stacks is the desired block to move to the destination or not? If yes, then the algorithm will move the block to the destination position. If the algorithm can not find the desired block, then it will move the blocks between inputStack and tempStack to find the desired block. Performing this process iteratively will eventually terminate the program after moving all the blocks to desired position in the desired output order.

Complexity for Algorithm1

In general the recursion function of complexity can be written as : $t(n) <= O(n) + t(n-1) <= c*n + t(n-1)$, where $c*n$ time is needed for N comparisons to find 1^{st} block to be placed on the destination location where c is some constant and $t(n-1)$ is total time for $N-1$ blocks. Now, solving the recursion equation gives : $t(n) <= c*n+c*(n-1)+t(n-2) <= ... <= c*(1+2+...+n)$, which reduces to $t(n) <= c*n(n+1)/2, i.eO(N^2)$.

Alternate Approach

General approach is to place all the blocks individually at different obstacle free locations from any given input configuration. For this approach we assume that we will have N obstacle free locations available on the table. We can roughly divide the algorithm in 2 steps, where the first step is to find and place N blocks on N different obstacle free locations and step 2 will be to put those blocks in the destination location as per desired output stack order. Step 2 is just placing each block to the destination location hence it will be done in $O(N)$ time complexity. For Step 1 let's assume dimensions of table is $L \times B$ where L is length of table and B is breadth of table. If we have blocks of size 1 pixel we need to check each $L \times B$ pixels. Which makes complexity of step 1 as $O(L \times B)$.

Algorithm 1: Block World solver for robotic arm

Result: Simulation ends with odered stack

```
outputStack = <input(color order for output Stack)>
destination = <input(3D location)>
inputStack, tempStack = identifyStacks();
iL = inputStack.length;
tL = tempStack.lenth;
oL = outputStack.length;
if iL+tL<oL then
    while outputStack!=null do
        c = outputStack.pop();
        block = blockWithColor(c);
        pick block & move block to destination with the path found by biRRT;
        destination.z += block.height
    end
else
    while outputStack!=null do
        block = blockWithColor(outputStack.pop());
        if block = blockWithColor(inputStack.pop()) then
            pick block from inputStack & move block to destination with the path found by
             biRRT
        if block = blockWirhColor(inputStack.pop()) then
            pick block from tempStack & move block to destination with the path found by
             biRRT
        else
            if inputStack = [] then
                swap inputStack and tempStack;
            pick block from inputStack & move block to destination with the path found by
             biRRT
        end
    end
end
```

Algorithm 2: identifyStack

Result: inputStack, tempStack

```
blockList = List of cylinders with their location;
inputStack = list() outputStack = list() while blockList != null do
    if inputStack == null or (isInRadius(inputStack.first,blockList.first()) then
        inputStack.add(blockList.pop());
    if tempStack == null or (isInRadius(tempStack.first,blockList.first()) then
        tempStack.add(blockList.pop());
    end
end
```

Algorithm 3: isInRadius

Result: Answer

```
l1 = center Of Cylinder1 in XY plane;
l2 = center Of Cylinder2 in XY plane;
r1 = radius of Cylinder1; r2 = radius of Cylinder2; Answer = (Boolean)
 euclideanDistance(l1,l2) < r1+r2;
```

Comparison

The 1^{st} approach shall be used if N^2 is $< L \times B$ else the 2^{nd} approach. Advantage of the 1^{st} approach is that for a small count of N the algorithm will be very fast, but its disadvantage is that because it is dependent on the number of blocks, the time increases by $f(x^2)$ function. Advantage of the second approach is that if $N^2 > L \times B$, the time complexity will be fixed no matter what the value of N is, as the algorithm is independent of the count of blocks. However its disadvantage is that it will be slow in comparison to the 1^{st} algorithm, if N has a value such that $N^2 < L \times B$.

2.2 RRT vs Bidirectional RRT

Rapidly-exploring random tree (RRT) is one of the search algorithms which is widely used to determine obstacle free path for planning a robot motion. In RRT is a search algorithm which searches in the space by developing randomly and rapidly expanding tree in the space. A root of the tree always stays at the location intial location in the space. After that, until the goal is found in the space, the algorithm generates a random point and connects it to the nearest point in the tree. It checks for obstacle between nearest point and new point before adding it to the tree. The point is not added to the tree if an obstacle free path is not available. Because of the random point generation, the algorithm is bound to converge after some time T.

However, RRT is very slow while execution. It takes approximately 10000 iterations to reach a point with an error less than 0.45 euclidean distance. A variation of RRT is available where two trees are build from initial point and goal point. Whenever a new point is added, a distance between that point and the nearest point in other tree is calculated. If the distance is less than some threshold error and an obstacle free path is present between those two point, a link between them is developed. This algorithm can find a path with error less than 0.01 euclidean distance in less than 100 iterations. Thus, bidirectional RRT is much more efficient than RRT. The figure 2 shows the relationship between error and iteration for both the algorithms. The graphs values are the average for 100 trials.

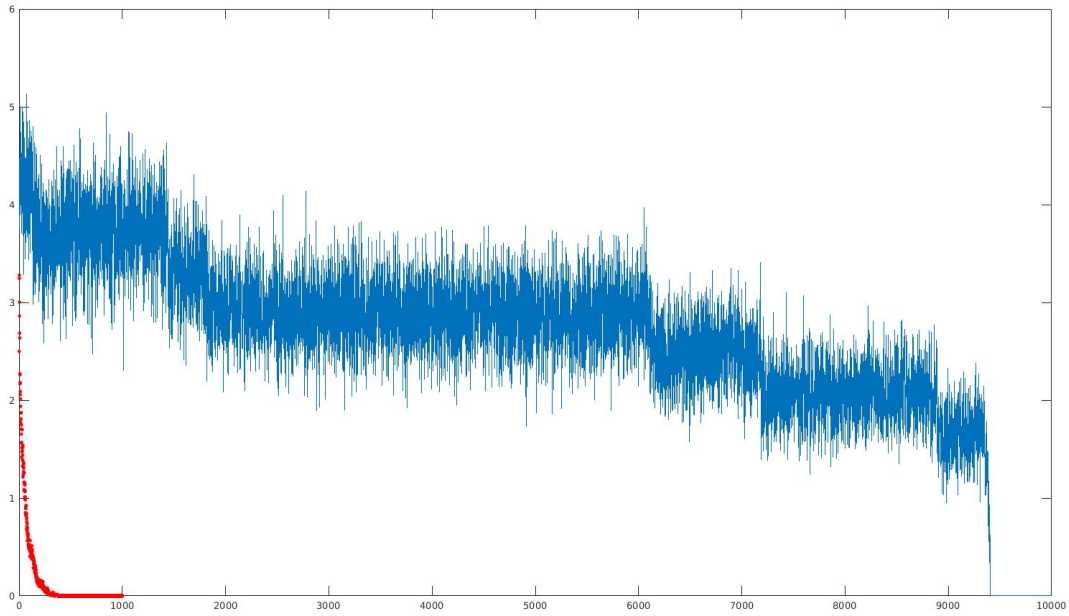


Figure 2: Comparison of RRT(Blue) and BiRRT(Red)
X axis: Number of Iterations
Y axis: Error

3 Results

3.1 Simulation

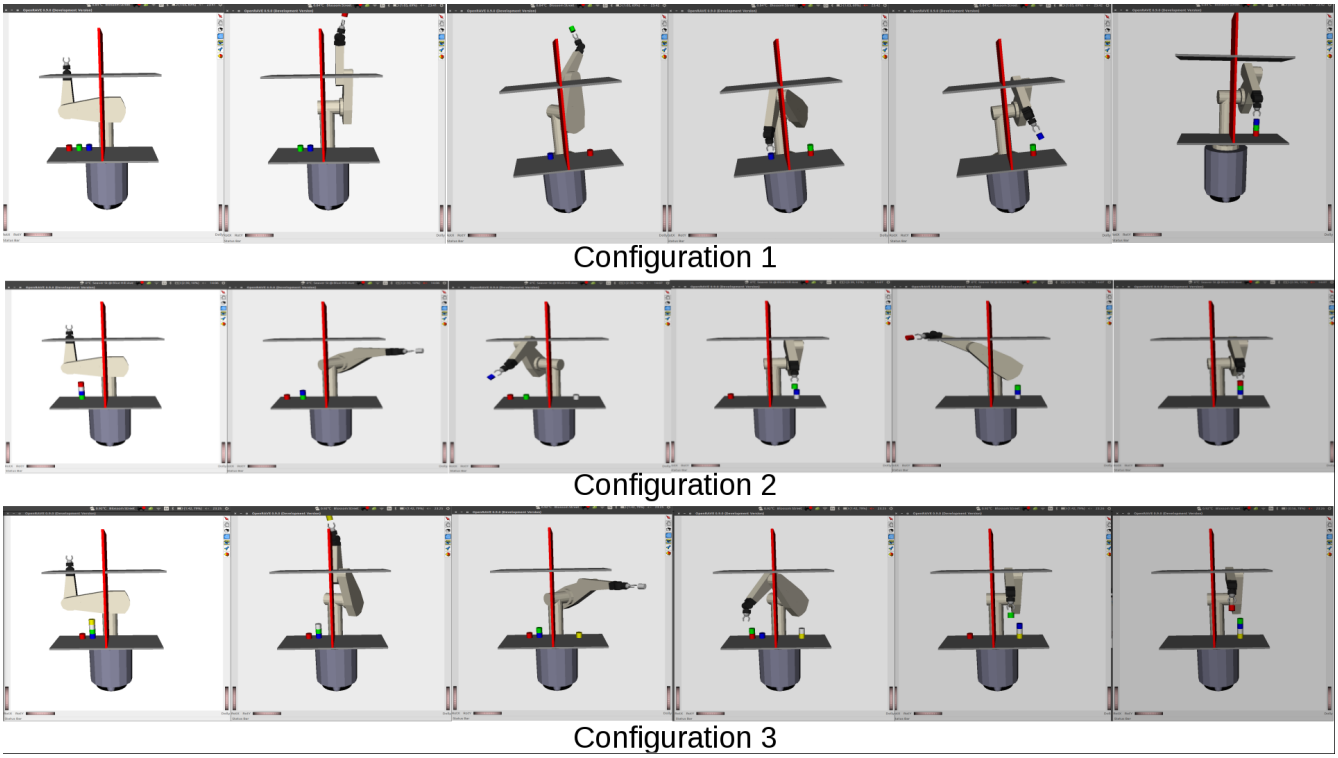


Figure 3: Simulation steps
Configuration 1: 3 blocks, all at different locations
Configuration 2: 4 blocks, all stacked in a single column
Configuration 3: 5 blocks, stacked in two different columns

Here in figure 3, it shows three different initial arrangements of blocks with different number of blocks for each of them. With that it also has two obstacles in the environment. It can be seen that for each arrangement, algorithm is able to move the blocks as per the desired output order successfully. Results show some of the intermediate configuration of blocks, which helps to visualize how the algorithm approaches each initial configurations.

3.2 Simulation run-time comparison

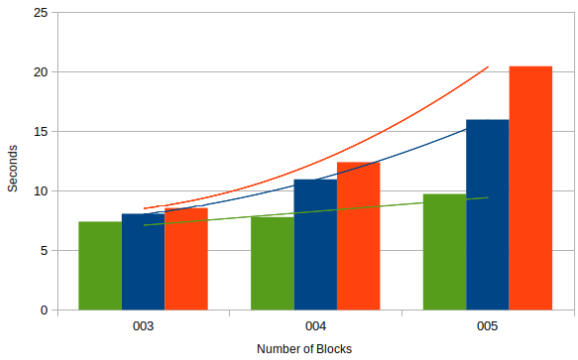


Figure 4: Comparision of different configurations
Green: Configuration 1: all the blocks are at different location
Blue: Configuration 2: all the blocks are in one stack
Orange: Configuration 3: the blocks are distributed in two stacks

The figure 4 shows the time of execution for all three configurationd. The tests were performed on a machine with i5 processor (2.2 Ghz), 8GB ram, and Ubuntu 17.04 operating system. The graph shows an average of 10 trials. Following are the conclusions of this simulation.

- It can be seen that for configuration 1 we get linear time graph which is correct as all the blocks are placed in different locations.
- Time complexity for the algorithm when using two stack approach is N^2, and that can be clearly seen by the graph curve(Blue and Red) that it is equivalent to curve for N^2.

4 conclusion

In our analysis, we found two algorithms that can be used for solving blocks world problem based on size of table and the number of blocks. Our analysis also shows the effects of different initial configurations of blocks with same number of blocks and different number of blocks for same configuration. It can be seen that if all the blocks are placed at different locations it takes the least amount of time. The analysis also shows the difference in speed for finding obstacle free path when using BiRRT vs RRT. With the speed factor, the analysis also shows that error factor for BiRRT is much less than RRT. Thus we can conclude that if we can somehow transform the initial configuration of blocks such that they all are placed on different locations, then placing them in a desired order will be very fast with using BiRRT for finding obstacle free path.

5 Links

Simulation Video: https://youtu.be/NG-w5wqSm1Q
Code Dump: https://github.com/srsanghavi/RoboticsProject.git