CISC 3060 Lab Assignment #4

Out: 11/17 Due: 11/28 end of day

This lab will offer you the opportunity to explore having a robot carry out a preplanned path. On the surface, it seems straightforward – the robot should go to each of the points on the path. But there are some issues with that: The preplanned path is dense in points, with very little distance between them; it includes easy long, straight-line sections and more dangerous sharp turns. The locomotion mechanism of the robot – wheels and motors – may not allow it to follow the path exactly.

Step 1: Download the programs pp.py, astar.py and pp_encmap.py from Blackboard. If you have your MoveIt node working, you can use that. Otherwise download the MoveIt.py node from Blackboard as well. Finally, you will need the file enclosed_ocmap.txt from Blackboard.

The files pp_encmap.py and astar.py are the path-planner – an occupancy-map based tree search as we covered in class. However, the file astar.py replaces the file bfs.py that we used in class, and it includes a faster and more efficient tree search method called A*. The file pp_encmap.py is very similar to the file maps1.py that you downloaded to test the path planner previously. However, it automatically loads the file 'enclosed_ocmap.txt' into the numpy array gMap for you and it also includes a function planPath(start,goal) which makes it easier to call the path planner on an (x,y) start and end couple. planPath will display the path overlayed on the occupancy grid and return the list of paths.

Finally, the file pp.py includes two parts: in the main program it calls the path planner for a prespecified start and end position ((2,.01) and (2,-4) respectively) but then it starts a follower node that will issue points on the path one by one to the *movegoal* topic.

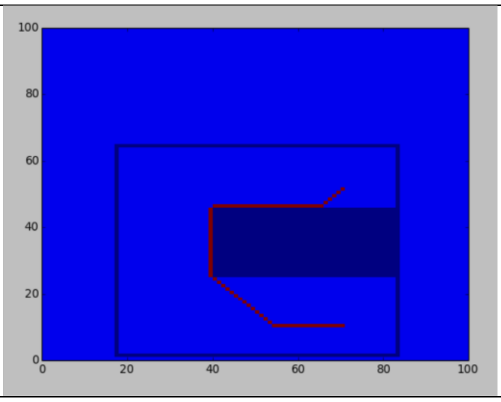


Figure 1: Matplotlib window displaying the planned path (2,0.1) to (2,-4).

In step 1, you need to download the programs and map file all into the same directory. You will need three linux shell windows: One to start to Turtlebot Gazebo, as usual. In another window, go to the directory location you stored the files, and start the MoveIt node. In the final window, go to the directory location that you stored the files and start pp.py.

You should first see the planned path, as shown here in Figure 1. This map follows the same conventions as we discussed in the class. It has been dilated to allow for a non-point robot. The thick blue central obstacle is the dilated center wall. The red path is the planned robot path. As soon as you kill the planned path map window, pp.py will start issuing (x,y) goals to the MoveIt node via the movegoal topic. Figure 2

(a) through Figure 2(d) shows a series of snapshots of the robot moving along the path from the start, Figure 2(a), to the end, Figure 2(d).

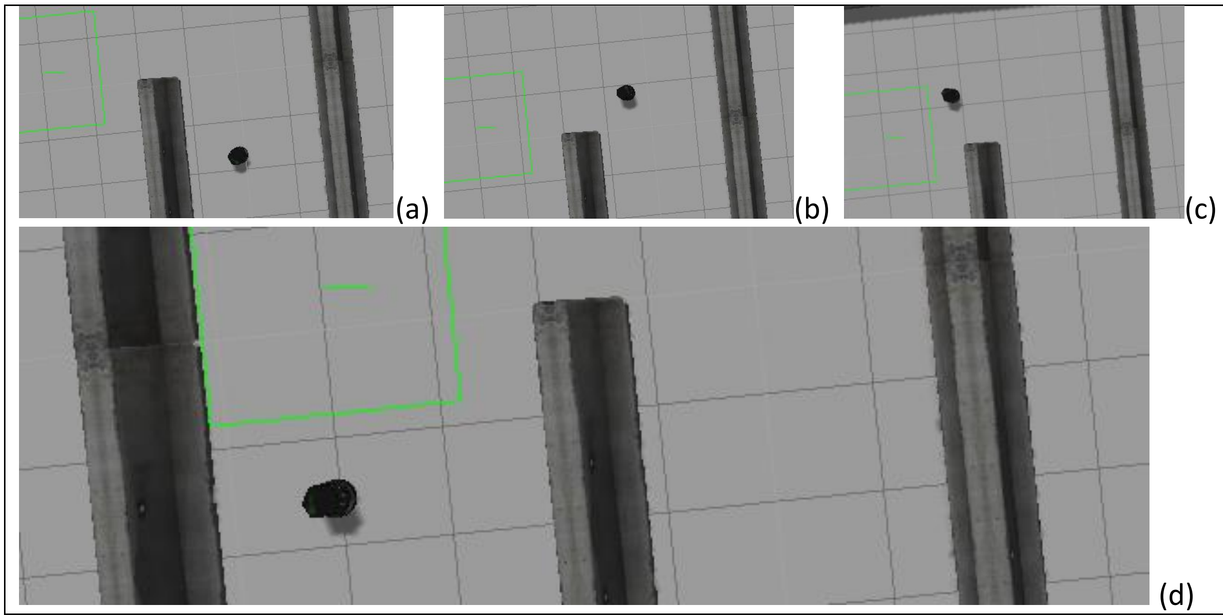


Figure 2: Snapshots of the robot location as it follows the planned path.

In addition to issuing the (x,y) locations to the movegoal topic, pp.py also keeps track of every position that the robot has occupied (the gTrackX and gTrackY global lists that are updated in the positionCallback function of pp.py). Once the path has finished, pp.py will display a matplotlib window with a scatter graph of these locations, as shown in Figure 3.

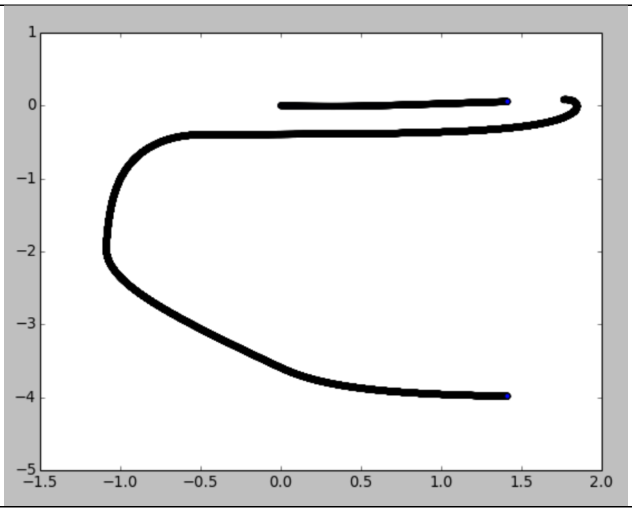
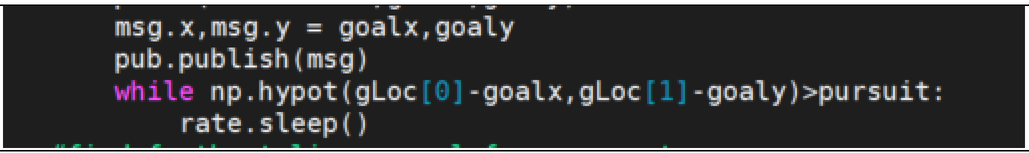


Figure 3: Track of all locations the robot occupied during path following.

The top, disconnected part of the path in Figure 3 is due to the robot moving from (0,0) to the start location (2,0.1) as the first step in pp.py.

You should make sure that you get the results shown previously before you start Step 2.

Step 2: The follower node sequences the (x,y) locations to the *movegoal* topic by checking to see when the robot has approached the previous goal, at the point in the code shown in Figure 4.



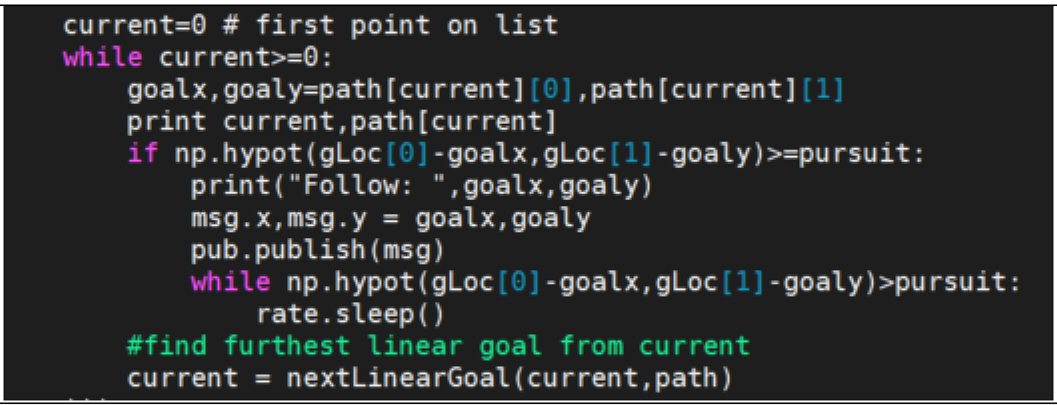
```
    msg.x,msg.y = goalx,goaly
    pub.publish(msg)
    while np.hypot(gLoc[0]-goalx,gLoc[1]-goaly)>pursuit:
        rate.sleep()
```

Figure 4: Location in follower node code where (x,y) output is sequenced.

A variable called 'pursuit' controls how close the robot has to be to each point, before the next point is issue. This variable controls a lot of the features of the path that is generated (Figure 3) including how closely it follows the planned path (Figure 1). You need to modify this variable and observe the results.

Modify pursuit to the values in the range {0.25, 0.3, 0.5, 1.0, 2.0} and observe all the results. Document your experiments with track graphs (like Figure 3) and explain **why** the robot behaved as it did and what the **pros** and **cons** were. Note that you can switch the start and goal values so that the path will run in reverse – this can save you time. If ever the robot hits anything (MoveIt will issue a "Bump" message) then terminate – you will need to restart it!

Step 3: Issuing locations one by one is not the most efficient way for the robot to move. Afterall, if it knew that a subsequence of points was in a straight line, it should just aim for the **last** point in the line.



```
current=0 # first point on list
while current>=0:
    goalx,goaly=path[current][0],path[current][1]
    print current,path[current]
    if np.hypot(gLoc[0]-goalx,gLoc[1]-goaly)>=pursuit:
        print("Follow: ",goalx,goaly)
        msg.x,msg.y = goalx,goaly
        pub.publish(msg)
        while np.hypot(gLoc[0]-goalx,gLoc[1]-goaly)>pursuit:
            rate.sleep()
    #find furthest linear goal from current
    current = nextLinearGoal(current,path)
```

Figure 5: Modified follower node main loop.

Modify the follower node so that before it issues each point, it checks to see if subsequent points are in a line and skips ahead to the last point in the line. This will require two changes. The first is to change the main loop of the follower node so that it can check for straight lines. Change the code between the 'gLogging=True' down as far as the 'gLogging=False' to be as shown in Figure 5. It is quite similar to the old loop. The main difference is a call at the end of the while loop to get the next point to send out, rather than just using a for loop to go through the path. You need to write the function nextLinearGoal, which take the index of the last point in the path sent, current, as well as the path, as arguments and returns the **index** of the next point to send.

To write this function, remember the equations for the slope (m) of a line given two points on the line, and the offset (c) of a line give one point and the slope: y = mx +c. If you know the slope and offset, then whenever you are given a point (x1,y1) you can determine whether it lies on the line or not by evaluating r = y1-m*x1-c. This (y=mx+c) will be zero for a point on the line, and non-zero otherwise!

The easiest way to see if you are on a straight line is to use the first two points to calculate m and c, then sum the value of r for the third, fourth, etc. until you reach a point where the sum of r exceeds some threshold value. That is, when the points have accumulated too much error from a straight line. You can use 0.1 as your default value of the threshold. Generate output to the screen from your function so you can keep track of how many 'lines' your program thinks there were in the path.

Write the code as instructed and run the program again. Compare the results with those in step 2. Document your experiments with track graphs (like Figure 3) and explain **why** the robot behaved as it did and what the **pros** and **cons** were.

Change the threshold around the range of 0.1 and observe how it affects trajectory following compared to the previous experiments. Document your experiments with track graphs (like Figure 3) and explain **why** the robot behaved as it did and what the **pros** and **cons** were.

Lab report: Your first section should present the objectives of this lab assignment and overview the rest of the document. You should then have one section per step in which you describe your methods and present your result. Do not write this as a list of 'what you did' chronologically – instead present your final solution and present and explain your observations as in prior reports.