Mobile Robot Systems

Assignment 2

Anik Roy

February 13, 2020

1.1 Potential Field Method

Exercise 1

(a) The plot of the potential field for moving towards the goal can be seen in figure 1. This field is calculated by taking the vector from the position towards the goal, and squaring the magnitude of the vector while keeping the same direction. This ensures that the robot 'slows down' as it gets closer to the goal, and that it will always attempt to move towards the goal. The figure shows how all the vectors point towards the goal, and that vectors close to the goal have a smaller magnitude

(b) In order to avoid obstacles, we also need to find a vector field which will let us move away from obstacles (a field that is repulsive). In order to do this, I first find the distance from the position to the surface of the obstacle by finding the euclidean distance and subtracting the radius. Since positions closer to the obstacle should be more 'repulsive', the reciprocal of this distance is the magnitude of the final vector. The direction of the final vector is directly away from the centre of the obstacle towards the robot. I also scale this by a tunable parameter to ensure the robot makes progress. This process is repeated for each obstacle, and the vectors produced are summed so that all the obstacles are taken into account. The resulting vector is also capped to a maximum speed.

(c) Figure 3 shows how combingin the two vector fields results in a new field which allows the robot to take a path to the goal while avoiding the obstacle. Since the vectors from the obstacle have a repulsive effect, and the vectors towards the goal have an attactive effect, they combine these effects to create vectors which go towards the goal but also away from the obstacle.

(d) In the previous part, the obstacle was at the position (0.2,0.3). Since the robot started at (-1.5,-1.5), and the vectors for the goal are direct, they will pass directly through the center of the obstacle. The vectors from the obstacle are in the opposite direction, as the direction of these vectors is the direction from the robot to the obstacle flipped. This means they will cancel each other out to produce a zero

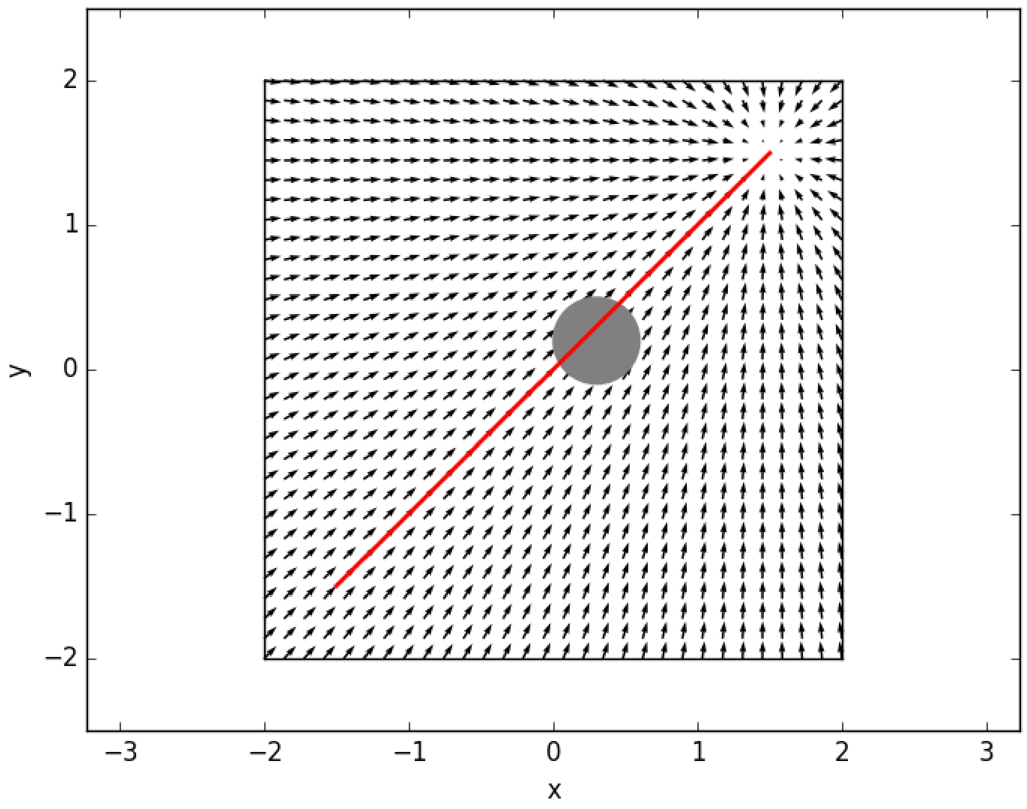


Figure 1: Potential field with the goal

vector somewhere between the start position and the obstacle, stopping the robot. This can be seen in 4, where the robot gets stuck halfway between the start and the obstacle. One possible solution is to change the direction of the vectors coming from the obstacle. Currently, they are directly in the direction of the robot, but they could be made to be skewed slightly so that the robot would always move in one direction. Once the robot moves in the direction, vectors towards the goal should allow the robot to keep moving. This still ensures the robot does not hit the obstacle, since the vectors are still repulsive, just in a slightly different direction.

(e) Figure 6 shows how the problem in part (d) can be fixed. The vectors coming from the obstacle are now rotated by 1 radian anti-clockwise (which can be tuned). This means that the robot will now take the path on the right of the obstacle, as it is pushed in that direction by these vectors. To calculate the vectors, I use the standard 2-D rotation matrix to ensure that the magnitude of the vector stays the same (it is rotated from its original direction around its base).

(f) The solution from part (e) still works with two obstacles, with the robot taking a similar path as it is repulsed from both the obstacles.

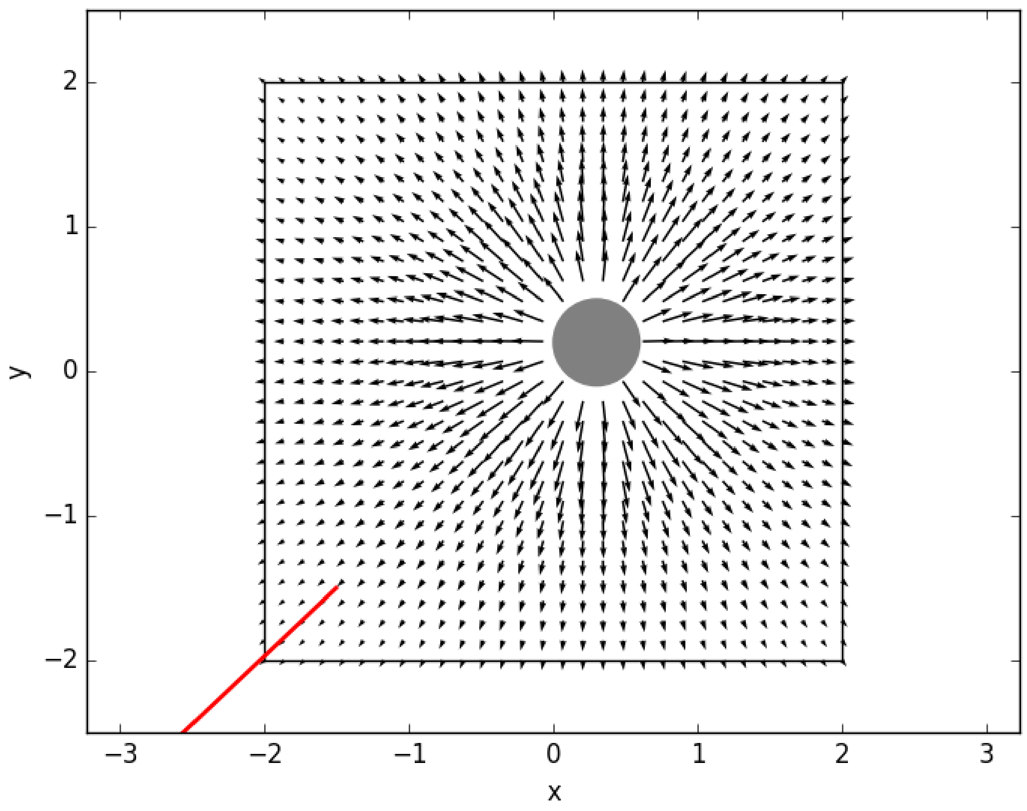


Figure 2: Potential field with the obstacles

Exercise 2

(a) The equations for feedback linearization:

$$\dot{x_p} = u\cos\theta - \epsilon\omega\sin\theta$$

$$\dot{y_p} = u\sin\theta + \epsilon\omega\cos\theta$$

Rearranged for u and ω:

$$u = x_p\cos\theta + y_p\sin\theta$$

$$\omega = \frac{(-x_p\sin\theta + y\cos\theta)}{\epsilon}$$

(b) The point of feedback linearisation is to allow a non-holonomic robot to appear to move holonomically, by controlling a holonomic point and calculating how the robot would move as if 'pulled' by that point.

(c) The plot below shows the trajectory of the robot as it navigates from the start to the goal.

(d)

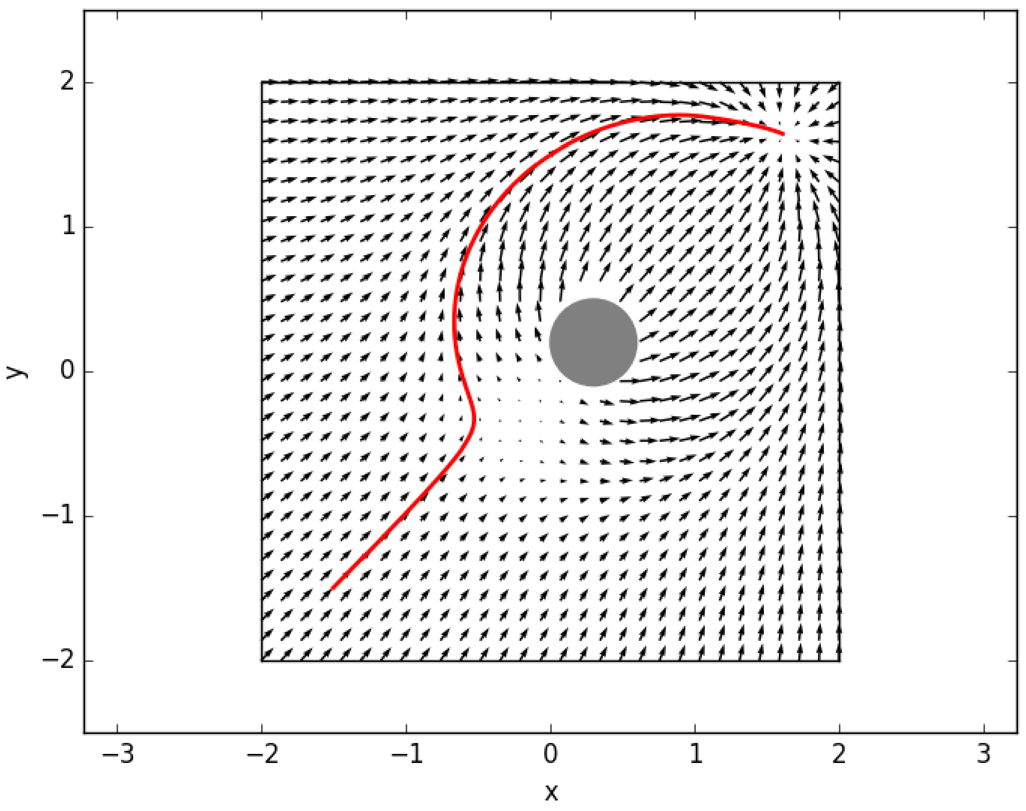


Figure 3: Potential field with both the goals and the obstacle

1.2 RapidlyExploring Random Trees

Exercise 3

(a) The `sample_random_position` method finds a random position that is not occupied (or unknown) within the occupancy grid. It does this by finding the size of the array (`grid.values`) backing the grid object, and from this calculating the possible positions in world coordinates that the grid represents. The `get_position` method of the grid allows us to find the position encoded by the first (top left, index [0,0]) and last (bottom right, index [len(grid), len(grid[0])]) elements of the occupancy grid. From this we have the set of coordinates we can sample from - we have the maximum and minimum for both x and y coordinated. We then sample uniformly from this range (both x and y), and check if that position is free. We continue sampling until we find a position that is free.

(b) The method of Rapidly exploring Random Trees (RRT) is used to plan a path to the goal. This method samples random points in the free space and attempts to connect the point to other points to build up a tree which explores the area of the free space. Figure 9 shows paths generated by my rrt method.

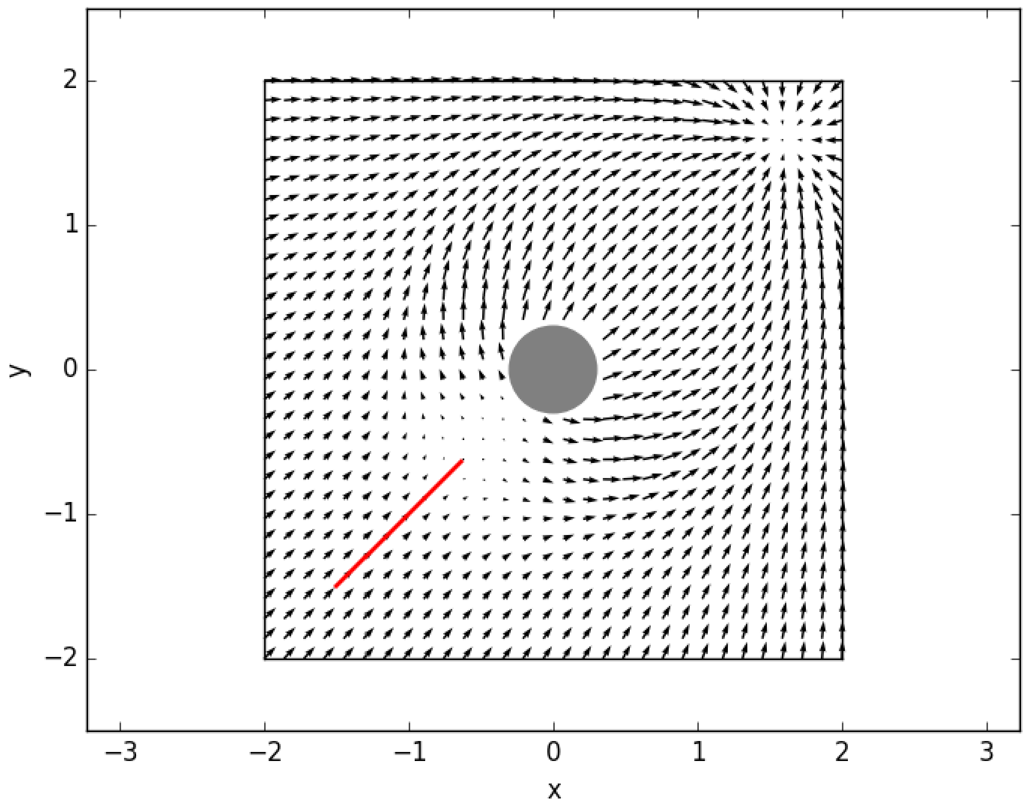


Figure 4: Potential field with the obstacle at (0,0)

(c) RRT will often find a suboptimal path due to the fact the the random points that are sampled will just be connected to the closest pre-existing node. This closest point may not always be the best point to connect to, since the path to that node may be longer that it needs to be. A solution is to use the RRT* algorithm, accumulating ‘costs’, which represent the distance taken by the shortest path from the root to the node. We can then connect to the nodes with the least cost rather than just the nearest neighbour. We can also carry out a ‘rewiring’ step, which will use the new node generated to find shorter paths for nearby nodes, changing the parent of nearby nodes to the new node if using the new node will reduce the cost to the nearby nodes. This ensures shorter paths are created from the new randomly generated position.

(d) Figure 10 shows the implementation of RRT* generating shorter and smoother paths.

Exercise 4

(a) Feedback linearized is implemented here the same as in exercise 2

(b) Motion primitives can be used to generate edges for non-holonomic robots. This

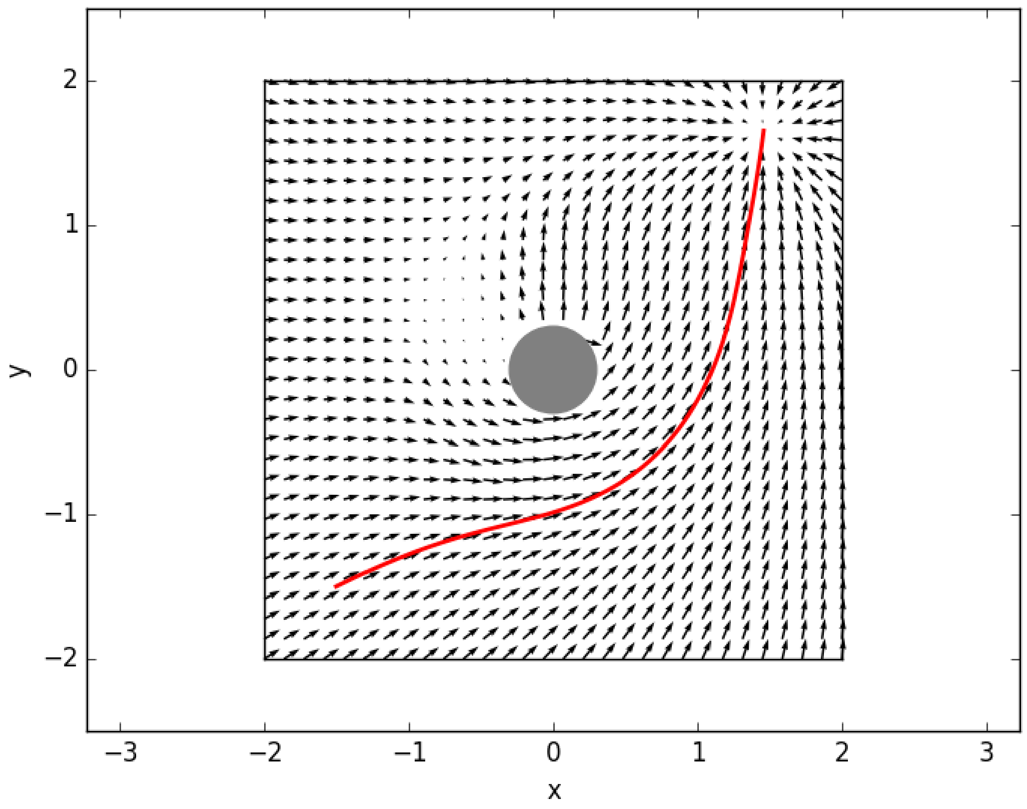


Figure 5: Potential field with the obstacle at (0,0), fixed

means we don't have to solve a boundary value problem (differential equations) to generate edges which are valid trajectories for non-holonomic robots. However, it does mean that the paths we generate are not always the best paths to take.

(c) The `get_velocity` method generates a velocity vector based on a robot position and a set of points which represent the desired trajectory of the robot. First, it finds the point on the path which is closest to the position of the robot. The subsequent point on the path is the direction in which the velocity is, since we want to make sure the robot gets closer to the desired path. If the closest point is the point at the end of the path, then the vector is just in that direction (avoiding an index being out of bounds). Figure 11 shows how a path is generated by rrt, which the robot then follows using `get_velocity` and `feedback_linearized`

(d) SLAM

(e)

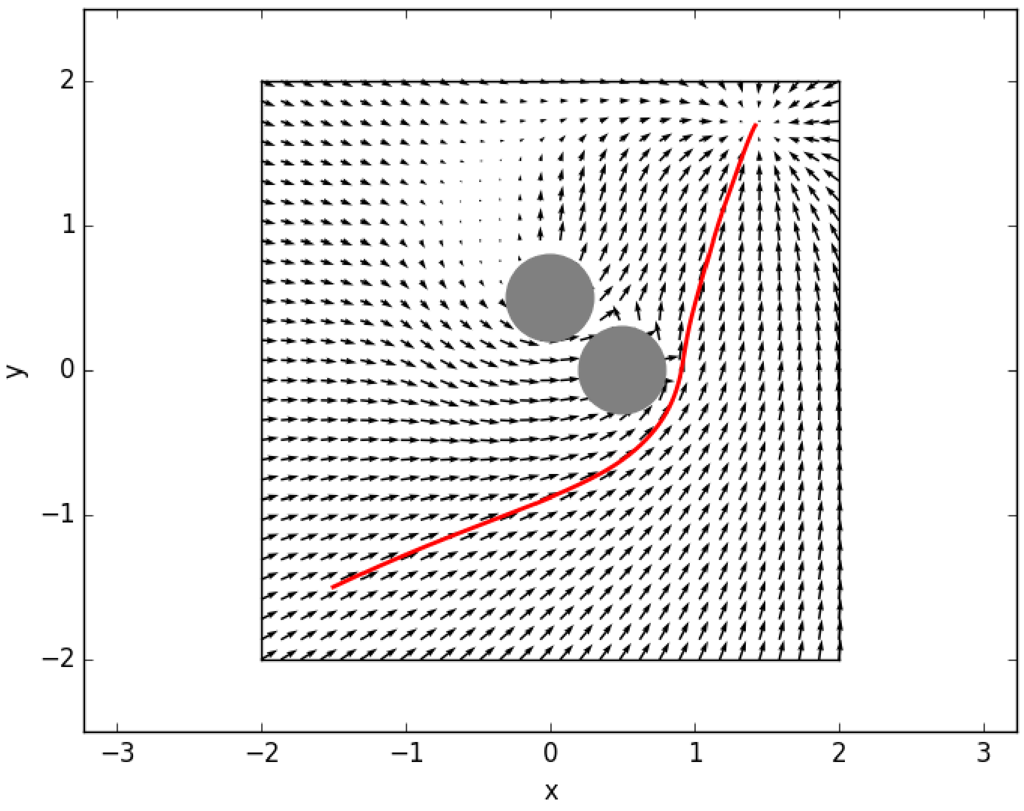


Figure 6: Potential field with two obstacles at (0,0.5) and (0.5,0)

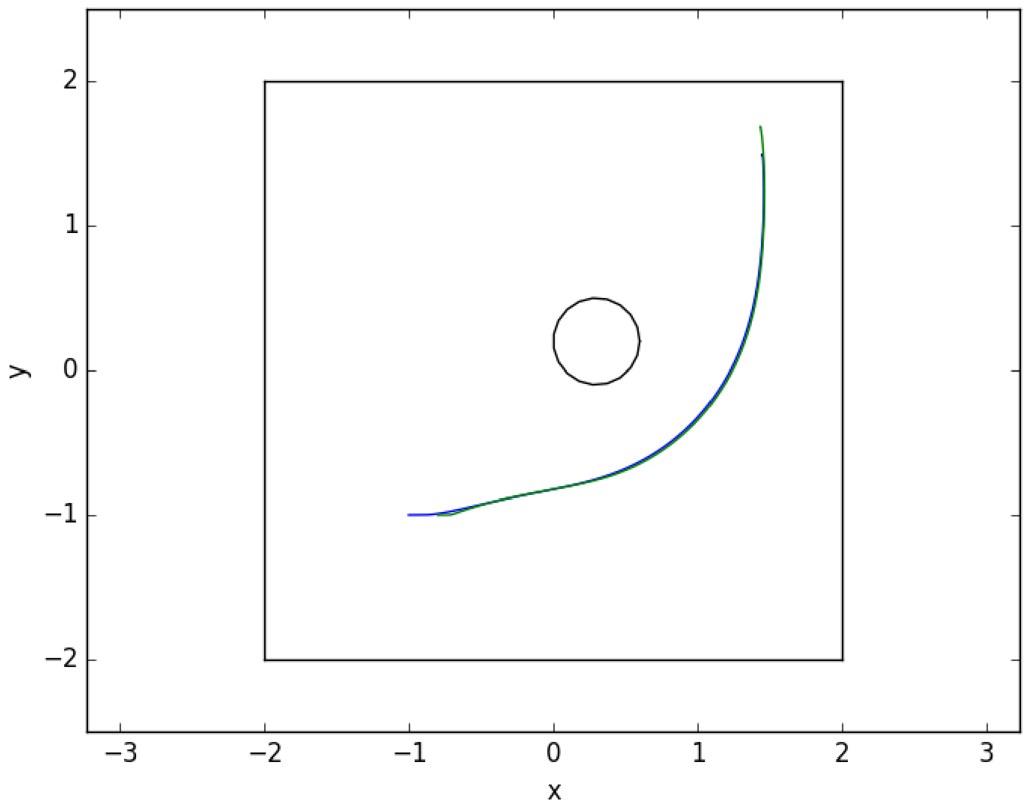


Figure 7: Trajectory of robot in potential field

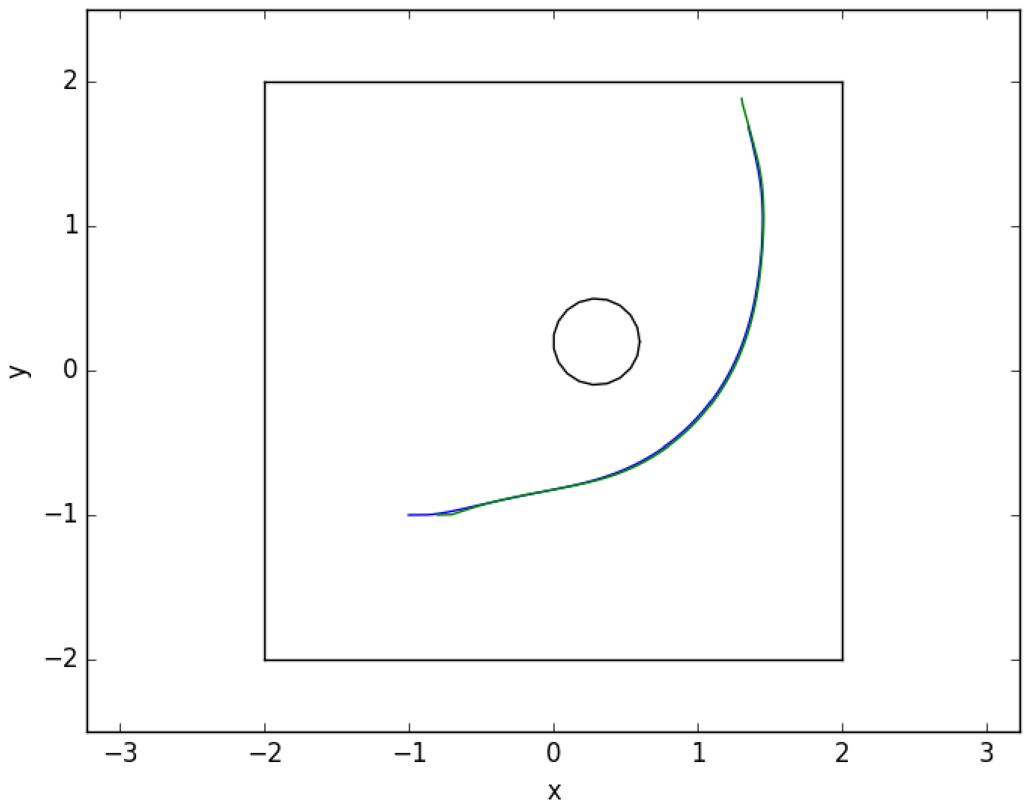


Figure 8: Trajectory of robot in potential field, using relative position

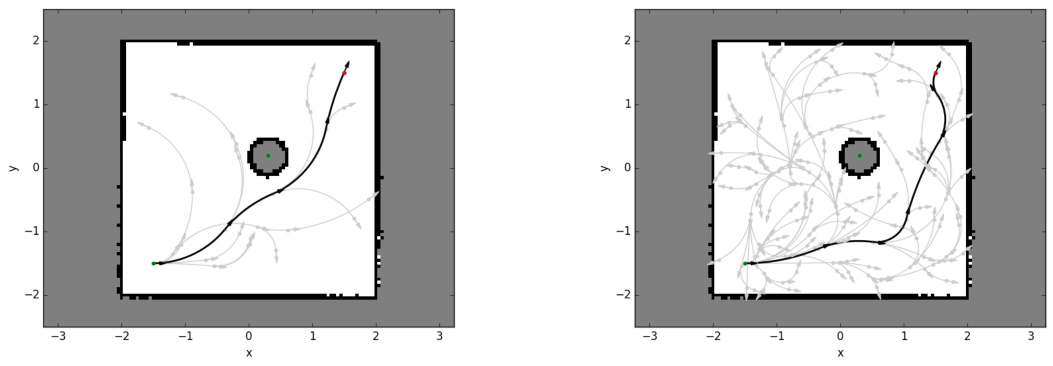


Figure 9: Paths generated by RRT

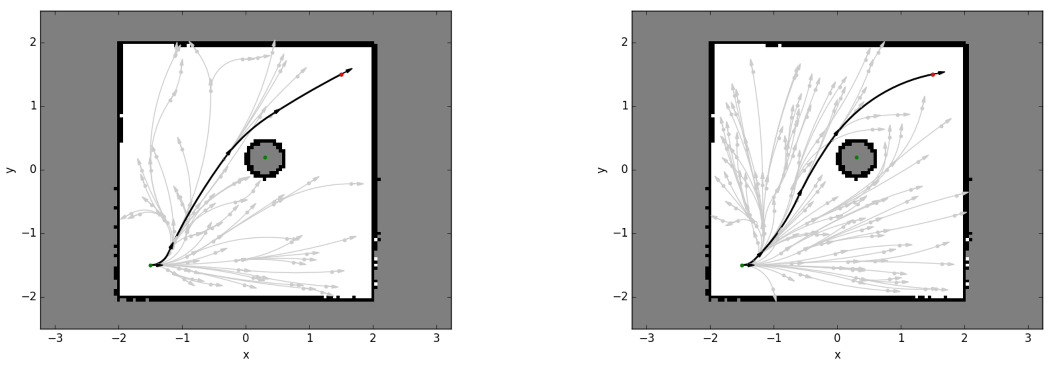
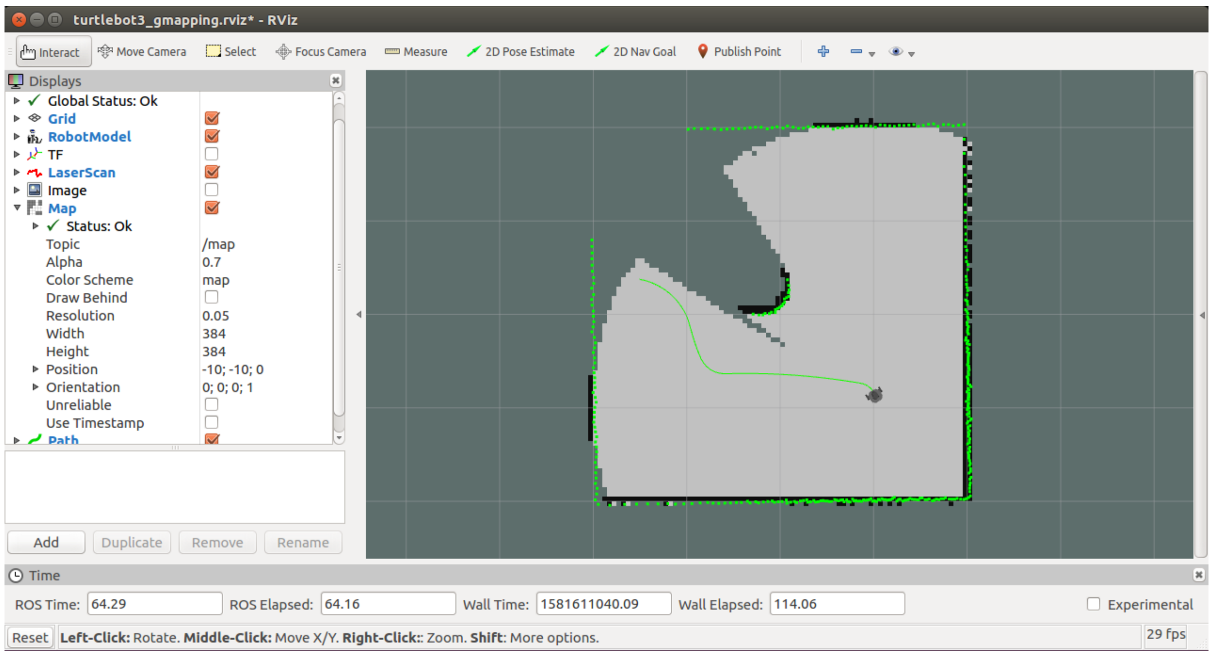


Figure 10: Paths generated by RRT*



Global Status: Ok
Grid ✓
RobotModel ✓
TF
LaserScan ✓
Image
Map ✓
Status: Ok
Topic /map
Alpha 0.7
Color Scheme map
Draw Behind
Resolution 0.05
Width 384
Height 384
Position -10; -10; 0
Orientation 0; 0; 0; 1
Unreliable
Use Timestamp
Path ✓

ROS Time: 64.29 ROS Elapsed: 64.16 Wall Time: 1581611040.09 Wall Elapsed: 114.06 Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click:**: Zoom. **Shift**: More options.

Figure 11: RViz window showing RRT with SLAM