

# Mobile Robotic System Practical 2

Elim Yi Lam Kwan (ylk25)

## 1 Exercise 1

### 1.1 Part (a)

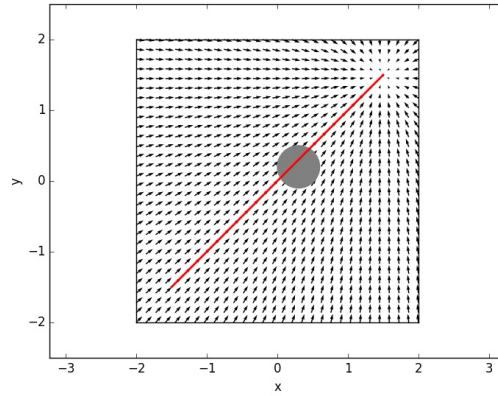


Figure 1: Attractive Velocity Field towards the Goal

The velocity field (attractive force  $\vec{F}_a$ ) to reach the goal equals to the negative gradient of the attractive potential field  $\nabla U_a$ . It was formulated as:

$$\vec{F}_a(x, y) = -\nabla U_a = \frac{\rho_a(x, y)}{\|\rho_a(x, y)\|} G_a(\|\rho_a(x, y)\|) \quad (1)$$

$$\text{with } \rho(x, y) = \begin{bmatrix} x_{goal} - x \\ y_{goal} - y \end{bmatrix}$$

$$G_a(\rho) = \begin{cases} \frac{\text{Max}_a}{1 + e^{\frac{1}{\alpha}(-(\rho - \beta))}} & , \text{ for } \frac{\text{Max}_a}{1 + e^{\frac{1}{\alpha}(-(\rho - \beta))}} \geq 0.1 \\ 0.1 & , \text{ else} \end{cases} \quad (2)$$

where  $\rho_a$ ,  $G_a(\rho)$  were the directional velocity vector and the scaling function respectively.  $G_a(\rho)$  was clipped to a lower bound of 0.1.  $x_{goal}, y_{goal}$  were the coordinates of the goal positions.  $\text{Max}_a$  corresponded to the maximum attainable scale of the attractive velocity, and was set as 0.75.  $\alpha$  represented the radius of the impact and  $\beta$  represented the offset applied to the horizontal axis of  $G_a(\rho)$ .

We first determined the direction of the velocity vector  $\rho_a$  by calculating the relative location of the current position with respect to the goal position. Moreover, we re-scaled it as a unit vector. The magnitude of the final velocity at  $(x, y)$  was then determined by scaling the magnitude of  $\rho_a$  with a Sigmoid function  $G_a(\rho)$  with appropriate offset. This is because the magnitude of the velocity vectors should decrease as it approaches the goal. The goal position should have the lowest potential on the map. Furthermore, we would like to ensure a minimum attractive force towards the goal at all locations of the map, hence  $G_a(\rho)$  was clipped to have a minimum value of 0.1. These have been reflected in Figure 1 with the increasing small arrows around the goal position (1.5, 1.5). Sigmoid was used to

smoothen the rate of decrease in velocity, which prevented jittering motions of the robot. Offset  $\alpha$  determined the range at which the velocity starts to decrease, while  $\beta$  shifted the Sigmoid function to the right horizontally. They were determined as 0.2 and 0.5 based on empirical experiments.

## 1.2 Part (b)

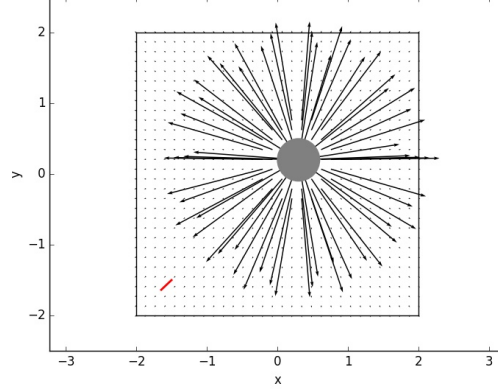


Figure 2: Repulsive Velocity Field from the Obstacle

The velocity field (repulsive force  $F_r$ ) to repel the robot from the obstacle equals to the negative gradient of the repulsive potential field  $\nabla U_r$ . It was formulated as:

$$\vec{F}_r(x, y) = -\nabla U_r = \frac{\rho_r(x, y)}{\|\rho_r(x, y)\|} G_r(\|\rho_r(x, y)\|) \quad (3)$$

$$\text{with } \rho_r(x, y) = \begin{bmatrix} x - x_{obstacle} \\ y - y_{obstacle} \end{bmatrix}$$

$$G_r(\rho) = \begin{cases} \frac{\text{Max}_r}{1 + e^{\frac{1}{r}(\rho - \beta)}} & , \text{ if } \rho \leq r \\ 0 & , \text{ else} \end{cases}$$

where  $\rho_r$ ,  $G_r(\rho)$  were the directional vector and the scaling function.  $x_{obstacle}, y_{obstacle}$  were the coordinates of the centre of the obstacle.  $\text{Max}_r$  corresponded to the maximum attainable scale of the repulsive force, and was set as 1.  $r$  represented the range of the repulsive force and was set as the radius of the obstacles plus an arbitrary offset.  $\beta$  represented the offset applied to the horizontal axis of  $G_r(\rho)$ .

A similar navigation function  $G_r(\rho)$  to Part(a) was applied. The major difference is  $G_r(\rho)$  is the inverse of a normal Sigmoid function, while  $G_a(\rho)$  is an ordinary Sigmoid function. This is the case because when the distance between the robot and the obstacle reduces, the repulsive force should increase. Moreover, when the robot is far away from the obstacle, the repulsive force should tend to 0. The centre of the obstacle should have the highest potential on the map. This was reflected in Figure 2.

Nevertheless,  $G_r(\rho)$  and  $G_a(\rho)$  were similar because attractive and repulsive forces should be within the similar numerical range. One point to note is that repulsive force should be more dominant than

the attractive force when the robot is close to the obstacle, therefore:

$$(\text{Max}_r = 1) > (\text{Max}_a = 0.75) \quad (4)$$

### 1.3 Part (c)

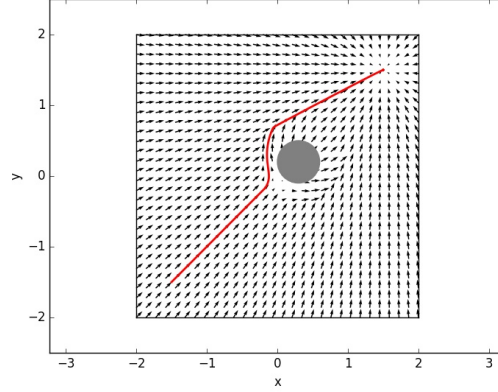


Figure 3: Resultant Velocity Field with Obstacle Avoidance

The resultant velocity field combined the attractive and repulsive forces.

$$\begin{aligned} \vec{F}(x, y) &= \vec{F}_a(x, y) + \vec{F}_r(x, y) \\ \text{where } \|\vec{F}(x, y)\| &< \text{Max Speed} \end{aligned} \quad (5)$$

As demonstrated by the red trajectories in Figure 3, the robot successfully avoided the obstacle and arrived at the goal location following the guidance from  $\vec{F}(x, y)$ . The robot initially travelled towards the goal in a straight line following  $\vec{F}_a(x, y)$ . Around the obstacles, the resultant field  $\vec{F}(x, y)$  pointed side-way under the influence of the repulsive force, which allowed the robot to avoid the obstacle. Finally, the dip in potential around the goal position allowed the robot to slow down and reach the goal. Nevertheless, the resultant magnitude of  $\vec{F}(x, y)$  was clipped to 0.5, which was the maximum speed allowed by the robot.

### 1.4 Part (d)

Figure 4 showcased one of the disadvantages of the potential field methods - the robot might get stuck at the local minima. When the obstacle was placed at the direct path towards the goal, the attractive and repulsive forces were exactly opposite and they cancelled out each other. The robot would then stop at the local minima. In these cases, stochasticity could be introduced to alleviate the problem. With added noises, neighbouring vectors are unlikely to cancel out each other, thus, local minima could be avoided.

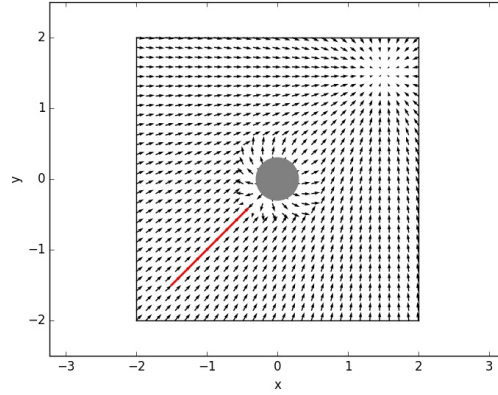


Figure 4: Disadvantage of Potential Field Methods - Local Minima

### 1.5 Part (e)

Our solution to the local minima problem was to introduce Gaussian noise to the resultant vector field  $\vec{F}(x, y)$ , such that:

$$\vec{F}(x, y) = \text{Norm}(\vec{F}(x, y), \sigma) \quad (6)$$

where  $\sigma$  corresponded to the covariance matrix of the distribution and was set to 0.2 after empirical experiments.

As shown in Figure 5, neighbouring vectors no longer pointed in the same direction locally. Moreover, in the overall formation, they still followed the global flow because the magnitudes of the noises added were small. This allowed the robot to avoid the local minima shown in Figure 4 while travelling towards the goal.

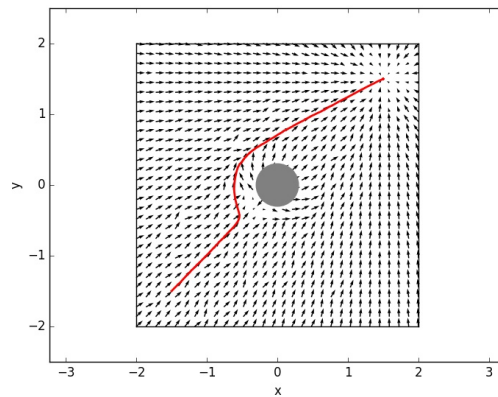


Figure 5: Tackle Local Minima Cases with Stochastic Algorithm

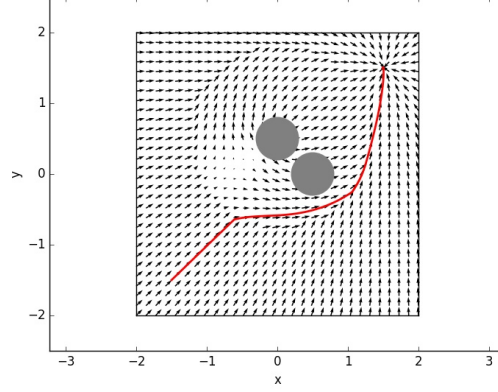


Figure 6: Tackle Closely Placed Obstacles Cases with additional Virtual Obstacle

## 1.6 Part (f)

In the case of closely placed obstacles, relying on the stochastic algorithms alone did not work. Closely placed obstacles generated a concave-like environment, which resulted in the repulsive vector fields from the obstacles guided the robot towards the opening of the concave blockage. The robot would get stuck at the local minima between the two obstacles when the repulsive force cancelled out each other. The stochastic algorithm failed in these cases because the global flow also failed. Therefore, we have derived an improved solution. There were two phases: local minima detection and insert virtual obstacle at the local minima(s). The idea of the virtual obstacle was inspired by the research of M. G. Park and M. C. Lee [1].

The robot is at a local minima when the resultant force  $\vec{F}(x, y)$  is close to 0, but it is not at the goal position:

$$(x, y) \text{ is a local minima if } \begin{cases} \vec{F}(x, y) < \text{threshold} \\ x, y \neq x_{goal}, y_{goal} \end{cases} \quad (7)$$

The threshold was set as 0.05 upon inspecting the vector field at the minima. When local minima was detected at  $(x, y)$ , a virtual obstacle with a small radius (e.g.,  $\frac{1}{3}$  of the cylinder radius) would be inserted at the  $(x + \epsilon, y + \epsilon)$ , where  $\epsilon$  is a very small number, e.g., 0.001. Then, the vector field  $\vec{F}(x, y)$  was re-generated and it would equate to:

$$\begin{aligned} \vec{F}(x, y) &= \vec{F}_a(x, y) + \vec{F}_{r1}(x, y) + \vec{F}_{r2}(x, y) + \vec{F}_{virtual}(x + \epsilon, y + \epsilon) \\ \vec{F}(x, y) &= \text{Norm}(\vec{F}(x, y), \sigma) \end{aligned} \quad (8)$$

where  $\vec{F}_{r1}(x, y)$ ,  $\vec{F}_{r2}(x, y)$ ,  $\vec{F}_{virtual}(x + \epsilon, y + \epsilon)$  represented the repulsive forces from the two physical obstacles and the additional virtual obstacle. Note that  $\epsilon$  was required to avoid division by zero error in Equation 3. Figure 6 shows that our implementation successfully steered the robot away from the opening of the concave environment. Moreover, we acknowledge that our method relied on successful local minima detection, which might not be the case if the robot oscillate at the the local minima. Hence, future work could focus on developing more robust local minima detection techniques.

## 2 Exercise 2

### 2.1 Part (a)

The set of equations that provide feedback linearization for a differential drive robot by controlling a holonomic feedback point at a distance  $\epsilon$  from the robot are:

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \end{bmatrix} = u \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix} + \epsilon \omega \begin{bmatrix} -\sin \theta \\ \cos \theta \end{bmatrix} \quad (9)$$

The  $\dot{x}_p, \dot{y}_p$  were equivalent to the current velocity in the x and y directions respectively.  $\theta$  was the angular velocity of the current pose. Moreover, the control outputs (output velocity  $u$  and output angular velocity  $\omega$ ) could be formulated as a function of  $\dot{x}_p, \dot{y}_p$  as shown below:

$$u = \dot{x}_p \cos \theta + \dot{y}_p \sin \theta \quad (10)$$

$$\omega = \epsilon^{-1}(-\dot{x}_p \sin \theta + \dot{y}_p \cos \theta) \quad (11)$$

### 2.2 Part (b)

The set of equations that control a non-holonomic robot is non-linear because its controllable degree of freedom is less than the total degrees of freedom. By assuming the robot was controlled by a holonomic feedback point at a distance  $\epsilon$  from the robot allowed us to translate the non-linear system to an equivalent linear system through a change of variables. This enabled us to apply feedback control techniques to a holonomic point.

### 2.3 Part (c)

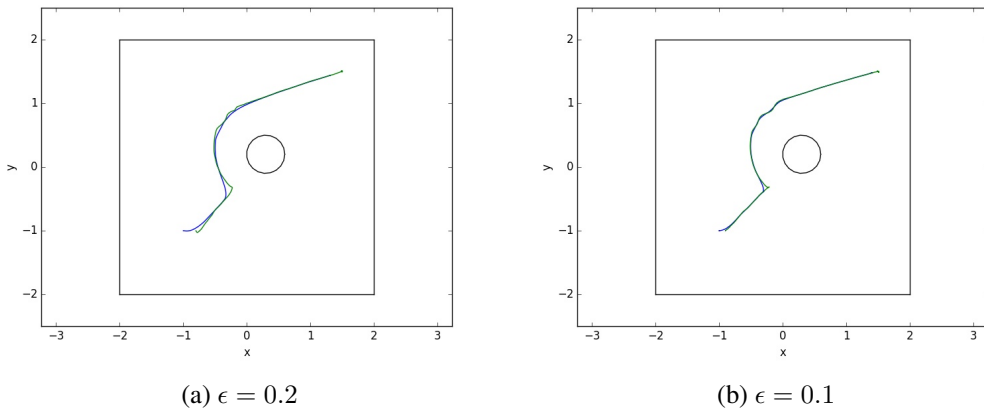


Figure 7: Velocity Tracking using Absolute Positions. Figures show the Linearized (Green) and the True (Blue) Trajectories.

As shown in Figure 7, the robot followed the trajectories similar to that of Figure 3. Moreover, we observed that the linearized trajectories tended to overshoot when the robot was making sharp turns. This could be understood as we assumed the robot is being pulled by a holonomic point in front of it and motion controls applied would not be fully reflected on the robot.

We also compared the difference in trajectories when  $\epsilon$  was reduced. As shown in Figure 7b, there were less overshoot. This is expected as the virtual holonomic point is now closer to the robot, hence the robot would response more swiftly to the motion controls applied.

## 2.4 Part (d)

The equations in Part 2(c) did not rely on the absolute pose of the robot. Moreover, in experiments with physical robots, obstacles were detected by the robots and translated to the system as relative positions with respect to the robots' locations. Hence, adopting relative positions better resembled real-world constraints. The y-axis of the robot coordinate system would be the direction vector from the centre of the robot to the holonomic point, and the x-axis was perpendicular to that. To translate a point  $(x_{abs}, y_{abs})$  in the world coordinate to a relative position  $(x_{relative}, y_{relative})$ , scaling and rotation would be required:

Scaling:

$$\begin{aligned} scale_x &= x_{abs} - x_{abs\_robot} \\ scale_y &= y_{abs} - y_{abs\_robot} \end{aligned} \quad (12)$$

Rotation:

$$\begin{bmatrix} x_{relative} \\ y_{relative} \end{bmatrix} = \begin{bmatrix} \cos k & -\sin k \\ \sin k & \cos k \end{bmatrix} \begin{bmatrix} scale_x \\ scale_y \end{bmatrix} \quad (13)$$

where  $k = -\theta_{abs\_robot}$  and  $(x_{abs}, y_{abs}, \theta_{abs\_robot})$  was the robot's pose in the world coordinate.

Figure 8 shows the results of using relative positions instead absolute position. The trajectories obtained was similar to 7a as expected.

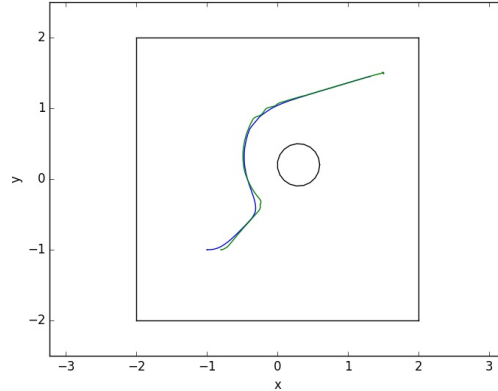


Figure 8: Velocity Tracking using Relative Positions. Figure shows the Linearized (Green) and the True (Blue) Trajectories.

### 3 Part 3

#### 3.1 Part (a)

Random positions were first obtained by Equation 14:

$$\text{Position}_{i \sim x, y} = \begin{cases} (+1) \times X \times (\text{Dimension}_i) & \text{for } S > 0.5 \\ (-1) \times X \times (\text{Dimension}_i) & \text{for } S \leq 0.5 \end{cases} \quad (14)$$

where  $S, X \sim \mathcal{U}(0, 1)$

For each position coordinate, we sampled 2 random variables  $S, X$  from the uniform distribution.  $S$  determined the sign and  $X$  scaled the magnitude.  $\text{Dimension}_i$  corresponds to the width and height of the bounding box. Finally, we validated the random generated positions by cross-checking it with the `occupancy_grid · is_free(x, y)` function. We generated positions  $(x, y)$  until a valid one is found.

The valid random positions would form new nodes in the Rapidly-exploring Random Tree. Moreover, uniform distribution was used as it tends to sample points from unexplored space. This allowed us to search the space more efficiently.

#### 3.2 Part (b)

Firstly, angular velocity at starting position  $\theta_{p_{start}}$  could be translated to a tangent line. Its gradient  $m_{start}$  and y-intercept  $c_{start}$  would be:

$$m_{start} = \tan(\theta_{p_{start}}) \quad (15)$$

$$c_{start} = \theta_{p_{start}} - \theta_{p_{start}} * m \quad (16)$$

Secondly, given two points  $p_{start}, p_{final}$ , there are a family of circle that pass through them. With the additional information of the angular velocity at  $p_{start}$ , we have 3 pieces of information (points location and slope at  $p_{start}$ ) to solve for the 3 unknowns (circle center  $(h, k)$  and radius  $r$ ). From the circle equations:

$$\begin{cases} (x_{start} - h)^2 + (y_{start} - k)^2 = r^2 \end{cases} \quad (17a)$$

$$\begin{cases} (x_{final} - h)^2 + (y_{final} - k)^2 = r^2 \end{cases} \quad (17b)$$

$$\begin{cases} \frac{(y_{start} - k)}{(x_{start} - h)} = \frac{-1}{m_{start}} \end{cases} \quad (17c)$$

Equation 17a minus Equation 17b equals to:

$$h = \frac{x_{start}^2 - x_{final}^2 + y_{start}^2 - y_{final}^2 + 2(y_{final} - y_{start})k}{2(x_{start} - x_{final})}$$

Moreover, from Equation 17c,  $k = y_{start} + \frac{x_{start} - h}{m_{start}}$ , and substituting it to the above results in:

$$h = \frac{x_{start}^2 - x_{final}^2 + y_{start}^2 - y_{final}^2 + 2(y_{final} - y_{start})(y_{start} + \frac{x_{start} - h}{m_{start}})}{2(x_{start} - x_{final}) + 2(y_{final} - y_{start})(\frac{1}{m_{start}})} \quad (18)$$



Therefore,  $h, k, r$  could be obtained with Equation 18, 17c and 17a respectively. Then, a preliminary  $\theta_{final}$  could be obtained as:

$$m_{final} = \frac{(h - x_{final})}{(y_{final} - k)} \quad (19)$$

$$\theta_{final} = \arctan(m_{final}) \quad (20)$$

The final piece of information to determine  $\theta_{final}$  is to check whether  $\theta_{final}$  travelled in the same direction as  $\theta_{start}$ . Normally, they should be in the same direction, with the exception  $p_{start}, p_{final}$  formed the diameter of the circle. Hence, cosine similarity check was used:

$$d = \theta_{final} \cdot \theta_{final} \quad (21)$$

$$\theta_{final} \begin{cases} = \theta_{final} + \pi & , \text{ for } d < 0 \text{ and } Distance(p_{start}, p_{final}) \neq 2r \\ = \theta_{final} & , \text{ else} \end{cases} \quad (22)$$

Finally, object collision was conducted by discretizing the calculated arc into an arbitrary number of points, and cross-check with `occupancy_grid.is_occupied(x, y)`. If occupied, the function return *None*.

Figure 9 shows the resultant trajectories using RRT. It demonstrated that `adjust_pose()` was functioning properly, with the resultant arrows pointing in sensible directions and none of the arcs passed through the obstacle.

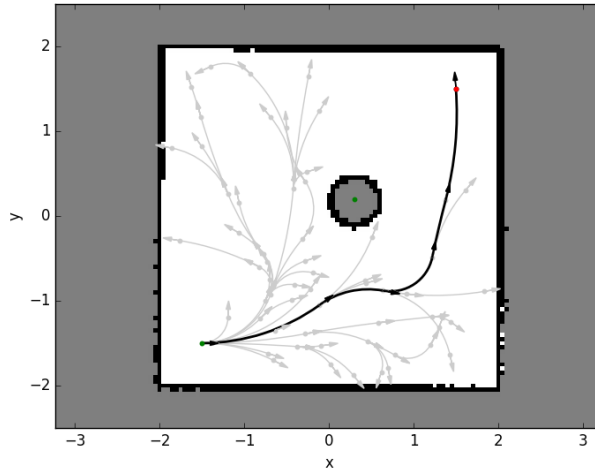


Figure 9: Trajectories obtained with RRT

### 3.3 Part (c)

In RRT, we first sampled a random position and connected it to the nearest existing tree node given that it satisfied the motion primitives and collision check. The tree would tend to expand to unexplored areas due to random sampling and would stop once the goal location is found.

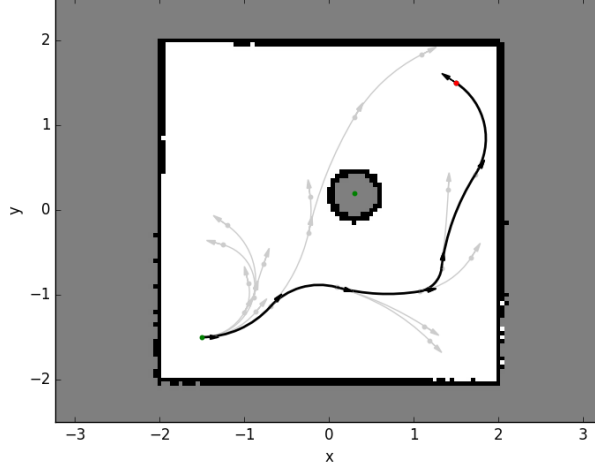


Figure 10: RRT forms non optimal cubic graphs

Although RRT is probabilistically complete <sup>1</sup>, it usually generates non-optimised path as shown in Figure 10. This is because the existing state graph biases future expansion. Moreover, another observation is RRT often requires a lot of iterations (400 to 700) to find the solution.

To ensure asymptotic optimality <sup>2</sup>, a potential solution would be to adopt RRT\* [2]. In RRT\*, each node has an additional cost attribute that records the distance each vertex has travelled relative to the starting location. Furthermore, each newly appended node would be considered as potential parents for its neighbouring nodes. If it forms a path with a lower cost compared to the existing parent, the new node would replace the parent. This rewiring process allowed us to optimise the motion plan for minimising travelled distance while planning. It seems to be a promising solution for overcoming the drawbacks identified.

### 3.4 Part (d)

In our implementation of RRT\*, the cost attribute in node  $v$  with parent  $u$  was defined as:

$$v_{cost} = u_{cost} + Arc_{u \text{ to } v} \quad (23)$$

where  $Arc_{u \text{ to } v}$  was the arc length between  $u$  and  $v$ . It could be determined with trigonometry given the connected circle centre  $(h, k)$  and radius  $r$ :

$$\theta_1 = \arctan\left(\frac{x_u - h}{y_u - k}\right) \quad (24)$$

$$\theta_2 = \arctan\left(\frac{x_v - h}{y_v - k}\right) \quad (25)$$

$$(26)$$

<sup>1</sup>Given a solvable problem, the probability that the planner solves the problem tends to one as the running time tends to infinity.

<sup>2</sup>For large inputs it performs at worst a constant factor worse than the best possible algorithm

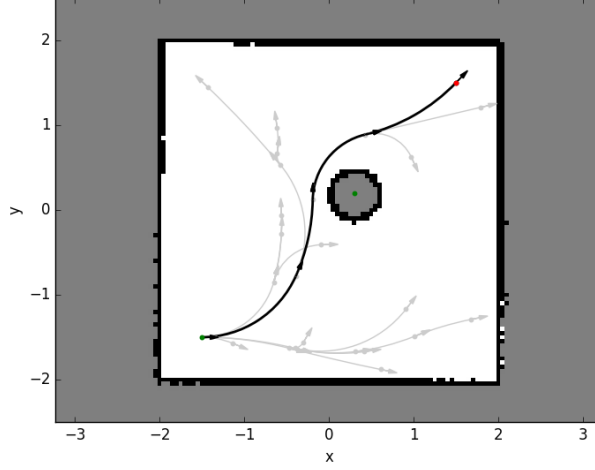


Figure 11: Trajectories obtained with RRT\*

If the cross product of angular velocity  $\theta_u$  and direction vector (u.position - center) was larger than 0, the arc should be clockwise and  $\theta_{a1}, \theta_{a2}$  would be swapped. Then,

$$Arc_{u \text{ to } v} = 2\pi r \times \frac{\theta_2 - \theta_1}{2\pi} \quad (27)$$

Different to RRT, which only considered Euclidean distance, we appended randomly sampled position  $v$  to node with the lowest cost (Equation 23) that lies within a certain range (e.g.,  $0.1 < \text{Euclidean Distance} < 0.8$ ). After that, we gathered all the vertices that lie within the search radius from  $v$ . The rewiring procedure is formulated in Algorithm 1. Note that, during the rewiring process, we had to validate whether the new angular velocity is similar to the existing one. If the neighbour has existing children, one should not modify its pose angle.

---

**Algorithm 1:** RRT\* Rewiring

---

```

for each Neighbors Node  $n$  do
  if  $cost(n, v) < n.cost$  then
     $Node_{v \text{ to } n} = adjust\_pose()$ 
    if  $n$  has no children then
       $n.pose = Node_{v \text{ to } n}.pose$ 
       $n.parent = v$ 
       $v.add\_children(n)$ 
    end
    else if  $n.pose \approx Node_{v \text{ to } n}.pose$  then
       $n.parent = v$ 
       $v.add\_children(n)$ 
    end
  end
end

```

---

Figure 11 shows the trajectories obtained with RRT\*. Compared to RRT trajectories in Figure 10 and 9, the motion planner found a more direct path, which had a shorter length and was clipped closer

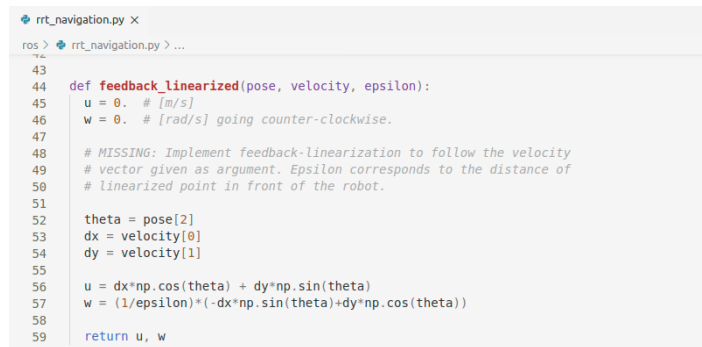
to the obstacle. The figure also shows that arcs tended to branch off from the node(s) at the starting area. This is because the planner tried to minimise the total travelled distance. Furthermore, we also observed that RRT\* often solved the problem with fewer iterations.

The disadvantages of RRT\* lies in the additional complexity and scalability. If the motion primitive function is non-deterministic (e.g. grid-search) and the collision checker is expensive, it would slow down the algorithm. Moreover, the complexity of the algorithms increases significantly as the tree grows.

## 4 Exercise 4

### 4.1 Part (a)

Feedback linearization simplified the robot to a holonomic point. This enabled us to perform motion planning and feedback control on a holonomic point in Part 4(c). Figure 12 shows the working solution of it.



```

43
44 def feedback_linearized(pose, velocity, epsilon):
45     u = 0. # [m/s]
46     w = 0. # [rad/s] going counter-clockwise.
47
48     # MISSING: Implement feedback-linearization to follow the velocity
49     # vector given as argument. Epsilon corresponds to the distance of
50     # linearized point in front of the robot.
51
52     theta = pose[2]
53     dx = velocity[0]
54     dy = velocity[1]
55
56     u = dx*np.cos(theta) + dy*np.sin(theta)
57     w = (1/epsilon)*(-dx*np.sin(theta)+dy*np.cos(theta))
58
59     return u, w

```

Figure 12: Feedback Linearization for Exercise 4(a)

### 4.2 Part (b)

The disadvantages of using motion primitives instead of straight lines were the increased computational complexity. Also, the resultant paths might be less efficient. If straight lines were used to connect vertices, only the collision checker is required. Moreover, in our case, the connected circle and corresponding angular velocity have to be computed as well. Although these are deterministic equations, it still adds complexity. On top of that, the motion primitive imposed difficulties in obtaining a valid pose, hence more iterations may be required by the planner. Secondly, the travelled distance on a straight line is shorter than that on an arc, hence, enforcing motion primitive may induce unnecessary distance cost.

The advantage of motion primitive is smoother trajectories, which resemble the motion of a real robot more accurately. A physical robot is unlikely to turn in sharp angles, if motion primitives were not adopted, the robot is more likely to fall out from the simulated trajectories, which increases localisation errors. Hence, drafting a smoother motion plan helps alleviate sim-to-real challenges.

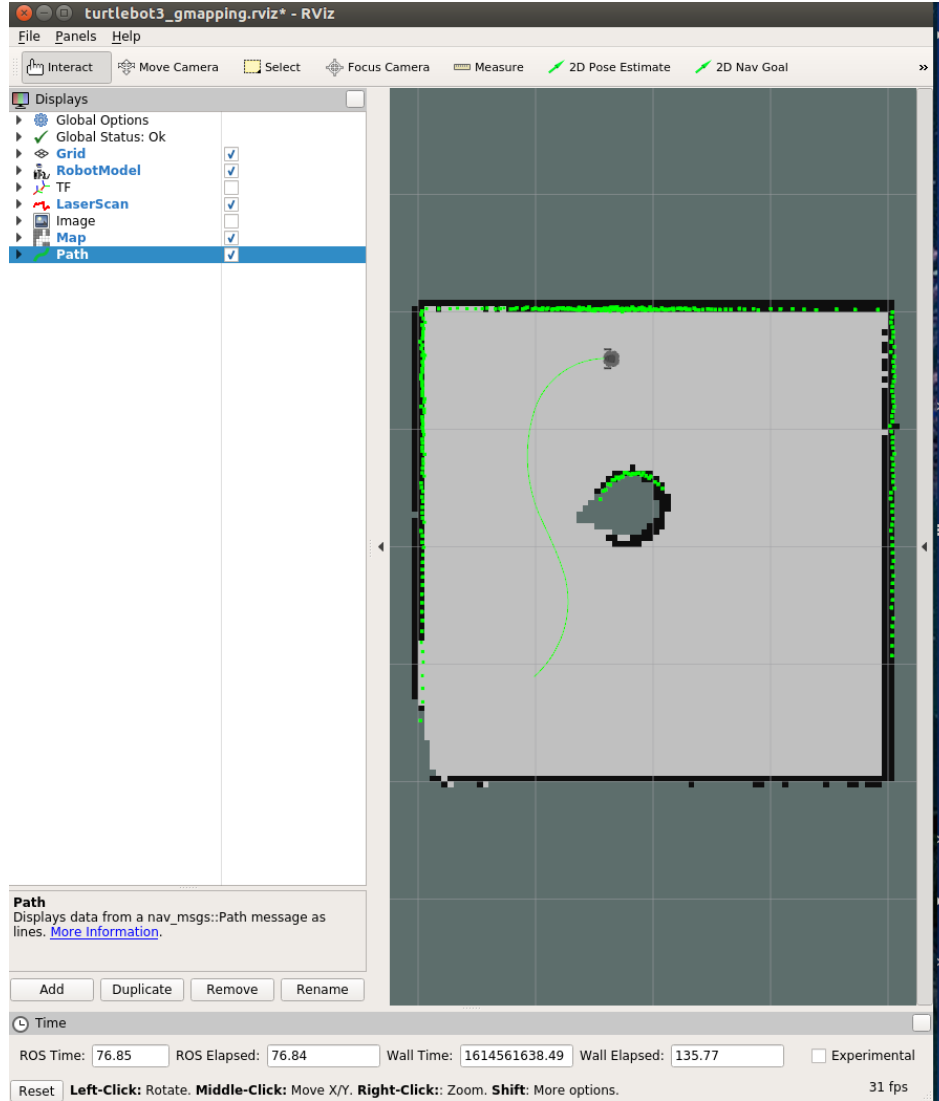


Figure 13: RViz window that shows the robot travelling towards its goal.

### 4.3 Part (c)

Path points were updated by the RRT controller every 2 seconds. When the plan was not being updated, the robot was expected to follow the trajectories defined by the path points. Moreover, the measured position of the robot might not be accurate when the plan is ready, because time is required to compute the plan and the robot already moved. Hence, the first part of the function is localisation. We determined the path point that was closest to the measured position  $position_{measured}$ , and assumed that this is the current position of the robot. After that, we generated the velocity vector such that it guided the robot towards the next point in the path point array, i.e. given the closest path point to the robot has index  $i_{closest}$ ,  $velocity = path\_points[i_{closest} + 1] - path\_points[i_{closest}]$

Secondly, the robot might show unpredictable motion when waiting for plans being updated. If the new plan is in the opposite direction of the previous plan, and the robot already followed the past plan and travelled to another direction before receiving the new instruction, it would turn in circles. These

phenomenons were caused by the delay in the action-perception loop and could be alleviated with appropriate feedback control on speed. We quantified uncertainty with the localisation error  $e_{loc}$ :

$$e_{loc} = \text{Distance}(\text{path\_points}[i_{closest}], \text{position}_{measured}) \quad (28)$$

Ideally, the higher the error, the slower the robot. Vice Versa. Hence, we defined a scaling factor  $s$  with the Sigmoid Function:

$$s = \frac{2}{1 + e^{\frac{1}{0.3}(e_{loc}-0.6)}} \quad (29)$$

Moreover, we also clipped the robot's final speed to a threshold  $Max\_Speed$ , such that it is less likely to exhibit unwanted behaviours. Therefore, the final velocity  $v_{final}$  with close-loop speed control is represented mathematically as:

$$v_{drafted} = s(\text{path\_points}[i_{closest} + 1] - \text{path\_points}[i_{closest}])$$

$$v_{final} = \begin{cases} v_{drafted} & , \text{ for } v_{drafted} \leq Max\_Speed \\ Max\_Speed & , \text{ else } \end{cases} \quad (30)$$

Figure 13 shows the trajectory (in green) at a given time, and the robot was following it.

#### 4.4 Part (d)

The slam.launch file launched the slam\_gmapping node and specified the hyper-parameters of the SLAM algorithms. It helps generate a 2-D occupancy grid map from the laser and pose data collected by the agent. GMap utilised the Rao-Blackwellized particle filter to determine the current position of the robot on the occupancy map. Through the SLAM node, the rrt\_navigate.py file could obtain the occupancy map and current position of the robot. Moreover, the RRT algorithm acted as our motion planner. Every 2 seconds, we queried the RRT algorithm with the robot's current position and occupancy map, and it would generate a suggested trajectory (in the format of a path point array) that leads to the goal. Furthermore, during that time interval, additional localisation was performed by mapping the robot to the closest path points based on its position data. Hence, the combination of slam.launch and RRT motion planner allowed us to achieve SLAM and motion planning, i.e. updating a map of an unknown environment while simultaneously keeping track of an agent's location and directing it towards the goal.

## References

- [1] Min Park and Min Cheol Lee. "A new technique to escape local minimum in artificial potential field based path planning". In: *Journal of Mechanical Science and Technology* 17 (Dec. 2003), pp. 1876–1885. DOI: 10.1007/BF02982426.
- [2] Sertac Karaman and Emilio Frazzoli. *Sampling-based Algorithms for Optimal Motion Planning*. 2011. arXiv: 1105.1186 [cs.RO].