Introduction to Robotics

Assignment 1

Jingyu Huang

Queens' College

Dynamical Systems and Motion Control

Exercise 1

(a) Differential equations for the motion of differential-wheeled robot:

$$\begin{cases} \dot{x} = u \cdot \cos\theta \\ \dot{y} = u \cdot \sin\theta \\ \dot{\theta} = \omega = \cos t \end{cases} \tag{1}$$

(b) Euler's Method is implemented as:

$$\begin{cases} x_{n+1} = x_n + dt \cdot \dot{x} = x_n + dt \cdot u \cdot \cos\theta \\ y_{n+1} = y_n + dt \cdot \dot{y} = y_n + dt \cdot u \cdot \sin\theta \\ \theta_{n+1} = \theta_n + dt \cdot \dot{\theta} = \theta_n + dt \cdot \cos t \end{cases} \tag{2}$$

Euler's method is a first order numerical procedure where the gradient multiplied by the size of the timestep is simply added to the current value to the the value at the next timestep.

(c) The plot of the approximation by Euler's method is shown as follows:

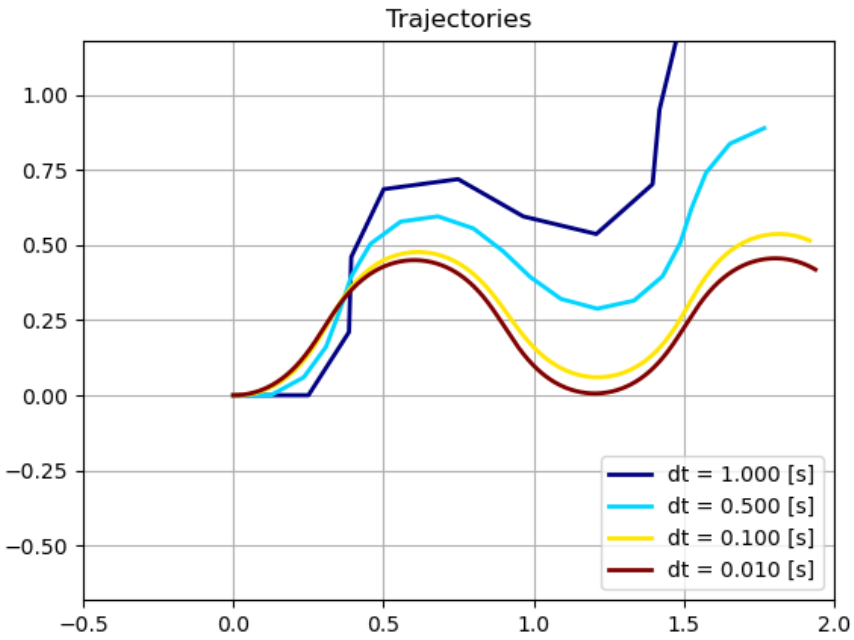

Figure 1: Plot of kinematics simulation using Euler's method.

At large timesteps, the program is run very quickly by as is obvious from the graph, the deviation from the actual value (a sinusoide) is also large. At small timesteps the program takes longer to run but the result is more accurate.

(d) RK4 is implemented as:

$$\begin{cases} k_{1x} = u \cdot \cos\theta \\ k_{1y} = u \cdot \sin\theta \\ k_{1\theta} = \cos t \\ k_{2x} = u \cdot \cos(\theta + k_{1\theta}/2) \\ k_{2y} = u \cdot \sin(\theta + k_{1\theta}/2) \\ k_{2\theta} = \cos(t + dt/2) \\ k_{3x} = u \cdot \cos(\theta + k_{2\theta}/2) \\ k_{3y} = u \cdot \sin(\theta + k_{2\theta}/2) \\ k_{3\theta} = \cos(t + dt/2) \\ k_{4x} = u \cdot \cos(\theta + k_{3\theta}) \\ k_{4y} = u \cdot \sin(\theta + k_{3\theta}) \\ k_{4\theta} = \cos(t + dt) \end{cases} \tag{3}$$

$$\begin{cases} x_{n+1} = x_n + 1/6 \cdot (k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x}) \\ y_{n+1} = y_n + 1/6 \cdot (k_{1y} + 2k_{2y} + 2k_{3y} + k_{4y}) \\ \theta_{n+1} = \theta_n + 1/6 \cdot (k_{1\theta} + 2k_{2\theta} + 2k_{3\theta} + k_{4\theta}) \end{cases} \tag{4}$$

RK4 is a fourth order numerical procedure where a weighted average of 4 increments is added the current value in order to calculate the value of the next timestep. Noteably the $k1$ part of RK4 is the same as Euler's method.

(e)

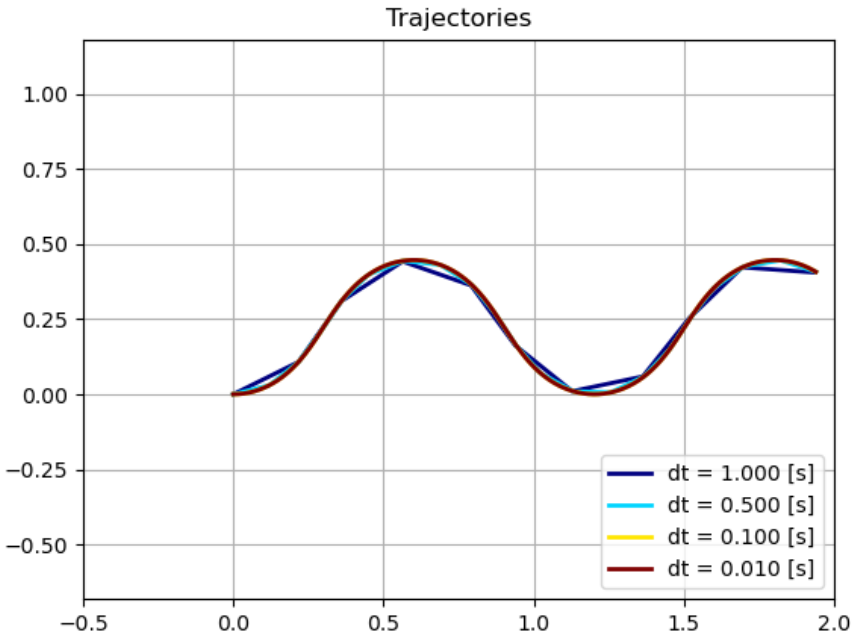

Figure 2: Plot of kinematics simulation using RK4.

As a fourth order solution, RK4 makes good approximation even at large timesteps as

error from previous steps do not accumulate as much as in Euler's method, due to the weighted average.

With very small timesteps, it might be preferable to use Euler's method as the two gives similar results while Euler's method involves less computation.

(f)

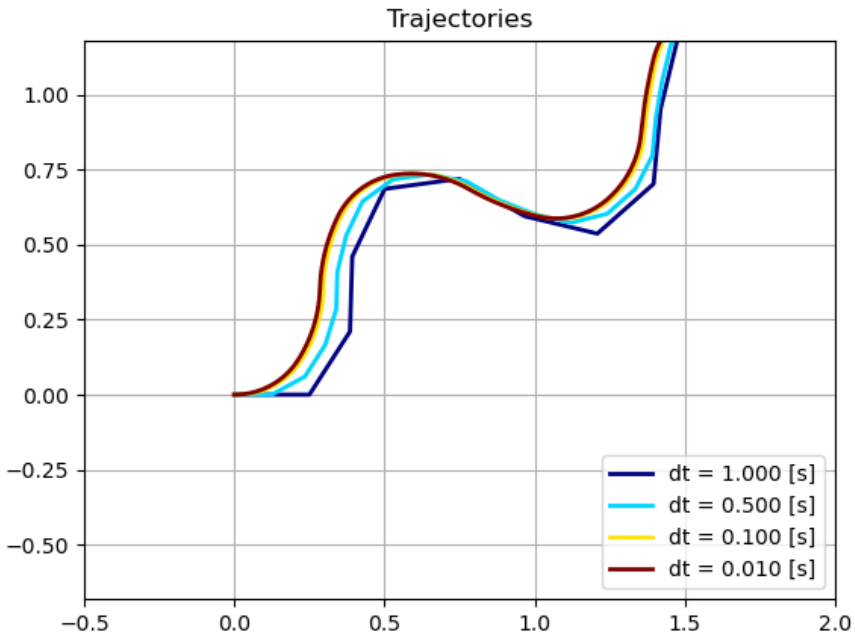

Figure 3: Plot of kinematics simulation using Euler's method with perception-action loop running at 1Hz.

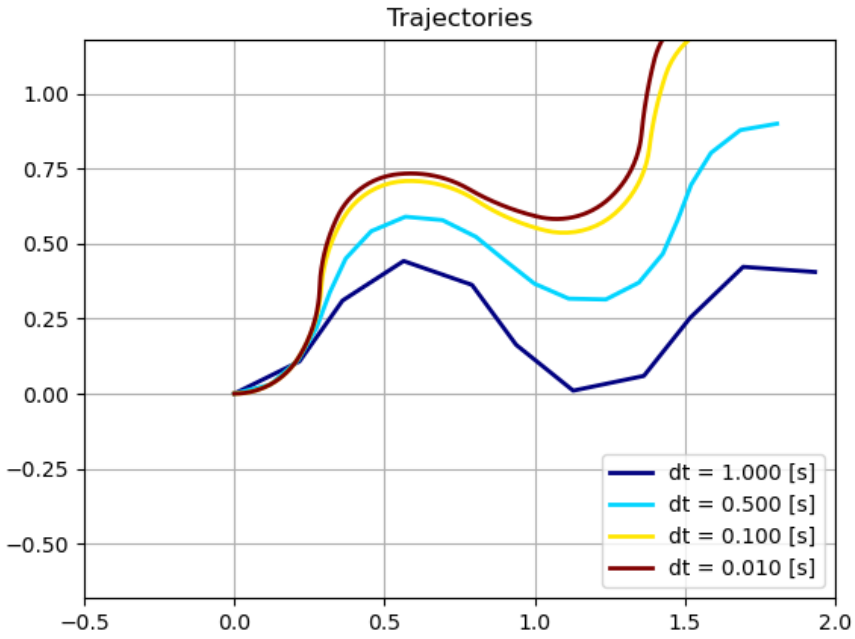

Figure 4: Plot of kinematics simulation using RK4 with perception-action loop running at 1Hz.

With a perception-action loop of 1Hz, Euler's method is stuck at the same state that $dt = 1.0s$ is in for all values of dt, due to increased error accumulation with increased number of timesteps. RK4 suffers more greatly from the increased error accumulation due to delayed perception-action loop. At $dt = 1.0s$, it performs reasonably well, but deviation increases as the timesteps grow smaller and the number of timesteps increases. At $dt = 0.010s$, The trajectory plotted by RK4 is virtually the same as the one from Euler's method.

(g) To resolve the issue due to the perception-action loop delay, an internal counter is created is keep track of time between the perception-action loop. The counter is incremented by dt at each timestep and reset every 1 second. ω is updated using the sum of perceived time and the counter time to better keep track of where it is heading. The result is as follows:

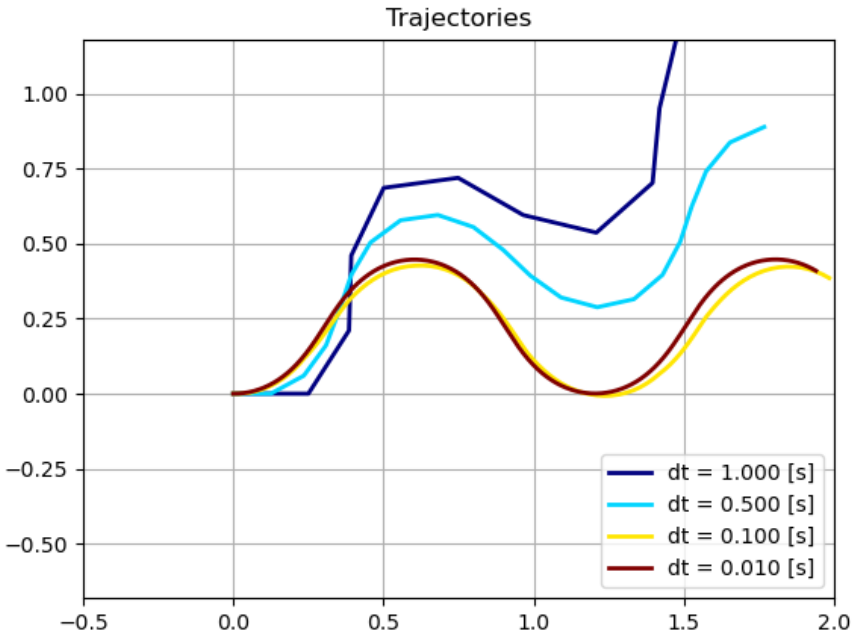

Figure 5: Plot of corrected kinematics simulation using Euler's method with perception-action loop running at 1Hz.

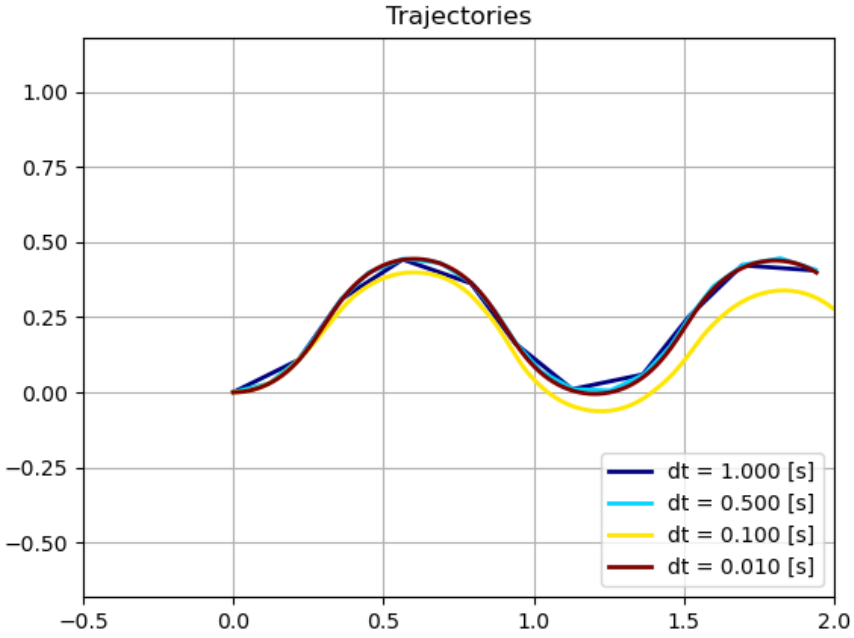

Figure 6: Plot of corrected kinematics simulation using RK4 with perception-action loop running at 1Hz.

As shown, both the Euler's method and RK4 are reasonably restored to their original performance. Interestingly, both exhibits slightly higher deviations at $dt = 0.10s$, which is rather unexpected. This can be due to floating point error or other errors inherent to

the numerical system, though it is unknown why the same behaviour is not exhibited in other timestep sizes.

Exercise 2

(a) The Braitenberg controller is implemented using a very simple linear setup. As the speed of each of the wheels are not directly accessible, an imitation of the 'explorer' archetype is created by controlling the forward velocity and the angular velocity.

Firstly, the room for maneuver is calculated for each of the five sensor directions by deducting the safety distance from the actual measured distances.

The forward velocity is scaled linearly with the room for maneuver in the forward direction divided by the safety distance. This way the robot slows down as it approaches an obstacle, giving it more time for turning away.

The angular velocity is scaled linearly with the difference between room for maneuver in $front\_left$ and $front\_right$ and, to a lesser degree, $left$ and $right$. This is so that the controller takes greater priority in avoiding obstacles closer to the from, in order to minimise the chance of running into them.

Finally, the forward velocity is clipped between 0.01 and max_u. This is to push the robot to keep moving, as the vanilla implementation of Braitenberg explorer will stop in front of an obstacle if running straight into it. The result is as follows:

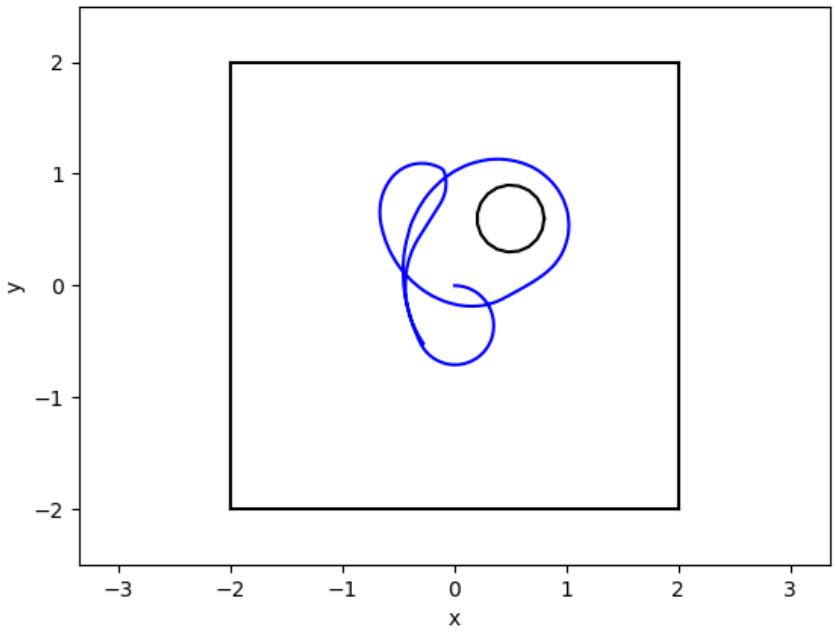

Figure 7: Path taken by Braitenberg controller robot over a period of roughly 1 minute.

As is obvious from the graph, the robot is able to successfully navigate around the obstacle and roam the playground.

(b) The rule-based robot is implemented with roughly the same rationale, for the ease of

comparison.

Firstly, if the room for maneuver in the forward direction is less than 0, meaning the distance between the robot and the wall in front is less than the safety distance, then forward velocity u is set to 0.05. Otherwise, u is set to max_u.

Next, the weighted differences between room for maneuver in $front\_left$ and $front\_right$ and between $left$ and $right$ is again computed. If this value is higher than a positive threshold, then ω is set to max_ω. If this value is lower than a negative threshold, then ω is set to $-max\_\omega$. Otherwise, ω is set to 0

The resulting trajectory is shown as follows:

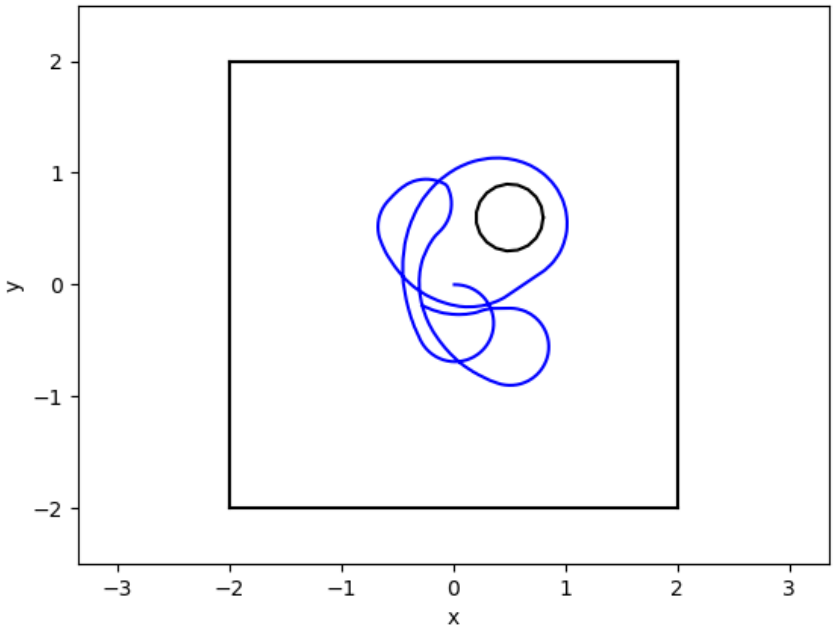

Figure 8: Path taken by rule-based controller robot over a period of roughly 1 minute.

It can be seen that the shape of the trajectory is mostly the same as the one above, since the logic of the rule-based controller is designed to emulate that of the Braitenberg controller. The difference is that the rule based controller covered more distance than the Braitenberg controller, which is expected as the forward speed is set to max most of the time for the rule-based controller while that of the Braitenberg controller is constantly scaled by the safety distance.

(c) Both controllers are fairly robust when obstacles are placed far away from each other. However, the rule-based controller has a tendency of running into an obstacle under unfortunate circumstances where the aim of the robot is too straight and the $front\_left$ and $front\_right$ sensors are unable to detect the obstacle in time. The more gradual approach of the Braitenberg controller seems to resolve this issue much better as the robot starts slowing down earlier than the robot with rule-based controller.

When trying to move through a narrow gap between two obstacles, the rule-based controller seems to be able to handle the situation faster than the Braitenberg solution, since its speed is not affected as much by the proximity to obstacles as the Braitenberg con-

troller. Again the rule-based controller is slightly less robust in this case due to its faster speed and might graze the obstacles.

(d) The robots are tested again under a very high noise of 1. Where the trajectories are shown as follows:

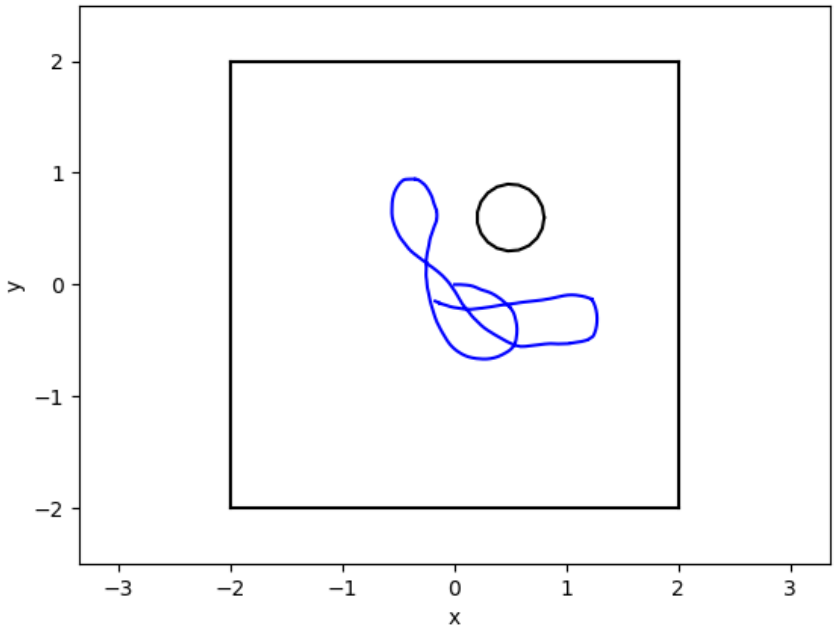

Figure 9: Path taken by Braitenberg controller robot with a sensor noise of 1.

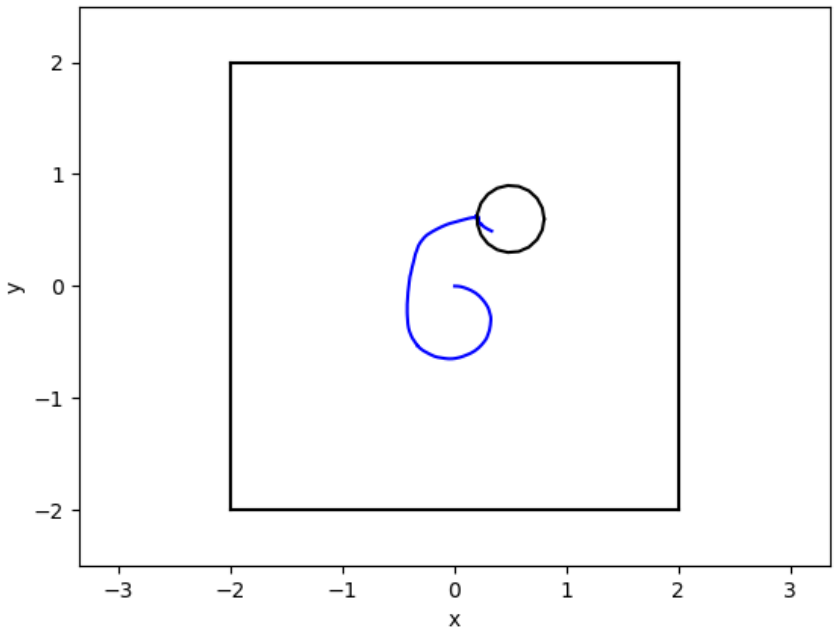

Figure 10: Path taken by rule-based controller robot with a sensor noise of 1.

Note that due to the random nature of noise, these graphs are not completely representative of the performances of the robots. In fact, both robots are liable to run into obstacles under such high noise conditions. However, after running the experiment multiple times, the Braitenberg controller feels slightly more robust under these conditions than the rule-based controller, though the difference is not very significant and may have been heavily affected by the limited sample size. If the Braitenberg controller is indeed more robust, the interpretation would be again that it has a smoother transition of velocities, allowing it to better deal with changing conditions. Also, the frequent pauses of the Braitenberg controller under high noise may also contribute to its ability to survive longer.

Localization

2.2 Sinusoidal disturbances

(a) Screenshot of setput:

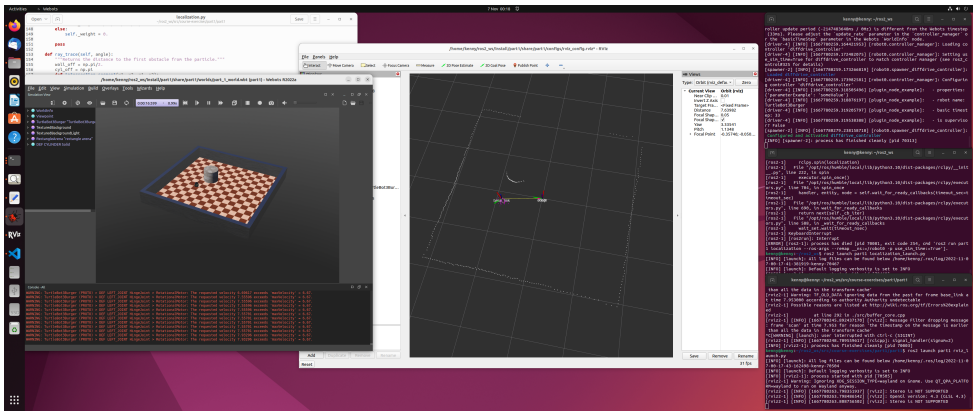

Figure 11: Localization setup.

Most of the setup is exactly the same as it is for previous exercises. However, there is a new window of *rviz*, where the white dots represent laser readings from the robot and the pink dots represent the point cloud generated from the particle filter, and as shown in the setup they are very well converged around the robot, which is represented by the three principal arrows with the label *base_link*. The implementation of the particle filter is discussed below.

(b) The implementation of particle initialization and validity check is straight forward: The initialization generates a random coordinate within the playground, and the validity check makes sure that its coordinate is within the playground, and that it is more than 0.3 distance away from the center of the cylinder (which makes sure it is outside the radius of the cylinder).

(c) Examining where the move function is called, one can see that the input parameter *delta_pose* has a non-zero value in its first index, representing the forward distance move by the particle in the particle frame, 0 in its second index, as the robot cannot have lateral movement, and another non-zero value in its last index, representing the rotation of the robot. These are easily transformed into the pose of the particle in world coordinate using basic trigonometry with the current yaw of the robot.

(d) If the robot does not start at $(0, 0)$, then its initial position must be known and the origin of the odometry modified accordingly to meet the initial position of the robot. Otherwise the positions of the particles will be constantly offsetted, which, combined with the presence of obstacles, will cause great problems with localization.

(e) Firstly, a validity check is made on the particle. If the particle is invalid, its weight is directly set to 0.

If a particle is valid, then the weight of the particle is computed by summing the likelihoods of getting the readings at the particle's present location given the present readings from the actual robot, which are the five laser sensor readings. The likelihoods are computed using the pdf of a Gaussian with mean equal to the actual robot reading and standard deviation of 0.8, taken at the particle reading pointing in the same relative direction. As a result, the weight is the sum of the values from 5 pdfs, corresponding to the difference between the five laser sensor readings of the actual robot and the particle: the more similar the readings are, the higher the weight of the particle.

To handle resampling and particle collapse, a constant number of 50 particles are kept alive at all times. Firstly, tthe max-weighted particle and the mean weight of all particles are found. For all particles that have weights below the mean, they are relocated to the position of the max-weighted particle, otherwise they are left untouched.

(f) The particles will always converge quickly without fail, therefore there is no need to explain for unsuccessful or slow convergance. However, it is noticed that after running the simulation for extended periods (e.g. after $1 - 2$ minutes), the particles will start to lag behind the actual robot. The cause of this issue is believed to be increasing error of the laser scan published under "/robot0/TurtleBot3Burger/scan".

(g) Strangely, manually changing the position of the robot seems to alter the entire odometry, where the simulation seems to believe that instead of the robots being moved, the entire map is moved underneath the robot. As a result, this causes problems as mentioned in part (d) of this exercise, which cannot be resolved by tuning parameters. If a small manual movement on the robot is made however, the particles are able to follow the robot with a small offset, showing that the particle filter is indeed robust.

(h) The trajectories and error over time plots are shown in the following page. Interestingly from the first graph, one can see that particles are trying to follow a fixed trajectory while that of the actual robot evolves slightly overtime. This is most unexpected as nowhere in the code are the particles asked to follow any trajectory, yet they do it spontaneously purely using the mechanisms of resampling.

It can be seen from the sharp initial drop in error that the particles does converge quickly, and indeed follow the robot very closely during the first lap around the playground. Errors grow in a unique periodic fashion where the particles and the robot always coincide as the robot try to go around the obstacle, but grows apart as the robot roams in the more open areas. This can be due to the fact that errors in the laser measurements increase as the value of measurement grows larger.

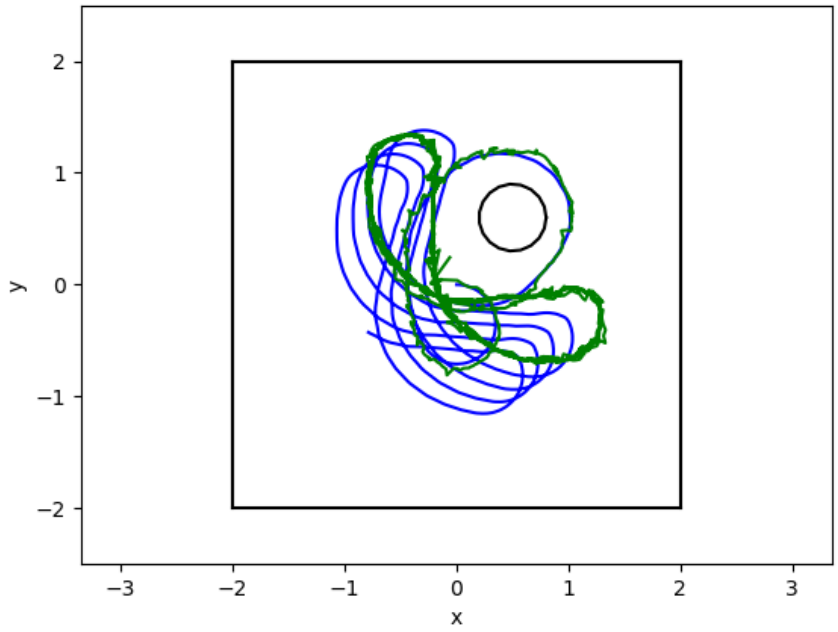

Figure 12: Path taken by Braitenberg controller robot (blue) and particles (green).

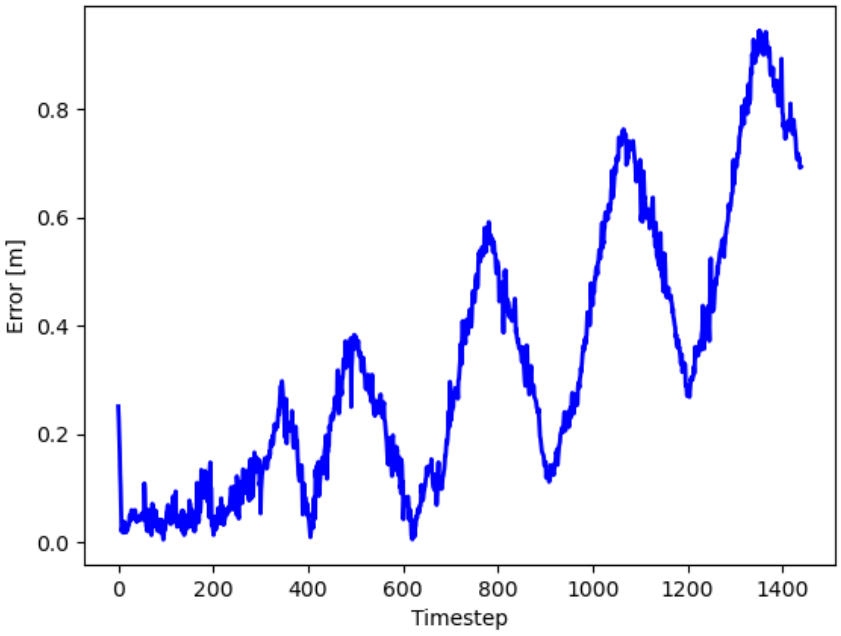

Figure 13: Error over time between the path of the Braitenberg controller robot and particles.

(i) If the system is assumed to be linear, then the extended Kalman filter would be optimal, as shown in the lectures. However, the slowly evolving trajectory of the robot and the nonlinearity in the behaviour of the laser scanners as observed in the earlier parts of the exercise shows that the system is not completely linear. As a result, the performance

advantages in terms of accuracy is difficult to determine without actually carrying out tests.

In terms of computational cost, Kalman filter ususally has the advantage. However, extended testing has not yet been carried out for the particle filter to determine the optimal number of particles, which could possibly be lower than the present number of 50.