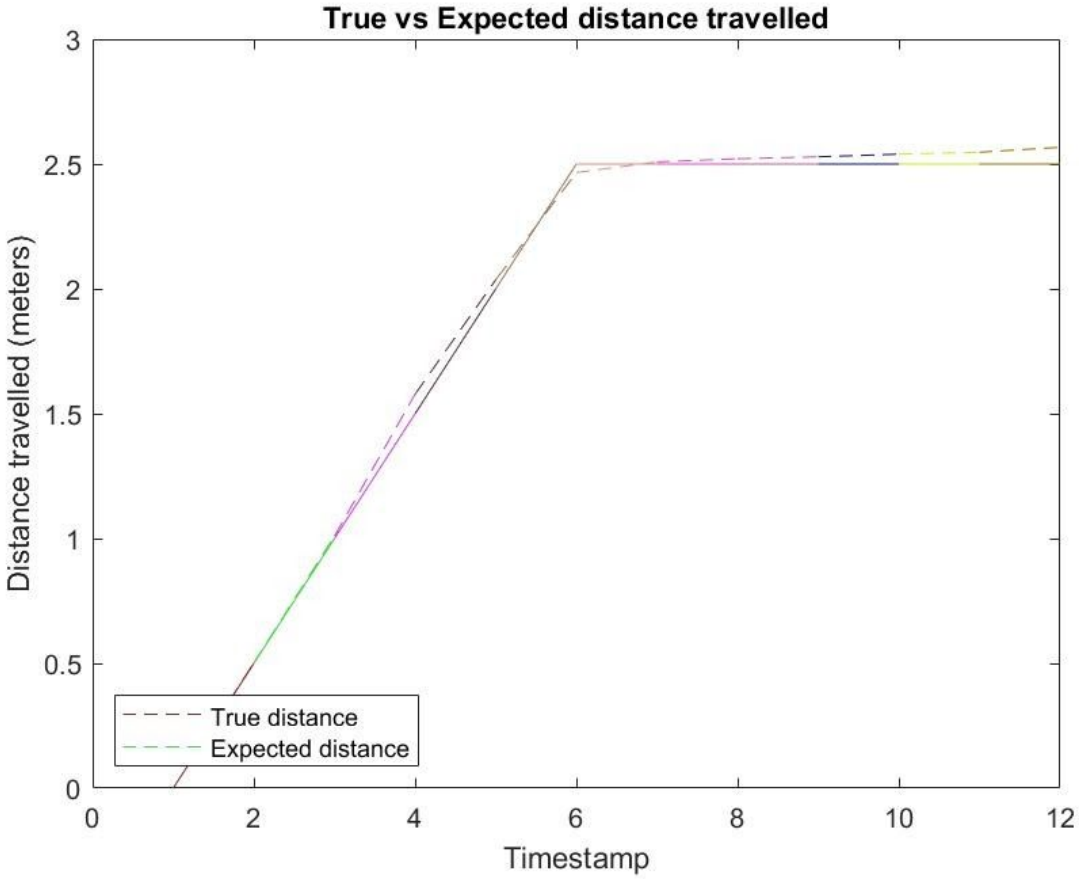
Jonathan Moon and Jonathan Nusantara
AMR
Lab 1 Report

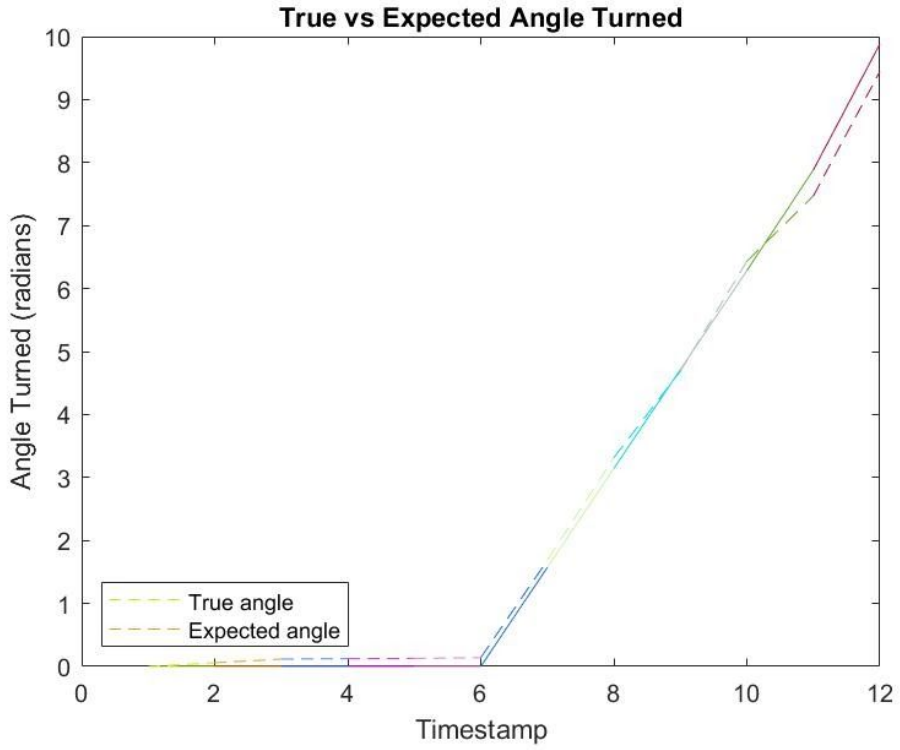
2 Post-Lab Assignment

The group should submit one post-lab report Lab1report.pdf, on Canvas, one week after the lab took place, by 11:59 PM.

2.1 Actuation and sensor errors (20 points)

(a) In 1.3, the function OpenLoopControl.m commands the robot to perform a series of motions. Create two figures, one for the distance traveled and one for the angle turned. On each of these figures, plot the actual data (from your saved PoseData) and the expected motions based on the function. Plot each motion segment (one command in OpenLoopControl.m, either a forward motion or angle turned) in a different color and use a different line style to distinguish between the expected and the actual motions.





Note: legend is incorrect. The expected distance/angle travelled/turned will be denoted by a solid line.

(b) Based on the figures and analyzing the difference between the expected and actual motions, comment on the actuation errors you observe. Does the robot perform the desired motion perfectly? does the error, if any, depend on the speed? How can we mitigate such errors? How fast do you think we should drive the robot?

The robot does not perform the desired motion perfectly. It appears as though the error gets worse the faster the robot is travelling/turning during those moments. From the first graph, we can see that there is some difference in measurements as the robot travels faster (timestamp 3-4). From the second graph, we also noticed a big difference between the two measurements at timestamp 10. This is most likely due to an error from the sensor's measurement, possibly due to noise or wheel slippage.

One way to go the desired speed as well as achieve precision in angle and distance travelled is to slow down as you approach the desired distance (if you call turnDist) or desired time (if you decide to set the fwd velocity and pause for some time). Without loss of generality, the same should be said about angles (i.e. slow angular velocity as you reach the target). I think the robot should be driven as fast as what is permitted by safety of humans, other robots, and itself. Based on the experiment, velocity below 0.5m/s seemed acceptable. By using this slowing down method, the robot should be able to go any speed that is below the max speed dictated by safety.

(c) For the depth data collected in 1.4, taking into account the location of the sensor with respect to the robot (estimated in 1.4), calculate the difference between the actual measurements and the expected measurements. Do they match? is the error constant? does it change with the distance? what could be the sources of the error? comment on the precision and accuracy of the depth sensor.

At 2 ft, the xy offset is (2.6, -0.98) in centimeters. At 3 ft, the xy offset is (5, -0.071) in centimeters. The calculated offsets did not match and the errors are not constant. The X offset seemed to change with the distance, where the error increased as distance increased. One source of the error may actually be human error. When measuring the distance using the meter rule, there may be some human error due to the ruler being angled during measurement. Thus, the offset calculations may be inaccurate. Another source of error may also be the noise from the sensor. Since there is an offset, this shows that the depth sensor is not too accurate. However, after calculating the average of multiple measurements with various depths, there is not a lot of variance among the measurements, and so the sensor seemed to be pretty precise.

(d) If the distance measurement error is a constant bias (a constant error regardless of the distance), how would you take that into account when using the measurements? if the error is not constant, how would you take that into account?
Since the error is not constant, we can create a linear model on how much the error will increase as the detected range measurement increases. However, in order to create an accurate model, we will need more data points (measure form 4ft, 5ft, and so on)

2.2 Sensor frequencies and delays (25 points)

(a) For the data sets obtained in 1.5(a)–(c) observe and compare the timestamps of the data. What do you notice? How often can you get the different sensor information? How does this frequency depend on your code?
This frequency depends on how many sensors we are measuring from in the code. The function that makes these measurements is the readStoreSensorData(). Based on our collected data, if we are to record all sensor measurements, it took about 0.9 seconds for each measurement. If we are to comment out some of the code that measures the sensor, then those sensors will not be measured, and each loop of our program will be faster. For example, when we are only measuring the result from overhead localization, the measurement is recorded every 0.05 second. Some sensors may take longer to measure, such as the bump sensor that would take about 0.1 second for each measurement. As a result, there will be a higher frequency in sensor measurements when there are less number of sensors measured as stated in the readStoreSensorData().

(b) What are the implications of these delays on your control code and the robot behavior? how might you mitigate any unwanted behaviors?
With these delays, it may cause our robot to make decisions based on past measurements, which may cause it to make imperfect controls/decisions. For example, when the robot receives the truthPose from the overhead localization, it may be the information from a few milliseconds ago. As a result, when the control algorithm calculates the velocity based on truthPose, it would be from the previous location, thus making the wrong control. In order to mitigate this, one way is to create more functions to read the sensors. Instead of reading all of the sensors using the function readStoreSensorData(), we can create a function for each of the sensors. We can then just measure the sensors as needed. This prevents the delay caused due to getting measurements from sensors that we do not need and to have a better robot control.

(c) For the dataset in 1.5(a) what do you observe about the timing of the depth and tag information with respect to the other sensors? When doing localization (as you will do in Lab 2), how should you deal with the delay in these measurements?

In the recorded data of the depth and tag information, there is a column that specifies the amount of delay for the depth or tag information to be processed. As a result, we can calculate the time in which this information is actually recorded. For example, if the time stamp is at time 3 seconds and the delay is 0.5 second, then the measurement is actually made from the state of the robot at time 2.5 seconds.
When doing localization, we may be calculating the location based on the depth/tag and the truthPose. We would need to match the depth measured with a previous truthPose. This may be calculated by using the amount of delay, in which we will try to match the timing of the truthPose measurement and the timing of the depth/tag measurements after taking the delay into account. This allows the most accuracy localization.

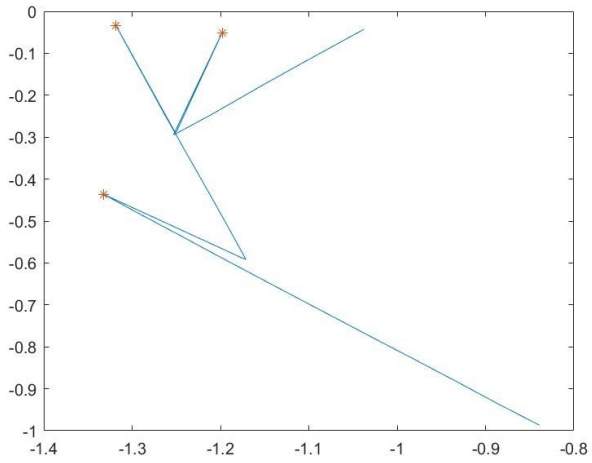
(d) For the datasets in 1.5(d), how does the number of tags affect the delays?
Looking at the collected data, we cannot really determine how the number of tags affect the delays. In the first 10 recordings, we can see how there are less delays when there are less number of tags. However, the amount of delays tend to vary at every measurement. As a result, we cannot really conclude the relationship between the amount of delays and the number of tags.

(e) In Part 1.6(e)–(f), did the robot react differently to the presence of a wall? How so?
Yes, the robot reacted differently. After the other sensor measurements except the bump sensor are commented out, there is a faster response to the robot. When all sensors are measured, it will take some time for the robot to backup after hitting the wall and it can easily be seen how the robot is still moving towards the wall after hitting it. However, after commenting out, the robot will almost immediately back up after hitting the wall.

(f) If so, what is the reason for the different behavior? If the behavior was identical, why would one expect commenting out parts of the code would lead to different behaviors?
The reason is that the robot (or the RaspberryPi) would spend some amount of time recording measurements from the sensors. These sensors will be recorded every time the readStoreSensorData() function is called.

2.3 Sensor Information (35 points)

(a) Plot the robot trajectory from Part 1.6(a)–(d).
(b) Indicate places where the robot's bump sensors were triggered (for example, with a *).



(c) How might you build an obstacle map if you only had information from the robot's bump sensors?

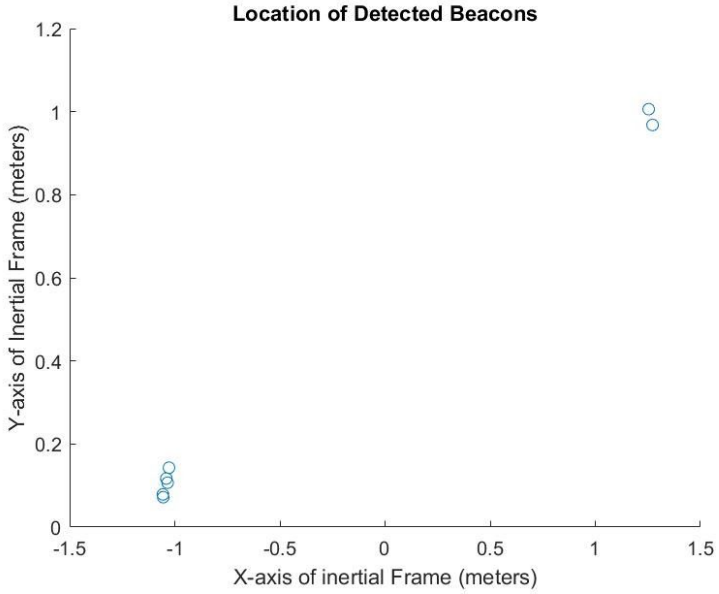
Suppose you run the robot for a long enough time such that we get many locations where the robot's bump sensors were triggered. What this essentially provides is a point cloud of the obstacle map that you can reconstruct into surfaces (via surface reconstruction or some interpolation).

(d) Using the function robot2global.m, and the location of the camera in the robot-fixed frame you obtained during calibration, plot the locations of any detected beacons (indicate their locations with a different symbol than was used in part (b).) Keep in mind that the robot pose timestamps are not perfectly aligned with the beacon timestamps. How did you handle the timing mismatch? Did you pick the "closest" timestamp? Did you interpolate the data, and if so, which data? Justify your choice.

To handle the timing mismatch a few preprocessing steps were taken to rectify the issue.

1) In order to get the correct time at which the beacon data was recorded, the time delay was subtracted from the recorded time of the beacon data.

2) Linear interpolation was used to estimate the pose of the robot at the time the beacon camera data was acquired. To delve into the details, two truthPoses whose time of recording "surround" the adjusted time (i.e. recorded time - beacon data; refer to step 1) of recording of the beacon camera data. The pose of the robot at the adjusted time of recording of the beacon camera data was then interpolated from these surrounding truthPoses, weighted by the difference in time.



(e) Discuss the precision of the camera. Are the locations of the detected beacons consistent, or do they move around a lot?

The precision of the camera appears to be not very high. The locations of the detected beacons are inconsistent, wandering around by a rough noise ball with a radius of several centimeters.

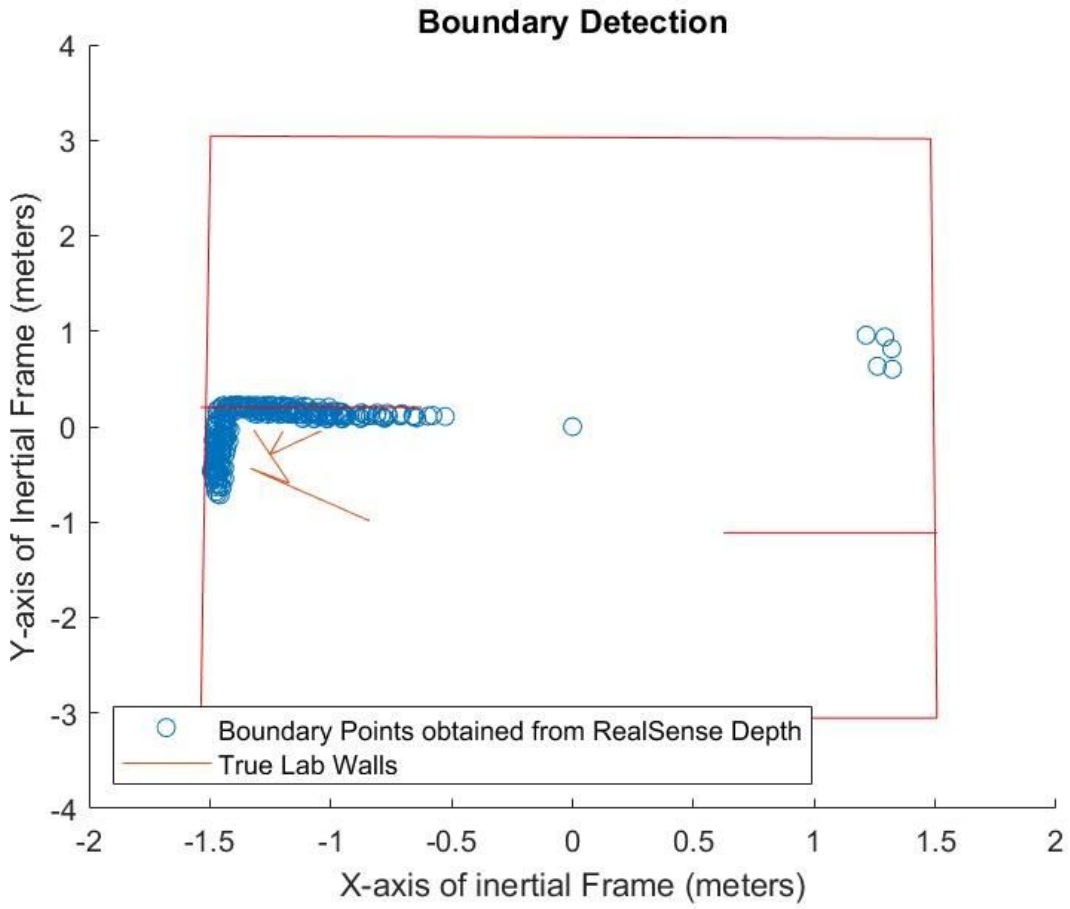
(f) Write a function depth2xy.m to convert the raw depth measurements into 2D local robot coordinates (this function will be similar to lidar range2xy.m from Homework 1.) Make sure to take into account the location of the camera you obtained during the lab. The minimum distance for the depth sensor is 17.5cm; if an obstacle is closer, the sensor will return 0. How are you taking into account the nonlinearity in the range error, if at all?

```
function[depthXY] = depth2xy(depthR, camera2center, angles)
% depth2xy: convert depth measurements into x/y
% coordinates (in robot local frame)
%
%   depthXY = depth2xy(depthR,camera2center,ANGRANGE) returns the
%   x/y coordinates (in robot local frame) of depth measurements.
%   For values that are below the minimum distance for the distance sensor,
%   the coordinates are returned as (0,0) since this is an invalid value
%   under normal circumstances.
%
%   INPUTS
%       depthR      1-by-N vector of scan ranges (meters)
%       robotRad    robot radius (meters)
%       angles    total angular field of view of depth (radians) (1-by-2)
%
%   OUTPUTS
%       depthXY     2-by-N matrix of x/y scan locations
%
%   NOTE: Assume depth is located at front of robot and pointing forward
%         (along robot's x-axis).
%
%   Cornell University
%   Autonomous Mobile Robots
%   Homework #1
%   Moon, Jonathan

depthR
N = length(depthR);
depthXY = zeros(2,N);
for i = 1:N
    theta = angles(N-i+1);
    if depthR(i) < .175
        continue
    end
    depthXY(1,i) = depthR(i) + camera2center(1);
    depthXY(2,i) = depthR(i)*tan(theta)+camera2center(2);
end
```

In the case that the depth measurement is below the minimum distance, depth2xy will return (0,0) in the robot's reference frame as this technically is an invalid pair of coordinates that the depth sensor can output. This is an indication that the a point in the inertial reference frame should not be plotted.

(g) Plot the lab map and the robot's depth data from Part 1.6(a)–(d) in global coordinates (you may use the function robot2global.m from Homework 1 for the depth data). Keep in mind that the robot pose timestamps are not perfectly aligned with the depth timestamps. How did you handle the timing mismatch? Did you pick the "closest" timestamp? Did you interpolate the data? Justify your choice.



To handle the timing mismatch a few preprocessing steps were taken to rectify the issue.
1) In order to get the correct time at which the beacon data was recorded, the time delay was subtracted from the recorded time of the beacon data.
2) Linear interpolation was used to estimate the pose of the robot at the time the beacon camera data was acquired. To delve into the details, two truthPoses whose time of recording "surround" the adjusted time (i.e. recorded time - beacon data; refer to step 1) of recording of the beacon camera data. The pose of the robot at the adjusted time of recording of the beacon camera data was then interpolated from these surrounding truthPoses, weighted by the difference in time.

(h) How well did the depth sensor measure the truth map? For example, are the walls detected as straight lines? What may be some sources for errors?
The walls are not detected as straight lines; rather the points seem to delineate a rough region where the boundaries are. One of the error sources may be the noise from the RealSense camera measurement. Additionally, since the depth is calculated using the robot pose, there may be noise from the overhead localization too, causing a cascade of noise contributions. Another contributing factor is the estimation of the robot pose required by the timing mismatch between depth sensor time queries on the truthPose

(i) Load the depth returns from the stationary robot run in Part 1.5(a)–(c) (all the runs that have depth information). If the depth noise is modeled as a Gaussian, $w_{depth} \sim N(0, \sigma^2)$, estimate σ.

$\vec{\sigma}_B$ $\vec{\sigma}_C$ are vectors where each element is the standard deviation for a particular angle (9 evenly spaced angles within [-27deg, 27deg]). Vector B is from section 1.5b and C is from 1.5c.

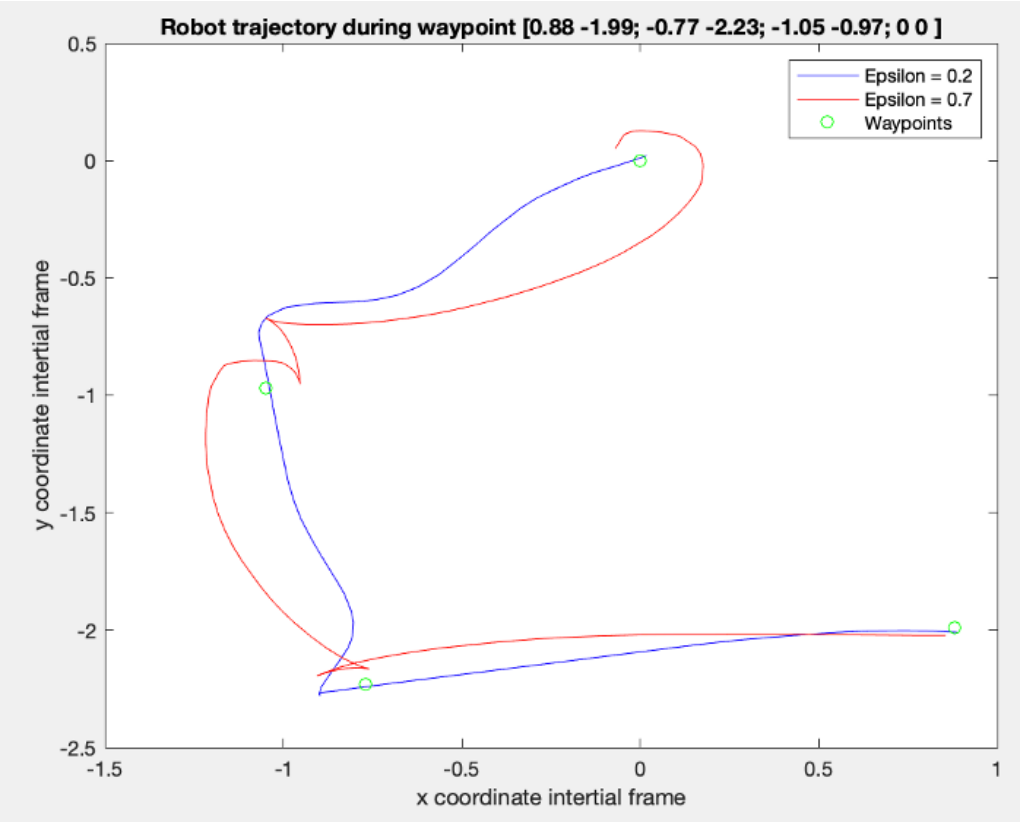
$\vec{\sigma}_B =$ 0.0026 0.0032 0.0042 0.0065 0.0035 0.0060 0.0056 0.0055 0.0028

$\vec{\sigma}_C =$ 0.0031 0.0029 0.0049 0.0057 0.0039 0.0056 0.0056 0.0049 0.0031

2.4 Waypoint Follower (25 points)

(a) Plot all the robot trajectories from Part 1.7 on the same figure (you should have one for each member of the group) and indicate which ε was used by each robot for feedback linearization.

(b) Plot the waypoints on the same figure.



(c) Comment on any differences between the robots' trajectories. Why wouldn't the trajectories be exactly the same? Did members of the group handle this task differently?

The blue plot has a smoother trajectory between the waypoints. The trajectory is more or less pretty straight, with some turns as it moves from one point to another. On the other hand, the red plot has a rougher trajectory. We can see how it has wider turns, which we can say is a less efficient path (considering the most efficient path is a straight line from one point to another). In some cases, due to the wide turn, it needed to back-up at point (-1.05, -0.97) before moving to the next destination. Due to the wide turn angle, it cannot turn in time, requiring it to back up.

The trajectories are different because they have a different epsilon. The epsilon is a value that would determine the robot's turning radius. The blue plot has an epsilon of 0.2, while the red plot has an epsilon of 0.7. Since the red plot has a bigger epsilon. It explains the wider turn angle seen in the plot.

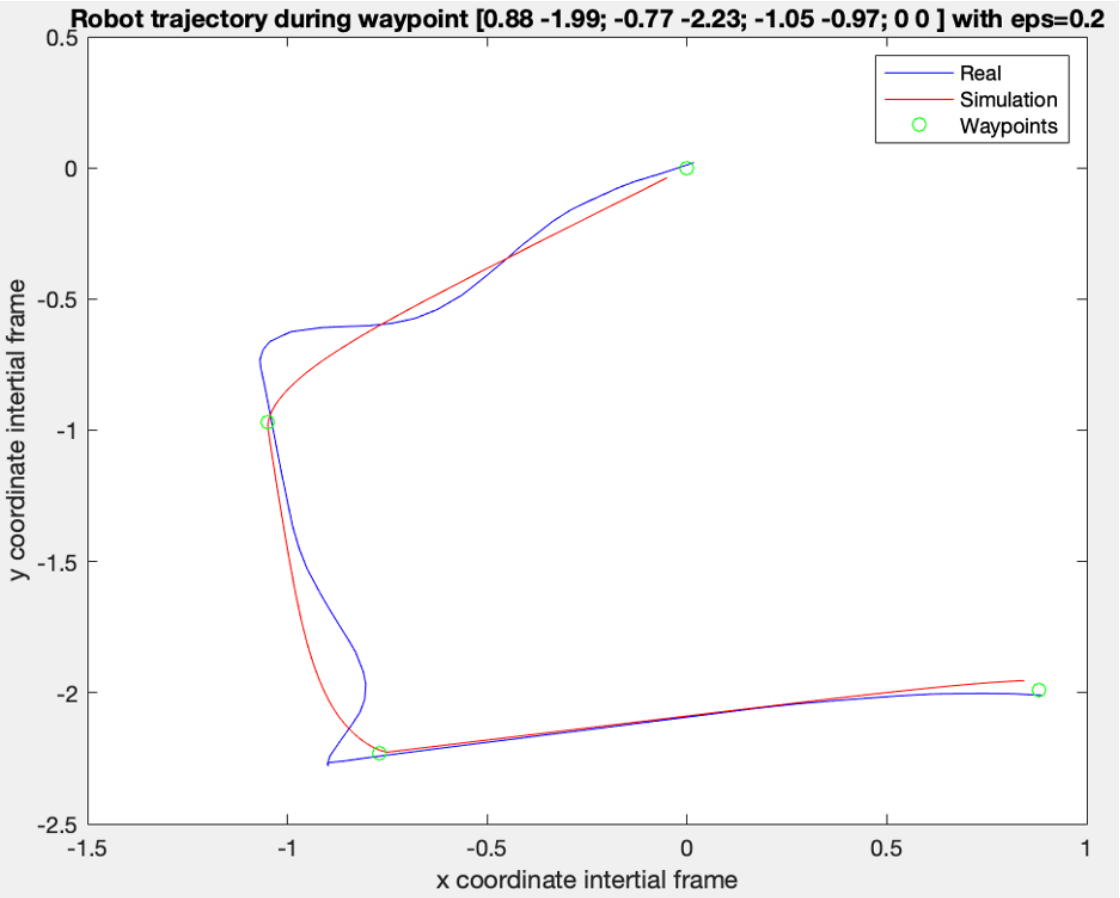
Both members handle this task similarly, as a result the only cause of this difference in trajectory is the epsilon, and not the program itself.

(d) How did the value of ε for feedback linearization affect the robots' trajectories? How would the trajectory have been different for a holonomic vehicle?

In feedback linearization, epsilon is used as an imaginary x-coordinate point in the body frame, which will then be multiplied with the angular velocity in order to move sideways (y axis body coordinate). This is needed because the robot is non-holonomic, so it cannot move along the body y-axis freely. The bigger the epsilon, the bigger the turning radius would be. This is also seen in the plot in our homework. As a result, we can see in our plot above that when the epsilon is smaller, the trajectory is straighter from one point to another. When the epsilon is larger, the robot has wider turns during rotations and movements between points. This also results in a less smooth trajectory.

A holonomic vehicle is free to move anywhere (can move sideways). As a result, it does not need an epsilon value. The trajectory will be a straight line from one point to another, as the movement is not constrained.

(e) For one of the ε values, run the same waypoints and the same code in the simulator. Plot the resulting (simulated) trajectory. On the same figure, plot the actual trajectory. Do you expect both trajectories to be the same? Are they? If they are not the same, explain what the possible sources of difference are.



The blue plot above shows the real trajectory of the robot with eps = 0.2, while the red plot is the simulated trajectory with the same eps value.We can see a difference between the two, where the red plot has a smoother trajectory through the waypoints. The simulated trajectory also passes through two of the waypoints right on the dot. One possible source of error would be the

sensor noise in the real world. Due to the sensor noise, the robot has less accurate information of where it is currently at. So, there may be slight differences in the trajectory of the robot, due to a different accuracy of truthPose recorded. Additionally, the error is due to a difference in the time delay between truthPose measurements. Looking at the dataStore.truthPose data, the real robot recorded measurements at an average of 0.9 second, while the simulated robot recorded measurements at every 0.1 second. As a result, it may overshoot a bit (traveling further than required) as it passes the points. This can be seen as it passes the second and third point, where it travels more than required and has to perform a sharper turn to go to the next waypoint.

2.5 Lab preparation (mandatory, not graded) (Nusantara)

(a) Which robot did you use?
Eve
(b) For the code you wrote as part of Homework 1, were there any coding errors that were not discovered when using the simulator?
No.
(c) How much time did you spend debugging your code in the lab?
We did not really debug, but the total time spent in writing or adjusting the code would be about 1 hour.
(d) Did you observe any behaviors in the lab that you did not expect, or that did not match the simulated behaviors?
No, everything is expected.
(e) What was the contribution of each group member to the lab and the report?
During the lab itself, Jonathan Nusantara worked more on the physical robot, such as moving and placing them at the right position. Jonathan Moon worked on writing the codes and running the function and programs during the lab. For the lab report, we first discussed what each of the tasks is about. We then split the work among us, before then discussing our findings.