Part 1.2: Robot path with no control

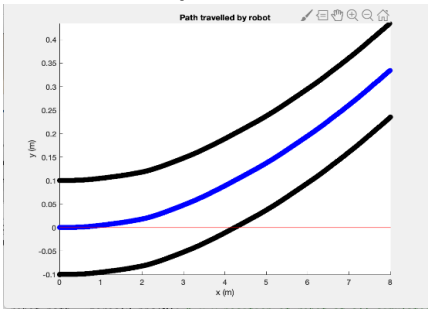


Figure 1 Original (First Run)

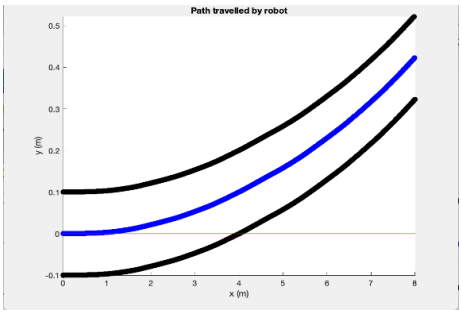


Figure 2 Fifth Run (Y Scale change to 0,5)

Possible factors that cause the trajectory of the robot to deviate is the uneven strength of the motors (for example the left motor duty cycle is slightly stronger than the right motor duty cycle), the uneven surface (rough surface for example).

Positional Disturbances

Encoder Noise: my_noise2 function called in newpos function

```
robot.encL(i)=robot.encL(i-1)+norm(robot.lWheel(:,i)-robot.lWheel(:,i-1))*my_noise2;
robot.encR(i)=robot.encR(i-1)+norm(robot.rWheel(:,i)-robot.rWheel(:,i-1))*my_noise2;
function x = my_noise2()
    x=1+sqrt(0.0001)*randn;
end
```

Positional Disturbance (shown in both the my_noise function and is called in the newpos function):

```
dL=(vL*dT*dir_old)*my_noise+robot.lWheel(:,i-1); % Intended positions of left and right wheels
dR=(vR*dT*dir_old)*my_noise+robot.rWheel(:,i-1);
function x = my_noise()
    x=1+sqrt(0.002)*randn;
end
```

Motor Power offset: found inside the newPos function

```
lDC = robot.lMot*1.002; % Left and right motor duty cycle | left motor is 0.2% stronger
rDC = robot.rMot;
```

Part 1.3: ***Justification for time and PPS Value***

I chose to use N=30 seconds as the simulated movement time of the robot as this is the approximate time that would be spent aligning the robot in the one axis of movement. In the project guideline document, we are allotted 120 seconds [1]. X-axis movement is one of the 6 additional tasks; thus an approximation is 120seconds/6tasks = 20 seconds. Another reason for this choice is that 30seconds multiply by 0.10m/s ~3m which is the maximum range of motion.

PPS chosen was 1500. The two deciding factors were the computing speed for the simulation and the accuracy of the graph. If the chosen PPS value is too high the time to generate the simulation may take too long which will prolong tuning times. However, if the chosen PPS value is too low, the accuracy may be compromised. For example, when the value of 50 was chosen, the graph of the trajectory was inconsistent each time the simulation was run.

Part 1.4: R was chosen as the size of the wheels used in our robot project which was 0.5m. wMax=1.16. This value is based on the max rpm of our DC motor which was 70 rpm. Separation between the wheels d was chosen as 0.30m.

Part 1.5: ***How spd, dT, ctrl_enable are used***

Spd**-** Defined in main_script.m and is used to set the speed in the getDC function. The desired speed is then converted to the duty cycle used to determine next position of the robot with newPos.m.

dT- A value initialized in main_script.m which determines the frequency of PID control in drive_robot.m by calculating the ctrl_interval variable

Ctrl_enable Initialized in main_script.m to choose PID control by case in drive_robot.m

The relationship between the control interval dT and the simulation time resolution pps

If these two are not properly matched the simulation runs into problems of properly positioning the robot when using PID control. When pps is not much larger than dT then the simulation is unable to properly perform the drive_robot.m function, which can be seen in the code that the control interval depends on both values:

```
Tstep=T(2); % Step size of simulation
ctrl_interval=round(dT/Tstep);
```

Part 1.6

The function first calculates the angular velocity (w) required to drive the wheel at a given speed (v) according to the radius (r). The duty cycle of DC motor is defined as the percentage of time the motor is in the high state [2]. As such, we can achieve the desired driven angular velocity with the maximum DC output by taking a ratio (w/wmax).

Part 2.1: ***What is the value of the proportional coefficient k?***

The value of the proportional coefficient K is originally 0.1 as stated in drive_robot function. Since the ctrl_enable variable is now set to 1, the proportional control case is run in the code. As the deviation in the robot's x position becomes larger, the motor duty cycle will be readjusted by a factor of K*pos_error (proportional coefficient k*deviation in x position). This will ensure that the robot will go in the opposite y-direction and back to the desired equilibrium position to drive in a straight line (x-pos=0).

Part 2.2: ***getError function, Encoder accuracy and Additional Sensors***

getError is called in drive_robot.m to determine the magnitude of adjustment needed to make when the robot deviates from the desired trajectory along the x axis. It finds the difference between the left and right encoders. Although using encoders for short-range odometry is quite accurate, error increases rapidly as distance travelled by the robot increases [3]. Encoder error can be further increased by slip. This can cause the encoder reading to be smaller than the actual distance travelled leading to even larger discrepancies in data collection and the control system.

An Inertial Measurement Unit sensor can be used to optimize odometry. This device combines 3 axis gyroscopes with 3 axis accelerometers. Together they provide accurate reading of angular and linear acceleration. These values can then be used to extrapolate the position and direction of the robot based on its known starting position. Furthermore, these values can be incorporated into the PID algorithm to improve control as described in [4].

Part 2.3/2.4 (Original K is 0.1)

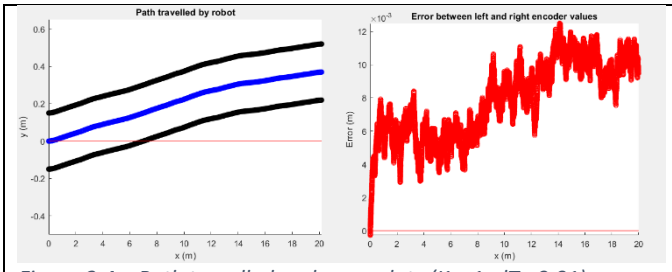


Figure 3,4 – Path travelled and error plots (K = 1, dT =0.01)

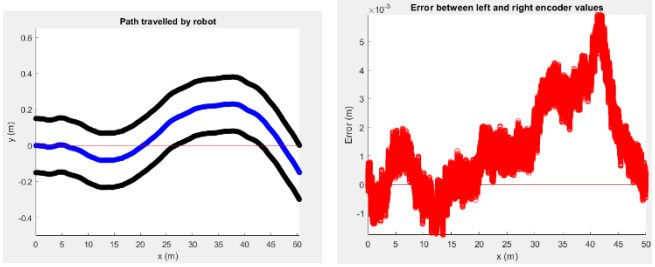


Figure 5,6 - Path travelled and error plots (K = 25, dT = 0.01)

When the value of K small, the makes gradual corrections towards the desired trajectory, graph is concave down (figure 3). In addition, the error oscillations are steadily increasing (figure 4) without settling at a constant range of values. Thus, for the small K case, we have observed that the robot slowly makes correction and has a long settling time. On the other hand, larger K values cause the movement of the robot to move towards desired trajectory (figure 5). However, it is evident that the robot made too large of a correction and overshoots the X-axis. This overshoot is also reflected in error (figure 6) where

we observe very large oscillations which increases the time it takes for the error to settle back at equilibrium (y=0). Hence, for large K values, the correction factor is too large and overshoots the equilibrium position. By these two graphs we can say that K is an important factor in determining the settling time or how quickly the robot will attempt to shift towards the control range.

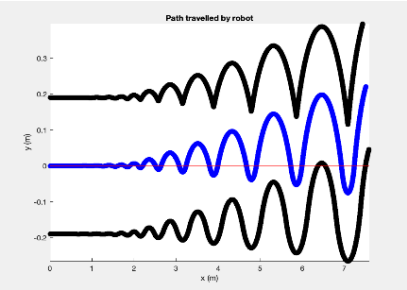


Figure 4 – K=100, dT = 0.1

As dT gets smaller the frequency that the PID control is initiated increases. Thus, changing dT affects how often the control loop is executed. attempt to correct its trajectory towards the desired path. When this number is too large, the robot will not attempt to correct its trajectory fast enough. Severe overshoot, seen in figure 7, will result where the robot continually makes a larger and larger corrections.

Part 3.1: ***Discretizing the terms of the PID, since in this numerical simulation and frequency of calculation***

The Integral can be discretized by accumulating the encoder error at each time we execute the control loop. As the integral term will be multiplied by constant gain (kI), there is no need to multiply each discrete error value by a differential, as this can be accounted for in the tuning process. A discretization of the derivative term (kD) by taking the difference of the current and previous error value and dividing by the time step which will yield an approximation of the derivative. For both values, the resolution of the simulation should be used, since this gives a smaller time step and therefore a more accurate approximation of the continuous system.

Part 3.3: ***Approach to tuning the PID parameters***

To tune the PID I began by following the Zeigler-Nichols closed-loop tuning process. To achieve oscillatory motion a compass was implemented. My PID tuning process was as follows:

1. Set the gain value (kP) to the smallest value such that there are constant oscillations while integral and derivative terms set to 0.
2. This critical value of kP was the ultimate gain (P_u) and the period of oscillation was the ultimate period (T_u). These were used to calculate the baseline tuning parameters as described in the Zeigler-Nichols tuning method [5].
$$kP = 0.6 \cdot P_u \qquad kI = 0.5 \cdot T_u \qquad kD = 0.125 \cdot T_u$$
3. Each PID parameter was then individually increased and decreased to determine their behaviour and trends.
4. The trends were then used for adjustment to achieve a smooth trajectory to minimize frequency and deviation from desired path.

The trends that were found in step three of the tuning process are as follows. kP increases with the oscillation frequency, while oscillation amplitude decreased. On the other hand, the integration term KI and derivative term kD possess a damping influence for the system. When we compare figure 8a and figure 9a, we can see that amplitude of the oscillatory motion decreases when integration control is added. If we examine 8b and 9b the errors are also an indicative of damping. Each figure 8b,9b are underdamped but we can see that the error oscillations decrease much faster in figure 9b. Similarly, the addition of derivative control dampens the system (figure 9a-10a) allowing the robot to follow its intended trajectory with very high precision. In fact this control creates an overdamped system which is observed in figure 10b, the error does not overshoot and slowly approaches equilibrium at x=15m.

Although underdamped and overdamped control are very visible, it is very difficult to observe critical damping effect in this simulation. The disproportional motor power is analogous to a forced

system which causes the system to constantly diverge from equilibrium as seen in figure 10b. This is problematic in determining the critical damping case because we are unable to precisely determine when the system settles on equilibrium. Thus, it is difficult to achieve a figure that can be described as critical damping with certainty.

Underdamped cases – Error plots' oscillations decrease slowly

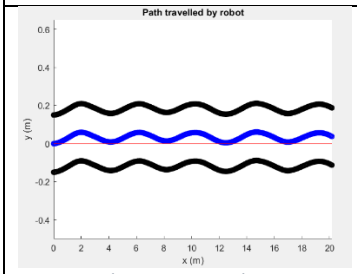
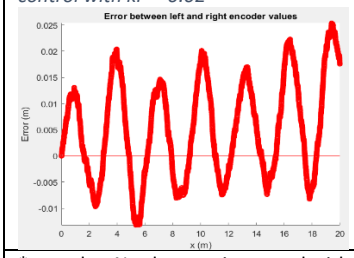


Figure 7 a,b - Proportional + compass control with kP = 0.02



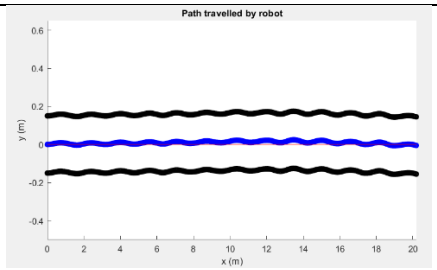
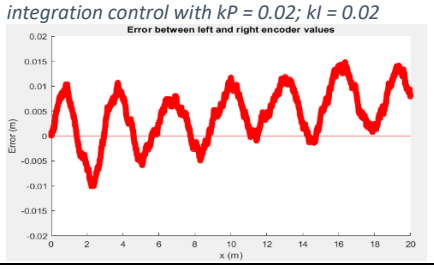


Figure 8 a,b – Proportional + compass integration control with kP = 0.02; kI = 0.02



Overdamped – Errors oscillate in proximity

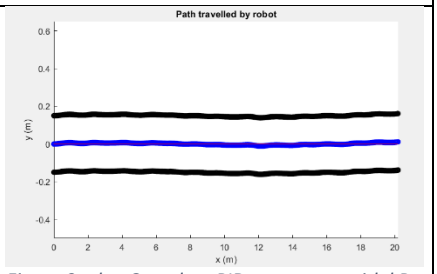
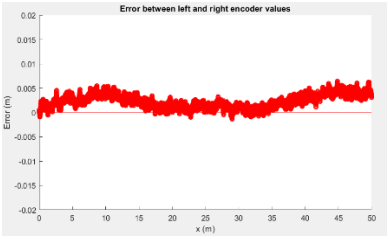


Figure 9 a,b – Complete PID + compass with kP = 0.02, kI = 0.02, kD = 0.008



*note that N value was increased with various controls to visualize long term behaviour in part 3

<u>Part 3.4:</u> ***Compare the behaviour of the system using the PID to other control schemes***

In part 3.3 we observed the oscillatory behaviour of proportional control with a compass and how integration and derivative created a damping effect. Now we will consider 3 other control schemes: integration only, derivative only and proportional derivative control.

Only I Control

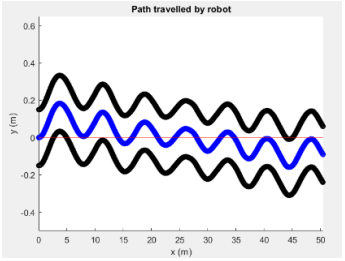


Figure 10 - Control with kI = 0.001

Only D Control

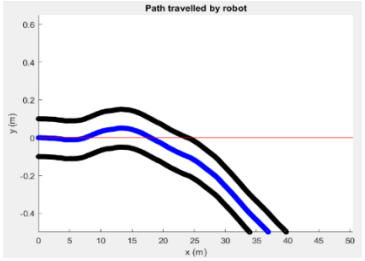


Figure 11 - Control with kD = 0.065

PD Control

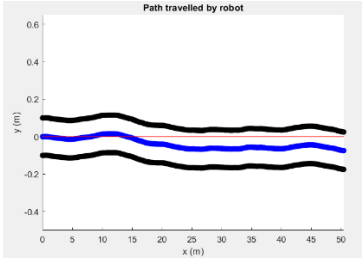


Figure 12 – PD control kP = 0.01, kD = 0.07

Integration control in (figure 11) attempts to prevent the robot from diverging off the X-axis. However, due to the disbalanced DC motor power output (part 1.2), the robot reverts in the direction of the stronger motor. Overall, it performs adequately and is essentially proportional compass control (figure 7) with larger oscillations. Only derivative control the robot can prevent itself from following the no control case when it travels in the positive y direction (figure 1). Using D control creates a new problem involving the stability of the robot's trajectory. Since the derivative control takes the difference of speeds and adjusts the speed based on this value the D gain acts very quickly. As a result, the robot's trajectory can suddenly change and go off track as in figure 12.

Theoretically a combination of PD control should work well. In figure 8 with only proportional control we saw that the trajectory was consistent, albeit having oscillations. With D control it tends to prevent the system from quickly changing it speeds and direction as in figure 12. Therefore, if we

integrate derivative control into proportional, we expect a damping of the oscillatory behaviour. This is confirmed by figure 13, where can see that the robot follows a very smooth path that does not deviate quickly from the desired trajectory.

Part 3.5: ***Limitations of PID control and simulation***

Of the 6 control systems, PID control with compass was followed the trajectory most accurately and thus it was chosen to test. As discussed in part 1.2, disturbances included encoder noise, positional disturbance and motor power offset. we configured the motor offset to have the left side double the right side, there is a huge deviation away from the intended trajectory (figure 14). In this case the left motor is so strong that the PID control is unable to move in that direction.

When positional disturbance reaches a certain threshold, both the direction and speed of the robot are heavily impacted. A closer examination of the axis of figure 15 show that the distance travelled in 30 seconds indicates the robot was moving far faster than the 0.2m/s that was used as a parameter. Lastly, increasing the noise of the encoder causes a complete collapse of the system. In fact, as seen in figure 16, the robot begins travelling in a circle as none of the PID parameters can react to the constantly shifting noise. Overall, most of the simulations function properly until the disturbances become exceptionally large. All the figures shown below devolve only at error that were larger by tenfold in terms of noise and the motor power offset operates past the likely power offset caused by manufacturing defects. Thus, although there is a limitation to PID control, most of these limitations exceed the bounds of what will be encountered realistically.

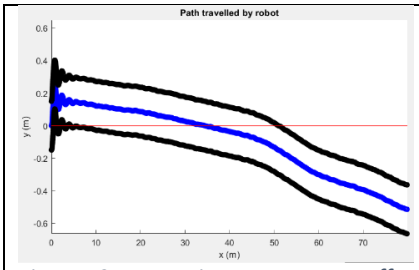


Figure 13 - Increasing motor power offset by making left motor 2 x stronger

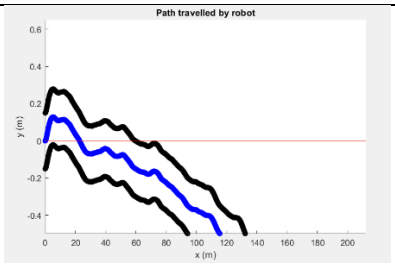


Figure 14 - Increasing mean noise of positional disturbance from 1 to 20

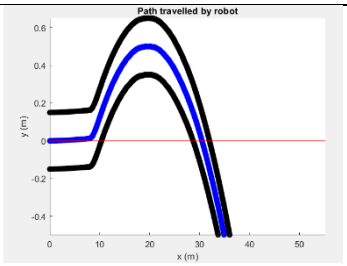


Figure 15 - Increasing mean noise of encoder from 1 to 35

*PID Terms were kept constant in all three above simulations

Part 3.6: ***Practical Issues in Real world implementation***

PID implementation in a real life could present timing errors when processing data. For example, in the simulation we are modelling the encoder and other sensors using various functions. In order to ensure that the data is processed sequentially, the code is written in order. However, if we were to use a serial processor with actual sensors, we would need to ensure that all the data being used for control is sent in the correct order so that the PID loop is evaluated correctly.

Integral wind up can occur when the sum of previously errors in one direction dominate, which will result in overshoot becoming prolonged [6]. The algorithm can be improved to reset the integral in certain situations such as: setting pre-determined bounds or zeroing the integral as it crosses the desired trajectory to prevent this from occurring. Derivative noise could arise when taking the measurement of speed and acceleration to calculate the derivative terms. As mentioned in part 2.2, by using an IMU high precision measurements can be taken an incorporated to the calculation of the PID terms. Furthermore, to account for noise in the data, by using a low pass filter we could reduce the severity of noise on the data.

Generally, the introduction of hardware components and measuring uncertainties requires the adaptation of software that ensures data from various sensors are acquired in the proper order and calculations are optimized to environmental conditions.

References

[1] "ESC204 Praxis III project guideline." Toronto.

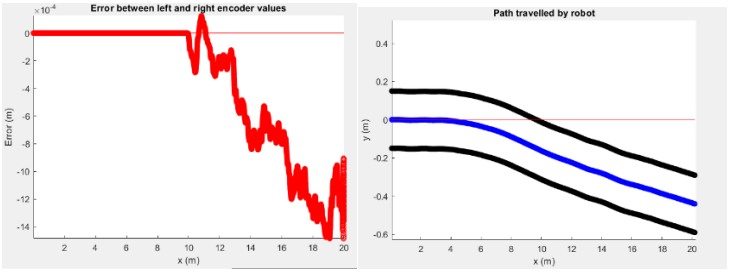
[2] G. Dockery, "How to Calculate a Pulse Width," *Sciencing*, 02-Mar-2019. [Online]. Available: https://sciencing.com/calculate-pulse-width-8618299.html. [Accessed: 04-Apr-2020].

[3] Toledo, Piñeiro, Rafael, Acosta, Daniel, Acosta, and Leopoldo, "Improving Odometric Accuracy for an Autonomous Electric Cart," *MDPI*, 12-Jan-2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/1/200. [Accessed: 04-Apr-2020].

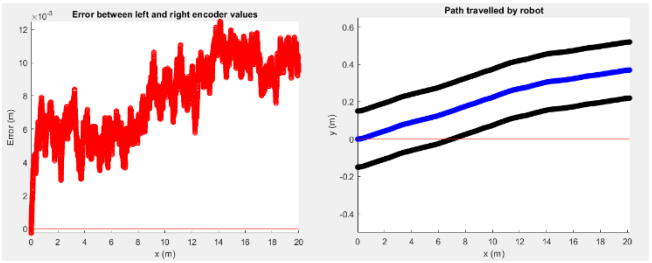
[4] C. Bagwell, "Robot localization using an IMU with PID control," *Mbed Operating Systems*, 20-Oct-2015. [Online]. Available: https://os.mbed.com/users/cardenb/notebook/robot-localization-using-an-imu-with-pid-control/. [Accessed: 04-Apr-2020].

[5] V. Vandoren, "Understanding PID control and loop tuning fundamentals," *Control Engineering*, 26-Jul-2016. [Online]. Available: https://www.controleng.com/articles/understanding-pid-control-and-loop-tuning-fundamentals/. [Accessed: 04-Apr-2020].

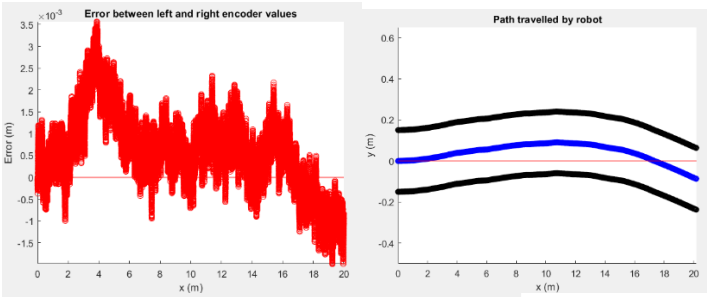
[6] M. Tham, "Discretised PID Controllers," *Google Groups*, 06-Nov-2014. [Online]. Available: https://www.shorturl.at/bflN8

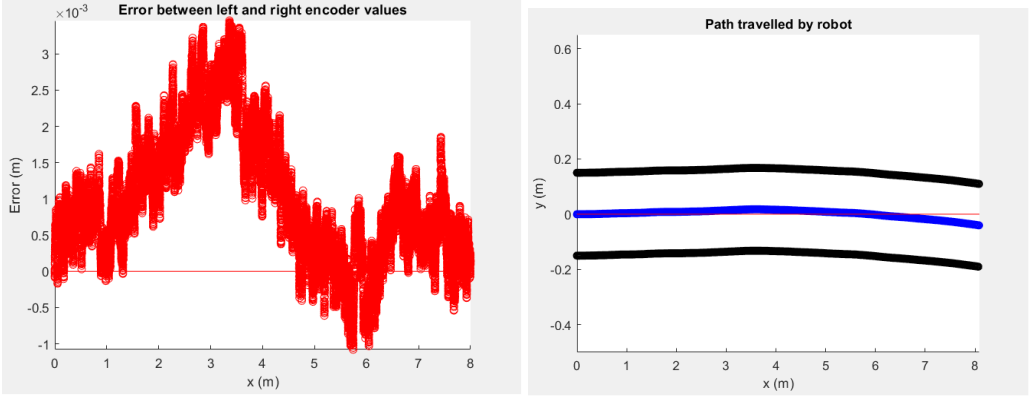

Error between left and right encoder values
Error (m)
x (m)
Path travelled by robot
y (m)
x (m)

K = 100

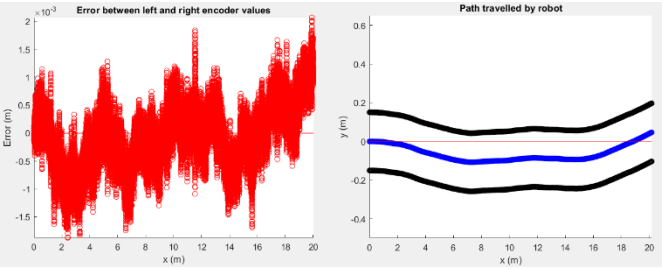

Error between left and right encoder values
Error (m)
x (m)
Path travelled by robot
y (m)
x (m)

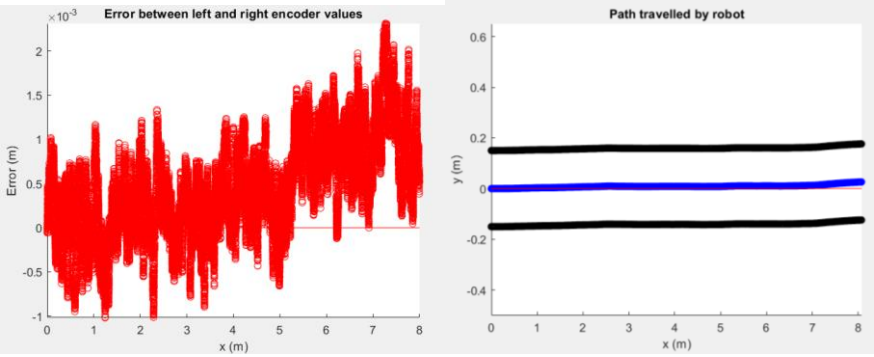
K=1


Error between left and right encoder values
Error (m)
x (m)
Path travelled by robot
y (m)
x (m)


Error between left and right encoder values
Error (m)
x (m)
Path travelled by robot
y (m)
x (m)

K=10


Error between left and right encoder values
Error (m)
x (m)
Path travelled by robot
y (m)
x (m)


Error between left and right encoder values
Error (m)
x (m)
Path travelled by robot
y (m)
x (m)

K=100

K=25