6051 MAA Individual Project

Motion Control, Line Following, Obstacle Detection and Pre-Planned Path Navigation of a Wheeled Robot in a Constant Environment using Dead reckoning, on/off and PID controller Algorithm.

Name: Ravi Chaudhary (10947986)

Supervisor: Dr Khaled Al Khudir

Report submitted to the school of Mechanical, Aerospace and Automotive Engineering, Coventry University in partial fulfilment of the requirements for the degree of Bachelor of Engineering.

April 2023

Abstract

Effective path navigation requires the integration of various sensing, planning, and control techniques, and has been the subject of extensive research in robotics. To thoroughly examine and has a comprehensive understanding of various aspects of path navigation, a Simulink study has been conducted on a wheeled robot. To analyse the results in the Simulink environment, the real wheeled robot parameters as well as the encoder sensor parameters has been considered through a study and certain calculations on its velocity, distance travelled has been performed. Three of the common algorithms has been utilized (Dead Reckoning, on/off and PID control algorithm) for the robot's motion control, line following and the obstacle detection. The motion control of a robot was performed in an open loop system, close loop system and utilizing motor physics in a closed loop system. Incorporating motor physics into the study led to more accurate results that closely resemble real-world scenarios. The line following and the obstacle detection involves line sensor and ultrasonic sensor respectively. The line sensor was used to identify the environment and the path through the colours in the map to follow the line path and the ultrasonic sensor was used as a feedback sensor which measures the wave reflected from the obstacle surface to give the value corresponding to the distance of an obstacle. Overall, the PID control algorithm was found better over the Dead reckoning and on/off algorithm for the path navigation process. Therefore, eventually the pre-planned path navigation has been performed combining the motion control, line following and the obstacle detection using PID control algorithm together with Dead Reckoning. The robot successfully followed the line and stopped before the certain distance from the obstacle smoothly.

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to my respected supervisor, **Dr. Khaled Al Khudir**, for his invaluable supervision, unwavering support, and determined commitment throughout my final year. His expert guidance, diligent efforts, and timely responses were all instrumental in the successful completion of this project.

I am also grateful to my parents who stood by me during challenging times throughout my academic journey. Their assistance and contribution were also crucial in ensuring the successful completion of this project.

Contents

List of Figures:

1. Introduction

Path navigation is the process of finding an optimal path from a starting point to a desired goal while avoiding obstacles in between. It means the path navigation involves several processes such as detecting the obstacle and following the path using different sensors. It is one of the essential components of autonomous navigation. Since autonomous navigation could be tele-autonomous to fully autonomous, this thesis covers the path navigation in the fully autonomous navigation spectrum. The full autonomy is again classified into two approaches:

1) Heuristic approach: It cannot ensure an ideal outcome, but it is sufficient to meet some immediate objectives. For example: Maze solving vehicle, robotic vacuum.
2) Optimal approach: Usually, it requires more information about the environment. The robot creates a model of the surrounding or updates the one that has been provided, then determines the optimal path to follow in order to reach the desired goal **(Douglas, MATLAB, 2020).**

The constantly changing environment and the obstacles makes the path planning and navigation difficult. In order to create a plan, it has to build up the model of the environment overtime and the model needs to be constantly updated due to constantly changing environment.

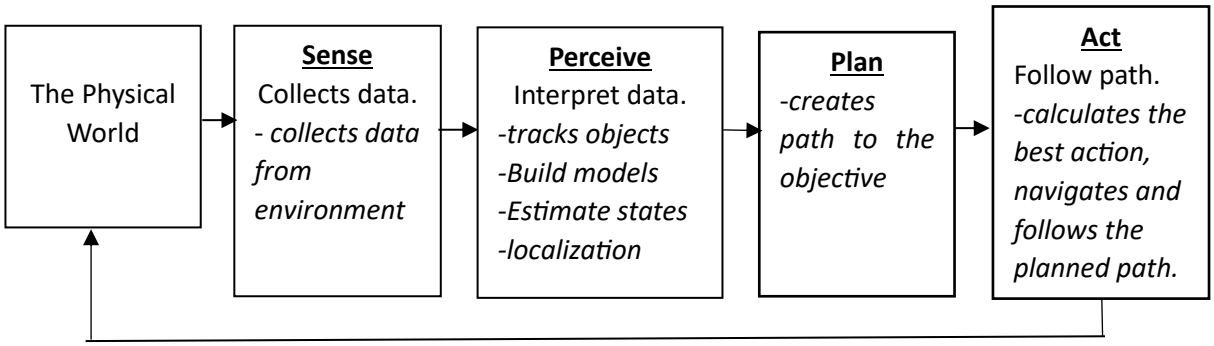


Figure 1: *steps of autonomous navigation.*

To plan and navigate the path, the robot first needs to identify its position, which is done using a **particle filter** that can handle non-Gaussian probability distributions. If a trustworthy map is available, the particle filter can be used for localization. Otherwise, **SLAM (Simultaneous Localization and Mapping)** is used to simultaneously build a map and determine the robot's position within it in an environment that is constantly changing **(Douglas, MATLAB, 2020).**

SLAM using Pose Graph Optimization:

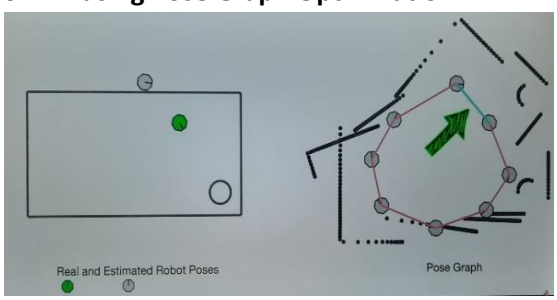


Figure 2: *Pose graph created by the estimated. Robot moves.*

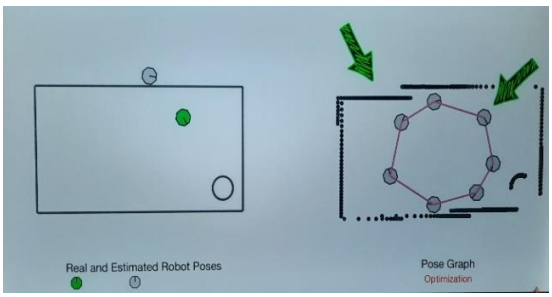


Figure 3: *Correction of Pose graph.*

As shown in **fig 2 and 3,** the optimization part of the pose graph optimization involves creating a loop closure between the estimated poses of the robot and settling the pose graph to equilibrium to

balance all the forces imposed by the constraints between each estimated robot pose. By optimizing the pose graph, the current pose and model of the environment can be estimated more accurately, while also updating past poses. The model of the environment can be further improved by introducing more loop closures.

After creating a pose graph and optimizing it, a map of the environment is built with all these data points using **Binary Occupancy Grid**. Then path planning is used to determine an optimal path from the robot's current position to a desired goal location, taking into account obstacles and other constraints. Navigation techniques are used here to enable the robot to follow the planned path and reach the goal location. This may involve using sensors to detect the robot's position and orientation, as well as obstacles in the environment, and adjusting the robot's trajectory or speed as needed to stay on the planned path **(MATLAB, 2020).**

The thesis will be focused on navigation techniques; line following, obstacle detection and motion control individually in a Simulation environment using different sensors and algorithms to follow the pre-defined path in a constant environment. Accuracy and smoothness will be observed due to the application of different algorithms such as PID control, Dead Reckoning, and on/off algorithm.

1. Literature review

Among different literature sources, one journals' articles and conference proceeding paper have been described in this literature study which are related to path navigation techniques in a mobile robot using different algorithms.

One of the studies were conducted on Dead Reckoning Navigation technique. The article describes a Dead Reckoning Navigation system for autonomous mobile robots that uses sensor fusion with a Kalman filter to estimate position and heading. The experiment was created using "Seoul National University Mobile Robot" which was performed in a typical indoor environment. The robot uses encoders, accelerometers, and gyroscopes to measure displacement, heading angle, and angular velocity of the robot along with the use of dead reckoning algorithm. The encoder errors were found to be the primary source of error, which were either systematic or non-systematic. Systematic errors include unequal wheel diameters, difference between actual and nominal wheelbase, and misalignment of wheels, while non-systematic errors include travel over uneven floors, unexpected objects on the floor, and wheel slippage. The experiment considered only systematic errors since non-systematic errors are unpredictable. The experimental model assumed that the mobile robot moves in two-dimensional smooth indoor environments. The gyroscope provides independent heading information but was also subject to errors such as bias and scale factor error. The experiment also involved the observability analysis aimed to determine how much the system could exclude errors and estimate states such as position and heading angle. The system was approximated as a piece-wise time invariant system, and the rank of the total observability matrix was investigated. The results showed that the Dead Reckoning Navigation system using the encoder and the gyroscope had unobservable states such as X, Y position error, and heading angle error, and therefore, the designed Kalman filter could not completely eliminate these errors. Other states such as the left and right encoder scale factor error, the wheelbase error, the gyro bias, and the gyro scale factor error were observable, and could be estimated precisely by filtering. The trajectory used in experiment can be seen in **fig 4** and was obtained through measurements conducted in experiment.

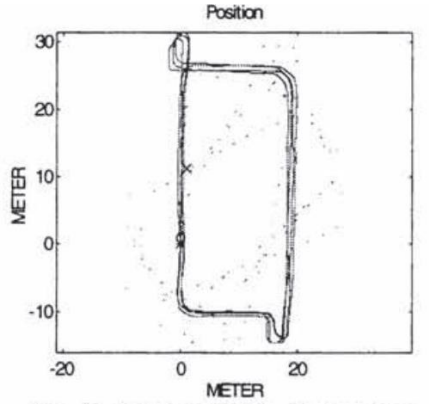


Figure 4: *Trajectory obtained through experiment.*

The result obtained from the proposed navigation algorithm (Dead reckoning navigation system with Kalman filter) and the encoder-only navigation algorithm were compared. The mobile robot travelled 490 m for 20 minutes, with an average velocity of 0.4 m/sec. The proposed navigation algorithm was found to estimate the position and heading angle more precisely than the encoder-only navigation algorithm, which produced a wrong position and heading angle after the run. The state variances of the Kalman filter were shown in fig 5, with the position and heading error variances having increasing characteristics. However, the error variance of other states (sensors, gyroscope) of the system converged as shown in **fig 5.**

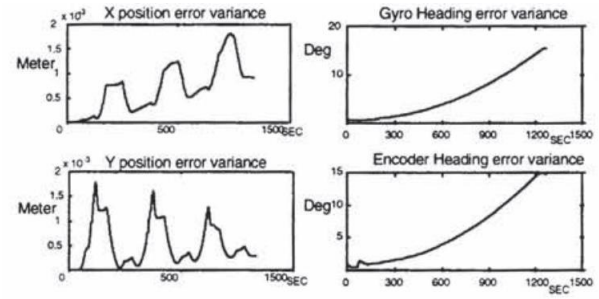


Figure 5: *Position and heading angle error variances of the navigation system.*

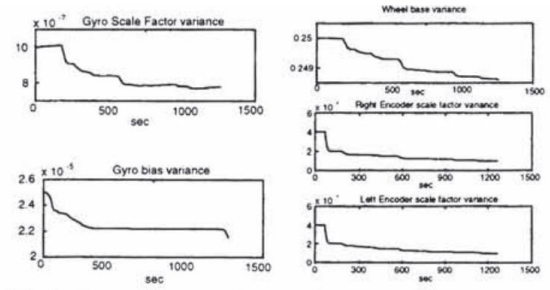


Figure 6: *Sensor error variance of the navigation.*

Concluding the experiment, the Dead Reckoning Navigation system compensated for encoder and gyroscope errors using a Kalman filter, which estimates and compensates for both systematic and stochastic errors of encoder and gyroscope respectively. The Kalman filter was able to combine encoder and gyroscope measurements to compensate for errors. Additionally, the system still produced unfiltered position and heading angle even if the filter fails. The observability analysis of the dead reckoning navigation system showed that the system can provide reliable position and heading angle for a limited time. However, an absolute positioning system was required to compensate for navigation errors and prolong navigation time and distance. Experimental results demonstrated that the proposed mobile robot navigation system can provide reliable position and heading angle without external positioning aids only for a certain period of time. **(KyuCheol Park 1, Dead Reckoning Navigation For Autonomous Mobile Robots, 1998)**

Another study was done on a paper which conducted the experiment on the three wheeled robot for the line following. An ARM cortex-3 based microcontroller and infra-red line sensors was used in the robot to navigate in partially structured environments. The analysis was done using on/off control and

a dynamic PID control algorithm was proposed to improve navigation reliability, and experimental results. The line follower robot uses IR sensors to detect the path and direction, with a suitable distance between the sensors and ground surface, and between each sensor depending on the line width. The received analog signals from the sensors are converted to digital form using a signal processing circuit and sent to the microcontroller directly. The robot in this experiment used the TI Stellaris LM3S811 microcontroller with internal oscillators, timers, UART, USB, SPI, PWM, ADC, analog comparator, and watchdog timers. The IC L298 was chosen as a motor driver to control two motors, with a current capacity of 2 amps per channel and 45 volts output. Gearbox motors were chosen over DC motors, and three wheels were used, with two motors and wheels at the rear and a passive caster wheel at the front. The robot was powered by a 7.6V regulator and controlled by the microcontroller, which received signals from eight infrared sensors and sent appropriate signals to the driver IC to change motor direction. The microcontroller processes data from sensors and sends instructions to the driver, which powers the motors according to inputs.

Initially a Quadratic-line detection algorithm was used for detecting the position of a black line in a map. Eight optical sensors were used, and the correct position of the line was determined by locating three consecutive sensors with higher output readings than the other five sensors. The position of the line was calculated using the coordinate value at which the output value of the quadratic curve was the maximum. The error between the line position and the centre position of the robot was then calculated as e = -x.

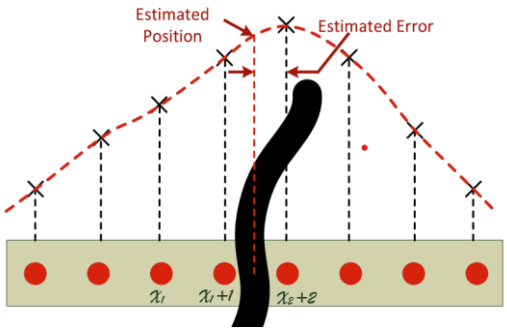


Figure 7: *The line detection algorithm via quadratic interpolation.*

Later the PID controller was introduced to accurately follow a racing track by processing the error between the sensors and the track. The controller calculated the current position and generated velocity commands for the wheels based on proportional, integral, and derivative controls. The proportional control determined the magnitude of the turn, while the integral control increased the magnitude over time until the robot centres over the line. The derivative control reduced the oscillating effect over time. However, setting the KP, Ki, and KD factors required manual trial and error.

Overall, an experiment was conducted to compare the performance of a line following robot controlled by a simple on/off control and a dynamic PID algorithm. The results showed that the dynamic algorithm controlled robot outperformed the simple control robot in all criteria that is listed in the **table 1.** The dynamic PID control algorithm of robot produced higher velocity, smoother tracking, and lower tendency to deviate from the line. The experimental results due to on/off algorithm and the Dynamic PID algorithm has been presented in the table below. **(Engin1, 2012).**

Table 1: *Experimental result obtained through Line Following Robot experiment* ***(Engin1, 2012)****.*

Criteria	Dynamic PID algorithm	Simple (on/off)
Time to complete one whole circuit	47.6s	71.4s
Line tracking	Smooth	Not so smooth
Velocity	0.2m/s	0.14m/s
Tendency to astray from line	Low	High

1.1. Concluding literature review

The literature review describes the different experiment done on the navigation system for a mobile robot. The 1st literature was focused on the Dead Reckoning Algorithm that uses a Kalman filter to estimate position and heading distance by fusion encoder and gyroscope data. It was found that the dead reckoning navigation system compensated for errors using a Kalman filter, but an absolute positioning system was required for longer navigation periods. The system was able to provide reliable position and heading angle only for a limited time without external aids. Another study on the line following by a mobile robot conducted an experiment and compared the result obtained using PID algorithm and on/off algorithm where the PID controller algorithm was found to be smoother, quicker and has low tendency to astray from line during line following. Overall, the PID controller algorithm was found better for precise navigation techniques over Dead Reckoning Algorithm and On/Off algorithm. The similar study will be conducted in a Simulink using Dead Reckoning, On/Off algorithm and PID control algorithm to better understand the navigation techniques in mobile robot.

1.2. Hypothesis

Since the methodology of this project is based on navigation techniques like motion control, line following and obstacle detection of a wheeled robot using all these three algorithms: Dead Reckoning, on/off and PID control, it can be hypothesized that the PID algorithm will produce more accurate results in term of distance covered and smoothness. Unlike in literature, the Dead Reckoning algorithm won't be combined with the Kalman filter rather the closed-loop system will be generated using encoders. Therefore, the results have been expected okay but not as precise as the results generated by the PID controller as the controller has the ability to mitigate the errors and disturbances.

2. Methodology

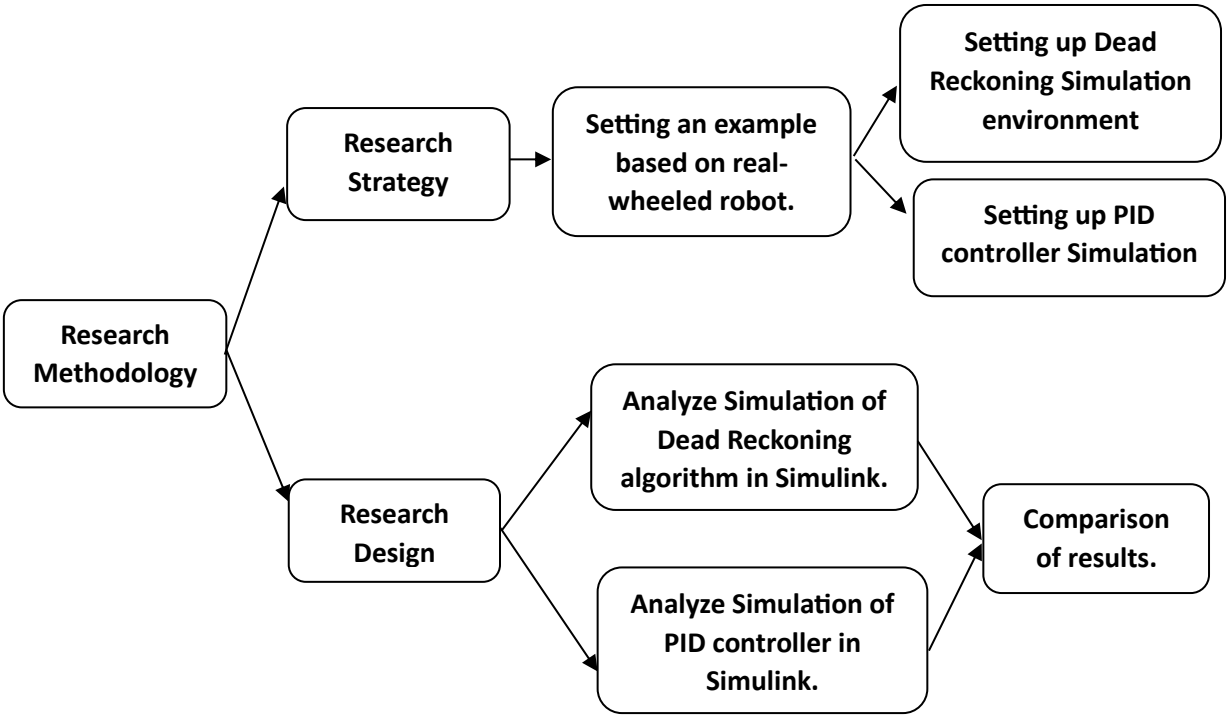


Figure 8: *Strategic Plan Chart.*

2.1. Dead Reckoning Algorithm

Dead Reckoning also termed as on/off control is the method of estimating current position using a previously established reference position and forwarding that position based on known or anticipated speeds over elapsed time and course, distance calculation and direction traveled. This method, despite giving accurate positional information, is susceptible to inaccuracies due to factors like speed inaccuracy or direction estimates. Since the new projected value will have its own errors in addition to being based on the previous location, which had flaws, the errors will also be cumulative. Dead reckoning doesn't necessarily require the use of sensors for navigation, but it can make use of sensors like accelerometers, gyroscopes, and magnetometers to estimate the distance and direction of movement and increase accuracy and effectiveness **(Stefan Edelkamp, 2012).**

While the Dead Reckoning's outcome isn't particularly accurate, it can be combined with a PID controller to produce better results.

In this project, the Dead Reckoning Algorithm is also called as an on/off algorithm because the robot's position and movement are estimated using the Dead Reckoning algorithm, while binary decisions are made using an on/off algorithm that compares sensor measurements to a threshold value. For example, the robot may stop at a specific distance from an obstacle using the on/off algorithm. The two algorithms work together to specify the robot's velocities and states.

2.2. PID Control Algorithm

PID controller is the feedback mechanism highly used in the control systems to control a process by continually updating the output based on the difference between the desired outcome and the measured process variable. PID controller uses Proportional, Integral, and Derivative terms to calculate the output signal. The control and navigation of mobile robots can be enhanced by combining PID controllers and dead reckoning. A PID controller can improve the accuracy of the robot's control by adjusting the robot's movement based on its predicted position and velocity utilizing the data provided by dead reckoning. Dead reckoning and PID controllers can be integrated in the following ways to achieve improved results.

- **Incorporating dead reckoning into the PID control loop:** The PID controller modifies the output signal to the robot's motors based on the estimated position and velocity provided by dead reckoning. This can enhance the robot's movement precision by compensating for errors in its estimated position.
- **Using PID control to rectify dead reckoning inaccuracies:** Dead reckoning estimates are susceptible to cumulative inaccuracies, which over time may result in the robot's estimated location deviating noticeably from its actual position. The robot's mobility can be adjusted using PID control to correct these errors based on the discrepancy between the robot's estimated and actual position**. [(Songxiao Cao, 2022)].**

2.3. Defining a real wheeled robot example with calculation

Assume, the encoder (indicated by red line) is attached to the wheel motor shaft as shown in **fig 9.**

- **1 full revolution of a wheel = 360˚**

The encoder counts the discrete number of ticks corresponding to the rotation. For a specific example, **E6B2-C** encoder sensor has been chosen with the number of ticks per rotation is 10. The tick count per rotation for a certain encoder sensor is constant and can be found in the manufacturer's datasheet.

- **1 rotation = 360 ticks** [https://docs.rs-online.com/7639/A700000007380478.pdf] **(OMRON, n.d.)**

- $$\boldsymbol{Number\ of\ Wheel\ rotation} = \frac{\boldsymbol{Total\ Encoder\ ticks}}{\boldsymbol{Tick\ Count\ per\ Rotation}} \dashrightarrow \boldsymbol{equation\ 1}$$

Figure 9: *robot wheel with encoder attached.*

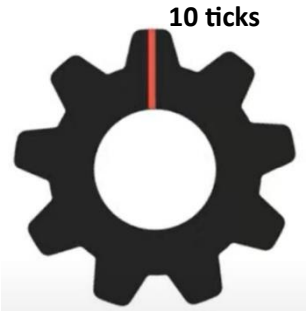


Figure 10: *9 ticks per rotation of a wheel*

The encoder will be used as an odometer to determine the distance travelled by the robot wheel by multiplying the rotation of encoder to the circumferential distance of the wheel.

- **1 complete rotation = 1 circumferential distance** = $2 * \pi * R$

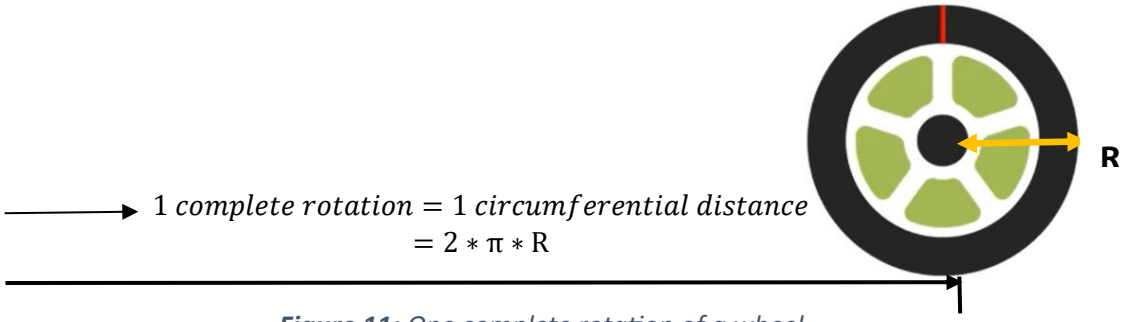


Figure 11: *One complete rotation of a wheel.*

Here,

- **R = radius of a wheel**

The total distance travelled by the wheel is the number of wheel rotation multiplied by its circumferential distance. i.e.,

- $\boldsymbol{Distance\ travelled} = \frac{\boldsymbol{Total\ Encoder\ ticks}}{\boldsymbol{Tick\ Count\ per\ Rotation}} * \boldsymbol{2 * \pi * R} \qquad \boldsymbol{(from\ equation\ 1)}$

The circumference of a specific wheel in a robot depends on the radius of a wheel and therefore, it remains constant. Similarly, the ticks count per rotation is also constant. The only variable in the above formula is the total number of encoder ticks which depends on the rotation of the wheel.

The common size of the robot wheel is considered for this example analysis i.e.,

- **Diameter of wheel (D) = 160 mm** **(IQ, n.d.)**
- **Radius of wheel (WheelR) = 80 mm = 0.8 m**

Since, distance travelled by the robot for 1 rotation of a wheel is,

⇨ $\boldsymbol{Distance\ travelled} = \boldsymbol{(1) * 2 * \pi * R = 0.502\ m}$

2.4. Implementation of Odometer in Simulink

The distance travelled by one of the wheels of the robot is found to be 0.502 m. The distance travelled by the robot as a whole can be found using the formula:

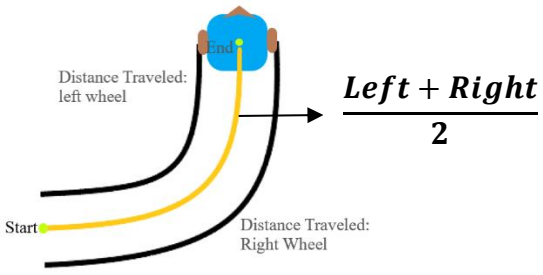
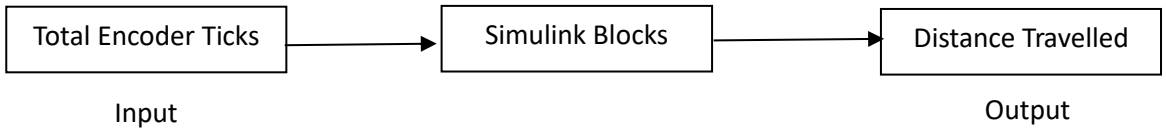


Figure 12: *Distance travelled by a whole robot.*

$$\textbf{distance travelled by robot} = \frac{\textbf{Left wheel} + \textbf{Right wheel}}{\mathbf{2}}$$

The distance travelled by the robot will be analyzed in the Simulink environment setting up an odometer. The general idea to implement the odometer in the Simulink has been shown below in the block diagram.



2.4.1. Simulink Model showing a odometer setup

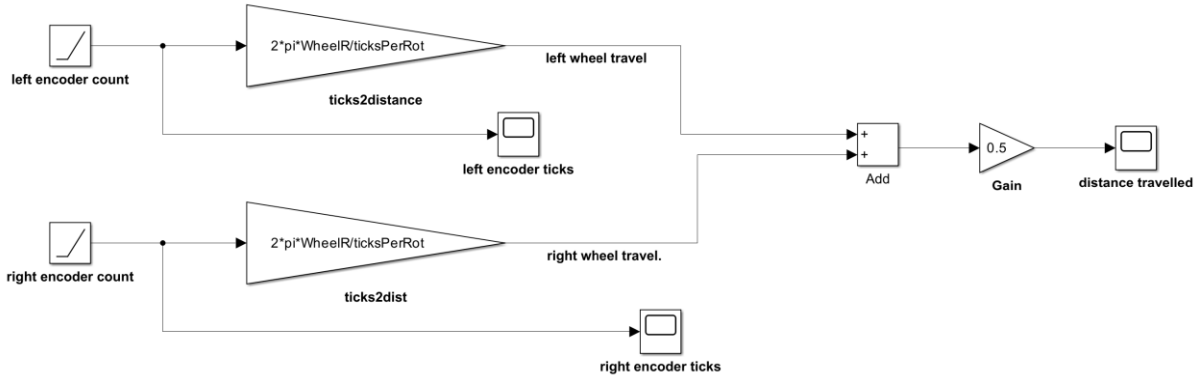


Figure 13: *Simulink model used as an odometer using encoder sensors.*

Model: It shows the graphical modelling environment to build an odometer from the encoder sensor inputs.

- Ramp block is used as a representative of a robot behavior moving in a straight line.
- The gain block is used to change the encoder ticks to distance travelled.

2.4.2. Results of odometer setup in Simulink

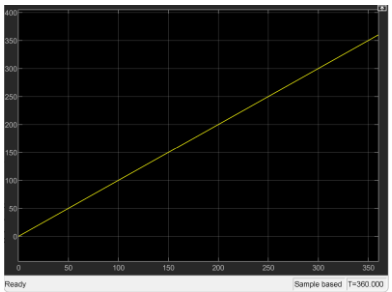


Figure 14: *left wheel rotation.*

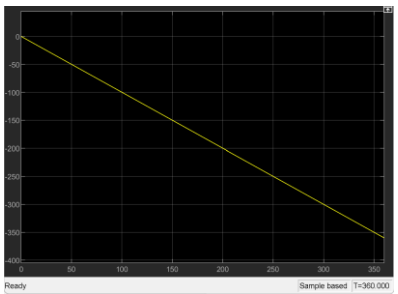


Figure 15: *Right wheel rotation.*

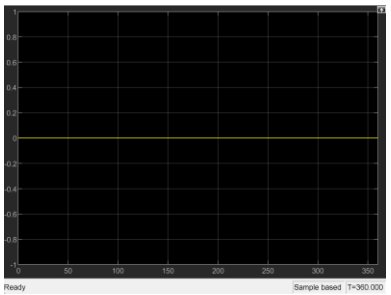


Figure 16: *Distance travelled by Robot*

It can be seen that the left and right encoder scopes **(fig 14 and 15)** are non-zero values indicating that the wheels are rotating. The left encoder shows the positive rotation of the wheel, and the right encoder shows the negative as the value was set to be negative. However, the distance travelled is zero as the robot technically isn't moving and the wheels are just rotating in one place.

3. Robot Motion Control

The objective of motion control is to observe and analyze the robot how smoothly and how close or far it stops from the reference distance using the Dead reckoning and PID controller algorithm.

3.1. Motion Control using Dead Reckoning Algorithm

The Dead Reckoning algorithm will be used to move the robot to **2 m distance** and stop its movement once it reaches to the target distance.

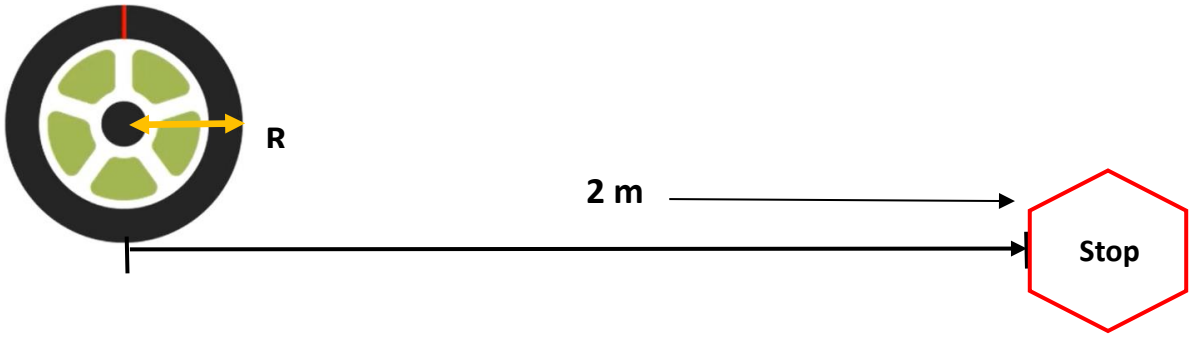


Figure 17: *showing the distance between starting and ending point.*

3.1.1. Logic to move the robot in Simulink.

The robot will be moved forward controlling its two variables. The linear velocity **'v'** which represents the speed of the robot's movement in a straight line and the angular velocity '**Ω**' that represents how fast the robot is rotating in one place. Since, the target is to move the robot in a straight line, the value of angular velocity will remain 0. The state flow chart will be used to build the logical part of the algorithm and will be added to the previous Simulink model to observe the results.

3.1.2. Simulink Model for the motion control using Dead Reckoning

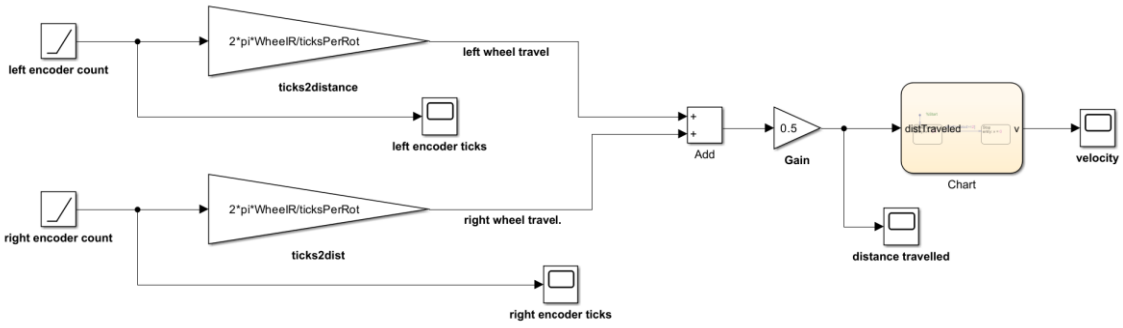


Figure 18: *Simulink model with integrated state flow chart.*

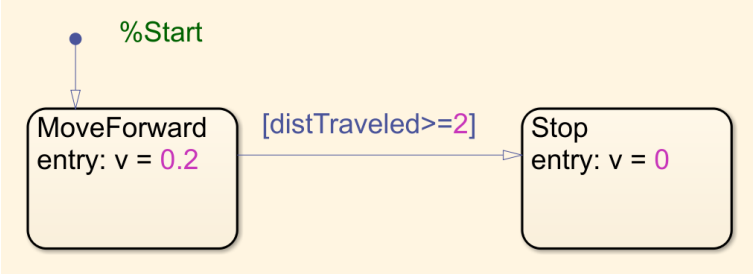


Figure 19: *State flow chart describing two modes of operation.*

Simulink Setup: In the state flow chart, the state execution statements are defined i.e., 'move forward' and 'stop' state. The velocity of the robot movement is set to be **0.2m/s** and its stopping velocity is **0 m/s.** Since, the target is to move **2 m**, the transition condition between the move forward state to the stop state has to be greater or equal to 2. **This setup algorithm is called dead reckoning because the transition keeps happening in between the defined states till the objectives is made and it is only based on the specified variables in the state space with no controllers being used.**

3.1.3. Results of Motion Control (distance and velocity) using Dead Reckoning

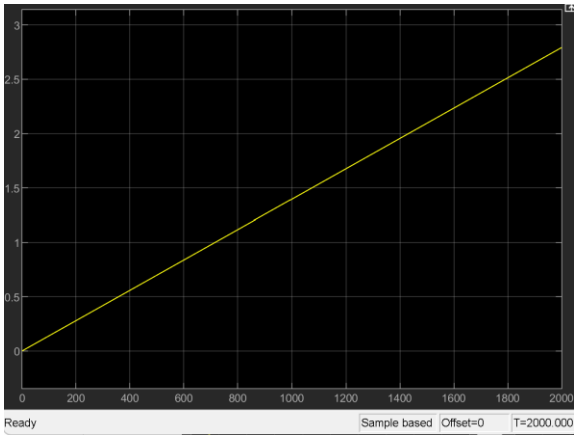


Figure 20: *Distance travelled by robot.*

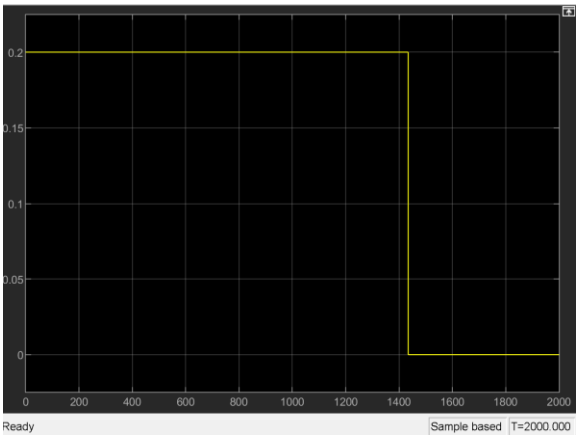


Figure 21: *Velocity transition of a robot.*

It can be observed that the distance travelled by the robot has gone beyond 2 m. similarly, the velocity of the robot falls to zero at the sampling time close to 1400 sec. It assures that the setup algorithm is functional and now the challenge is to stop the robot exactly at it reaches the distance of 2 m.

So far, the input sensor used is just a simulated value and the system describes the open loop as there is no connection established between the robot velocity and the input encoder value. The mobile robotics training library will be used further to build a closed loop system and to visualize the robot's given left and right wheel velocities.

3.2. Motion Control using Dead Reckoning in a closed loop system:

3.2.1. Closed loop Simulink model of Dead Reckoning algorithm

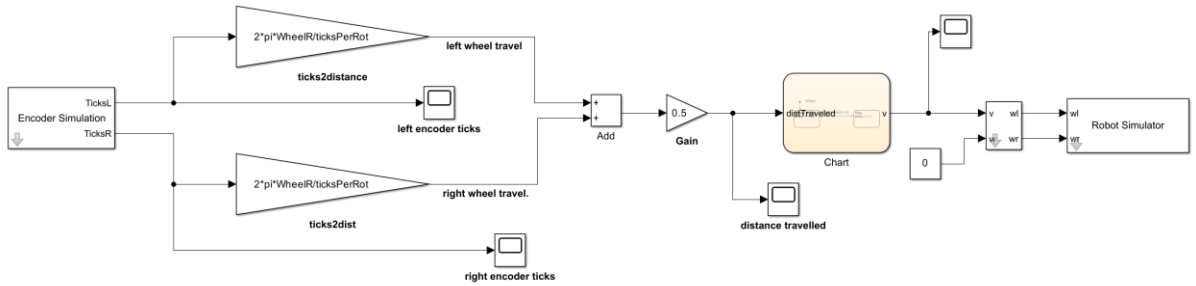


Figure 22: *Closed loop Simulink model to observe the distance travelled and velocity of the robot.*

Simulink setup: To establish a closed-loop system the robot encoder block was used instead of ramp block. The simulator block was used to visualize the robot's left and right wheel velocities. The simulator takes in left and right wheel angular velocities as inputs. The output of the state flow chart however is a linear velocity. Therefore, it needs a conversion where the utility blocks come in use which converts linear velocity output to the angular velocity.
The initial position of the robot was set to be (1,1) at 0˚ angle in a pre-generated map of a robot simulator block. Therefore, the commencing point of the robot will be (1,1) in a map.

3.2.2. Results obtained using Dead Reckoning in a closed loop system

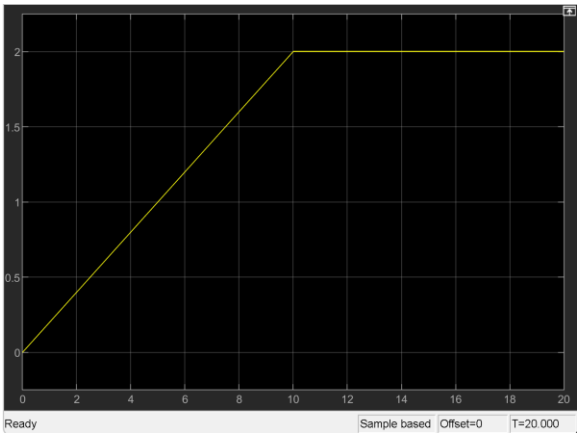


Figure 23: *Distance travelled by the robot.*

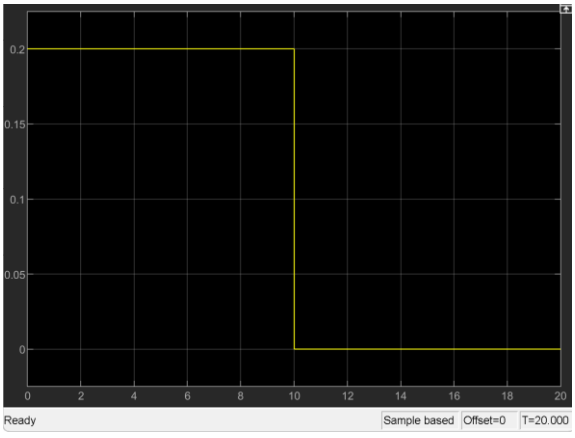


Figure 24: *velocity of a robot.*

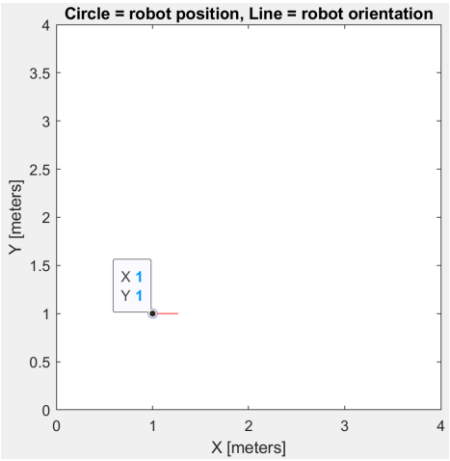


Figure 25: *Initial position of a robot during rest.*

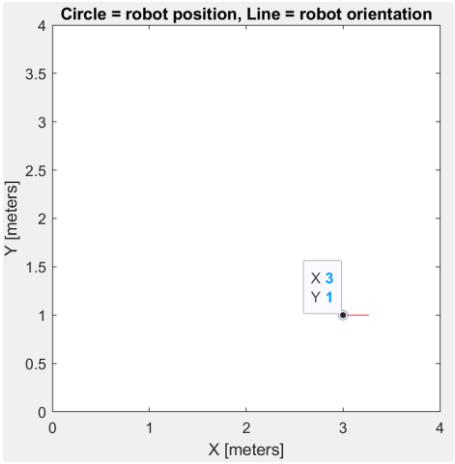


Figure 26: *Final position of a robot.*

Fig 25 and 26 shows the robot started moving horizontally from 1m mark and stopped itself at 3 m mark signifies that the robot successfully travelled 2 m distance and stopped moving once it reaches there. Looking at the distance travelled graph in **fig 23**, the robot tapered off near the distance of 2 m mark. Similarly, comparing the velocity graph with the distance travelled, it can be observed that the velocity of the robot dropped to zero as soon as the distance of 2 m is reached. Overall, the simulation of a closed loop system with the use of simulated sensors was helpful to fulfil the objective.

3.2.3. Concluding result obtained using Dead Reckoning in a closed loop system

It can be observed that the robot exactly travelled the distance of 2 m which is practically impossible. So far the required distance was specified in the state flow chart and the wheel velocity was directly fed to the simulation but there is no such factor specified which affects the input and overall result. It means the simulation done so far doesn't include the actual physics of the robot in the real world. The wheeled robot in the real world consists of motor physics from which the maximum and the minimum speed of the robot can be determined. Similarly, the motor itself has initial slips, slacks, noise, and friction due to which the exact expected result cannot be achieved.

Therefore, the further analysis will be done including the motor physics and the results will be compared between the Dead reckoning algorithm and the PID control algorithm.

4. Robot Motion Control (Use of Motor Physics)

To mimic the real hardware, the motor physics has been added to the simulation instead of directly feeding the wheel velocity to the simulator.

4.1. Robot Motion Control using Dead Reckoning (Use of Motor Physics)

4.1.1. Simulink Model representing Dead Reckoning and the use of motor physics

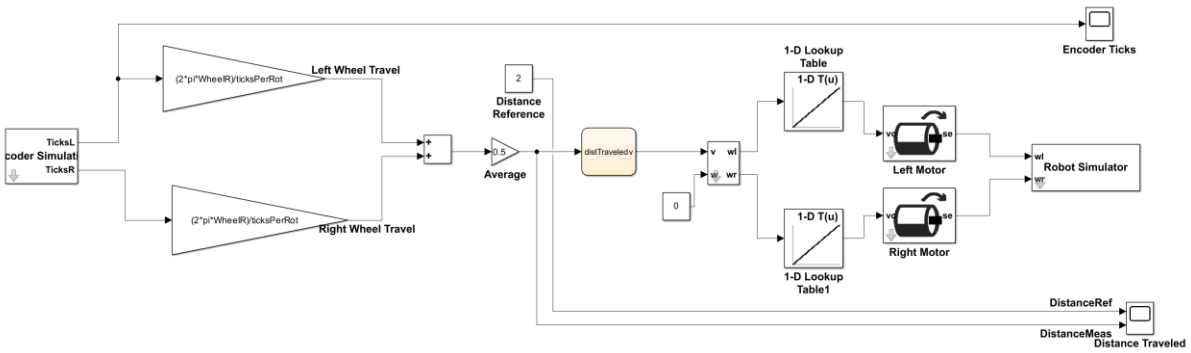


Figure 27: *Simulink model showing the use of motor physics in Dead Reckoning Algorithm.*

Simulink Setup: The parameters in the motor model describes the physics of the model. Therefore, the good motor model is required to design an accurate controller. For the particular analysis, the motor model, and the parameters of the "Vex motor" has been used which is developed by the Vex platform and is obtained from MathWorks website **(MathWorks, 2020)**. The output of the motor was specified as "position" rather than speed and it was because the objective was to measure the distance travelled which can be calculated as a difference between initial and the final position. Since, the output of the state flow chart was a linear velocity which was converted to the wheel velocity using the utility block (to wlwr) as described previously. The **lookup table** converts both the wheel velocities to a voltage that the motor can understand. The lookup table contains the voltage Vs speed characteristics for the motor and was obtained experimentally. The lookup table parameters used for this specific analysis was taken from **Mobile Robotics Training (MathWorks, MathWorks, 2020).**

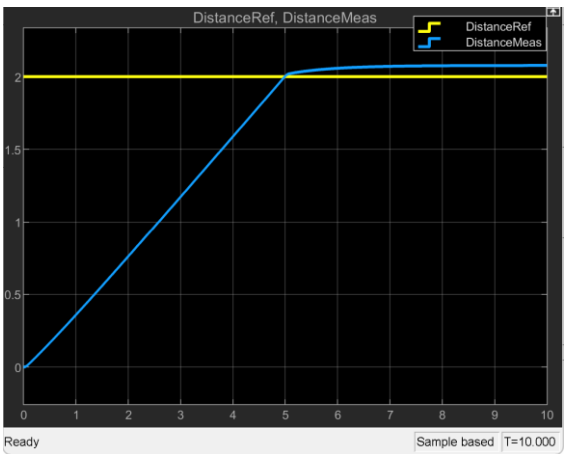


Figure 28: *Measured distance Vs Reference distance.*

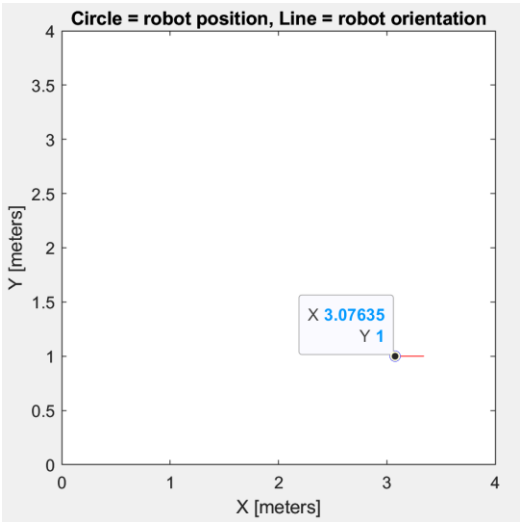


Figure 29: *final position of a robot.*

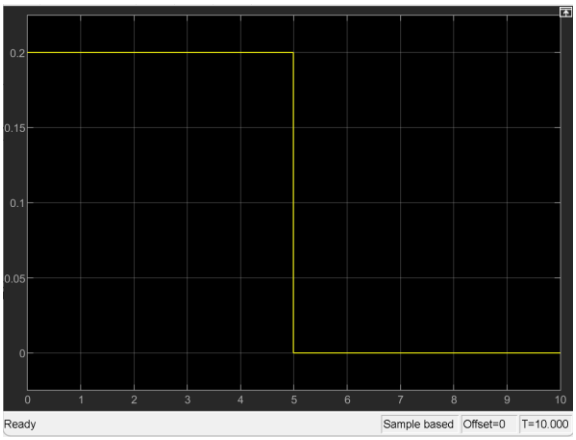


Figure 30: *maximum velocity of a robot.*

4.1.2. Results of Dead Reckoning using motor physics

It can be observed that the measured distance is beyond the reference distance. The robot covers the distance of 2 m at the sampling time of 5 sec. There is quite large steady state error between the reference distance and the measured distance as shown in **fig 28**. The **Fig 29** shows the robot exceeds the distance by 0.07 m. It seems like the robot didn't gradually slow down as it approached the destination.

4.1.3. Concluding results of Dead Reckoning using motor physics

Since the use of actual motor is affected by various external factors as described previously, the result was expected. But specifically in the simulation it can be said that the results are affected due to the dependence of Dead Reckoning algorithm on estimating the robot's position based on its previous position rather than any other external factors that can only be observed in the real world scenario **(Stefan Edelkamp, 2012)**. The inaccuracy in the reading the position overtime results the error as shown in **fig 28**. Similarly, the rise time of the robot is quite high as the Dead Reckoning algorithm doesn't consist of any controller to adjust the maximum and minimum velocity due to which the rise time of the robot depends only on the specific velocity provided. This is also the reason the maximum or minimum velocity it can go is only the specified velocity i.e., 0.2 m/s as can be seen in **fig 30.**

The simulation result of the Dead Reckoning algorithm (use of motor physics) in a wheeled robot looks like in **fig 31.**

Figure 31: *Demonstration of simulated result in a robot using Dead Reckoning algorithm.*

The actual result after applying in the real hardware can be expected as such in the **fig 32** and it is due to the influence of external factors.

Figure 32: *Demonstration of expected result in a real hardware using Dead Reckoning algorithm.*

4.2. Motion Control of a robot using PID control

The idea was to control the wheel velocity of the robot based on input error. The PID control can adjust its output based on the motor response such as speed capabilities, time delays or nonlinearities. This helps to compensate for any disturbance in the system improving the overall performance and stability of the control system **(Hardik S Jain, 2019).** The values of PID controller used will be adjusted individually and co-ordinately with P, I and D controller to achieve the stable and faster response.

4.2.1. Simulink Model representing PID controller for a motion control of a robot

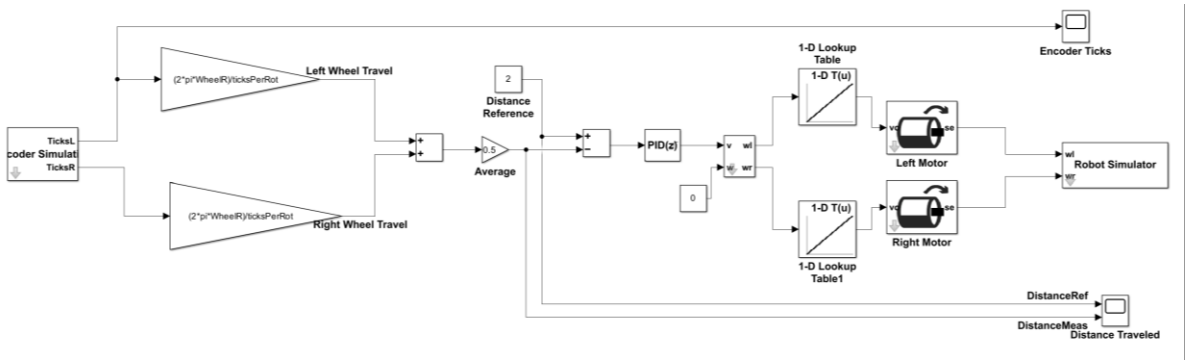


Figure 33: *Simulink Model with a PID controller integrated for a motion control.*

Simulink setup: The PID controller follows the similar procedure as Dead reckoning algorithm but instead of flow chart, the PID controller block is used which continuously functions to minimize the error. Since, the PID outputs the velocity of the robot "v", this needs to be converted to the wheel velocity using the utility block (to wlwr) as described previously. The **lookup table** converts both the wheel velocities to a voltage that the motor can understand.

The upper limit and the lower limit velocity of the robot needs to be specified in the PID controller to achieve the practical result. The maximum velocity that the robot can move depends on the specification of motor and the radius of robot's wheel which is calculated as following.

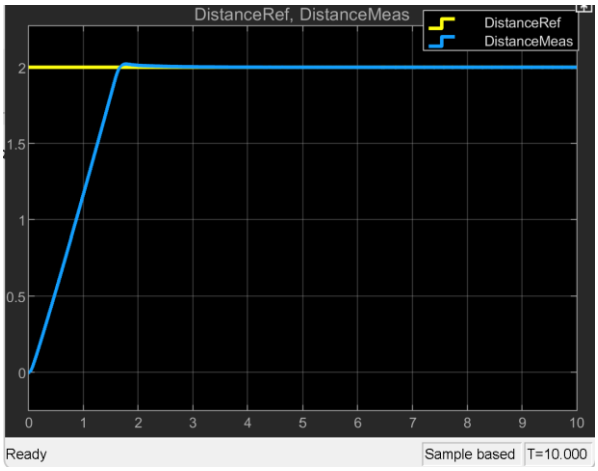
- $\boldsymbol{Max\ Velocity\ (v) = radius\ of\ wheel\ (WheelR) * max.angular\ velocity\ of\ motor}$

In the case of this analysis, the Vex motor has been used and it has the maximum angular velocity of 13 rad/sec **(MathWorks, MathWorks, 2020).** The radius of a robot wheel as specified previously is **0..08 mm**.

- $\boldsymbol{Max\ Velocity\ (v) = 1.04\ m/s}$

Therefore, the upper limit that the wheel can rotate at the velocity of 1.04 m/s and the lower limit is -1.04m/s and is set to the PID controller.

4.2.2. Results of a motion control of a robot using P only controller (P = 5.6)



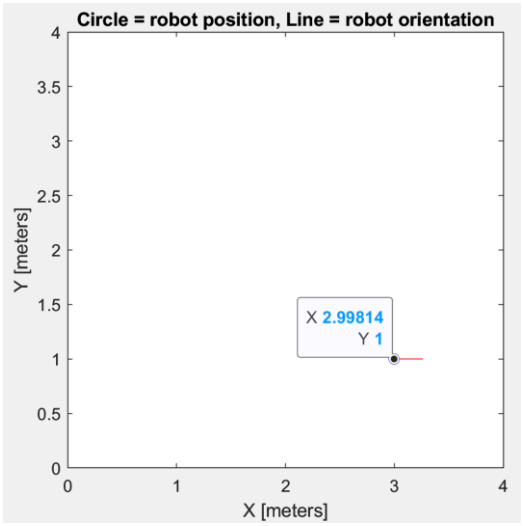


Figure 34: *Reference Vs measured distance using P = 5.6* ***Figure 35:*** *final position of a robot at p=5.6.*

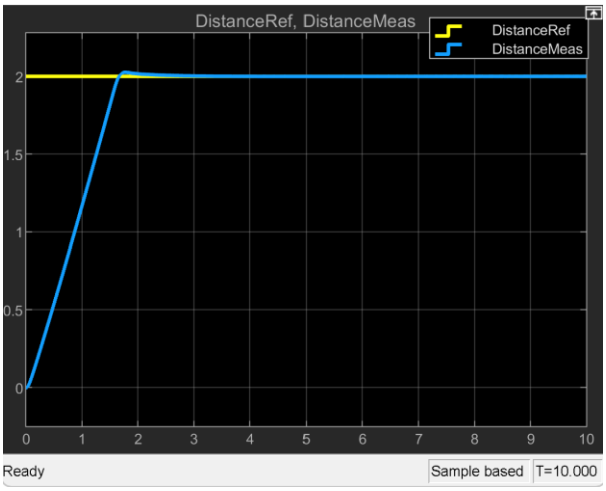
The graph in **fig 34** shows, the measured distance reached the reference distance at the fastest time of near 1.6 sec and settles down within a second. The tiny overshoot amplitude can be observed. Increasing the proportional gain further started oscillation making the response unstable. **Figure 35** shows the robot's final position whose starting position was (1,1) in the map and it showed that the robot moved the distance of 1.998m.

4.2.3. Concluding Results of a motion control of a robot using P controller

The result showed the smoother response of the robot as it approaches the distance of 2 m. The measured and reference distance obtained using P controller was much closer than the dead reckoning algorithm. Similarly, the robot approaches the distance faster as P controller used, functions to decrease the rise time. The P controller uses the feedback from sensors to continually adjust the robot's movement and errors which results the robot to stop gradually as it approaches the destination. But, according to the nature of proportional gain, the offset will always remain either

overshooting or undershooting the target meaning it tends to result in non-zero steady state **(ni, 2023).** This may not be noticeable in the simulation but has major effect in the real hardware. Therefore, the result was observed further using the integral term along with proportional as it can eliminate the steady state error.

4.2.4. Results of a motion control of a robot using P and I controller (P = 5.6 and I =2)



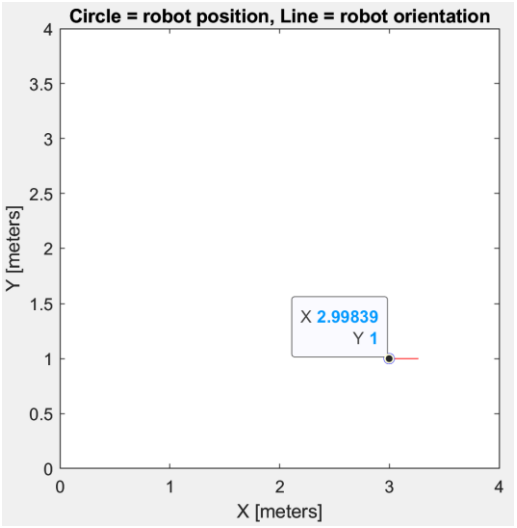


Figure 36: *Reference Vs measured distance using P=5.6 & I=2.* ***Figure 37:*** *final position of a robot*

Initially while the simulation was run, the integral exhibits the issue called whiner as it was the scenario of saturated output (output of the integral controller has reached its maximum limit i.e., 1.04 m/s in this case) which was accumulating the error values. It was solved changing the anti-windup method to clamping, which means the controller was instructed to calculate the error during whiner. The result obtained using the PI controller wasn't very different than the result obtained using P controller, but it can be assured that the PI controller will eliminate the steady state error quite well if tested in real hardware as compared to the P only controller. Still the 100% optimal result cannot be achieved due to the factors like initial friction, slip, slacks, noise, and robot weight as described earlier.

The simulated result using **P and PI controller** in a robot look like in **fig 38**

Figure 38: *Demonstration of simulated motion control result using Dead Reckoning algorithm.*

The actual result using **PI controller** in the real hardware may look like in **fig 39**, although it eliminates steady state error, the external factors affect the result in real hardware.

Figure 39: *Demonstration of expected motion control result in a real hardware using Dead Reckoning algorithm.*

The actual result of **P only controller** in a real hardware can be predicted as shown in **fig 40** as it tends to result in non-zero steady state error. It will be more likely to result in undershooting (not reaching the target) due to the simulation shows the robot covers the distance of 1.99839 m which is less than 2m.

Figure 40: *Demonstration of expected motion control result in a real hardware using Dead Reckoning algorithm (P=5.6).*

4.3. Overall conclusion of Dead Reckoning and PID control Results in Motion Control

The well optimized results have been achieved using the PI controller. The peak overshoot amplitude was very small. Therefore, there seems no need to use the Derivative controller along with PI controller which may reduce the overshoot amplitude but increase the rise time. The robot was lacking behind only the 10 mm to reach the destination although in real world it might be larger due to external factors, using PID controller either couldn't eliminate the error completely providing 100% optimal result. It is because the external factors such as robot's weight and friction are unable to be confirmed in the simulation as these characteristics aren't currently modelled in the simulation. Therefore, there might need of hardware trial and error to optimize the performance after the simulation result.

Similarly, the results achieved from the P and PI controller were far better than the Dead Reckoning algorithm. The P and PI results were smoother, quicker and produced closer response of a robot to the reference distance as the controller used in PID control eliminates disturbances and error whereas Dead reckoning algorithm is slow and has higher inaccuracy due to its dependency on the previous position measurement, a small error overtime increases inaccuracy.

5. Line Following using on/off control and PID Control

Line following is the fundamental to the path following of a robot. It basically uses the sensors that could be either the vision sensor, ultrasonic sensor, or the line sensor. The ongoing analysis for the line following will be based on the line sensors. The basic principle of the line sensors are they emit a light and measure light reflected from the surface. These sensors are also called the colour sensors. It detects the colour of the environment and the path which generally needs to be of different colour and according to the colour it has a certain range of sensor value through which it determines the line following **(Learn, n.d.).**

Procedure: The analysis was planned to be done in a black line (path) and the grey environment. The sensor value at those colours should be known. These values are obtained using a feature in a Simulink called external mode where the robot needs to be placed in an environment and in a line (path) respectively to measure the sensor value. Since, there is no actual hardware available for this analysis, the pre-measured sensor value in a grey-environment **(2950-2980)** and in a black-line **(2850-2880)** was used which was provided in MathWorks Mobile Robotics Training **(MathWorks, MathWorks, 2020).**

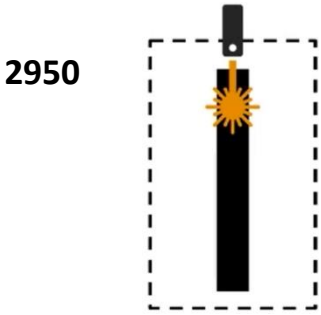


Figure 41: *Sensor value of a black line.*

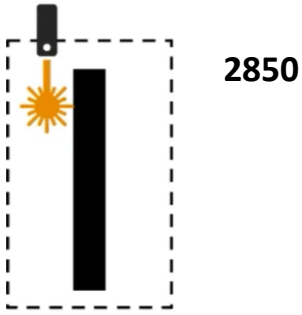


Figure 42: *Sensor value of a grey environment.*

The analysis will be carried out using only one sensor. So, there was possibility that the robot only follows the edge of the line (either the left edge or the right edge).Therefore, the line sensor was positioned in such a way that the half of it is on the line and half of it on the environment while moving. This was obtained computing a threshold value that lies between line and the environment which is the average of line and environment value.

$$\boldsymbol{threshold\ value} = \frac{\boldsymbol{Environment + line}}{\mathbf{2}} = \mathbf{2900}\ (\boldsymbol{in\ this\ case})$$

Simulation Setup: For the simulation, it requires a map where the robot can travel. The size of the map depends on the size of the robot as there should be sufficient space where the robot can travel. Therefore, the custom map of size **[3m x 1.6 m]** has been generated as the axle length between the wheels (width) was specified as 0.28 m earlier. Now it can be confirmed that the map generated has the enough space for the movement of a robot. The black and white version of the map as a mat. extension was used which clearly shows the line path as black and the environment as white as shown in **fig 43**. The mat. extension of the map generates the variable for simulation which was loaded in the Robot Simulator and the starting position of the robot in the map was specified which is marked in red in **fig 43**.

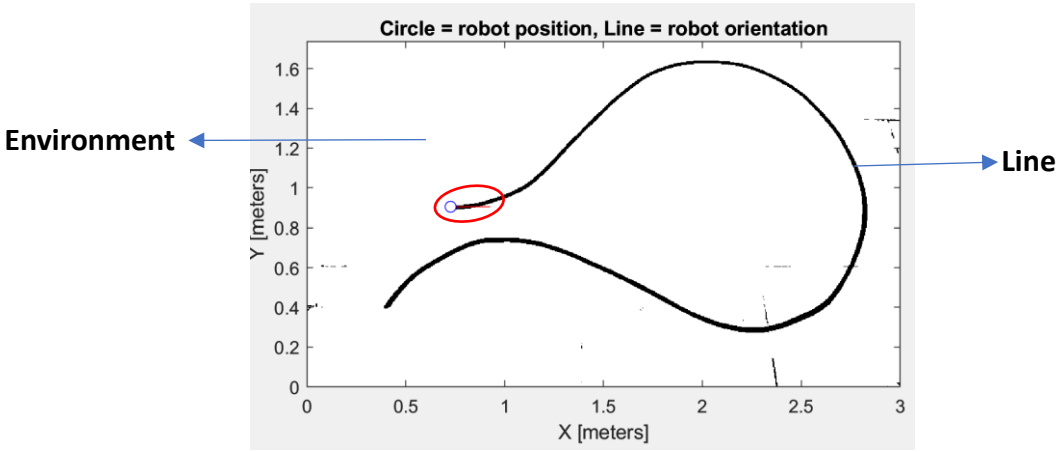


Figure 43: *Map with a black line as path and white space as an environment for line following.*

The line sensor block was used to represent over real line sensor. Similarly, the sensor value parameters previously found for the line and environment was specified in the line sensor to make the block output values like actual sensors. The simulation for the line following was setup.

5.1. Line following using On/Off and Dead Reckoning Algorithm:

The general idea is to check the sensor value if its greater or smaller than threshold value. If the sensor value is greater than the threshold value, the robot will turn left otherwise it will turn right. This makes robot sensor to sense the environment and line at the same time to keep the track. Similarly, the condition for the line is also checked which signifies the end of the line. Once, the end of the line is reached, algorithm will stop otherwise it keeps repeating the process of turning left and right. **Since, the algorithm keeps shifting between these two states till the threshold value of sensor (reference) isn't achieved , it is also called on/off control and together the state flow charts are used where the Dead Reckoning algorithms are specified.**

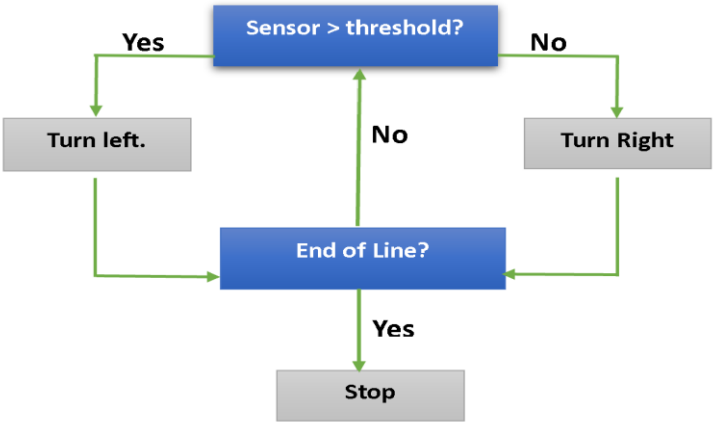


Figure 44: *Representation of line following process of on/off algorithm in a block diagram.*

5.1.1. Simulink Model of on/off algorithm representing line following

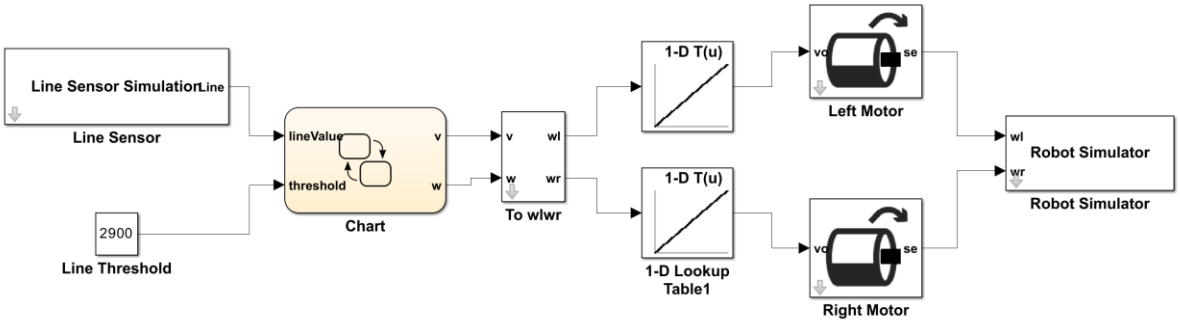


Figure 45: *Simulink Model representing line following using on/off algorithm.*

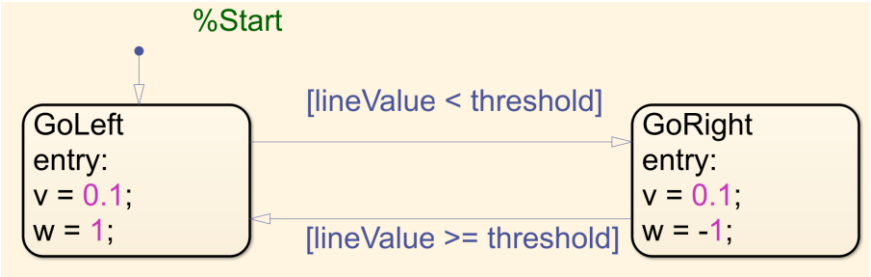


Figure 46: *State flow chart with the linear and the angular velocity specified for line following.*

The state flow chart was used to generate the linear velocity **'v'** and the angular velocity '**Ω**' to help the robot track the line. The two states have been defined in the state flow chart, "Go left and Go right". The velocity is set to be 0.1 m/s rather it could be higher with the velocity range calculated up to 1.04 m/s. The smaller velocity was chosen because the sensor can sense the environment precisely, but it also depends on the quality of sensor. The positive value of omega signifies the robot to turn left and similarly, the negative (-1) specifies the robot to turn right. The conditions have been defined that will cause the algorithm to move between two states. As discussed earlier, the line value lesser than the threshold, the robot will move from left to right and the greater or equal line value to the threshold, makes the robot moves from right to left.

5.1.2. Results of on/off algorithm used in line following

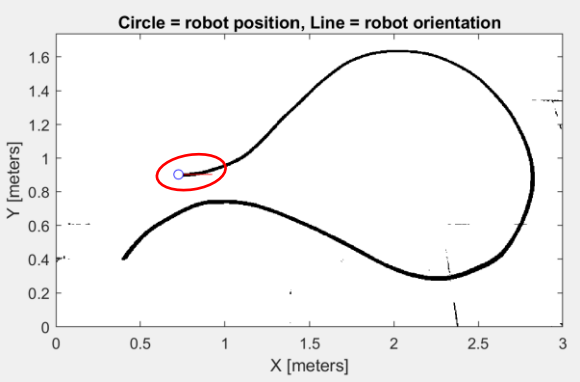


Figure 47: *Starting position of a robot at v =0.1 m/s*

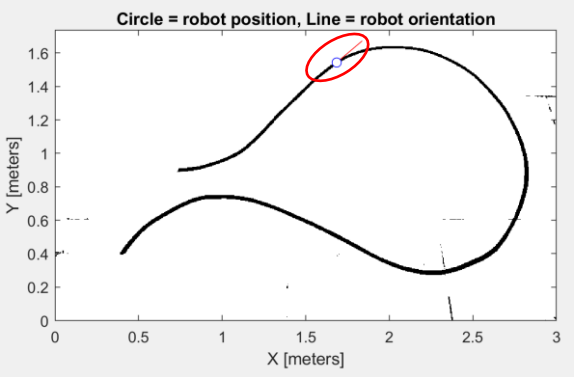


Figure 48: *Robot following the line track at v=0.1m/s*

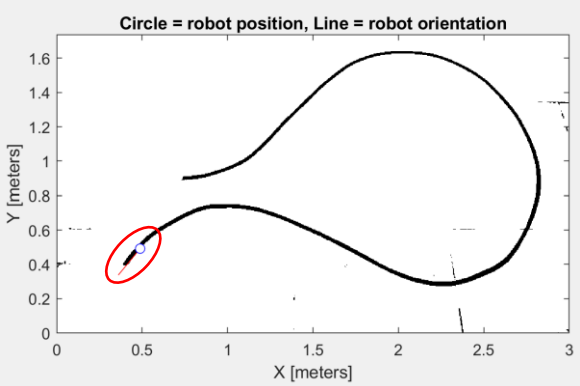


Figure 49: *Line followed by robot till the end of track successfully at v = 0.1s.*

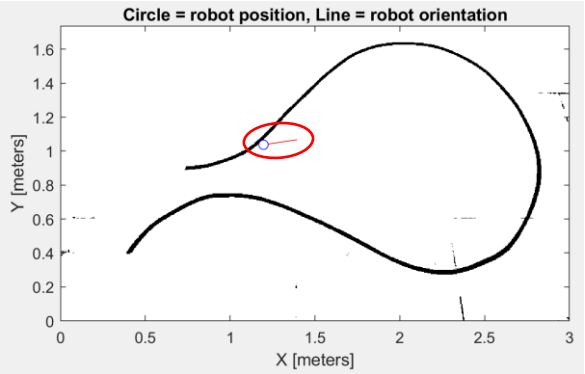


Figure 50: *Robot got off-track at v=0.16 m/s*

The robot successfully tracks the line. The simulation was used to test the results, changing the value of linear velocity 'v'. The lower velocity than 0.1 m/s successfully tracks the line, but the speed of the robot gets slower. The higher velocity than 0.1m/s transverses the line even quicker and the highest velocity that the robot follows the line is found to be 0.15 m/s but the robot motion gets jerky with the increase in velocity. The velocity above 0.15 m/s results the robot to be off-track as shown in **fig 50** and it is because the sensor cannot sense the environment precisely due to high speed. Overall, the result obtained using Dead Reckoning algorithm produced jerky moment of the robot during line following as there were no any controllers used to eliminate the disturbance or errors as described previously in control section. Again, the result in the actual hardware may even more jerky due to various external factors related to motor friction, slip, slag, and robot's own weight.

5.2. Line following using PID

The logic is to utilize the line sensor to measure the current value which is then compared with a reference value i.e., the threshold value to calculate an error. The error fed into the PID controller. The linear velocity is the constant value i.e., 0.1 m/s as the controller outputs the angular velocity '**Ω**' based on the input error. This means the controller only controls the turning rate of the robot assuming the constant linear velocity.

5.2.1. Simulink Model of PID Control algorithm representing line following

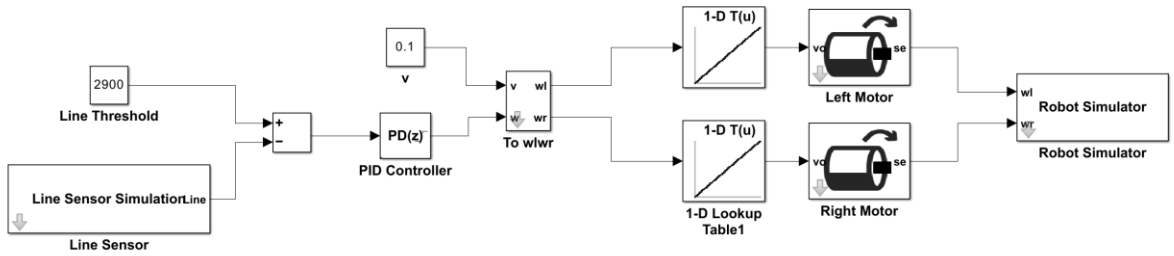


Figure 51: *Simulink Model representing line following using PID control algorithm.*

The parameters of the PID controller were set. Initially, "P" controller was used, and the results were tested using "PD" controller later. The integral controller wasn't considered for the test because the line following doesn't depend on previous error values. In fact, keeping track of some of the errors might produce inaccurate and unnecessary results, resulting in drifting values of the robot **(SHEIKH FARHAN JIBRAIL, 2013).**

5.2.2. Results of P controller (P = 0.031)

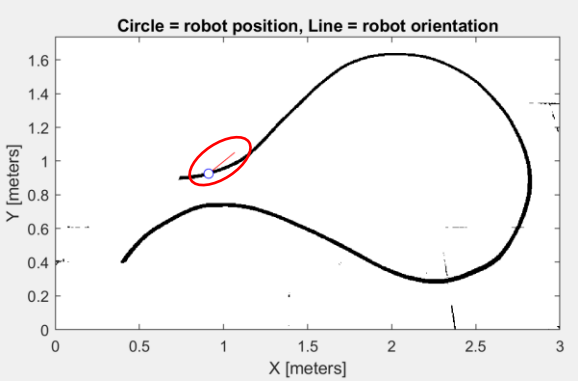


Figure 52: *Starting position of a robot at "P" controller value of 0.031*

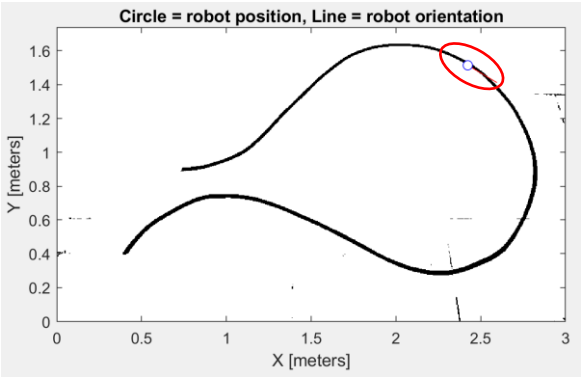


Figure 53: *Robot following the line track at "P" controller value of 0.031*

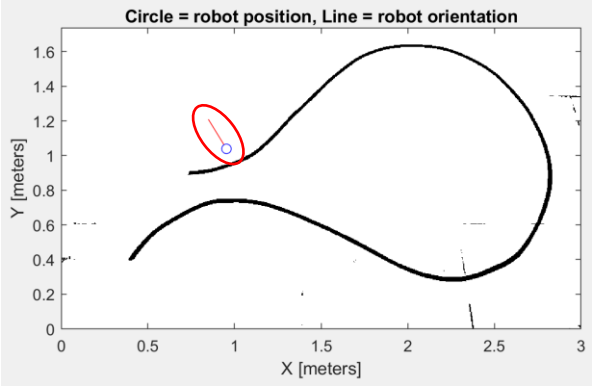


Figure 54: *Robot rotating within same space at P= 0.03*

After certain trial and error, the best "p" controller value obtained was "0.031". At this controller value, the robot's motion was found to be a bit unstable initially but as the robot approaches forward, the stability as well as smoothness in the motion can be seen. Similarly, the movement was quite jittery in the turning points of the map. The proportional gain value below 0.031 made the robot rotate in a same place as shown in **fig 54** and it was because the gain value in the PID block becomes too high causing the proportional gain to amplify the error signal and the resulting control signal exceed the range of system's actuator makes the robot to rotate in a same place.

5.2.3. Results of PD controller (P = 0.031 and D = 1e-9)

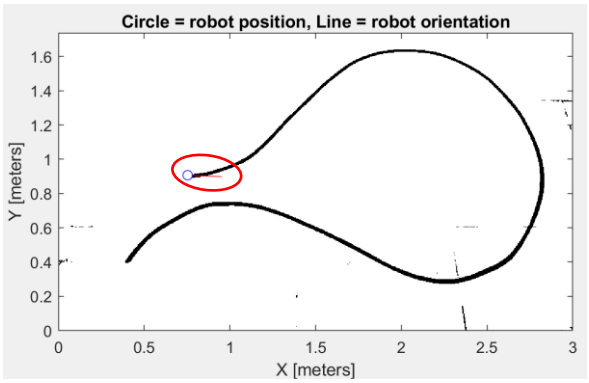


Figure 55: *Starting position of a robot at "P" "P" = 0.031 and "I" = 1e-9.*

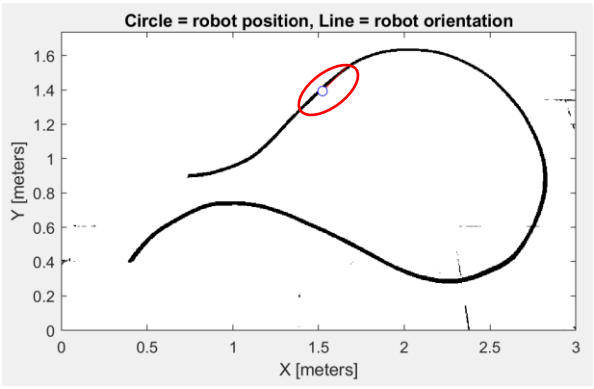


Figure 56: *Robot following the line track at "P" = 0.031 and "I" = 1e-9.*

Since, the "D" controllers are very sensitive to the noise, initially the smaller gain value was used for the trial and error and the Derivative gain value of **1e-9** along with Proportional gain **0.031** was found to be the suitable one. As in "P" controller, initially the robot motion was oscillating and settles down into a smoother motion. The obtained response of the robot using PD controller was lot quicker than the "P" only controller. The oscillatory motion of the robot settled earlier than in the "P" only controller. The motion in the turnings was found to be as jerky as in the P controller but in a straight line, the "PD" controller shows the smoother response than the P controller.

5.3. Overall conclusion of on/off algorithm and PID control Results in line following

The “PD” controller produced little smoother response than the “P” controller and it is because the “P” controller only uses the current error between the actual and desired position of the robot to generate a control signal due to which it has high probability to prone to overshoot or oscillation especially when it encounters turns or sudden change whereas the “D” term in “PD” controller compensate the future errors which uses both the current error and the rate of change of the error (i.e., derivative) to generate the control signal **(Sena TEMEL).**

Similarly, in term of speed, the results obtained using on/off algorithm and “PD” controller were similar but in term of smoothness, the results of on/off algorithm were jittery than the results obtained using “P” controller. This is because most of the errors and disturbances in the “P” controller were fade in the PID controller whereas the on/off algorithm are based on a simple threshold value that determines when the robot's actuators are turned on or off which lacks the ability to provide continuous control over the robot's motion. The simulated results can’t be clarified with just the pictures, the simulation files will be uploaded together with the report for clear analysis. Similarly, the obtained result in the simulations were quite indifferentiable but it would surely have greater impact if tested in an actual hardware.

6. Obstacle Detection

Obstacle detection is the basic way to perform localization for robots. Localization is the ability of a robot to know its position on the map. For the obstacle detection analysis, the ultrasonic sensor was utilized that uses the ultrasonic waves and measures the wave reflected from the obstacle surface to give the value corresponding to the distance of an obstacle. Ultrasonic sensors usually contain 3 essential values that characterizes them i.e., 1) the maximum range (farthest distance that the sensor can measure), 2) minimum range (minimum distance that the sensor can measure) and the 3) resolution that tells the smallest changes that the actual ultrasonic sensor can measure. These values can be obtained from the manufacturer’s datasheet or using the experiment. Here, for this analysis the values that used were obtained from the experiment and is available in mobile robotics training provided by MathWorks **(MathWorks, MathWorks, 2020).**

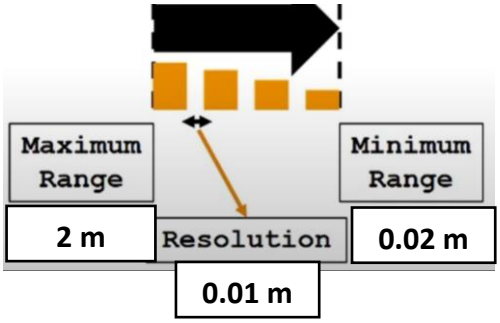


Figure 57: *Resolution, maximum and minimum range of a ultrasonic sensor.*

$$\boldsymbol{threshold\ value} = \frac{\boldsymbol{Maximum\ Range} + \boldsymbol{Minimum\ Range}}{\mathbf{2}} = \mathbf{0.1}\ (\boldsymbol{in\ this\ case})$$

Simulation Setup: Similar to the line following, the custom map was used as an environment and the obstacle was specified with a black block. The colour of the environment was specified as white whereas only the obstacle was made of black colour in the map.

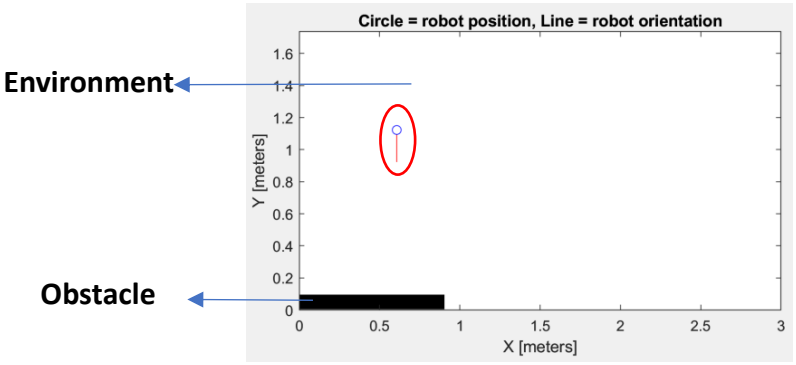


Figure 58: *Map with an obstacle.*

The mat. extension of the map was used which generates the variable for simulation and was loaded in the Robot Simulator block and the starting position of the robot in the map was specified, marked in red in **fig 58**. The ultrasonic sensor block was used to represent over the real ultrasonic sensor. The parameters for the ultrasonic sensor described earlier i.e., maximum range (2 m), minimum range (0.02 m) and the resolution (0.01 m) was used in ultrasonic sensor block output values as an actual sensor. The simulation for the obstacle detection was setup.

6.1. Obstacle Detection using On/Off and Dead Reckoning Algorithm

The basic logic is to check the value returned by the ultrasonic sensor, if it is greater than the threshold value, the robot is still a distance away from the obstacle and robot keeps going forward towards the obstacle. If the sensor value is less than or equal to the threshold value signifies the robot is within the threshold distance from the obstacle and it should stop.

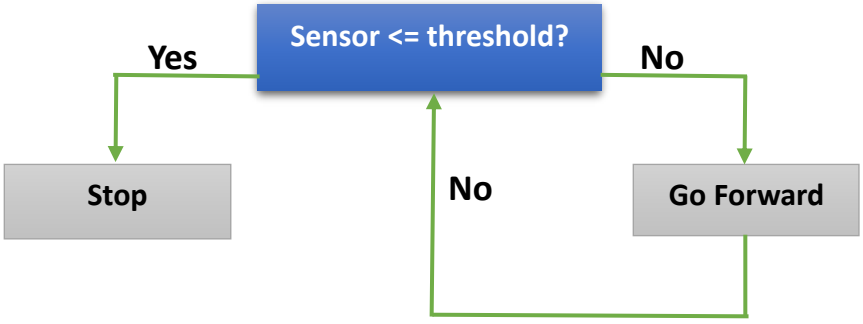


Figure 59: *Representation of obstacle detection process of an on/off algorithm in a block diagram.*

6.1.1. Simulink Model of on/off algorithm representing obstacle detection

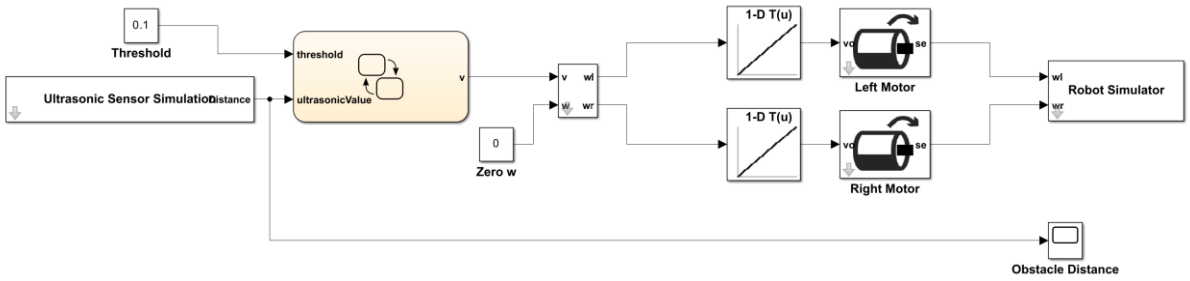


Figure 60: *Simulink Model representing obstacle detection using on/off algorithm.*

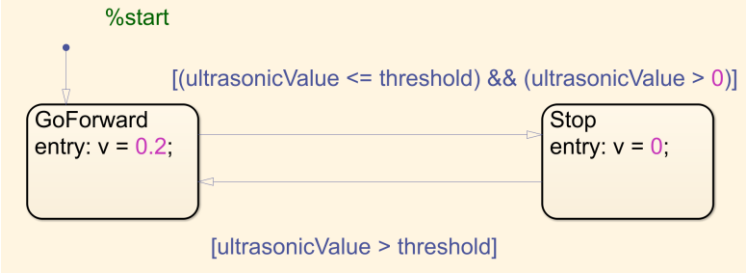


Figure 61: *State flow chart with the linear velocity specified for obstacle detection.*

The goal was to utilize the information of ultrasonic sensor to generate the linear velocity **"v"** value so that the robot stops 0.1 m (threshold value) away from the obstacle. In the state flow chart, the Go Forward state is specified with the constant linear velocity of 0.2 m (since it is about the distance measurement the same velocity is used as in motion control) and the second state i.e., stop state the velocity should be zero for the robot to stop, so the velocity was set to zero. Since, the algorithm shuttles between these two states , these are connected with the back and forth connections where the conditions are applied. The ultrasonic value less than or equal to threshold value, it moves from go forward to the stop state and the greater ultrasonic value to the threshold, makes the robot move from stop state to the goal state.

6.1.2. Results of on/off algorithm used in obstacle detection

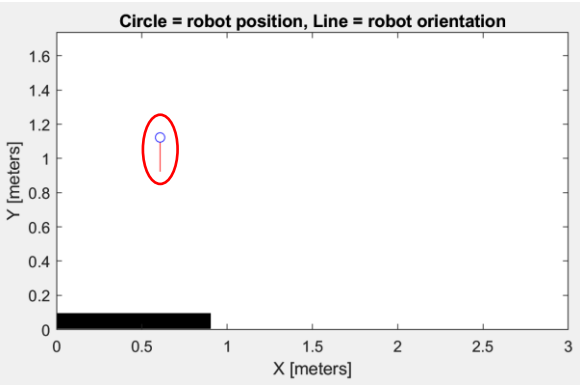


Figure 62: *Starting Position of the robot.*

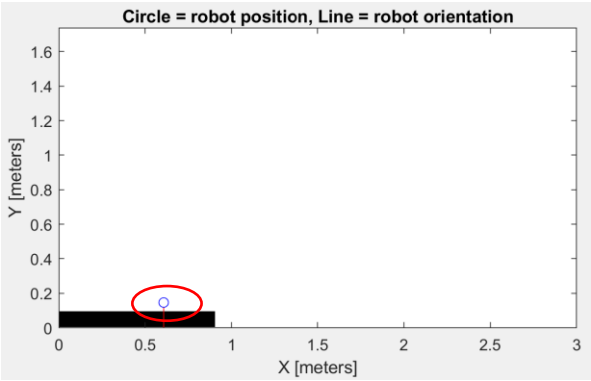


Figure 63: *Final position of the robot* ***(v = 0.2m/s)***

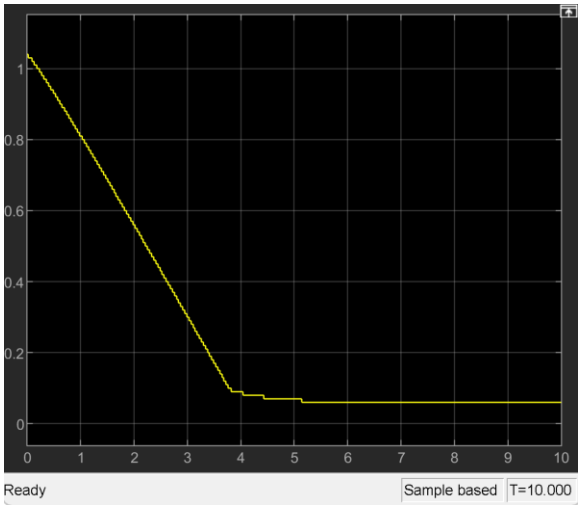


Figure 64: *Obstacle Detection graph showing Distance Vs velocity* ***(0.2 m/s).***

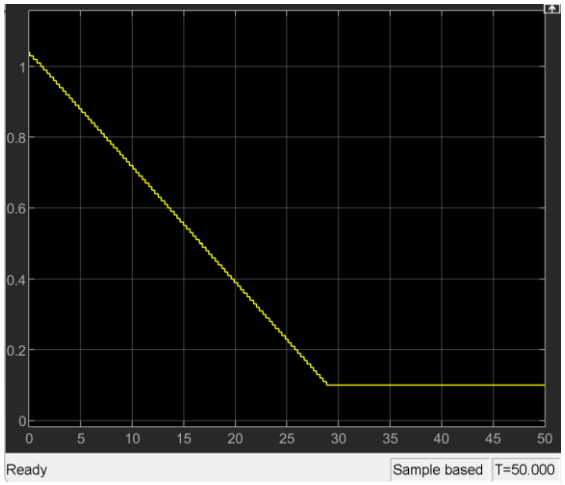


Figure 65: *Obstacle Detection graph showing Distance Vs Velocity* ***(0.025 m/s)***

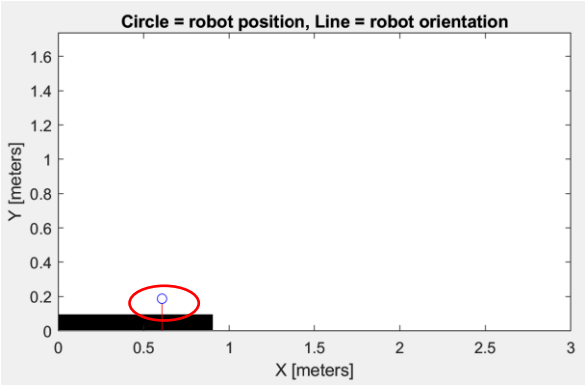


Figure 66: *Final position of the robot* ***(v = 0.025m/s)***

The **fig 62** and **fig 63** shows that the robot moves in a straight line and stops just before the obstacle. The obstacle detection graph at velocity 0.2 m/s **(fig 64)** shows that the robot moves towards the obstacles and exceed the threshold value (0.1) meaning it stops beyond the reference distance. This is because the robot stops immediately from the velocity of 0.2 m/s to 0 m/s which is physically impossible. The obstacle detection graph with the velocity 0.025 m/s **(fig 65)** shows the better result where the robot stops exactly at the reference distance (0.1 m) but it slows down the robot almost by 25 sec. Therefore, the accuracy and inaccuracy in the result is based on the speed of the robot travelling. The staircase like response is generated because of the way in which the sensor modelled uses a resolution parameter which is the minimum distance that it can see.

6.2. Obstacle Detection using PID controller

The ultrasonic sensor monitors the current distance of the obstacle which is compared with the desired reference value i.e., the threshold value (0.1 m) to calculate an error. This error is faded into the PID controller. In this case, where the position is being measured, the PID controller outputs a velocity "v" value based in the input error. The angular velocity is zero since the robot is moving forward in a straight line.

6.2.1. Simulink Model of PID Control algorithm representing obstacle detection

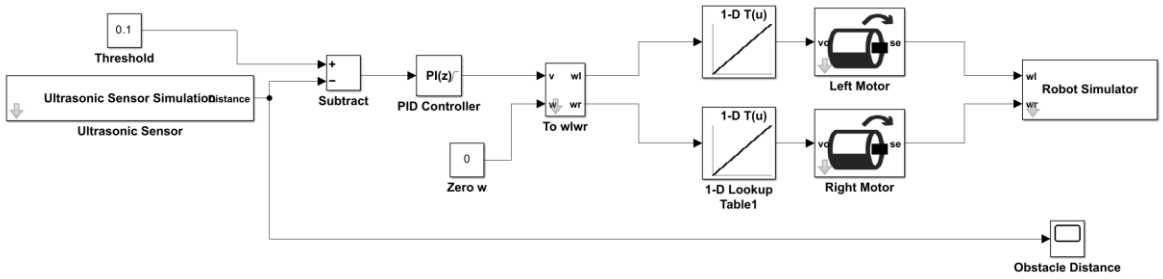


Figure 67: *Simulink Model representing obstacle detection using PID control.*

The parameters of the PID controller are set. Initially, "P" controller is used, and the results are tested using "PI" controller later.

6.2.2. Results of P controller (P =-0.1)

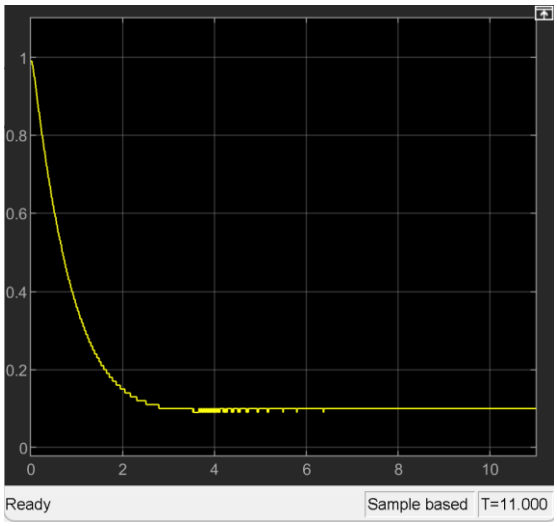


Figure 68: *Obstacle Detection graph showing Distance Vs Velocity* ***(P=-0.1).***

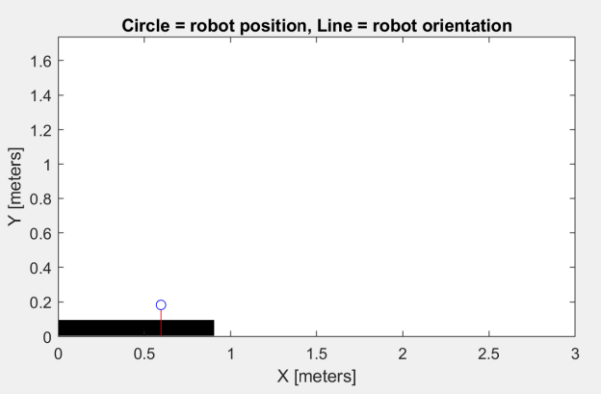


Figure 69: *Final position of a robot* ***(P=-0.1).***

Initially, the positive gain value 0.1 was chosen and it was observed that the robot was moving in backward direction instead of forward. The reason was the output of the controller i.e., linear velocity 'v' was negative, and this happens because the value of ultrasonic sensor decreases as it goes closer to the obstacle. Therefore, the sign of the gain was flipped over to get the correct behaviour of a positive linear velocity. After changing the proportional gain value to negative (-0.1), the robot moves

forward and stops at a particular distance from the obstacle as seen in **fig 69.** The obstacle detection graph in **fig 68** shows the robot stops exactly 0.1 m away from the obstacle and the reducing velocity can be observed as the robot approaches closer to the destination (0.1 m). Similarly, the robot reaches and stops at the destination smoothly within 3 sec whereas the results from on/off algorithm was way off from the destination within 3 sec.

6.2.3. Results of PI controller (P =-0.1 and I =-0.001)

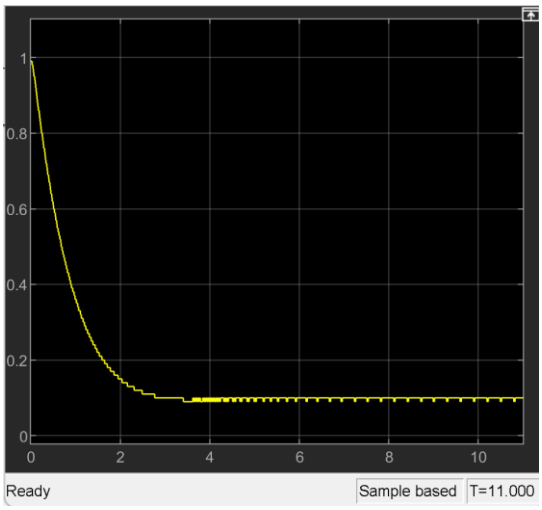


Figure 70: *Obstacle Detection graph showing Distance Vs Velocity* ***(P=-0.1 and I=-0.001).***

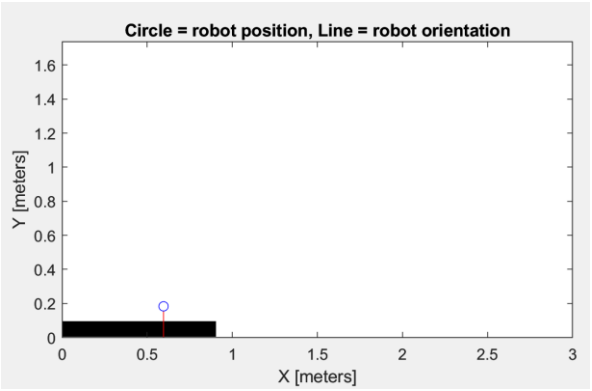


Figure 71: *Final position of a robot.*

Initially, the bigger integral gain value shows slight oscillation of the robot towards the end. The integral gain value of -0.001 somehow eliminates the problem of oscillation. The obstacle detection graph in **fig 70** still shows small oscillation towards the end which is called chatter and is due to the resolution parameter of the ultrasonic sensor. In real life it usually gets damped by the robot mass, wheel friction and so on.

6.3. Overall conclusion of on/off algorithm and PID control Results in obstacle detection

Since, the "P" only controller might result in an offset or a steady state error, the integral gain was added together with proportional gain but there was not much difference observed in the simulation. The robot approached towards the destination and stopped smoothly when the P and PI controllers were used but the On-Off algorithm exceeded the destination point and then stops. Unlike PID controller, the on-off algorithm doesn't contain any controllers where the robot could receive the feedback and adjust its output accordingly. The states specified in the state flow chart keeps shuttling within each other till it reached the destination and its suddenly tried to stop at the destination point which is physically not possible and gets-off way behind the destination. Therefore, the accuracy of the on-off algorithm depends on the speed of the robot where the lower speed was found to be more accurate, but it increases the time to reach the goal.

The simulated result using **P and PI controller** in a robot look like in **fig 72.**

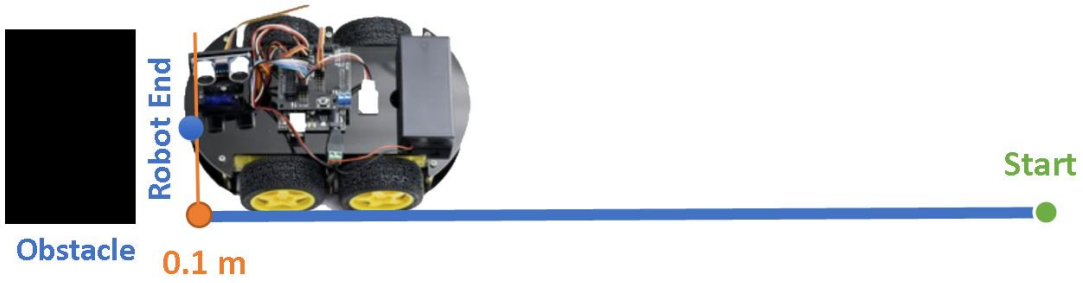


Figure 72: *Demonstration of simulated obstacle detection result in a robot using PID.*

The simulated result using **on/off algorithm** in a robot look like in **fig 73.**

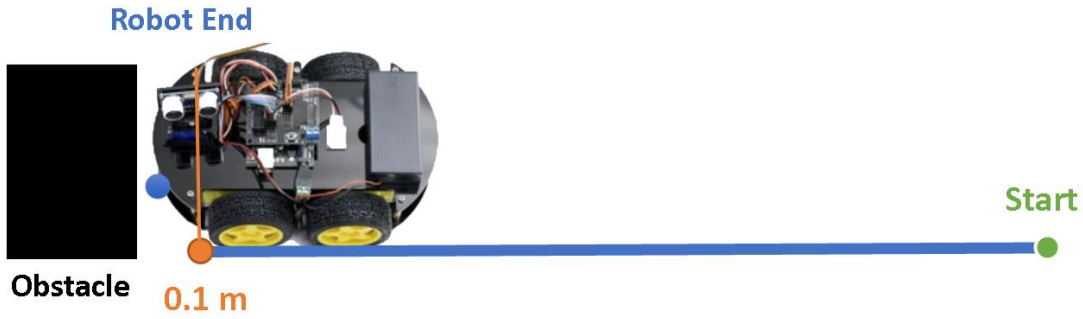


Figure 73: *Demonstration of simulated obstacle detection result in a robot using on/off algorithm.*

7. Pre-defined Path Navigation

Path navigation refers to the various tasks the robot performs to complete a large objective. The general idea to the path navigation has been already analysed and performed in the motion planning, line following and the obstacle detection part. The results are also analysed and found that the PID controller approaches the result way better than the Dead Reckoning and On/Off control. Here the objective is to use the PID controller in the State flow chart bringing all the logic together that has been performed in previous sections in a single supervisory logic to observe that the robot actually follows the path in sequence. All the procedures and parameters are already known and simply the task needs to be sequenced. Therefore, this can also be called as pre-defined path navigation.

Procedure: The half-track of the same map used in the line following was generated and the obstacle was also added in the map. There was a small modification in the map i.e., black line indicated by "C" in **fig 74** which was used to check the end of the line condition at the end of the line following. The starting position of the robot was set in the robot simulator. The general idea of this path navigation

was to commence the robot at position A, observe if it successfully travels all the way to path B and C and stops before the obstacle D as shown in **fig 74.**

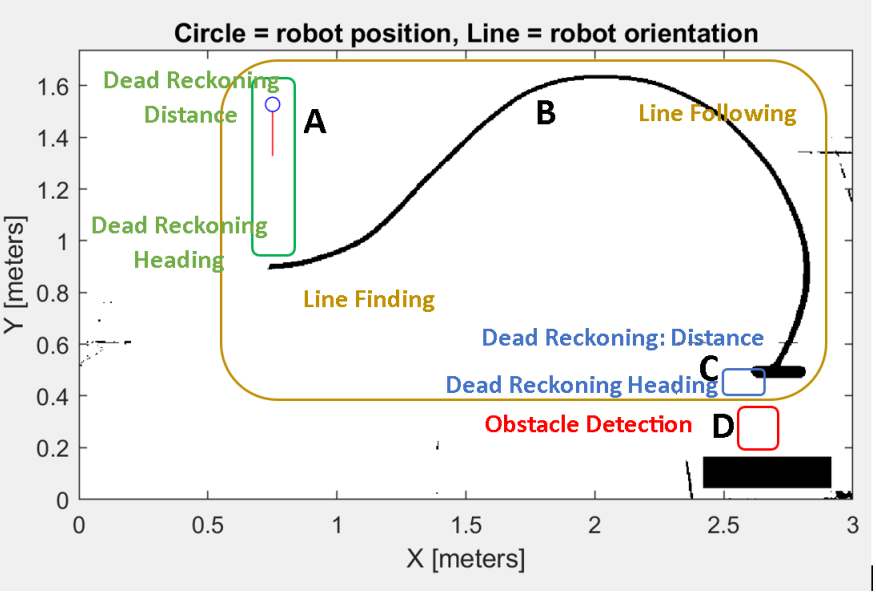


Figure 74: *Map with line and obstacle together.*

Simulink Setup: The state flow is used to represent the different segments of navigation (A, B, C, and D). Each of the algorithms designed (motion control, line following and the obstacle detection) previously were implemented as a sub state in each superstate. The appropriate transition conditions were used to sequence the task. The transition runs through all the superstates sequentially following the substates defined. The PID controllers were used inside the state flow chart and the same PID controller values were used here that were tested in the previous parts. The distance controller used in state flow chart outputs the linear velocity 'v' as in motion planning and obstacle detection whereas the controller used in the line following outputs the angular velocity which has been named as Head CTRL in the in the state flow chart. The line sensor and the ultrasonic sensor along with their threshold value are connected with the input of state flow chart.

7.1. Simulink Model of combined PID Control and Dead Reckoning Algorithm to perform pre-defined navigation

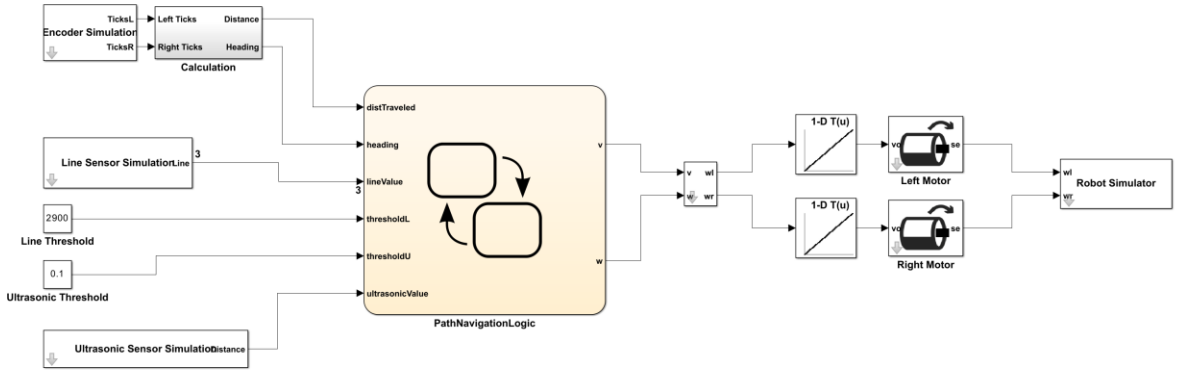


Figure 75: *Simulink Model representing pre-defined navigation using PID and Dead Reckoning.*

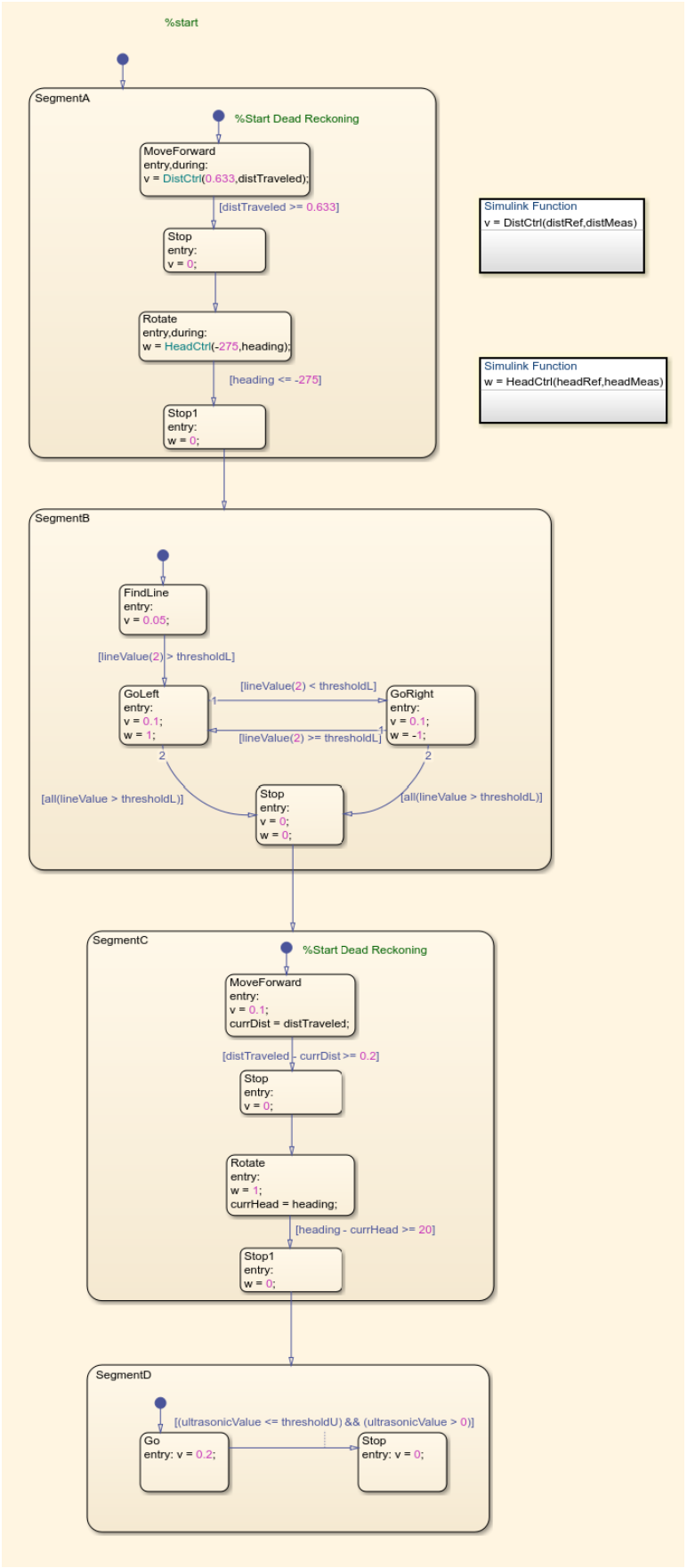


Figure 76: *State flow chart with all the 4 segments i.e., motion control, line following, end line checking and obstacle detection.*

7.2. Results of navigation performed.

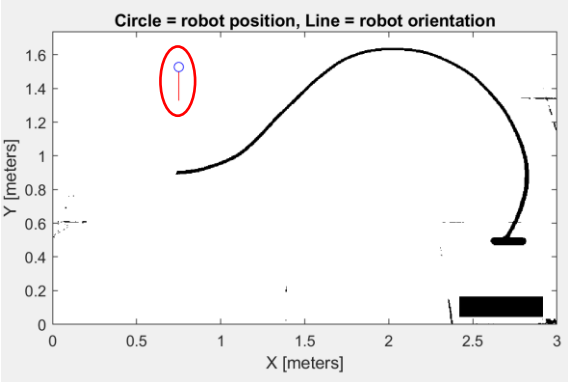


Figure 77: *starting position of a robot.*

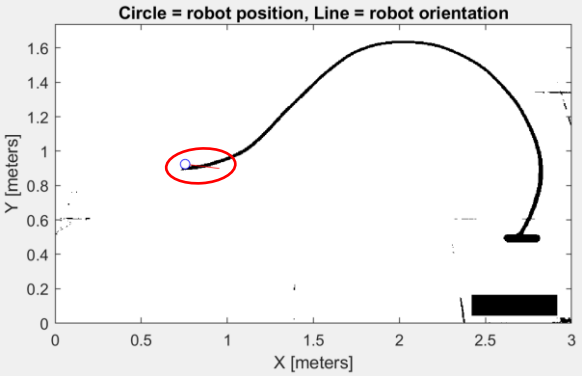


Figure 78: *Robot searching for a line.*

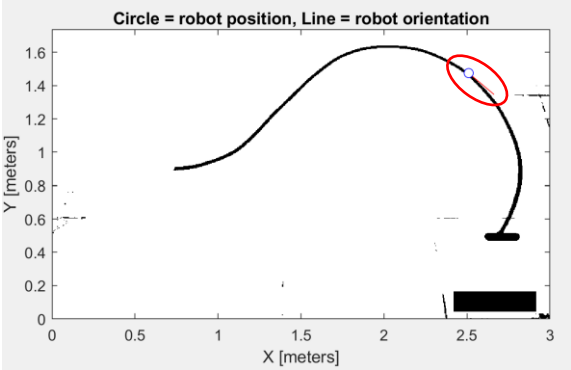


Figure 79: *Robot following a line.*

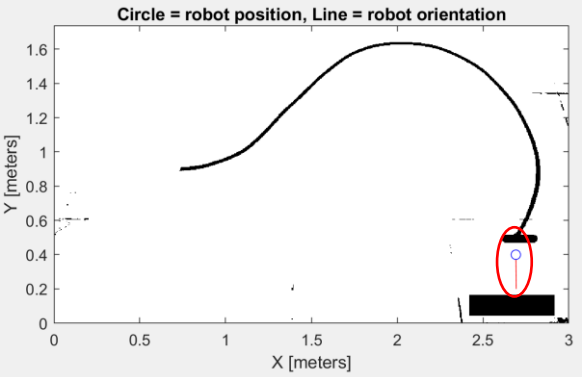


Figure 80: *Robot heading towards obstacle.*

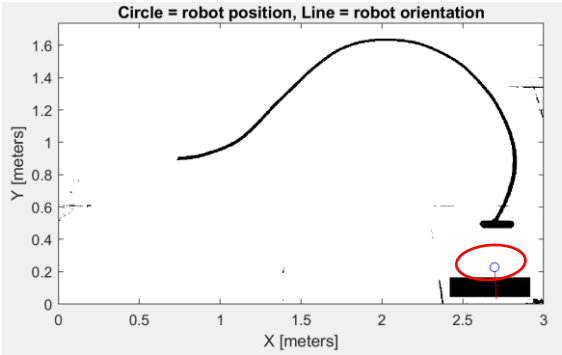


Figure 81: *Robot stopped before obstacle.*

The pre-path navigation technique was successfully implemented, and it worked successfully. The robot started moving in a free space in a map and as it detected the line, it started transverse along the line. At the end of the line, robot directed its motion towards the obstacle and after certain distance it stopped exactly before the obstacle as the algorithm provided. The path navigation was smooth as the PID controller was used along with the state flow chart to minimize the disturbances and errors.

8. Overall result from motion control, line following, obstacle detection and path navigation.

In the motion control, initially the Dead reckoning performed in a closed-loop system accurately reached the reference distance as the wheel velocity was directly fed to the simulation but there was no such factor specified which affects the input and overall result. After the motor physics was applied, it was found that the measured distance crossed the reference distance by almost 0.1 m which was the case of steady state error. The problem was solved using PID controller. The “P” controller minimized the steady state error but still there was small offset which was tried to eliminate using “PI” controller. Eventually, the steady state error was almost eliminated, the robot stopped very close to the end point.

In the line following, using the on/off algorithm robot successfully followed the line, but the velocity had an impact on the accuracy and smoothness of the motion. A lower velocity than 0.1 m/s resulted in slower motion but accurate tracking, while a higher velocity of up to 0.15 m/s resulted in quicker motion but with jerky movements. Any velocity above 0.15 m/s caused the robot to go off-track due to imprecise sensor readings at high speeds. On the other hand, The best "P" controller value obtained after trial and error was 0.031. Lower values resulted in the robot rotating in the same place, while higher values cause jittery movement. For the "D" controller, a small gain value of 1e-9 and a proportional gain of 0.031 were found suitable. The robot's response using PD controller was quicker than using P only controller, with smoother motion in straight lines but still jerky in turning points.

In obstacle detection, the robot moves in a straight line and stops before an obstacle using on/off algorithm. At a velocity of 0.2 m/s, the robot moves towards the obstacle and stops beyond the reference distance due to physical impossibility of stopping immediately. At a velocity of 0.025 m/s, the robot stops exactly at the reference distance but slows down the robot by 25 seconds. The accuracy and inaccuracy of the result is based on the robot's speed in an on/off algorithm. On the other hand, a positive proportional gain value of 0.1 caused the robot to move backwards instead of forwards due to the negative output of the controller as the ultrasonic sensor value decreases. Changing the proportional gain value to negative (-0.1) made the robot move forward and stop at the precise distance from the obstacle. The integral gain value of -0.001 eliminated oscillation but resulted in small chatter towards the end. The obstacle detection graph in **fig 70** showed that the robot stopped exactly at 0.1 meter away from the obstacle, with reducing velocity as it approached. The robot reached and stopped at the destination smoothly within 3 seconds, unlike the on/off algorithm which produce less accurate result.

Eventually, all of these algorithms were combined for the path navigation. The combined analysis of PID with Dead reckoning was done and the result showed the robot smoothly followed the line and stopped before the obstacle precisely.

9. Discussion

The different algorithms were used to understand the navigation process and to analyse the accuracy of their results of those algorithms in the path navigation techniques. It was found that the inaccuracy in the Dead Reckoning and the on/off algorithm was higher. During obstacle detection using Dead reckoning, the robot stopped nearly 3 cm beyond the reference line almost hit the obstacle and similarly, in the motion control, the measured distance obtained was way far than the reference distance **fig(28)**. The movement of the robot was found slower as it only depends in the specific

velocity specified whereas the PID controller can adjust the velocity according to the range of speed robot can travel. Overall, the results obtained using PID was smoother and precise either in the motion control, line following and obstacle detection whereas the inaccuracy and the slow response can be observed from the Dead reckoning and on/off algorithm. However, the results within the PID controller were also comparable. In motion control, it was found that the "P" (proportional) only controller has an offset between the measured and the reference distance that resulted in a non-zero steady state error. This is because a proportional controller only uses the current error between the setpoint and the process variable to determine the control output and does not take into account the past errors or the rate of change of the error. As a result, when the proportional gain was set too high, the controller resulted in overshoot and oscillations around the setpoint . Conversely, low proportional gain resulted in undershoot and slow response. In either case, the steady state error was found **(ni, 2023).** The steady state error was somehow eliminated using the Integral controller, but the problem of Whiner arises which was solved changing the anti-wind-up method to clamping. Especially, in the line following, the integral control wasn't very useful as the robot during line following doesn't depend on previous error values. In fact, keeping track of some of the errors might produce inaccurate and unnecessary results, resulting in drifting values of the robot **(SHEIKH FARHAN JIBRAIL, 2013).** It was found that the simulation results were more realistic using the motor physics as it relates the actual robotics component. The PID controller adjusts output based on the motor response such as speed capabilities, time delays or nonlinearities to compensate for any disturbance and improves performance**.**

On the other hand, although the Dead Reckoning and on/off algorithm results may not be better than the PID controller, but the results can be improved combining it with other localization techniques such as GPS or other visual odometry. Similarly, using the good integration algorithm such as Kalman filter or Particle filter could improve the results of Dead reckoning and on-off algorithm. The quality of sensors used with the Dead Reckoning also affects the result which can be solved using high-quality sensors **(KyuCheol Park 1, Dead Reckoning Navigation For Autonomous Mobile Robots, 1998)**. Overall, the Dead-Reckoning algorithm alone is not very reliable for complex control system where precise position and orientation information are crucial.

10. Conclusion

Overall, the main objective of the project was to understand the navigation techniques of the robot. This has been achieved by performing simulation analysis individually on the motion control, line following and the obstacle detection. Eventually, all these algorithms were put together and performed sequentially in a simulation to visualize how the path navigation in a robot functions. Through this analysis, a basic understanding on the path navigation techniques has been developed. The comparison of Dead Reckoning algorithm with the PID controller in the navigation process shows that the Dead reckoning algorithm is simple to implement but the results obtained from it is not very reliable as there are lots of inaccuracies. It can be confirmed from motion planning results where the robot stopped way beyond than the end point and the similar result can be seen in the obstacle detection where the robot stopped very close to the obstacle crossing its threshold value. On the other hand, PID controller shows impressive results on all the path navigation techniques used. Although the result wasn't 100% optimal, the motion planning and the obstacle detection shows that the robot covered the distance and stopped smoothly near the end point. Similarly, in the line following, the PID controller shows the smooth movement of the robot whereas the on/off algorithm produced jerky motion. This confirms that the errors and disturbances generated are somehow faded in the PID but due to the on/off algorithm and Dead reckoning algorithm depend on their previous position to

determine the current position, the sum of small errors in each of the position arise huge difference and there are no controllers to mitigate it. Although the result obtained using PID controller are better, but it won't provide same result if tested in real hardware due to the various factors like robots own weight, motor's friction and so on and these parameters definition aren't still available in simulation. Therefore, there may need of trial and error in the real hardware after the simulation results to achieve optimal results. Ultimately, the hypothesis that was made come to be correct as the PID control algorithm was found best among the on/off and dead reckoning algorithm for path navigation technique.

In relation to this project, I performed the motion control in the actual hardware named Bittle which is a Quadruped Robot Dog. I created different behaviours and gaits such as jumping, flipping, running, and climbing which was the part of this project. The other objective related to the path planning in a Bittle couldn't be achieved due to technical difficulties and the lack of sources available for that actual robot. I have written the technical report on how I created those motions on Bittle and can be found in the link added in my logbook or links below:

1. Technical report: Click here
2. C++ files of different motions created in Bittle: Click Bittle_new_skills.zip

3. Simulink models used in this report: Click here

References

A.R. Mohd Azizi, M. M. (2009). Dead Reckoning of a Skid Steer Mobile Robot using Fuzzy. *European Journal of Scientific Research* , 305-314.

Douglas, B. (2020, 07 08). *MATLAB*. Retrieved from Understanding SLAM Using Pose Graph Optimization | Autonomous Navigation: https://www.youtube.com/watch?v=saVZtgPyyJQ&t=5s

Douglas, B. (2020, 07 08). *MATLAB*. Retrieved from What Is Autonomous Navigation? | Autonomous Navigation: https://www.youtube.com/watch?v=Fw8JQ5Q-ZwU

Engin1, M. (2012). PATH PLANNING OF LINE FOLLOWER ROBOT . *IEEE.* Amsterdam, Netherlands: IEEE Xplore.

Hardik S Jain, A. S. (2019). DC Motor Speed Control using PID Controller, IR Sensor and PWM Hysteresis. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*.

IQ, V. (n.d.). *VEX IQ*. Retrieved from Understanding VEX IQ Wheels: https://kb.vex.com/hc/en-us/articles/360035955171-Understanding-VEX-IQ-Wheels#:~:text=Their%20diameter%20is%20set%20up,consistent%20and%20easy%20to%20turn.

KyuCheol Park 1, H. C. (1998). DEAD RECKONING NAVIGATION FOR . *ScienceDirect*.

KyuCheol Park 1, H. C. (1998). Dead Reckoning Navigation For Autonomous Mobile Robots. *ScienceDirect*.

Learn, F. (n.d.). *Future Learn*. Retrieved from Line sensors and how to use them: https://www.futurelearn.com/info/courses/robotics-with-raspberry-pi/0/steps/75899#:~:text=Infrared%20(IR)%20detection-,Line%20sensors%20detect%20the%20presence%20of%20a%20black%20line%20by,a%20light%20sensor%20(receiver).

MathWorks. (2020, 08 14). *MathWorks*. Retrieved from Mobile Robotics Training Toolbox: https://uk.mathworks.com/matlabcentral/fileexchange/62961-mobile-robotics-training-toolbox

MathWorks. (2020, 08 14). *MathWorks*. Retrieved from Mobile Robotics Training Toolbox: https://uk.mathworks.com/matlabcentral/fileexchange/62961-mobile-robotics-training-toolbox

MATLAB. (2020, 07 08). *MATLAB*. Retrieved from Understanding SLAM Using Pose Graph Optimization | Autonomous Navigation: https://www.youtube.com/watch?v=saVZtgPyyJQ&t=251s

ni. (2023, 03 30). *ni*. Retrieved from The PID Controller & Theory Explained: https://www.ni.com/en-gb/shop/labview/pid-theory-explained.html

OMRON. (n.d.). *RS*. Retrieved from Incremental Rotary Encoder OD 40 dia.E6B2-C: https://docs.rs-online.com/7639/A700000007380478.pdf

Sena TEMEL, S. Y. (n.d.). *P, PD, PI, PID CONTROLLERS.* MIDDLE EAST TECHNICAL UNIVERSITY, ELECTRICAL AND ELECTRONICS, ENGINEERING DEPARTMENT.

SHEIKH FARHAN JIBRAIL, R. M. (2013). *PID CONTROL OF LINE FOLLOWERS.* National Institute of Technology ROURKELA.

Songxiao Cao, Y. J. (2022). Design and Experiments of Autonomous Path Tracking Based. *applies sciences*, 16.

Stefan Edelkamp, S. S. (2012). DeadReckoning. *Science Direct*, 20.