IVR Coursework 2: Robotics Report

Weiting Goh (S1450710), Isabella Chan (S1330027),
and Ink Pansuwan (S1409577)

November 24, 2016

1 Introduction

In this IVR assignment, there are three tasks: 1) following a line, 2) following a broken line, and 3) following a line to an obstacle, circumvent the obstacle and find the line again. This report aims to capture functions and limitations of the `EV3` motors and sensors; the methodology and results of these tasks; and discuss possible improvements to the algorithm.

For the robot to complete these tasks, it needs to sense and act on the environment. We used sensors to transduce physical quantities in the environment to electrical signals. These electrical signals (such as pixel intensity of the color sensor) allow the robot to make informed decisions on the environment.

The effectors of the robot are wheels. They allow the robot to rotate and move around the two dimensional plane (such as on a floor where these tasks are tested out). The actuators of these wheels are electric motors - `LargeMotor` and `MediumMotor`. We used PID controllers to moderate the actuators towards the desired goal state.

In the next few subsections, we elaborate the motors and sensors and our understanding of them from experiments we ran. Then, in Section 2 we go into details of the methodology we employed in each task, with the corresponding results elaborated in Section 3. Lastly, we evaluates the results and discuss the performance of our algorithm. The code for the project is listed in the appendix.

1.1 Motors

The `LargeMotor` and `MediumMotor` were used for the wheels and servo respectively. To control how fast the motor turns, `duty_cycle_sp` - the width of the voltage on-time of each period in percentages, is used. A larger duty cycle corresponds to more voltage transmitted to the motor, and hence more power. And a negative duty cycle causes the motors to spin in the opposite direction. We found that using `duty_cycle_sp` to regulate the movement of the robot and servo is convenient and effective. The following two experiment allows us to map time, distance, tacho-counts and duty cycle:

First, we investigate the relationship between the duration the wheel motors are ran, t, and the distance, $dist$, it covers, as shown in Figure 1. It is conclusive that $t \propto TC \equiv Dist$, since number of tacho-count is proportional to the distance covered by the wheel.

Next, we investigate the relationship between the duty cycle and the $dist$ for a fixed duration. Figure 2 shows that the distance covered by the robot is proportionate to the duty cycle of the motor for a fixed t.

With all these information, we are able to find the ratio for the number of tacho-counts required for the robot to move a certain distance (in centremetres), given a `duty_cycle_sp`. This is implemented in the `Robot` class.

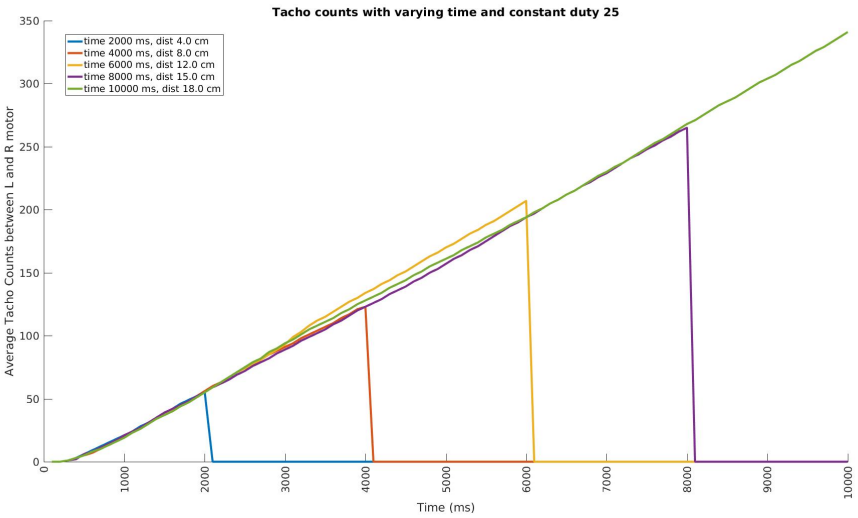

Figure 1: In this experiment, the `duty_cycle_sp` is set to be 25 and the distance covered by robot time is manually calculated.

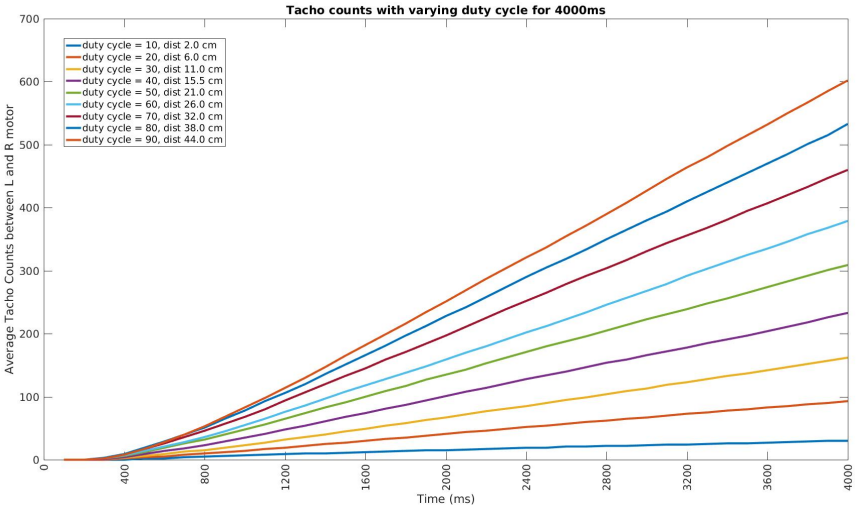

Figure 2: The distance covered by robot in 4000ms increases linearly as the `duty_cycle_sp` increases.

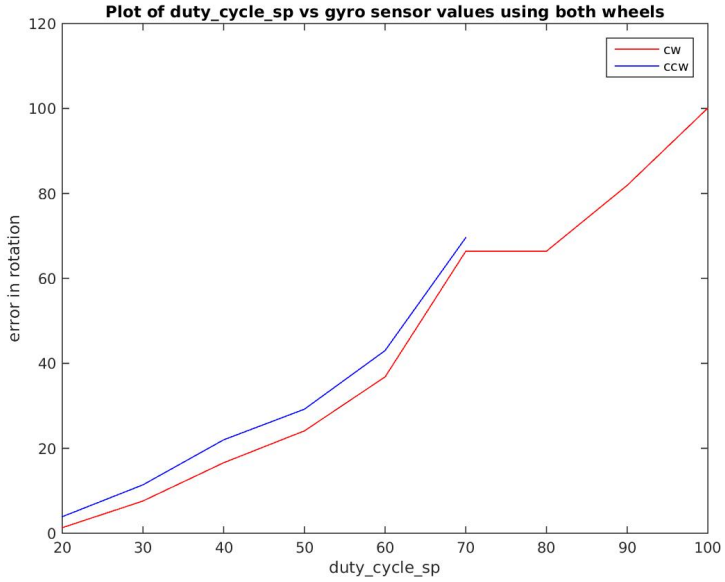

Figure 3: Error in rotation for varying `duty_cycle_sp` for rotating $\pm 90°$ using both wheels. No useful data was collected for `duty_cycle_sp` greater than 70 as the rotation was too fast and violent.

1.2 Sensors

The gyro, colour and ultra sonic sensors were used to measure current angular position, reflected pixel intensity and distance from object, respectively.

Gyroscope

First, we monitored the relationship between running large motors and changes in gyroscope readings. The large motors are commanded to rotate until a specified gyroscope value is read. This is done in two different set-up: 1) both motors are set at equal speed but turns in opposite directions; 2) only one motor was ran. The results are plotted in figures 3 and 4:

It is conclusive that:

1. The error in rotation increases as the `duty_cycle_sp` increases and the

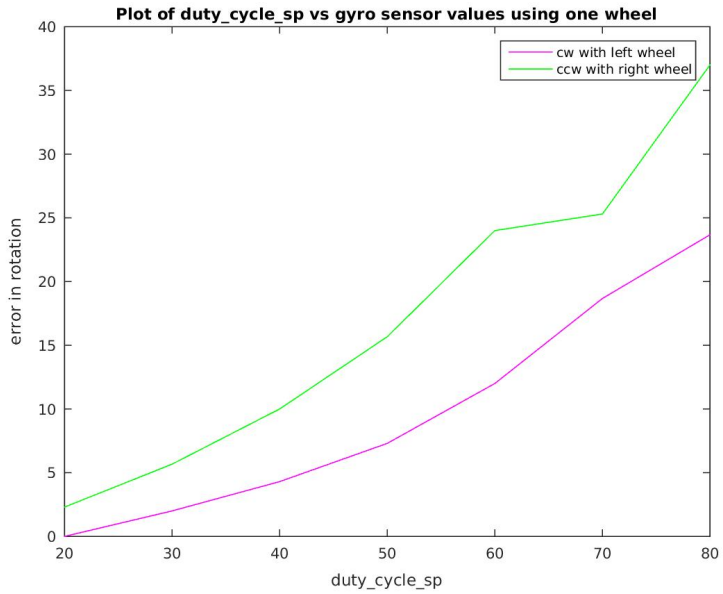

Figure 4: Error in rotation for varying `duty_cycle_sp` for rotating $\pm 90°$ using only one wheel.

errors are similar in both directions when both wheels are run.

2. The two `LargeMotor`s we had were dissimilar. The right motor turns more than its counterpart when the same `duty_cycle_sp` is applied, causing the rotation to be greater in the right than left. The same is experienced when the robot is moving forward but more apparent when turning.

Ultrasonic Sensor

The ultrasonic sensor measure the distance from itself to an object using the difference in period for the reflected wave.

It is observed that:

1. Range of sensor: 3cm to 255cm; But any object within the range of 3cm to the sensor will be read as 255cm away.

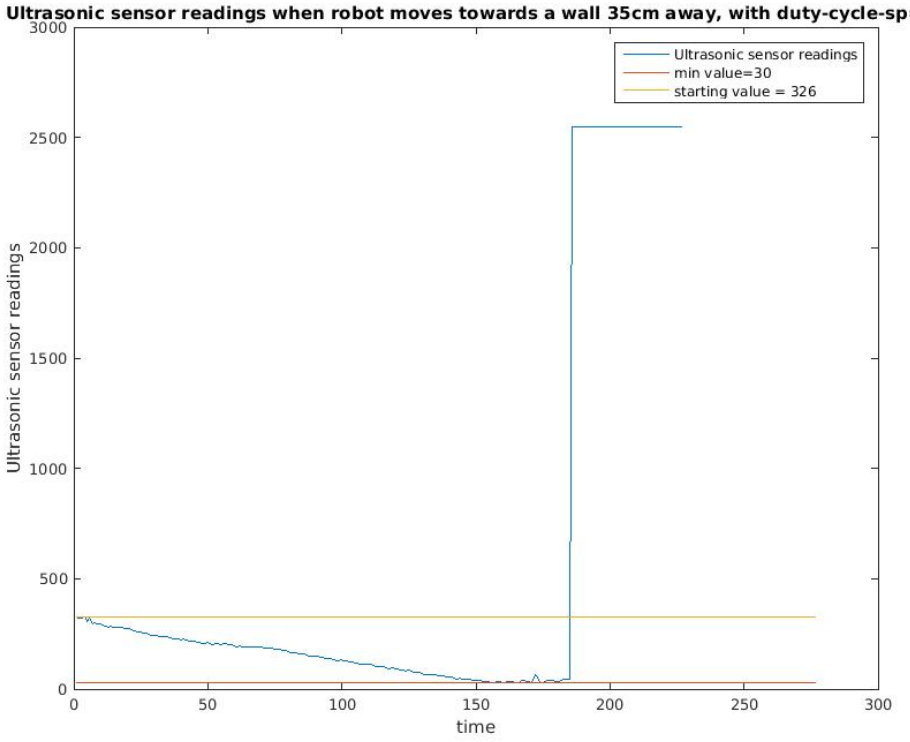

Figure 5: The ultrasonic sensor has a range of 3cm - 255cm. When sonar is closer than 3cm to a surface, the value shoots up to 255cm

2. When a flat object is 3.5cm away and sonar faces it directly: sonar reads 3.4cm.

3. However, while rotating the object, the edge of the object reads 255cm. This is due to the wave being reflected away from the the object's surface and it is greatest when at a corner.

1.3 Our Build

Putting these motors and sensors together, we have our robot.

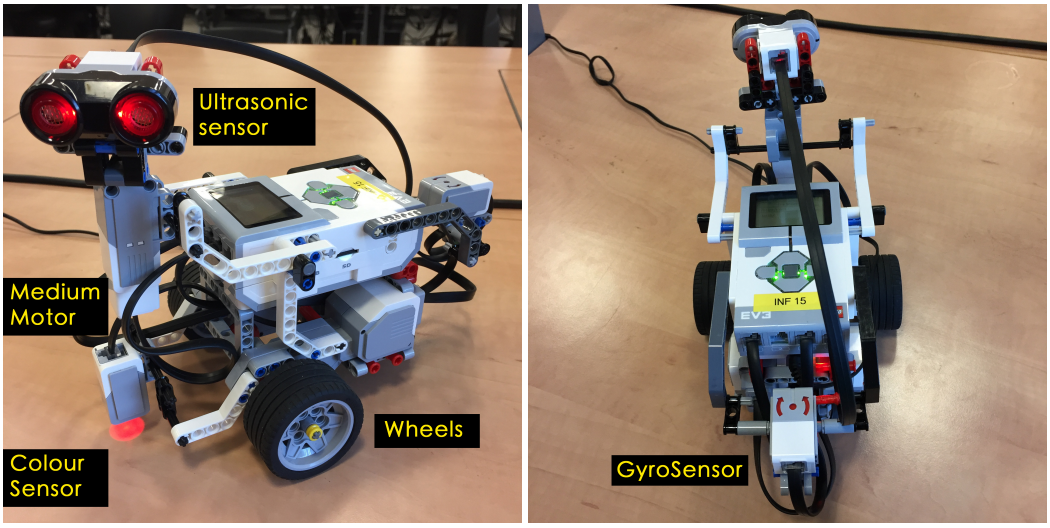

Figure 6: Set up of the robot.

2 Methodology

In this section we describe the decisions made by the robot to complete the task. As control theory and PID controllers are central to all our algorithms, we provide an understanding of the concept before delving into task-specific methodologies.

2.1 Control Theory Revisited

We saw that the motors are imperfect, and sensors are noisy. Control theory allows us to monitor the state of the plant (robot) and then adjust the actuators using $signal(t)$ according to the difference, $e(t)$, from the desired state, $value_{desired}$. The effectiveness and limitations of the PID controllers will be evaluated in the next section.

$$\begin{aligned} e(t) &= value(t) - value_{desired} \\ signal(t) &= K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt} \end{aligned} \tag{1}$$

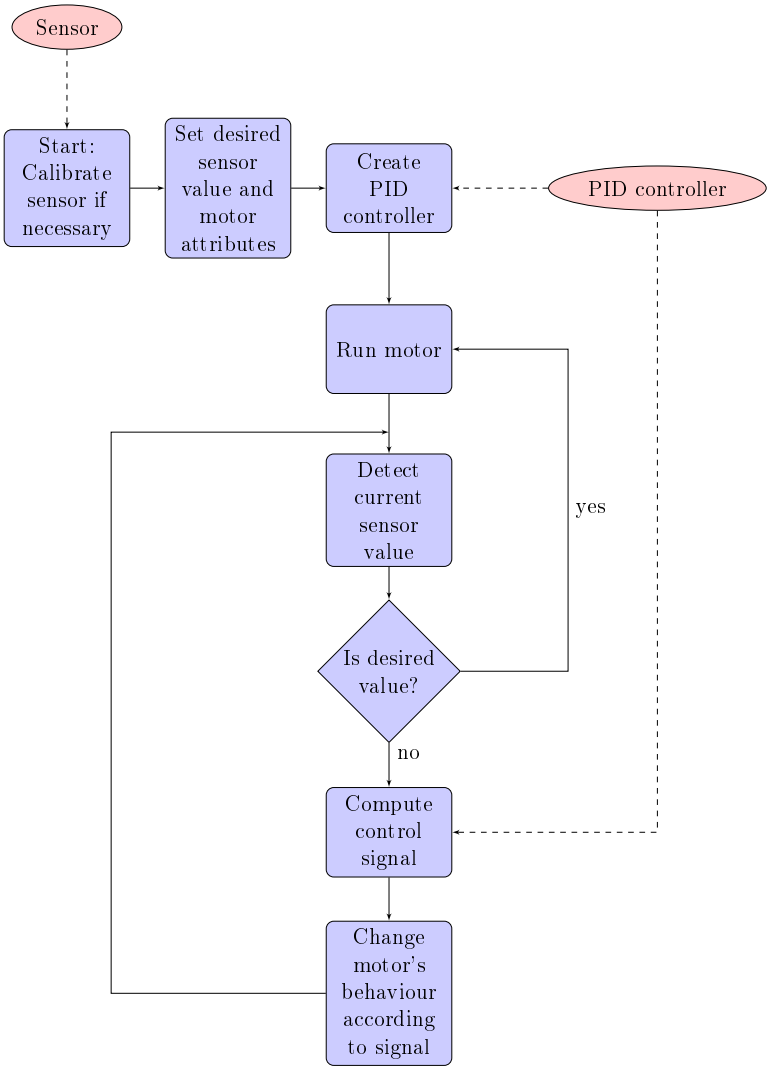

Figure 7: Flowchart for PID controller.

We created a generic `Controller` class that outputs a $signal(t)$ based on $e(t)$. The usuage of the class is fairly generic, and an outline of it is presented in Figure 7 .

2.2 Task A: Follow a line

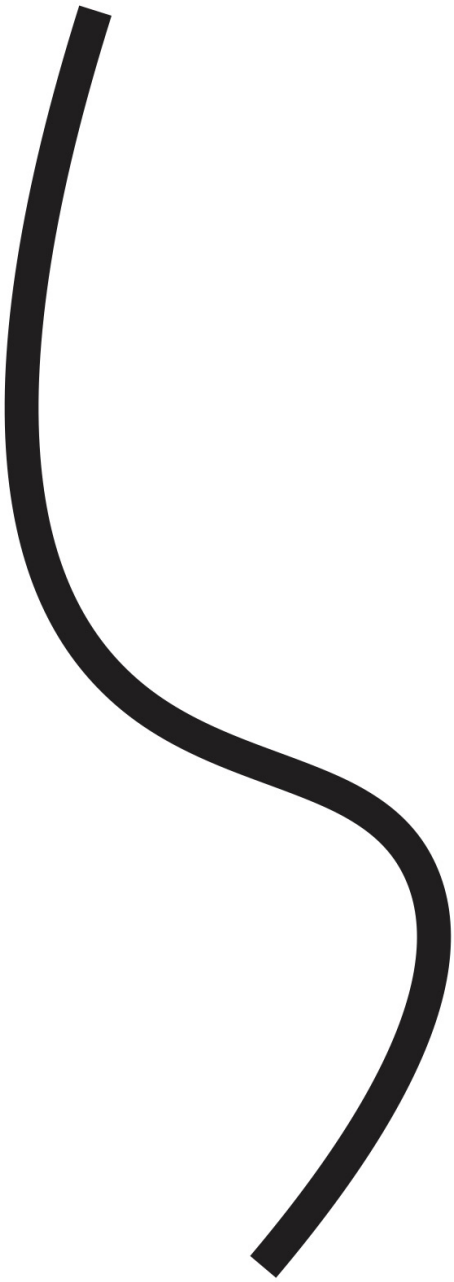

Figure 8: Picture for task A.

In this task, we aim to have the robot follow the edge of the line - which will be the desired state for the PD controller. As the robot *"hugs"* the line and goes in a clockwise fashion, a positive error corresponds with color sensing more *white* than expected. The left wheel will be sped up to reduce this error. The reverse is true for a negative error.

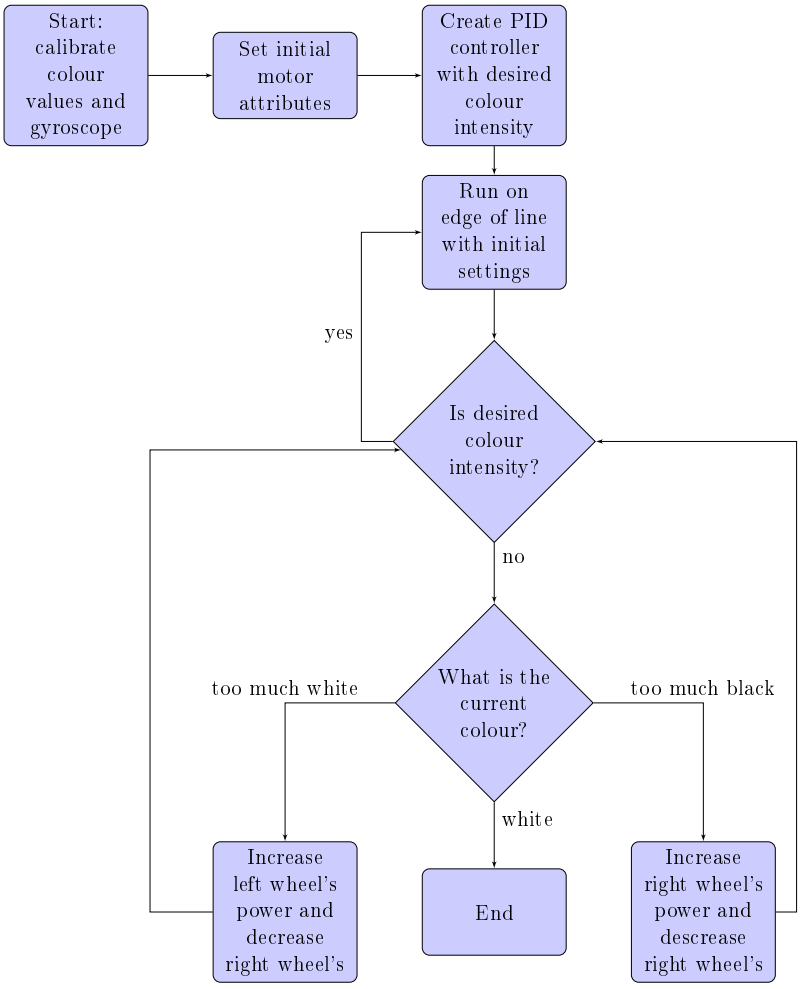

Figure 9: Flowchart for task A: Follow a line

The controller error is taken into account at each time step of the movement. A `duty_cycle_sp` is used as reference value, in our case `20`, so that the robot keeps on moving even if the error is negligible.

```
1 while True:
2   signal, err = control.control_signal(col.value())
```

```
3   if abs(v+signal) >= 100:  signal = 0 # prevent overflow
4   L.run_direct(duty_cycle_sp = v + signal)  # going CW
5   R.run_direct(duty_cycle_sp = v - signal)
6
7   if col.value() > WHITE or io.btn.backspace:
8       L.stop()
9       R.stop()
10      break
```

The `WHITE` value stops the movement if the end of line is reached. However, in many cases, the acute turns in the task (see Figure 17) causes the robot to *falsely* breaks out of the loop. Nevertheless, this was avoided after tuning of the PD controllers discussed later.

2.3 Task B: Follow a broken line

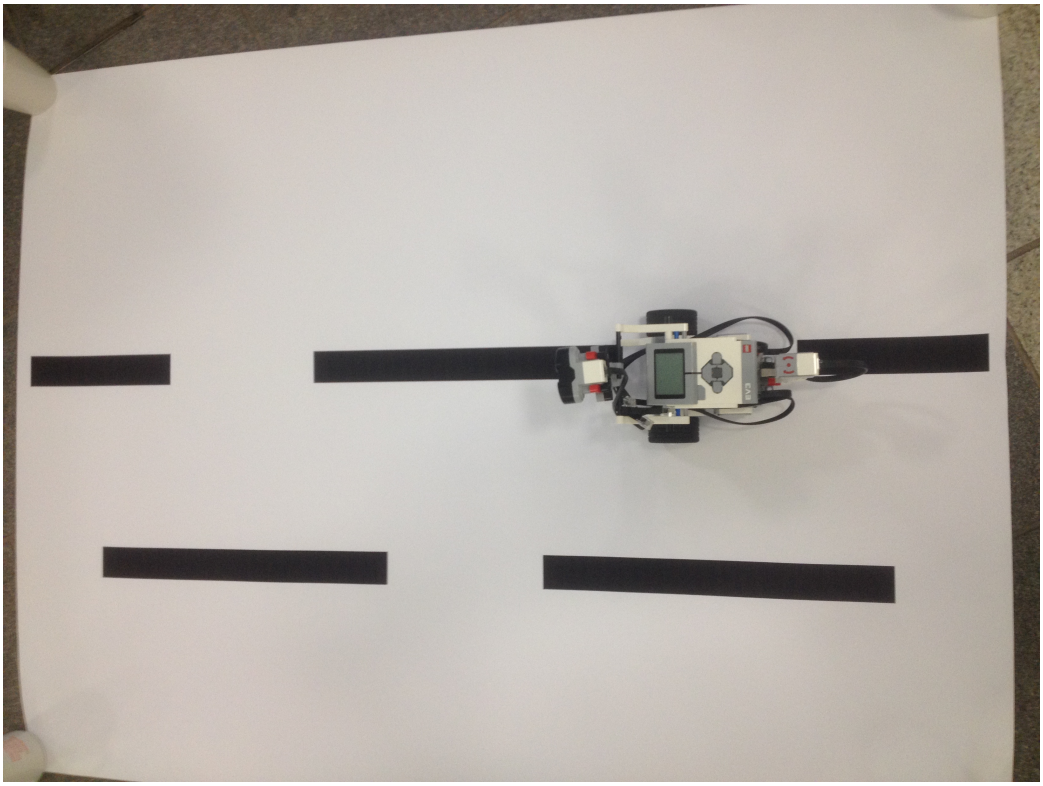

Figure 10: Picture for task B.

In this task, the robot is programmed to start on the left line. In general, the robot aims to stay on the edge facing each other of the lines. In contrast to following the edges facing the border of the paper, we find that this approached is preferred as it prevents the robot from being confused with the multiple lines it will have had seen before turning.

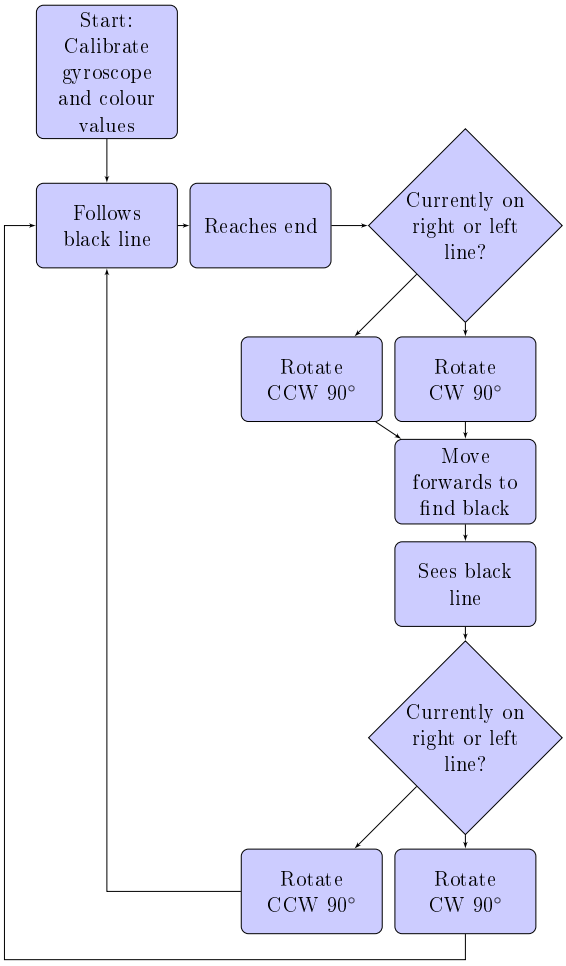

Figure 11: Flowchart for task B: Follow a broken line

Also, as the problem is simply one that changes the direction whenever the line is found, a recursive `main()` function that calls the following functions is used.

	Functions	Outcome
1	`follow_line`	Using colour PID controller, robots moves along the line by adjusting `duty_cycle_sp` on the wheels. Stops when white is detected.
2	`turn_on_spot`	Rotates both wheels in opposite direction in `duty_cycle_sp` adjusted by the gyroscope PID controller.
3	`forward_until_line`	Moves forward until the edge of a line has been detected. Using gyroscopescope PID controller to maintain a straight line.

2.4 Task C: Follow a line to an obstacle, circumvent the obstacle and find the line again

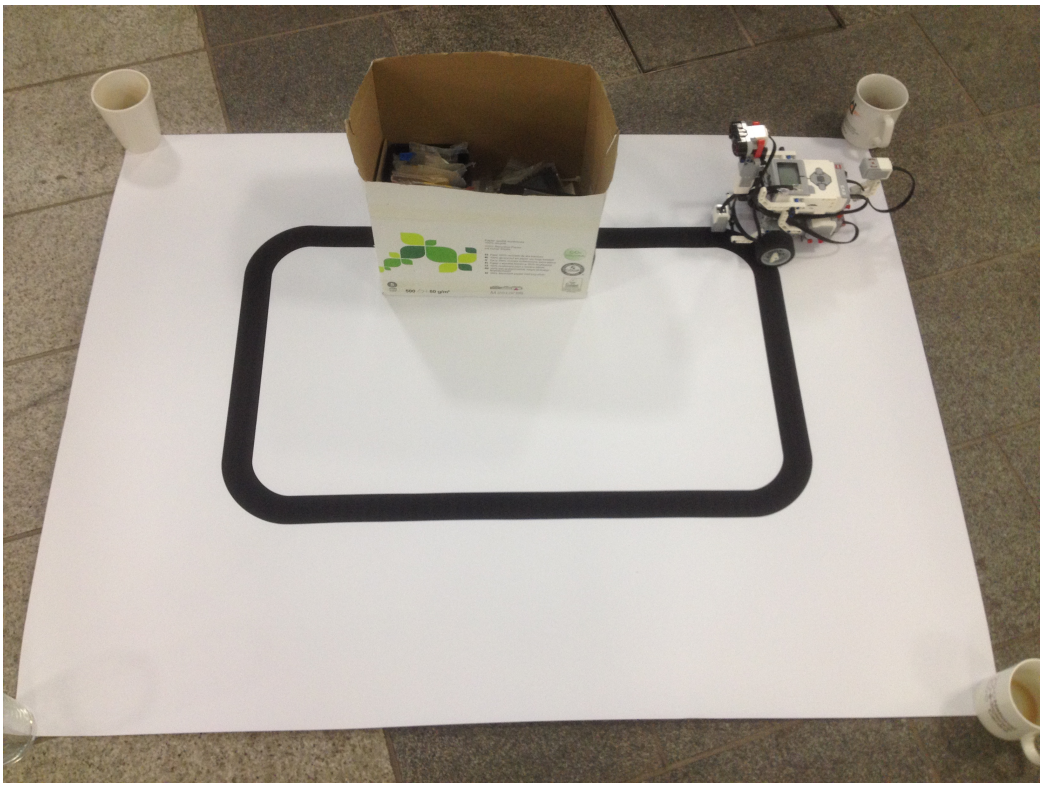

Figure 12: Picture for task C.

Building on task B, we have an object of unknown size for us to avoid while moving along the line. Hence, we cannot naively fix the distance nor amount of turning. However, we do know that the object will have an edge that defines the end of it, and the size of the robot. Hence, we redefined the problem as one that finds the edge of the object, and then avoid it by going along the perimeter of the object while maintaing a suitable distance away from the object. Since out next task is to find the line, and the line may appear anywhere during the course of avoiding the object, we actively sense for the line while we move along the perimeter of the object.

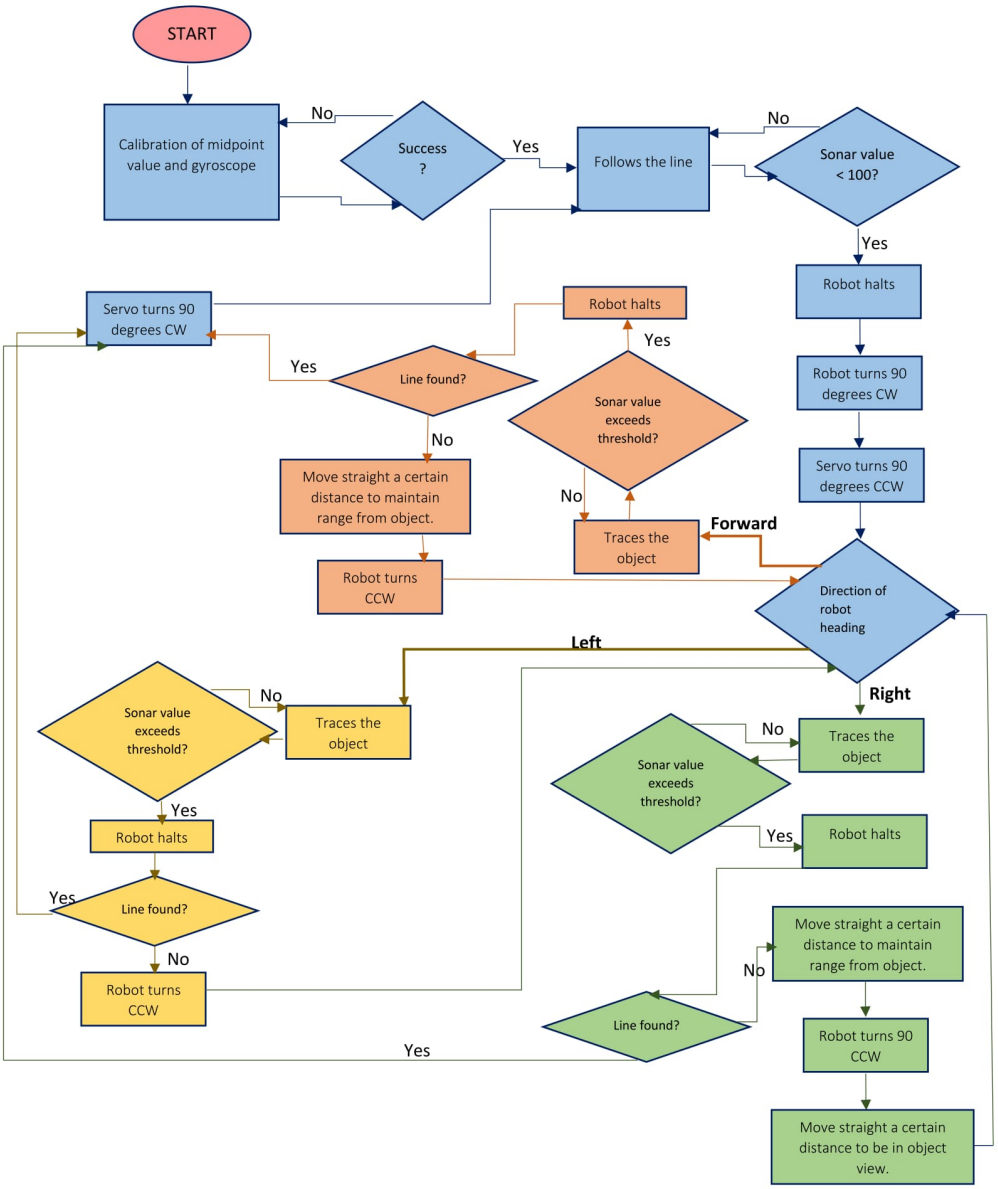

Figure 13: Flowchart showing the steps in task C

In addition to the perimeter, we also note the state of the robot, and refine the direction it should turn with respect to the robot's native heading.

2.4.1 Outline of each function used in the approach:

	Function	Outcome
1	`follow_until_dist`	Using color PID controller, robot moves along the line by adjusting duty_cycle_sp on left and right wheels. Stop when sonar sensor detects a desired value i.e 10 cm.
2	`move_in_range`	The goal is to move the robot in a parallel fashion to the object. Ultrasonic sensor reads the initial reading and feed into PID controller to maintain this value by adjusting speed of left and right wheels. It halts when sonar sensor detects threshold value (edge detection) i.e threshold = 40 .
3	`blind_forward`	The robot moves blindly forward for extra tacho counts. This is to maintain the distance from the object. No PID controller is implemented in this function.
4	`turn_on_spot`	Using gyroscope controller, the robot or the servo rotates at a specified angle on spot. Speed_sp is used to adjust speed of the motors.
5	`calibrate_gyroscope`	Calibration of the gyroscope so that the current orientation of the gyroscope is set to 0.

3 Result

Here, we explore the results of tunning the PID controller and the results for each tasks. As we log the value for the gyroscope and the relevant sensors for the tasks, we have a sense of the PD controller are doing while it is executing the task.

3.1 PID Controller

We use task A as an exaple for calibrating the PID controller. The steps we took are :

1. Increase k_p until it starts oscillating - hence overshooting; then set k_p to be half of that value.
2. using a range of k_d, and find the best value that reduces the overshoot.

The oscillation is obvious in the graph below, as the robot moves along a straight line.

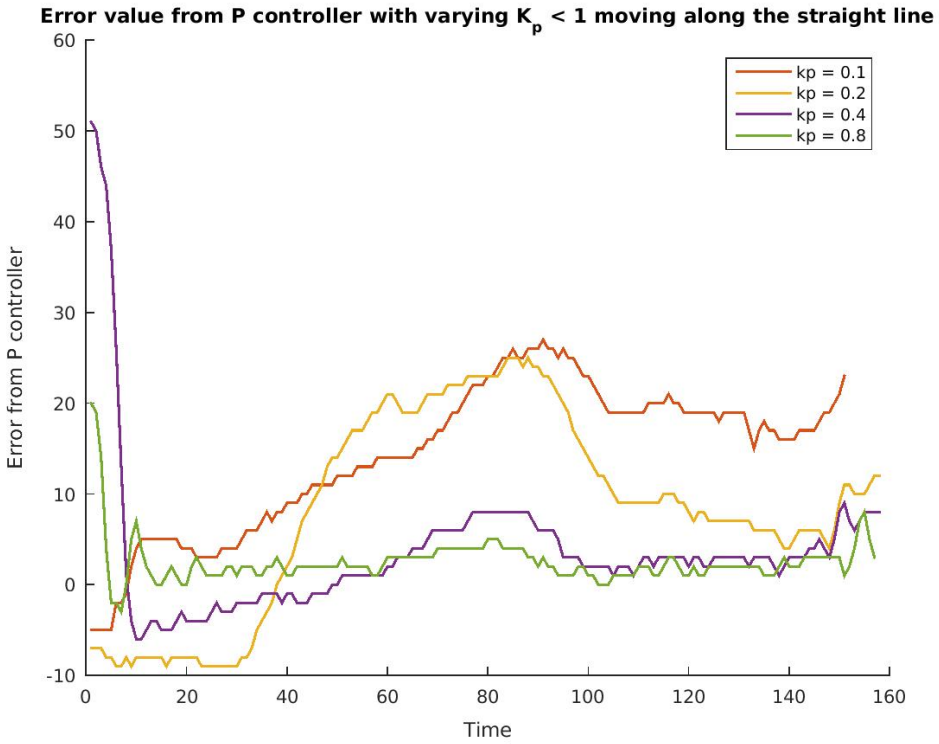

Figure 14: Varying the k_p while moving along the straight line.

The oscillation is even more obvious as it moves along the curve. The overdampened system finds it hard to reduce the error while moving along the curve.

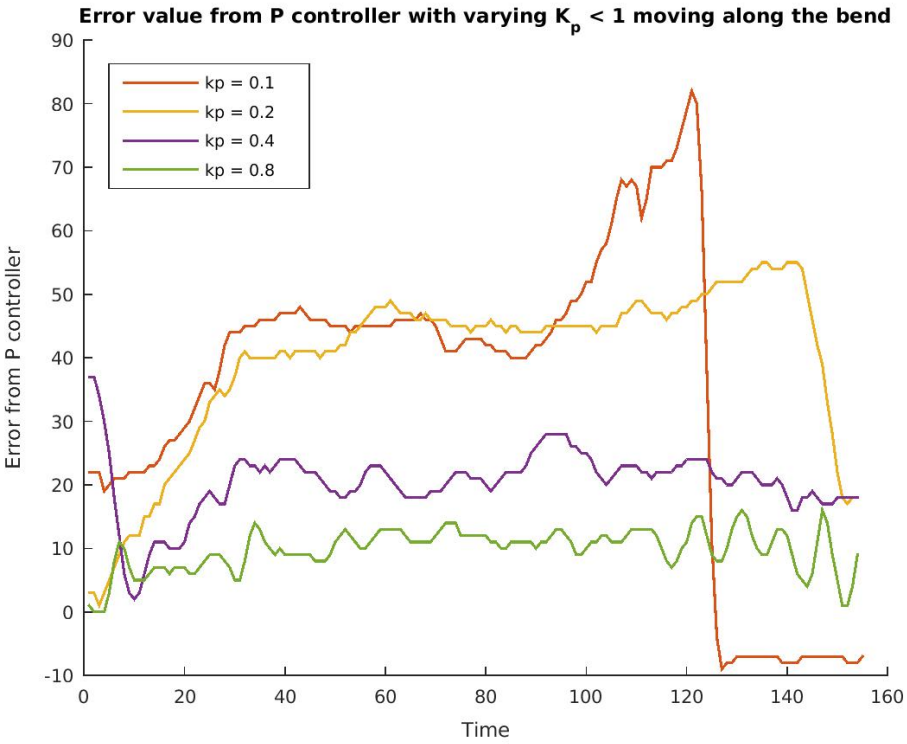

Figure 15: Varying the k_p while moving along the curve.

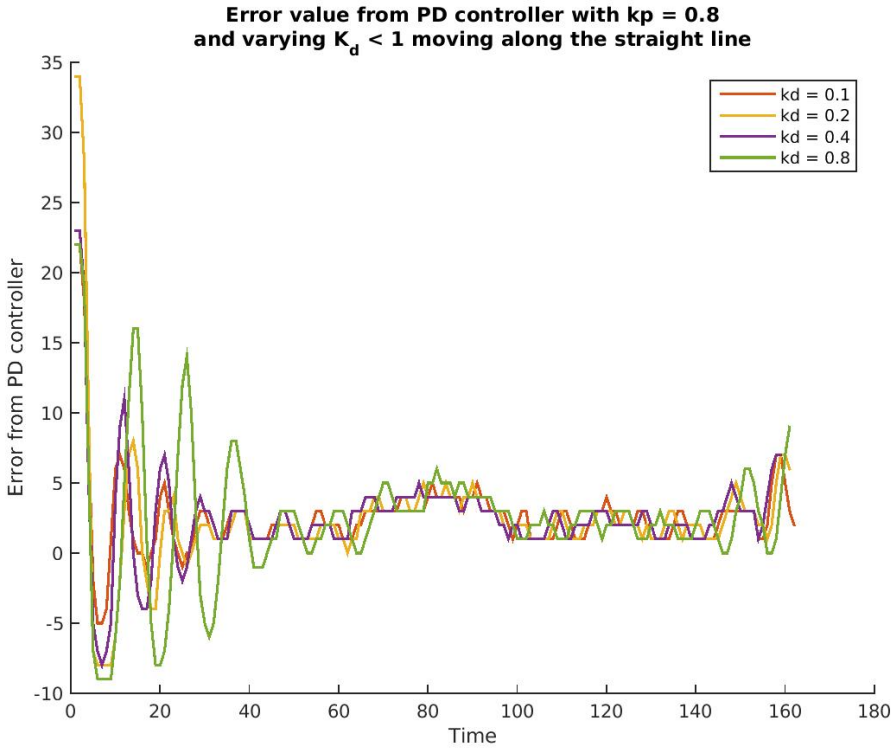

Figure 16: As the k_d increases, the amplitude of the overshoot decreases for robot moving along a straight line with $k_p = 0.8$

3.2 Task A: Follow a line

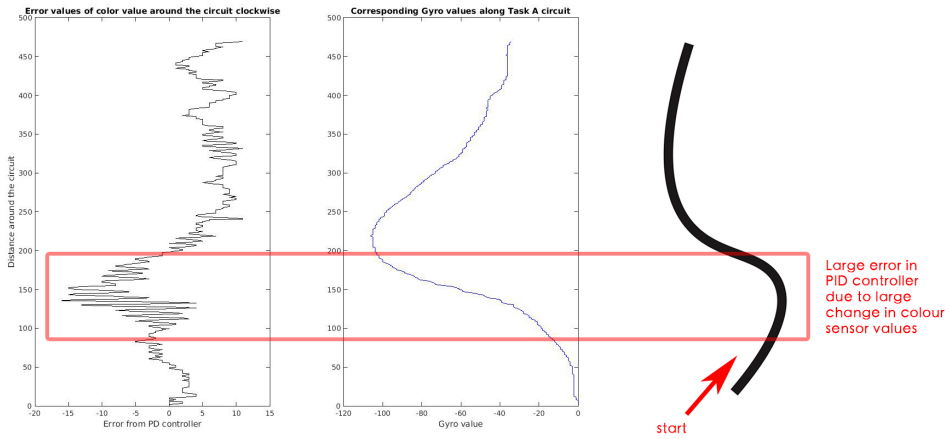

Figure 17: Recorded gyroscope value for task A when the robot follows a line till end.

After tuning the robot, the PD controller for the robot is able to avoid problem of *falsely* halting when it sees the `WHITE` value. This is effective as our `duty_cycle_sp` is small and the effect of a large change in color sensor value is avoided by choosing a small enough k_p to avoid overshooting the desired state.

3.3 Task B: Follow a broken line

Our robot can follow the edge of a line, sense that it has reached the end of the line, rotate perpendicular to the current line and face the next line, move forward until a black line has been detected, rotates to face straight and repeat. Figure **??** shows the recorded gyroscope and colour sensor values, and explains what they tells us about the behaviour of the robot.

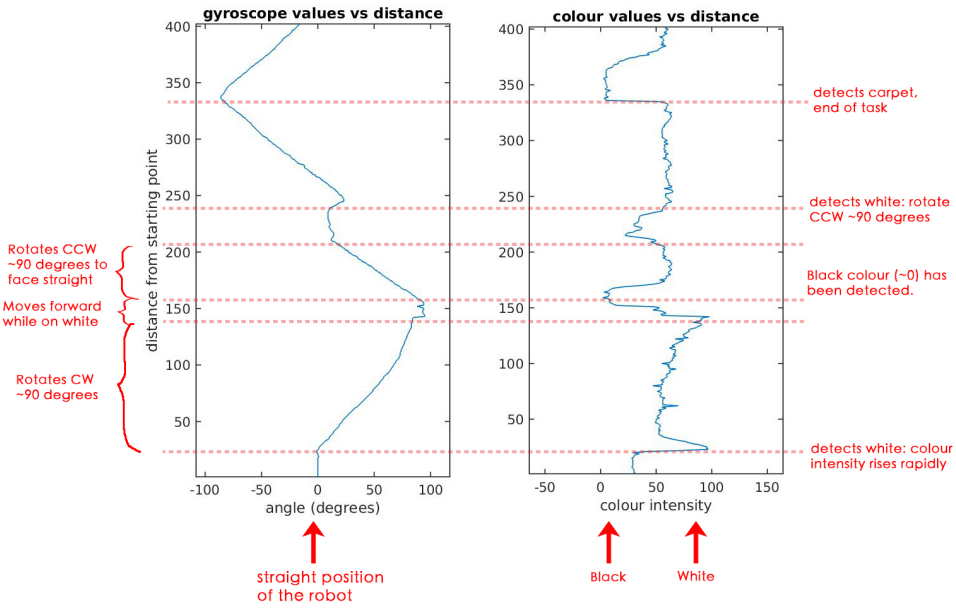

Figure 18: Recorded gyroscope (left) and colour (right) values for task B when the robot follows a broken line.

3.4 Task C: Follow a line to an obstacle, circumvent the obstacle and find the line again

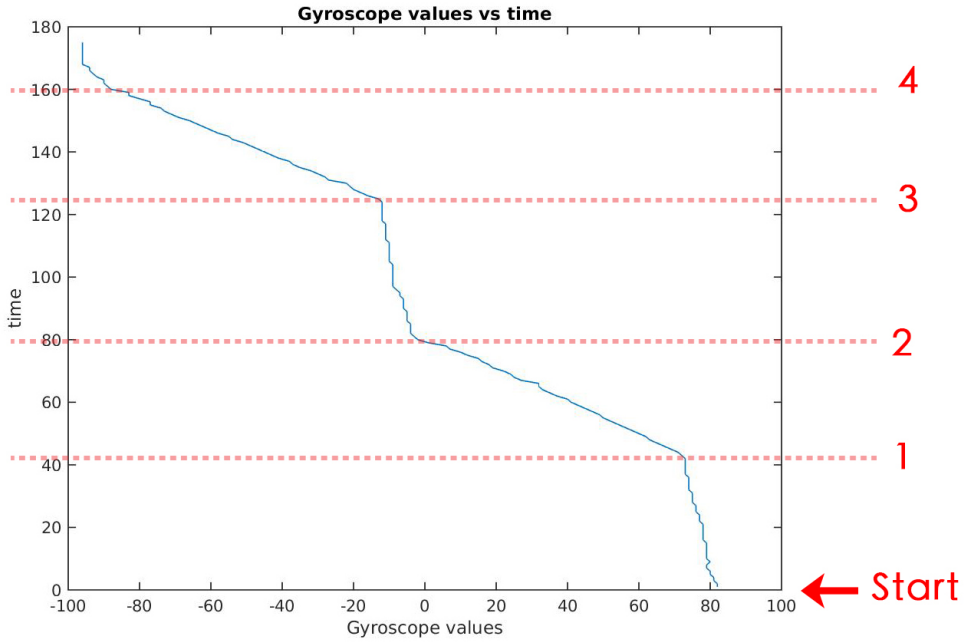

Figure 19: Recorded gyroscope values for task C when robot encounters the object. This shows how the robot circumvents the object, starting from the object being on its left side.

1. From start - 1: Robot starts off with facing 90 degrees (east) where the object is on its left-hand side. However, its orientation is not always perfect, in this case it starts with facing 82 degrees. The robot then traces the object until the edge is reached by using sonar PID controller to maintain its range from the object when threshold is not reached. After that, it moves extra tacho counts to maintain the distance from the object. The gyroscope value is decreasing over time, suggesting that the robot is actually getting closer to the object. However, this is rather trivial (82 degrees - 78 degrees). By eye-balling the graph, the slope of the first half is steeper than the second half. This is likely to be because we use ultrsonic PID controller to maintain the distance

in the first half while blind_forward function that naively moves the robot forward is used in the second half. Therefore, if the wheels are not correctly aligned to begin with, the robot will move closer to (or further away from) the object without any correction. Anyhow, gyroscope value still decreases over time when ultrasonic sensor PID controller is used. Two possible explanations include imperfect calibration of the PID controller and sonar being noisy hence gives varying values for object of the same distance away which (falsely) triggers the PID controller.

2. From 1 - 2: Then the robot rotates 90 counter-clockwise, reaching 0 degree on the gyroscope and is ready to head forward.

3. From 2 - 3: The robot continues to trace the object by moving forward. It halts when the edge is reached then moves extra tacho counts to maintain the distance from the object. The gyroscope values still show that the robot has the tendency to move closer to the object. Possible explanations are previously discussed.

4. From 3 - 4: The robot then turns 90 degrees counter-clockwise and moves forward until the line is found. It stops when line is found.

4 Discussion

Possible sources of errors, further improvements and future directions will be discussed in here. In general, there are a few things that could be improved for the whole assignment:

1. Possible source in error comes from the equipment themselves as there are delays in reading sensor values and sending commands to motors.
2. The rotation is not perfect. It became obvious that the rotation is not perfect. One possible source for this is the quality of the motor, as we noticed that the left motor required more power to than the right motor. Another possible source of error is that the two motors are not synchronised and thus the commands cannot be carried out simultaneously, resulting in lags. The only way to improve this is to purchase motors with higher performance.
3. If there had been more time, we could experiment with tolerance for PID controller to obtain a smoother movement.

4.1 Task A: Follow a line

1. Adding tolerance might improve the smoothness of the robot.

4.2 Task B: Follow a broken line

1. Due to the drift in gyrosensor and that the motors do not have the same performance (left wheel is slower), in rotating the robot to a desired angle, we give its a little offset in order to correct this problem.
2. When the robot detects a line, it is facing perpendicular to the line. One possible improvement is to ensure that when it rotates CW or CCW to be on its 'straight' position, it stays on the black line so it never lose track of it.

4.3 Task C: Follow a line to an obstacle, circumvent the obstacle and find the line again

1. Gyroscope readings from task C shows that the robot has the tendency to move closer to the object. If the size of the object was bigger, meaning longer distance to travel, the robot may potentially crash into the object. Hence, we could explore further with the calibration of the ultrasonic PID controller to get better error adjustments.

2. Instead of purely relying on the ultrasonic PID controller, we could integrate gyroscope PID controller in to enhance the performance of the range to be maintained as well as achieving a more 'actively' sensing the object style.

References

[1] ev3dev community Hempel R., Lechner D. Ev3 dev documentation. URL http://www.ev3dev.org/docs.

[2] J. Sluka. A pid controller for lego mindstorms robots. URL http://www.inpharmix.com/jps/PID_Controller_For_Lego_Mindstorms_Robots.html.

[3] Introduction to pid controller design. URL http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID#1.