Milestone 4

University of Cape Town
Engineering Design (ME)
EEE3099S

Project Report

November 2020

Prepared By

Azhar Ebrahim (EBRAZH001)
Taqi Enari Syed (ENRMOH001)

Table of Contents

Introduction

The design specifications required the robot to act as a line follower.
Given were 3 different tracks, each with a unique mathematical curve. Also given was a complete Simulink model of the robot. The model included not just the robot's motors but also a line sensor model that could be replicated and coded in order to simulate multiple sensors. In order for the controller design to be deemed a success, it had to allow the robot to follow the line and then stop completely at the finish line. Initially, two preliminary designs were brought forward, each with their own merits and potential downfalls. Through rigorous testing and troubleshooting procedures, the better design was chosen and consequently implemented.

Another vital facet of system design was the consideration of the given cost function (Provided below). The integral was evaluated with the given bounds and henceforth the minimization process of this cost parameter was initiated. It was thus found that the effect of time and number of sensors both affected the cost function in a linear way.

Cost Function simplified:

$$J = \frac{n}{5}\int_0^t e^2 dt$$

$$J = \frac{n}{5} e^2 t$$

Where **n** is the number of sensors, **t**, the amount of time taken to complete the race, and **e**, the error between the centre of the robot's front and the line.

Modeling and Experiment

The first step in allowing the robot to communicate effectively with the line sensors was to format the sensor output in a way that allowed the robot to process and thereafter act upon the dynamic sensor output. This was done by binarizing the sensor output through the use of comparators. The comparators were configured with a calculated threshold value.

Prior to the threshold calculations, the respective readings of black and white needed to be determined. This was done through a simple test consisting of moving the sensors as far wide on the sensor as possible. In this case 0.175m from the centre. The duty cycle was set to 0 to disable any motion and thus it was observed that the sensor being on white rendered an analog reading ranging between 0 to 0.48. To find surety about the reading of black type, the sensor was moved to the centre whilst still keeping the duty cycle at 0.

Threshold of comparator = (minimum{Black output}+maximum{White output})/ 2.
This calculator yielded the “dead zone” value between the black and white sensor outputs so that both colours could be effectively and consistently differentiated.
Scoping the output of the Line sensor, it was determined that the colour black outputs a value between 0.49 and 1, whereas white causes the output to be between 0 and 0.48.

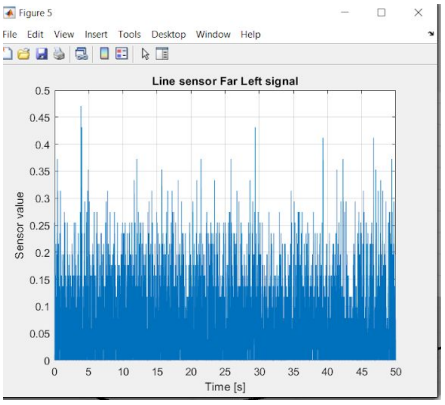

Figure 1: Line sensor output on white

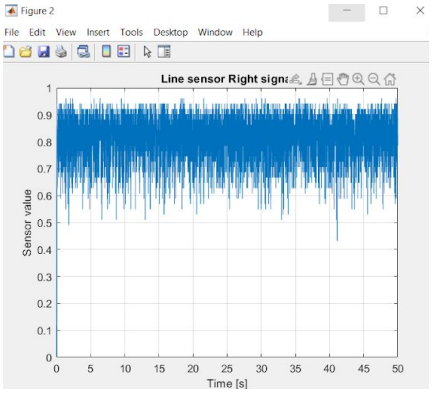

Figure 2: Line sensor output on black

Mechanical Design

A total of four sensors were placed on the robot. The intention was to have two placed closely together near the y-axis centre of the robot, with the other two placed on the outside of these inner ones. The inner two sensors would execute the line follower algorithm whereas the outer two sensors would facilitate finish line detection. The secondary design conjured in the preliminary stage of the design process had two sensors, however due to the sensors needing to be in close proximity of one another in order to complete track 3, this design was unable to run on track 2 where the line was out of range of the sensors detection range. This was one of the reasons why the 4 sensor design was chosen.

At first, the sensors were placed at the front edge of the robot. However, it was found that due to the orientation of the robot at its starting position in track 2, not only were the sensors at wide positions needed, however, the sensors had to be pushed back as well. The following table shows the sensor positions. As one may see, the sensors were pushed back from the initial front position to the back of the robot to aid in track 2.

Sensor	X position	Y position
Left Sensor	0.05	0.03
Right Sensor	0.05	-0.03
Far-Left Sensor	0.05	0.060
Far-Right Sensor	0.05	-0.060

Figure 3: Table showing the sensor positions

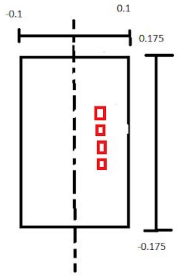

Figure 4: Orthographic view of the robot with the red boxes as sensors.

Algorithm Design

A step by step breakdown of the algorithm design is provided below.
Each step shall have an accompanying explanation in blue below it.

Raw outer line sensor output was fed into the Duty Cycle block:

Duty cycle for speed:
The raw sensor signals are fed into comparators
This binarizes the sensor output. If the signal is on black, it will have a value >0.50 and therefore output a logic HIGH through the comparator. The polar opposite will apply if the sensor is on white.
The line sensor signals of the outer sensors are fed into a duty cycle toggle system.
The toggle system enables the robot to stop absolutely when it detects the finishing line
The duty cycle toggle system consists of an enabled subsystem.
The line sensor signals of the outer sensors are fed into a mux.
Allows the implementation of finishing line detection logic.
This is then fed into a combinational logic block.
If the inputs are 1 and 1 respectively the subsystem is enabled.
The outer sensors being 1 and 1 would mean that both sensors are on black and the only time the outer sensors are ever on black is when the robot is on the finish line.
Enabling the subsystem sets the output of the duty cycle to 0
This forever stops the robot, this output is held.
The enabled subsystem has an initial value equal to the 0.2
This allows the robot to move at the start as the 0.2 is fed directly to the Duty Cycle input of the robot.

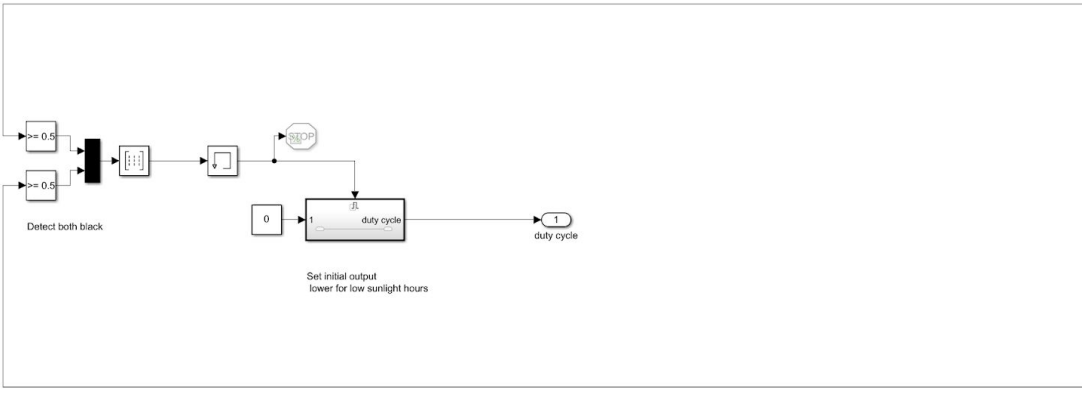

Figure 5: Duty cycle toggle

Algorithm Design for line following:

The inner sensor line signal outputs are responsible for the direction. They follow a symmetrical algorithm.
The output of the line sensor is sent through a comparator.
Unlike the outer line sensor output comparator, this comparator has a logic argument <0.50. Due to this, the comparator outputs a 1 on white and 0 on black.
This value is fed into a gain block with gain = 2 and then subtracted by 1.

Let the aforementioned value be assigned x. Thus the output of the two above steps would yield some value y where y=2x-1. This would allow y to be either 1 or -1.
The value y is fed into the directional input of the robot.
When one sensor is on black, its corresponding y value is -1 and thus its corresponding wheel turns backward. This aids the robot to turn the tight corners in track 3.

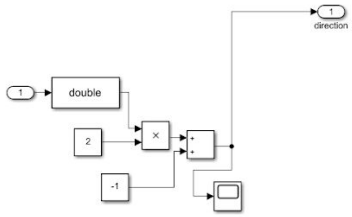

Figure 6: Implementation of directional polarity algorithm

Whole system:

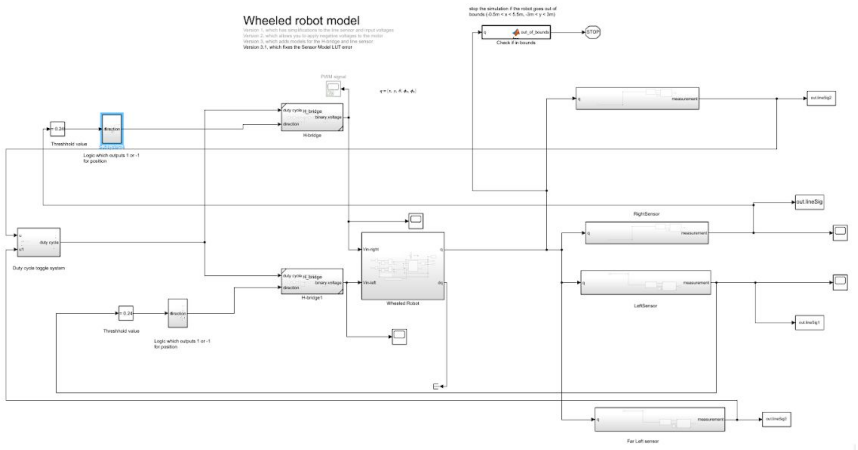

Figure 7: Overview of the whole system

Testing (Mechanical)

The robot was tested by altering the following parameters:
-Position of sensors
-Time of Day
-Speed

Position of sensors
Initially, all four sensors were placed on the front edge of the robot. However, it was determined that we needed to space the inner sensors far apart in order to account for the starting position of the robot in track 2. This meant that the robot would follow the line less accurately which would be evident in track 3. Therefore, the sensors were shifted more to the centre of the robot, which allowed the sensors to be close to another ensuring maximum accuracy. Also, this accounted for the different starting position found in track 2. Placing the outer sensors on the front edge would be ideal to minimise the error,e, however this was unattainable.

Time of Day
Initially all the performance tests were run at 10AM. Lighting conditions in this setting favoured any tests run and thus higher duty cycles could be run. However, to maintain design integrity, the worst case scenario had to be accounted for. Thus, the operation time was changed to 8PM. The lighting in this condition would be poor and thus the sensor would struggle to pick up information accurately. It was thus found that at low lighting conditions, a lower duty cycle was needed to run the robot successfully or otherwise it would miss out important line information and end up not stopping where it is meant to

Speed
The speed of the system is easily controlled by changing the initial condition of the enabled subsystem. Ideally, this value would be as close to 1 as possible ensuring that the robot achieves maximum speed. However, increasing the duty cycle decreases the samples taken, making our robot follow the line less accurately. Upon testing, by varying the duty cycle, it was seen that the robot could operate at higher speeds for track 1 and 2 opposed to track 3. This is obvious, due to the sharp chichanes and curves found in track 3. Therefore, maximising our speed/duty cycle for track 3, did not affect the accuracy of the other tracks too greatly, but merely the time. Increasing the speed to greatly, resulted in the robot not being able to brake properly at the finish line, due to the inefficient following of the track.

Testing (Software)

-MUX mapping logic
-Data Type testing
-Direction for line following

Data Type Testing

Due to the system being fully simulated on a virtual platform, the signals moving through the system weren't limited to solely voltage. It was thus important that Data Type testing between phases was conducted properly lest data type mismatches render Logic errors which are notoriously difficult to pick out and can cause the system to behave incorrectly. The general data type chosen for the system was Double. However, components such as comparators output Boolean variable type and components such as MUXs require boolean inputs. It was due to this, that Cast To Double blocks were implemented between system stages to maintain a standard variable type. Simulink's built-in compiler assisted greatly with diagnoses of any data type mismatches. This allowed the system to be free of any logic errors or runtime errors.

Direction for line following

The tests to determine whether our algorithm for line following was implemented correctly was done in isolation. Initially the algorithm was only implemented on one wheel to see the effects of it. For example we would apply it to our right wheel and see that our right wheel never crossed a black line, but rather rotated backwards, opposed to the left wheel which would simply stop at a black line causing inaccuracies. We then applied the algorithm to both wheels and ran all the tracks, one run with the algorithm implemented and one without. It was evident that the algorithm was helping with the accuracy of following the line.

MUX mapping logic

The MUX logic was pivotal for functionality of the finishing line detection algorithm as it mapped the logic for the wheel duty cycles. The testing for this feature was done through scoping the wheel voltages to ensure that the wheels had absolutely stopped when the robot reached the finish line.

Recorded track times with perfect completion (10am):
%Track 1=27.89 @0.2 DC which is optimised
%Track 2=25.49 @0.2 DC which is optimised
%Track 3=46.19 @0.2 DC which is optimised

Conclusion

The final design was objectively a successful one. It fulfilled all the requirements as per the Milestone document. The finishing line detection algorithm and the tight turns at the end of track 3 proved to be very challenging however through the use of unconventional ideas such as the wheel reverse mechanism, these issues were resolved and design integrity maintained. The robot arguably could have been programmed faster in a real life scenario where the black line is constant rather than a grouping of black dots as was the case in the simulation however, the simulation time as it stands, was perceived by us as satisfactory. The reversing task was however not possible within this design due to the use of the enable block.