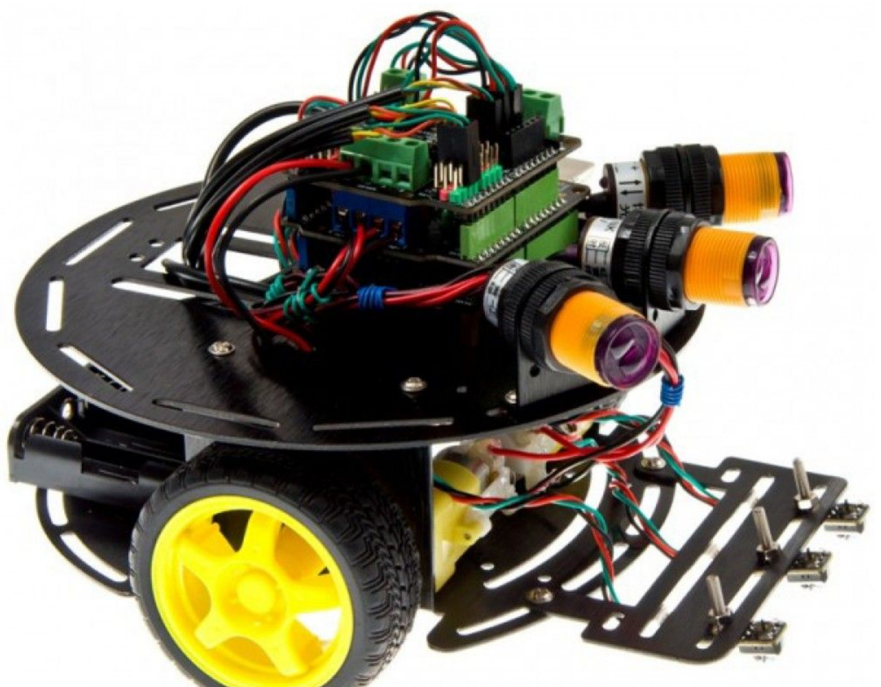
EEE3099S 2021

Milestone 2: Junction Detection

Done By:
Luke Baatjes (BTJLUK001)
Jonathan Campbell (CMPJON005)



Date of Submission:
17 September 2021

Project Description 2

Simulink diagram 2

Line Following Algorithm 3

Line Configuration Algorithm 4

Appendix 5

Bibliography 7

Project Description

The aim of this project was to design a line-following robot in the Matlab and Simulink environment. The first milestone was a guide in that it helps to familiarise the student in controlling a robot using simulation software. To demonstrate this, the robot needed to move in a straight line and stop within 10% of 1 meter. The robot also needed to be capable of rotating 90, 180, and 270 degrees within a 10% error of those values.

To understand Matlab and Simulink, onramp courses were compulsory before beginning the project.. The robots are then simulated in simulink using software compatible with an arduino. This was done using the Robotics simulator blocks that have encoder simulators and robot simulators, to simulate control of the robot.

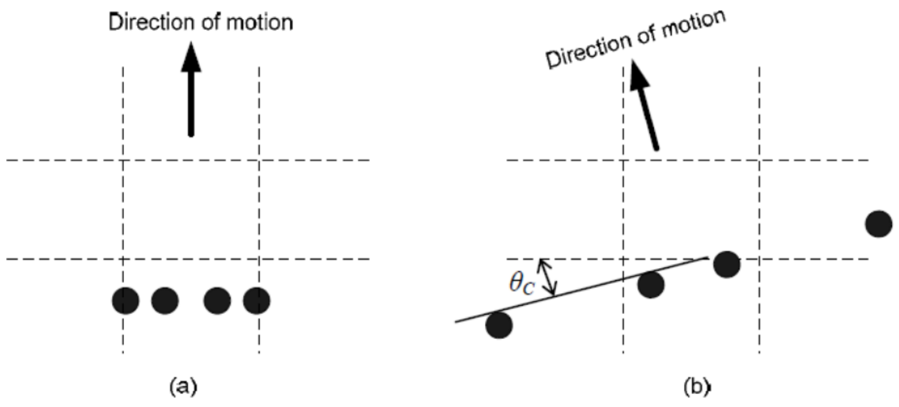
Once the robot showed acceptable control in simulation, digital output and input blocks were used to verify the control on the Romeo V2 board using simulink software, compatible with arduino. The robotic platform makes use of the Romeo V2 board, which is compatible with the Arduino Leonardo board. Therefore, the Simulink Support Package for Arduino Hardware Library was used to communicate with the robotic platform.

Simulink diagram

The simulink diagram for milestone 2 is not that different to the simulink diagram from milestone 1 with the exception of the added stateflow charts to control junction detection. For this milestone a line sensor simulation block would provide the four sensor outputs which were sent to the Line Following and Line Configuration algorithms. The v and w outputs (seen in figure 3 in the appendix) of the stateflow charts are then converted to angular wheel velocities for the left and right wheel. These wheel velocities are sent to a lookup table and then passed to the motors for simulation using a robot simulator block. To simulate the LED, The d (duty cycle) and p (period) was received from the stateflow and then sent to the PWM (square wave generator seen in figure 1 of the appendix) which then controlled the flashing of the LED which could be seen using a scope block in simulink.

Line Following Algorithm

To construct the line following algorithm all we use is the 2 middle line Sensors 2 and Sensor 3.We don't need to use the other line sensors as those sensors are too far away thus they will only trigger when the robot is too far away. There will create a large error angle when using between the angle of the robot and line.



Sensors arrangement: (a) sensors placed too near, (b) sensors placed too far

To make sure the robot is staying on line a stateflow block which is continuously being run is used to control the direction of the robot. To do this the robot is set with a small initial angular velocity to the right or left depending on which sensor has reached the white background. In the line following block (see figure 4) there is a left turn state and a right turn state which are connected with 2 conditions which allow them to switch between each other.

When the switch is triggered the robot starts moving to the left and straight until Sensor 2 detects the white environment back-round, this allows the stateflow chart to change blocks because the condition is reached. The right turn block then sets the angular velocity to the to turn to the right this makes. Thus the robot does not turn as sharply. When the Sensor 3 reaches the other side of the line the condition will be triggered and the left turn block will start, thus going back and forth until it has reached the other side. This makes the robot move in a zig-zag path to reduce the amount of zig-zags the angular velocity can be reduced so that the robot moves slower this also prevents the robot from having a large error angle as to get the best line detection it need to be moving perpendicular to the crossing.

Line Configuration Algorithm

The line detection algorithm took some time to be developed. First a flow diagram was developed to visualise the flow of control from one state to another. Once this was established, implementation of the actual simulation from the flow diagram took place.First the simulink part of the simulation was implemented and then the stateflow charts were implemented thereafter. The simulink implementation was described above therefore, the stateflow algorithm is described next.

In the stateflow implementation the angular wheel velocity, period of the LED, and the duty cycle of the LED were handled. The line tracking algorithm was used again just as in the Line Following algorithm, but, in this case the Line Configuration Algorithm handled the angular velocity of the wheels for turning, whereas the Line Following Algorithm was responsible for setting the wheel velocity.

From the line tracking state we would transition to the different detection states, namely, the T junction, the Cross junction, the Dead End junction, the End of Maze junction, the Right T junction, the Right Turn junction, the Left T junction, and the Left Turn junction. The sensor values for each of these states are summarised in the appendix beneath figure 1 in the appendix. As described in the Line Following Algorithm above, the sensors were used to detect a possible junction. When the robot came to a junction the sensors would be read and depending on which combination of sensors (shown in figure 1 of the appendix) were detected, the robot would then stop. The robot would then move forward some distance then stop and read the sensor values again to be certain of its detection and transition to the corresponding state.

While in the current state the LED frequency was set which was then sent to an LED to flash according to the frequency. The initial state of the LED is in the off state. While tracking the line the LED would flash at a set frequency (1Hz). At a possible junction detection the LED would flash at 5Hz and then once the robot stopped and detected a junction state, the LED would remain in the on state until the switch was turned off.

Appendix

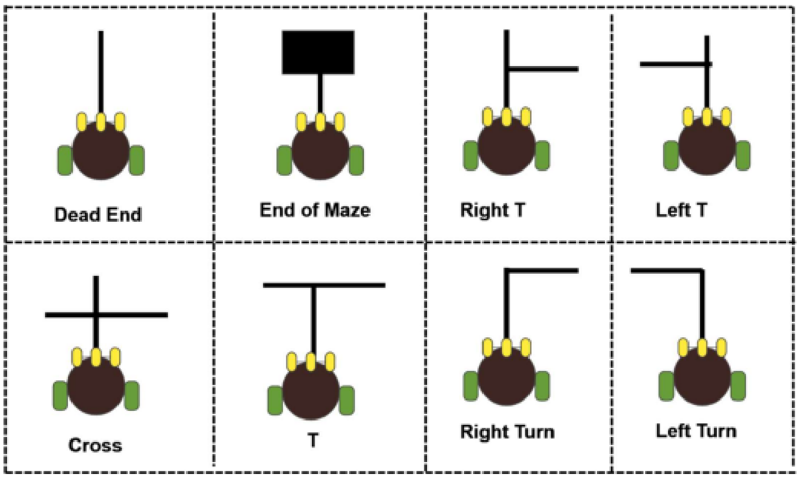


Figure 1: Different detection states

Sensor key: (Sensor access reading can be seen in figure 3)
Sensor 1 - Line(1)
Sensor 2 - Line(2)
Sensor 3 - Line(3)
Sensor 4 - Line(4)

Dead End sensor values: all sensors are logic LOW (0)
End of Maze sensor values: all sensor values set to logic HIGH (1)
Right T: sensor 1 logic LOW, sensor 2, 3, 4 logic HIGH then only sensors 2 and 3 logic HIGH after moving forward some distance.
Left T: sensor 1, 2, 3 logic HIGH, and sensor 4 logic LOW then only sensors 2 and 3 logic HIGH after moving forward some distance.
Cross: sensor All sensors logic HIGH and then on;y sensor 2 and 3 logic HIGH after moving forward some distance.
T junction: all sensors logic HIGH then after moving forward some distance all sensors logic LOW.
Right Turn: sensor 1 logic LOW and sensors 2, 3, 4 logic HIGH and then moving forward some distance before all sensors logic LOW.
Left Turn: sensor 4 logic LOW and sensors 1, 2, 3 logic HIGH and then moving forward some distance before all sensors logic LOW.

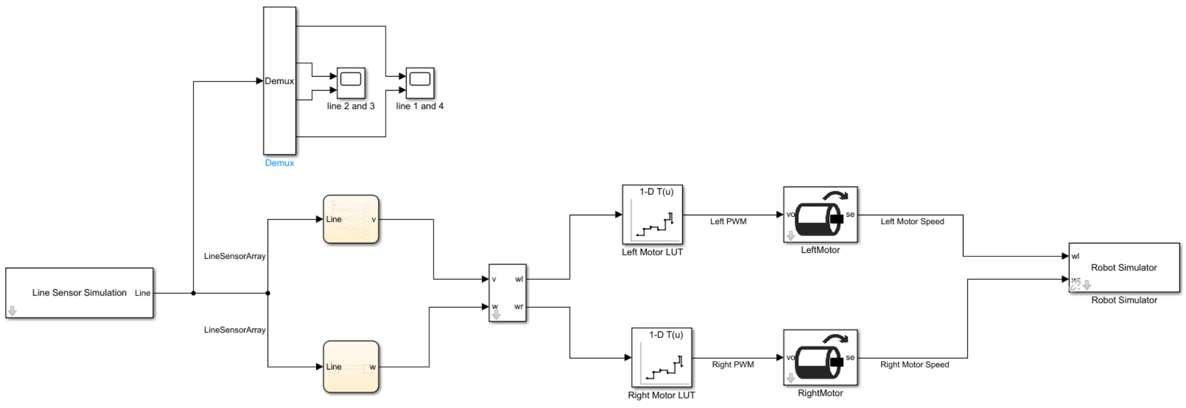


Figure 2: simulation diagram

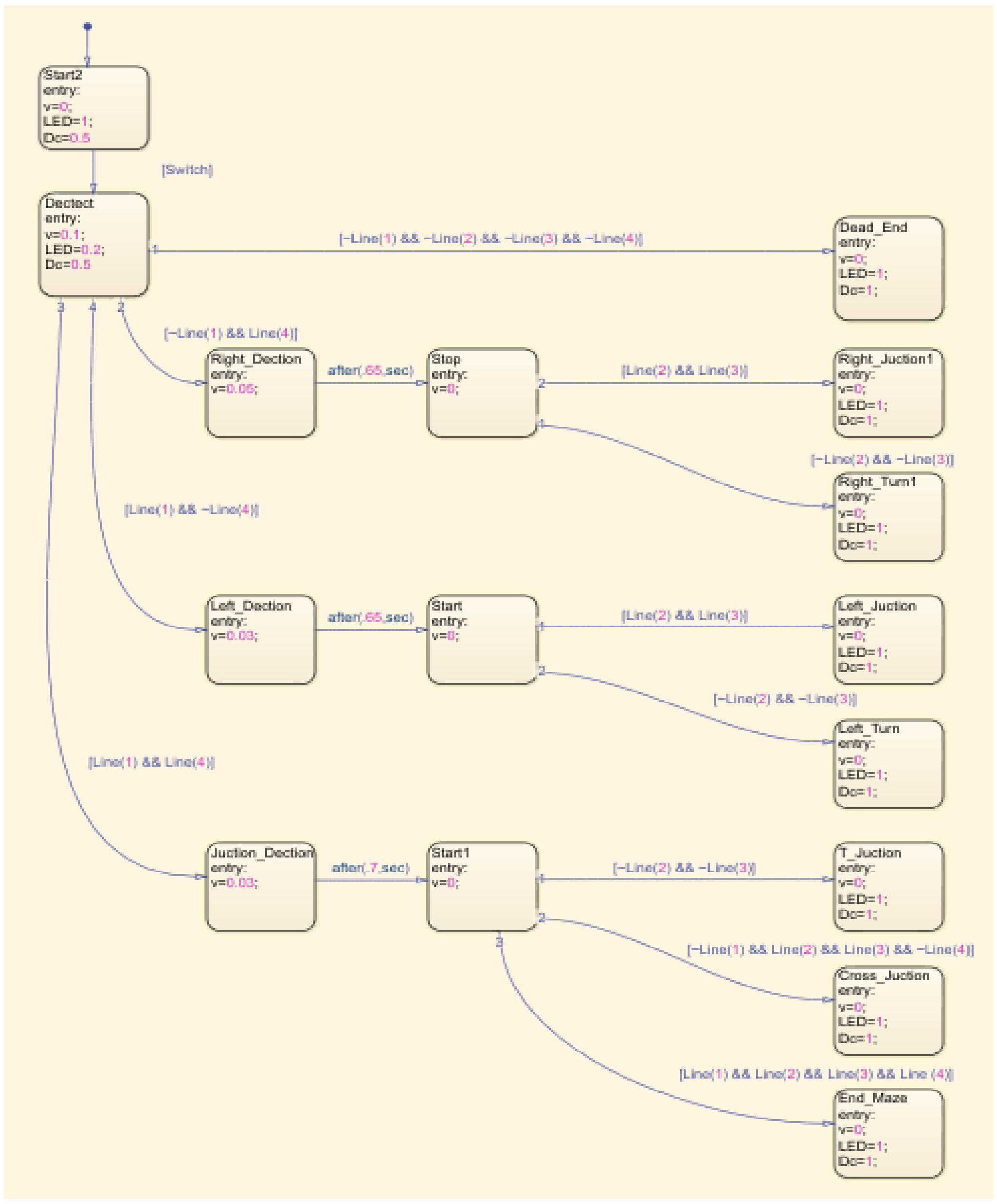

Start2 entry: v=0; LED=1; Dc=0.5
[Switch]
Dectect entry: v=0.1; LED=0.2; Dc=0.5
[~Line(1) && ~Line(2) && ~Line(3) && ~Line(4)]
Dead_End entry: v=0; LED=1; Dc=1;
[~Line(1) && Line(4)]
Right_Dection entry: v=0.05;
after(.65,sec)
Stop entry: v=0;
[Line(2) && Line(3)]
Right_Juction1 entry: v=0; LED=1; Dc=1;
[~Line(2) && ~Line(3)]
Right_Turn1 entry: v=0; LED=1; Dc=1;
[Line(1) && ~Line(4)]
Left_Dection entry: v=0.03;
after(.65,sec)
Start entry: v=0;
[Line(2) && Line(3)]
Left_Juction entry: v=0; LED=1; Dc=1;
[~Line(2) && ~Line(3)]
Left_Turn entry: v=0; LED=1; Dc=1;
[Line(1) && Line(4)]
Juction_Dection entry: v=0.03;
after(.7,sec)
Start1 entry: v=0;
[~Line(2) && ~Line(3)]
T_Juction entry: v=0; LED=1; Dc=1;
[~Line(1) && Line(2) && Line(3) && ~Line(4)]
Cross_Juction entry: v=0; LED=1; Dc=1;
[Line(1) && Line(2) && Line(3) && Line (4)]
End_Maze entry: v=0; LED=1; Dc=1;

Figure 3: Line Configuration algorithm

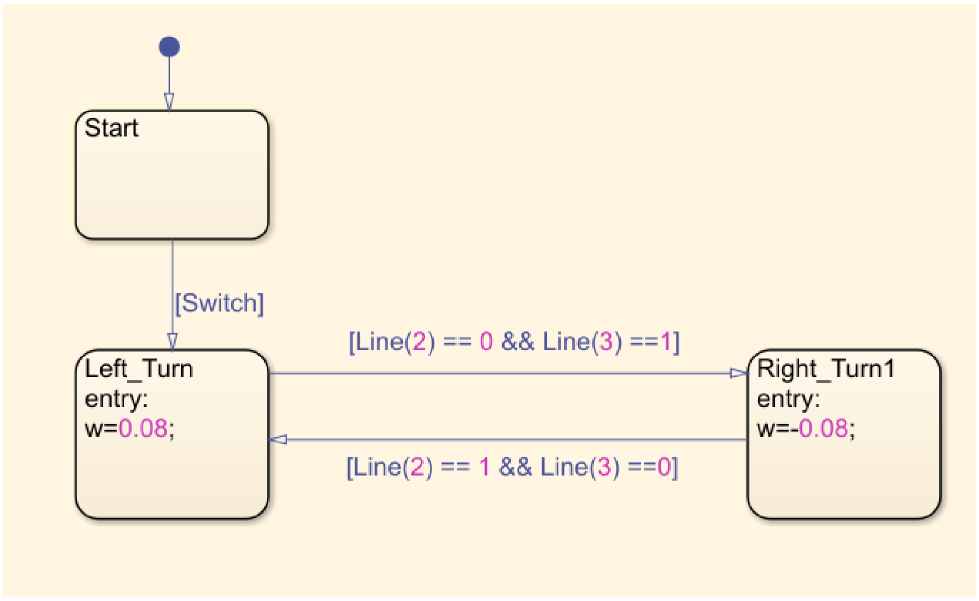


Figure 4: Line Following algorithm for angular velocities

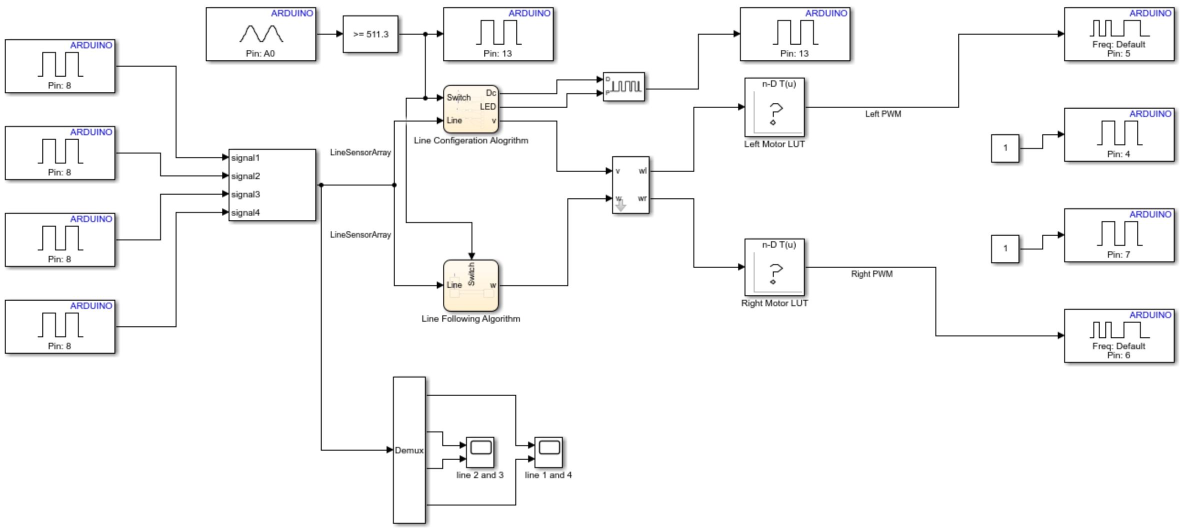


Figure 5: Hardware simulation diagram

Bibliography

[1]N. Chowdhury, D. Khushi and M. Rashid, "Algorithm for Line Follower Robots to Follow Critical Paths with Minimum Number of Sensors", 2021. [Online]. Available: https://core.ac.uk/download/pdf/229655734.pdf. [Accessed: 15- Sep- 2021].