University of Cape Town

EEE3099S

Mechatronics Design

Milestone 2

Ryan Jones
JNSRYA006

Khavish Govind
GVNKHA001

Sean Poole
PLXSEA003

01/09/22

1 Distance Control

The Distance Control Algorithm is as follows:

1. Apply a velocity to the robot.
2. Convert encoder ticks into distance using the shown equation in Figure 1.0.1
3. Average the distance from each wheel
4. Once the distance inputted is equal to 1m or xm, using the s constant, set the velocity to 0 m/s.

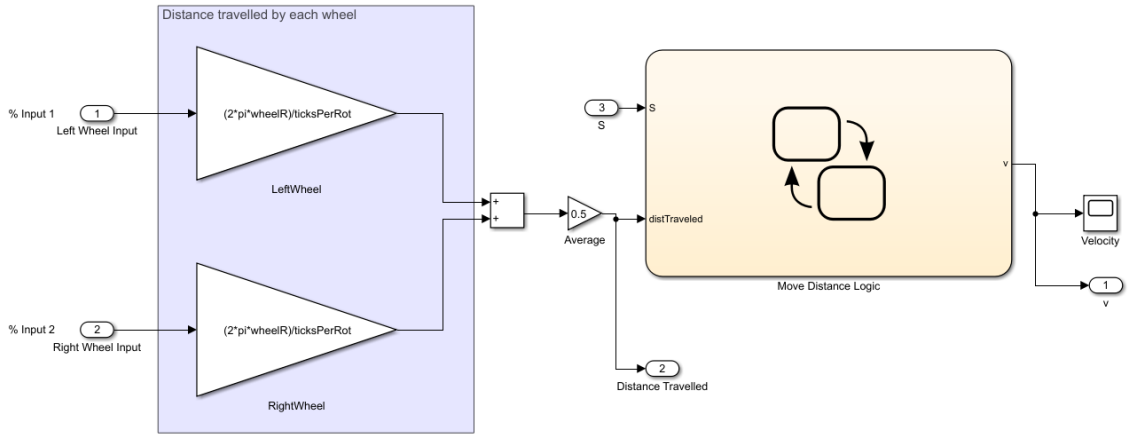

Figure 1.0.1: Distance Control with inputs

Figure 1.0.1 above, shows the Functional Blocks used within the whole distance control subsystem. The left blocks contain the equations used for rotation of the left and right wheels, the output which goes into the State Flow block. The State Flow block contains the logic used for distance control of the robot and outputs a velocity for the robot.

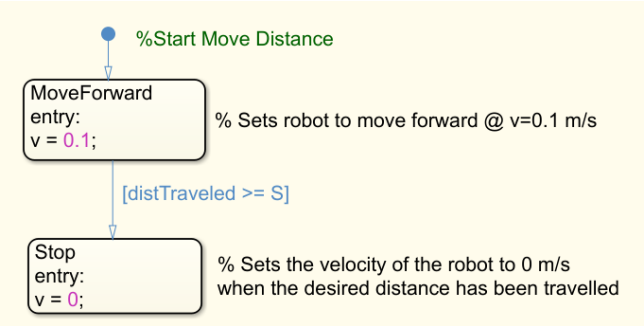

Figure 1.0.2: Distance control logic

Figure 1.0.2 above, shows the two state blocks used for moving forward logic and distance control within the Move Distance State block. It shows the velocities outputted and the conditions for the stop state block to be activated.

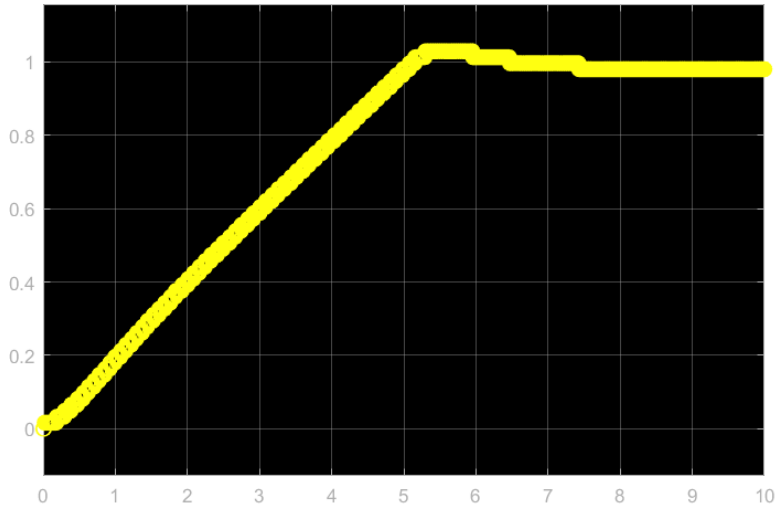

Figure 1.0.3: Scope Output of Distance vs. Time

Figure 1.0.3 above, shows the distance vs time graph of the simulation of the robots distance control logic. The graph indicated slight overshoot of less than 5% and has perfect control following the stop command.

2 Rotational Control

The Rotational Control Algorithm is as follows:

1. Apply a rotational velocity to the robot
2. Convert the ticks from the encoder into a distance for each wheel.
3. Add the two distances noting that one is in the opposite direction.
4. Divide this by the length of the axle
5. Convert the answer from radians to degrees.
6. Subtract the degrees from an inputted heading change.
7. Put that output through a proportional controller to ensure that the heading is as close as possible to the inputted heading.

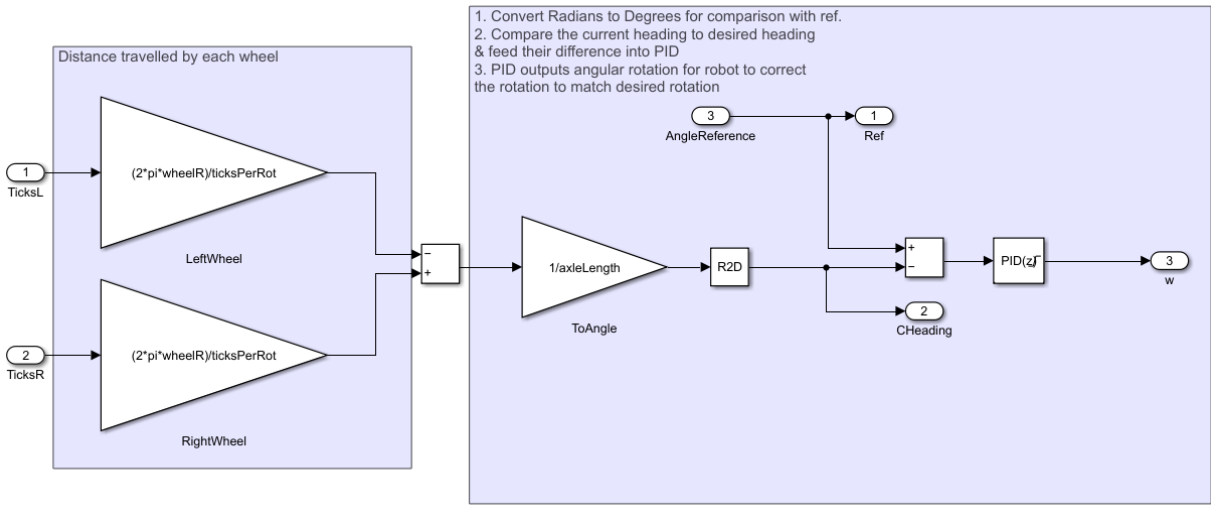

Figure 2.0.1: Rotational Control with inputs

Figure 2.0.1 above, shows the functional Blocks used on Simulink to initialize and do the rotational logic. Illustrating the logic for the left and right wheel and into the logic blocks containing the angle reference outputting to a PID controller to rotational velocity.

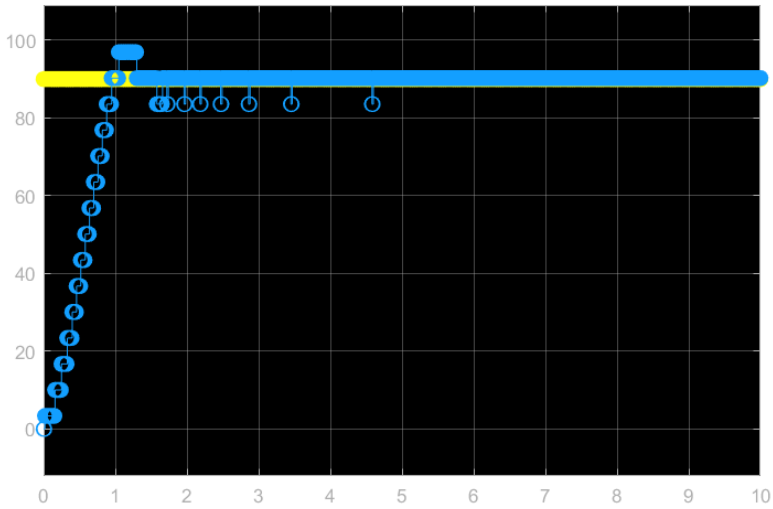

Figure 2.0.2: Scope Output of Rotation *(Degrees)* vs. Time

Figure 2.0.2 above, shows the graph showing simulation of the Rotation of the robot vs time. This is for a rotation of 90 degrees, there is slight overshoot of less than 5% though, however, the robot corrects and settles to just over 90 degrees ($\sim$*90.26 degrees*), which is within the specified 10% error following the overshoot.

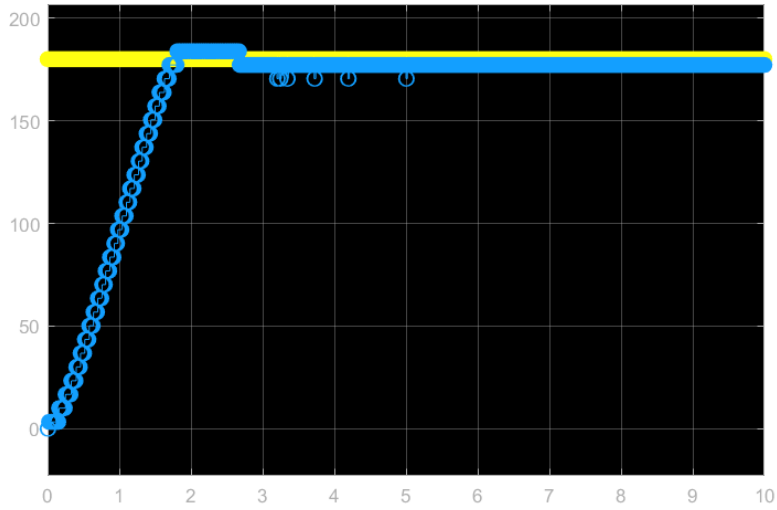

Figure 2.0.3: Scope Output of Rotation *(Degrees)* vs. Time

Figure 2.0.3 above, shows the graph showing simulation of the Rotation of the robot vs time. This is for a rotation of 180 degrees, there is slight overshoot of less than 5% though, however, the robot corrects and settles to just under 180 degrees (∼*177.17 degrees*), which is within the specified 10% error following the overshoot.

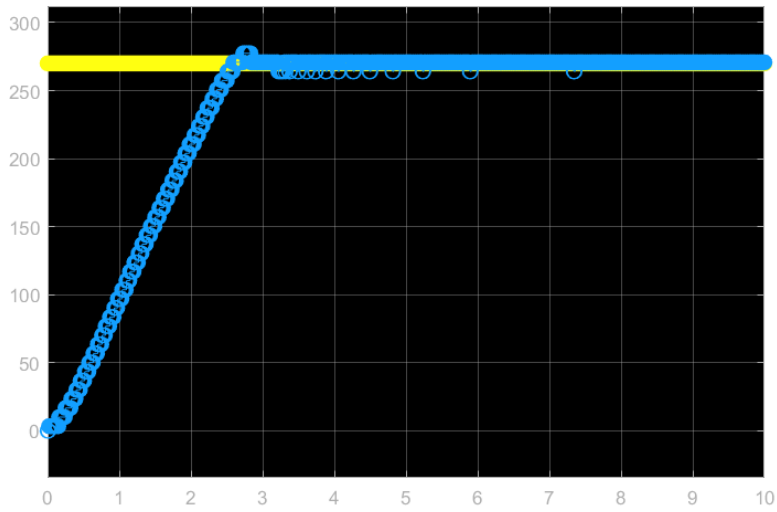

Figure 2.0.4: Scope Output of Rotation *(Degrees)* vs. Time

Figure 2.0.4 above, shows the graph showing simulation of the Rotation of the robot vs time. This is for a rotation of 270 degrees, there is slight overshoot of less than 5% though, however, the robot corrects and settles to just over 270 degrees (∼*270.75 degrees*), which is within the specified 10% error following the overshoot.

3 Line Following

The Line Following algorithm is as follows:

1. State Block takes in a Line Array (A set of values indicating whether the sensor is detecting a line set at a value of 20 and no line (environment value) to a value of 1)
2. Line Array enters the Follow Line Logic State Block
3. Line Array enters the Straight State block
4. If line is detected (value of 20) velocity is set to 0.035m/s and rotational velocity is set to 0 rad/s
5. If Sensor 2 detects no line (value of 1),the Bear Right State block is activated, setting rotational velocity to 0.2 rad/s
6. When the line if fully detected by both sensor 3 and 2, the Straight State block is activated, velocity is set to 0.035m/s and rotational velocity is set to 0 rad/s
7. If Sensor 3 detects no line (value of 1),the Bear Left State block is activated, setting rotational velocity to 0.2 rad/s
8. When the line if fully detected by both sensor 3 and 2, the Straight State block is activated, velocity is set to 0.035m/s and rotational velocity is set to 0 rad/s
9. The velocity and rotational velocity are outputted from the Line Logic state block every state change

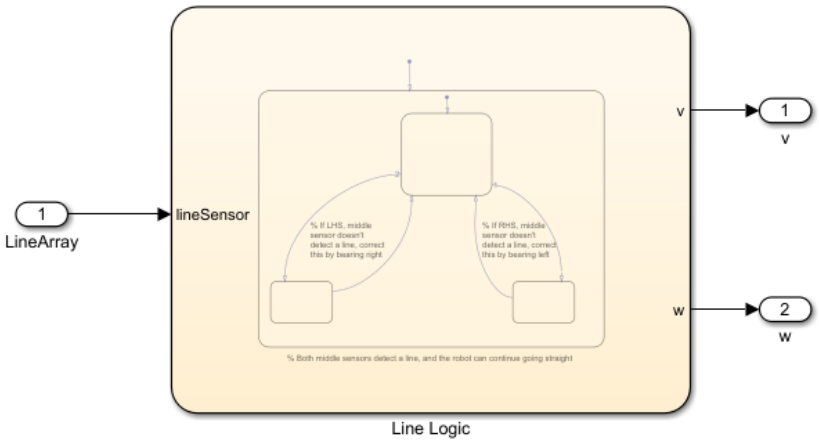

Figure 3.0.1: Line Following Control with inputs

Figure 3.0.1 above, shows the Line Logic state block containing the 3 state logic blocks for the line following. This state block shows the Line Array taken in and the Velocity and Rotational Velocity outputted by the Line Logic to be taken in by the robot simulator.

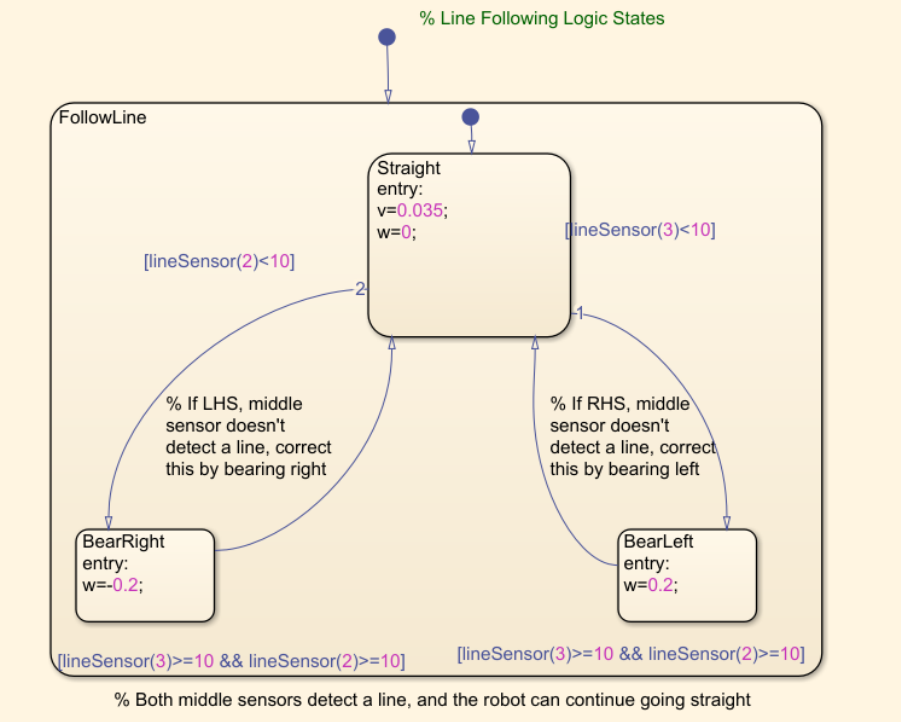

Figure 3.0.2: Line Following Logic

Figure 3.0.2 above, shows the 3 separate states - **Straight, Bear Right, Bear Left** - for which the robot will be in depending on the line array, also shown is there conditions required to move between these 3 states

All MATLAB Files are available on GitHub here