LAB 6 – FINAL REPORT

ME 375 – CONTROLS II

Gabriella Giachini

May 1 2022

Introduction

This report summarizes the process to build and program a fully automated two-wheeled robot that accomplishes “line following” and “herding” behaviors by using infrared (IR) and ultrasonic sensors. The robot is assembled with a Skitter robot kit and controlled by a myRIO device programmed by a LabVIEW algorithm.

The first task was to construct the robot and perform several tests and controller design calculations to obtain its custom closed-loop controller. Then, the behavior of the sensors was studied to later apply its functionality properly. Finally, a state machine was developed and coded with different cases for the robot to be able to initially use the IR sensors to follow the line and at the end, switch to herding where the ultrasonic sensors will be used staying at a set distance away from an object.

State Machine Design

A state machine was designed to best develop the logic for the robot to interact with the given environment. Five different states were developed: start, go forward, too left, too right, stop/change, switch to herding that will be triggered by the binary output of IR sensors. The state machine is shown in the table and graph below. To aid in sharp left or right turns, case 4, which was originally simply for stopping, is used to recall and go to the previous step.

Table 1: State machine

State	Inputs								Output
	000	001	010	100	011	110	101	111	
0	0	X	1	X	1	1	X	1	Start
1	0	2	1	3	1	1	5	5	Forward
2	4	2	1	3	1	1	X	1	Turn right
3	4	2	1	3	1	1	X	1	Turn left
4	*	2	1	3	1	1	X	1	Stop/ Change
5	5	5	5	5	5	5	5	5	Herding

*go to previous case

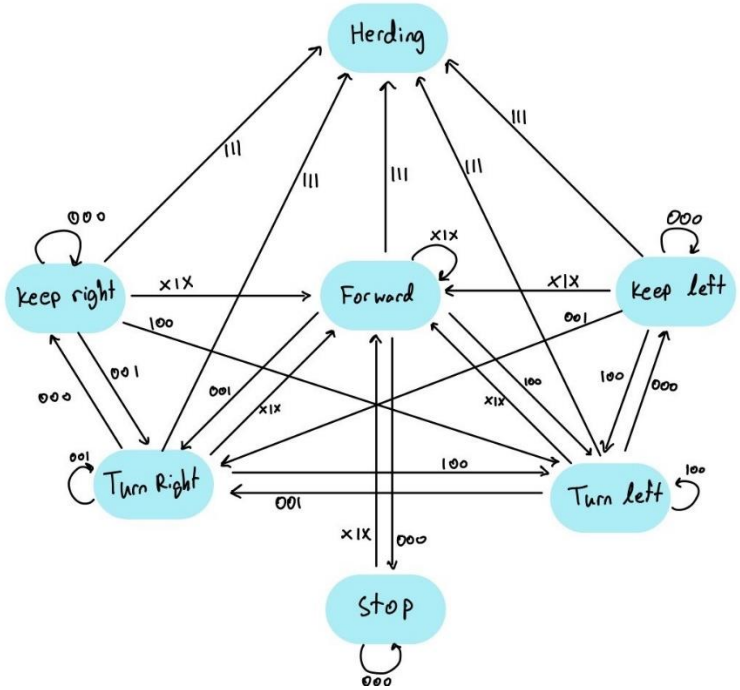

Figure 1: State Machine Transition Diagram

Controller Design

A closed-loop custom, proportional integral, controller was designed to control the speed of the robot wheels by using the pole placement method. The goal was to have a 2% settling time of 0.1s, and overshoot less than 10% and a steady state error of zero. This process yielded the following controller and was applied as shown in figure 2:

$$C(s) = \frac{1.22s + 86.132}{s}$$

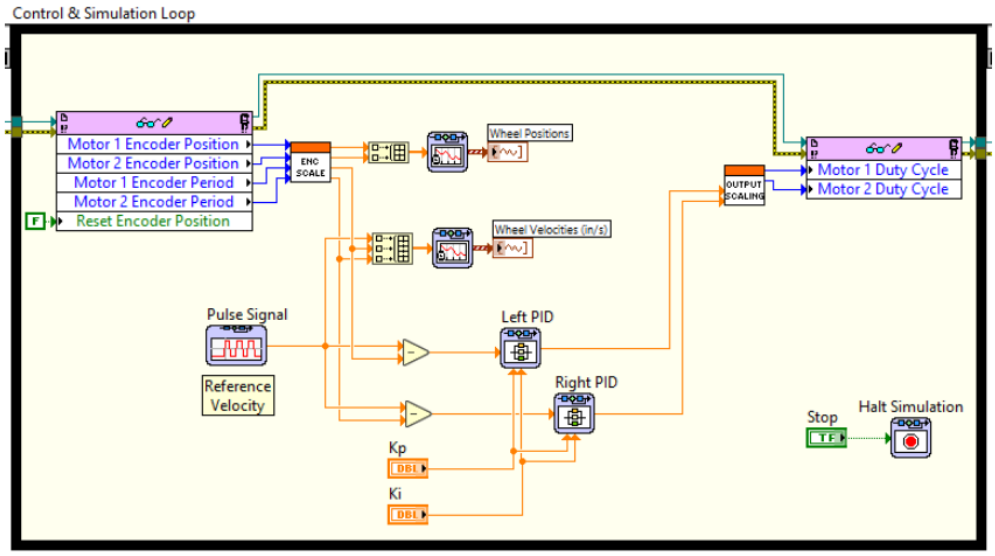

Figure 2: Proportional Integral Controller

By performing several tests, the gain parameters were slightly changed to account for the already existing imprecision of the robot (e.g.: motors, and thus wheels, did not turn at the same time). Moreover, friction compensation was added to the controller to maintain good speed control as it removes the dead-band error. The friction factor also needed to be tuned as too much friction would increase the overshoot (and make the robot noisier, more frantic). Therefore the controller used is the following and was applied as shown in figure 3:

$$C(s) = \frac{1.225s + 104.48}{s} \; with \; friction \; compensation \; of \; 2V$$

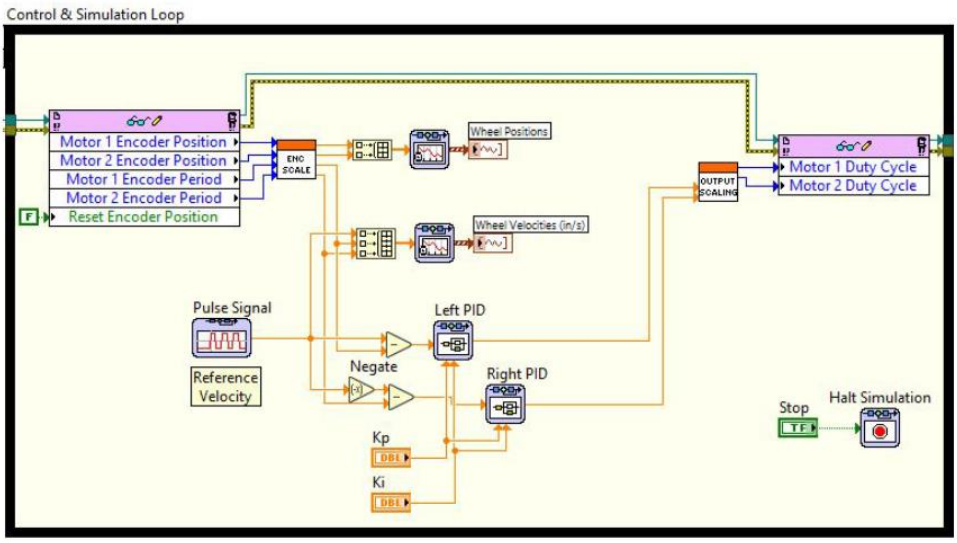

Figure 3: Proportional Integral Controller + Friction Compensation

To perform the line following and herding, the robot uses two different sensors: infrared and ultrasonic. A FPGA encoder, PWM sensor VI and FPGA sensors bit-file was used to convert the raw voltage data of the sensors to usable data in LabVIEW. Out of the five IR sensors on the robot, only the three middle ones were used to aid with line following as they output binary true/false depending on whether the robot sees white/black.

On the other hand, the ultrasonic sensor measures the time of flight, this is the time that it takes for the ultrasonic pulses sent by the sensor to hit an object (in front of the robot) and come back. This sensor is then used for the herding problem as the time of flight is transformed into a distance which can be used to be at 9inches away from a moving object.

Performance

The robot's line following, and herding functioning was tested in the track shown in figure 4. Initially it goes through an almost straight path, then makes a sharp right turn and finally crosses a white line which shifts the robot into herding mode. In this mode, a moving object is placed in front of the robot, and it is supposed to move back and forth depending on how far away it is from the moving object.

Figure 4: Track used for testing the robot

The following experimental data shows the performance of the controller while executing line following (see Figure 5). One can see that initially the robot has almost a constant velocity, except for a few peaks at 0 in/s where the robot is adjusting its position to be on the line. In other words, as the robot gets off track, depending on which side, one of the wheels velocity is set to zero and the other at a set velocity. This way, it can slightly turn and get on track again. This process is amplified as the robot takes a sharp right turn and it can be seen in the graph (between 9 and 11s) as the right wheel remains at 0 in/s for a longer time, while the left wheel remains at 10 in/s to make the right turn. Finally, at the end of the plot both wheels go to 0 in/s as they reach the horizontal white line (which triggers herding mode).

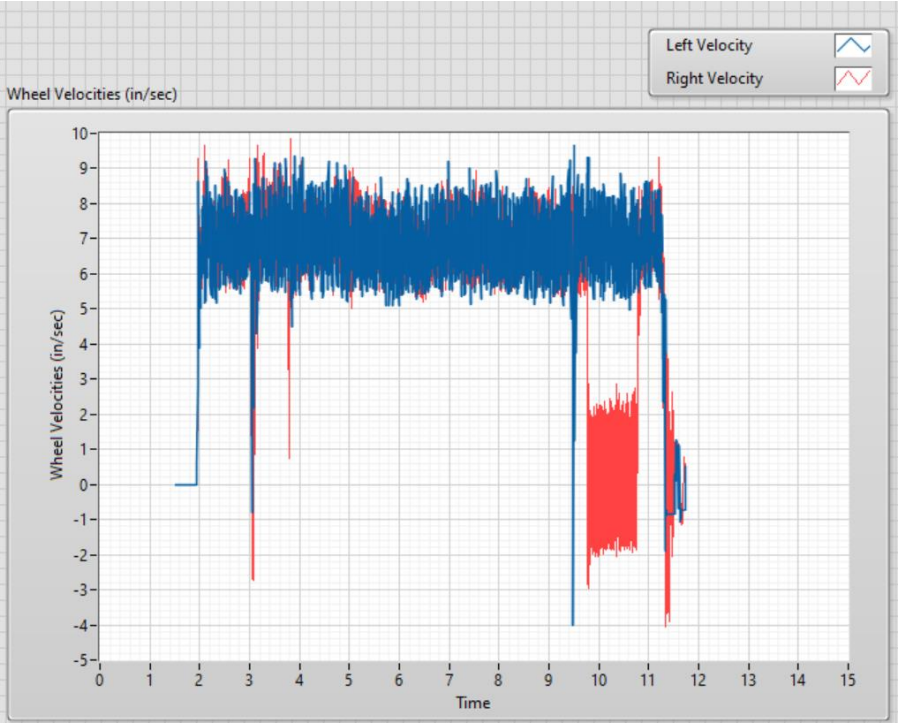

Figure 5: Line following velocity plot

The next experimental data shows the performance of the controller while executing herding (see Figure 6). One can see how the robot moves back and forth, trying to maintain a distance of 9 inches, as the object is moved closer or farther away from it. The plot is not constant as the code was set up in a way such that the distance determines how fast the robot goes. For example, as the robot gets closer to being 9 inches away from the object, it goes slower and slower. On the other hand, when it is very far away it is much faster.

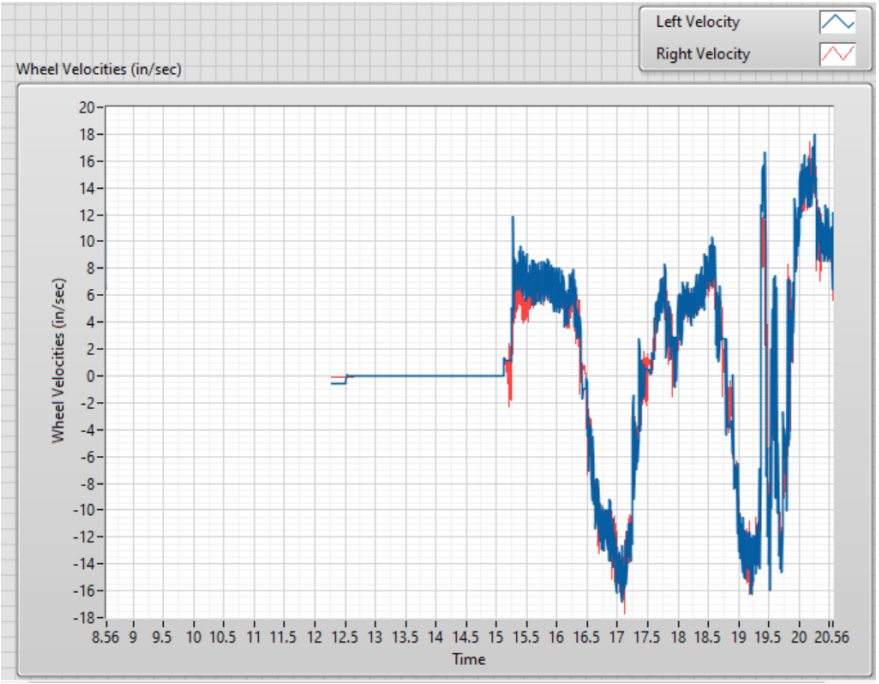

Figure 6: Herding velocity plot

Code Description

The following figure shows the complete code used:

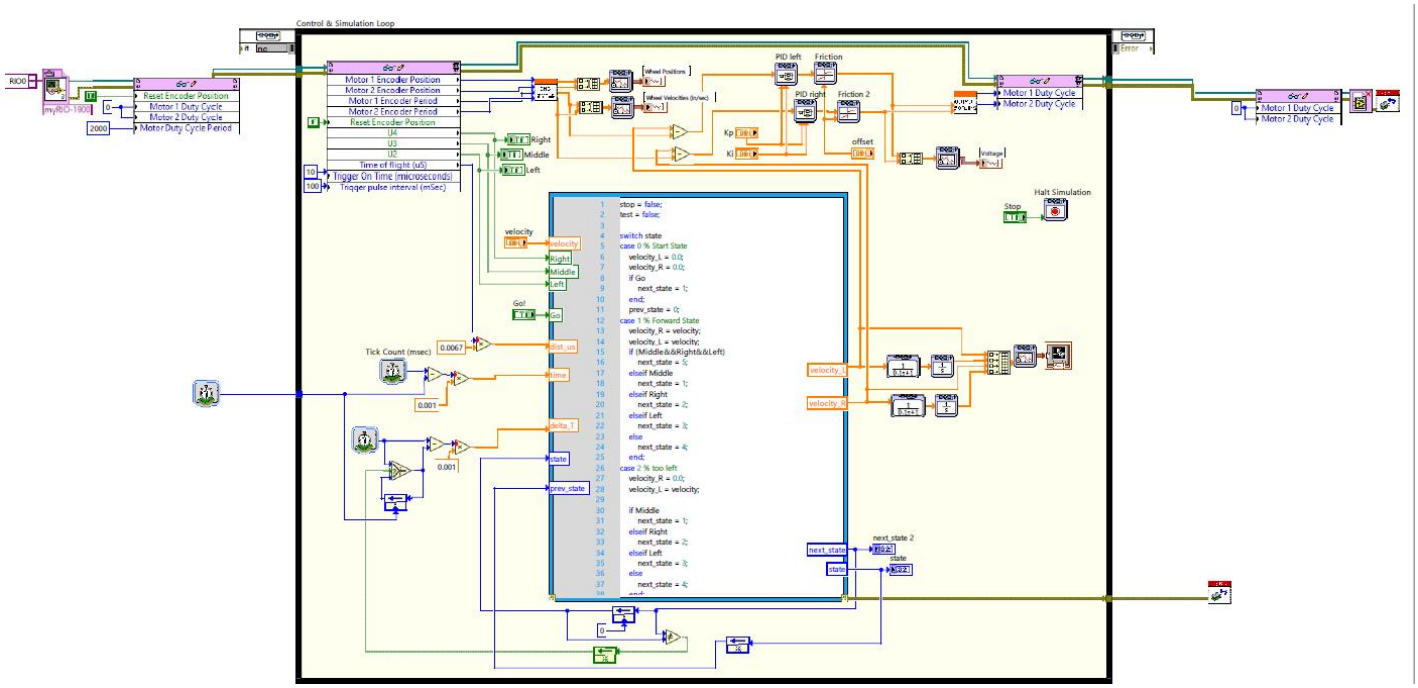

Figure 7: Overall block diagram VI

To aid in the description of the code, it will be separated into different sections. The first is the upper right corner where one can see the closed loop controller for each motor along with the friction compensation (see Figure 8). The controller's input comes from the Encoder Scaling sub-vi, which is fed the motor's encoder position, and the controller's output goes into the Output scaling VI. The second section is the upper left corner where the data from the sensors is collected and transformed into usable data types, e.g.: IR sensors data to true/false indicators and ultrasonic sensor time of flight data to distance values (see Figure 9). And this is fed into the MathScript block, the last section of the code, where the main code to follow the line, switch to herding, perform herding and maintain distance is written. This block accepts the sensors data and outputs

the velocity of each wheel into their respective controller. Moreover, feedback nodes are used to keep track of the next state and previous state values for the next iteration. See figure 10 for the overall structure and see the compressed file for the full MathScript code. Overall this code consisted of 5 different cases (states) based on the finite state machine built beforehand (see figure 1), where state 0 starts the robot, state 1 tells the robot to go forward (both wheels move at the same velocity), state 2 tells the robot to go right (right wheel has no velocity), state 3 tells the robot to go left (left wheel has to velocity), state 4 is used to either stop or recall previous state to aid in making sharp turns and finally state 5 is turns on herding mode. Inside each state the logic developed in the finite state is added to force the robot to change states autonomously.

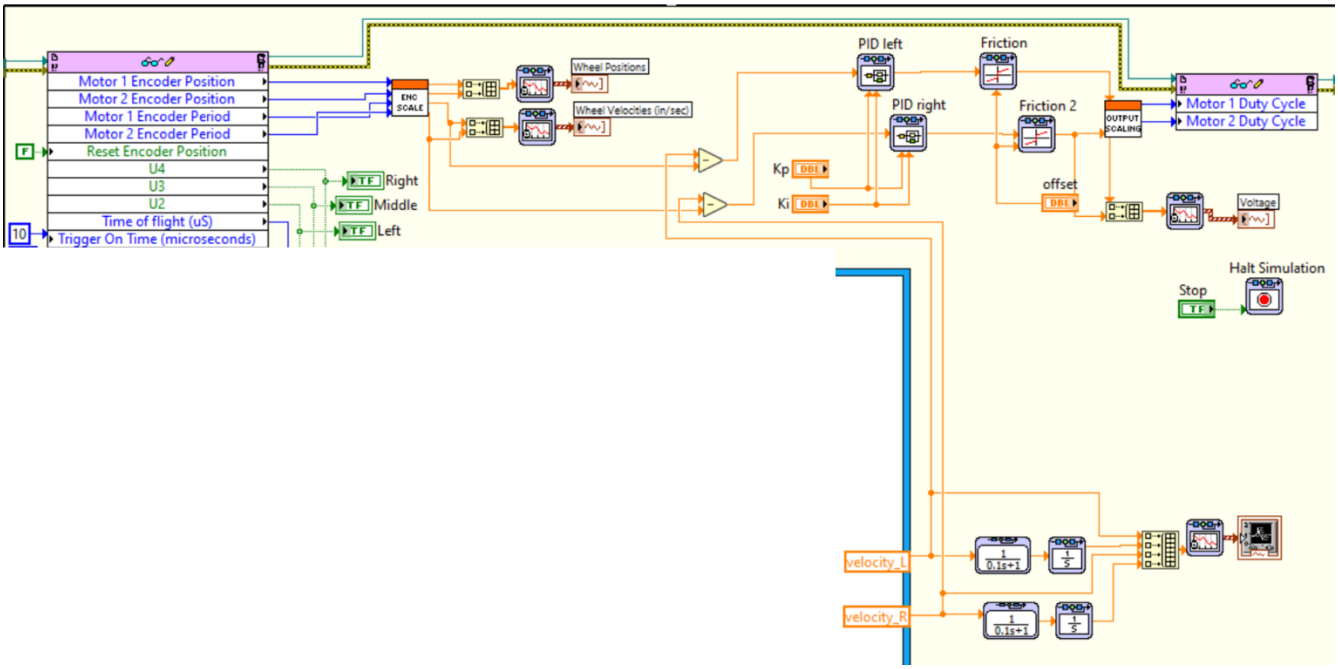

Figure 8: Controller

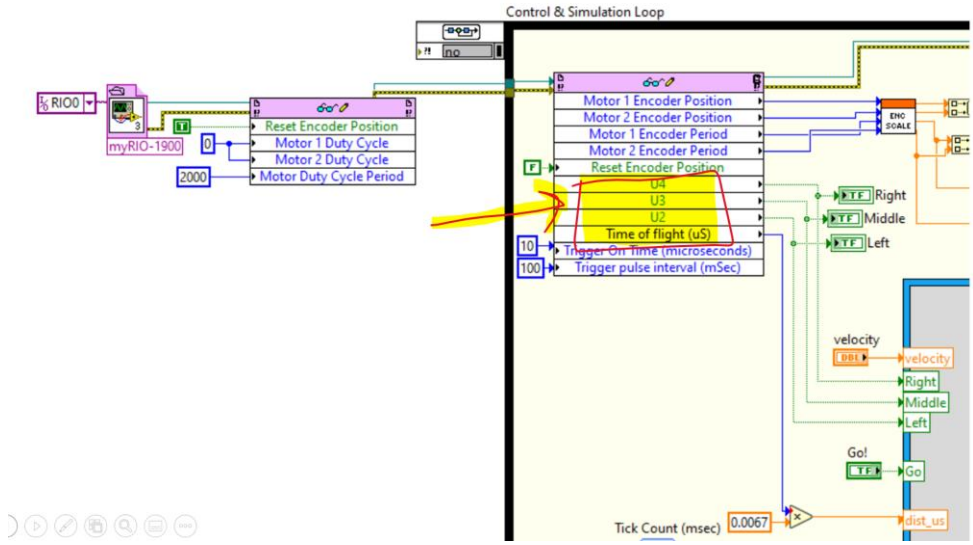

Figure 9: Upper left corner with sensors output

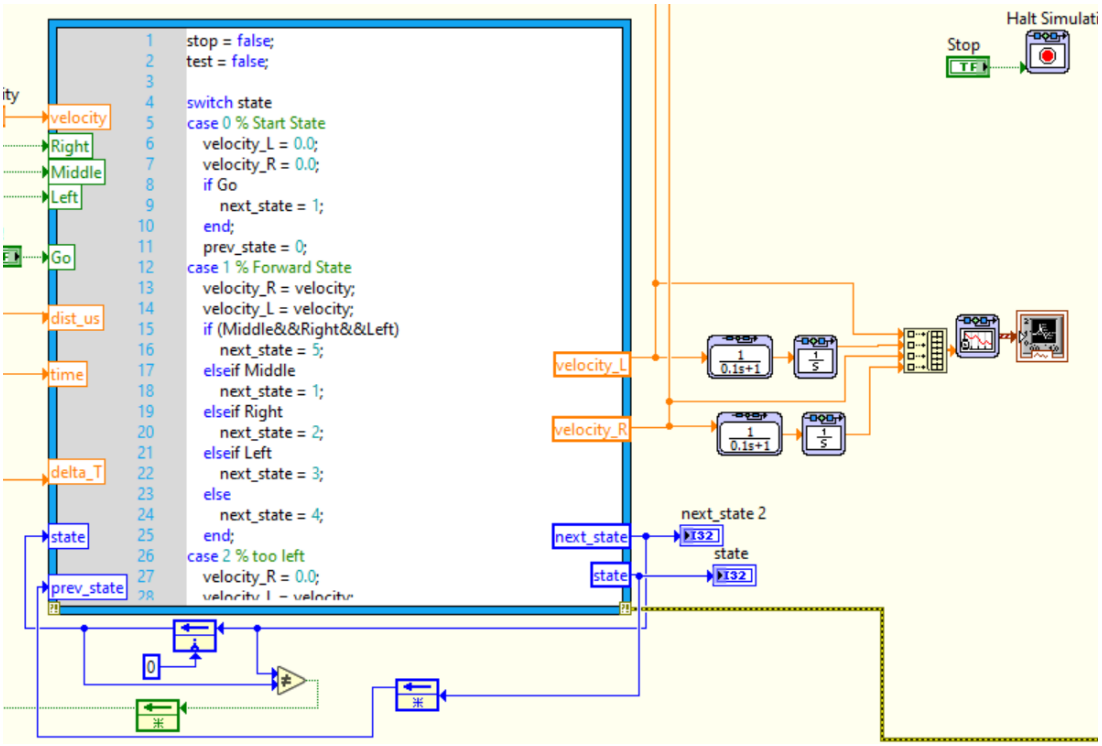

Figure 10: MathScript block

The sub-VI used in the main VI helped to apply the controller to the code. One of the sub-VIs is the encoder scaling VI, which is used to scale and filter the encoder information (see figure 11). The other sub-VI is the output scaling VI, which scales the motor voltage to a 40MHz clock ticks and is then sent to the FPGA bit-file.

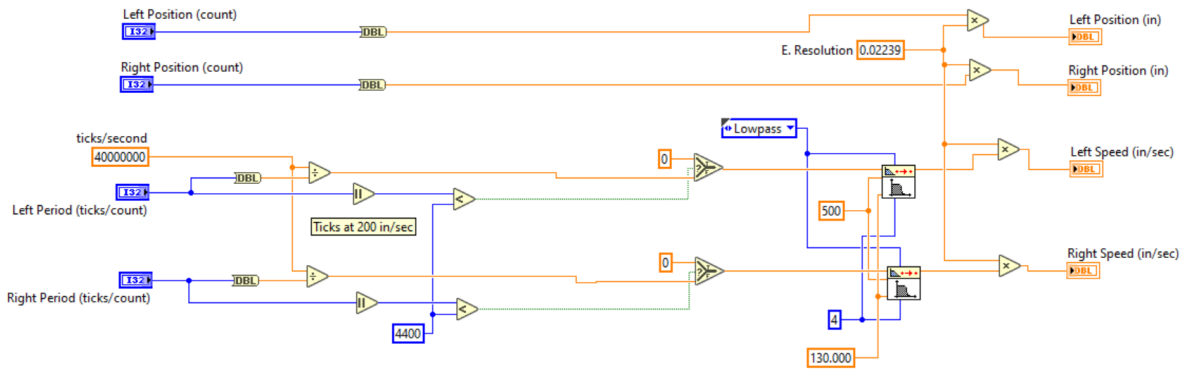

Figure 11: Encoder Scaling VI

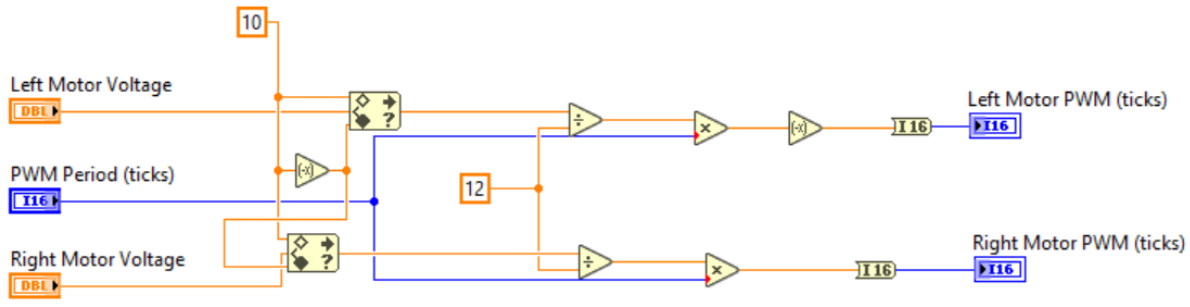

Figure 12: Output Scaling VI

Discussion

Overall the robot performed very well, and it hit all the criteria. The line following was precise and robust enough where the robot could find the track again once it got lost. However, it was not very smooth. This could be improved by changing the velocity of each wheel when adjusting to follow the line (this is not the case when following the line for sharp turns). The current code makes one of the wheels go at 0 in/s, while the other one is at the specified velocity. To make the transition smoother, one could decrease the velocity of the first wheel instead of fully stopping it.

On the other hand, the herding code was very robust as well as it sped up when too far and slowed down when too close. However, the robot shook a lot and was a little unstable while performing herding and this should be improved.