Autonomous Lane-Assist Emulation Robot

Team Members: Eli Buckner, Catherine Ambrose, Alper Ender, Rutuja Chaturbhuj, Samuel Eddy, Yung Han Chen, Ben Wagner
Department of Electrical and Computer Engineering
North Carolina State University - NCSU
Raleigh, North Carolina

***Abstract*—this report documents a project to improve an existing autonomous line following robot in several ways. The team was expected to improve the speed, precision, and consistency of the robot and to add "lane assist" functionality to the robot. These improvements were made by changing the robot's microcontroller board to an Arduino Uno R3 Microcontroller A000066, adding a SainSmart L293D Motor Drive Shield, and adding two QTR-8A Reflectance Sensor Arrays. Additionally, the team improved the robot by testing multiple control algorithms and choosing the one with the overall best results. The team built a track to best resemble a road in order to test the robot and compare the different controllers. After testing, the team found that each controller, along with the hardware updates, improved the existing line following robot. However, a phase lead controller that was designed in the MATLAB app called "SISOTOOL" proved to give the best control out of the controllers that were designed for the system.**

I. INTRODUCTION

A. *Background*

There are many new safety systems being developed for cars, especially with the rising excitement regarding self-driving cars. Two of these safety features are lane departure warning and lane keeping assist systems. Both systems involve detecting when the car goes over a line unintentionally and then either alerting the driver through a noise or light signal (lane departure warning) or by correcting this action for the driver (lane keeping assist).

Lane keeping assist has the option to use either intervention or control, where intervention simply augments driver commands whereas control eliminates the need for driver input and essentially creates a driverless car. If lane keeping assist functions as a control system, it has to take into consideration the current lane position, pedal activity, and whether or not the driver is touching the steering wheel. It takes in information about the current lane position using CCD cameras and converts this into offset torque through Electric Power Assisted Steering (EPAS) [6]. It can even use the images from the cameras to determine whether there is a solid or dashed line on either side of the car [7]. The offset torque generated through EPAS can either provide torque to the steering wheel or a differential wheel angle [6]. If the driver were to steer in a way that opposed the EPAS torque, for example to change lanes, they would be able to with no real extra effort [7].

The inspiration of this project came from the effort to emulate a lane assist technology, but the platform in which the project was started came from a North Carolina State University (NCSU) electrical and computer engineering course (ECE) on embedded systems (ECE 306). This course had taught the basics of embedded systems to students by building a semi-autonomous robot off of an MSP430 family microcontroller. Over the semester, students built a control board for the microcontroller, attached an internet-of-things module to the board, and set up a detector and emitter circuit to be able to follow a black line. The purpose of this course was to introduce students on how to setup an embedded system to have all the parts working together.

1. **Texas Instruments (TI) MSP430 Microcontroller**
   The microcontroller utilized for the ECE 306 course was a MSP-EXP430FR5739 Experimenter Board (MSP-430). The MSP-430 has 32 general purpose input/output (GPIO) pins with 12 analog to digital converter (ADC) channels. The ADC on the microcontroller has 10 bits of resolution. The MSP-430 is a low power unit with a minimum operating voltage of 2 volts and a maximum operating voltage of 3.6 volts.
2. **Controller Board**
   The controller board for the ECE 306 course was designed by Jim Carlson. It has 2 full H-bridge circuits on the board, a liquid crystal display (LCD), a potentiometer thumbwheel, an infrared (IR) light-emitting diode (LED) and 2 IR detectors, and an internet of things (IOT) module.
3. **Vehicle**

The vehicle for the course was a plexiglass chassis mounted by the MSP-430, the controller board, a six volt battery pack, two direct current (DC) motors, and two lego wheels.

4. **Vehicle Performance**

The vehicle had a rudimentary control system without any implementation of modern digital control methods. The controls were fully algorithmically based: If one led did not read the line, the motor from the opposite side would turn off and vice versa. The vehicle had to have a low duty cycle for its pulse width modulation (PWM) in order to successfully follow the black line, and could not vary its motor speeds. This lead to choppy and slow responses.

5. **Vehicle Track**

The black line demonstration track of the ECE 306 course was a single-lane course that had little resemblance to an actual road. The model of the track is given in Fig. 1.

Fig. 1. The ECE 306 black line demonstration track

B. Purpose

The purpose of this project was to use the techniques learned in the ECE 536 - Digital Control Systems course to improve the response of the car originally used in ECE 306 as described above. While the original car was choppy and slow, the new car shall be smoother and have faster response times. This will be accomplished by testing a variety of control systems learned in ECE 536 and to determine which control method allows for the best response. The new car shall also emulate a real car. The original car followed one line but the new car will follow two and use a "lane assist" algorithm.

C. Goals

The goal of this project is to take the existing vehicle and create a "Lane assist" algorithm to emulate how a car might perceive a lane and correct itself according to where it thinks it is in the lane. This algorithm will be similar but much simpler to the algorithms that are used in cars with lane assist technology today. The aim is to use the methods learned in class in order to develop sophisticated control algorithms that meet certain design parameters that have been self-set, and compare the results of these algorithms to decide which one gives the best control. The Design Parameters that have been set for this device are as follows:

Design Parameters

- Rise Time< 0.25 s
- Steady State (Step)< 0.2 cm
- Settling Time < 1 s
- Gain Margin > 30
- Phase Margin > 60
- Percent overshoot < 10%

To achieve these design parameters we have chosen to implement our control via the following control algorithms that were learned in class:

Controllers

1. PID
2. Phase Lead
3. Prediction Observer
4. IH-LQR
5. Current Observer
6. Root Locus

In order to emulate what cars on the road might actually see, it is necessary to stimulate the device with all types of "lanes" that a car might see. To do this, the device will be coded to deal with the following "lane" cases:

1. One Solid Line and One Dashed Line
2. Double Solid
3. Double Dashed
4. One line - Dashed or Solid
5. Intersection

Using these cases the vehicle shall determine which case it is currently seeing and follow/assist appropriately.

D. Method

1. **Total System Diagram**

The system has the following important components:

I. Sensors: 2 QTR light sensor arrays were used to obtain real-time inputs about the position of the robot with respect to the 2 lines (e.g. a lane). This relative position of the robot allows computation of the error, which is used by the control algorithms to align the robot within the lanes.

II. Micro-Controller Board: Inputs from the sensors are accepted by the controller board and the parameter 'motor-differential' is calculated. The speeds of the two motors are adjusted on the basis of the value of the motor differential.

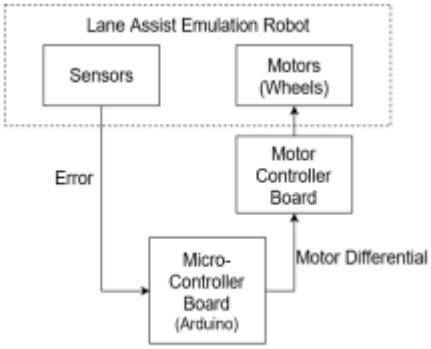

Fig. 2. Overall System Diagram

III. Motor Controller Board: The motor controller board was used to interface the motor with the system. The motor speeds are set according to the value of the variable motor-differential.

IV. Motors (Wheels): The motors and wheels assembly, using the 'motor differential', allows the robot to navigate through the lanes precisely.

2. Sensor Feedback

A variable 'error' was defined and it stated how far off is the robot from where it is supposed be. For the robot to follow the lane, the robot must exactly be in the center of the lanes. Hence, the reference point was fixed to be the center of the lanes. And the error was defined as the difference between the actual position of the robot and the center of the lanes. The error was a measure, of how much corrective action is needed to bring back the robot to between the lanes.

The light sensors are an array of sensors, and each of the sensors in the array is weighted according to its distance from the center of the lanes. Thus, the light intensity detected by the sensor arrays, was proportional to how far the robot was from the center of the lane. Thus, the error is proportional to the distance the robot is away from the reference point.

3. Reference Point.

a. System Identification

The reference point was fixed to be along unit step for the system identification phase.

b. All Other Times

The reference point was fixed to be the center of the two lanes for all conditions other than the system identification.

4. System Identification

A controller had to be designed to control to the robot. In order to design a controller transfer function, the transfer function of the system needed to be identified first. Hence, system identification was performed on the robot initially to identify

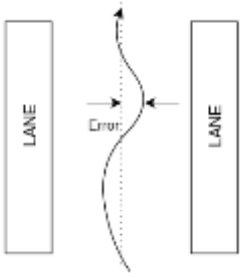

Fig. 3. Error detection using light sensors

the order of the system and obtain a transfer function (Gz). Knowing the transfer function, and its order, of the plant, the design of a precise controller could be done.

After the transfer function was obtained, various controller algorithms were implemented in MATLAB to design controller for the system. The best controller designs were then implemented with the original transfer function on the robot.

5. Controller Design

The controllers mentioned above were designed using MATLAB simulation for a number of set speeds. For every controller transfer function (Dz) obtained , the step response of the overall system transfer function (Tz) was plotted, where,

$$Tz = \frac{Gz \cdot Dz}{1 + (Gz \cdot Dz)} \quad (1)$$

The ability of each of the controllers to emulate the lanes was observed with the robot, and the ones which achieved the best responses were included in the final control algorithm.

II. RESEARCH

A. Sensors

1. Available sensors

1. IR light sensors and IR LED pair:

It is an infrared (IR) LED and infrared sensor in combination. The sensors work on the principle of the

reflectance of light. The sensor is usually a photodiode. The intensity of light reflected depends on the surface from which the light has been reflected. Thus, the intensity of light reflected from a white surface will be different from that reflected from a black surface. This allows the sensor to distinguish the white line from the black background.

Advantages:

· Easy to use.

· Good response time.

Disadvantages:

· Easily affected by the ambient lights. (IR photodiodes usually detect light waves in the range of 700 nm -850 nm. The wavelengths of most of the IR ambient lights also fall in this range; making it difficult to)

· No predefined Arduino library available (Makes integration of the sensors with whole system difficult.)

· Can sense only between 2 different lights.

2. LED and LDR pair:

They work in the similar way as the IR sensor and IR LED. The LED in this case works in the visible range of light wavelengths. The only difference is the use of a LDR(light dependent resistor) instead of a phototransistor.

Advantage:

· Can be used to sense several different lights.

Disadvantages:

· Slow response time as LDR have a very slow response time to changes in the surrounding lights.

· Ambient light interference is too high.

3. QTR Reflectance sensors:

Array of 8 sensors which are IR LEDs and phototransistors pairs that provide analog measurements of IR reflectance.

Advantages:

· Parallel reading of values is possible with microcontroller

· Interfacing with Arduino is easier because, there are Arduino libraries available for these sensors.

2. Decision Matrix

The QTR sensors were selected because it met the following criteria:

1. Interfacing with Arduino was easy, due to the predefined libraries.
2. Reading multiple values from the array simultaneously is possible
3. Very fast response time.

3. Sensor Input into the System

Two QTR-8A infrared sensors were selected for the vehicle. Each sensor array consisted of eight sensors that read analog values from 0 to 1023 depending on the values sensed from each sensor. These sensors were utilized as the feedback into the system and were utilized to compute the position and error of the system.

4. Mounting the Sensors

The sensors were mounted using wood cut from the NCSU ECE makerspace. Using size M2 screws, the sensors were fastened to the wood, which was in turn mounted to the chassis of the vehicle. The sensors stand less than 1 centimeter off the ground which allows for proper readings from each sensor. The sensors should ideally be completely perpendicular to the vehicle; however, variations may occur and are factored into the software. Fig. 4 shows the final mounting location of the sensors.

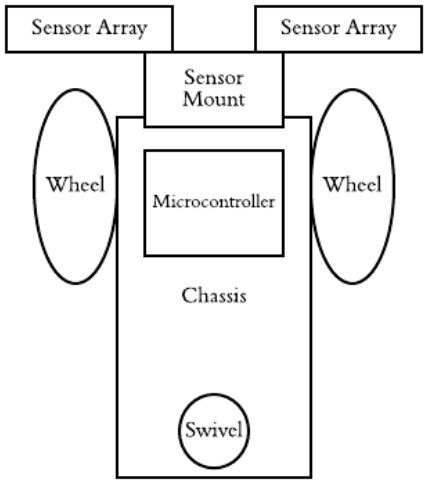

Fig. 4. The overall diagram of the vehicle for the sensor mounts.

5. Additional Sensor Hardware

The Arduino Uno has 6 analog input/output (IO) pins. In order to record every value of the sensor arrays, 8 IO pins are necessary for each array, totaling to 16 IO pins in order to obtain the values from each sensor. To resolve this issue, the team decided to utilize two 8 to 1 multiplexor chips. This logic device has a maximum delay of four microseconds, and so the microcontroller has enough time to read in every sensor value all in one interrupt service routine. A prototyping board was

utilized to connect the two sensors and the microcontroller to the multiplexers.

6. Error Sensing Algorithm

The sensors read in discrete values that need to be transferred into a value for the vehicle to identify its location on between the lines. An example of sensor reading is shown in Tab. 1**.**

A weighted average algorithm was employed to obtain the location of the lines along the sensor arrays and convert the discrete values into a usable value. The weighted average algorithm is shown below where S_i is the sensor value from each sensor.

Sensor Number	Left Sensor Array	Right Sensor Array
0	0.00	0.00
1	0.00	0.00
2	0.00	0.00
3	1008.00	1009.00
4	1003.00	0.00
5	0.00	0.00
6	0.00	0.00
7	0.00	0.00

Tab. 1. An example of readings from the left and right sensor

$$Weighted\ Sum = \sum_{i=0}^{8} s_i * i$$

$$Total = \sum_{i=0}^{8} s_i$$

$$Line\ Val = \frac{Weighted\ Sum}{Total} \qquad (2,3,4)$$

The line value (Line Val) is a value between 0 and 7 that is the location of the center of the black line as read by the sensors. Since each of the eight sensors on the array is located 1 centimeter apart, we can reasonably justify that a Line Val of 3.5 corresponds to 3.5 centimeters from the side of the sensor array. These units were utilized to determine the position of the vehicle and its error from the center. Through calibration at the beginning of the program, the sensors calibrate their location as the starting, calibrated Line Val location (Initial Line Val). This is the zero-error location. In order to get the sensed value (Sensed Val) from the sensors, Eq. 5 was used.

$$Sensed\ Val = Line\ Val - Init\ Line\ Val \qquad (5)$$

The Sensed Val from both sensors were averaged together to obtain the position (pos) of the vehicle.

$$pos = \frac{Left\ Sensor\ Sensed\ Val + Right\ Sensor\ Sensed\ Val}{2} \qquad (6)$$

This position is the

$$error = pos - ref \qquad (7)$$

The position value is the current line value calculated from the sensors, and the reference (ref) value is the point to which the error value is calculated upon. On most cases, the reference value is zero in order to calculate the error based upon a zero-error location. Only during the system identification phase was the reference value a non-zero value. This non-zero reference value brings in a step input to which the vehicle responds.

The following example will demonstrate the application of determining the error of the vehicle. Taking Tab. 1 as the calibrated value of the car, the values in Tab. 2 were sampled some time after calibration.

Sensor Number	Left Sensor Array	Right Sensor Array
0	65.00	77.00
1	58.00	75.00
2	56.00	72.00
3	49.00	70.00
4	51.00	1023.00
5	1002.00	1023.00
6	927.00	70.00
7	45.00	75.00

Tab. 2. An example of readings from the left and right sensor some time after calibration

Characteristics	Table X1	Table X2
Left Sensor Line Val	3.45	5.06
Right Sensor Line Val	3.00	4.26
Position	0.00	1.25
Reference	0.00	0.00
Error	0.00	1.25

Tab. 3. An example of calculations from the readings

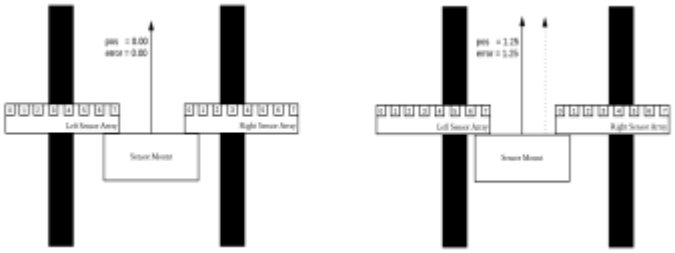

Fig. 5. The visualization of the tabulated data and calculated values. The left figure displays the vehicle during calibration, and the right figure displays the vehicle some time after calibration

Tab. 3 calculates the Line Val, position, and error values from the two sets of tabulated data, provided that the reference value is zero.

Fig. 5 displays the visualization of these two tables on how the vehicle calculated position and error from sensor inputs.

B. MCU

The microcontroller used for this car is the Arduino Uno. The Arduino Uno is a fairly simple 16 Hz microcontroller board with 14 digital input/output pins, 6 analog inputs, a USB port, a power jack, and an ICSP header [3].

1. Decision Criteria

a. The Arduino Uno allows for 6 of the digital pins to be used as Pulse Width Modulation inputs which are necessary to run the motors being used.
b. It is 68.6mm x 53.4 mm which is small enough to fit on the existing chassis.
c. The Arduino Uno allows for a sampling time of as little as 3 milliseconds.
d. It has a USB port which allows for easy connection to a computer.
e. The Arduino IDE is easy to use and open-source.
f. It includes a reset button for easy debugging.
g. The Arduino Uno can be powered off of an external power supply up to 12V which can be easily achieved with batteries.

2. Processor Speed vs. Desired Time Constant

In order to maintain a sampling time of 10 milliseconds, an interrupt system was implemented. An interrupt is sent every 10 milliseconds that reads the sensors and then finds the position based on these readings.

Pins A5, A4, and A3 are used as the select bits for the 8 to 1 multiplexor that leads to the sensors. Pin A0 is the input from the left sensor and pin A1 is the input from the right sensor (both through the multiplexor). 12V is input to Vin. The 5V output is used from the Arduino Uno to the sensors.

Fig. 6. Arduino Uno Pinout

C. Motor Controller

Background on L293D Motor Shield

L293D device is a quadruple half-H driver which provides bidirectional drive current and it is designed to drive inductive loads. It can drive 4 DC motors or 2 stepper motors with individual 8-bit speed selection.

Decision reasons

The L293D was chosen as the motor controller board. There were several reasons for that.

1) It could provide wide supply-voltage range from 4.5 to 36volts, so as we have different voltage input, we can have

a test for different current range to change the torque and speed.

2) L293D shield also has internal ESD protection so the components of the motor drive circuit are safe from electrostatic discharge.

3) It can have extremely fast response time so we don't need to additional design a controller for our motor control

to have torque and speed control.

4) L293D contains high noise immunity inputs, which can reduce our EMI in our system.

5) For output current 600mA to 1.2A per channel, it is suitable for our small DC brush motors.

6) It is easy to get and achievable for limited budget.

Instead of using motor driver itself, we use motor shield so we can plug it into our Arduino and wire the motors directly. The shield has the protection for our chip, so if there is a surge voltage more than 5V, overheating is still not a problem for our project.

Steering the wheels

Besides setting a base speed, we derive two equations for our PWM.

1) Base Speed=(PWMR-PWML)/2 (8)

2) Motor Differential Value=(PWMR-PWML) (9)

Since the motor drive shield support 2^8 digits, we can manipulate our speed with 256 sets and we also set up our base speed with different speed to test our robot.

Also, we extract the motor differential value from our controller and this variable is derived from our error signals, which is the distance between sensor and the lines. Then we time the error signals to a constant Kprop to enlarge the signal as our motor differential value. Hence we tune our speed and direction from these two equations.

For our motor's analysis, we analyze the relation between power efficiency and torque in the graph below. In order to increase our torque and speed, we increase our rated voltage from 4.5 to 6 volts which we can achieve best output power when torque is 15 oz-in and rpm equals to 100.

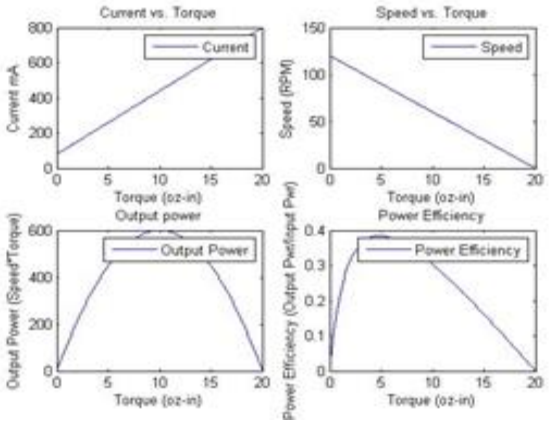

Fig. 7. Torque Power Relation at Input Voltage=4.5V

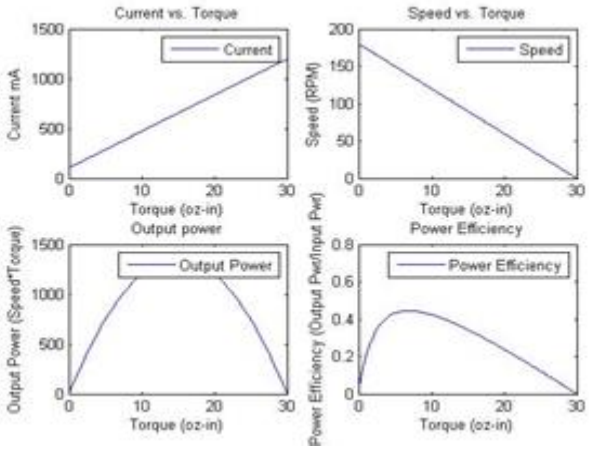

Fig. 8. Torque Power Relation at Input Voltage=6V

D. Chassis

Most of the chassis came from the original, black-line following robot made in ECE 306. This includes the frame of the robot, which is a red, 6 inch by 4 inch polymer board, the two 5 Volt DC motors, the two Lego wheels, the battery pack, and the rear pivot wheel. All of these parts were previously mounted on the chassis. Only the two DC motors and the battery pack had to be unmounted and remounted during the construction of the robot. The motors and battery pack had to be unmounted in order to more easily test that they were functioning properly. In addition, the motors had to be unmounted to more easily solder them to the motor shield. The only addition to the chassis in this project was the addition of wooden sensor mounts, which stabilize the sensor arrays in place and hold them at the precise locations needed by the sensor code.

1. Polymer Board

As mentioned above, the polymer board came from the original ECE 306 robot and was constructed at this time. The board was purchased from Amazon and was originally 12 inches by 24 inches with a thickness of 0.12 inches. The board was cut to be 6 inches in length and 4 inches in width, which provided enough space for the microcontroller board, motors, and rear pivot wheel. The battery pack was mounted on the bottom side of the polymer board and holes were drilled to provide a more direct route for the wiring and the microcontroller board.

For the lane-assist emulation robot in this project, the polymer board was kept the same size, which provided slightly more of challenge to mount the new parts. Although they were temporarily removed, the battery pack and the DC motors were kept in the

same locations on the board. The lego wheels were kept attached to the motos in the same way, and the rear pivot caster remained mounted to the board in the same spot. The wooden sensor mounts were mounted to the front of the robot and approximately take up the first inch of the board from side to side. Holes were drilled into the board which allowed the sensor mounts to be held in place by nuts and bolts.

2. **Motors**

   The 5 Volt motors were provided by the school and professor during the construction of the original robot. Each motor was kept in the same location and was mounted with three zip ties. Two of the zip ties go from the front of the motor to the back, and the other zip tie goes from side to side. The two zip ties that go from front to back went on first, and the zip tie that goes side to side went on last. Each motor was carefully held in place while the zip tie was being tightened as much as possible.

   During the construction of the lane-assist emulation robot, one of the motor's wiring broke away from the solder and had to be resoldered. Once completed, the motors were mounted in place, then connected to the motor driver shield.

3. **Lego Wheels**

   The lego wheels were also taken from the original robot since they fit the DC motor connection piece, which is a metal, hexagonal piece that sticks out from the side of the motor. Like the polymer board, the wheels and hubs were also purchased from Amazon. The 'balloon' style lego wheels come from the Technic lego series and are 94.8 mm in diameter. The Lego hubs are of the same style and series and are 44 mm in diameter.

4. **Pivot Wheel**

   The rear pivot wheel came from the original robot and was purchased from Amazon. The wheel is two inches in diameter and the top plate of the wheel is 2.56 inches by 1.88 inches. The top plate of the wheel has four holes in it; one at each corner, which were used to mount the wheel to the polymer board. The pivot wheel location was decided to be placed in the middle of the polymer board and about a half inch away from the rear edge. Once the location was decided, four holes were drilled into the board and the holes in the top plate of the wheel were lined up. Then, bolts were placed through the holes of the board and the holes in the top plate, and nuts were screwed on tightly to hold the wheel in place on the board. The mount and location of the rear pivot wheel was not changed for the new, lane-assist emulation robot.

5. **Battery Pack**

   The original robot battery pack was also kept and this was kept in the same location, on the bottom side of the polymer board, directly in the center of the board. The battery pack holds four double-A batteries and provides approximately six volts. Two holes were drilled just in front of the battery pack mount, which allows the connection wires to be connected the control board in a more direct route. The connection wires were soldered onto the battery pack, but the other end of the wires were fitted with plastic connection pieces, which allow the power to be easily connected and disconnected from the control board.

6. **Why did we keep these things?**

   The original robot chassis was kept the same as much as possible in order to better emphasize and measure the improvement of the lane-assist emulation robot. The polymer board provided enough space for the new control board, motor shield, and sensor mounts, and while we could have improved the shape, weight, and design, this was not necessary. The two DC motors would have been replaced if not to more accurately measure the improvements of the lane-assist emulation robot. Additionally, the lego wheels and pivot wheel could have been replaced with more dynamic and less resistant options, but in an effort to keep the robot as similar as possible, were kept. The battery pack on the other hand, had no inclination of being replaced, as it provided sufficient voltage for both robots and the size and mount location were not restrictive for either.

7. **What We Added**

   The only addition made to the lane-assist emulation robot was the sensor mounts. Because the new robot required two, eight sensor arrays on each side of the front of the robot, holding these in place would be more challenging than the original robot's one emitter and two sensors. In addition, the accuracy of each sensor reading is vital to the robot's success, and thus, the team decided that mounting the sensors in a permanent, stable, and precise location would improve the robot's track results.

   a. **Sensor Mounts**

      The sensor mounts were constructed of wood in the North Carolina State University Senior Design Workshop. The design of the mounts was based around the size of the sensor arrays. Knowing the width of the sensor arrays, two 3 inch by 1 inch by

0.5 inch pieces of wood were cut, and in these two pieces, a slim slot was cut into one of the 3 inch sides about 0.75 inches deep, and 0.2 inches tall. These slots allowed the sensor arrays to slide in and be secured in place.

Once the two sensor-slotted pieces were constructed and tested, a wooden mounting block was constructed. This piece was about 4 inches by 1.5 inch by 1.5 inch and was mounted to the top side of the front of the new robot by drilling holes into the board and securing the piece with wood screws. Once this piece was mounted to the robot, four holes were drilled into the board, two at each front corner and two 1 one inch in from the front corners. The two sensor-slotted pieces were then mounted to the mounting piece by drilling a wood screw in the corresponding locations from the bottom side of each sensor-slotted piece and into the mounting piece. The locations of our sensors were then measured as this would correspond to the width of our tack lanes.

III. DESIGN

A. *System Identification*

In order to identify a system a step input or an impulse input must excite the plant and the outputs must be measured [1]. Usually this would be simple with a system by manipulating what goes into the system, but for this system it is more complicated for a couple of reasons.

1. With a step input or an impulse the system would change the motor differential value to +1 because the motor differential variable is the output to our system (eg. the motor controller boards). If the input is a step input the bot would then begin to turn in a constant circle. If the input is an impulse the bot would turn left and stay left. To get a usable response the bot must stay around 0 or 1.
2. If the motor differential variable were to change to a +1 because of a forced input, the turn in the car would be very small. It may be too small to detect what is a variation in position on the track and what is noise from inconsistencies from the sensors.

The system identification methods were manipulated to avoid these complications. First, instead of having an open looped system that is excited by an input, feedback was provided to the system and then the system could be treated as an open loop by Eq. 10 [1]. This solved problem 1. A simple proportional controller to amplify the error to the system was included such that a more distinct variation in position with a given input could be observed (Fig. 9). The equation for finding the plant's transfer function (G(z)) then becomes Eq. 11 by including the proportional controller and solving for G(z).

$$T(z) = \frac{G(z)}{1 + G(z)} \tag{10}$$

$$T(z) = \frac{K_p G(z)}{1 + K_p G(z)} \Rightarrow G(z) = \frac{T(z)}{K_p(1 - T(z))} \tag{11}$$

A test track was also needed that could be used for system identification. The end goal was to be able to navigate a curved track that simulates a road, and a curved road would be a wide variety of inputs instead of a simple step or impulse, so a straight track was created to do system identification. The idea was that at a certain time (t=0) the input (U(z)) would

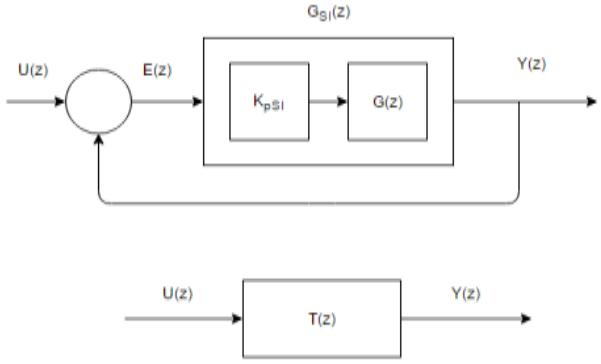

Fig. 9. Block diagram of system identification method.

change and the output (Y(z)) could then be recorded through a data acquisitioning technique. Then the input would be known, which would be determine a priori, and the output would be known, which would be obtained through testing.

After some preliminary testing, the team decided to use the MATLAB system identification toolbox technique for system identification because a step input could be used in this method instead of an impulse input [1]. The impulse input was not providing a clear enough response to get usable data and the step input was.

The team spent a few days testing the bot's response to a step input. The set speed PWM values of the bot's motors were varied to get a plant transfer function for each speed in case the team wanted to change the speed of the bot while running the track in the future. For each speed, a different

proportional controller constant had to be implemented because the stability of the bot was a function of the proportional controller and the set speed. The set speeds and their respected proportional controller constants are shown in Tab. 4.

Each category (eg. set speed/Kp combination) was run 3 times to get 3 different sets of data that represented the response/output of each trial. These 3 sets of data was merged into one set in the system identification toolbox in MATLAB [2] (Fig. 10).

Set Speed (PWM Value 0-255)	Proportional Controller Constant (K_p)
160	23
140	15
120	10
100	7

Tab. 4. Set Speed PWM Values and their respected proportional controller constants

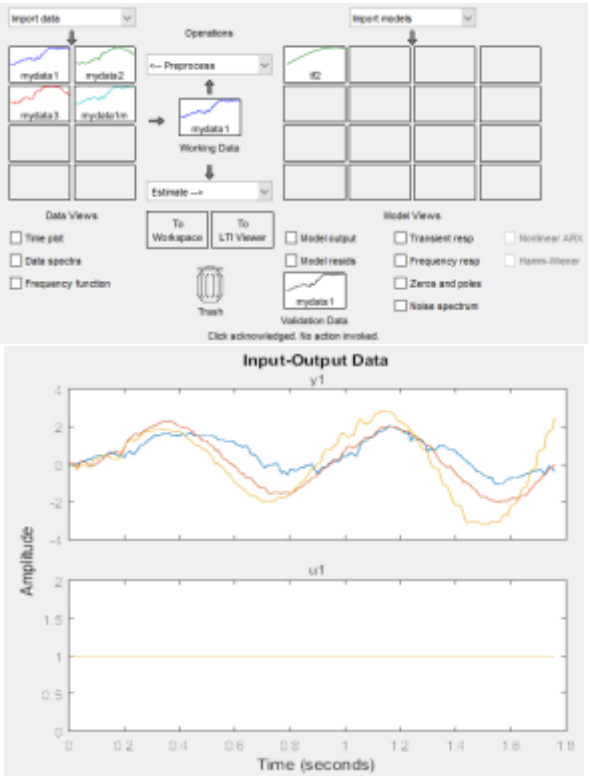

Fig. 10. System Identification toolbox and merged data from step inputs for set speed 160

The resulting transfer functions found from this method represented the closed loop transfer function (T(z)). In order to find the plant's open loop transfer function Eq. 11 was utilized. The resulting G(z)'s were recorded in Tab. 5.

Set Speed	Transfer Function (G(z))
160	$G_{160}(z) = \frac{0.0003315z - 0.0002177}{z^2 - 2.003z + 1.006}$
140	$G_{140}(z) = \frac{0.002538z - 0.002473}{z^2 - 2.034z + 1.038}$
120	$G_{120}(z) = \frac{-0.0001002z + 9.706e-05}{z^2 - 1.998z + 1}$
100	$G_{100}(z) = \frac{0.0000364z - 0.0000173}{z^2 - 1.998z + 1}$

Tab. 5. List of open loop transfer functions of the bot's plant at different speeds

B. Full Track

In order to truly test the lane-assist functionality of the robot, the team wanted to design the track to be as similar to a real road as possible by including lane-line variation, multiple kinds of curves, and an intersection. With this in mind, the team wanted to build two tracks: one to test the intersection and curve functionality and one more traditional circular track with inconsistent curving. Rather than building two tracks on separate pieces of paper, the team decided to build a track on each side of a paper board to reduce cost and required storage space. Material limitations forced the team to use a white background paper with black electrical tape for the lanes. With an opposite-colored track, the robot would work in the exact same way with the only difference being that the sensor values would be flipped to the other end of the spectrum.

1. Design of Track

As mentioned above, one side of the track was designed to test the intersection and curving functionality, while the other was built as a more traditional track. To test the robot's intersection functionality, the team designed a two-laned, four-way intersection with dashed lines separating each lane. The intersection was designed to integrate with the testing of the robot's curve-taking

ability so that, if the robot turns left, right, or goes straight through the intersection, it would then enter a curve. Each curve only consists of one inside-lane-line, and as perhaps the most difficult test for the robot, one of the four curves is dashed.

The design of the other side of track stuck with the simpler and more traditional track design. This track was designed to test the robot's ability to handle s-curves in conjunction with parts of the track that are straight. In addition, the team designed the track to test how the robot would respond to lane-line-variations at the same time, with parts of the track having lanes that are solid, dashed on one side, dashed on the other, or both dashed.

The team designed also designed each track to test the robot's ability to handle unforeseen problems, like if one of the lane-lines doesn't exist, if that one line is dashed, or if both lines of a lane are dashed, similar to a three or more lane highway.

2. Building the Track

Since the center of each sensor array was required to be directly over the center of each lane line, every part of the lane had to maintain a distance of 12.7 centimeters, making the construction of the track a careful and slow process. To ensure the lane maintained the 12.7 centimeter distance at all parts, the team constructed a 2 inch by 12.7 centimeter piece of cardboard. After one side of the lane was placed on the paper, the next step was for one team member to carefully drag the cardboard along the tape already on the track while another team member drew a line along the edge of the cardboard with pencil. Then, a team member could go back through and place tape along the outside of the pencil-drawn line. This method made maintaining the 12.7 centimeter distance much more manageable.

C. Proportional-Integral-Derivative (PID) Controller

The PID controller was designed using the pidtool function in Matlab. The G(z) found during System Identification for varying set speeds were used with a sampling time of T=0.01. The G(z) for each speed was passed into the pidtool function in matlab and the type of controller was changed to PIDF. The bandwidth and phase margin were adjusted in the PID tool (Fig. 11) until the overshoot was 0% and the settling and rise time were minimized (Fig. 13**)**. The control parameters (Fig. 13) were then inserted into Eq. 12 to calculate a D(z) for each G(z) (Tab. 6).

Fig. 11. Bandwidth and phase margin for 120 set speed G(z)

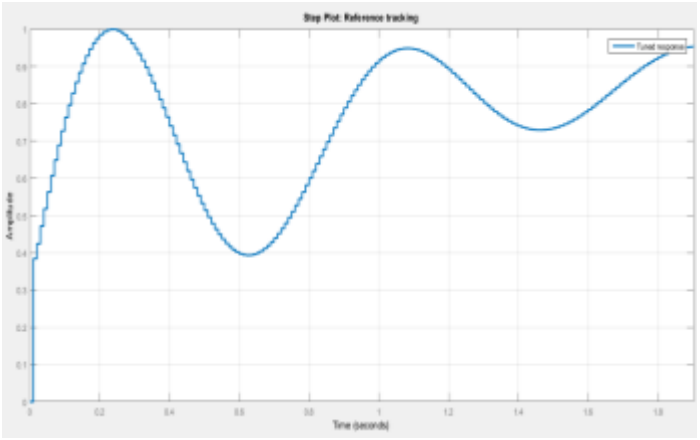

Fig. 12. Step response for 120 set speed G(z)

Controller Parameters

	Tuned
Kp	28.1731
Ki	103.0956
Kd	1.7305
Tf	0.014061

Performance and Robustness

	Tuned
Rise time	0.14 seconds
Settling time	4.17 seconds
Overshoot	0 %
Peak	0.999
Gain margin	-5.13 dB @ 7.22 rad/s
Phase margin	60 deg @ 10.5 rad/s
Closed-loop stability	Stable

Fig. 13. Control Parameters for 120 set speed G(z)

$$D(z) = K_P + K_I\left[\frac{T}{z-1}\right] + K_D\left[\frac{1}{T_F + \frac{T}{z-1}}\right] \quad (12)$$

Set Speed	D(z)
80	$\frac{15.62z^2 - 30.34z + 14.73}{0.154z^2 - 0.2981z + 0.144}$
100	$\frac{35.67z^2 - 69.63z + 33.98}{0.01525z^2 - 0.02049z + 0.005246}$
120	$\frac{-33.96z^2 + 62.84z - 29.06}{0.01726z^2 - 0.02451z + 0.007256}$
140	$\frac{2.127z^2 - 3.957z + 1.841}{0.01406z^2 - 0.01812z + 0.004061}$
160	$\frac{6.744z^2 - 13.06z + 6.321}{0.01366z^2 - 0.01732z + 0.003659}$

Tab. 6. PID controllers for different speeds

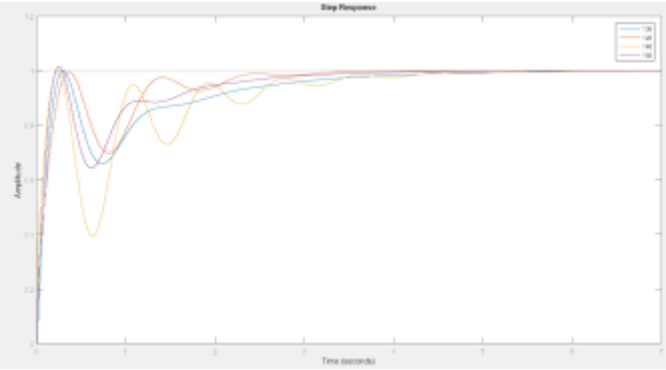

Fig. 14. Step responses for each D(z)

D. SISO-Lead Controller

SISO Lead Controller is a Phase-Lead Controller, which is obtained using the Sisotool command in MATLAB. The procedure for designing a Phase-lead controller is as follows [1]:

1. The controller zero is placed at a location co-incident with the system pole, so that it cancels it.
2. A random desired pole location is selected.
3. The controller pole is placed to the left of the zero. This results in a phase lead controller.
4. The gain K_c is selected in such a way so the resulting controller is a phase lead controller.

The design of a phase-lead controller is based on trial and error methods [1]. The equation for calculating Dz is given as Eq. 13:

$$Dz = \left| \frac{K_d\,(z - z_0)}{z - z_p} \right|_{z=1} \tag{13}$$

Where,

- K_d : Gain
- Z_0: zero of the controller (obtained from the Sisotool adjustments)
- Z_p: pole of the controller (obtained from the Sisotool adjustments)

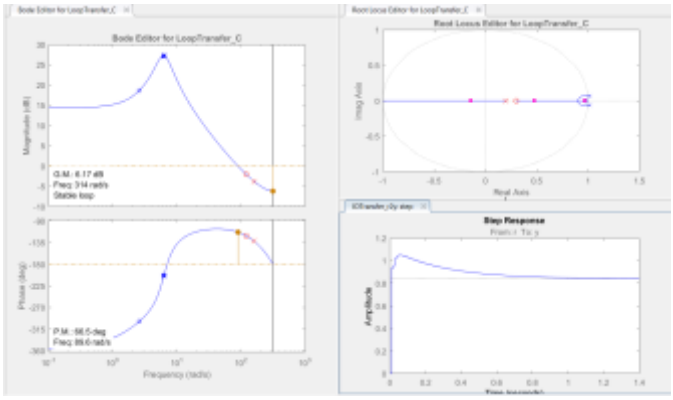

Fig. 15. SISOTOOL window in MATLAB

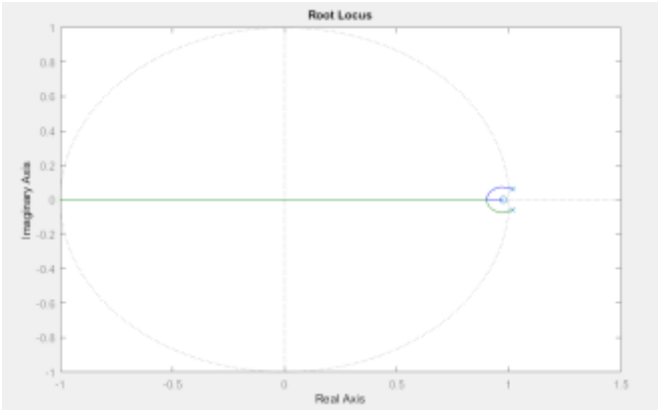

Fig. 16. Root locus of G(z) for SS=140

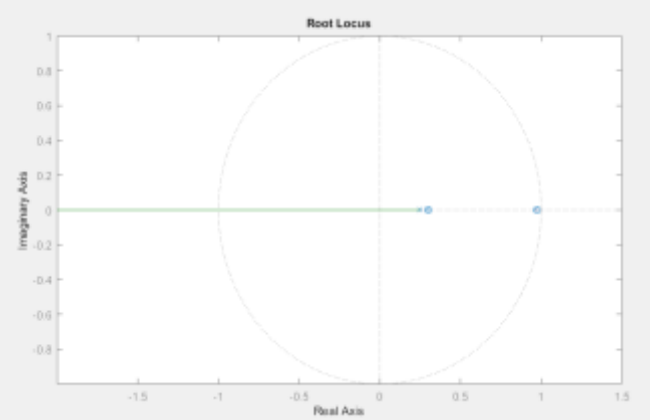

Fig. 17. Root locus of T(z) for SS=140

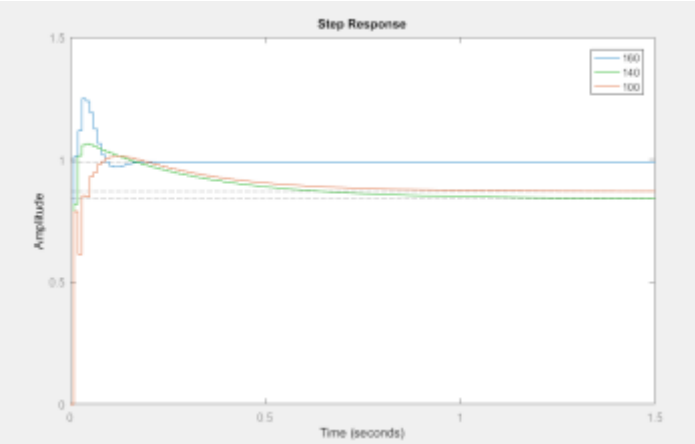

Fig. 18. Step responses of the system (Tz) for various set speeds

Set Speed	K_{prop}	G(z)	D(z)	T(z)
160	23	$\frac{0.0003315z - 0.0002177}{z^2 - 2.003z + 1.006}$	$\frac{139.3z - 39.36}{z - 0.07}$	$\frac{1.013z^2 - 0.9691z + 0.1996}{z^3 - 1.06z^2 + 0.1771z + 0.1291}$
140	15	$\frac{0.002538z - 0.002473}{z^2 - 2.034z + 1.038}$	$\frac{272.9z - 259.2}{z - 0.07}$	$\frac{0.8185z^2 - 1.043z + 0.2393}{z^3 - 1.515z^2 + 0.6051z + 0.07214}$
100	7	$\frac{3.636 * 10^{-5}z - 1.724 * 10^{-5}}{z^2 - 1.998z + 1}$	$\frac{1000z - 950}{z - 0.95}$	$\frac{0.7882z^2 - 1.137z + 0.3618}{z^3 - 1.219z^2 + 0.1187z + 0.3528}$

Tab. 7. Lead controllers list

E. Prediction Observer Controller

A desired characteristic equation was needed for the prediction observer controller. This characteristic equation was desired to simulate the performance parameters that were set for the team (rise time, settling time, overshoot, etc.). To find the poles that satisfy these requirements, the team looked to the MATLAB SISO tool.

A discrete transfer function of gain 1 and no poles nor zeros was passed into the tool. This was necessary such that the SISO tool could recognize that it was discrete and the design could then be done in the Z domain (Fig. 19).

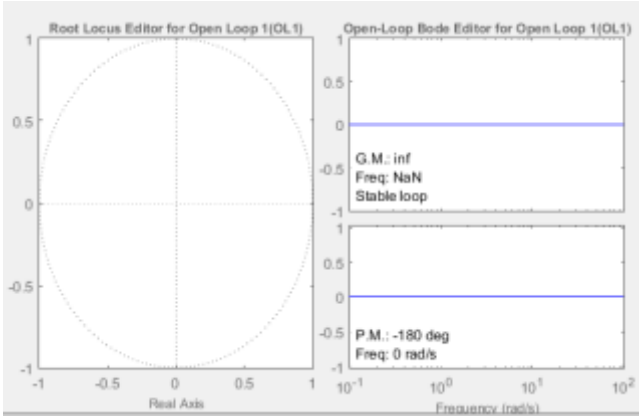

Fig. 19. Blank siso tool Z domain plane and bode plots

The SISO tool provides a design requirements feature that shows where on the Z plane a certain requirement must have a pole location. A max of 2 requirements was chosen because this would provide a crossing point that could satisfy 2 requirements at the same time. A settling time requirement was chosen as 1 second and a percent overshoot requirement was chosen as 10% (Fig. 20).

Fig. 20 shows the desired open-loop pole location for the system. These pole locations can be found in Eq. 14.

$$P_{1,2} = 0.96034 \pm j0.05225 \tag{14}$$

The desired pole locations were put into a matrix and expressed as Eq. 15.

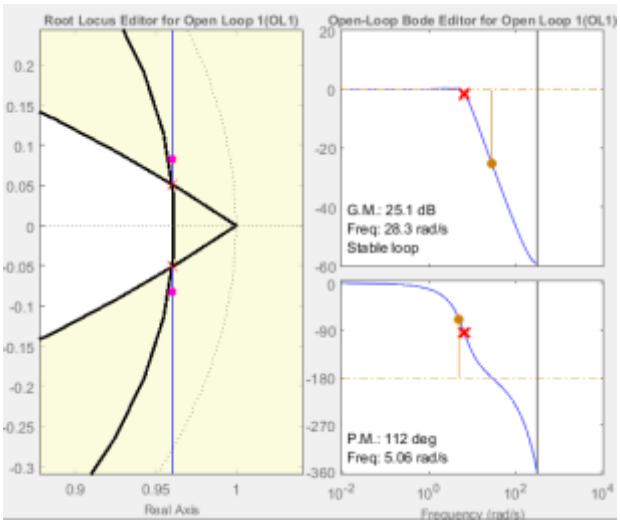

Fig 20. Root locus of poles at a desired location.

$$P_{des} = \begin{bmatrix} 0.96034 + j0.05225 & 0.96034 - j0.05225 \end{bmatrix} \quad (15)$$

Therefore, the desired characteristic equation was defined by Eq. 16.

$$C(z) = z^2 - 1.921z + 0.925 \quad (16)$$

Now, using the known transfer functions found from system identification, Ackermann's formula can be used to find K and G matrices to design a prediction observer controller to give a characteristic equation defined above.

The state space model of the system's 160 set speed plant is shown below in Fig. 21. Only the 160 set speed transfer function was used in the design of the prediction observer controller because after the first prediction observer design was finished, simulation and testing showed the controller to not be an optimal choice. Furthermore, the controller could be tuned for different speeds by varying the gain of the controller.

$$A = \begin{bmatrix} 2.003 & -1.006 \\ 1.000 & 0.000 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 3.315 * 10^{-4} & -2.177 * 10^{-4} \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \end{bmatrix}$$

Fig. 21. State space model of the system plant found from system identification methods.

The MATLAB function "acker" was used to find the K gain matrix for each state [1]. The A, B, and P_{des} matrices were given to the acker function as parameters and the function returned the 1X2 K matrix (Eq. 17).

$$K = \begin{bmatrix} 0.0823 & -0.0810 \end{bmatrix} \quad (17)$$

Originally the prediction observer desired characteristic equation was set to the same desired characteristic equation as in Eq. 16. However, doing this produced a steady state step response to be negative. After further trial and error the new desired characteristic equation was set to be 0.1 times the original. This resulted in the new characteristic equation having the following pole locations (Eq. 18).

$$P_{desPO} = \begin{bmatrix} 0.0960 + j0.0052 & 0.0960 - j0.0052 \end{bmatrix} \quad (18)$$

Again the MATLAB function "acker" was used to find the G matrix used in a prediction observer controller [1]. Instead of passing in the A, C, and P_{des} matrices as the arguments, A^T, C^T, and P_{desPO} were passed into the "acker" function. The calculated G matrix can be found below (Eq. 19).

$$G = \begin{bmatrix} 2.3337 \\ -4.7649 \end{bmatrix} \quad (19)$$

Finally, the prediction observer controller was determined by Eq. 20 [1], and the resulting transfer function was Eq. 21.

$$D_{PO}(z) = \mathbf{K}(z\mathbf{I} - \mathbf{A} + \mathbf{BK} + \mathbf{GC})^{-1}\mathbf{G} \quad (20)$$

$$D_{PO}(z) = \frac{578.1z - 567.7}{z^2 - 0.1097z - 0.1143} \quad (21)$$

The resulting closed-loop transfer function is Eq. 22.

$$T_{160}(z) = \frac{0.19166(z - 0.9812)(z - 0.6567)}{(z^2 - 0.1921z + 0.00925)(z^2 - 1.921z + 0.925)} \quad (22)$$

Note that the closed loop transfer function has a part of the characteristic equation that is identical to the desired characteristic equation from Eq. 16. The other part of the characteristic equation is identical to the desired characteristic

equation produced by the desired pole locations of the prediction observer from Eq. 18.

To check the effectiveness of the controller a step input was simulated to excite the system. The response was recorded and the result can be found in Fig. 22.

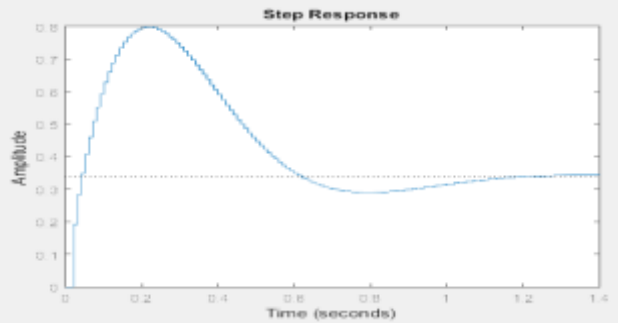

Fig. 22. Step response of prediction observer controller being implemented into the set speed 160 plant.

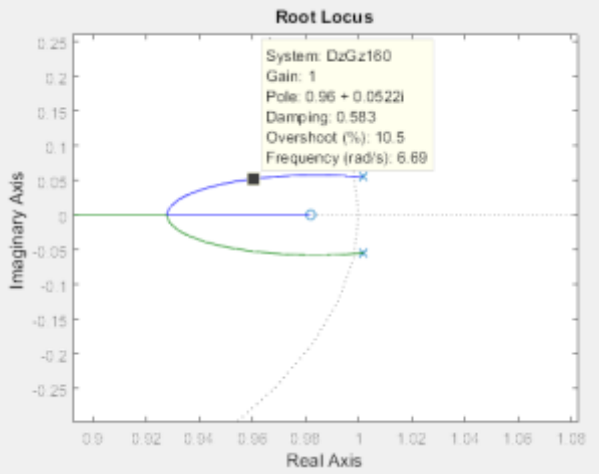

Fig. 23. Root locus of open loop system which shows pole location

The response's steady state value was about 0.34 which resulted in a steady state error of 0.66. The prediction observer proved to be difficult in bringing the steady state error down to 0, hence why the error was so large. There was no overshoot in the simulation because the response never went above 1. Also, the settling time was about 1 second which matches one of the design requirements set during the design of the desired characteristic equation.

The root locus of the open looped system was plotted to check the location of the poles (Fig. 23). Ideally the controller would place the poles in the desired location as was indicated in Eq. 14.

The pole locations at a gain of 1 were, according to the root locus, 0.96+- j0.0522, which was the desired pole location of the prediction observer controller.

Next, the desired pole locations were tuned to try to find a more desirable step response than the one above. This was done by decrementing the real parts of the desired pole location at small intervals and, by following the same steps above, implementing a controller into the system and exciting it with a step. The pole locations that gave the best step response is Eq. 23.

$$P_{best} = \begin{bmatrix} 0.85 + j0.052 & 0.85 - j0.052 \end{bmatrix} \quad (23)$$

The step response for these pole locations can be found in Fig. 24.

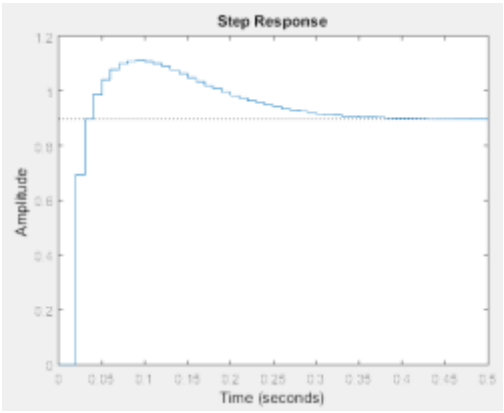

Fig. 24. Step response of tuned prediction observer controller and plant

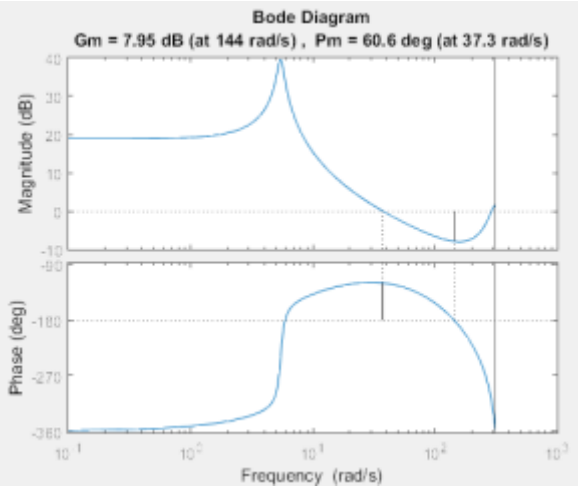

Fig. 25. Gain margin and phase margin of tuned prediction observer and system plant

The gain and phase margins for these pole locations can be found in Fig. 25.

Based on the goals set earlier, this controller satisfies all design requirements for a step input except two. The overshoot (above 1) was about 11% which is a little more than 10% which was the goal. This is probably due to moving the pole locations in order to tune the controller to behave properly. Also, the gain margin was 7.95 dB, which was much smaller than the goal (30 dB). However, the gain margin being off of the original goal was not much of a concern for the team.

F. Infinite-Horizon Linear-Quadratic Regulator Controller

Optimal design of an Infinite-Horizon Linear-Quadratic Regulator (IH-LQR) Controller stabilizes a regular system by minimizing the quadratic cost function, shown in Eq. 24, where *N* is finite, and **Q**(k) and **R**(k) are symmetric [1].

$$J_N = \sum_{k=0}^{N} \mathbf{x}^T(k)\mathbf{Q}(k)\mathbf{x}(k) + \mathbf{u}^T(k)\mathbf{R}(k)\mathbf{u}(k) \tag{24}$$

Designing the cost function requires some trial and error [1]. First, the steady-state gain matrix **K** is calculated with Eq. 25, where **P**(k) is equal to Eq. 26.

$$\mathbf{K}(k) = [\mathbf{B}^T\mathbf{P}(k+1)\mathbf{B} + \mathbf{R}]^{-1}\mathbf{B}^T\mathbf{P}(k+1)\mathbf{A} \tag{25}$$

$$\mathbf{P}(k) = \mathbf{A}^T\mathbf{P}(k+1)[\mathbf{A} - \mathbf{B}\mathbf{K}(k)] + \mathbf{Q} \tag{26}$$

Matlab conveniently provides a function in the Control System Toolbox that can make these calculations called ***lqr*** [4]. In addition to the state-feedback gain matrix **K**, ***lqr*** returns the solution **S** of the associated Riccati equation and the closed-loop eigenvalues **e.** The algebraic Riccati equation is shown in Eq. 27 and the syntax for ***lqr*** is shown in Figi. 26.

$$A^TS + SA - (SB + N)R^{-1}(B^TS + N^T) + Q = 0 \tag{27}$$

```
[K,S,e] = lqr(SYS,Q,R,N)
[K,S,e] = LQR(A,B,Q,R,N)
```

Fig. 26. Syntax for lqr in MATLAB

The next step is to calculate the Kalman Gain Matrix, **G**, for a Kalman state estimator using the covariance equations Eq. 28, Eq. 29, and Eq. 30, where **M**(k) is the covariance of the prediction errors, as shown in Eq. 31 [1]. **M**(k) also uses the Kalman filter equations shown in Eq. 32, where **q**(k) is the actual state estimate at k and $\mathbf{q}^-(k)$ is the predicted state estimate at the sampling instant k [1].

$$\mathbf{G}(k) = \mathbf{M}(k)\mathbf{C}^T[\mathbf{C}\mathbf{M}(k)\mathbf{C}^T + \mathbf{R}_v]^{-1} \tag{28}$$

$$\mathbf{P}(k) = \mathbf{M}(k) - \mathbf{G}(k)\mathbf{C}\mathbf{M}(k) \tag{29}$$

$$\mathbf{M}(k+1) = \mathbf{A}\mathbf{P}(k)\mathbf{A}^T + \mathbf{B}_1\mathbf{R}_w\mathbf{B}_1^T \tag{30}$$

$$\mathbf{M}(k) = E\{[\mathbf{x}(k) - \bar{\mathbf{q}}(k)][\mathbf{x}(k) - \bar{\mathbf{q}}(k)]^T\} \tag{31}$$

$$\mathbf{q}(k) = \bar{\mathbf{q}}(k) + \mathbf{G}(k)[\mathbf{y}(k) - \mathbf{C}\bar{\mathbf{q}}(k)]$$
$$\bar{\mathbf{q}}(k+1) = \mathbf{A}\mathbf{q}(k) + \mathbf{B}\mathbf{u}(k) \tag{32}$$

Using these equations in Matlab in conjunction with a *for loop*, as shown in Fig. 27 below, we are able to calculate the Kalman Gain Matrix **G**.

```
Rw=5; Rv=10; M=eye(2); B1=[0;1200]; N=3000;
for k=1:N % Equations D5, D6, and D7
    G = M*C'*inv(C*M*C' + Rv);
    P = M - G*C*M;
    M = A*P*A' + B1*Rw*B1';
    [k,G'];
end
G
```

Fig. 27. MATLAB code for loop

Now that we know the Kalman Gain, **G**, and the steady-state gain, **K**, we are able to calculate the IH-LQR Controller transfer function, $\mathbf{D_{ce}}(z)$, which is shown in Eq. 33 below.

$$D_{ce}(z) = z\mathbf{K}[z\mathbf{I} - \mathbf{A} + \mathbf{G}\mathbf{C}\mathbf{A} + \mathbf{B}\mathbf{K} - \mathbf{G}\mathbf{C}\mathbf{B}\mathbf{K}]^{-1}\mathbf{G} \tag{33}$$

At this point, we have the design for the controlled system, excluding the key values. This is where trial and error becomes important. By adjusting the values for **R** in the cost equation and $\mathbf{R_w}$, $\mathbf{R_v}$, and the $\mathbf{B_1}$ matrix in the covariance equations, we are able to compare the step responses and determine the best overall design. In addition, the number of *for loop* iterations **N**, in Matlab, can be adjusted to improve the design.

For the Lane-Assist Emulation Robot, the best results were found by minimizing the value of **R** as much as possible to 0.01, and through trial and error, the best values for $\mathbf{R_w}$, $\mathbf{R_v}$, **N**, and the $\mathbf{B_1}$ were found to be 5, 10, 3000, and [0;1200], respectively.

These values yielded a controller, $\mathbf{D_{ce}}(z)$, with the transfer function shown in Fig. 28, and a new closed-system transfer function shown in Fig. 29**.**

```
Dcez =

   2587.2 z (z-0.7459)
  ---------------------
  (z-0.424) (z-0.00727)

Sample time: 0.01 seconds
Discrete-time zero/pole/gain model.
```

Fig. 28. IH-LQR Controller in ZPK format

```
Tz =

          0.85766 z (z-0.6567) (z-0.7459)
  ---------------------------------------------------
  (z^2 - 0.01942z + 0.004885) (z^2 - 1.557z + 0.6349)

Sample time: 0.01 seconds
Discrete-time zero/pole/gain model.
```

Fig. 29. Closed-loop transfer function with

The step response for this system is shown in Fig. 30, and information about the step response is shown in Tab. 7.

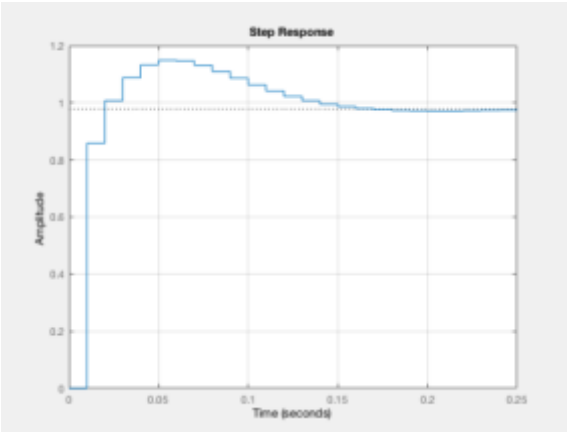

Fig. 30. Simulated step response of IH-LQR controller

Controller	Infinite-Horizon Linear-Quadratic Regulator
Gain Margin (dB)	7.3023
Phase Margin (°)	65.4571
Rise Time (seconds)	0.01
Settling Time (seconds)	0.14
Minimum Settling Time (seconds)	0.9715
Maximum Settling Time (seconds)	1.1485
Overshoot (%)	17.4858
Undershoot (%)	0
Peak (%)	1.1485
Peak Time (seconds)	0.05
Steady-State Error (%)	0.0224

Tab. 7. Step response info of IH-LQR controller

G. Root Locus Controller

1. Advantages of Root Locus

1) Root locus technique in control system is easy to implement in a linear system.

2) Using root locus method we can easily predict the performance of the whole system.

3) Root locus provides the better way to indicate the parameters like Kd and compensating poles and zeros.

2. Disadvantages of Root Locus

1) Since root locus based on a linear system, so it may not perform well on a nonlinear system.

2) The approach does not consider the optimal solution at all. It only works well on systems defined through time domain tests.

3) Loses significance at high frequencies or high degrees of damping so if our sample change to smaller than 0.01, it may not perform well and the designs are susceptible to noise and drifts.

3. Analysis for root locus method

In our original system, we derive a second-order transfer function. At T=0.01 the transfer function have two conjugate poles at 0.999+0.477i , 0.999-0.477i and a real zero at 0.9058. Hence, we try to place two compensating zeros which are equal to two poles and we also place two poles to improve the original zero. First, we find our new Kd in root locus plot, where the ideal value for Kd is at damping equal to 1. Then we place our compensating poles close to 0 and 1 to realize

which parameters can reach our best performance for our new controller. As we analyze the performance for our new design, our first priority is to reduce system overshoot so the robot would not oscillate and can follow the track precisely. Then we make rise time and settling time as fast as it can. However, it is also preferable that rise time should not faster than our system response time so our system can still respond. At last, we check our phase margin and gain margin to make sure these parameters can meet our requirement.

For different Kprop and base speed, we obtain different values of transfer function and new Kd. The tables below show the different scenarios for the analysis.

Base Speed	Kprop	Kd	Compensating Poles	Compensating Zeros
160	23	29.5	1, 0.5	1.01
160(Part 2)	23	4190	1, 0.25	1.0015
140	15	92.2	1, 0.1	1.017 0.0609i
120	10	250	1, 0	0.990
100	7	46200	1, 0.8	0.990

Tab. 8. Root Locus Design for Kd, Compensating Poles and Zeros

Base Speed	Original Tz	Modified Tz
160	$\frac{0.02556Z-0.02105}{Z^2-1.944Z+1}$	$\frac{0.7552Z^3-2.146Z^2+2.025Z-0.634}{Z^4-2.765Z^3+2.405Z^2-0.5161Z-0.1235}$
160 Part 2	$\frac{0.0076Z-0.00499}{Z^2-1.944Z+1}$	$\frac{1.389Z^3-3.694Z^2+3.224Z-0.9176}{Z^4-1.864Z^3+0.06545Z^2+1.466Z-0.666}$
140	$\frac{0.03737Z-0.03641}{Z^2-1.996Z+1}$	$\frac{0.234Z^3-0.704Z^2+0.70675Z-0.2367}{Z^4-2.9Z^3+2.671Z^2-0.63851Z-0.1329}$
120	$\frac{-0.001002Z+0.0009711}{Z^2-1.999Z+1}$	$\frac{-0.02505Z^3+0.07274Z^2-0.07038Z+0.0227}{Z^4-3.023Z^3+3.0715Z^2-1.07Z+0.02269}$
100	$\frac{0.0002545Z+0.00001207}{Z^2-1.998Z+1}$	$\frac{1.68Z^3-4.153Z^2+3.271Z-0.7965}{Z^4-2.118Z^3+1.244Z^2-01271Z+0.003512}$

Tab. 9. Original Tz and modified Tz

We derived our Kd from our root locus plot, but for some cases the value of Kd may not be desirable since the original transfer function varies. For base speed equal to 100 and 160, Kd is much larger than others. The graphs below show root locus plot.

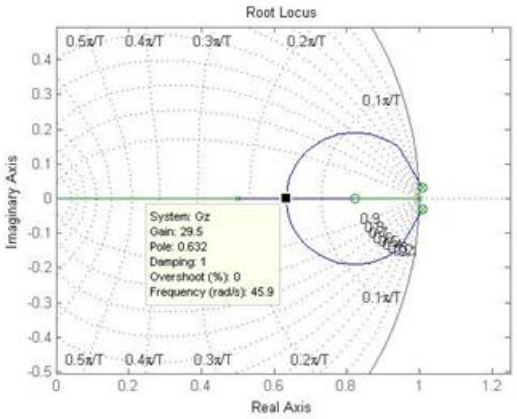

Fig. 31. Root Locus for Base Speed 160

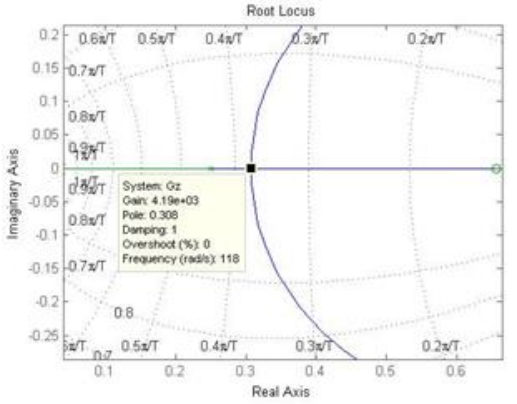

Fig. 32. Root Locus for Base Speed 160 for Second Analysis

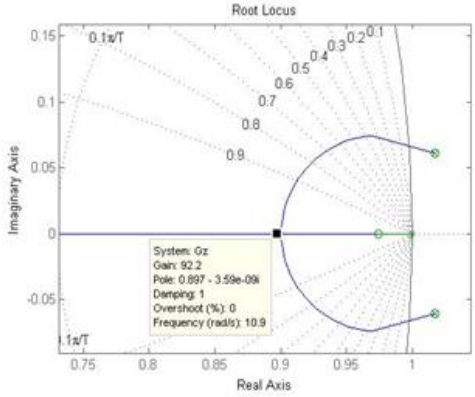

Fig. 33. Root Locus for Base Speed 140

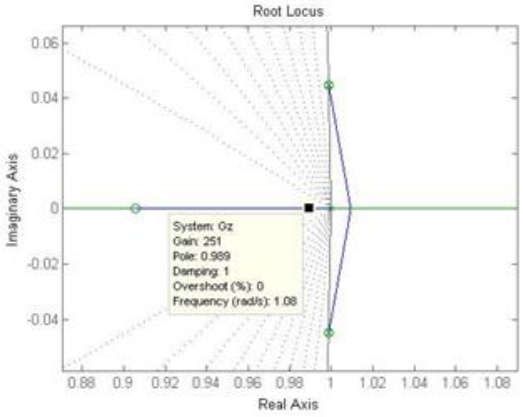

Fig. 34. Root Locus for Base Speed 120

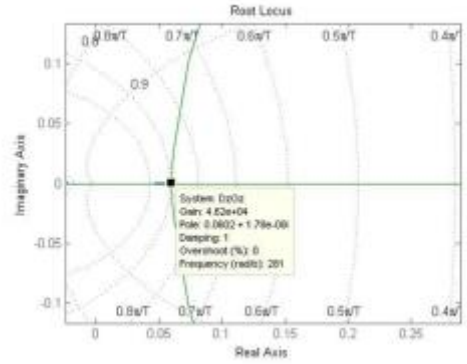

Fig. 35. Root Locus for Base Speed 100

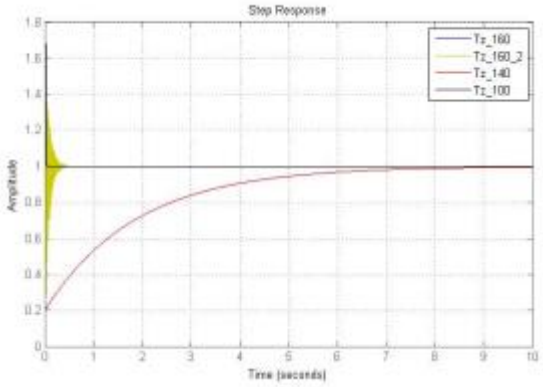

Fig. 36. Step Response for Different Speed

We found out than there is a trade-off between steady state error and response time, if we make our response time shorter , steady state error would increase a lot. For some of the cases, error after 20 seconds make our system unstable.

Still the root locus controller method is not flawless, as we use MATLAB to check the performance for our system. Due to the numerical round off error in MATLAB, an exact cancellation was not possible. A system is still unstable after compensating poles and zeros. As we try to replicate the same method like problem 4 in 2013 quiz 2. The response takes much longer to blow up and the steady state error after 20 seconds is undesirable for some cases. Even if a perfect poles and zeros cancellation were possible, the system would not be practically implementable because of internal stability issues. For example, if you designed a controller with an unstable pole that cancelled a zero in the plant, the output of the controller could grow unbounded to the point that an actuator would saturate or the plant could reach some physical limits. Therefore, the linear model of the plant would no longer hold and you wouldn't get the desired cancellation effect.

H. Current Observer

Using the transfer functions of the vehicle, for the various set speeds, developing a control observer controller became very similar to how it was done in class.

After the discrete transfer function equations for the different plants were inputted, it was necessary to find the state space representation of the discrete transfer function of the plant(s). The following Fig. 37 will strictly show the design process for the Gz_{160ss} for simplicity.

$$Ad = \begin{bmatrix} 2.0200 & -1.0210 \\ 1.0000 & 0 \end{bmatrix}$$

$$Bd = \begin{bmatrix} 0.2500 \\ 0 \end{bmatrix}$$

$$Cd = \begin{bmatrix} 0.1022 & -0.0842 \end{bmatrix}$$

$$Dd = \begin{bmatrix} 0 \end{bmatrix}$$

Fig. 37. State space representation of set speed 160

After finding the state space representation, a predictability Matrix was needed. The values used were randomly assigned to this matrix and changed through trial and error. Using the predictability matrix, the A, B, C and D matrices are calculated and will be used for the rest of the controller.

$$P = \begin{bmatrix} 8 & 8 \\ 8 & 7 \end{bmatrix} \quad (34)$$

$$A = P^{-1} \times Ad \quad (35)$$

$$B = P^{-1} \times Bd \quad (36)$$

$$C = P \times Cd \quad (37)$$

$$D = Dd \quad (38)$$

The response needed to be faster so the poles need to be adjusted accordingly. The values of .9 x pole 1 and .85 x pole 2 were used as the desired pole locations that were plugged in the acker() function in MatLab and resulted with the following gain matrix.

$$K = \begin{bmatrix} 0.4021 & 1.3618 \end{bmatrix} \quad (39)$$

As stated before, a quicker response was desirable, so using the matrix of the desired poles and multiplying it by a fraction will decrease the rise time.

$$P_{err} = .5P \quad (40)$$

Now using the acker() function with these new, much lower poles another gain matrix was calculated. Using equations 9-73 with this new gain matrix, the controller can be found.

$$G = acker(A', (C * A)', P_{err})' \quad (41)$$

Which yields:

$$G = \begin{bmatrix} -19.8425 \\ 16.0704 \end{bmatrix} \quad (42)$$

Then to get Dz:

$$Dz = z * K * inv((z * eye(2) - A + G * C * A + B * K - G * C * B * K)) * G \quad (43)$$

After some MatLab manipulation, this provides the following transfer function

$$Dz = \frac{33.99z^2 + 375.3e2z}{11.1z^2 - 7.174z + 1.693} \quad (44)$$

Multiplying this transfer function by the Gz_{160ss} then applying feedback to the system gives the closed loop transfer function Tz. Then a unit step input can be applied to verify the results are what were expected. The following plot is of all the different plants and their corresponding closed loop step responses.

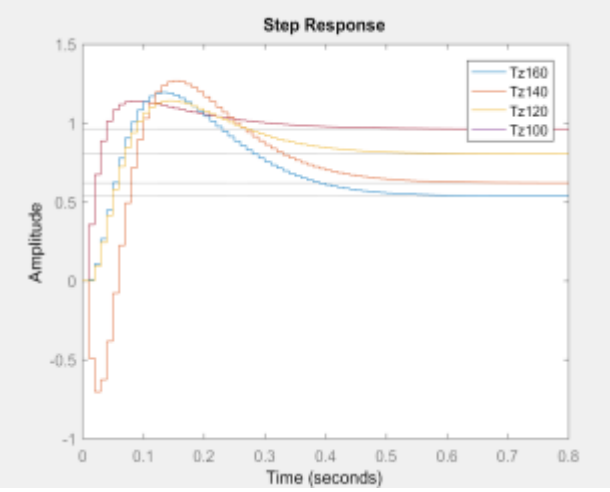

Fig. 38. Step responses of different current observer controllers for each speed

I. Transferring Controllers from Matlab to C++

With the system transfer function identified, digital controllers were created in order to stabilize the system and to optimize the response parameters. With the digital controllers designed in MATLAB, the controllers needed to be transferred into C++ notation.

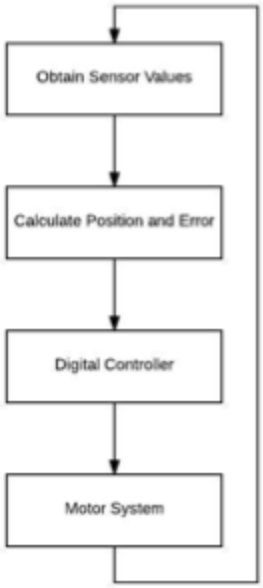

Fig. 39. The overall coding algorithm of the system in C++

The designed controllers in MATLAB are in the transfer function format below.

$$\frac{A_0 z^n + A_1 z^{n-1} + \cdots + A_n z^0}{B_0 z^n + B_1 z^{n-1} + \cdots + B_n z^0} \tag{45}$$

The controllers were transferred to their difference equations using the following equations.

$$\frac{Y(z)}{X(z)} = \frac{A_0 z^0 + A_1 z^{-1} + \cdots + A_n z^{-n}}{B_0 z^0 + B_1 z^{-1} + \cdots + B_n z^{-n}}$$

$$Y(z)(B_0 z^0 + B_1 z^{-1} + \cdots + B_n z^{-n}) = X(z)(A_0 z^0 + A_1 z^{-1} + \cdots + A_n z^{-n})$$

$$Y(z)B_0 z^0 + Y(z)B_1 z^{-1} + \cdots + Y(z)B_n z^{-n} = X(z)A_0 z^0 + X(z)A_1 z^{-1} + \cdots + X(z)A_n z^{-n}$$

$$B_0 Y(k) + B_1 Y(k+1) + \cdots + B_n Y(k+n) = A_0 X(k) + A_1 X(k+1) + \cdots + A_n X(k+n)$$

$$B_0 Y(k) = A_0 X(k) + A_1 X(k+1) + \cdots + A_n X(k+n) - B_1 Y(k+1) + \cdots + B_n Y(k+n)$$

$$Y(k) = \frac{A_0 X(k) + A_1 X(k+1) + \cdots + A_n X(k+n) - B_1 Y(k+1) + \cdots + B_n Y(k+n)}{B_0}$$

Fig. 40. List of equations used for coding controllers

The difference equations were utilized to transfer the values to the motor differential (M_D) values as the output to the Error (E) values as the input. The difference equation gave the current motor differential values to give to the motors.

$$M_D(k) = \frac{A_0 E(k) + A_1 E(k+1) + \cdots + A_n E(k+n) - B_1 M_D(k+1) + \cdots + B_n M_D(k+n)}{B_0} \tag{46}$$

The current motor differential calculated from the controller equations are typecasted as integers and sent to the motors.

J. Navigation Algorithms

1. Finite State Machine

The navigation algorithm for the vehicle was based on a finite state machine (FSM) due to its simple implementation and trusted ability by embedded systems engineers. The finite state machine of our vehicle has four states: initialize (init), follow the left line state, follow the right line state, and the intersection state. The FSM's state transition diagram can be seen in Fig. 41.

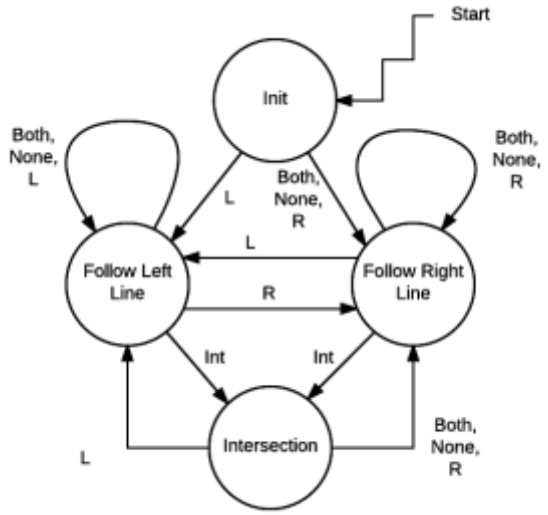

Fig. 41. Finite state diagram

The transitions for the FSM are given below:

- R = Only the right line sensed
- L = Only the left line sensed
- Both = Both lines sensed
- None = None of the lines sensed
- Int = Intersection sensed

2. Intersection Algorithm

The intersection algorithm was implemented to determine when the vehicle hit an intersection on our intersection track. The intersection track is shown in Fig. 42 and was modeled after a real intersection layout. The vehicle enters the

intersection state when sum of sensors on the right sensor array of the vehicle exceeds a certain preset threshold. When the vehicle enters the intersection state, the vehicle first comes to a complete stop. Then, the vehicle picks one of the three potential movements of turning right, turning left, or continuing straight. These movements are open loop and were hardcoded for our intersection track. The vehicle drives on the right lane of the road and will also turn into the right lane of its chosen turn direction.

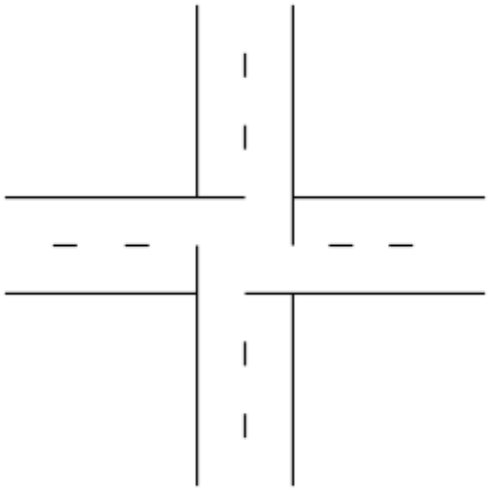

Fig. 42. Intersection track center layout

V. RESULTS

After designing and simulating the controllers, the team tested each of the controllers and recorded the results. Each controller was tested on a straight track with a step input, as well as on each of the full tracks. Below are the results for the step input response and full track response for each of the tested controllers. In addition, the team discusses their conclusions for each of the controllers.

A. PID

Although several of the designed PID controllers had good rise times, settling times, and overshoots when they were simulated in Matlab, they did not perform well on the track and proved to be unstable. The only PID controller that proved to be stable was the one designed for a set speed of 80. The others could be improved with some tuning but were still not the best controllers for the system.

1. Step Input Response

The PID step response was fairly oscillatory however it did not appear to go entirely unstable very quickly. It managed to follow the straight line to the end but it never leveled out to go completely straight.

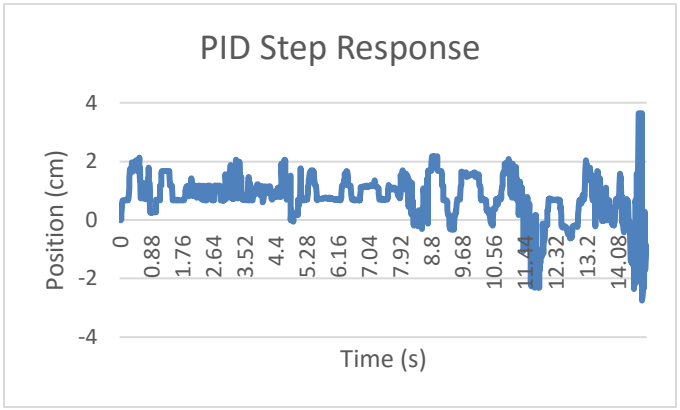

Fig. 43. Real step response to PID controller

2. Full Track Response

The PID controller followed the track and took the curves well but it still showed a lot of oscillatory behavior. Depending on the amount of oscillatory behavior when it got to certain curves it was able to continue following the track but at other times, the two did not pair well and the vehicle ran off the track.

B. Lead

1. Step Input Response

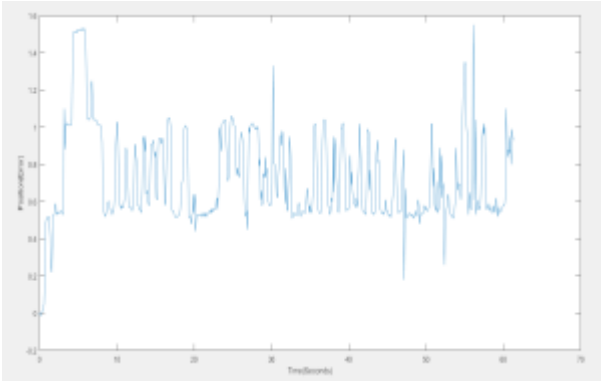

Fig. 44. Real step response to the phase lead controller

2. Full Track Response

The robot could navigate through all the curves and varied track types. It could navigate through the both the solid lines, one dotted line, two dotted line and just 1 solid line too. The robot could also navigate through the intersection and it was stable all the speed (100, 140, 160).

The Siso Lead Controller was probably the only controller that worked at all the set speeds and navigated through all the curves properly.

C. Prediction Observer

1. Step Response

The first prediction observer controller's simulation showed a steady state error of about 0.6, which was undesirable for the team. When that controller was tested on the real bot, the response was recorded on Fig. 45.

Note that this controller's simulation resembled that of the simulation from Fig. 22. The steady state error in the simulation was about 0.6, and in this one it was about 0.5. Also, it never reached 1 which was consistent with the simulation model. This gave confidence to the team that the model of the system was accurate.

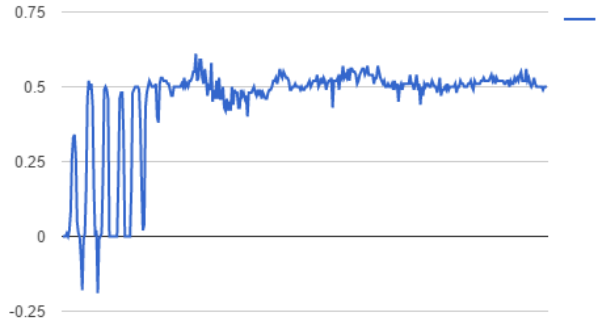

Fig. 45. Real step response of first prediction observer controller

The second prediction observer controller was tested on a step input, and that response was plotted in Fig. 46.

Fig. 46. Step response of system with second (better) prediction observer controller

This response was acceptable for the group. It had no overshoot at all, and settled in less than 0.25 seconds. Also, it looked as if the steady state error was 0, which was better than the simulated steady state error of about 0.1.

2. Full Track Response

When this controller was tested on the full track, the response was fairly choppy. It did follow the path relatively well; however, when it reached the area of the track that contains the dashed lines, it ran off of the track.

D. Infinite-Horizon Linear-Quadratic Regulator Controller

While the experimental results of the Infinite-Horizon Linear-Quadratic Regulator (IH-LQR) Controller varied a great deal from the simulation, the robot actually performed quite well. Overall, the robot remained stable and completed the track in an appropriate amount of time and did well compared to the other controllers.

1. Step Input Response

The IH-LQR Controller measured step response, shown in Fig. 47, varied significantly from the simulation, but overall, performed well.

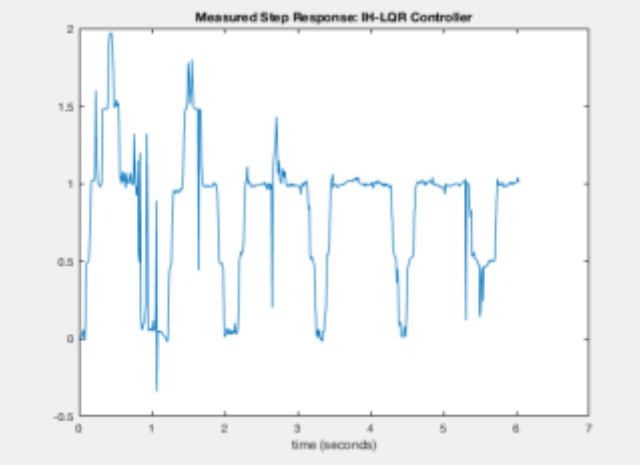

Fig. 47. Real step response to the IH-LQR controller

As shown, the robot's response incurs nearly %100 overshoot in the first half second, but because of the fast rise time, the robot displays stable behavior on the track.

2. Full Track Response

The IH-LQR Controller performs well in both the intersection and the traditional track. Despite a high measured overshoot, the controller's rise time and settling time allow the robot to quickly adjust and handle the course quite smoothly. This controller however does not allow the robot to complete the tracks with much speed, and therefore, is not the best for the robot.

E. Root Locus

1. Step Input Response

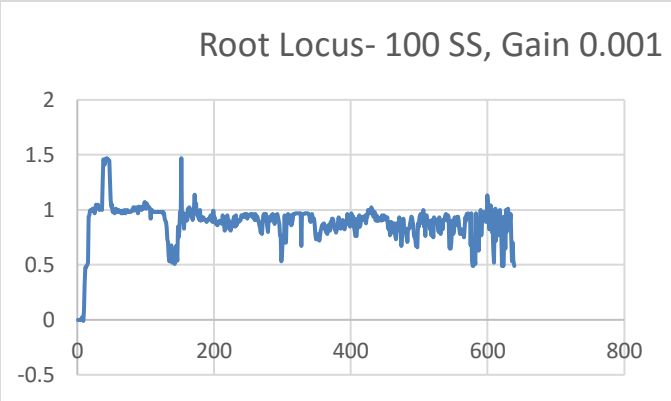

Fig. 48. Real step response to the root locus controller

This is the response to a step input when the machine followed the line. It shows that when it started, it has some peak values to some instance and these overshoot may lead to the bot off track. In a steady state, it will damp among 0.6 to 1 which is not the same result as simulation but close. All of the root locus controllers had similar responses when we tested it.

2. Full Track Response

Though, we can use root locus to finish our track, there are still much of things can be desired. Compare with SISO control, one of our best controller, we found that robot moves comparatively awkward and a little be unstable. The robot cannot finish our track every time and the front of the robot will swing severely so it may be off track sometimes. There are several reasons for that. First, the Kd we applied may not be accurate enough since in the demo we use set speed 100 and Kd equal to 46200. However, any value between 46000 to 56000 may applied in simulation. Second, the overshoot is too high from this case, it is up to 67% which is not desirable. The last reason is in order to make our machine stable, we put an extra gain for all our controller but root locus has a smallest gain 0.001 compare with other may have 0.2 to 5, so the gain indicate how aggressive response may happen if we did not make our gain smaller.

F. Current Observer

After playing around with the numbers for each controller this is the result. Only two of the designed controllers settle near the desired value of 1. The rise time is very desireable and the settling time is also less than 1 second for each of the design controllers. In simulation this controller's response was very good; however in real life the response was much different.

1. Step Input Response

Once converting each of the designed controllers into C ++ the actual responses could be observed.

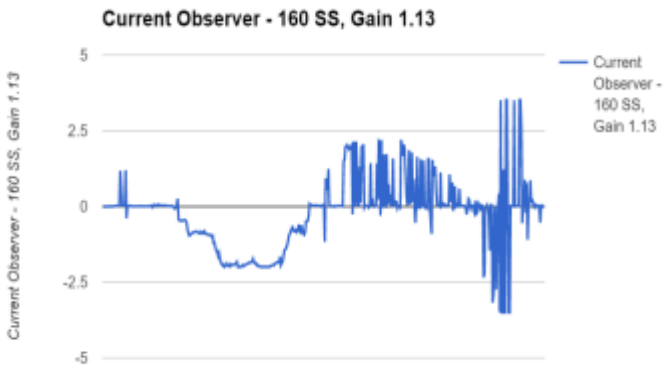

Fig. 49. Real response of the current observer controller

This is the response to a step input when the bot was actually following the line. It is easy to see that the vehicle went unstable pretty quickly and never recovered. All of the current observer controllers developed had similar responses when tested.

2. Full Track Response

When taking the controllers produced by MatLab and deploying them to the Arduino to see the actual performance on the final track the response was obviously not similar to that of the simulation. When testing each controller for the current observer, the vehicle seemed to start to correct to the line but shortly after it would stop correcting then go unstable. On the final track, the vehicle was set and initialized then proceeded to basically drive straight until it was completely off the line.

VI. CONCLUSION

A. Best Controller

Among the six controllers designed by the team, the SISO-Lead controller (Eq. 47) navigated the bot smoother and responded quicker on the full track in addition to a quick step response. Also, it was more reliable than the other controllers because it would behave consistently among many trials. The simulation step measurement represented the measured step response similarly with respect to rise time and settling time.

$$D(z) = \frac{272.9z - 259.2}{z - 0.07} \tag{47}$$

B. Difficulties

1. Obtaining Reliable Transfer Functions

Obtaining the plant transfer function for the vehicle was more difficult than expected. The fact that it was necessary to close the loop of the made it more interesting of a problem than the examples shown in class due to the additional block diagram algebra that needed to be applied. Having to close the loop with a temporary controller of only a gain term allowed the the acquisition of semi reliable data to estimate the plant transfer function with. Using essentially a proportional gain controller and supplying the vehicle with a step input of one centimeter, the location of where the bot is on the line can be recorded. Using this method was tedious and took a very long time to get something that gave a reasonable plant transfer function. Running multiple tests never led to the same data which never gave the same transfer function. This made it very difficult to say which transfer function was the actual one.

2. Reliable Sensor Values

Each sensor array contains eight sensors which each take an analog pin. With the motor driver board shield there are only 6 analog pins available for use so to remedy this issue, two analog MUX's were used. These allowed for the use of the sensors but they added restrictions on the sampling time of the sensors. Having these MUX's in the system added a small amount of uncertainty in the authenticity of the sensor reading because it was hard to tell if it was the actual reading or if the MUX had slightly altered it. Also any glare or random reflection results in bizarre reading that can drastically affect the control of the line.

3. Cheap Motors

The motors use plastic gearing internally which leads to unreliable control. The plastic gearing makes it very easy to break the gears if too much torque is applied. Also, Even though the same voltage and current are applied to both motors, they are responding differently to speed commands that are sent to them. This is mostly due to their cheap nature. The unequal response by the motors is also affecting the performance of the controllers because they are not getting the same response from both sides of the vehicle. Having this sort of issue lead to an oscillation response while the bot followed the track.

4. Track Building

Having the track handmade is both a good and a bad thing due to the way the error and differential values are calculated.

Since the total error, over both sets of sensors, compares the individual errors of each set it becomes very important to have the lines at the exact same distance apart throughout the track. If the lines are not at the same distance apart it is possible that one will read error while the other does not which means the total error will read a value when in reality it actually has no error. Also, originally the track was on a large piece of cardboard but that caused issues with the unevenness of the surface. When using the track on the cardboard, the vehicle would randomly lose where it was on the line because of bumps in the track scraping the surface of the sensors which resulted in bizarre readings.

C. Improvements

Due to the limited budget, the team had to design around cheap parts that behaved inconsistently. The most inconsistent components that could be improved upon were the motors and the sensors. Also, the team designed the controllers around the plant's transfer functions that were obtained with uncertainty due to unreliable sensors.

The motors responses to a certain voltage were not as linear as the team would have hoped for. One motor would typically output a higher RPM to voltage ratio than the other even though both motors had the same specifications. Also, the reliability of sensed input from the QTR sensor arrays was poor. This may have been due to the analog signals from the QTR array going through the analog MUXes. If the signals from the sensors could have been passed to the microprocessor without a middle component it could have improved both the signal reliability and the sampling speed.

If the team could have found certain plant transfer functions that showed consistent behavior for every trial, the simulated design for the controllers of the bot could have resembled the actual response better.

VII. REFERENCES

[1] Charles L. Phillips, H. Troy Nagle, Aranya Chakrabortty, *Digital Control System Analysis & Design*, 4th Edition, Boston, MA, Pearson Prentice Hall, 2015

[2] "System Identification Toolbox Documentation". *Mathworks.com*. N.p., 2016. Web. 1 Dec. 2016.

[3] *Arduino Uno* [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno

[4] "Control System Toolbox Documentation". *Mathworks.com*. N.p., 2016. Web. 2 Dec. 2016.

[5] L293/L293D[Online]: http://www.ti.com/lit/ds/symlink/l293.pdf

[6] J. Pohl and J. Ekmark, "*A Lane Keeping Assist System for Passenger Cars*", Volvo Car Corp., Gothenburg, Sweden, May, 2003.

[7] B. Howard. (2013, September 3). *Extreme Tech* [Online]. Available: http://www.extremetech.com/extreme/165320-what-is-lane-departure-warning-and-how-does-it-work

[8] "Difference Between LDR And Photodiode". *Difference Between*. N.p., 2016. Web. 8 Dec. 2016.

[9] *Arduino Library For The Pololu Reflectance Sensors*. 1st ed. 2016. Web. 8 Dec. 2016.