Department of Mechanical and Industrial Engineering
GINA CODY SCHOOL OF ENGINEERING

MECH 6621 (Microprocessors and Applications)
Final Project Report

Prepared for:

Dr. Brandon Gordon

Prepared by:

Ali Sayedsalehi - ID: 40203268

Hassan Razzaq - ID: 40185719

Mohammad Shamsiani - ID: 40183232

Nazeer Rahim Bhati - ID: 40206086

May 6th, 2022

Contents

1. Optional Tasks

- Replacing Arduino functions in the HIL simulator with register-level programming
- Design of steering controller for autonomous driving down the track without hitting the sides
- Implementation of brake, steering, speed, and traction controllers in the 3D graphics simulator
- Design and testing of the brake, speed, and traction controllers using Simulink

2. Group Members' Responsibilities

Ali Sayedsalehi - ID: 40203268

- Write the function for ADC conversion using register-level programming and no polling
- Carry out everything related to designing the PID controller for steering in order to move the car down the track without hitting the sides
- Implementation of brake, traction, and speed controllers in the 3D graphics simulation
- Writing the project report for the sections mentioned above

Hassan Razzaq - ID: 40185719

- Design and Implementation of traction controller in HIL simulator
- Modification of external interrupt functions using register level programing and replacing Arduino library functions
- Writing the project report for the sections mentioned above

Mohammad Shamsiani - ID: 40183232

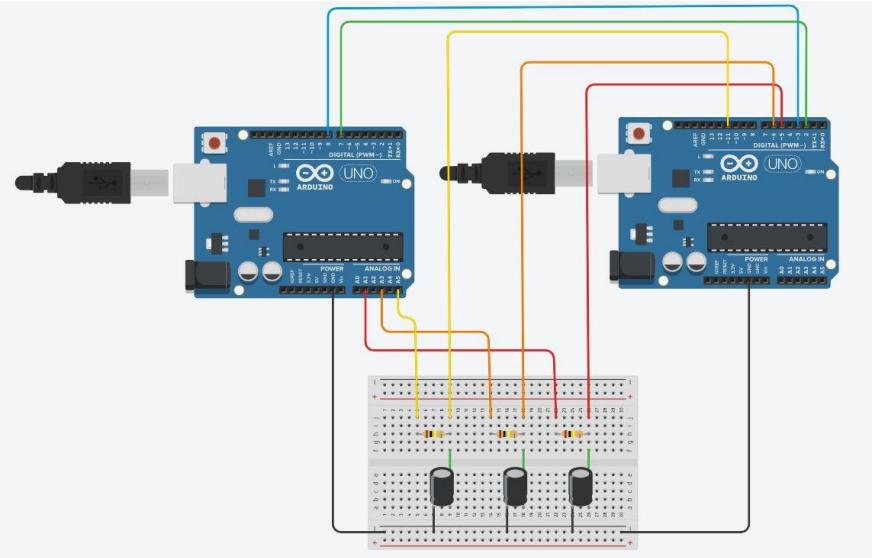
- Constructing Simulink model for tuning the controllers
- Design and implementation of brake controller in HIL simulator
- Writing servo pulse width generator function using register level programing.
- Implementing atomic access on HIL simulator using registers
- Writing the project report for the sections mentioned above

Nazeer Rahim Bhati - ID: 40206086

- Design & analysis of the Speed/Cruise PID controller
- Design and implementation of speed controller in HIL simulator
- Design and implementation of low pass filter (RC- filter) to reduce high-frequency noise
- Defining I/O pins & generate PWM pulses (Digital to Analog Converter), Using register level programming

3. Introduction

A hardware-in-the-loop simulator of a car consisting of two Arduino Uno boards is used to design and test three controllers for braking, traction, and speed. The front and back wheels speeds are used as feedback by the controller, and servo pulses for back wheels' DC motors are used as actuator signals to drive the car. The controllers above are used in combination with a steering controller to autonomously drive a car down the track in the 3D graphics simulation.



4. PID controllers' Gains

For all the controllers in the project, trial and error through experimentation is used to determine the gains. The experimentation is carried out either in the HIL, the 3D graphics simulation, or the Simulink environment.

How to find the three gains Kp, Kd, and Ki:

- First, we start with all gains as zero. Then Kp is increased until overshoot occurs. At this point, controller is unstable.
- Second, Kd is increased to dampen the overshoot and stabilize the controller.
- Third, Ki is increased in order to reduce the steady state error.
- Finally, based on the behavior of the actuator and the change in the magnitude of error, the three gains are changed and optimized using trial and error.

5. HIL

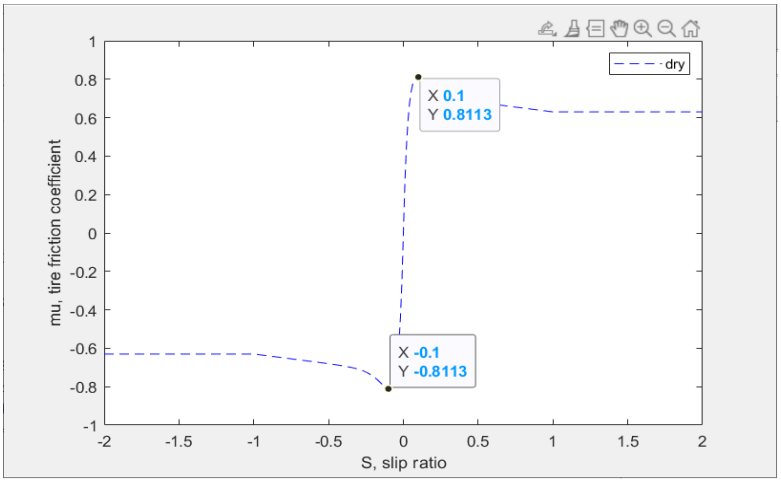
Switching Between Controllers

Since the three HIL simulators act on the same actuator signal, namely, motor voltage, there needs to be a strategy to switch between them. If the actual speed is up to 10% more or less than the desired speed (desired speed bound), the speed controller will be active. If The actual speed is more than the desired speed bound, braking controller will be active. If the actual speed is less than the desired speed bound, traction controller will be active.

```
if  ( (wb <=  w_des + 0.1 * w_des) && (wb >= w_des - 0.1 * w_des)){
       u1 = speed_PID(wb , w_des , dt); // designed and tunned for maintaining the
                                        // speed or dealing with small disturbances
                                        // in the interval of 10% deviation around
                                        // desired speed.
}else if (wb > w_des + 0.1 * w_des){
       u1 = brake_controller( dt , sr , w_des );// desined to keep slip ratio
                                                // at the lowest amount to generate minimum
                                                // friction coefficient in case of braking.

}else if ( wb < w_des - 0.1 * w_des){
       u1 = slip_controller( dt , sr , w_des ); // desined to keep slip ratio
                                                // at the highest amount to generate maximum
                                                // friction coefficient in case of acceleration.
}
```

Slip Ratio

Traction and brake controllers are designed to reach two different determined slip ratios corresponding to the maximum and minimum friction coefficient, to generate the highest friction force with a negative or positive sign in case of deceleration or acceleration. The BW model is used to calculate the friction coefficient. As illustrated on the right, brake and traction controllers need to reach -0.1 and 0.1 slip ratios to generate the desired absolute value of maximum friction coefficients (approximately 0.8).



RC Low Pass Filter

To design the RC low pass filter, an online tool [2] is used. After a process of trial and error, the following values and results were obtained:

Pin number	PWM frequency (Hz)	Cut-off frequency (Hz)	R (Kilo Ohm)	C (micro farad)	Peak-to-peak ripple voltage (V)	Settling time (sec)
5	980	1.59	100	1	0.013	0.23
6	980	1.59	100	1	0.013	0.23
11	490	0.72	100	2.2	0.011	0.5

It should be noted that although a lower cut-off frequency would yield better signal accuracy, the settling time will rise as a result of a lower cut-off frequency. For our application, which is wheel tachometer, a settling time of 0.23 sec is not so high as to cause problems; Furthermore, a peak-to-peak ripple voltage of 0.013 V gives enough accuracy as to reasonably read the tachometer values.

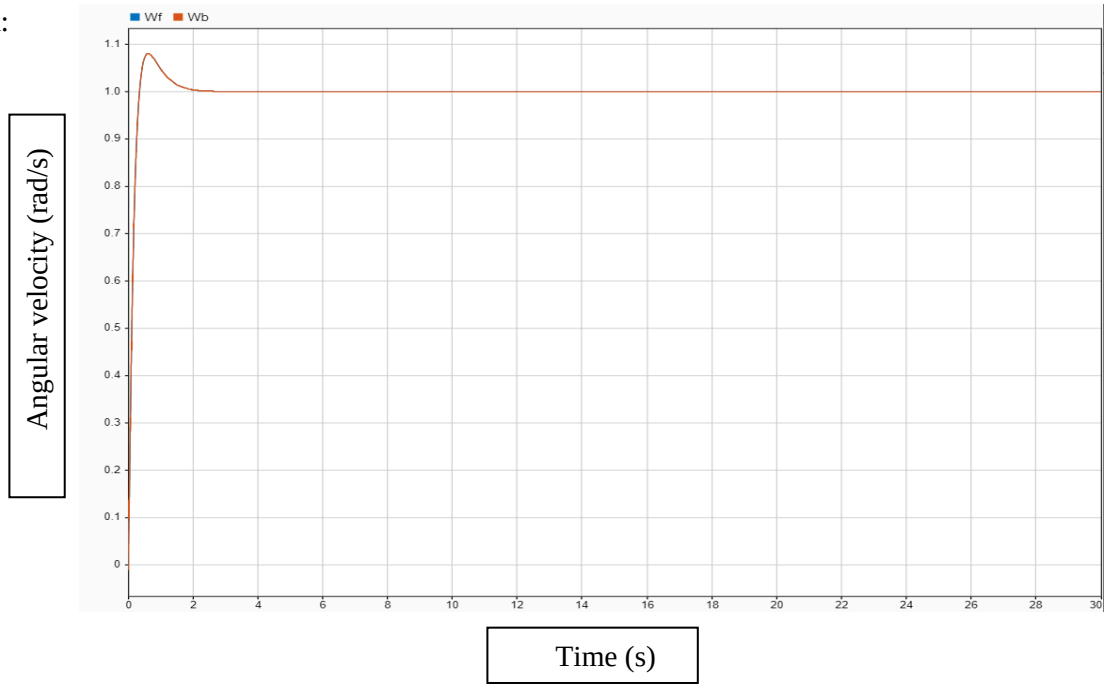
6. Results

In the following figures, results from the HIL simulator are compared with those of the Simulink simulation.

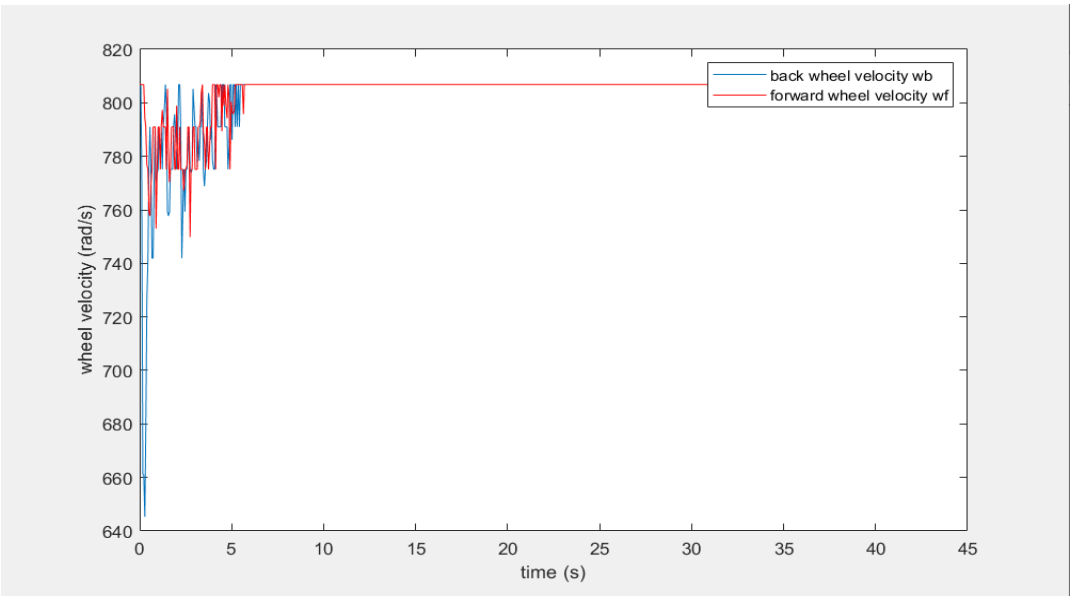
Speed Controller

The following figures shows how angular velocity changes with respect to time:

Simulink:



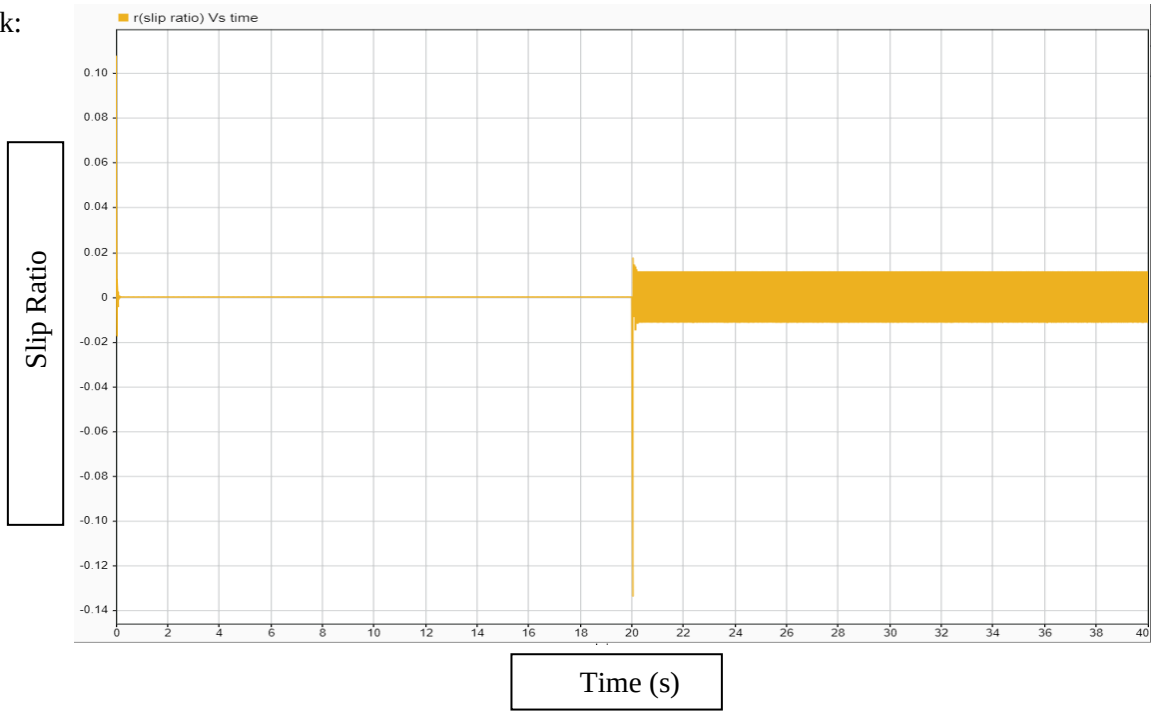
HIL:



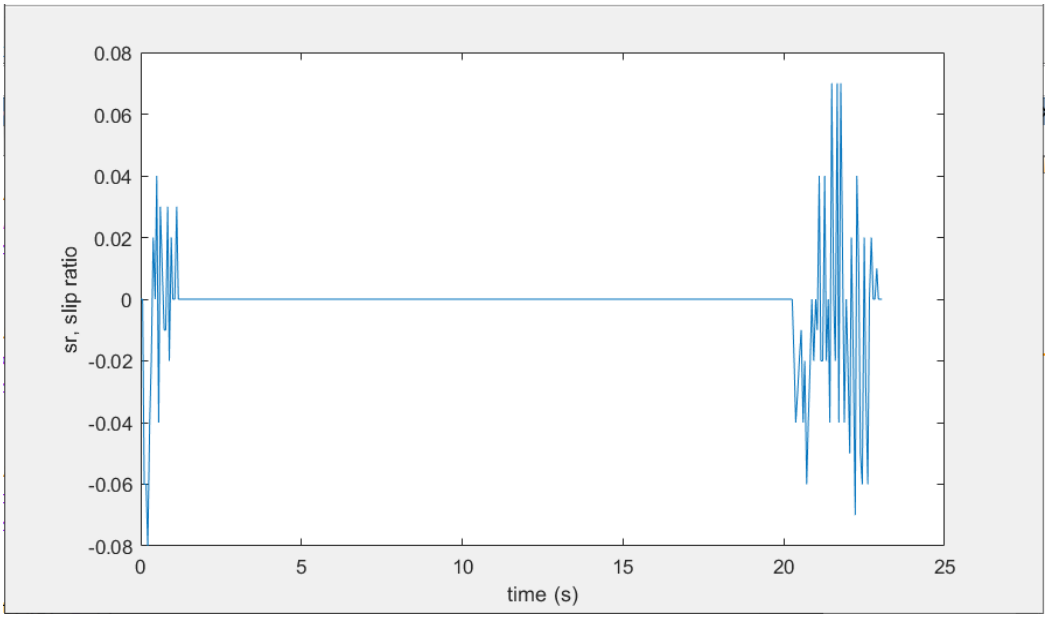
Traction and Brake Controllers

In order to test these controllers, an experiment is conducted. At time = 0 sec, the traction controller is activated because the speed of the car is set to its maximum. At time = 20 sec, the braking controller is activated because the speed of the car is set to zero. The following figures show the slip ratio with respect to time:

Simulink:

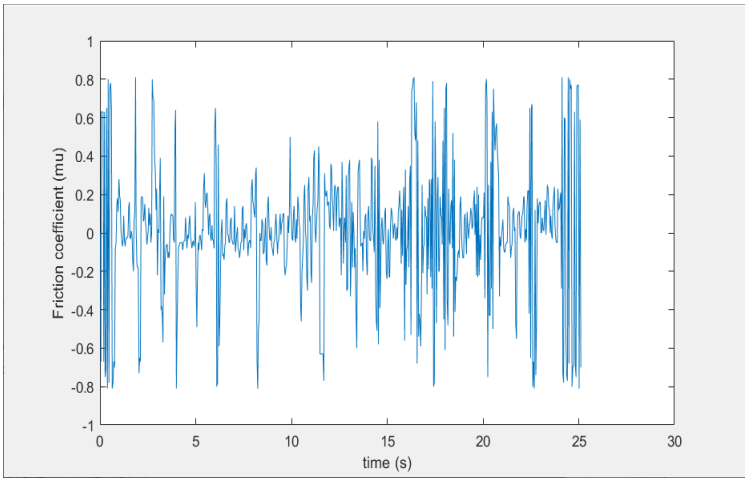


HIL:



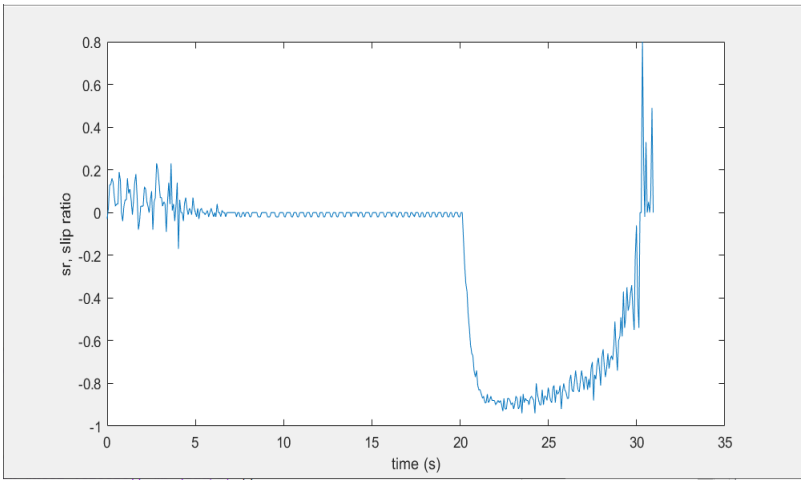
As can be seen from the figures above, the slip ratio is around 0.1 at the beginning (traction) and around -0.1 at the 20 seconds mark (braking), which is what we expect.

The figure on the right demonstrates how the friction coefficient changes for the same experiment mentioned above. It can be seen that the controllers keep the friction coefficient between -0.8 and 0.8.



ON-OFF controller

An on-off controller was tested for the traction and brake controllers. The figure below shows its performance for the same experiment mentioned above. As it can be seen, the slip ratio values fluctuate around the desired values, namely, 0.1 and -0.1, which makes this controller unsuitable for our application.



On-Off Controller

7. Register Level Programing

Analog to Digital Conversion (ADC)

The objective is to develop an ADC code using a register-level program. Instead of polling to find out when the ADC conversion is complete, this ADC code uses the ADC conversion complete interrupt function. The program then calculates the average of the simulator Arduino's PWM output signals while it is waiting for the ADC conversion to complete. Using an if-else statement, we determine the pin and set the appropriate pin bits to activate the channel. Each time we go to the interrupt function, we set a Boolean variable to true and start a new ADC conversion. When another ADC conversion is complete, the program then jumps into the interrupt function again and executes the mentioned steps all over again.

Servo pulse write function

This function uses timer1 because timer1 is a 16-bit timer and therefore is able to record more time in the form of an integer. This means that a pulse with a maximum width of 65535 can be produced, which when multiplied by the prescaler value becomes the duration of the pulse width. In this function, using compare match interrupt, we set the desired pins to the maximum voltage (5 volts pullup resistor) for a desired period. Then, in the overflow interrupt function, we turn off the pin (i.e. zero voltage).

PWM– Digital to Analog Conversion

```
void DAC_write(int PIN, int output){

    if (PIN == Y1_PIN) OCR0B = output;// pin 5
    if (PIN == Y2_PIN) OCR0A = output;// pin 6
    if (PIN == Y3_PIN) OCR2A = output;// pin 11

}
```

In this function, pins 5, 6 and 11 are first defined as simulator outputs. Timer zero is used for pins 5 and 6 and timer 2 is used for pin 11. Because the 0 and 2 timers are 8-bit, they can record a maximum of 255 integers. Since the number 255 is divisible by 5 by 51, the value of the analog voltage multiplied by the number 51 determines the value of the voltage input. This multiplication value in the prescaler determines the duration of the pulse width with the highest voltage, i.e. (5 volts). This pulse signal is converted to analog voltage output between 0 and 5 volts using the PWM technique and sent to the analog pins of the controller.

External Interrupts for pins 2&3 (simulator)

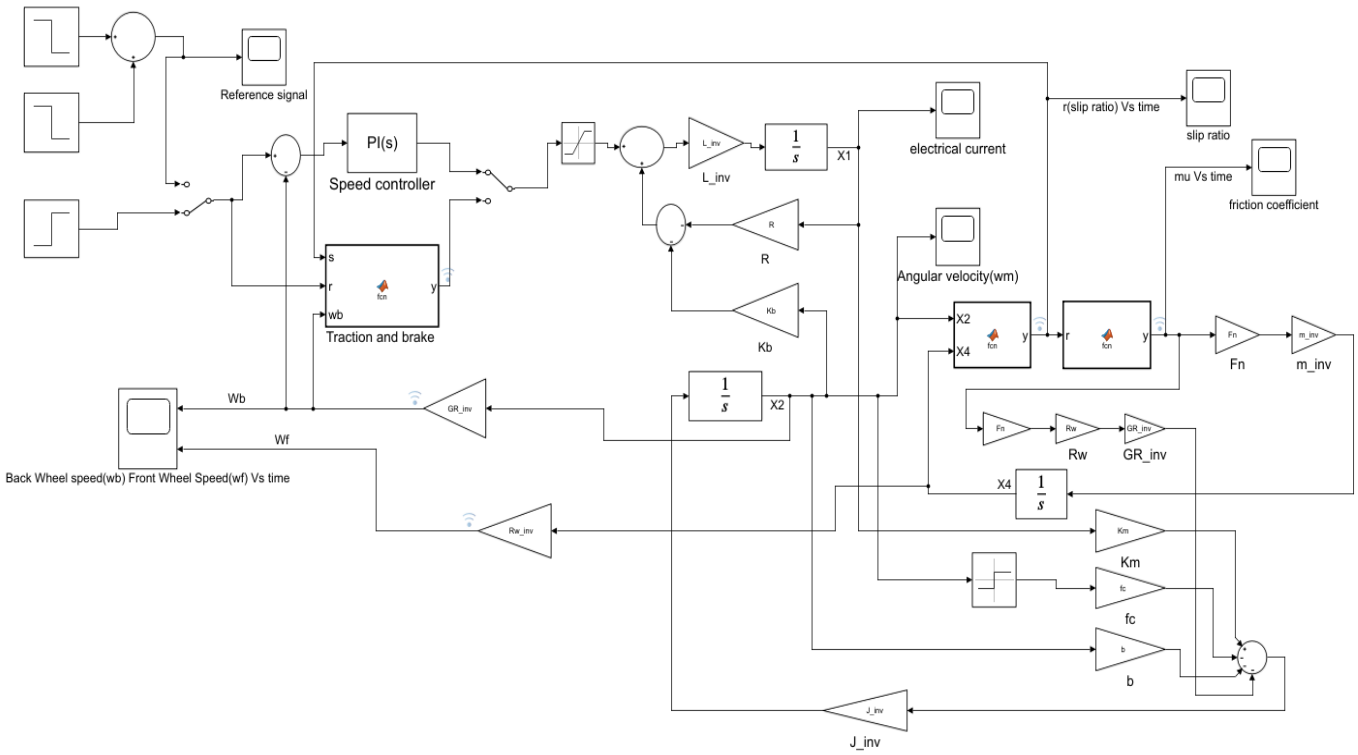
In this function, the signals generated by the servo pulse function in the controller are received by the simulator. In order to not spend too much time to receive the signals from the controller, this function and its related settings are such that only if the voltages of the input pins change, the simulation stops, and in the interrupt function, the value of the input integer is read, which is multiplied by the prescaler. In this function, time is recorded at the moment of voltage increase; Then, pulse width is calculated by calculating the period during which voltage is set to HIGH. The current voltage is then stored until the next call. The main purpose of this function is to avoid delay in the program and interrupting the program unless a change in the input voltage takes place.

Atomic Access

In the simulator, when we are assigning the servo pulse values to appropriate variables, atomic access is used to avoid interrupting the program in the middle of value assignments.

8. Simulink Model

Simulink is used to model the mathematical system equations. Here, brake, traction, and speed controllers are modelled. This model helps us optimize the controllers more accurately and verify their performance in a separate environment. The figure below shows the block diagram model.



9. Steering Controller

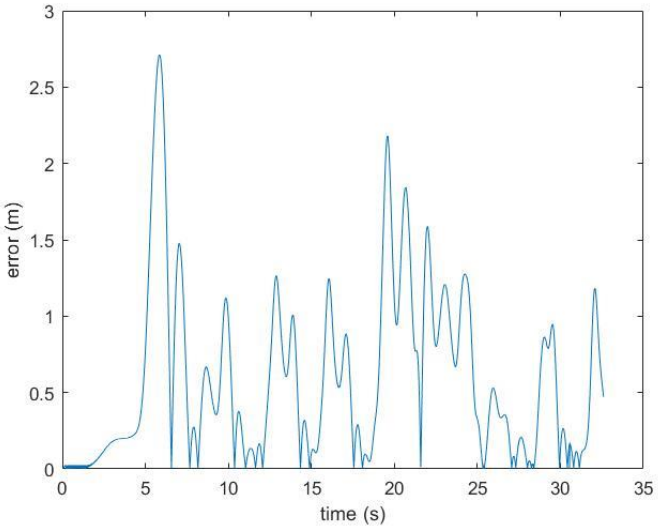
As the car moves along the track, it needs to minimize its distance from the middle of the road. If the distance between the car and the middle of the road increases, the steering angle increases in order to return the car to the proper position. For this strategy, the error used in the PID controller would be the minimum distance from the car to the middle of the road. The controller output would be the steering angle.

We first put the x and y coordinates of all the points on the track into two 1D arrays. It is worth noting that in order to have a stable start, we define the initial location of the car as the first point on the track. The next step would be to find the distance between the car and the closest point on the track to the car. We then need to find out when the steering angle should be negative and when it should be positive. To do this, we first determine on which side of the road the car is, i.e., is it on the right side of the middle line of the road or on the left side of the middle line. Here, the formula for determining the sign of the determinant of two vectors is used. For instance, for vectors $\overrightarrow{AB}$ and $\overrightarrow{AM}$, where M (X, Y) is the query point: position = sign ((Bx - Ax) * (Y - Ay) - (By - Ay) * (X - Ax))

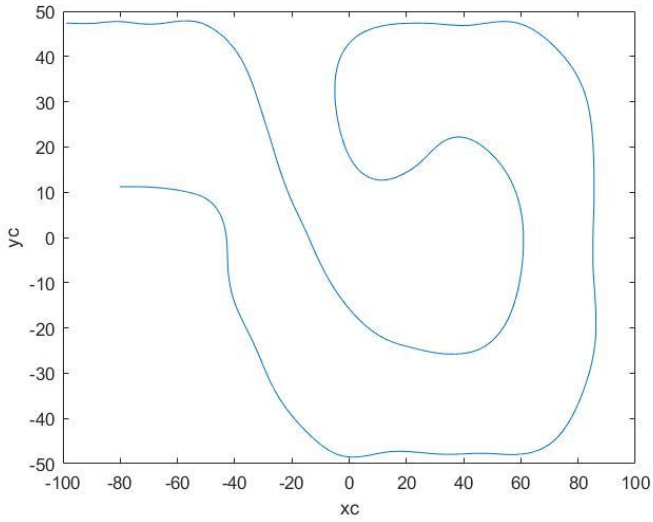
This equation gives 0 when the point is on the line, and gives 1 or -1 if the point is on one side or the other. In our case, A is the closest point on the track to the car; B is the point right in front of it which is also on the track; M is the location of the car. Later on, we multiply the roadside number by the absolute value of the steering angle to get the final steering angle.

```
        /// Find out on which side of the road the car is **********************
        double road_side = 0;
        double position = 0;

        // sign of the determinant of vectors AB and AM,
        // where M(xc, yc) is the location of the car.
        // A(x_closest, y_closest) and B(x_front, y_front)
        position = (x_front - x_closest) * (yc - y_closest) - (y_front - y_closest) * (xc -
x_closest);

        if (position > 0) {
                road_side = -1; // car is on the left side of the road
        } else if (position < 0) {
                road_side = 1; // car is on the right side of the road
        } else if (position == 0) {
                road_side = 0; // car is on the road
        }
```

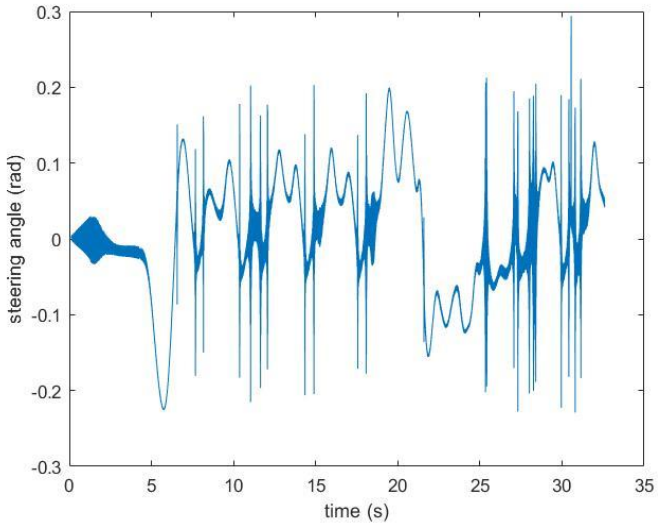
The calculated PID gains are: Kp= 0.08, Kd=0.01, and Ki=0.001. It is worth noting that from a simulation point of view, it is possible to further optimize the gains in order to reduce error; However, this approach causes very fast twitching of the wheels which is physically impossible, because the wheels and the steering system have inertia. If further error reduction is desired, the strategy for the steering controller should be in a way as to predict the road's curvature, as opposed to responding to it. That way, the controller would be more sophisticated and more complicated. Nevertheless, the current approach works very well.



Error (min distance from the car to the middle of the road) with respect to time



Car's location on the track based on its x and y coordinates

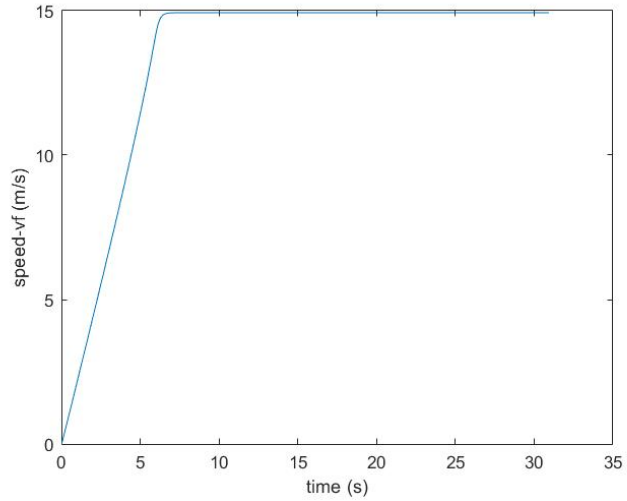


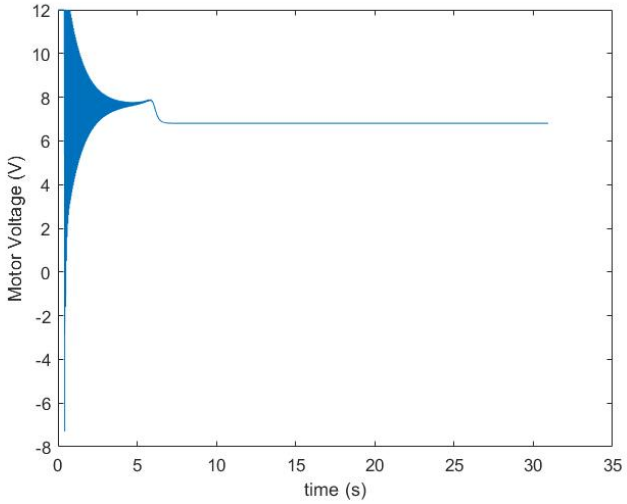
Steering angle (rad) with respect to time

10. Implementation of All the Controllers in the 3D Graphics Simulator

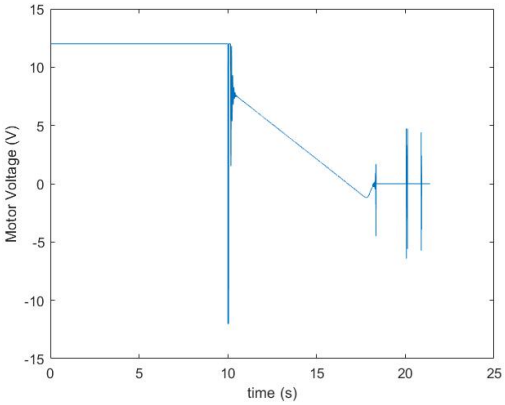
First, each controller is implemented separately and its response is tested. Then, all of the controllers are combined using an if-statement. Braking is only executed when the car is facing a sharp turn. To implement this, first the angle between the car's direction of movement and the road is calculated, using the vectors tangent to the road and the car's trajectory. If the angle is more than 0.2 radians, the braking controller is executed. Moreover, in order to simulate manual pressing of the brake pedal, a KEY() function is utilized; When B is pressed on the keyboard, the car brakes and eventually stops. The traction controller is only executed when the car has to reach a certain speed from a stationary condition or from a much lower speed (less than 10 percent of the desired speed). In all other instance, the speed controller is active to maintain the car's speed. The following plots are the results of individually testing the brake, speed, and traction.

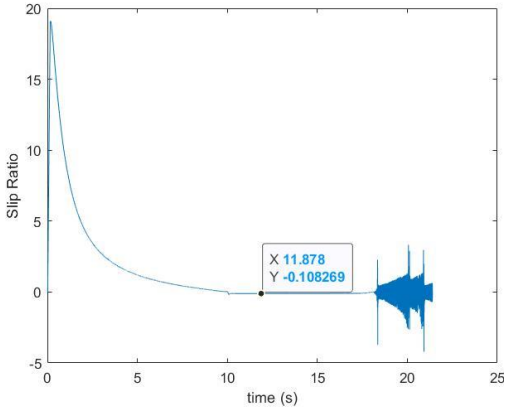
Speed controller for a set speed of 15 m/s (figures below):

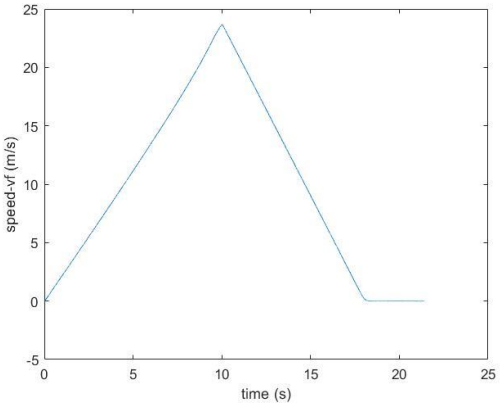




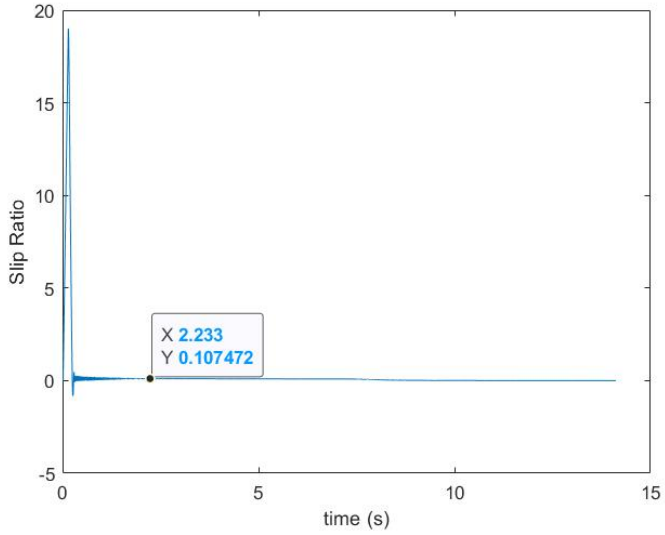
Brake controller (figures below): (The car goes with maximum motor voltage and brakes at time = 10 s until it fully stops)

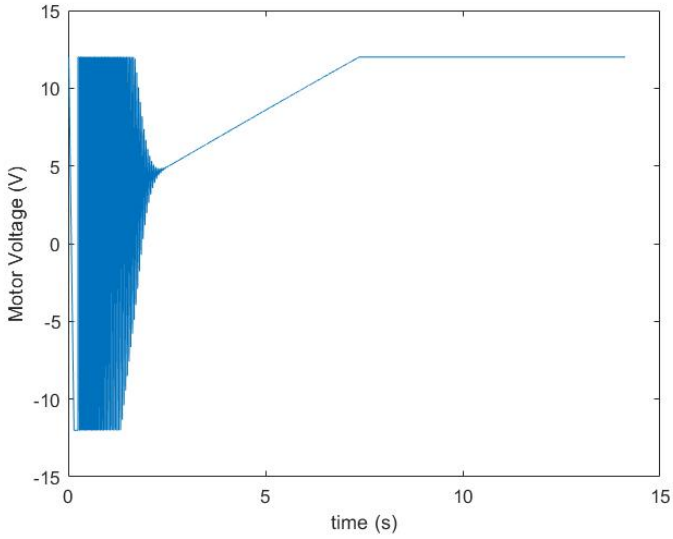




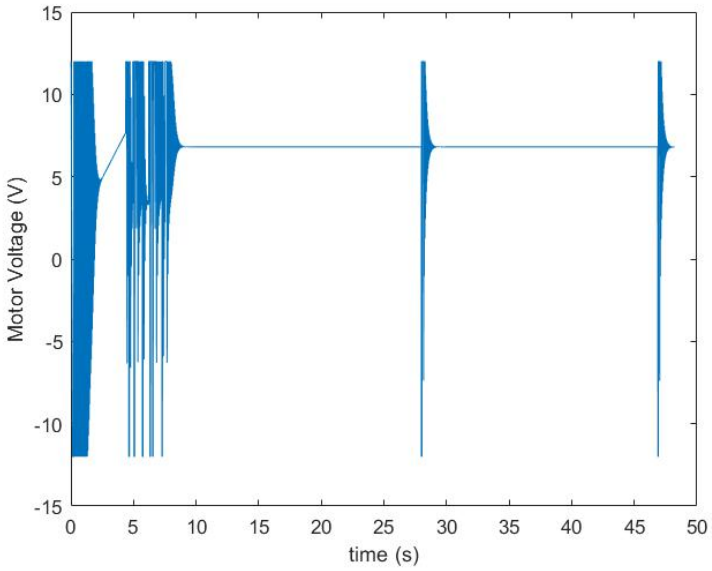


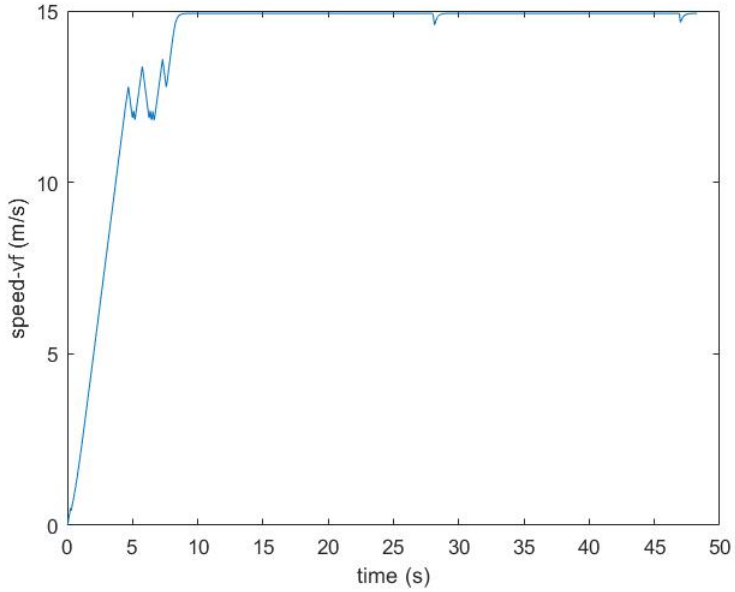
Traction controller (figures bellow): (The car accelerates to reach 15 m/s as quickly as possible)





Combination of all the controllers: First, the car accelerates as fast as possible. Then, it tries to maintain a speed of 15 m/s. At the same time, it brakes when faces very sharp turns. The figures below demonstrate the change in parameters:





11. References

[1] Lecture materials

[2] http://sim.okawa-denshi.jp/en/PWMtool.php