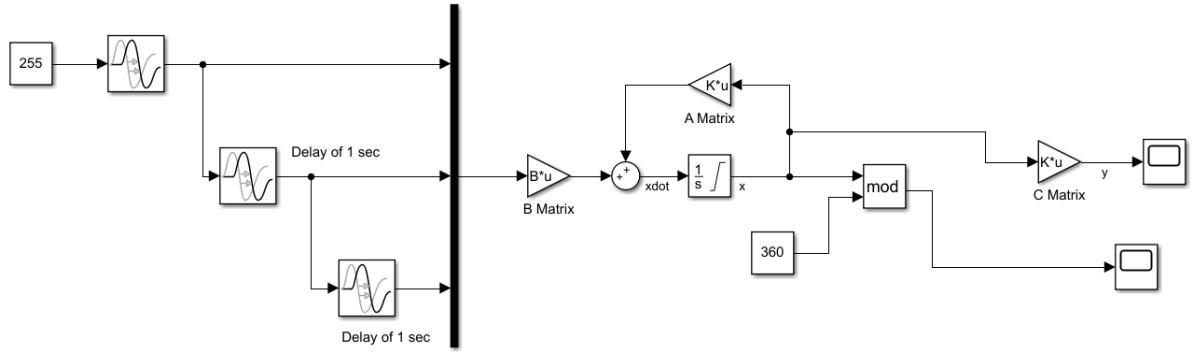
Susan Zhao
ESE 505 – "Smart Highway"

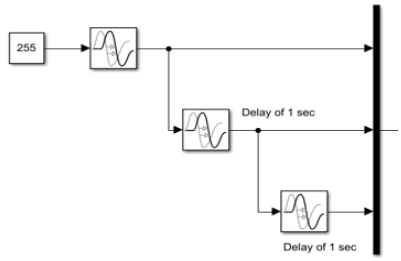
DYNAMIC MODEL

a. The following Simulink model was created as a dynamic model of the train.



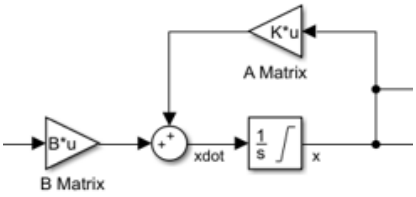
To break down the sections of this model, the sections are separated into input, dynamics, and output.

INPUT



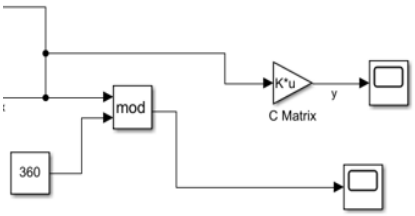
In a real-life example, the cars will each start at a different position. To develop this model similarly to this real-life example, delays are placed after the initial 255 PWM input signal. By the moment the third car begins, the first and second car will be in a different position along the track. This is as expected in a real-life application, as the cars cannot start at the same position on the track.

DYNAMICS



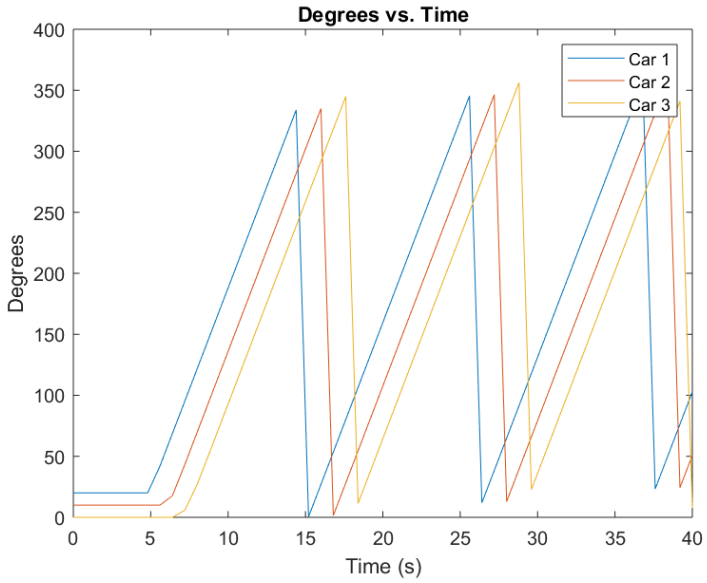
The values of the matrix are presented in the Matlab code [1]. The motor was assumed to have a constant K of 1900rpm for 9V. I assumed linearity and scaled this constant for a 5V application.

OUTPUT

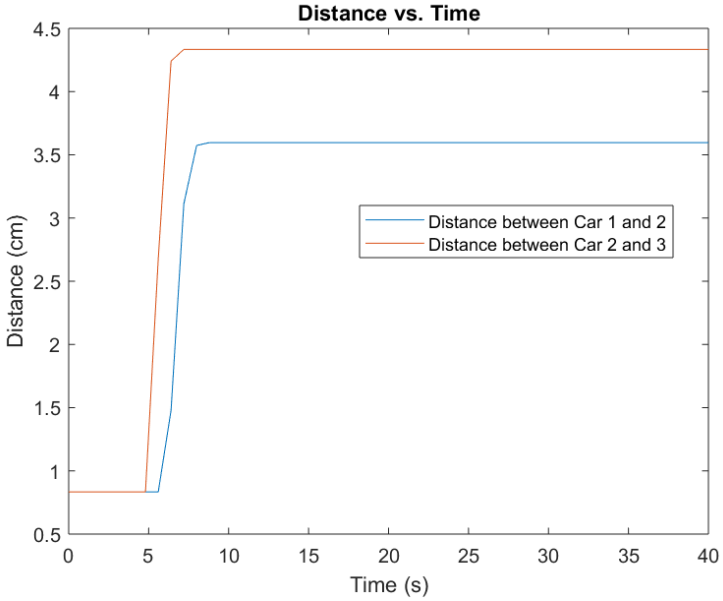


In this lab, the track is a circle. The output values are in degrees, with 0/360 degrees as the top center point of the track. To maintain this range, the output from the dynamic model goes through the mod math function box and fed into a scope to produce Output Graph 1. Output Graph 2 depicts the track distance (cm) between each car.

OUTPUT GRAPH 1

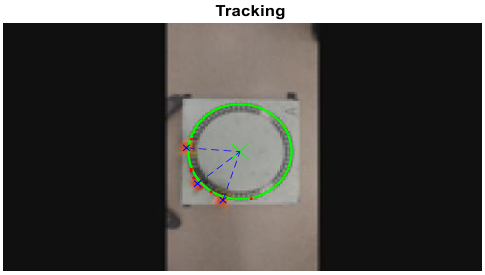


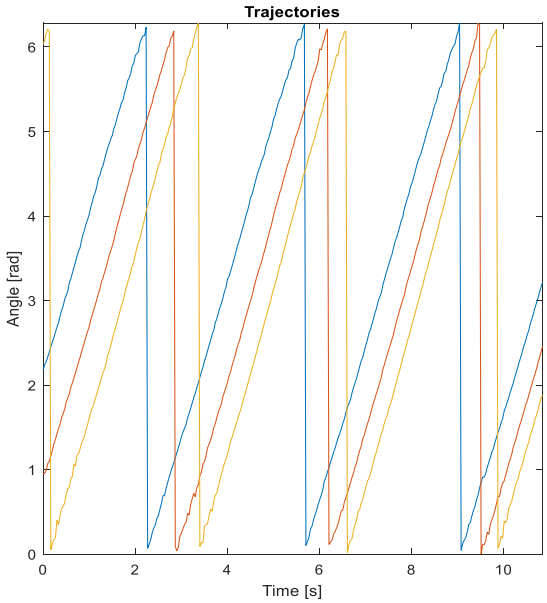
OUTPUT GRAPH 2



In Graph 2, the distance ends up being constant after all the time delays due to the constant and equal speed (PWM signal of 255) between each traincar. There is some confusion concerning this plot, as the distance between the cars differ by 1cm, while I expected them to be equal. This may be attributed to the delays.

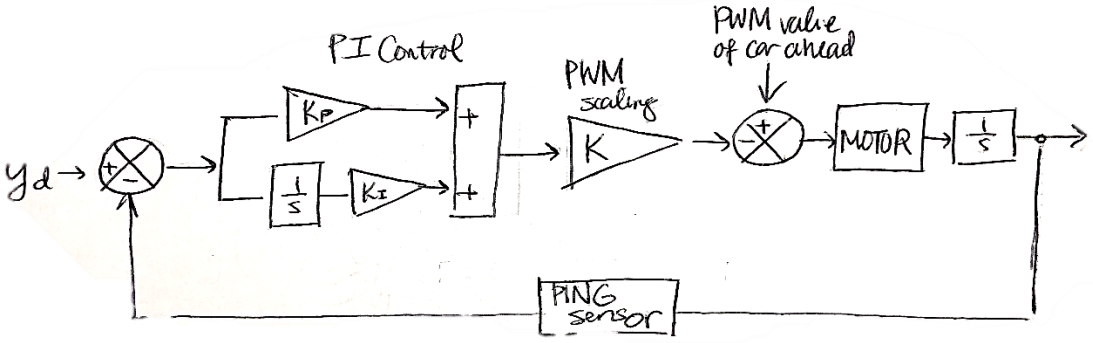
Data gathered from lab





The graph on the right was created using Jake Welde's *process.m* matlab script. The video processed was one of our tests using three cars moving at constant, equal speeds. This output is similar to the output of the dynamic model. Key characteristics are the linear slopes representing constant velocity, the angle vs. time relationship most easily seen by the shape of the graph, and the constant even spacing between the outputs which represents the spacing between the cars.

BLOCK DIAGRAM

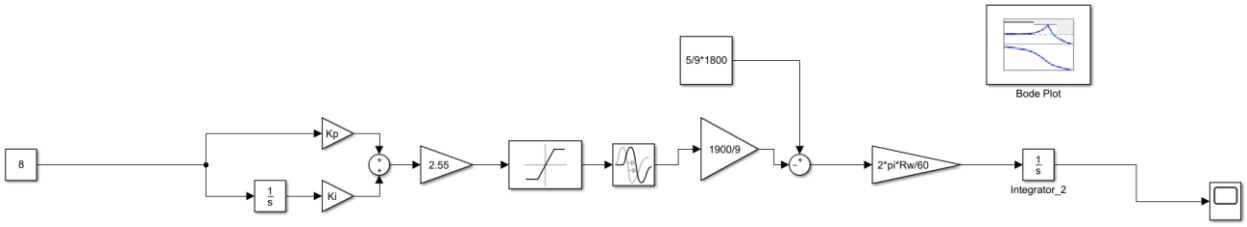


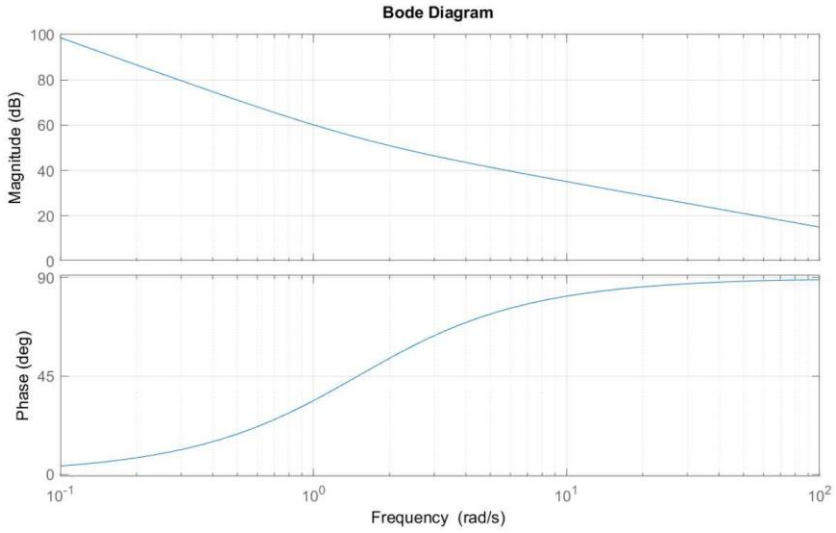
Final version of the *train_control_law* function is in Appendix Section [2].

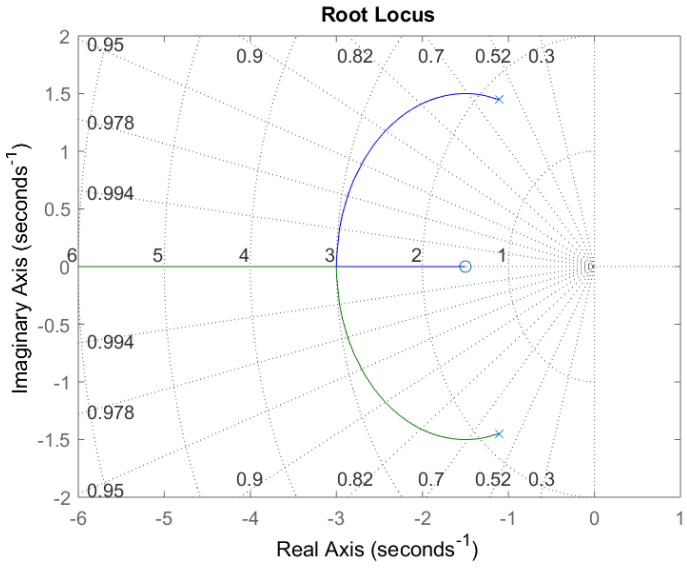
b. This controller utilized position control to attain a desired distance y_d away from any car in front of it. The error is fed into a PI controller with gains of Ki = 15 and Kp = 10. These values were chosen to follow the lead car with zero steady-state error and enable a high enough proportion gain to account for error within an acceptable time. The result is summed up and scaled into a PWM byte constrained to a range of 0 to 255. This value is compared against the car in front, which the controller

assumes is a car moving at a constant speed. This is applied to the motor, which outputs a velocity. The integrator block utilizes the data collection time delay to produce a position calculation. The controller assumes the Ping Sensor to be a perfect 1:1 sensor, and this looped back to be compared to the desired 8cm in a closed loop.

BODE PLOT AND ROOT LOCUS

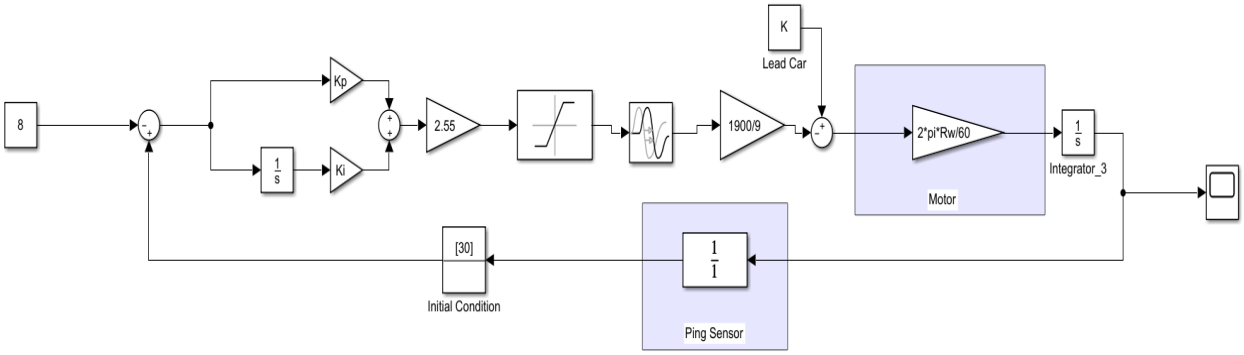






c. Above is the open-loop frequency response of the controller. For frequencies less than 1Hz, the response begins at 100dB which translates to an output that is 100000 greater than the input. The phase is near 0 degrees, meaning that the output will not lag behind the input. As the frequency increases, the magnitude response decreases while the phase response shows the output lags further and further, reaching 90 degrees out of phase. For the application of this lab, it is probably best that the lag is reduced as best as possible, therefore favoring smaller frequencies.

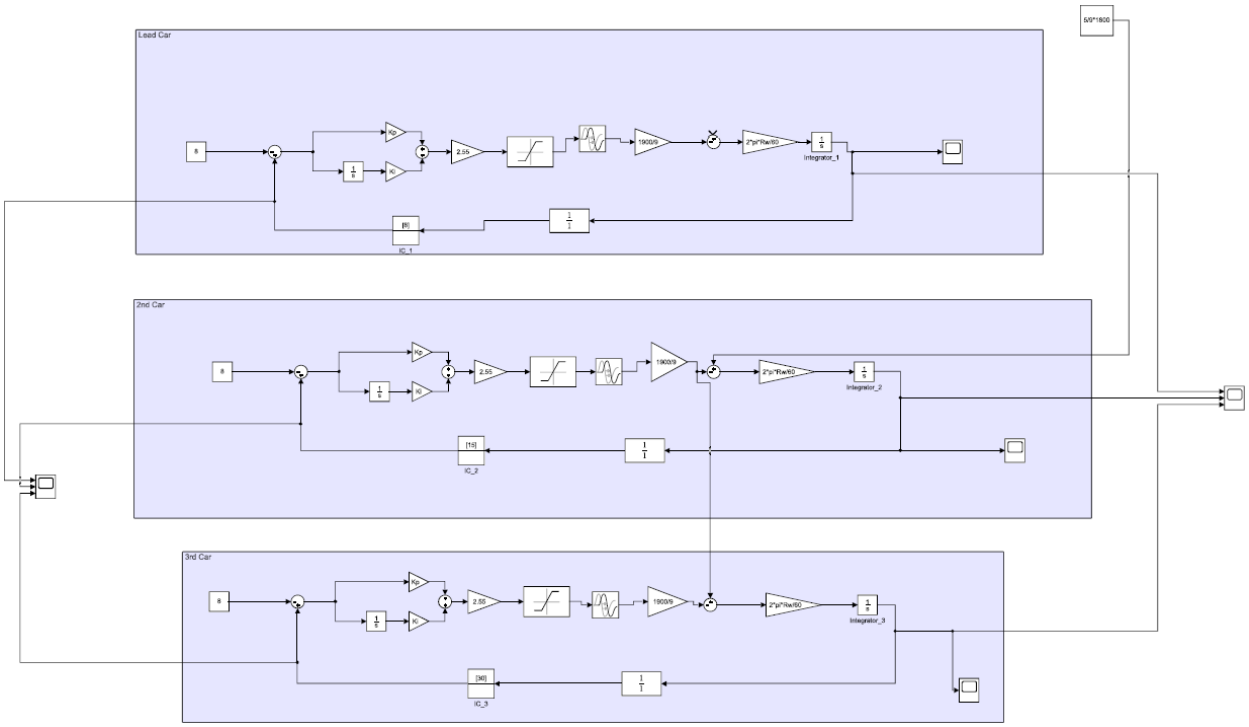
SINGLE-VEHICLE SYSTEM

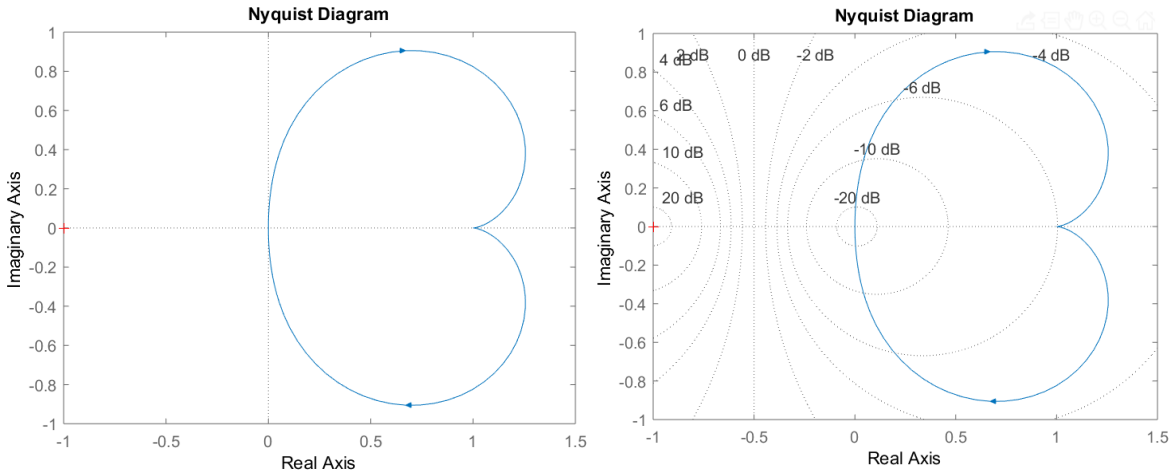


d. Above is a Simulink model of the single-vehicle system. The results of this model are pitted against the desired compliance criteria:

1) *Lead car at constant speed, follow with zero steady-state error*: The lead car was set to run at a constant speed. Using integral control, the following cars can follow with zero steady-state error
2) *Vary desired following distance in proportion to the steady control*: This was not implemented into this simulink model, but I see this as easily controllable inside the code. Currently, the desired distance and the steady speed are input as constants. They can be linked such that the distance is in proportion to the steady control and implemented within the code.
3) *Bandwidth for tracking speed changes of at least 1 rad/sec*: The delay in this system was minimal, where the resulting bandwidth was at least 1 rad/sec.
4) *Stability margins of at least 6dB and 60 degrees of phase margin*: As the phase does not cross the -180 degrees line, this compliance is not attained, as GM and PM are inconclusive. As seen in the root locus plot, the system is stable and has roots to the left of the y-axis, however it cannot be concluded that there are stability margins of at least 6dB and 60 degrees of phase margin.
5) *Emergency stopping: 0% when distance is less than 5cm*: This was implemented in the code [2], where if any signal received by the Ping sensor was read in to be less than 5cm, the output PWM signal was immediately set as 0 -leading to an emergency stop.

MULTI-VEHICLE





e. *Multi-vehicle string stability requirement*: The Nyquist Plot uses the plot of the Open-Loop transfer function G(s) and draws conclusions about the closed-loop T(s)=G(s)/(1+G(s)). The Nyquist plot does not encircle -1 + j0, therefore the system is stable.

f. **Link, UPenn ESE505 - Trains Lab**: https://youtu.be/iZ2CZPi2Xqs

APPENDIX

[1] Part a) Matlab Script

```
close all
clear all
R = 30; %radius of track in cm
K = (5 / 255) * 1900 * 2 * pi / 60; % calculation from pwm to rpm
B = [K/R 0 0; 0 K/R 0; 0 0 K/R]; % b matrix
u_mat = [255; 255; 255]; %input matrix (PWM values)
```

[2] Train_control_law function

```
byte train_control_law(double y)
{
  byte train_control_pwm;
  double KP = 10;
  double KI = 15;
  static double i_control = 60.0;
  double err = 0.0;
  double i_int = 0.0;
  double yd = 8.0;

  if (y <= 5) {
    return 0;
  }
  err = y - yd;
  i_int = KI*err*deltaT_ms*0.001;

  //Cap integral control
  if ((i_control >= 100  && i_int < 0) || (i_control <= -100 && i_int > 0) ||
(i_control <= 100 && i_control >= -100)) {
    i_control += i_int;
  }

double train_control_percent = i_control + KP * err;
train_control_pwm = constrain(2.55 * train_control_percent,0,255); // scale
percent into PWM byte
  return train_control_pwm;
}
```

[3] Part c) Open-Loop Gain Values

```
Kp = 10;
Ki = 15;
Rw= 1;
```

[4] Part e) Nyquist Plot Matlab Script

```
K = 1900/9;
Kp = 10;
Ki = 15;
B = 2*pi*0.01/60;
G = tf([0 K*Kp K*Ki],[1/B K*Kp K*Ki]);
nyquist(G)
```