Adaptive Cruise Control System using MATLAB

Report by By Vahin Reddy, B.Tech Mechatronics – VIth Sem

Introduction

Matrix Laboratory, or MATLAB for short, is a programming language and tool that helps solve complex mathematical problems. Simulink, which comes with MATLAB, is a Model-Based Design system that supports system-level design, simulation, auto-code generation, and creation of embedded system logics. Using the block-based approach of Simulink, we can simplify the programs used to create various system models. With MATLAB and Simulink, we can model various control systems that we come across on our day-to-day basis. Adaptive cruise control system is among these control systems that is currently being implemented on most modern vehicles. We can use MATLAB/Simulink to simulate the adaptive cruise control system used in vehicles.

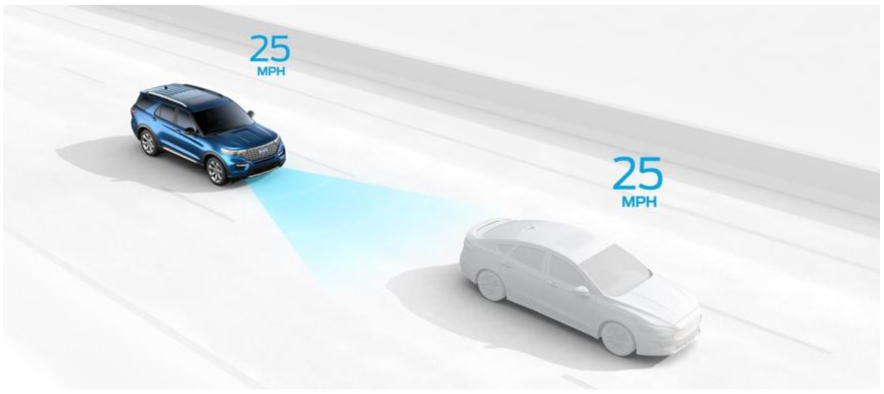


Figure 1: Adaptive Cruise Control used in Ford [1]

To create the adaptive cruise control system, we will be using elements from the Model Predictive Control Toolbox. This toolbox offers various controllers and control systems that can be implemented in vehicle models. For our analysis using the given data, we will be using the Model Predictive Control – Adaptive Cruise Control Example provided by Math Works [2] The example can be opened using the command:

- openExample('mpc/AdaptiveCruiseControlExample')

Procedure and parameters

The subjects of this model will be the Ego Car which will contain the adaptive cruise control system and the Lead Car that will be driving ahead. The adaptive cruise control system will consider:

- *Speed Control* – driving along at a set speed
- *Spacing Control* – maintain a safe distance from the lead car

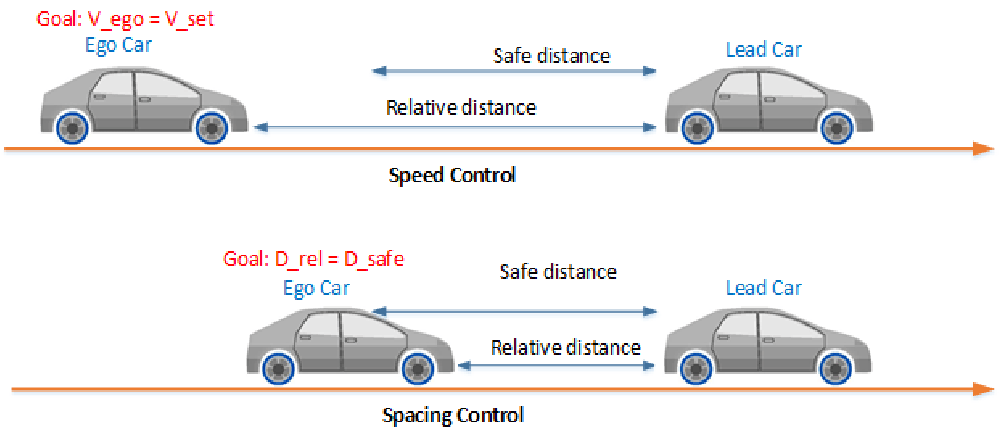
As the Ego Car drives behind the Lead Car, it will consider one of the controls mentioned above based on the distance obtained between the cars. If the:

⇒ Ego Car is far from Lead Car, then Ego Car will travel at set speed – *Speed Control(V_{set}).*

$$D_{rel} \geq D_{safe}$$

⇒ Ego Car approaches the Lead Car, then Ego Car will maintain distance – *Spacing Control(D_{safe}).*

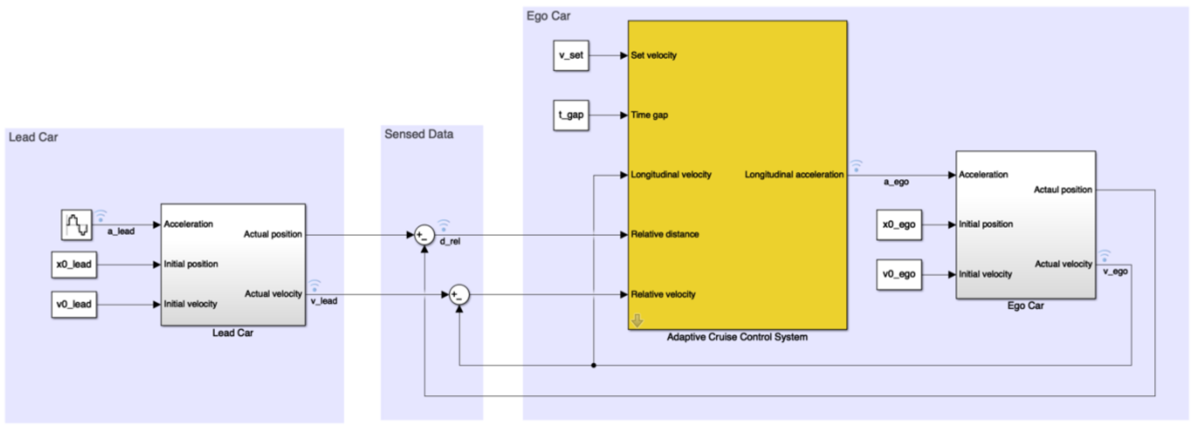
$$D_{rel} < D_{safe}$$



The given parameters are:

- Ego speed: 80 Kmph
- Safe Distance: 120 meters
- Time Gap: 120 sec

Simulink Model

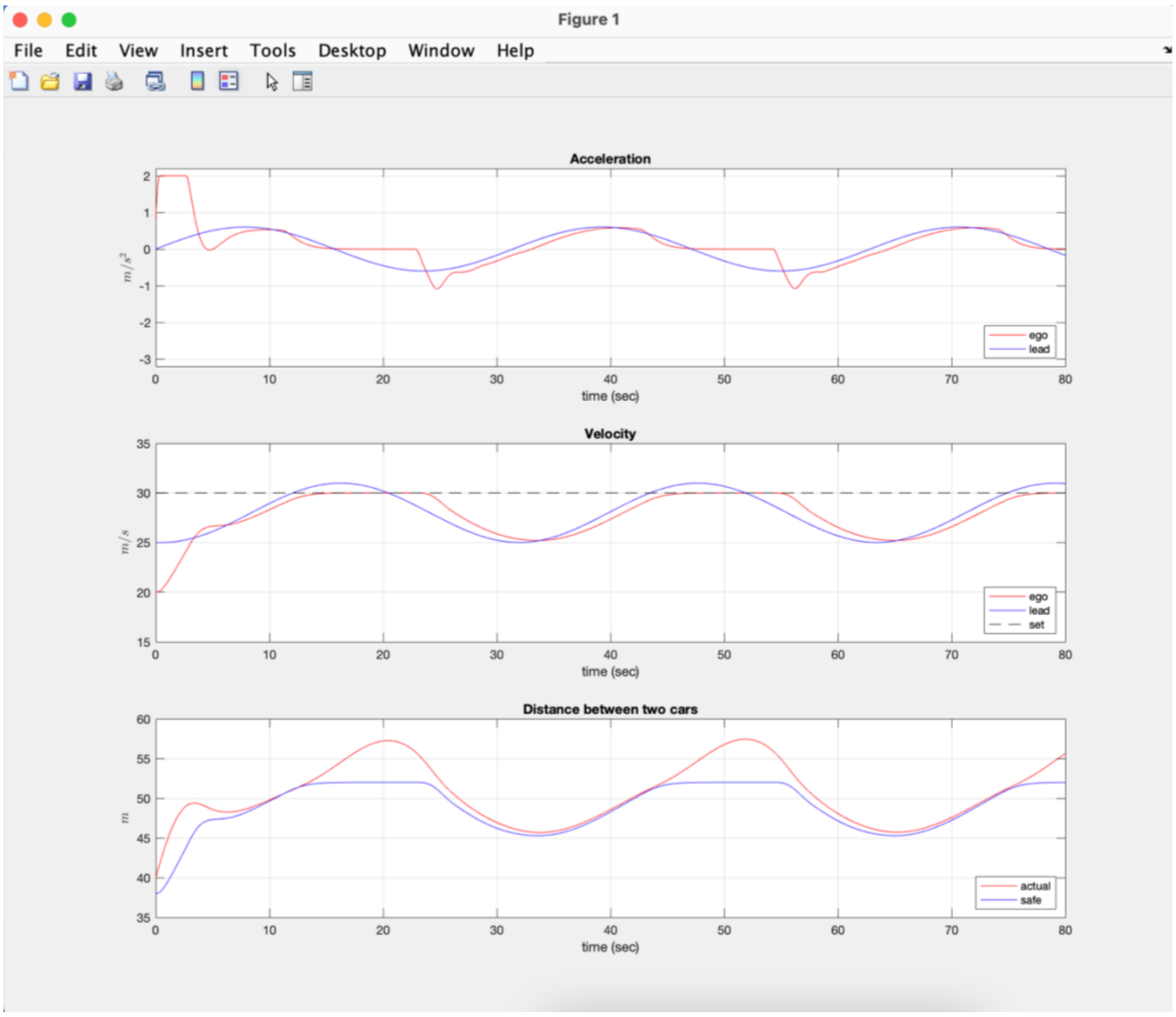


Code used

```
%% Initialize file path and open Simulink Model simulation

addpath(fullfile(matlabroot,'examples','mpc','main'));
mdl = 'mpcACCsystem';
open_system(mdl);

%% Set the value of variables as per the example

Ts = 0.1;
T = 80;

G_ego = tf(1,[0.5,1,0]);

x0_lead = 50;   % initial position for lead car (m)
v0_lead = 25;   % initial velocity for lead car (m/s)

x0_ego = 10;   % initial position for ego car (m)
v0_ego = 20;   % initial velocity for ego car (m/s)

t_gap = 1.4;
D_default = 10;

v_set = 30;
```

```
amin_ego = -3;

amax_ego = 2;


%% Reload the simulation and plot the graph


sim(mdl)

mpcACCplot(logsout,D_default,t_gap,v_set)


%% Clear file path and close graph


rmpath(fullfile(matlabroot,'examples','mpc','main'));

bdclose(mdl)
```

Graph of example values



From the graphs above we can analyze the acceleration, velocity, and distance between both cars.

In the first 10 seconds, both vehicles start 2 meters apart from each other, but then move apart from each other. The Ego Car must accelerate to reach the velocity that will overcome this difference in distance.

By the next 10 seconds, both cars are at relatively the same distance apart. This is the reason the acceleration moves to zero and the velocity of the vehicle is maintained until it reaches the set velocity. Soon after the distance between the two cars increases.

From 20 to 30 seconds, the Lead Car begins to slow down. To maintain the distance, the vehicle begins to decelerate and slowdown in velocity to keep the Ego Car from colliding with the Lead Car.

This process continues to happen as the Lead Car accelerates and decelerates.

Inputting parameters to the simulation model

```
%% New Parameters


Ts = 0.1;

T = 100;


G_ego = tf(1,[0.5,1,0]);


x0_lead = 50;   % initial position for lead car (m)

v0_lead = 25;   % initial velocity for lead car (m/s)


x0_ego = 10;   % initial position for ego car (m)

v0_ego = (80/3.6);   % initial velocity for ego car (m/s)


t_gap = 1.2;

safe_D = 120;

D_default = (safe_D - t_gap*v0_ego);


v_set = 30;


amin_ego = -3;

amax_ego = 2;
```

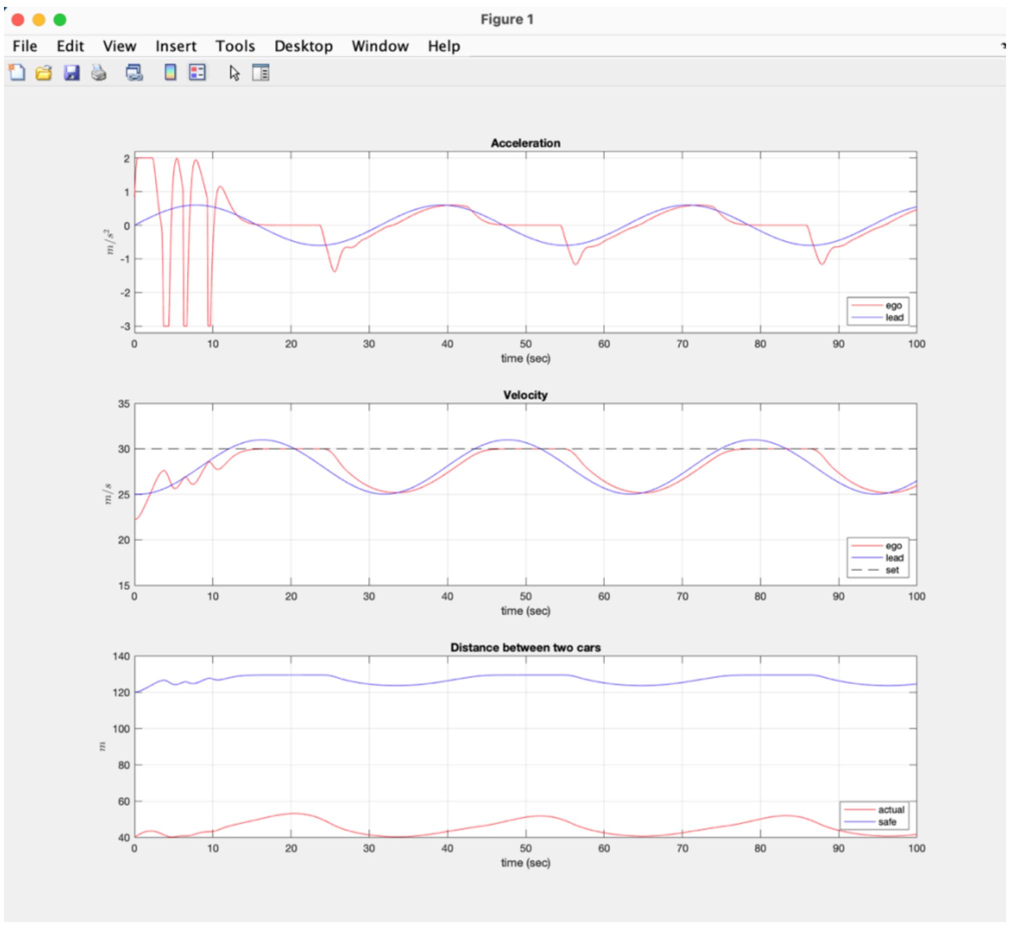
- We begin change the parameters of the existing model with '`v0_ego = (80/3.6);`'. Here we set the initial velocity of the Ego Car to 80 km/h and converting it to m/s.
- We set the Time Gap from 1.4 to 1.2 so that we can set it to 120 seconds – '`t_gap = 1.2;`'
- Lastly, to set the safe Distance to 120 meters, we must use the formula:

$$D_{safe} = D_{default} + T_{gap} \times V_{ego}$$

This will allow us to receive a value for D_default which will maintain the safe distance of 120 meters:

```
safe_D = 120;

D_default = (safe_D - t_gap*v0_ego);
```

Results



From the graphs above we can analyze the acceleration, velocity, and distance between both cars.

In the first 10 seconds, the safe distance is maintained at 120 meters. For the Ego Car to keep up the Lead Car, the Ego Car must accelerate to match the speed of the Lead car. Due to parameter error, the Ego Car accelerates and decelerates rapidly at an irregular interval.

By the next 10 seconds, The Ego Car maintains its velocity near to the Lead Car and at the set velocity. The vehicle no longer accelerates, and the velocity remains constant. The safe distance of 120 meters is maintained throughout the whole duration from here on.

From 20 to 30 seconds, the Lead car begins to decelerate and slow down. The Ego Car abruptly decelerates to maintain the safe distance but soon after begins to accelerate along as the Lead Car accelerates. From here on, the cycle repeats but the acceleration error initially does not reoccur.

Conclusion

To conclude, we looked at how MATLAB and Simulink can be used to model commonly used control systems such as adaptive cruise control. By loading up the Model Predictive Control – Adaptive Cruise Control Example provided by Math Works, we were able to simulate two vehicles: Lead Car and Ego Car. The adaptive cruise control used in the Ego Car was able to consider whether the system should control the speed or control the spacing between it and the Lead Car. After initializing the model and loading in the example parameters, we analyzed how the Ego Car was able to maintain the acceleration and deceleration which controlled the velocity of the vehicle while maintaining the specified safe distance. When we loaded our parameters on to the simulation, the program was able to show a similar plot to that of the example parameters. Due to error that would require further investigation, the plot had an irregular acceleration at the beginning. However, after 10 seconds, the model was simulated as predicted and was able to maintain a relatively constant 120-meter safe distance between the Lead Car and Ego Car.

To improve the accuracy of the model, more parameters could be added to create a robust control system that can factor in other disturbances that may occur during real life circumstances.

Reference

[1] "Adaptive Cruise Control," Ford Motor Company, [Online]. Available: https://www.ford.com/technology/driver-assist-technology/adaptive-cruise-control/.

[2] A. Bemporad, N. L. Ricker and M. Manfred, "Adaptive Cruise Control System Using Model Predictive Control," MathWorks, [Online]. Available: https://www.mathworks.com/help/mpc/ug/adaptive-cruise-control-using-model-predictive-controller.html.