Title
A Dynamic Visualization Environment For The Design And Evaluation Of Automatic Vehicle Control Systems

Author
Xu, Z.

Publication Date
1995

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA. BERKELEY

A Dynamic Visualization Environment for the Design and Evaluation of Automatic Vehicle Control Systems

Z. XU

California PATH Research Report
UCB-ITS-PRR-95-45

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

December 1995

ISSN 1055-1425

Final Report on project MOU 57D

A Dynamic Visualization Environment for the Design and Evaluation of Automatic Vehicle Control Systems

by

Z. xu

†This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation.

A Dynamic Visualization Environment for the Design and Evaluation of Automatic Vehicle Control Systems

by

Z. Xu

Executive Summary

Dynamic Visualization is a project supported by Caltrans through PATH. The objective of this project is to develop a software which can animate automated highways, visualize the dynamics of automatic vehicles, and help the design and evaluation of automatic vehicle systems. In this report we will summarize the accomplishments of this project, describe in detail the functions of the developed software, and explain how to use the software.

The report consists of 5 sections and 4 appendices. In Section 1, we present the motivation of starting this project and the tasks of this project. In Section 2, we present the development results of the project. This section has 3 subsections. In Subsection 2.1, we present the specifications of the dynamic visualization environment. These specifications were first developed in this project and severed as the guidelines for the development of this project. In Subsection 2.2, we present a longitudinal and a lateral vehicle dynamic models. The longitudinal model is developed based on a validated model offered by an automobile manufacture. This model has the similar input-output response as the original model but it is much simpler and is suitable for online simulation. In Subsection 2.3, we present the capabilities of our dynamic visualization system. The capabilities are listed in two parts: (1) animation and (2) online simulation. In Section 3, we present some application examples of the dynamic visualization system. This section contains 4 subsections. In Subsection 3.1, we report the application of this program on the visualization of some advanced vehicle control systems. In Subsection 3.2, we describe the application of this program on system parameter identification. In Subsection 3.3, we present the application of this program on robustness study of vehicle control. In Subsection 3.4 we present the application of this program on gas mileage visualization. Sections 4 and 5 contain the concluding remarks and acknowledgments, respectively. Appendix A lists the deliverables and publications of this project. Appendices B and C give user manuals of the main dynamic visualization program and its modified versions for mass identification and gas mileage visualization. These manuals tell users how to use and how to modify the programs. Finally, Appendix D gives the source code of the programs.

Contents

List of Figures

List of Tables

A Dynamic Visualization Environment for the Design and Evaluation of Automatic Vehicle Control Systems

by

Z. Xu

Southern California Center for
Advanced Transportation Technologies
EE - Systems, EEB 200B
University of Southern California
Los Angeles, CA 90089-2562

Abstract

Dynamic Visualization is a project supported by Caltrans through PATH. The objective of this project is to develop a software which can animate automated highways, visualize the dynamics of automatic vehicles, and help the design and evaluation of automatic vehicle systems. In this report we will summarize the accomplishments of this project, describe in detail the functions of the developed software, and explain how to use the software.

1 Motivation and Project Tasks

When we do simulations or experiments, we need to analyze large amounts of data. Although plotting the data into curves is a useful method in data processing, the plots sometimes may become messy if there are too many plots and the ratio of observation duration to the sampling period is too high. In addition the plots are static and there is no explicit connection between them. Because of these reasons, a more efficient method is needed to analyze the data. The dynamic visualization is such a method.

The main idea of dynamic visualization is to take advantage of a human's spatial intuition. It is said that a picture is worth a thousand words. Evidently a moving picture is worth ten thousand words. When we see some curves on paper, we may not fully appreciate if the system behavior is good or not. Some system characteristics may not be visible by looking at the plots. However, when we see a moving picture, we can easily recognize the system performance based on our intuition. Thus, by transforming the numeric data into an animation, we can examine large amounts of data with better comprehension and efficiency. This can be very helpful for the control design and evaluation. Due to these reasons,

we started the project to develop a dynamic visualization environment for the design and evaluation of automatic vehicle control systems. This environment goes far beyond simple animation systems by providing a set of tools to perform interactive visualization with on-line modification of many visualization parameters such as time and space resolution and to perform online simulation. To be specific, this project has the following tasks:

- To develop specifications for the basic dynamic visualization environment,
- To develop and catalog vehicle models which are suitable for online simulation,
- To develop and implement a dynamic visualization environment which can visualize the longitudinal and lateral dynamics of automated vehicles,
- To collaborate on the development of control system strategies,
- To develop tools to aid in the identification of system parameters of vehicles,
- To study the performance and robustness of promising controllers by simulation.

In the following sections, we will present in detail the development and the results of this project.

2 Development and Results

The project started in September 1991. It was assumed to end in August 1994. With a no cost extension, it ended in June 1995. During this period, we developed two versions of the dynamic visualization program. In the first version, the vehicle model is a very simple model. In the second version the model, although is still simple, is derived from a validated nonlinear vehicle model, and hence is more realistic. In addition, the second version of the program can handle a variable number of vehicles and has more user interface capabilities and more functions. Based on the two versions of dynamic visualization programs, we developed a program to visualize the gas mileage and a program to identify the vehicle mass. We will now describe in detail these results.

2.1 Specifications of the Dynamic Visualization Environment

In the development of the dynamic visualization system, we first had to determine the specifications of the system. We had to decide what need to be visualized and how to visualize them. After some study, we focused on a basic set of capabilities that a dynamic visualization system should have. In this subsection, we describe these specifications and discuss the issues involved in the design of a dynamic visualization environment for automatic vehicle control systems. These specifications have served as the guidelines during the design phase of the dynamic visualization system.

The dynamic visualization system is required to display the dynamics of individual vehicles or platoons of vehicles. Therefore the following basic parameters should be visualized:

Parameters representing the dynamics of individual vehicle, namely, position, velocity, acceleration, and jerk;

Parameters directly related to the performance of the controller, such as the desired position of a vehicle and the position error;

Parameters representing the dynamics of the platoon or a group of vehicles, for example, the string stability;

Parameters directly related to human factor issues, such as the ride quality.

Typically, we get explicit data for vehicle position and velocity and sometimes acceleration from the simulations or the experiments. However, data for some other parameters may not be directly available. Thus, the dynamic visualization system should include some data processing algorithms to estimate the missing parameters from the data set provided by the simulation or experiment. For example, jerk data is not available from simulations in many cases. Since jerk is directly related to the comfort of passengers and the quality of the ride, it is important to visualize this parameter. Thus, we must build a post-simulation data processor to estimate the jerk from position, velocity, and acceleration information.

In order to have a good visualization of all the parameters mentioned above, an effective user interface is very important. It is one of the major components of the design of the dynamic visualization environment. It is also one of the main reasons why the dynamic visualization system is much more advanced than an animation system.

When users use our dynamic visualization tool to visualize the dynamics of platoons of vehicles, they should be able to

- set up the graphic representation of the static scene, i.e. number of platoons, number of vehicles per platoon, the parameters to be visualized, etc.,
- stop at any interesting scene,
- run the animation backward and then forward, examining how things lead up to the situation,
- review a particular interval of time over and over again,
- change the time and space resolution arbitrarily.

These interactive interface abilities provide users with a comprehensive control over when and how to replay a given scene and the ability to home-in on the precise moment of

interest in a time-efficient manner. At the same time, users can be hinted by the correlation between various display components for the underlying cause of a given behavior. In fact, without an effective user interface, visual engineering as discussed above would be an awkward task if one had to change and compile the program when a certain parameter had to be changed.

To develop this interactive user interface, the programming paradigm has to be carefully chosen. The development of an environment with the characteristics mentioned above depends heavily on the mode of programming chosen. For example, in procedural programming with a character-based interface, the application program is always in control while it is running. In this case, the application allows users to provide input only at certain pre-specified sections of the program where multiple levels of menus should be traversed before a certain action can be taken. Clearly this programming paradigm is not suitable for development of a dynamic visualization environment. In fact an event-driven framework is necessary for this development, where applications are embedded in an environment which prepares them to respond to many different events at any time. In this paradigm users are in control most of the time. The application starts by setting up a static scene and then enters a loop from which different functions can be invoked when a designated event is queued.

As mentioned above, an effective user interface should offer users the ability to change the time resolution arbitrarily. Thus, we should provide an environment which best brings out the subtleties associated with the dynamics. Since the time resolution is determined by the time step used in simulations or experiments, in order to increase the resolution, we have to know the data information at other time instances between the time interval in the original data. In this case an interpolation technique has to be used. Employing an interpolation technique is also crucial when the data provided to the environment has non-uniform time steps. This may happen when automatic step sizing is used to find the solutions of a set of differential equations by some simulation software package like Matrixx, or MATLAB. Furthermore, some time steps in a non-uniform time step case may be too large to generate real time graphics. Consequently, without interpolation, visualization would not convey the appropriate time-dependent properties of the system to users. The interpolation technique to be used could be linear, quadratic, or any other nonlinear technique. Using linear interpolation is well justified when the time step in the original data is small enough, since parameters of the dynamics of vehicles can be considered to be differentiable, and differentiability in effect says that locally, parameters change linearly. To summarize, we have to use an interpolation technique in the dynamic visualization system. By employing this technique and considering the capabilities of the computer we used, the dynamics can then be reviewed in real time or slower or faster than real time, as chosen by users. Also it can accept data with non-uniform time steps.

To help users better understand the complexity of the dynamics of platoons of vehicles, the dynamic visualization program must offer the ability to magnify the behavior of

interest. This is another reason why the dynamic visualization program is far beyond basic animation systems. To explain this, let us consider the dynamics of a given vehicle in a platoon. Assume that this vehicle is automatically controlled, and denote the position of vehicle i at time t by $x_i(t)$, the desired position of the vehicle by s;(t), and the error by $e_i(t) \triangleq$ s;(t) $-$ z;(t). The visualization program can be used to visualize $e_i(t)$, but since $e_i(t)$ may be very small compared to other objects on the scene, its visual impact would be limited and detecting its presence may not be easy. Instead, one can visualize $\alpha e_i(t)$, where α is an on-line changeable number, and thus facilitate the observation of position errors in controller performance, the effects of nonlinearities or any other relevant system characteristics, with the appropriate resolution.

Since the software is designed to visualize numeric data describing the dynamics of platoons of vehicles, it mainly accepts and processes data from off-line simulations and experimental data captured during in-vehicle tests. However, on-line simulation in the software is also desirable. If the dynamic visualization environment has on-line simulation capability, then users can interactively specify the speed profile for the leading vehicle and create various scenarios and for vehicle following including disturbances.

2.2 Vehicle Longitudinal and Lateral Model

As said in the previous subsection, online simulation is an important part of the dynamic visualization program. In order to perform online simulation, a vehicle dynamic model is essential. In developing the vehicle model, two things have to be considered. One is that the model has to be realistic so that simulation results will reflect the actual performance of controllers. Another is that the model has to be simple so that online simulation can be performed reasonably fast. In this subsection we will present a simple but realistic vehicle dynamic model.

Vehicle dynamics include lateral dynamics and longitudinal dynamics. We first present the lateral dynamic model and the following notation will be used:

M: vehicle mass (kg)
I,: vehicle moment of inertia $(kg\,m^2)$
l_f: distance from mass center to front wheel axis (m)
l_r: distance from mass center to real wheel axis (m)
C: cornering stiffness $(\mathrm{N/rad})$
δ: steering angle (rad)
y_{br}: lateral deviation (m)
$\hat{V}$: modified vehicle speed (m/s), $\hat{V}$ = max (***V, 4)***
V: vehicle speed (m/s)
β: angle between vehicle movement and vehicle longitudinal axis (rad)
ψ: angle (rad) between the vehicle longitudinal axis and the X axis in the fixed geometry coordinate

ψ_r: road curving angle (rad) relative to the X axis in the fixed geometry coordinate

The dynamic lateral model is as follows:

$$\begin{aligned}
\dot{\beta} &= -\frac{4C}{M\hat{V}}\beta + [\frac{2C(l_r - l_f)}{M\hat{V}^2} - 1]\dot{\psi} + \frac{2C}{M\hat{V}}\delta \\
\ddot{\psi} &= \frac{2C(l_r - l_f)}{I_z}\beta - \frac{2C(l_r^2 + l_f^2)}{I_z\hat{V}}\dot{\psi} + \frac{2Cl_f}{I_z}\delta \\
\dot{y}_{br} &= V sin(\beta + \psi - \psi_r)
\end{aligned} \tag{1}$$

The block digram of this model is shown in Figure 1

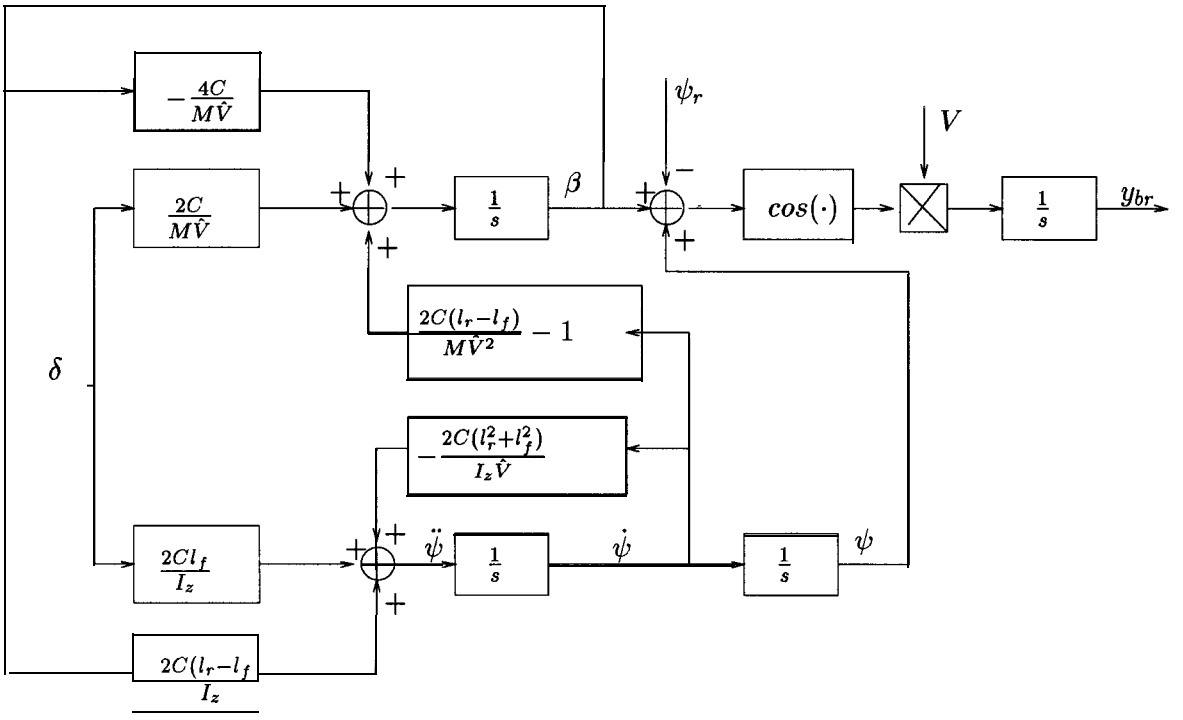

Figure 1: Block diagram of a lateral vehicle model

This model is similar to that developed in [1]. We implemented it in our visualization program because it is realistic and simple. In this model the $\hat{V}$ is introduced to avoid the zero in the denominator. It should be noted that the freeway in our visualization environment is straight and hence ψ_r is zero except in ramps.

Now we present the longitudinal model.

With the help of engineers of an automobile manufacturer, we obtained a validated longitudinal vehicle model. As shown in figure 2, this model consists of four parts: engine, torque converter, transmission, drivertrain, and brake.

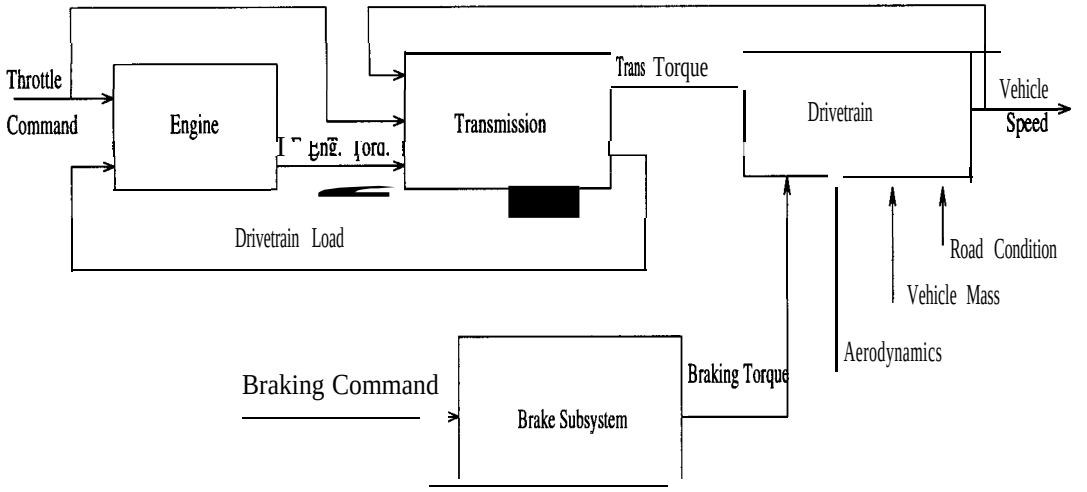

Figure 2: Longitudinal Vehicle Model

The output of the engine subsystem is the engine speed that is a nonlinear function of the air/fuel ratio, the exhaust gas recirculation (EGR), the cylinder total mass charge, the spark advance, the drivetrain load and the throttle angle. The spark advance, EGR, and air-to-fuel ratio are the outputs of an internal controller (inside the engine block of figure 2) whose inputs are the throttle position, engine speed and drivetrain load.

The torque converter takes the engine speed and the transmission shaft speed (which is proportional to the vehicle speed) as inputs and converts them into the pump torque which is the resistant torque to the engine and the turbine torque which is the driving torque.

The transmission subsystem is responsible for transferring turbine torque to the drivetrain depending on the vehicle speed and throttle angle. The transmission considered in Figure 2 is an automatic transmission with hydraulic torque coupling and four forward transmission gears. For a certain gear state, the transmission torque output is a linear function of the turbine torque. The gear state is a nonlinear function of the throttle angle and vehicle speed.

The drivetrain subsystem receives transmission torque and/or braking torque input and outputs vehicle speed, acceleration or deceleration. The vehicle speed and acceleration are affected by the road condition, aerodynamic drag and vehicle mass. The relationship between vehicle speed and transmission torque is also nonlinear.

The brake subsystem, which includes the brake actuator, receives braking commands and outputs braking torque. It behaves like a first order low pass filter [2] with some time delay. Based on experiments, it is found that the time delay is noticeable only in the very beginning when the brake is applied and is very small later on. In addition, it is also found from experiments that the dynamics of the brake subsystem are much faster than those of the drivetrain. In our design, both the time delay and the dynamics of the brake subsystem are ignored, leading to a constant gain relationship between the braking command input and braking torque output. In our simulations, however, the time delay and the dynamics of the brake subsystem are taken into account.

For longitudinal control, the system in Figure 2 may be considered as having two control input variables: throttle angle command and braking command, and one output: vehicle speed. The other inputs such as aerodynamic drag, road condition, and vehicle mass changes are treated as disturbances.

Although this model is validated and is very good in evaluating the performance of controllers, it is complicated and requires a lot of calculations which result in slow simulation speed. On the other hand, online simulations are required to be near real time and subject to the limit of computing power, hence the model has to be simple. Because of these reasons, we developed and implemented a simpler longitudinal model based on the validated model. The simplified model is presented as follows.

For the brake subsystem, we ignored the time delay and modeled it with a first order linear system. Let f_b denote the braking force which is approximately proportional to the braking torque and f_{bcmd} denote the braking force command. The simplified brake model is

$$\dot{f}_b = -5f_b + 5f_{bcmd}. \tag{2}$$

For the model describing the dynamic relationship between throttle and vehicle velocity, the following simplifications were made based on the validated model:

- The dynamics of the transmission were ignored and the gear state was approximated as a simple nonlinear function of vehicle speed. In the validated model, the transmission has some dynamics and the gear state depends on the vehicle speed and throttle angle. Our simplification is justified by the facts that the dynamics of the transmission are much faster than the drivetrain dynamics and the speed increases/decreases with throttle angle in normal driving.
- The torque converter was described by two simple nonlinear algebraic functions. In the validated model the torque converter consists of two look-up tables and some algebraic functions. By interpolation, two simple functions were found to replace these two look-up tables.
- The intake manifold pressure inside the engine was eliminated from the state variables. The manifold pressure is a state variable in the original vehicle model. From

simulations it is found that the dynamics associated with the manifold pressure are much faster than the dynamics associated with engine speed and vehicle speed and hence they are ignored, leading to the elimination of manifold pressure from the state variables. Furthermore, the air-fuel mixture flow rate, denoted by $MDAIR$, ***was*** approximated by a single function of throttle angle. $MDAIR$ is the flow rate of air-fuel mixture charged to the engine cylinders. Based on the simulation with the original model, this function was identified so that the input-output relationship is very close to that in the original model.

After these simplifications, the block diagram of the simplified longitudinal model is shown in Figure 3.

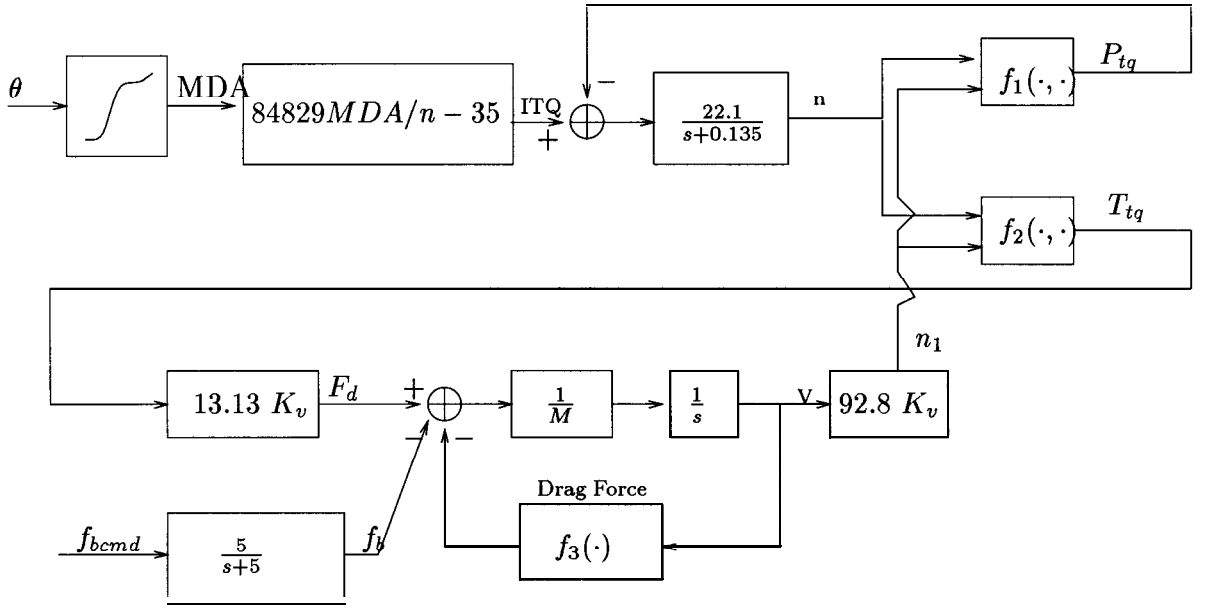

Figure 3: Simplified Longitudinal Vehicle Model

In the block diagram the ***MDA*** is a function of θ given in look-up table 1. The K_v is related to the gear state and is a function of vehicle speed. This function is given by look-up table 2. The variable ratio is equal to $92.8K_v$ ***V/n.*** The functions are as follows:

$$f_1(n, n_1) = 0.00617 * tan^{-1}[25(1 - n_1/n)] * n^2 * \text{IO}.00455 * tan^{-1}[25(1 - n_1/n)] \quad (3)$$

$$f_2(n, \text{n} - 1) = \min[3.1592,\ \text{maz}(1.356,\ 3.1592 - 2.068 n_1/n)] * f_1(n, n_1) \quad (4)$$

$$f_3(V) = \begin{cases} 22.46 + 2.146V + 0.5067V^2 & \text{v} > 0.4(m/s) \\ 500.1V & \text{v} \leq 0.4(m/s) \end{cases} \quad (5)$$

θ (deg)	3	5	7	9	11	13	15	17	19	21	23	25
MDA	0.168	0.198	.37	.59	.9	1.36	1.83	2.57	3.15	3.9	4.8	5.6
6' (deg)	27	29	31	33	35	37	39	41	43	45	47	49
MDA	6.25	7.1	8.0	8.8	9.8	10.5	10.8	11.3	11.5	11.7	11.9	12.1
θ (deg)	51	53	55	57	59	61	65	69	73	77	81	85
MDA	12.3	12.5	12.7	12.9	13.1	13.3	13.7	14.1	14.5	14.9	15.3	15.7

Table 1: look-up table for the function ***MDA(B)***

V (m/s)	0	2	5	7	10	12	15	18	20	40
k_v	2.4	2.4	2.4	1.47	1.47	1	1	0.67	0.67	0.67

Table 2: look-up table for the function $K_v(V)$

The physical meaning and units for some variables are explained below:

V : vehicle longitudinal speed (m/s)
n : engine speed (rpm)
θ: throttle angle (deg) $3 \leq \theta \leq 85$
ITQ : total torque generated by air-fuel (N-m)
P_{tq}: torque used in torque converter (N-m)
T_{tq}: driving torque to the transmission (N-m)
F_d: driving force (N)
f_3: drag force (N)
$\boldsymbol{Fb}_{cmd}$: braking force command to the brake actuator (N)
f_b: braking force from the brake actuator (N)

The physical explanation of this model is as follows. The throttle angle θ generates the flow rate ***MDA*** of air-fuel mixture through a nonlinear function (lookup table). The MDA/n is proportional to the mass of air-fuel mixture charged into the engine cylinder during one cycle. The ***ITQ*** is the total torque generated by the combustion of air-fuel mixture. It overcomes the resistance of the pump torque (P_{tq}) and generates the engine speed n. The torque converter converts the engine speed into driving torque or turbine torque (T_{tq}) and the pump torque through two functions $f_1(\cdot,\cdot)$ and $f_2(\cdot,\cdot)$ which depend on the transmission shaft rotation speed n_1 and the engine speed. The turbine torque generates the driving force through the transmission and tires. The rest of the block diagram is ob-

vious.

In terms of differential equations, the longitudinal model is summarized here.

$$\begin{aligned} \text{Ii} &= -0.135\ \boldsymbol{n} + 22.1(ITQ - f_1\ (\boldsymbol{n},\ 92.8K_v)) \\ \dot{V} &= (F_d - f_3(V) - f_b)/M \\ \boldsymbol{fb} &= -5\boldsymbol{fb} + 5f_{bcmd} \\ F_d &= 13.13K_v\ f_2(n, 92.8K_v) \\ ITQ &= 84829 * MDA(\theta)/n - 35 \end{aligned} \tag{6}$$

As said earlier, this model was simplified from a complex validated model. Thus we need to see how close the input-output relationship is to the original validated model. Figure 4 shows the speed responses of two models to the same step throttle input of 20 degree. It is clear that the two responses are very close, which shows the validity of the simplification.

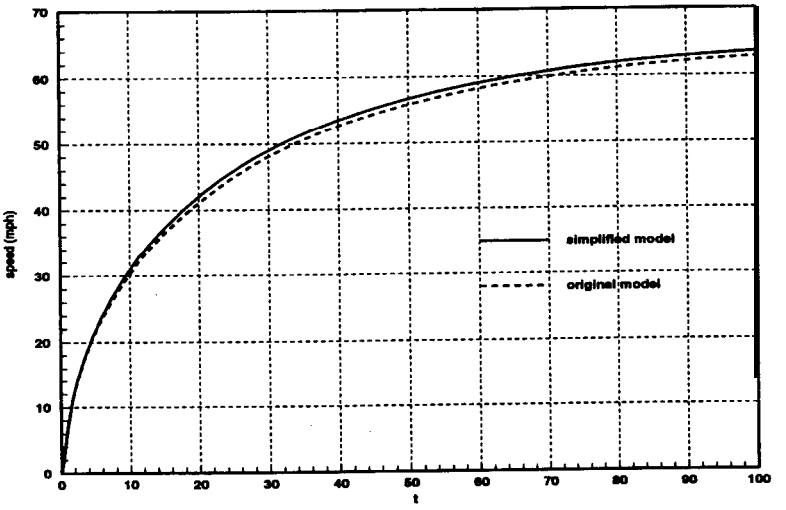

Figure 4: Speed responses to the same step throttle input

2.3 The Dynamic Visualization Environment

Based on the specification requirements for the dynamic visualization environment as described in subsection 2.1, two versions of the dynamic visualization program were developed. Version 1 is more focused on the comparison of different controllers and version 2 has more power in online simulation. Both versions have the same animation capabilities. With slight difference, tow versions of the program offer users the following capabilities:

1. Users can visualize the dynamics of one to three platoons with up to ten vehicles in each platoon. The platoons are displayed side by side. This parallel platoon environment provides a convenient way of comparing the performance of different control

laws. For example users can easily compare the performance of an automatically controlled vehicle platoon to a platoon driven by human drivers. It also provides a convenient way of comparing the performance of two implementations of the same control law with different controller parameters. The program can easily be reconfigured to visualize one platoon only or two or three platoons with any number of vehicles, up to ten in each platoon.

2. Users can visualize each vehicle's longitudinal position, velocity, acceleration, and jerk simultaneously. The positions of all vehicles are displayed graphically on a simulated section of freeway, with accurate scaling relative to the freeway lanes and vehicle size. The velocity of the vehicles can be identified by the relative velocity of road markers. Provided that the data set contains all the necessary information, users can also visualize acceleration and jerk. This is done by looking at the heads of the driver and passenger in each car through the glass sunroof of the car. The driver holds the steering wheel, so his head is straight during constant acceleration but it moves if it is subjected to jerk. It is assumed that the passenger cannot anticipate any changes in vehicle velocity. The passenger's head moves backwards when the vehicle is accelerating and forward when the vehicle is braking, like a mass suspended from a spring. This assumption allows users to get a feeling of what it would be like if they were inside the vehicle they are observing.

   For a more accurate reading of vehicle velocity, acceleration and jerk there are digital meters attached to each vehicle. Users have the option to turn on or off all or some of these digital meters. Because these digital meters are attached to each vehicle, users can get a good reading even if some vehicles change lane. This arrangement of digital meters also makes it possible for users to visualize the dynamics of an automated freeway where different number of vehicles are in different lanes. If the there is no lane change and the number of vehicles in each lane is the same, version one of the program is a better choice. In version one, both digital and analog meters are offered, but these meters are not attached to each vehicle. The position of the meters is such that identifying the meter corresponding to each vehicle can be done at a glance. The rightmost meter corresponds to the rightmost vehicle in the platoon. Each analog meter has three pointers, each one of which corresponds to a car in each lane. The color of the pointer is the same as the color of the vehicle it corresponds to. This facilitates makes quick comparisons of the performance of the controllers in different lanes of traffic.

3. Users can evaluate the performance of the controller by observing the difference between the actual position and the desired position of each vehicle. The ideal or desired position is a function of the control law chosen (for example time headway or constant headway). The desired time headway and/or constant headway can be easily changed online. During the animation, users can use the mouse to pull down a menu in which they can edit the desired time headway and/or constant headway. The desired time

headway and/or constant headway can be different for different vehicles.

4. Users can start, stop and replay the graphic animation of the platoon dynamics at any point of interest. The time step is adjustable over a wide range, thus allowing the playback speed to vary from slow motion to fast forward. Frame by frame advance is also available.

   The data set does not need to include data points for every frame that will be displayed. Data sets typically consist of only a few hundred frames and the time step does not have to be uniform. The program can display animations with apparent resolution of thousands of frames and with any chosen time step. It achieves that by employing a linear interpolation technique to estimate data for time instances that fall between existing frames. This in effect can reduce or increase the speed of animation since the program updates the screen at a fixed frame rate, regardless of the chosen time step. This enhances the illusion of continuous motion as well as the apparent time resolution.

5. Users can change the scale by which any position deviations are magnified. When the controller algorithm is nearly perfect, the position error may become very small. By magnifying any remaining errors users can evaluate the performance of their design and iterate until they are fully satisfied.

6. Users can interactively modify the zoom factor by selecting the length of the road to be displayed. This allows the viewers to focus on a section of the platoon instead of getting an overall 'helicopter' view.

7. Users can interactively choose the focal point for view. In the current versions of the program one vehicle appears in a fixed position and the changes in vehicle speed are visualized by the motion of the road markers and the movement of other vehicles relative to the fixed vehicle. This enables users to visualize the platoons with good space resolution. The users can use the mouse to pick any vehicle as the reference vehicle and drag it to any position on the screen. (When the reference vehicle is dragged, other vehicles move accordingly.)

8. Users can change the spacing between the road markers. This gives users a reference point by which they can sense (and even measure) distances with ease unparalleled in the real world.

9. Users can visualize the lateral dynamics. If the data file contains information on lateral position, the program can display the lateral position of each vehicle and hence the users can use it to visualize the lateral-related dynamics such as lane change, merge, cut-in, etc. The program can accept data which can either have or not have fields for lateral position and lateral speed. The users only have to modify the plain text configuration file to specify if the data file contains the lateral position and speed.

The above functions are related to animation. The following functions are related to online simulation. During online simulation, the program has all the functions listed above and in addition has the following functions.

1. Users can perform simulation and animation simultaneously. The program offers a realistic vehicle model which is derived based on a validated but complicated vehicle model. The program also offers a simple longitudinal controller and a simple lateral controller for demonstration. The longitudinal controller is a fixed PID controller [3] which has been successfully tested in real vehicles.

   The model and controllers are written in separate C functions. The users can substitute them with their models and controllers and visualize the performance of their controllers.

2. Users can choose different time steps for animation and for simulation. In order to obtain good accuracy in simulation, the simulation time step should be small. If the same small time step is used for animation, the animation may be too slow and look unrealistic. To increase the simulation accuracy and the animation speed, two separate slider bars are offered by the program. Through these two slider bars, the users can choose the simulation time step and animation time step independently.

3. Users can virtually drive a vehicle longitudinally during online simulation. The program offers a slider bar for the throttle angle and a slider bar for the brake. Users can use the mouse to change these slider bars and hence control the longitudinal position/speed of a vehicle. The default vehicle to be driven is the leading vehicle in a platoon thus the user can interactively change the trajectory of the leading vehicle and check the performance of the following vehicles. Users can also choose any other vehicle to drive.

4. Users can interactively change the lateral position of any vehicle during online simulation. At any moment of time, the users can select any vehicle for lane change with the mouse. The program offers functions for automatic lane change and manual lane change. If the automatic lane change option is chosen, the program will control the lateral position of the selected vehicle. If the manual lane change option is chosen, a steering wheel will appear and users can control the lateral position of the selected vehicle with this steering wheel. When a vehicle changes lane, other vehicles will respond accordingly. Therefore, this function will enable users to simulate and visualize the dynamics associated with lane changes. This function together with the above functions will also enable users to virtually drive any vehicle.

5. Users can choose different vehicles to form a platoon. The program offers a set of vehicle makers and models like Ford Lincoln Town Car, Ford Escort, etc. Users can select the desired model by clicking the mouse button on the ‘Car Model’ option. When a model is chosen, the program will determine the following car features: (1)

weight, (2) engine size, (3) length, (4) width, (5) maximum acceleration, (6) maximum deceleration, (7) maximum brake force (in terms of deceleration), (8) maximum speed, and (9) horse power. Of these features, the maximum acceleration, maximum deceleration and maximum brake force are used in the controller implemented in the current online simulation. The other features are offered for future applications. If users do not like the numeric values of these parameters, they can modify them online.

6. Users can change the roadway commands for headway and speed during online simulation. These headway and speed commands are the recommendation to all automatically driven vehicles and will be obeyed in a normal situation. The users can modify the desired headway for each vehicle as long as the headway is larger than that specified by the roadway. The modification is done by clicking the left button of the mouse on a selected vehicle and choosing the option 'Car Features'. Using this function, users can form a non-uniform platoon. In the 'Car Features' menu, the users can also change some car feature related parameters such as weight, maximum acceleration, etc. as well as the values of variables 'userl', 'user2', . . . which are offered for users' applications.

7. Users can add and remove vehicles during online simulations. In order to add a vehicle, the users can use the mouse to select the 'add car' option. When this option is chosen, the users can further specify if they want to add a vehicle in front of a platoon, behind a platoon, or from an on-ramp. In order to remove a vehicle, users can drive the vehicle to an off-ramp or simply choose the 'off-ramp' option. After the users choose a vehicle in the right most lane and selects the 'off-ramp' option, the vehicle will disappear at the next off-ramp automatically. If the selected vehicle is not on the right most lane, a lane change maneuver has to be done first.

8. Users can change the on-ramp and off-ramp configuration during online simulation. To do this, the users need to use the mouse to pull down the 'Roadways' menu and select the 'Ramp Settings' option. When these are done a dialog window for changing the configuration of the ramps on the roadway will appear. Through this window, the users can view what and where are the ramps down the freeway as well as add, modify, and delete a ramp. The users can also change the freeway ramp configuration through a configuration file. For more detail, see the online help for roadways.

9. Users can change the grade of the highway during online simulation. To do this the users can use the mouse to pull down the 'Roadway' menu and select the 'Hill Grade' option. When this option is chosen, a dialog window will appear and users can specify the grade of the freeway and the starting and ending position of the uphill or down-hill. For more information see the online help.

10. Users can simulate different evolutionary representative system configurations (ERSCs). It is expected that IVHS will be developed and implemented evolutionarily from low level to high level [6]. From low level to high level, more automation will be

added gradually. To show this evolution, 4 levels are offered in this program. In level 1, the system has one designated auto-lane, and the automated functions include autonomous intelligent cruise control (AICC), blind spot warning, check-in, check-out, etc. In level 2, the system has all the functions of level 1 plus longitudinal collision avoidance. In level 3, lane keeping function is added. In level 4, the system has two auto lanes and one manual lane, and automatic lane change function is added. During online simulation, when a level is chosen, the software will switch on or off the corresponding functions. In every level, the user (driver) has to initiate the check-in procedure before performing the lane change maneuvers from the manual lane to the automated lane. Otherwise, a warning signal will be flashing.

11. In either animation or online simulation, the program offers the online help. Users can select different topics for help by using the mouse.

In the current dynamic visualization program, many options are controlled by a parameter configuration file. This file is a plain ASCII file where users only need to change some numbers to change the configuration. In particular, users can specify in the configuration file the number of platoons, the number of vehicles in each platoon, and the desired headway for each platoon. Users can specify if they want to do online simulation or just animation. Users can specify which of the following data fields are available: lateral position, longitudinal velocity, lateral velocity, acceleration, jerk. The longitudinal position is always required, but users can specify in the configuration file if the longitudinal positions are relative to the front vehicle (which is always the case when the data is collected from an experiment) or relative to the starting point (absolute). If the position is relative, then the position of the first vehicle in a platoon is not needed and the program will estimate the position from the speed of the first vehicle. If the acceleration and jerk are not available, users can specify if they want the program to estimate them. Finally users can also specify if the platoon leaders in different platoons have the same longitudinal trajectory, which happens in simulations when we want to compare different control algorithms. In the configuration file, users don't have to specify everything each time. If some items are missing, the program will assume some (meaningful) defaults. The description of the parameter file and the default parameter setup can be found in the user manual in the Appendix.

In the dynamic visualization program, the effective user interface is implemented using the OSF/MOTIF widgets and gadgets. The program is event oriented. Most of the functions have to be triggered by the user. Otherwise, the functions are inactive and do not consume any CPU time.

3 Application Examples of The Dynamic Visualization Environment

As said before, the dynamic visualization program was developed to help the design and evaluation of advanced vehicle control systems. During the development of this program, we have used it to visualize some automatic vehicle control systems (AVCSs) and obtained some useful results. We also modified the program for some specific applications. In this section, we report these applications. In subsection 3.1, we report the application of this program on the visualization of some AVCSs. In subsection 3.2, we describe the application of this program on system parameter identification. In subsection 3.3, we present the application of this program on robustness study of vehicle control. Finally in subsection 3.4 we present the application of this program on gas mileage visualization.

3.1 Visualization of Automatic Vehicle Control Systems

During the development of the dynamic visualization program, we have used it to compare the performance of vehicles under automatic control and vehicles under manual control. To do the comparison, we first simulate the two platoons one of which is under automatic control and another one of which is under manual control. The model for the manual controlled platoon is as follows:

$$\begin{aligned} \dot{X}_i &= V_i \\ \dot{V}_i &= 0.37(V_{i-1} - V_i) \end{aligned} \tag{7}$$

where X_i and V_i are the longitudinal position and velocity of the i-th vehicle respectively. This model is called Pipe model [7]. The model for the automatically controlled platoon is as follows:

$$\begin{aligned} \dot{X}_i &= V_i \\ \dot{V}_i &= A; \\ \dot{A}_i &= k_1(X_{i-1} - X_i - l_0 - \lambda V_i) + k_2(V_{i-1} - V_i - XA;) + k_3(A_{i-1} - A;) \end{aligned} \tag{8}$$

where A_i is the acceleration of the i-th vehicle, l_0 is a constant related to the vehicle length and extra safe space, and $k_j, j = 1, 2, 3$ are some design constants for the controller. This model is adopted from [8]. During the simulations the first vehicles in both platoons (platoon leaders) are given the same longitudinal trajectory. After the simulation, we use the visualization program to visualize the simulation results. Because of the parallel environment offered by the program, the simulation results are clearly illustrated and compared. Through the animation, it is easily found that the automatic control can achieve larger capacity and the manual control has smaller jerk. To compare the results, one can invoke the software package and run 'demo v 1' where the 1 means demonstration 1 and v may be either 1 or 2, indicating the version number,

Of course the above conclusion is made for a particular set of $k_j, j = 1, 2, 3$. For some other set of these parameters, the jerks may be smaller or larger. To achieve good tracking, small jerk, and short headway, these parameters need to be fine tuned. Actually we have used the dynamic visualization program to help the selection of these parameters. In this application, we visualize the simulation of two platoons both of which are described by (8) but have different $k_j, j = 1, 2, 3$. The animation offers good comparison of the performance of the controllers with different parameters. For example, by visualizing the simulation results, we can easily see which set of parameters is better. Even if the position errors are small, we can still use the **scaler** slider bar offered by the program to amplify the position error and tell which set of parameter is better. To observe this, one can run 'demo v 2'.

During the development of this dynamic visualization program, we have collaborated on the development of control system strategies with other research groups. We simulated the performance of different throttle controllers for automatic longitudinal vehicle following, animated and compared their results with the visualization program, made suggestion on the adjustment of parameters, and simulated again. This process has been repeated many times. The controllers we visualized are the PID controller with gain scheduling, fixed gain PID controller, and adaptive controller [3, 4, 5] which were designed by Professor Ioannou's group at the University of Southern California. The visualization program did offer a good comparison of these controllers and give the viewers a dynamic feeling of the performance of these controllers. To see the comparison of these controllers, one can run 'demo v 3' which animates the simulation results of the three controllers under a common speed trajectory. In this demo, the top lane shows the vehicles with the adaptive controller, the middle lane shows the vehicles with the PID controller with gain scheduling, and the bottom lane shows the vehicles with the fixed gain PID controllers.

Another application of the dynamic visualization program is to animate experimental results and evaluate the performance. We have used this program to animate both four vehicle following experiments and two vehicle following experiments. Some of these animations are in 'demo v 4' and 'demo v 5'. 'Demo v 4' is the animation of a two vehicle following experiment where the control objective is to make the following vehicle track the front vehicle with changing desired time headway. This animation offers useful information about the position error, the adjustment of time headway, and the effect of the acceleration limit. 'Demo v 5' is the animation of a four vehicle following experiment where the control objective is to make the following vehicle track the front vehicle with a constant relative distance. One useful piece of information conveyed by the animation is about the slinky type effect i.e. the platoon stability. With the help of the **scaler** slider bar, the slinky type effect is clearly illustrated.

These actual application tests show that dynamic visualization is a very useful tool for the analysis of the complex and highly coupled dynamics of vehicle platoons.

3.2 Application in Parameter Identification

Although the prime application of the dynamic visualization program is to animate automated freeways and automatic vehicle control systems, with some minor modification, it can also offer help in the identification of system parameters. The key resource for this application is the parallel architecture of this program. As mentioned above, the program offers three lanes in parallel. In the application on parameter identification we can use one lane to display the vehicle some of whose parameters are under identification and use another lane to display the vehicle some of whose parameters are the estimated ones. Both of these vehicles will follow another vehicle in each lane and the leading vehicle will have the same longitudinal trajectory. The advantages of using the dynamic visualization program in parameter identification are as follows:

1. The program displays the states of each vehicle and the position error during simulation. Users can turn on the estimation algorithm when the initial conditions are appropriate and stop the estimation when it is necessary.

2. Users can sense the effects of parameter error. It is known that the parameter under identification may not necessary converge to the actual values. In simulation we can always check if they converge to actual values, but in practice, we can not. Thus we want to know the effects of system parameter error. Since the program lets users visualize the vehicle dynamics during simulation, users can compare the dynamic behavior of the vehicle with given parameters to the behavior of the vehicle whose parameters are under identification, detect the difference, and judge the effect of parameter identification error. For example, suppose that the parameter identification has 10% error. If the users see little difference in the behavior of two vehicles, then this error is acceptable and the identification algorithm is good.

3. Users can change the parameters to be identified at any time during simulation without re-starting the simulation. The program offers for each vehicle a table which contains 'Car Features' related parameters and can be brought up and changed online by clicking the mouse button on the vehicle. The users can fill in the table with the parameters they want to identify. Then during simulation, they can use the mouse to bring up the table and change the desired parameters and check the performance of the identification algorithm.

4. Users can change the parameters for the identification algorithm. In developing the identification algorithm, normally we can prove that the algorithm can guarantee the parameter or output convergence as $t \to$ co. However, how fast the actual convergence is depends on the choice of parameters for the algorithm and there is little theoretical guidance on the choice of these parameters. The choice of these parameters depends on experience and simulations. In addition to the 'Car Features' related parameters, the program also offers a set of user defined variables like userl, user2,... for the users' convenience. Users can use these variables in the identification algorithm as the adjustable parameters and change them at any time during simulation.

As an example, we will show how to use the program to identify the vehicle mass during automatic vehicle following.

The algorithm for identifying vehicle mass is developed based on the simplified longitudinal model as shown in (6). We can separate this model into a throttle-to-velocity model where $f_b \equiv 0$ and a brake-to-velocity model where the the driving force F_d is very small. From equation (6) we can see that the brake-to-velocity model is much simpler than the throttle-to-velocity model. Due to this reason, we identify vehicle mass only when the brake is applied. In other words, we develop the mass identification algorithm based on the brake-to-velocity model.

Since the mass identification algorithm to be developed will be used together with the control algorithm for automatic vehicle following, we first need to have some kind of longitudinal controller. In our program the throttle controller is the fixed gain PID controller chosen from [3]. The brake controller is the same as that in [3]. The switching logic is chosen to be a function of the desired acceleration and it has some hysteresis. Since this is not focus of this project, we omit the detail here. Interested readers can find out this switching logic from the enclosed source code.

Because the mass identification algorithm is active only when the brake is applied, we write the equations for vehicle following when the brake controllers is active here:

$$\begin{aligned} \dot{X}_r &= V_l - V_f \\ \dot{V} &= (F_d - f_3(V) - f_b)/M \\ \dot{f}_b &= -5f_b + 5M_0[k_1(V - V_l) - k_2(X_r - hV)] \end{aligned} \tag{9}$$

where V_l is the speed of front vehicle, h is the time headway, M_0 is a constant indicating the initially estimated mass, and k_1, k_2 are two constants.

We use adaptive control theory to develop the mass identification algorithm. Since the parameter to be identified or the uncertainty (M) do not exist in the equation for $\dot{X}_r$, we can use partial adaptation. Thus the reference model is

$$\begin{aligned} \dot{V} &= (F_d - f_3(V) - f_b)/M \\ \dot{f}_b &= -5f_b + 5M_0[k_1(V - V_l) - k_2(X_r - hV)], \end{aligned} \tag{10}$$

and the model whose mass is the estimated value is

$$\begin{aligned} \dot{\hat{V}} &= (F_d - f_3(V) - \hat{f}_b)/\hat{M} \\ \dot{\hat{f}}_b &= -5\hat{f}_b + 5M_0[k_1(\hat{V} - V_l) - k_2(X_r - h\hat{V})]. \end{aligned} \tag{11}$$

Based on (10) and (11) and following the standard Lyapunov method for adaptive controller design, we have the algorithm for the mass identification as below:

$$\frac{d}{dt}\left(\frac{1}{\hat{M}}\right) = (F_d - f_3(V) - \hat{f}_b)[\gamma_1(V - \hat{V}) - \gamma_2(f_b - \hat{f}_b)] \tag{12}$$

where γ_1, γ_2 are some properly chosen constants and $\gamma_1 > 0, \gamma_2 > 0$.

We can prove that, with properly chosen $\gamma_1, \gamma_2, \hat{V} \to V$, $\hat{f}_b \to f_b$, and $\hat{M} - M$ is bounded, if the brake is applied all the time. If V_l is persistent excitation, then we can further have $\hat{M} \to M$.

This identification algorithm needs the values of the idle driving force F_d and the air dynamic dragging force $f_3(V)$. Both of them are not measurable. We can only estimate them. The good thing is that both of them are small. The identification error caused by the inaccurate estimation of these forces will be small. This is especially true when hard braking is applied.

In addition to this problem related to F_d and $f_3(V)$, there are two other problems in implementing this identification algorithm. One is how to adjust the parameters γ_1 and γ_2. Although we can calculate off-line the values for γ_1 and γ_2, these calculated values only guarantee stability theoretically, and they give no information on the performance or the convergence speed. In order to achieve fast convergence, we want to be able to change them during the simulation. Another is when to turn on the algorithm or when to apply the brake. The dynamic visualization program offers good solutions to these two problems.

To perform the mass identification we configured the dynamic visualization program to visualize 4 vehicles. Vehicles 1, 2 are in the top lane, Vehicles 3, 4 are in the middle lane, and Vehicles 1, 3 have the same longitudinal trajectory. Vehicle 2 follows vehicle 1 under automatic throttle control and brake control which is described as in (9). Vehicle 4 follows vehicle 3 also under automatic control. The throttle controller for vehicle 4 is the same as that for vehicle 2, but the brake controller is as described in (11). The parameters γ_1, γ_2 are linked to the user defined variables, *user1*, user2 respectively. Then during the simulation, users can virtually drive vehicle 1 (vehicle 3). When the speed is sufficient and vehicles 2, 4 are close to the desired positions, the users can apply the brake on vehicle 1 (3), and the brakes in vehicles 2, 4 as well as the identification algorithm will be turned on accordingly. Since the variables *user1*, *user2* are online changeable, the users can change them at any time during the simulation to increase the convergence speed. When the speed becomes too low, users can release the brake and increase the throttle of vehicle 1 (3) and as a result the identification algorithm will become inactive. After the speed becomes sufficient again, the users can start another round of identification. Since the time step required for identification is much smaller than that required for the simulation without identification, the program allows users to change the time step during simulation. Users can select a large time step while the throttle is applied and a small time step while the brake is applied. Furthermore, if the identified mass has converged satisfactorily for a given value of mass, the users can change the actual mass and start the identification again without stopping the program.

To know more about the parameter identification, one can run ‘dynavisiden’ and see the demonstration. It should be noted that this project is to develop a visualization environment to help the parameter identification. The development of identification algorithm is not a focal point of this project. The mass identification is only an example which shows how to use this visualization environment. Future users can implement their own control algorithms and identification algorithms in the program with some changes in the source code.

3.3 Robustness Study of Vehicle Controllers

Robustness study of vehicle controllers is an important issue in control design. The dynamic visualization program not only offers the capability to simulate and visualize the dynamics of AVCS, but also offers the tools for robustness studies. The tools are offered in four areas.

1. It offers tools for users to virtually drive a vehicle so that they can introduce a disturbance to the traffic. For example users can increase the throttle to accelerate a vehicle, apply the brake to decelerate a vehicle, or turn the steering wheel to change lane. At the same time, the users can visualize the performance of automated vehicles under these maneuvers.

2. It offers tools for users to change the road grade so that they can introduce the disturbance force to drivetrain. To change the road hill grade, users can use the mouse to pull down the ‘Roadway’ menu and select the item for ‘hill grade setting’. When this is selected, the users then can specify the hill grade and the start and end point of the section of freeway with the selected hill grade. When the hill grade is changed, the users can check the robustness of the controllers during up-hill or down-hill conditions.

3. It offers tools for users to change each vehicle’s parameters such as vehicle mass, desired time headway, etc. The changes in the parameters such as vehicle mass allow users to check the robustness of the controllers under some model uncertainties and the changes in the parameters such as desired time headway allow users to check the robustness of the controllers to changes in initial conditions and to nonuniform platoons in which each vehicle has different characteristics and time headway.

4. It also offers some online changeable and user defined variables to let the user introduce some other disturbance.

3.4 Gas Mileage Visualization

Gas efficiency is a big concern in vehicle design and vehicle longitudinal controller design. In vehicle design the absolute gas mileage is of concern. That is, the designer wants to make the vehicle gas efficient. However, a gas efficient vehicle does not necessarily result in saving gas in actual driving. The actual gas efficiency depends on how the throttle and brake are controlled. For example, let us consider two kinds of control strategies for a given vehicle.

One is to apply the brake very actively and the controller has oscillation between applying throttle and applying brake even when the desired speed trajectory is smooth. Another is to apply the brake as few times as possible. It is obvious that the later control strategy will result in high gas mileage. Therefore, we need to consider gas mileage in the longitudinal controller design. Of course, the concern of gas mileage here is relative gas mileage. We want to compare the gas efficiency of different controllers.

Because of its parallel architecture, the dynamic visualization program actually offers a good way to compare the gas mileage of different controllers. For example, we can use one lane to display one platoon of vehicles under one control algorithm and use another lane to display the second platoon under another control algorithm. Instead of showing the velocity, acceleration, and jerk with meters, we can show the gas mileage with one set of meters. The two dials in one meter offer the direct comparison of the gas mileage of different controllers.

In the following we will present a program package which offers the visualization of gas mileage.

The program has a configuration of two platoons whose leaders have the same longitudinal trajectory. The program offers a set of speed profiles for the leading vehicle to simulate different traffic conditions like light, moderate, heavy, and local traffic. In light traffic, the speed for the leading vehicle is a constant of $29\mathrm{m/s}$. In moderate traffic the speed changes smoothly between 13.4 m/s and 23.8 m/s with a period of 40 seconds. In heavy traffic the speed changes smoothly between 4.5 m/s and 13.4 m/s with a period of 40 seconds. In local traffic the speed profile is as follows:

speed equal to t (m/s) as $0 \leq t \leq 18$ seconds,
speed equal to 18 m/s as $18 \leq t \leq 28$ seconds,
speed equal to $18 - 2.5(t - 28)$ (m/s) as $28 \leq t \leq 35$ seconds,
speed equal to 0 as $35 \leq t \leq 45$ seconds.

Then the speed repeats the cycle again. These speed profiles can be easily changed off-line. During the simulation, users can select any of these speed profiles or virtually drive the leading vehicle to generate desired speed profiles. In both platoons the longitudinal controllers are the fixed PID controllers adopted from [3]. However, in one platoon, the hysteresis for the logic switch which governs the switching between throttle controller and brake controller is smaller than that in another platoon, which implies the brake in one platoon is more active than that in another.

In addition to displaying the speed, and acceleration with analog meters, the program uses a set of analog meters to display the gas mileage of each vehicle and the average gas mileage of each platoon. Each meter has two dials corresponding to two vehicles in the same position of two platoons.

The gas mileage is calculated based on the formula: miles traveled divided by the gas used during this distance. The distance is easy to obtain during the simulation. To obtain the gas consumed in a period of time, we need a good engine model. With the help of an automobile company, we obtained a validated engine model. Based on the model we further developed a simplified engine model which has the state of engine intake manifold pressure and the fuel flow rate. The simplified block diagram of the engine model is shown in Figure 5. In this block diagram, the function f_4 maps the throttle angle, fuel-air-mix flow rate, and the intake manifold pressure into the change rate of the intake manifold pressure. The function f_5 maps the intake manifold pressure and engine rotation speed into the flow rate of fuel-air-mixture charged into the engine cylinders. The function f_6 maps the fuel-air-mix flow rate into the total torque generated by the cylinders. The P_{tq} is the pump torque used in the torque converter (see Figure 2 and Figure 3). For a given vehicle, the fuel flow rate is proportional to the fuel-air-mix flow rate. Combining this simplified vehicle engine model with the models for torque converter, transmission, brake, drivetrain, we will have the complete longitudinal vehicle model used in the gas mileage visualization program.

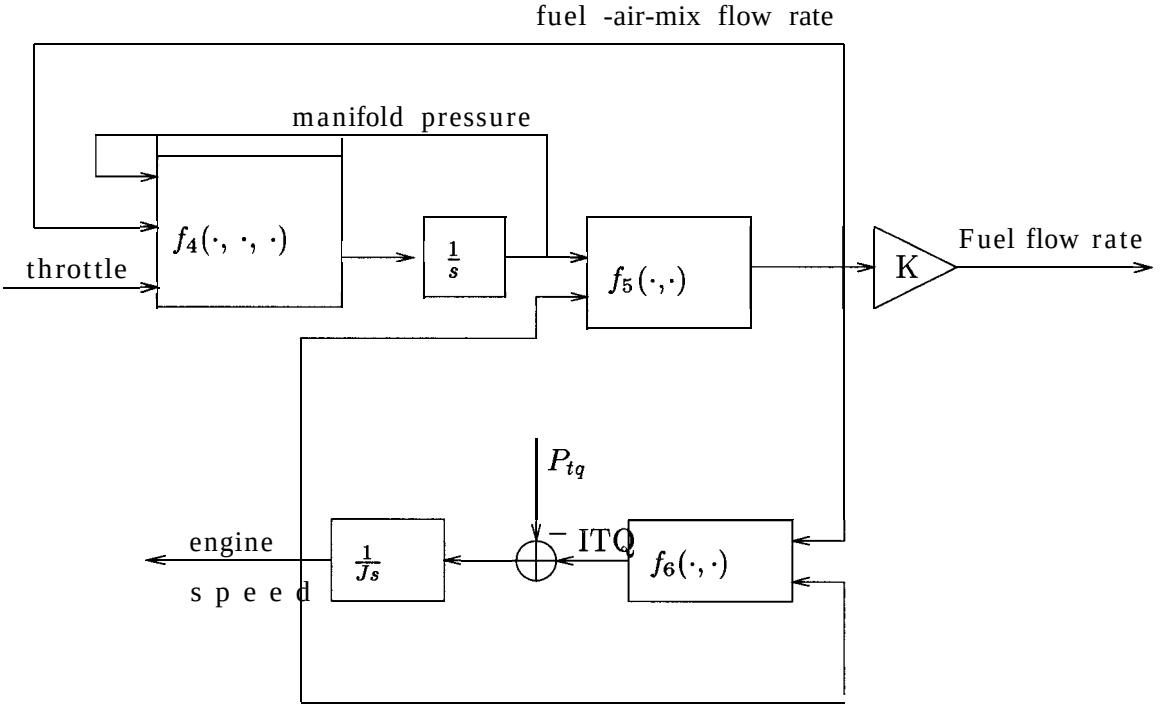

Figure 5: Simplified engine block diagram

In our program, we visualize the gas mileage over a time interval T_0 instead of visualizing the whole accumulated gas mileage. The T_0 is set to 100 seconds. users can change this

number and recompile the source files if different time interval is needed. The algorithm for calculating the gas mileage is as follows. Let ***T*** be the sampling period in the simulation, ***GAS(t)*** be the gas consumed up to time t, X(t) be the distance traveled up to time t, and ***GM(t)*** be the gas mileage at time t. Initially the ***GM(O)*** is set to 10 miles per gallon. Then the gas mileage is updated by

$$GM(kT) = \begin{cases} \boldsymbol{GM[(k-1)T]} & \text{if } GAS(kT) - GAS(k_0T_0) < \epsilon \\ (1-\alpha T)GM[(k-1)T] + \alpha\, \boldsymbol{T}\, \gamma \frac{X(kT)-X(k_0T_0)}{GAS(kT)-GAS(k_0T_0)} & \text{else} \end{cases} \tag{13}$$

where a is a design constant corresponding to time constant for the first order low pass filter, 7 is a constant corresponding to the unit change, k_0 is the maximum integer less than $(k-1)T/T_0$, and ϵ is a small number introduced to avoid a very small denominator. The physical meaning of the above equation is that: $\frac{X(kT)-X(k_0T_0)}{GAS(kT)-GAS(k_0T_0)}$ is the accumulated gas mileage from k_0T_0 to kT and it passes through a low pass filter $\frac{\alpha}{s+\alpha}$ to generate a smooth gas mileage.

For more information on the gas mileage visualization, one can see the enclosed source code and run 'dynavis-gas' demonstration program.

4 Concluding Remarks

The current dynamic visualization program has all the functions we promised to develop. It will greatly facilitate detecting undesirable phenomena in automatic vehicle following (platooning), designing and evaluating vehicle control systems, and comparing the performance of different control strategies. It is expected that this program will be a valuable tool to designers of automatic vehicle following and platooning systems. To further enhance the capability of this program, the following functions could be added.

1. Add more functions for visualizing the dynamics of roadways. The current dynamic visualization program mainly focuses on the visualization of dynamics of vehicles. For example, it is desirable to have functions which allow users to change the entry/exit configuration and traffic separation policy interactively.

2. Add more vehicle models for different applications. Currently we have implemented a simple but realistic model in the program. This model does not include the dynamics of suspension, tire, etc. In some applications, users may want to visualize and check these dynamics. Thus it is good to offer a selection of models in the program. These models may vary from simple, reduced order linear models to complicated, full order nonlinear models. In the same time the program will offer users the capability to choose different models for different purposes interactively. When a model is chosen, the program will give the brief description of the model, such as the inputs, outputs, states, frequency characters, etc.

3. Add functions which allow users to modify the controller interactively. Currently, the program has implemented a simple vehicle controller for demonstration. If users wants to modify the controller, they have to modify the source code and re-compile it. However, changing the source code is not easy because the users have to first get familiar with the source code. It is highly desirable for users to visualize/change the control laws interactively. One solution might be to link Matrixx/Matlab software with the visualization program. The users could design/change controllers by drawing diagrams (which is relatively easy), and Matrixx/Matlab will either do simulation or generate source code. This source code will be linked to the visualization program. Another solution might be to develop software which interactively asks the users to input some information like the order of the controller, the parameters in the controllers, and inputs and outputs of the controllers, and transform the users' inputs into source code.

The above desired enhancements are offered as a reference of future research. Other enhancements may also be desirable.

5 Acknowledgments

I would like to thank Dr. John Hauser who initially started this project. I would also like to thank Mr. Mehran Mesbahi, Mr. Alex Kanaris, and Mr. Paul Fan who helped in developing the programs.

References

[1] H. Peng and M. Tomizuka, "Vehicle Lateral Control for Highway Automation", *Proc. of American Control Conference,* 1990, pp. 788-794.

[2] H. Raza, Z. Xu, P. Ioannou, and B. Yang, "Brake Modeling for AVCS Application", USC CATT technical report to PATH, Report 94-01-01, January, 1994.

[3] P. Ioannou and Z. Xu, "Throttle and Brake Control System for Automatic Vehicle Following", *IVHS Journal,* Vol. 1(4), 1994 pp. 345-377.

[4] P. Ioannou and Z. Xu, "Throttle Control for Vehicle Following", ***1993 IEEE Conference*** on Aerospace ***Control Systems,*** Westlake, California, 1993, pp. 226-230.

[5] P. Ioannou, Z. Xu, S. Eckert, D. Clemons, and T. Sieja, "Intelligent Cruise Control: Theory & experiment", *Proc. of the 32nd IEEE Conference on Decision and Control,* San Antonio, Texas, Dec. 1993.

[6] P. Ioannou, M. Lai, J. Dickerson, and A. Kanaris, "Evolutionary Representative System Configurations and Roadway, Vehicle, Driver Functions", technical report, USC CATT, report 94-06-01, June 1994.

[7] L. A. Pipes, “ An Operational Analysis of Traffic Dynamics”, Journal of Applied Physics, 1953, 24, pp. 271-281.

[8] S. Sheikholeslam ,“ Control of a class of interconnected nonlinear dynamical system: the platoon problem”, ***Ph. D. Dissertation*** , University of California, Berkeley, 1991.

Appendix A: Deliverables and Publications

1. Software of Dynavis, Version 1.
2. Software of Dynavis, Version 2.
3. Software of Vehicle Mass Identification.
4. Software of Gas Mileage Visualization.
5. P. Ioannou and Z. Xu, "Throttle and Brake Control System for Automatic Vehicle Following", *IVHS* Journal, Vol. 1(4), 1994, pp. 345-377.
6. A. Kanaris, Z. Xu, and J. Hauser, "DYNAVIS: A Dynamic Visualization Environment for the Design and Evaluation of Automatic Vehicle Control Systems ", PATH technical Note, 94-8.
7. Z. Xu and P. Ioannou, "Adaptive Throttle Control for Speed Tracking", ***Vehicle System Dynamics,*** Vol. 23, No. 4, May 1994, pp. 293-306.
8. P. Ioannou and Z. Xu, "Adaptive Throttle Control for Automatic Vehicle Following", ***12th IFAC World*** Congress, Sydney, Australia, 1993.
9. P. Ioannou, Z. Xu, S. Eckert, D. Clemons, and T. Sieja, "Intelligent Cruise Control: Theory & experiment", *Proc.* ***of the*** *32nd* ***IEEE*** Conference ***on Decision and Control,*** San Antonio, Texas, Dec. 1993.
10. P. Ioannou and Z. Xu, "Throttle Control for Vehicle Following", 1993 ***IEEE Conference on Aerospace Control Systems,*** Westlake, California, 1993, pp. 226-230.
11. P. A. Ioannou, C. C. Chien & J. Hauser, " Autonomous Intelligent Cruise Control", *Proc.* ***of the*** *IVHS* ***America 1992 Annual Meeting,*** Newport Beach, CA, May 1992, pp. 97-112.
12. Xu & Ioannou, "Adaptive Throttle Control for Speed Tracking: Theory & Experiment", USC, CATT, Report, 92-09-01, Sept. 1992, also PATH working paper UCB-ITS-PRR-94-09.

Appendix B: User Manual of the Visualization Program

In the following we refer to the dynamic visualization program as DYNAVIS.

DYNAVIS is an interactive engineering environment developed specifically for the design and evaluation of automatic vehicle control systems. It can be run on any **Silicon Graphics computer.** It takes advantage of human's spatial intuition and transforms numerical data into an animation so that users can examine large amount of data with better comprehension and efficiency. Moreover, the capabilities of DYNAVIS go far beyond simple animation systems by providing a set of tools to perform interactive visualization with on-line modification of many visualization parameters such as time and space resolution.

The capabilities of this program greatly facilitate detecting undesirable phenomena in automatic vehicle following (platooning), designing and evaluating vehicle control systems, and comparing the performance of different control strategies. It is expected that DYNAVIS will be a valuable tool to designers of automatic vehicle following and platooning systems.

This manual describes how to use this program. The manual consists of two parts: (i) how to run DYNAVIS, (ii) how to modify the program.

(i). To Run DYNAVIS

Let DYNAVISDIR be the directory where the software is installed. DYNAVISDIR contains:

dynavisl	/** executable file for version 1 of DYNAVIS**/
dynavis2	/** executable file for version 2 of DYNAVIS **/
dynavisiden	/** executable file for parameter identification **/
dynavis-gas	/** executable file for gas mileage visualization **/
Doc/	/** document subdirectory **/
Data/	/** subdirectory for data files and parameter files **/
SRC_v1/	/** source code for version 1 of DYNAVIS **/
SRC_v2/	/** source code for version 2 of DYNAVIS **/
SRCiden/	/** source code for parameter identification **/
SRC_gas/	/** source code for gas mileage visualization **/

Depending on the application situation, some of these executable files and source code directories may be removed.

First change current directory to DYNAVISDIR. To start running version 2 program without changing default values, simply type

dynavis2

at the UNIX prompt. After this command is given, a logo will show up while waiting for the main program to appear. After a few seconds the outline of a window will appear. Use the mouse to move the outline window to any point on the screen. Click the left mouse button to place the window at the desired position. Drag the middle button on the title of the window to adjust the window's position after it is opened. At this point the program has been started and one should see the initial scene of a section of freeway and the vehicles on it.

To start running version 2 program with more options, type

```
dynavis2 [-v][-d][-h]
         [-n number of cars]
         [-f data-file]
         [-m memory size in 1000 frames]
         [-p parameter-file]
         [-t font]
         [-r ramp-file]
         [-c car-model-file]
```

These options are explained below:

-v: verbose mode.

In this mode, the program will show some information related to the execution of the program.

-d: debug mode.

This mode will show even more inside program information for the purpose of debugging. This mode is not particularly useful to the typical user running this program.

-h: help message.

This option will list the above options without running the program.

-n number-of-cars:

This option specifies how many vehicles are desired at the beginning of the simulation. The default is 4.

-f data-file:

This will set the program to read from the specified data file to re-play previously stored data. The data file format is as follows:

#cars t {time-headway}*#lanes {fixed-headway}*#lanes {CarDynamics}*#cars

where #cars is the number of vehicle for the frame at time t, #lanes is the number of lanes of traffic to be animated and is specified by the parameter 'Lanes' in the parameter file, and 'CarDynamics' is:

carID x y velocity-x velocity-y acceleration jerk

where 'carID' is an integer used to identify a vehicle. The #cars is normally equal to 'Lanes' x 'Cars' which is also specified in the parameter file. The data fields of 'time-headway' and 'fixed-headway' will be repeated #lanes times respectively. The first time-headway is for lane 1 (the top lane or the left most lane), the second time-headway (if #lanes > 0) is for lane 2 (the middle lane), and so on. The fixed-headway is similar. The data fields of 'CarDynamics' will be repeated #cars times. The first 'CarDynamics' is for the first car in lane 1, the second 'CarDynamics' is for the second car in lane 1, and so on up to the 'Cars'-th 'CarDynamics'. Then start the 'CarDynamics' of the first car in lane 2, the 'CarDynamics' of the second car in lane 2, and so on. The important thing is that all these data for one frame at time t must be in one line. The data for the next time instance will start in next line. These data fields are the maximum set of data fields. Depending on the parameter file, some data fields may be omitted. For example, if 'Ncars_given = 0' in parameter file, then #cars should be omitted. If 'CarID_given = 0', 'carID' should be omitted. If 'RelPosition_Y = 0', the lateral position y should be omitted. If 'SamePlatoonLeader= 1', the 'CarDynamics' for the first car in lane 2 and lane 3 should be omitted. Other data fields are similar.

-m number of 1000 frames:

This specifies the number of frames that is intended for the simulation. The larger the number, the longer the simulation time will be. Note that specifying large number may impede the performance since large memory may be required.

The default number of frames in each simulation is 6,000, which is about 5 minutes if the time step is not changed during the simulation. At the end of the simulation, if one wants to simulate for more than that time amount, it is suggested to save the result of each simulation to a file, delete the lines in the file so that only the last one or few lines remain, feed that file to the simulation as the initial states, and continue the simulation. Or alternatively, save the file and then rerun the program by specifying bigger number of frame and also read in the saved file to continue the simulation.

-t font:

Specify the font for displaying the text or numbers in the program screen. This option is usually not needed.

-p parameter-file:

Specify a parameter file to be used and tell the program what data fields are available. It is also called configuration file. This option is usually for animating data files generated by simulation of other programs or experiments. For data files generated by simulation with this program, this option is not needed. A sample of parameter-file is as follows:

Lanes= 1

Cars= 2

TimHdwyL1= 0.0

TimHdwyL2= 0

TimHdwyL3= 2.6

FixHdwyL1= 0.5

FixHdwyL2= 0.5

FixHdwyL3= 0.5

Time-Headway-given= 1

Fixed-Headway-given = 0

RelPosition_X= 0

RelPosition_Y= 1

Velocity_X= 1

Velocity-Y= 0

Accel-avail= 1

Jerk-avail= 0

EstimateAcceleration= 0

Estimate-Jerk= 0

Simulation= 0

SamePlatoonLeader = 1,

Ncars_given = 0

CarID_given = 0

The explanation of the items in the parameter files are as follows.

'Lanes' specifies how many platoons to be visualized (default is 2). It can be 1 , 2 or 3. Default is 2 for version 1 and 1 for version 2. 'Lanes' greater than 1 is normally used to compare the performance of several platoons each of which consists of the same number of vehicles. If the number of vehicles in each platoon or lane is different, 'Lanes' should be assigned 1 and then each vehicle's lane number is determined by its lateral position.

'Cars' specifies how many vehicles are in each platoon. This is mostly used in version 1 for comparing the performance of several platoons each of which consists of the same number of vehicles. When the number of vehicles in each lane is different, version 2 should be used and this parameter is the total number of vehicle. The default is 4 in version 1 and 8 in version 2.

'TimHdwyL1' specifies the roadway recommended time headway for lane 1, (and similar 'TimHdwyL 2' and 'TimHdwyL3' for lanes 2 and 3).

'FixHdwyL1' specifies the desired relative distance for lane 1 if applicable. The total roadway recommended headway is TimHdwyL * Velocity + FixHdwy.

'TimeHeadway-given' indicates if the actual time headway is given in the data file for each vehicle.

'Fixed_Headway_given' indicates if the actual desired relative distance is given for each vehicle.

'RelPosition_X' indicates if the longitudinal position given in the data file is relative (1 means yes, 0 means no). If 'RelPosition_X = 1', then the longitudinal position for the first vehicle of a platoon should be omitted form the data file. The program will estimate the longitudinal position based on the speed. This is actually what happens during a road test where the leading vehicle's position is normally not recorded.

'RelPosition_Y' indicates if the lateral position given in data file is relative to the center of a lane. If 'Lanes' is greater than 1 and 'RelPosition_Y=1', then the lateral position of a vehicle will be given in relative to the center of the lane where the platoon is positioned. If 'Lane = 1' and 'RelPosition_Y= 1' then the lateral positions of all vehicles are given in relative to the center of the top (first) lane. In all cases, if 'RelPosition_Y= 0', the data field for lateral position should be omitted from the data file. Users can introduce the flag variable 'Position-Y-given' in the program to handle more complicated situations. Note that when 'Lane' is greater than 1, the program will attach a vehicle to the platoon the vehicle is initially belong to even after it changes lane. Also note that the lane width used in program is 3.4 meters and the center of the middle lane has lateral position coordinator of -3.4 meters.

'Velocity-X' indicates if the longitudinal speed is given in the data file.

'Velocity-Y' indicates if the lateral speed is given in the data file.

'Accel-avail' indicates if the acceleration is given in the data file.

'Jerk-avail' indicates if the jerk is given in the data file.

'Estimate_Acceleration' specifies whether to estimate acceleration when it is not available in the data file.

'Estimate_Jerk' specifies whether to estimate jerk when it is not available in the data file.

'Simulation' indicates if online simulation will be performed.

'SamePlatoonLeader' indicates if the platoons in all lanes have the same platoon leaders (i.e. the platoon leaders have the same longitudinal trajectory). If it is equal to 1, then the data fields for the first car in lane 2 (and lane 3) should be omitted. Note that if the data-file offers the data fields for all the platoon leaders, 'SamePlatoonLeader' should be equal to 0 even though the platoon leaders have the same longitudinal trajectory.

'Ncarsgiven' specifies if number of cars per frame is given in the data-file.

'CarID_given' specifies if the car ID number is given in the data-file.

Users need to change the numbers in this parameter file based on particular situation. When a parameter file is specified, the users have to offer the corresponding data file.

-r ramp-file:

Specify a ramp information file. A sample of ramp file is as follows:

#Freeway		Inter 10 East	
# Name	Distance	On/off	Connector
Vermont Blvd	0.5	ON	
Hoover St	+0.5	OFF	
Washington Blvd	+1.0	ON	
#WEIRD Blvd	0.2	ON	
60 Freeway S	+0.6	OFF	connector
60 Freeway	+0.5	ON	connector
Santa Fe St	+0.5	OFF	

In this file the 'ON' and 'OFF' mean the on-ramp and off-ramp respectively, the distance 0.5 means 0.5 mile from the start, +0.5 means 0.5 mile away from previous ramp, and the line starting with # has no effect.

-c car-model-file:

Specify a car model file. A sample car model file is as follows:

# model	weight	length	width	speed	acc	brake	hp	engine
Ford Escort 4 (S):	2530	171	67	100	0.5	0.6	127	1.8
Lincoln Town Car V8 (L):	4055	219	77	150	0.6	0.8	210	4.6
Mercury Villager V6 (V):	3395	190	74	135	0.6	0.9	151	3.0
Nissan Sentra 4 (S):	2545	170	66	90	0.4	0.6	110	1.6

where acc and brake indicate the maximum acceleration and deceleration in g respectively. This file is not needed for animation.

Once the program is started, more information can be obtained from the online help.

Remarks

- Suppose the parameter file specifies 'Lane = 2', 'Cars = N', 'Position-Y = 0'. Then program will put the first N vehicles in the top lane, and put the next N vehicles in the middle lane. If the parameter file also specifies 'SamePlatoonLeader = 1', then data for the leading vehicle of the second platoon should not be offered in the data file and the program will automatically calculate the data for the leading vehicle of the second platoon based on the data for the leading vehicle of the first platoon.

- To run version 1 of the program, type

  dynavisl [-d] data-file parameter-file

  where -d is for debugging. In version 1, the data-file and parameter-file are necessary and they have the same format as in version 2. If for online simulation, the data file only needs one line to specify the initial conditions. Since version 1 can not handle variable vehicle numbers, 'Ncarsgiven' should be 0 in the parameter file and the #ncars data field should be omitted in the data-file. Version 1 also has online help which gives more information.

(ii). To Modify the Program

If users want to change the features of the program such as to implement their own vehicle dynamic models and control algorithms, they should read the whole source code carefully and have a fair understanding of the program first. There are a lot of comments in the source code which can help users to understand the program. Here we list some possible places that users may want to change and indicate how to change them. It is recommended the modification is based on the program of version 2. Thus what we described next is also based on version 2.

1. Change the initial states of the vehicles.

   This program automatically puts vehicles one after another evenly on 3 lanes of traffic. The initial states of the vehicles are set up by the program. If users wish to have different setting at the beginning of the program, they may want to change the program.

   The function that sets up the initial data for the vehicles is 'init_dynavis()' in 'input.c', which is called in 'main()' in 'dynavis.c'. The function 'init_dynavis()' reads in data if a data file is given or otherwise sets up the initial position, velocity, etc. of the vehicles by calling 'default_cardyn()' in the same file. The function 'default_cardyn()' is the place to change if different initial data are desired.

2. Add/Modify data structures for car status and flag.

   This program uses 2 data structures, 'CarStatus' and 'CarFeatures', to keep some specific and individual data for each car. The first one, 'CarStatus', is intended to keep some flags used for the internal program. The other one, 'CarFeatures', is to keep some physical data of the car. Users may have access to the data through online interface. Please see Source File 'carfeatures.h' for the specification of these two structures.

   If the users wish to add more fields in either of these two structures, there are a number of things to do:

   - Declare the new fields in 'CarStatus' and 'CarFeatures' in the 'carfeatures.h' file.

- Set the initial values for all the cars in 'setCarStatus()' and 'setCarFeatures()' if 'CarStatus' and 'CarFeatures' structures are modified. These two functions are both in 'carfeatures.c'.
- If the added fields in 'CarFeatures' are intended for the users to change online, there are more places to be changed:
  - In 'carfeatures.h', there is a 'car feature entries' list, including 'CARWEIGHT', 'CARVELOCITY', etc. Add a new name for each of the new fields right before the last entry, 'NCARFEATURES'.
  - In 'carfeatures.c', there is a 'car feature strings', which defines the string that is to be shown in the dialog window when users open it online and want to change the data. Add a new string label for each of the new fields after the last one IN THE SAME ORDER as they are added in the car feature entries list.
  - In 'applyeditCB()' of 'cardialog.c', add corresponding switch entries for the new fields, similar to those already there. Add 'case' for each new car feature entry. And in 'setParameter()', do the similar thing as above. The difference is: in 'applyeditCB()' the program is to read a new value from the text field, while in 'setParameter' it is to show the current value to the text field.

3. Add more changeable parameters in the parameters file.

   This program uses the structure, 'Platoon', to keep some flags which can be specified by the parameter file. This structure is defined in 'dynavis.h'. If users want to add more changeable parameters in the parameters file, they can do so by adding corresponding members in 'Platoon' structure and modify the function, 'readParamFile()' which is in and called by 'input.c'.

4. Add more things to visualize.

   This program can animate the position, velocity, etc. All these data are put in the structure variables indicated as 'FramePtr' type which is defined in 'dynavis.h'. If users want to visualize more things, they can modify the definition for this structure and put the new data to be displayed in this structure.

5. Change simulation routine.

   The main simulation routine is in the function

   simulation(FramePtr oldfptr, FramePtr newfptr, float cur-time, double timestep)

   which is in 'simulation.c'. This function calculates all the variable values for new frame structure based on the variable values in old frame structure, the current state and time step. Users can modified this function as long as they update all the members in structure 'newfptr'.

6. Change vehicle model.

   The vehicle model is called by function 'simModel()' which is in 'simmodel.c'. The function 'simModel()' updates the vehicle dynamic state based on throttle, brake, and steering wheel input. Users can substitute the model offered in this program with their own models. The state variables used in vehicle model are put in structure 'CarSimModel' which is defined in 'simmodel.h'. If users' models have more state variables, the users may need to set the initial values in 'setCarSimModel()' which is in 'simmodel.c'.

7. Change control algorithm.

   The control algorithm is implemented in function 'simControl()' which is in 'simcontrol.c'. The control algorithm calculates the commands for throttle, brake, and steering wheel. The calling tree of functions is: 'simulation ()' calls 'cal_CarDyn()' which calls 'simPackage()' which calls 'simControl()' and 'simModel()'. The function 'cal_CarDyn()' is in 'simulation.c' and 'simPackage()' is in 'simcontrol.c'. Users can substitute current 'simControl()' with their own control algorithms. Of course, the user should make necessary modification along this calling tree.

8. Change Xt/Motif resource: If users like to change the layout of the user interface, they can do that by changing either the resource files:

   Resource

   UserReource

   CarModel

   EditHill

   EditRamp

   or the fallback resource in 'resstring.c'

Appendix C: User Manual for Mass Identification and Gas Mileage Visualization

These two programs are the application of the main program of dynamic visualization. They have the similar setup as the main program of version 1 or version 2. Thus in this manual we only briefly describe how to use them.

(i). Run the mass identification demo.

To run the mass identification demonstration program, first go the subdirectory of DYNAVISDIR which contains the executable file 'dynavisiden', and then type

dynavisiden

or type

```
dynavisiden [-v][-d][-h]
            [-n number of cars]
            [-f data-file]
            [-m memory size in 1000 frames]
            [-p parameter-file]
            [-t font]
            [-r ramp-file]
            [-c car-model-file]
```

where the options are the same as in the main program, 'dynavis2'. However, the option [-n number of cars] is not used since the program is set to visualize four vehicles.

After a few seconds the outline of a window will appear. Use the mouse to move the outline window to any point on the screen. Click the left mouse button to open the window at the desired position. Use the middle button to adjust the window's position after it is opened. At this point one should see the initial scene of a section of highway and four vehicles in it. The first vehicles in both lanes have the same longitudinal trajectory. The second vehicle in top lane is the target vehicle whose parameter (mass in the demonstration) needs to be identified. The second vehicle in the middle lane is the imaginary vehicle whose mass is the estimation of the mass of the target vehicle. There are two color lines indicating the target mass and estimated mass. Initially, the two masses are set to be the same. To change the target mass, first pause the program through the 'command' menu, then click the right mouse button on the target vehicle and choose the 'Car Model' option. At this moment, the target vehicle mass can be edited. Or click the right mouse button on the target vehicle and choose the 'Car Features' option to change the mass (weight). The mass can also be changed without pausing the program. The method to change the mass for the imaginary vehicle is the same.

After the mass is set, one can continue the program, and use mouse to drive the first vehicle to suitable high speed. (This can be done by dragging the throttle slider bar.) When the both following vehicles are close to their desired positions, one can use mouse to apply the brake and the identification process will be turned on automatically. Then one can see how the estimated mass converges to the target mass. When the speeds become low, apply throttle to increase the speeds and the identification process will stop. When the speeds become high, one can apply brake and start the identification process again. In the 'Car Features' option there are several variables called userl, user2, ... which are user defined variables. In this application, 'userl' and 'user2' are used in the identification algorithms as two adjustable parameters. One can adjust them around the default values to increase the convergent speed. Since the identification algorithm needs smaller simulation time step, one can use larger simulation time step (about 0.05 sec.) when the throttle is applied and use smaller simulation time step (about 0.01 sec.) when the brake is applied. This can be done by moving the 'SIM-time-step' slider bar. Note that although one can virtually drive any vehicle, only drive the leading vehicle in this application. The manually driven vehicle is indicated by a small triangle in front of it.

(ii). To modify the identification program.

This program is developed only as a demonstration. For actual application, users may need to modify the program to implement their own identification algorithms. To modify the program, read Appendix B.

(iii). To run the gas mileage visualization program.

To run the gas mileage visualization program, first move to subdirectory, DYNAVISDIR, and type

dynavis-gas

or type

dynavis-gas [-v] [-d] [-h] [-g]
[-n number of cars]
[-f data-file]
[-m memory size in 1000 frames]
[-p parameter-file]
[-t font]
[-r ramp-file]
[-c car-model-file]

where the options are the same as in the main program, 'dynavis2' with two differences: (1) caraumber is either 2, 4, 6, or 8, (2) [-g] is a new added option which indicates if the gas

mileage data should be saved before exiting the simulation. The default in the program is 'no'.

After the window is opened, one can see the initial scene of a section of highway and the vehicles in it along with the meters corresponding to each vehicle. The top row of meters indicate velocities. Colored needles in the meters correspond to the cars of the same color. The middle row of meters indicate acceleration and the bottom row of meters indicate the gas mileage. There is an extra meter in the bottom row to indicate the average gas mileages of each platoon. After the program is started, the vehicles will follows some pre-determined speed profiles. Users can use the mouse to pull down the 'traffic' menu to select different traffic conditions. Users can also click the right mouse button on the leading vehicle, choose 'drive' option, and then apply throttle and brake to generate any desired longitudinal profile for the leading vehicle. Other user interface tools are similar to the main visualization program of version 1. Note that in this program the gas mileage takes the place of jerk in the main visualization program.

(iv). To modify the gas mileage visualization program.

To change the vehicle model and controllers, one can see Appendix B. The longitudinal vehicle model which contains gas consumption but does not contain the brake model is in 'gasmodel.c'. This model can be changed independently as long as the model gives outputs for vehicle speed, acceleration, and gas consumption. The algorithm which actually calculates the gas mileage is in function 'simModel()' in 'simmodel.c'. Users can modify the algorithm there. To change the speed profiles for different traffic conditions, simply modify the functions in 'roadway.c'.

Appendix D: Video Tape Demonstration and Source Code

We have created a video tape which shows an overview of the capabilities of the current version of dynamic environment and a sample session of using and interacting with DYNAVIS. The source code of the program is in the tape enclosed. The source code can also be obtained through ftp. For a copy of this video tape or to know how to ftp the source code, please contact the authors at the University of Southern California, Center for Advanced Transportation Technologies.

Phone: 213-740-4452

email: dynavis@talon.usc.edu, or dynavisQfalcon.usc.edu