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

# Longitudinal State Estimation for a Four-Vehicle Platoon

**A.W. Merz**

**California PATH Research Report**
**UCB-ITS-PRR-95-27**

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.

August 1995

ISSN 10551425

# LONGITUDINAL STATE ESTIMATION FOR A FOUR-VEHICLE PLATOON

A. W. Merz

Lockheed Missiles and Space Co.
Palo Alto, CA 94208
July 1, 1994

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.

## ACKNOWLEDGMENTS

**The author of this report was assisted by U. C. Berkeley Ph.D. candidates V. Garg and D. V. A. Swaroop, who were contributors to the PATH report by K. Hedrick, et al., cited in the references. Lockheed Missiles and Space, Inc. personnel L. Gordon and H. Peñafiel were helpful with respect to the C-code program provided to the author by U. C. Berkeley, and consultant D. Johnson developed computer graphics to illustrate the numerical results.**

## ABSTRACT

Longitudinal State Estimation for a Four-Vehicle Platoon

A. W. Merz

The estimation of longitudinal states in a four-vehicle platoon is derived, disc us sed and illustrated numerically. The general procedure in the process is the use of dynamic equations and data, for finding the estimates and root mean squared errors in the estimates of the states for each vehicle. Both dynamics and data are influenced by additive noise. The data can be any or all of the states, including the position and velocity relative to the preceding and lead vehicles. The nonlinear dynamics of the platoon are those in the computer code developed by U. C. Berkeley. Several additional parameters require numerical specification, including data and process noise levels. The control algorithm is applied to the estimates rather than to the actual states in the four-vehicle platoon.

The motivation for the filter is that its output includes estimates and root mean squared errors. The presence of a significant vehicle failure can be detected by data which is statistically far from its predicted value, as measured by a multiple of the root mean squared error. The prior estimates and their covariances change during a brief transient to the steady-state values, which depend on the data and its root mean squared errors. These errors are in addition to the dynamic errors which are present during the interval between successive data. The estimation error can be significant, both before and during the steady-state. The Kalman filter gives estimates which are both consistent and accurate, for use in the control algorithms, and for use in failure detection. The filter results are illustrated in a number of example cases.

Keywords: Automatic Braking, Brakes, Collision Avoidance Systems, Control Systems, Data Communication, Ranging Sensors, Traffic Platooning, Vehicle Dynamics

## EXECUTIVE SUMMARY

Representative studies of the Intelligent Vehicle Highway System as a high-dimensional control problem usually acknowledge the difficulty of the problem. This difficulty is due to both the complexity and the nonlinearity of an individual vehicle, and to the control aspects of the multi-vehicle platooning concept. Earlier work by various researchers indicates that platooning is feasible, but these conclusions vary quantitatively upon the dynamic and control models assumed. At present there is little parametric data describing the individual vehicles and sensors, so there may be significant errors present in some the component values assumed. Because good control performance and failure identification require accurate estimates of the states, it is essential that the relationhip between the data and its estimates be found, and this is the purpose of the Kalman filter.

The study begins by reviewing the statistics of automobile accidents, for which a large amount of data are available. These data show that only a small percentage of accidents are due to mechanical failure. The work here is then focused on the analysis of the existing simulation code for purposes of adding a filter. The control algorithm used for maintaining speed and separation between vehicles requires the combining of distances, velocities and accelerations to develop command brake and accelerator displacements. Up to the present, exact values of these states have been used in the control law simulations. These will not be available in practice, and the reduction in performance due to the errors in the estimates must be found.

The Kalman filter requires an extraordinary amount of computation, because the equations of motion are given in terms of tabular data, and because of various other nonlinear effects. The linearization needed for the integration of the matrix covariance equation implies recomputation of the transition matrix for each vehicle at each data time. The filter development for the longitudinal motion of the platoon has been concluded, and numerical results concerning several topics are shown. Extension to higher-ordered dynamic models will be relatively straightforward, if not easy.

Any emergency condition caused by a vehicle component failure generates data sequences which are beyond statistically credible bounds; e.g., three-sigma limits. When this condition occurs for a vehicle, a component failure has changed its dynamic characteristics. The intent of this research was the specification of maneuvers in emergency conditions. But the complexity of the dynamic model and the calendar constraints restricted the study to the development of the optimal filter. Identification of the emergency condition requires use of a computer program equivalent to that developed here.

## TABLE OF CONTENTS

# LONGITUDINAL STATE ESTIMATION

# FOR A FOUR-VEHICLE PLATOON

A. W. Merz
Lockheed Missiles and Space Co.
Palo Alto, CA 94803
July 1, 1994

## 1. FAILURE MODES

### 1.1 Introduction

Over the past several decades, the automobile and the system of highways of this country have been the subject of many statistical studies. The failures of automobiles and their drivers, and the loss of property and life caused by these failures, have been studied by a variety of agencies, including insurance companies, manufacturers of automobiles, and those concerned with public safety. The Intelligent Vehicle Highway System (IVHS) is a logical extension of the manually-controlled automobile system, which has been developed to alleviate traffic congestion problems in and near civic centers. This system has the potential of greatly increasing the traffic density on our freeways, by combining high-speeds and the sensors and other system components which make up the inter-vehicle control system. It is important that all sources of potential failure in these components of the IVHS be considered, so that realistic estimates of the probabilities of success are possible, because many of the sensors, controllers and algorithms are new and as yet untested.

Automobile drivers in the United States concede that these vehicles have reached a level of mechanical reliability which is nearly perfect, in that vehicle breakdowns represent a very small fraction of the total abount of vehicle traffic. For this reason, the great majority of automobile accidents are due not to mechanical failure, but to the driver and his control inputs Since the platooning of vehicles as envisioned in the IVHS implementation involves a combination of vehicle and control system dynamics, present knowledge suggests that any failures or breakdowns will be mainly due to failures in the component features devised for these systems. A tabulation of such candidate failures will be given in Section 1.3.

## 1.2 Statistics of Highway Vehicle Incidents

Appendix 1 is a listing of a number of agencies concerned with highway and automobile safety, including sources which have volumes of statistical data on these topics. Treat (1979) is one of many detailed studies and analyses of highway incidents obtained from one of these sources. This reference applies to the five years prior to 1977, but while fifteen years old, the qualitative conclusions of the report ar'e supported qualitatively by the independent studies of Finch (1971), Grandel (1985), Hatch et al. (1977) and Schmidt et al. (1974). The principal finding of these studies is that human error is by far the most frequent cause of vehicle accidents. Probable cause results are summarized by Treat as human factors 93%, environmental factors 34% and vehicle factors 13%. In this tabulation, accidents were considered to have more than one cause. Other probable causes were view obstruction 12%, slick roads 10%, transient hazards 5%, design problems 5% and control hindrances 4%. Brake systems and tires were probable causes in 5% and 4% of accidents, though "blow-out" corresponded to only .05% (1 occurrence) of the data base of 2,258 accidents. More frequently, small tread depth (6) and vehicle lights (7) were cited as subsidiary causes of accidents.

The most important conclusion in these sources is that human factors are the usual cause of a road accident. This implies that the condition of road or vehicle failures are only very rarely the cause of an accident, and that the driver should be able to compensate for these faults, the visibility constraints, automobile limitations, etc., by driving circumspectly. This fact has caused one of the projected tasks of the present study to be modified, from a ranking of causes of accidents to a focus on the estimation problem, which precedes the identification of a cause and the associated control response.

The IVHS study topic allows ignoring the "view obstruction" and "design problems" causes, but otherwise we have learned that slick roads, brake system failures and tire failures are the most probable mechanical and environmental causes of accidents. Nevertheless, they amount to a small minority of the causes, as noted above. Most of the cited references deal with "accidents," and the data is not necessarily applicable to automobile "failures." Such events as an engine failure which does not produce an accident would not be reported here. But, in fact, the same failure which does not cause damage in the single-vehicle case, if the driver is able to remain near the roadway, can cause a multi-vehicle accident in a platoon system. For this and other reasons, the sources of failure in the IVHS will depend on the new components which are designed and built for platoon system implementation.

## 1.3 IVHS Vehicle Failures

The Intelligent Vehicle Highway System has been studied and simulated by many sources over the past several years. Mechanical components of the system are being analyzed for their effectiveness in low-order platooning experiments in mid- 1994. At the present time, control laws are being devised for testing under laboratory conditions, and it is premature to consider effects of failures on the system performance. But such failures must be examined, and this section of the report discusses and quantifies these failures.

The IVHS couples a large number of mechanical and computer computer components, involving both hardware and software. All of these components can deviate excessively from their hypothetical performance, leading to an unexpected coupling, consequent unacceptable performance and failure of the system. It is the responsibility of the analyst to determine such deviations by simulation, and to find the best estimates of the performance under plausible assumptions regarding the components capable of failure.

The sources of such failures are listed in Table 1, with representative examples. This list illustrates the general causes of failure, but is not explicit because the final form of the system and its components have not yet been specified.

Table 1 - Possible Failure Sources in IVHS

1. State sensor (e.g., of rpm, position, speed) failures
2. Control system component (e.g., brake, steering) failures
3. Algorithm (e.g., excessive simplification) failures
4. Computer (e.g., coding, limitations to speed) failures
5. Environmental modelling (e.g., ice, fog, highway) failures
6. Operational (e.g., fuel, tire, cooling system) failures
7. Dynamic modelling (e.g., order, nonlinearities) failures

The indicated headings may be coupled. For example, a control system failure may be due to an environmental cause, and a dynamic modelling failure may be due to an operational cause. These headings merely suggest the large number of sources of failure.

The platooning of automobiles involves many components which have never been used before in the manner required. Component failure rates are unknown, because they have not yet been built or tested in sufficient quantities. But whatever the failure or cause for emergency control, estimates of each vehicle's state are needed. These estimates may have to be as accurate as the data allow, because intervehicle spacing should be accurately controlled, and because root mean squared errors are helpful in defining failures. This implies the use of a minimum variance or Kalman filter.

The output of the filter is used for identification of failures by the combined use of estimates and uncertainties. An emergency has occurred when a state takes a value which is sufficiently different from the current estimate. This difference is measured as a multiple of the root mean squared uncertainty. That is, since the dynamic equations include all sources of disturbance apart from such emergency or failure causes, the estimates must imply a nominal data vector and its uncertainty. When the data sequence is "extremely" far from the expected or predicted value, a failure is assumed to have occurred, and whether the failure is due to mechanical breakdown or to a broken sensor is a question which can be answered with enough redundant data. Successive data of the same kind raise the probability of correct identification of the cause of this failure, but the definition of failure is in terms of the root mean squared errors. Speed of identification is also an important characteristic.

For the conjectured platoon configuration, these failures can be grouped under four headings, for motion in two dimensions,

1. Longitudinal data (engine, actuators, sensors, software)
2. Longitudinal control (engine, brake, software)
3. Lateral data (steering, sensors, software)
4. Lateral control (steering, tires, software)

The approach taken here is to develop estimates of states by coupling the dynamic equations of the estimate in the same way as the actual state components are coupled. For any mathematical model of the dynamics, which includes table look-ups, saturations, dead-zones and other nonlinearities, the Kalman filter is the "best" way of coupling the states and the estimated states through the data, in that the root mean squared error in the estimate is a minimum. The motivation for this coupling is that, at any time, the control used is a function of the state estimate. This estimate depends on the noisy data, which is itself a function of the actual state. The measurement process introduces noise, so that no state can be precisely known, and the mean-square error in the estimate is also updated at each data reading, according to the covariances in the data and in the state estimate.

The important requirement in the development of this filter is that the ***error*** in the estimate of any state must be small enough that a linear equation describes its time variation between data-updates. The state itself can vary with time in any manner, and the engine, pump, and tire dynamics include several nonlinear processes. So, the basic dynamic vector equation for the state x is nonlinear, of the form,

$$\dot{x} = f(x,u,q), \qquad (1)$$

where the control vector is $u$, and $q$ represents random noise inputs. The dynamics of the estimated state $\hat{x}$ follow the same equation, but without the random uncorrelated noise $q$,**which** is presumed to have an average value of zero, and so cannot be predicted. The error or difference between the derivatives of state and its estimate is written as the following linear function or derivative of $dx$,

$$d\dot{x} = f(x,u,q) - f(\hat{x},u) = (\partial f / \partial x)dx + (\partial f / \partial q)dq \qquad (2)$$

This is equivalent to a difference equation, corresponding to a sample time $dt$,

$$dx(n+1) = F\ dx(n) + G\ q(n) \qquad (3)$$

where $F$ and G are matrix functions of the current state estimate and the time-step interval. The **covariance** equation uses these matrices for the time-update, which means that the dependence of the error has the small-perturbation format of **Eq.(3).** The error in the approximation in general can be determined only numerically, as discussed in Section 2.

## 1.4 Detecting Vehicle Failures

Because of the short response times characteristic of the operational IVHS, it is obviously essential that any significant vehicle failure be determined as soon as possible. A significant failure is a "large" and "rapid" departure of one or more of the vehicle states from the nominal operating condition. When the effect of a failure is large enough, such a failure is detectable by statistical means. That is, when an estimated state is "unreasonably" far in value from its predicted value, relative to the current rms uncertainty, it is because the actual state is inconsistent with the controls applied to it, and other forces are present in the dynamics. A simple example failure illustrates the basic approach taken in this study.

When a discontinuity occurs in the dynamic equations, indicative of a abrupt departure of the system from the nominal operating condition, a failure is the likely cause of the discrepancy. If the estimated engine rate, the rms uncertainty in the estimate and the current commanded accelerator displacement are inconsistent with each other, because the engine rpm has dropped by 20% for no apparent reason, the failure identification algorithm should recognize this failure. The ingredients of this identification task are the estimated value and its rms uncertainty, both based on the minimum rms error or Kalman filter output, which couples the control inputs, the dynamic equations of the estimated state, and the data, with the rms errors in the current estimates and the data. The dynamic equations have been developed by researchers at U.C. Berkeley, but the estimated system dynamics and the statistical features of this estimate have not been examined until now. The following section discusses the Kalman filter, and how it is applied to the failure-detection problem.

# 2. KALMAN FILTER DEVELOPMENT

## 2.1 Introduction

The derivation of the Kalman filter will be shown in some detail, for the longitudinal coupled motion of a four-vehicle platoon. The lead vehicle operates independently of those behind it, and its perturbations from idealized motion are included in the platoon dynamics. But the lead vehicle does not use a filter to estimate its state, because its data and control are considered to be sufficiently accurate, particularly since its speed control system is independent of its spacing from the other vehicles.

Subsequent vehicles in the platoon have longitudinal motions which depend on the control accelerations input by accelerator and brake, which depend on the motion of vehicles ahead of it, and on external inputs due to lead vehicle motion, road slope and surface imperfections, wind, and other perturbations. The controls depend on the vehicle estimates of the associated state components. The filter can be expressed in terms of the state components of individual vehicles, with external noise.

While all of the states of each vehicle are estimated, each vehicle need be concerned only with its own seventh order state, and the control inputs of both deterministic (feedback accelerations) and stochastic (zero-mean accelerations). The data for each vehicle can be any subset of its seven states (i.e., from one to seven quantities each sample time, with appropriate covariances), augmented by the range and range rate with respect to both the preceding vehicle and the lead-vehicle. Since vehicle position is a function of all of the other six states of the vehicle being controlled, its estimate will be improved by any data on the other states.

As many states as possible should be measured, and in the general case, presence or absence of data can be implied by the rms error in this data, so that when a state is not actually measured, its data uncertainty in the filter code is set at a large value. This implies that the associated “data” is worthless, but the simulation code is written as if it were present, for use if and when the uncertainty is reduced to a useful level.

The state estimates and their rms errors are output by the Kalman filter. The filter has the purpose of developing minimum-variance estimates from noisy data, while accounting for the dynamic noise present between data points. The “quality” of the data is implied by the variance in the data noise, so that, when the data is of high accuracy, relative to that of the prior estimate, the uncertainty is sharply reduced at the time of this data. If driving noise is present, this ***means*** that the next data will have a smaller effect on the uncertainty. On the other hand, if all data associated with a specific state is either absent or of poor quality, the associated covariance is not affected by this data. The eventual reduction in uncertainty of this state is then a function of the dynamic equations linking it to the covariances of the other states. In this way, the velocity can be estimated from successive position estimates, even when velocity data is absent. Coupling also occurs among the states in less obvious ways, as implied by the equations of motion.

## 2.2 Longitudinal Dynamic Equations

The dynamic equations for a single vehicle in the platoon are rather complex, even when only longitudinal dynamics are considered. A seventh-order system has been defined by Hedrick, *et* al. (1994), and the analysis in this Reference is relevant to both the filtering and control problems. The important assumption made here with respect to the filter development is that the estimated state and the actual state follow the same nonlinear dynamic equations, with the difference between them modelled as zero-mean driving noise. The listings in Appendix 2 show this as sequential calls to the numerical integrator subroutine ***ruk,*** in the main program ***long-sim.c.***

The states of a vehicle in the longitudinal degree of freedom are the following:

| | |
|---|---|
| *mair* | Air mass in intake manifold, kg |
| *weng* | Engine speed, l/s |
| $vel_x$ | Longitudinal velocity, m/s |
| $pos_x$ | Longitudinal position, m |
| *wwhl* | wheel speed, l/s |
| *alpha* | Throttle (accelerator) angle, deg |
| $T_{brk}$ | Brake torque, N-m. |

The input longitudinal control vector components are effectively those available to the driver of any vehicle; i.e., the control vector has the components,

| | |
|---|---|
| $alpha_c$ | Command throttle angle, deg |
| $T_{brkc}$ | Command brake torque, N-m |
| *rgear* | Gear reduction ratio |

The third control can take only four discrete values in the Berkeley code, and it has been ignored in the results to follow, because the limited scope of this study means that the speeds of the vehicles remain in a narrow band, making gear changes unnecessary. On the other hand, the tight control required over the speed makes the brake and throttle angle commands very nearly simultaneous. When the time-lags of brake and throttle controls are accounted for, the actual values of brake pressure and accelerator angle input occur together. This is not a desirable, efficient, or economical way to control a vehicle, but it is the effect of the tight performance requirements of the platoon position control system.

The time derivatives of the seven states of any vehicle are coupled, in the following ways:

$$
\begin{aligned}
&m_{air} = f_1(m_{air}, w_{eng}, alpha) \\
&weng = f_2(m_{air}, w_{eng}, w_{whl}, alpha) \\
&vel_x = f_3(vel_x, w_{whl}) \\
&pos_x = vel_x \qquad (1) \\
&wwhl = f_5(w_{eng}, vel_x, w_{whl}, T_{brk}) \\
&alpha = f_6(alpha, alpha_c) \\
&T_{brk} = f_7(T_{brk}, T_{brkc})
\end{aligned}
$$

The command or control inputs are the accelerator deflection and brake torque, $alpha_c$ and $T_{brkc}$, which propagate to actual deflection and torque, and which then influence the derivatives of air-mass, engine speed and wheel speed. The commands are derived from the command acceleration, which is a linear function of positions, velocities and accelerations of prior vehicles. The velocity rate changes due to the change in wheel speed, and finally the position rate changes due to the velocity change. This position rate is modified by four successive integrations of control changes; i.e., the transfer function relating position to accelerator change is of fourth order. Now it is necessary to show how the accelerator and brake torque controls depend on the states.

The relative motion of vehicle 1 with respect to the lead vehicle has the order of the control system, and is modelled by the following longitudinal acceleration (the gain-notation of Chang, 1993, is used here),

$$a(l) = q2\,\dot{\hat{\varepsilon}}(1) + q1\,\hat{\varepsilon}(1) + ka\ a_L + cp[\,pos_{xL} - (Lcar + Dsafe) - \hat{pos}_x(1)]$$

$$+q3\,[\,vel_{xL} - \hat{vel}_x(1)] + cf\ a_L\,, \tag{2}$$

where

$$\hat{\varepsilon}(1) = pos_{xL} - (Lcar + Dsafe) - \hat{pos}_x(1)\,,$$

$$\dot{\hat{\varepsilon}}(1) = vel_{xL} - \hat{vel}_x(1) \tag{3}$$

For the second controlled vehicle, the control is the same as for all succeeding vehicles, and the control acceleration is the sum of six independent terms. Three are due to the motion of the preceding vehicle, and three are due to the lead vehicle. This command acceleration for the *n*th vehicle is:

$$ii(n) = q2\,(n)\,\dot{\hat{\varepsilon}}(n) + q1\,\hat{\varepsilon}\,(n) - ka\ \hat{a}(n\text{-}1) + cp\,[\,pos_{xL} - n(Lcar + Dsafe) - \hat{pos}_x\ (n)]$$

$$+\ q3[\,vel_{xL} - \hat{vel}_x(n)] + \ cf a_L \quad , \tag{4}$$

and implementation of this control law requires that estimates be available for the distances and their derivatives. The optimal estimates use all of the data, accounting for coupling between states.

The position transfer function of vehicle *n* with respect to position of the lead vehicle and the prior vehicle is

$$X(s) = \frac{c_f s^2 + q_3 s + c_p}{D(s)} X_L(s) + \frac{k_a s^2 + q_1 s + q_2}{D(s)} X_P(s) \qquad 5)$$

where the characteristic function in the denominator is quadratic:

$$D(s) = s^2 + (q_1 + q_3)s + (q_2 + c_p). \tag{6}$$

An additional term *(q₄/s)* would be added to *D(s)* if the error integrator were included. The stability characteristics are not visibly different when this term is included, and it is ignored in the results to be shown in the following section of this report.

The modifications to the code needed for filtering concern noisy data and the statistical properties of the estimate. Its time-update is given by the same differential equations describing the actual state, and its data update uses the current covariance of the state with that of the data. The covariance matrix at any time describes the correlation of the error vector $dx$, which varies with time according to Eq.(3) of Sec.l.3. This error vector has the expected value of zero. The covariance matrix is estimated by using the computer code in special ways to be described. The post-data covariance at any time is used with the mean-square disturbance noise and data noise to update the covariance at the next data time. The interval between data readings allows the covariance to grow with time, according to the system dynamic equations, while the data causes a discontinuous reduction in covariance. The relative magnitudes of the positive and negative changes over a data cycle may yield an increase or a decrease in the mean-square error, depending on the current state, data and parameters. The post-data covariance variation will be illustrated numerically in Sec. 3.5.

The initial value of the covariance need not be accurately known, since it will typically initiate a brief (5 s to 10 s) transient before settling to a final value, assuming that the dynamic parameters (equations of motion, driving noise levels, etc.) and data parameters (e.g., sample interval, data noise levels) are constant. This constant-parameter assumption has been made in the numerical results to be shown. Since the data components are assumed to be samples of individual states (rather than nonlinear functions of states) any parameter modification has an apparent effect on the steady-state covariance. Reducing the time interval between samples must reduce the covariance, for example, and raising the driving noise level must increase it. Typically, however, only the sign of the change is known, and an additional use of the simulation is finding the ***size*** of this change due to an assumed parameter variation. This is incidental to the failure-detection problem.

The time update of the covariance is found numerically, by examining the columns of the transition matrix F, defined by the discrete state equation,

$$dx(i+1) = F\,dx(i) + G\,du(i) \tag{7}$$

where $dx$ and $du$ are the differences between actual and estimated state and control vectorts. The vectors themselves have the following components

$$\begin{aligned} x &= [m_{air}\ w_{eng}\ \ vel_x\ pos_x\ w_{whl}\ \ alpha\ \ T_{brk}]' , \\ u &= [alpha_c\ \ T_{brkc}\ \ R_{gear}]' \end{aligned} \tag{8}$$

The *F* and G matrices in Eq.(7) are determined by close examination of the U.C. Berkeley code named long_sim.c. The equations of motion are complex, and linearization to the form of Eq.(7) requires that the matrices be estimated numerically at each time. The matrices have the form shown below, where the asterisk * denotes a nonzero element in the matrix:

$$F = \begin{bmatrix} * & * & 0 & 0 & * & * & 0 \\ * & * & 0 & 0 & * & * & 0 \\ 0 & 0 & * & 0 & * & 0 & 0 \\ 0 & 0 & * & * & 0 & 0 & 0 \\ 0 & * & * & 0 & * & 0 & * \\ 0 & 0 & 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & * \end{bmatrix}, \qquad G = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & * \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & * \\ * & 0 & 0 \\ 0 & * & 0 \end{bmatrix} \qquad (9)$$

These non-zero terms are found by numerical methods. The terminology *F(1,2)*, for example, is the partial derivative of the first state $m_{air}(i+1)$ with respect to the second state, $w_{eng}(i)$. This is approximated numerically by making a small change in $w_{eng}(i)$ and noting the change given by the transition matrix to the updated value of $m_{air}(i+1)$. The ratio is then an approximation to the partial derivative,

$$F(1,2) = \partial m_{air}(i+1)/\partial w_{eng}(i) \qquad (10)$$

$$= [\dot{m}_{air}(w_{eng} + dw_e) - \dot{m}_{air} w_{eng}] / dw_e dt$$

This computation cannot be performed analytically, since the relation between the states is specified only in terms of tabular data for a specific vehicle at a specific time. An exception occurs with the accelerator and brake controls, which are assumed to depend linearly on the command value and the current value of the accelerator deflection and brake pressure. The corresponding matrix element is then known analytically.

From an equilibrium condition, a change in command accelerator position has the following sequential effects, as seem by the non-zero terms of *F* and G (each step corresponds to an increase in dynamic system order):

1. ***alpha-c*** changes *alpha*

2. ***alpha*** changes $m_{air}$ and $w_{eng}$

3. $w_{eng}$ changes $w_{whl}$ and $vel_x$

4. $vel_x$ changes $T_{brk}$ and $pos_x$

The forward velocity dependence on command accelerator position is of third-order, and the position dependence is of fourth-order. This order is the same as the number of the line on which the variable first appears; $vel_x$ first appears on line 3, etc.

The same equation updates the estimated state and the actual state, so that the difference or error $dx(i)$ between actual and estimated state at time $i$ follows the vector equation,

$$dx(i+1) = F\,dx(i) + G\,q(i), \tag{11}$$

if the transition and control matrices have the same numerical values for both the state and its estimate. The noise $q(i)$ represents the difference between the commanded and the actual control inputs, and it can also represent external noise signals not otherwise represented in the equations. An example is bumps in the highway surface, or wind gusts, both of which cause small deviations in the speed not present in a simplified dynamic model. It is assumed to have zero mean value in the analysis (and this assumption can be verified or tested), but its mean-squared value should be specified as large enough to give a conservative estimate of the performance. The covariance $V(i+1|i)$ is the expected value of the outer product $dx\,dx'$ at the time $i+1$, based on the data to time $i$. With Eq.( 11) this is derived as

$$V(i+1|i) = F\,V(i|i)\,F' + GQG' \tag{12}$$

where Q = E[$qq'$] is the covariance of the driving-noise vector, assumed known. As noted earlier, the noise vector can have more components than the control vector u. In the present simulation, $q$ is partly due to differences between actual and commanded deflections in accelerator and brake, and partly due to external sources; e.g., accelerations induced by road surface and wind The resulting time-update in Eq.( 12) is equivalent to a increased uncertainty between data points, which will be at least partly reduced again at the next data time.

The data-update of the covariance and of the state makes use of the data as a sequence of scalars at the same clock time. For each such scalar data, the covariance is reduced according to

$$V(i|j) = [I - K(j)\, h(j)']\, V(i|j-1)\,. \tag{13}$$

Here, *(i|j)* abbreviates "at time ***i*** due to measurement *j*." The filter gain is a function of the prior accuracy of the covariance ***V(i)*** and the mean-square error in the data ***rr(j)***;

$$K(j) = V(i)\, h(j) / [h(j)' V(i)\, h(j) + rr(j)]\,. \tag{14}$$

This is a vector, typically equal to one of the columns of the covariance matrix, since the ***h*** vector is made up of zeros, except for one element equal to 1 or -1.

It is assumed that the data available for controlling the longitudinal position is the nine-component vector formed of the separation, its rate, and the seven individual states for that vehicle;

$$y = [e \quad \dot{e} \quad m_{air} \quad w_{eng}\ pos_x \ \ vel_x \quad w_{whl}\ \ alpha \ \ T_{brk}]' + r \tag{15}$$

where, for vehicle ***i,***

$$e(i) = pos_x(i-1) - (L_{car} + D_{safe}) - pos_x(i) \tag{16}$$

$$e(i) = vel_x(i-1) - vel_x(i) \tag{17}$$

The data vector here involves the ***exact*** states, plus noise, while the control vector in Eq.(2) and (4) uses estimates for the state. Since the data is a linear process, it can be written as

$$y = Hx + r \quad , \tag{18}$$

where the ***H*** matrix is a set of row-vectors denoted ***h,*** each having one non-zero element, and the vector ***r*** is the error in the data due to measurement uncertainty. This matrix is independent of the state, because each component of the data is proportional to a specific state component, and there are no computations required in determining it. An advantage of the present formulation is that no matrix inversion is needed in Eq.(14), since the denominator ***h'Vh + rr*** is positive; it is the sum of diagonal elements of the covariance of a state estimate and of a data. The constant elements in the gain definition are irrelevant, since the data update of the estimate deals with the difference of actual and predicted data, so the constant terms cancel; i.e.,

$$\hat{x}(i|i) = \hat{x}(i|i-1) + K(i)[y(i) - j(i)] \tag{19}$$

Here the time index is $i$, and the optimal gain $K(i)$ is given in Eq.(14) in terms of the covariances of the state and of the data. This vector-matrix equation is a more compact version of the data-update, since here the gain matrix has the number of columns equal to the number of independent data (here assumed to be nine).

Notice that the steady-state condition of the filter may justify constant gains in the data-reduction process, which would eliminate a large amount of on-line computation in the filter implementation. Whether this approximation is valid will be examined in Sec. 2.5.

The components of the data vector are not specified in terms of specific sensors, but a paragraph from p. 57 of Hedrick, et al. (1994) is relevant here:

> "Differential vehicle information test data was available using a sonar system mounted at the centers of the front grills of the vehicles. The sonar system was capable of determining range . . . Soon to be available are a radar system and an optical triangulation system. The radar system will provide range, closing rate, and absolute vehicle speed."

Earlier PATH reports indicate that bias in the radar data can be significant, but the only apparent method of correcting for this bias is through use of centerline-sensors used by each vehicle for lateral data. This data is not simultaneous for the two vehicles and does not appear to be useful for this purpose. On the other hand, sonar and optical system data characteristics may have more nearly zero-mean error. If these are also not expensive, either one is suitable.

## 2.3 System Matrix Computation

The dynamics and control matrices in Eq.(7) of the prior section are updated with time. At any time, the code with defines the derivatives of the states (Appendix 2) shows that the acceleration is

$$vel_x = (f_{tractive} - C_a\, vel_x^2 - F_{roll\text{-}frict} + a_{lon})/MASS, \qquad (1)$$

where

$$f_{tractive} = Kt\ slip$$

$$slip = 1 - vel_x / (Hr\, w_{whl}) \qquad |slip| <= 0.15$$

$$F_{roll\text{-}frict} = \text{constant} = 187\ N\text{-}m,$$

and where $a_{lon}$ is the longitudinal acceleration bias due to the local slope of the roadway. This component of local gravity depends on thelocations of the vehicles relatgive to the slope changes in the highway. Here these changes are specified to occur discontinuously at input positions along the highway. Winds and road roughness are disturbances to be added to these equations.

Since x(3) = $vel_x$ and x(5) = $w_{whl}$, it follows that two of the elements of the transition matrix are

$$F(3,3) = 1 + (\partial vel_x dot / \partial vel_x) dv_x dt$$
$$F(3,5) = (\partial vel_x dot / \partial w_{whl}) dt, \qquad (2)$$

and the approximations to the derivatives are found as

$$F(3,3) = 1 + (\dot{velxpert} - \dot{vel_x}) / dvx\ dt,$$
$$F(3,5) = (vel_{xpert} - vel_x) / dww\ \ dt \qquad (3)$$

The individual perturbations are found using modifications to the current states, with the perturbation corresponding to the individual column of the transition matrix. For example, a perturbation in vehicle speed occurs in column 3, while column 5 relates to a perturbation in wheel speed. The perturbations **$dvx$** and **$dww$** allow computations of the perturbed derivatives of $vel_x$. This procedure is used for the **first** five columns of the matrix. The last two columns describe the simpler dynamics of the accelerator and brake, and no approximation is needed for the last two diagonal terms;

$$F(6,6) = exp(-dt/TAU\_THROTTLE)$$
$$F(7,7) = exp(-dt/TAU\_BRAKE), \qquad (4)$$

in the notation of the U. C. Berkeley computer program, long-sim.c. The control vector is accelerator deflection and brake pressure, so the G-matrix gives the effect on the updated state of the current control. As implied by the previous discussion, these elements are

$$G(6,1) = 1. - exp(-dt/TAU\_THROTTLE)$$

$$G(7,2) = 1. - exp(-dt/TAU\_BRAKE), \tag{5}$$

where the time constants ***(TAU)*** have values 0.011 s and 0.0756 s, respectively. These functions and all remaining elements of the ***F*** matrix are computed in Appendix 2.

## 2.4 Revision to Simulation Program

The computer program Long-sim.c has been developed by P. Yip in 1993, from earlier Fortran programs written by others. This is a code of some complexity, which has now been edited and expanded to include variations in the estimated state and the covariance for each vehicle in the platoon. The added portions of the code are indicated by the asterisk * in Table 2.

Table 2 - Simulation Program with Kalman Filter

```
* Read data and driving noise levels, initial state estimates and covariances
    Time loop
        Vehicle loop
            * Data_update: Improve estimate and covariance
              Controller: Define controls with state estimate
              Simple: Define state time-derivative
              Time-update of state: Runge-Kutta integrator
            * Simple: Define state estimate derivative
            * Time-update of state estimate: Runge-Kutta integrator
            * Cov_timeup: Time-update of covariance
        End vehicle loop
    End time loop
```

This revision more than doubles the computations needed, because of the statistical aspects of the problem. Both actual and estimated states must be integrated, and the covariance matrix diagonals are the mean-square errors in the estimates. The minimum-variance estimate or Kalman filter combines the imperfect data with the imperfect ***a priori*** estimate to yield an estimate which is “better” than either of its components. The simulated dynamics, estimation and control problem

***also*** permits the ***error*** in this estimate to be computed, and it should be consistent with the rms error found from the covariance matrix. The error is never known in actual applications, of course, so only the covariance can be computed on-line to give an estimate of the error. In our simulation, for which noise is artificially added to the otherwise "perfect" data, it is important to verify that the difference between the actual and estimated states be consistent with the rms error or standard deviation as in Fig. 2.1. Notice that the estimation error in this typical case is everywhere

```
time =  10.00
  eps      m-air  w_eng    v-car   x-car    w_whl   A_acc   T_brk
           dx[1]   dx[2]   dx[3]   dx[4]    dx[5]   dx[6]   dx[7]
           sd[1]   sd[2]   sd[3]   sd[4]    sd[5]   sd[6]   sd[7]

-0.331    4.461  284.058  25.196 238.681  78.744  45.914   10.713
        -0.0032   1.6794  0.2908  0.0339  0.7123  0.2080  -0.3709
         0.0138   2.2268  0.1962  0.0204  1.2010  0.4612   2.3151

-0.236  4.606  283.759  24.911  232.445  77.936  53.593  28.325
       -0.0080  0.7954  0.0209  -0.0211  0.3068  -0.1409  -1.1904
        0.0887  1.9959  0.1962  0.0204  1.2719  0.4612  2.3151

-0.174  4.542  278.895  24.614  226.272  76.818  49.949  45.524
       -0.0022  1.2026  -0.1534  0.0346  0.4080  0.4311  0.2975
        0.0657  2.0330  0.1962  0.0204  1.2670  0.4612  2.3151
```

Fig. 2.1 - Actual States, Estimation Errors and Standard Deviations

less than two standard deviations, or 2s, twice the square-root of the diagonal of the covariance. The filter would ***not*** be consistent, if substantial runs of error estimates were much smaller than 1s or much larger than 3s. Given the large number of numerical descriptors of the IVHS vehicle, the validity of the filter cannot be verified except by these methods. A certain lack of rigor must be accepted here, because the system has a number of nonlinear components, while the error in the estimate is modelled as normally distributed, of zero-mean and linear in its components. These assumptions are certainly not strictly true, but it is hoped that they are approximately so, because the on-line accuracy is to be judged through the covariance, as will be shown in Section 3.

## 2.5 Platoon Dynamics

The platoon dynamics include an intuitively evident feature: Since the motion of a vehicle depends on the motion of all vehicles ahead of it, and on none of those behind it, the order of the dynamics of vehicle ***n*** is ***n*** times the order of the first vehicle. Hence, if the dynamics of engine, wheels, etc., are ignored and the data are assumed perfect, the acceleration of each vehicle equals its command value. For simplicity, this is here assumed to be a linear combination of position and velocity error relative to the prior vehicle, or, in the notation of Sec. 2.2,

$$a_x(n) = q_1[pos_x(n-1) - pos_x(n)] + q_2[vel_x(n-1) - vel_x(n)]\,, \tag{1}$$

so the transfer function relating the ***absolute*** positions of this vehicle and the one ahead of it is,

$$\frac{X_n}{X_{n-1}}(s) = \frac{q_2 s + q_1}{s^2 + q_2 s + q_1} \tag{2}$$

The transfer function of vehicle ***n*** position and that of the lead vehicle is just the product of ***n*** consecutive such functions,

$$\frac{X_n}{X_o} = \frac{(q_2 s + q_1)^n}{(s^2 + q_2 s + q_1)^n}. \tag{3}$$

This function is based on a lower-ordered dynamic model, but it implies an interesting fact. The repeated poles and zeros correspond to a transient response at $t = 0$ which has the same damped sinusoidal components as that of Eq.(2), but they are multiplied by $t^{n-1}$. This means that the transient envelope of a vehicle far from the lead begins small and grows larger, due to $t^{n-1}$, and ends small, due to the exponential damping implied by the term $q_2 s$ in the characteristic equation. But at some intermediate time, this envelope can be “large”. The component magnitudes of the response depend on the details of the feedback gains which have been abbreviated in Eq.(3) above, but the “analysis” here may explain the so-called “Slinky” effects observed in multi-vehicle simulations by a fixed observer. The transients implied by this multiple-root function have been modelled numerically, but they appear to be of only incidental interest, as far as the estimation problem is concerned.

# 3. SIMULATION OF DYNAMIC MODEL TRANSIENT

## 3 .1 Introduction

The platoon computer simulation has the potential of refining the design of the control system in several ways. In the short time it has been operational, it has been used to show that nearly all of the nominal parameters of the filter reach a steady-state in a few seconds. This may mean that constant gains can be used in the filter without seriously affecting performance. Other conclusions can be expected as further use is made of the estimator.

The present version of the code is used in conjunction with several data files. These files include the initial values of the states, their estimates and the rms uncertainties of the estimates. Other data relates to the noise levels assumed to be present in the dynamic equations and in the data. The values used for illustrating the filter have not been verified or discussed with others, but have been taken from references dealing with IVHS parameters. New parameters such as driving noise levels are given plausible values, but the filter itself has been operational for too short a time to allow a verification of these input parameters. Hence, this portion of this report concerning numerical results merely suggests the types of results to be expected from the filter.

## 3.2 Functions Needed for Filter Implementation

In this Section, a nominal operating condition is chosen to illustrate the forms of the output from the program, as it combines the estimation function with the control and dynamics functions. Program long-sim now includes subroutines which initialize the state covariance matrix and the disturbance noise and data noise covariances. Each run of the simulation requires the following input parameters, as given in Appendix 2:

1. Integration step, final time, write interval and control interval. Initial values of states and their estimates. Main program long_sim calls the data file, four_car.dat

2. Noise seed, data matrix, driving noise covariance, initial state covariances for all vehicles, rms data noise. Function read.c calls initial_data.dat

3. Gains for feedback acceleration: prior vehicle acceleration, relative velocity and relative position, and lead vehicle acceleration, relative velocity and relative position.

Function controller.c

4. Values of slopes and locations of slope changes along highway. Function simple-1ong.c

Here, specific numerical outputs of the simulation code will be shown for a specific profile of lead-vehicle velocity on a typical segment of highway. The lead vehicle follows a speed profile which is initially constant, followed by a linear increase to a higher constant value. This higher level is maintained briefly before decreasing linearly with time to the original value. The departure by vehicle $n$ from the ideal separation $D_{safe}$ is defined by the distance

$$e(n) = pos_x(n-1) - (L_{car} + D_{safe}) - pos_x(n), \qquad (1)$$

where $L_{car} + D_{safe}$ is the desired value of the separation between the “front bumpers” of vehicles $n\text{-}l$ and $n$. The lead vehicle here has the index n = 0.

## *3.3* Control Gains

Each vehicle’s control system applies accelerator and brake inputs according to the linear combination of positions, velocities and acceleration of own vehicle, prior vehicle and lead vehicle, as given in Eq.(4) of Sec.2.2. Typical values for the feedback gains needed in this equation are

$$c_f = 2, \quad q_3 = .5, \quad c_p = .5, \quad q_4 = 0, \quad k_a = .5, \quad q_1 = .5, \quad q_2 = .5, \qquad (1)$$

and these (Eq.(6), Sec. 2.2) correspond to a quadratic characteristic equation for which all coefficients equal 1. The integral controller is ignored here; the natural frequency and damping ratio are 1 rad/s and 0.5. Higher damping may be required, for example by changing $q_1$ to *1 .O* and $q_3$ *to 1.5;* these changes gives the characteristic equation real roots. Limited tests with these gain changes showed only that the noise aspects of the control problem are more significant; i.e., the gains tabulated above give “good” results with respect to mean-squared departure from nominal separation; e = 0.

The control of all vehicles depends on both the preceding and the lead vehicles, but the sensitivity to the preceding vehicle is greater than that of the lead vehicle, as it should be. The six gains in the control equation multiply signals which are near zero; hence the command acceleration for each vehicle is near zero, on average. When the lead vehicle accelerates or decelerates, the imbalance in Eq.(2) yields a command acceleration to all of the subsequent vehicles, which is then processed with time according to the transition matrix. The additional control which may sometimes be needed, beyond those discussed here and in Finch (1971), is proportional to the integral of the error. This would control the effects of bias forces due to road slope, wind, or other cause of acceleration on one vehicle which is not present at the same time on all of the preceding vehicles. It could be an important contributor to the error, but its significance has not yet been estimated. It is noted again that control gains could be found so as to minimize a specific error measure. Answers to other aspects of the control problem solution are more urgently needed at this time, however. Since the emergency-identification problem deals with state estimates and their rms errors, rather than the performance resulting from a choice of gains, feedback gain values are of secondary interest.

## 3.4 Filter Gain Variations

Numerical results of the implementation of the filter in a nominal four-vehicle configuration are discussed in the following two sections of the report. The platoon is represented in the longitudinal dimension as having initial locations (2, -4, -10, -16) *m*, and with a common initial speed equal to 24.5 *m/s*. The typical run of 20 s is sufficient to show several cycles of physical motion of the vehicles. Here, we are concerned with the filter gain variations during the transient prior to steady-state operation. As shown in Fig. 3.1, filter gains derived from the output of the digital code have short-term transients preceding their steady-state values. These values depend on the dynamic equations, the data equations and all noise-levels.

Nearly all of the filter gains are constant to within 1% after the first 5 seconds of the filter implementation. But, because the dynamic equations are not generally linear, certain gains vary over a 20% to 30% range, even after 20 *s* of transient time. These gains relate to the engine speed in particular, which is obviously coupled to tire speed, velocity and position. They occur in part because the tabular engine data includes discontinuities and other nonlinearities encountered in the braking-accelerating short-headway platoon environment, while the covariance equation assumes that the dynamics are piecewise linear. In these *fgain* matrices, the row is the data component (Eq. (7) of Sec. 2.3) and the column is the state component (Eq.(2) of Sec. 2.2).

Two significant features are apparent. First, the initial gains are quite different from the later values, and second, those in column 2 (engine speed dependence on data) vary long after the others have reached steady-state values, at 5 to 10 s .

Time = 0.

| $m_{air}$ | $w_{eng}$ | $vel_x$ | $pos_x$ | $w_{whl}$ | *alpha* | $T_{brk}$ |
|---|---|---|---|---|---|---|
| 0.000 | 0.000 | 0.000 | -.984 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.000 | -.996 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.167 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.000 | 0.000 | 0.136 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.000 | 0.138 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | 0.198 | 0.000 | 0.000 |
| 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.854 | 0.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.054 |

. . .

Time = 20.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.000 | -.061 | -.264 | -.009 | -.089 | 0.000 | -.005 |
| 0.000 | -.001 | -.963 | -.010 | -.118 | 0.000 | 0.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.069 | 0.000 | 0.000 | 0.013 | 0.000 | -.001 |
| 0.000 | 0.009 | 0.002 | 0.001 | 0.009 | 0.000 | 0.001 |
| 0.000 | 0.000 | 0.134 | 0.001 | 0.016 | 0.000 | 0.000 |
| 0.000 | 0.209 | 0.003 | 0.001 | 0.477 | 0.000 | -.018 |
| 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.854 | 0.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.054 |

Time= 21,

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.000 | -.063 | -.264 | -.008 | -.0894 | 0.000 | -.005 |
| 0.000 | -.003 | -.963 | -.010 | -.113 | 0.000 | 0.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 0.000 | 0.05 1 | 0.000 | 0.000 | 0.019 | 0.000 | -.001 |
| 0.000 | 0.009 | 0.002 | 0.001 | 0.009 | 0.000 | 0.001 |
| 0.000 | 0.000 | 0.134 | 0.001 | 0.016 | 0.000 | 0.000 |
| 0.000 | 0.328 | 0.003 | 0.001 | 0.430 | 0.000 | -.018 |
| 0.003 | 0.000 | 0.000 | 0.000 | 0.000 | 0.854 | 0.000 |
| 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.054 |

Fig. 3.1 - Filter Gain Matrices at Three Times (Vehicle 2)

## 3.5 Representative Filter Code Results

The velocity variation of the lead-vehicle is shown in Fig. 3.2, over a 20 $s$ time interval. The subsequent vehicles closely match this speed, as shown in Fig. 3.3, and the corresponding position errors are as shown in Fig. 3.4. Notice that these position errors are only partly due to the errors in the estimates of the individual terms in Eq.(2) above; i.e., even if the estimates were perfect, position errors would exist due to limitations of the control system.

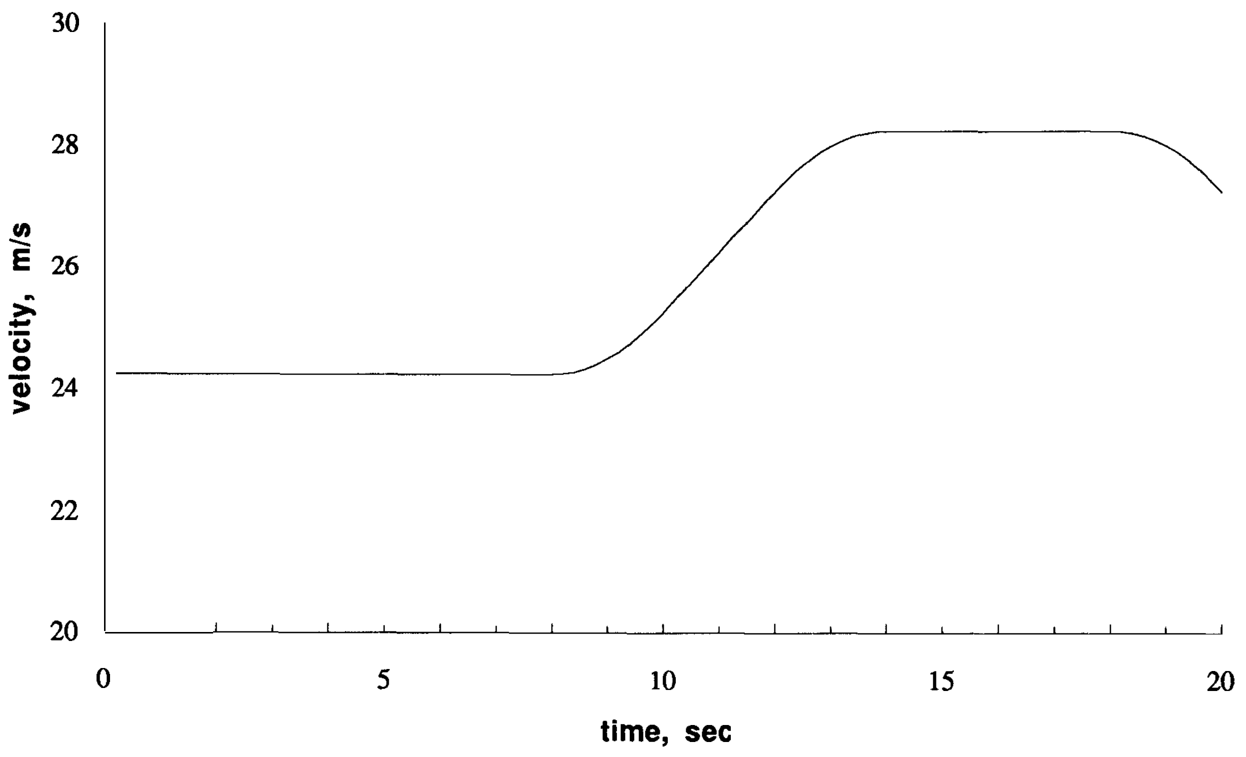
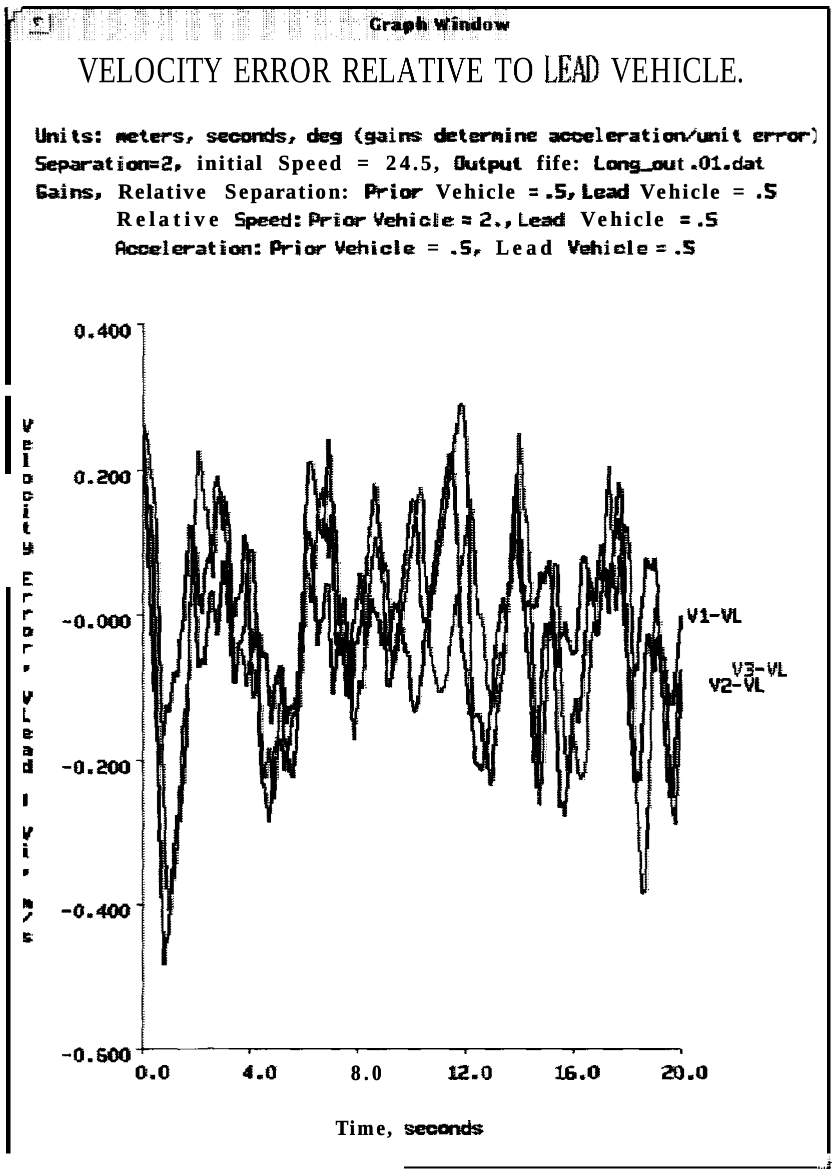


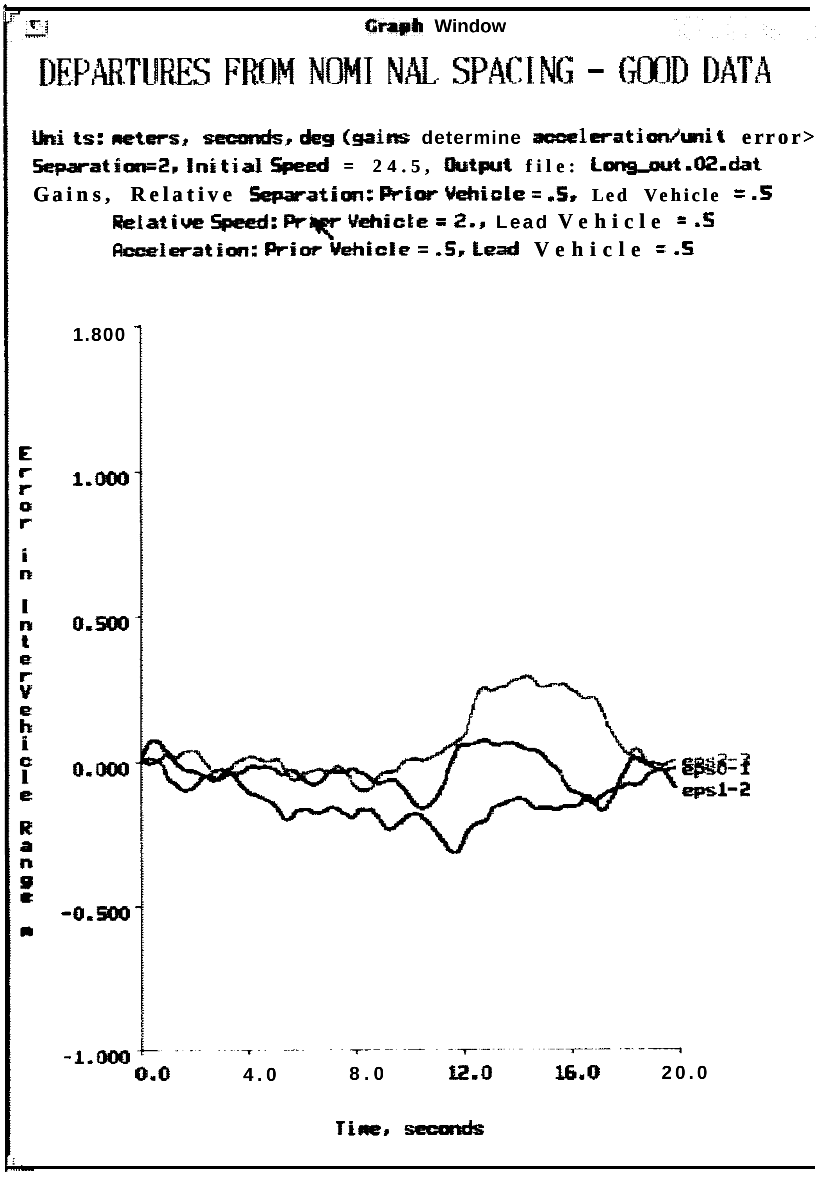
Fig. 3.2 - Velocity Variation of Lead Vehicle



VELOCITY ERROR RELATIVE TO LEAD VEHICLE.

Units: meters, seconds, deg (gains determine acceleration/unit error)
Separation=2, initial Speed = 24.5, Output fife: Long_out.01.dat
Gains, Relative Separation: Prior Vehicle = .5, Lead Vehicle = .5
Relative Speed: Prior Vehicle = 2., Lead Vehicle = .5
Acceleration: Prior Vehicle = .5, Lead Vehicle = .5

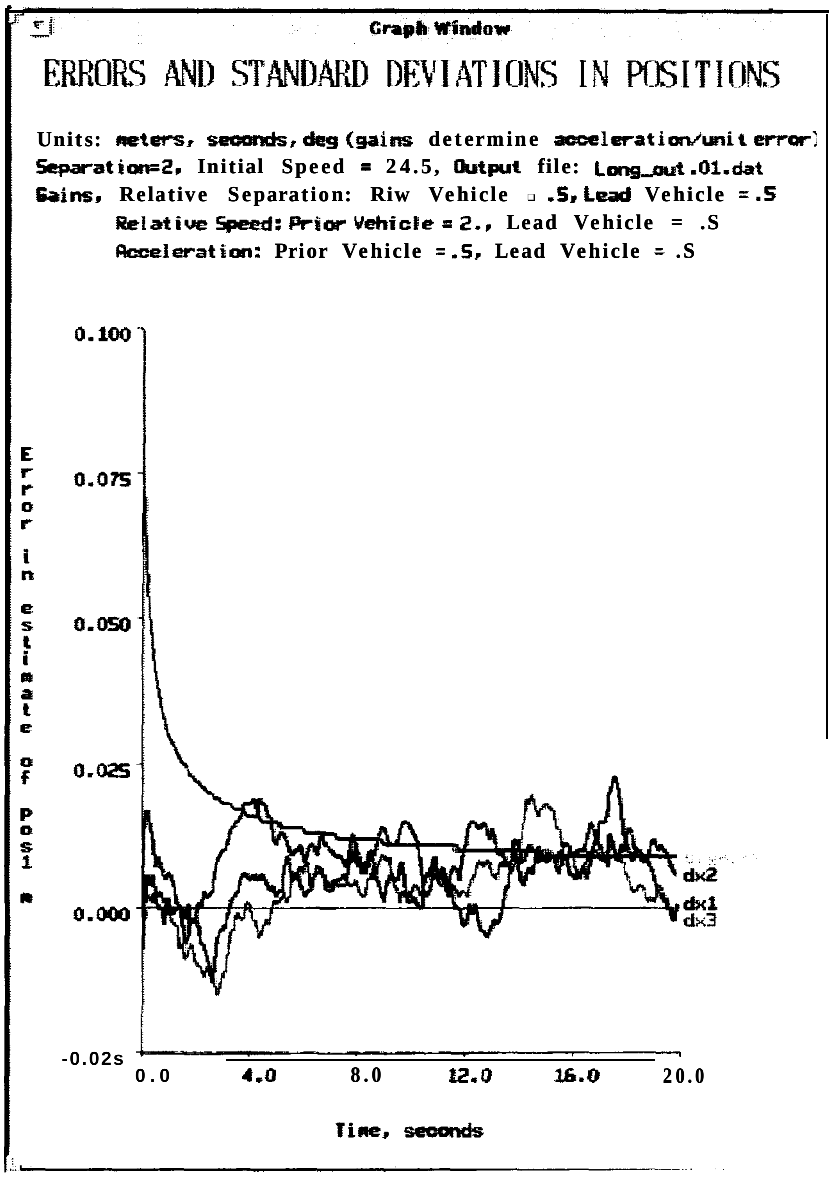
Fig. 3.3 Errors in Velocity for Consecutive Vehicles


Graph Window

DEPARTURES FROM NOMINAL SPACING - GOOD DATA

Units: meters, seconds, deg (gains determine acceleration/unit error>
Separation=2, Initial Speed = 24.5, Output file: Long_out.02.dat
Gains, Relative Separation: Prior Vehicle = .5, Led Vehicle = .5
Relative Speed: Prior Vehicle = 2., Lead Vehicle = .5
Acceleration: Prior Vehicle = .5, Lead Vehicle = .5

Fig. 3.4 - Errors in Position for Consecutive Vehicles

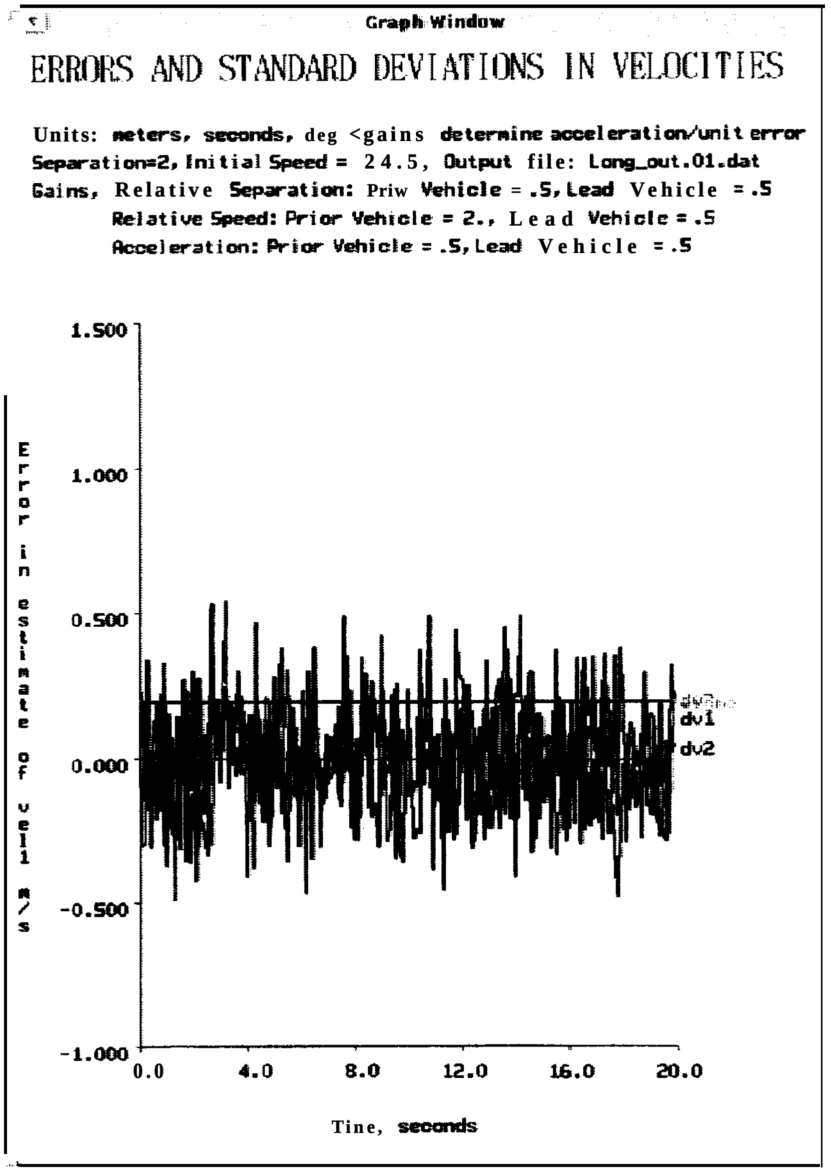


Graph Window

ERRORS AND STANDARD DEVIATIONS IN POSITIONS

Units: meters, seconds, deg (gains determine acceleration/unit error)
Separation=2, Initial Speed = 24.5, Output file: Long_out.01.dat
Gains, Relative Separation: Riw Vehicle ◻ .5, Lead Vehicle = .5
Relative Speed: Prior Vehicle = 2., Lead Vehicle = .S
Acceleration: Prior Vehicle = .5, Lead Vehicle = .S

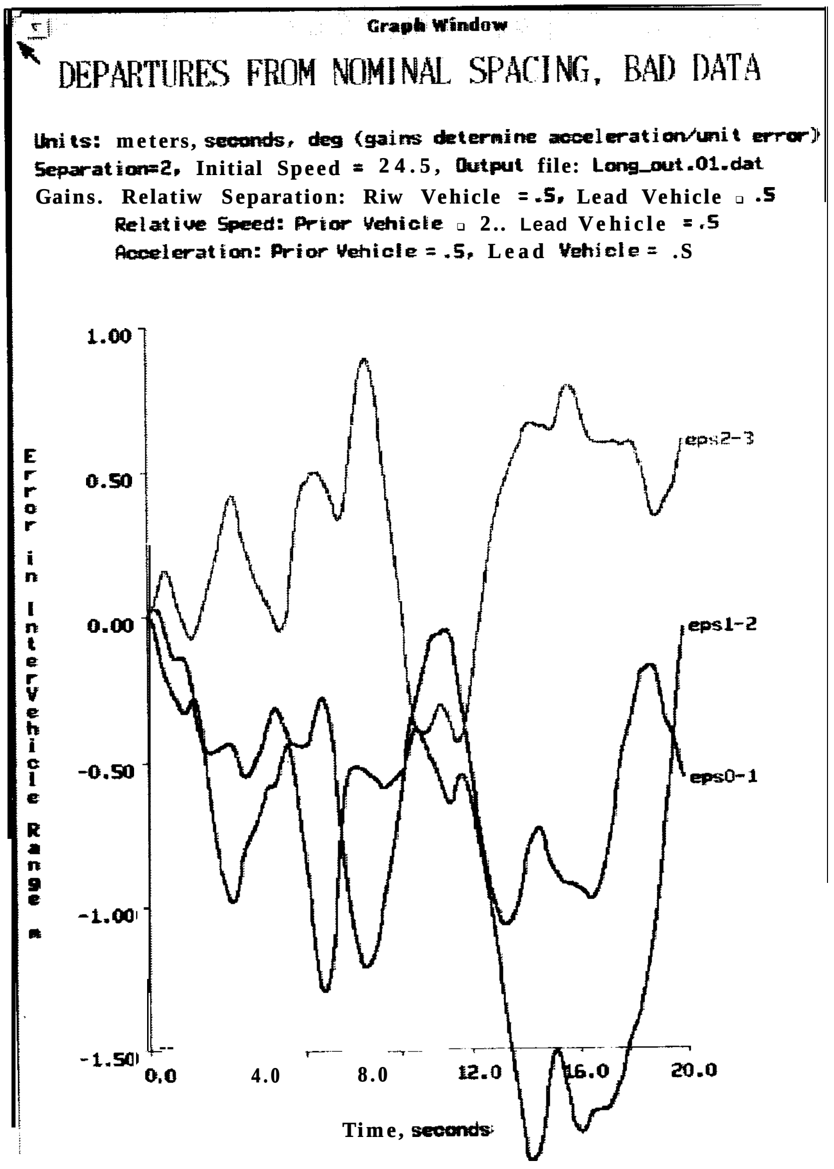
Fig. 3.5 - Errors in Estimate of Position
Data Errors 10 cm, 20 cm/s, rms



ERRORS AND STANDARD DEVIATIONS IN VELOCITIES

Units: meters, seconds, deg <gains determine acceleration/unit error
Separation=2, Initial Speed = 24.5, Output file: Long_out.01.dat
Gains, Relative Separation: Priw Vehicle = .5, Lead Vehicle = .5
Relative Speed: Prior Vehicle = 2., Lead Vehicle = .5
Acceleration: Prior Vehicle = .5, Lead Vehicle = .5

Fig. 3.6 - Errors in Estimate of Velocity



DEPARTURES FROM NOMINAL SPACING, BAD DATA

Units: meters, seconds, deg (gains determine acceleration/unit error)
Separation=2, Initial Speed = 24.5, Output file: Long_out.01.dat
Gains. Relatiw Separation: Riw Vehicle = .5, Lead Vehicle □ .5
Relative Speed: Prior Vehicle □ 2.. Lead Vehicle = .5
Acceleration: Prior Vehicle = .5, Lead Vehicle = .5

Fig. 3.7 Error Variation with Bad Data

The errors in the estimates of position and velocity are shown in Figs. 3.5 and 3.6 These are shown with the smooth curves representing the rms errors in position and velocity, as given by the diagonals of the covariance matrix. For rms range data errors of 10 cm, the steady-state range errors are about 1 cm for all of the following vehicles. The position estimation errors happen to be much less than the position location errors shown in Fig. 3.4, so for these parameters the filter provides slightly better results than those implied by assuming perfect knowledge of the vehicle states. Numerical tests show that with plausible values for system noise inputs due to throttle, brake and road surface, the rms relative position error in the steady state is usually about 10% of the inter-vehicle radar error. This conclusion implies that filtered data from "inexpensive" sensors could replace unfiltered data from "expensive" sensors.

As the accuracies of the sensors decrease, the average performance of the system must also decrease. When the position and velocity errors are increased to 10 cm and 100 cm/s, and all of the other sensors are given large rms errors (i.e., the corresponding data is absent), the oscillations become extreme, as illustrated in Fig. 3.7. This illustrates that the control data can be poor, even when it is filtered optimally.

A comparison test relates to the use of data from the lead vehicle. In Table 3 are shown the mean and rms displacement errors in *m* (for two runs), both with and without lead-vehicle data. The data from the lead vehicle generally has a beneficial effect on the error and its time-variation, because both mean and rms errors are smaller. The improvement is small, however, and the question requires further study.

Table 3 - Error Dependence on Lead Vehicle Data

| | | Mean Error | RMS Error |
|---|---|---|---|
| With | $\varepsilon(1)$ | −.11, −.03 | .07, .07 |
| Lead | $\varepsilon(2)$ | −.21, −.14 | .08, .08 |
| Data | $\varepsilon(3)$ | −.03, −.04 | .06 , .11 |
| | | | |
| Without | $\varepsilon(1)$ | −.11, −.15 | .08, .09 |
| Lead | $\varepsilon(2)$ | −.12, −.12 | .08, .11 |
| Data | $\varepsilon(3)$ | −.23, −.19 | .19, .15 |

## 4. CONCLUSIONS

Control of any real dynamic system is in terms of its estimated states, because feedback signals depend on ***measured*** output. If the sensor data has errors which are inconsistent with the need for accurate control, the sensors must be improved or the data must be filtered. When the parameters of the data and the controller are approximately known, tests with the code derived and discussed here will help in determining the performance to be expected. If performance is marginal, in fact, it may be necessary to add this filter capability to that of the operational IVHS system. In any case, the difference in performance obtained with and without a specific accuracy of data can be calculated by running the simulation with the filter under two conditions, corresponding to "good" and "bad" data of the type in question. If the improvement due to good data is very slight, and the good data is expensive, there is no reason for using it. This sort of trade-off can be expected in a system as complex as the IVHS, and it is a valid reason for developing a simulation which incorporates a filter.

The modelling of the dynamics in the U. C. Berkeley IVHS digital code is meant to be descriptive of both individual vehicles and their ***sensors*** and control systems. This is implied by the tabular data, the level of detail incorporated in the dynamics of the tires, the transmission, etc., and the high-order of the equations used to describe the transfer from one condition to another. Since the IVHS is intended for use with large numbers of vehicles, it will be necessary to evaluate system performance for vehicles very different from the nominal. It may therefore be possible to determine lower-ordered near-linear representations of platoon dynamics which are adequate dynamic models of the actual system. The point here is to acknowledge that the numerical characteristics of the actual vehicles in a platoon are neither known nor equal from one vehicle to the next. These variations may justify the use of simpler dynamic models, perhaps with statistically variable parameters, for design and analysis purposes.

The technical research effort for the present study was originally proposed to take place over a calendar year, at less than full time. Funding actually began some six-months after it had been scheduled, and it was required to be concluded on July 1, 1994. The resulting three-month interval was insufficient for the preliminary analysis, coding to augment the existing C-code program originated at U.C. Berkeley, debugging and review of the numerical output of the many programs written, and documentation. The dynamic equations had an unexpected complexity which required long study before incorporation into the Kalman filter. Many potential study topics have not yet been examined. For these reasons, the study results after the actual three-month contract interval are fewer than those projected at the time of writing the proposal. But the solution obtained for the coupled longitudinal motion provides a reference for the overall system simulation, in that the computer sequence of data, filter, control, time-update has been specified.

The principal product of the work is a computer code which develops filtered estimates of all of the longitudinal states for each of three vehicles in a four-vehicle platoon, together with rms errors in these estimates. A plot package has also been developed to transform any of the filter output parameters to graphical form. Future work with the filter should make use of this capability.

## APPENDIX 1 - SOURCES OF AUTOMOBILE COLLISION AND BREAKDOWN DATA

Information concerning automobile collision and breakdown data is obtainable from the following sources. These agencies represent industry and consumer-interest groups which can answer a wide range of questions dealing with both quantitative and more general matters.

| NAME | TELEPHONE |
|---|---|
| AAA San Francisco | 415-565-2102 |
| Librarian: R. Burke (MWF 8-5) | 415-565-2300 |
| American Automobile Manufacturing Association | 916-444-3767 |
| AAMA N. Highway Safety, Seattle, WA | 206-220-7640 |
| Auto Safety, Washington, DC | 202-366-2850 |
| Auto Safety Hotline | 415-744-3089 |
| DOT Bureau of Transportation Statistics Hotline | g-1-800-853-1351 |
| Sue Anne Connaughton | |
| Towed Passenger Vehicle Accidents | 202-366-5390 |
| Lee Franklin | |
| Motor Vehicle Defects | 202-366-5235 |
| Terry Anderson | |
| Tape files, consumer complains | 202-366-2768 |
| Motor Vehicle Research Safety | |
| Timothy Schaffer | 202-366-9550 |
| NTSA | 202-366-5307 |
| National Center for Statistics and Analysis: | |
| Accident Investigation Division | 202-366-4820 |
| Information Services | 202-366-4198 |
| Grace Hazzard | 202-366-5558 |
| Automobile Breakdown (Louann Hart) | 202-366-4198 |
| National Highway Traffic Safety Association | l-800-424-9393 |
| Hotline: Automatic data transmission | |
| Safety Systems Engineering and Analysis | 202-366-4850 |
| S.A.E. International, Transportation Statistics | 412-776-4841 |
| Transportation Safety Board Library, Barbara Post | 202-334-2990 |
| U.C. Berkeley Library, Catherine Cortleyou | 510-642-3604 |
| Vehicle Research and Test Center, Ohio | 513-666-4511 |

## APPENDIX 2 - LISTINGS OF NEW C-PROGRAMS

Below are given brief descriptions of various C-code programs written to augment the operational program written by U. C. Berkeley graduate student Patrick Yip, a coauthor of Hedrick, et al. (1994). The second program is the revision to longsim, which has been expanded to account for the additional computation required by/ the estimated state components.

1. Transition: Determine transition and control matrices for each vehicle. These are needed in order to give time-updates to the covariance matrix. The numerical procedure finds the elements of the $F$ and G matrices by perturbational methods.

2. Simple-long: Highway slope variations with position on highway. These lead to bias accelerations on the vehicles, which can differ when the platoon traverses a discontinuity.

3. Read: Parameter input values, including sensor matrix, mean-square driving noises, initial covariances of states for each vehicle, and rms errors in data (equal for all vehicles)

4. Output: Print seven-component state, error in estimate, and rms error as implied by covariance matrix diagonal elements. A typical segment of this subroutine has been shown in Fig. 2.1.

5. The main program Long_sim initializes the vectors and matrices, and begins a time loop followed by a vehicle index loop, as in the original version of the code. The principal addition is the Kalman filter, which essentially supplements the system dynamics with those of the estimated system and the covariance. Long_sim is preceded by data-update and cov_timeup. Function controller now gives the command controls in terms of the ***estimate*** of the state, rather than the state itself, as originally.

```
/* ********************************************************** */
/* ********************** TRANSITION ********************** */
#include <math.h>
/**** simple longitudinal model based on fortran code by Swaroop */
/*
    five state model(ma,w_eng,vel,pos,w_wheel) plus actuator dynamics

    input: t - current time in sec
           x - state vector
                ma              (mass of air in intake manifold in kg)
                w_eng                   (engine speed in rad/s)
                vel_x           (car velocity in m/s)
                pos_x                   (car position in m)
                w-wheel         (wheel speed in rad/s)
                alpha           (throttle angle)
                t-brake         (brake torque in Nm)
           u - control vector
                alpha-c         (commanded throttle angle)
                t_brake_c       (commanded brake torque)
                r_gear                  (gear reduction ratio)
        output: dxdt - time derivative of state vector

        written by P. Patrick Yip 7/93
        VDL UC Berkeley
        Mod by A. Merz 6/94
****/
#include "simple_long.h" /* defintions for the states and controls for the
                             simple longitudinal model */
#include "simple_car.h" /* predefined parameters for the car like mass, wheel
                            inertia, etc. */
#include "car_data.h"
/* macro for bounding function: output=BOUND(input,upper_bound,lower_bound)*/
#define BOUND(x, y, z) ((x) > (y)) ? (y) : (((x) < (z)) ? (z) : (x))
extem double F[NUM_STATE+1][NUM_STATE+1];
extem double G[NUM_STATE+1][NUM_CTRL+1];
extem double G1[NUM_STATE+1];
void  transition(dd,t,x,u,xdot)
double  dd,  t, *x, *u, *xdot;
{
  double slip,f_tractive; /* tractive force */
  double alon,t_wheel; /* hiway accel and wheel torque */
  double t_pump,t_turb; /* torque converter pum and turbine torques */
  double r_gear; /* gear speed reduction ratio */
  double ma_pert,w_eng_pert,vel_x_pert,slip_pert,t_wheel_pert;
  double w_wheel_pert,t_brake_pert,alpha_pert,f_tractive_pert;
  double ma_pert_dot,w_eng_pert_dot,vel_x_pert_dot,w_wheel_pert_dot;
  double alpha_pert_dot,t_pump_pert,t_turb_pert;
  double   dma,dve,dww,dalf,dvx,dwe,dtb,dt;
  double buf; /* temporary storage */
  int i;
  int j;
  void torque_converter(), engine();
  double fabs();
```

```
/* global table lookup data arrays */
extem double **tq_table1; /* torque converter data */
extem double **tq_table2;
extern int tq_table1_size;
extern int tq_table2_size;
extem double **tc_table; /* throttle characterisitic data */
extem int tc_table_size;
extem double **p-table;      /* engine map */
extern double **met table
extem double **mao_table-
extem double *we_table;              /* data */
extern int *p_table_size; /* pressure table size for each we data pt */
extem int we-table size;
/* assign gear reduction ratio according to what gear is engaged */
switch ((int) gear-engaged) {
case 1:
 r_gear = FIRST-GEAR-RATIO;
 break,
case 2:
 r_gear = SECOND_GEAR_RATIO;
 break,
case 3:
 r_gear = THIRD_GEAR_RATIO;
 break,
case 4:
 r_gear = FOURTH-GEAR-RATIO;
}
/* calc. pump and turbine torque for torque converter */
torque_converter(w_eng,w_wheel,R_DRIVE,r_gear,tq_table1,tq_table2,
                tq_table1_size,tq_table2_size,&t_pump,&t_turb);
/* time derivatives of engine states: ma-dot and w_eng_dot */
engine(alpha,ma,w_eng,t_pump,
        we_table,p_table,tnet_table,mao_table,
        we_table_size,p_table_size,tc_table,tc_table_size,
        &ma_dot, &w_eng_dot);
/* tire forces */
slip = 1.0 - vel_x/(Hr * w-wheel); /* Hr = wheel radius */
if ((buf=fabs((double) slip)) > 0.15) slip = 0.15 * slip/buf;
f_tractive = Kt * slip; /* linear tractive force model */
hiway(pos_x,&alon);
/* time derivative of long states vel (accel) and pos    */
vel_x_dot = (f-tractive - Ca * vel-x * vel-x - F_roll_frict + alon)/MASS;
pos_x_dot = vel-x;
/* time derivative of wheel speed (wheel ang. accel) */
t-wheel = Hr * f_tractive + t-brake; /* wheel torque */
w-wheel-dot = (t_turb/(R_DRIVE * r-gear) - t_wheel)/J_WHEEL;
/* throttle actuator dynamics */
/* constraint throttle angle command to physical limits */
buf = (BOUND(alpha_c,MAX_THROTTLE,MIN_THROTTLE) - alpha)/TAU_THROTTLE;
/* limit throttle rate to +/- MAX_THROTTLE_RATE */
alpha-dot = BOUND(buf,MAX_THROTTLE_RATE,-(MAX_THROTTLE_RATE));
/* brake dynamics */
t-brake_dot = (t-brake-c - t_brake)/TAU_BRAKE;
/* printf(" ma-dot, w-e-dot = %6.3f %6.3f \n",ma_dot,w_eng_dot);
```

```
 printf("v_x_dot,x_dot = %6.3f %6.3f, \n",vel_x_dot,pos_x_dot);
 printf(" w_w_dot,alf_dot,t_b_dot = %6.2f %6.2f %6.2f\n",w_wheel_dot,
        t_brake_dot,alpha_dot);    */
 dma=.01*dd;
 dwe=dd;
 dww=dd;
 dalf=dd;
 dvx=dd;
 dtb=dd;
 dt=(double) 0.01;
/*  At any time, zero all F, G, G1 matrix elements   */
 for (i=0;i<= NUM-STATE; i++) (
        G 1 [i]=0.0;
    for (j=0;j<= NUM-STATE; j++) {
     F[i][j] = 0.0;
     if( j <= NUM_CTRL ) G[i][j] = 0.0;


 I
/*  Get first column of F-matrix (two terms)   */
 ma_pert = ma+dma;
 engine(alpha,ma_pert,w_eng,t_pump,we_table,p_table,tnet_table,mao_table,
        we_table_size,p_table_size,tc_table,tc_table_size,&ma_pert_dot,
        &w_eng_pert_dot);
 F[1][1] = 1.+  (ma-pert-dot  - ma_dot)/dma*dt;
 F[2][1] = (w_eng_pert_dot - w_eng_dot)/dma*dt;
/* printf("ma_pert_dot,  ma-dot = %8.4lf %8.4lf \n",ma_pert_dot,ma_dot); */
/* Second column of F-matrix */
 w_eng_pert = w_eng + dwe;
 torque_converter(w_eng_pert,w_wheel,R_DRIVE,r_gear,tq_table 1 ,tq_table2,
        tq_table1_size,tq_table2_size,&t_pump_pert,&t_turb_pert);
/*    t-pump changed to t-pump-pert on June 23, 94:    */
 engine(alpha,ma,w_eng_pert,t_pump_pert,we_table,p_table,tnet_table,mao_table,
        we_table_size,p_table_size,tc_table,tc_table_size,&ma_pert_dot,
        &w_eng_pert_dot);
 F[ 1][2] = (ma-pert-dot - ma_dot)/dwe*dt;
 F[2][2] = 1. + (w_eng_pert_dot - w_eng_dot)/dwe*dt;
 w_wheel_pert_dot=(t_turb_pert/(R_DRIVE * r_gear) - t_wheel)/J_WHEEL;
 F[5][2]  =  (w-wheel-pert-dot  - w_wheel_dot)/dwe*dt;
/* Third column of F-matrix */
 vel_x_pert = vel_x + dvx;
 slip-pert = 1.0 - vel_x_pert/(Hr*w_wheel);
 if ((buf=fabs((double) slip-pert)) > 0.15) slip-pert = 0.15*slip_pert/buf;
 f_tractive_pert = Kt * slip-pert;
 hiway(pos_x,&alon);
 vel_x_pert_dot = (f_tractive_pert - Ca*vel_x_pert*vel_x_pert- F_roll_frict + alon)/MASS;
 F[3][3]  =  1.  + (vel-x-pert-dot  - vel_x_dot)/dvx*dt;
 F[4] [3] = dt;

 t-wheel-pert  =Hr*f_tractive_pert + t-brake;
 w-wheel-pert-dot  =  (t_turb/(R_DRIVE * r_gear) - t_wheel_pert)/J_WHEEL;
 F[5][3]  =  (w-wheel-pert-dot  - w_wheel_dot)/dvx*dt;
/* Fourth column of F-matrix */

 F[4][4] = 1.;          /*  x(i+l)  =  F(4,4)x(i)  +  F(4,3)v(i) */
```

```
/* Fifth column of F-matrix */
  w-wheel-pert = w-wheel + dww;

  slip-pert = 1.0 - vel_x/(Hr * w-wheel-pert); /* Hr = wheel radius */
  if ((buf=fabs((double) slip-pert)) > 0.15) slip-pert = 0.15 * slip_pert/buf;
  f_tractive_pert = Kt * slip-pert; /* linear tractive force model */
  torque_converter(w_eng,w_wheel_pert,R_DRIVE,r_gear,tq_table1,tq_table2,
        tq_table1_size,tq_table2_size,&t_pump_pert,&t_turb_pert);
  engine(alpha,ma,w_eng,t_pump_pert,we_table,p_table,tnet_table,mao_table,
        we_table_size,p_table_size,tc_table,tc_table_size,&ma_pert_dot,
        &w_eng_pert_dot);
  F[1][5] = (ma-pert-dot - ma_dot)/dww*dt;
  F[2][5] = (w_eng_pert_dot - w_eng_dot)/dww*dt;
  hiway(pos_x,&alon);
  vel_x_pert_dot = (f-n-active-pert - Ca*vel_x*vel_x - F_roll_frict + alon)/MASS;
  F[3][5] = (vel_x_pert_dot - vel_x_dot)/dww*dt;

  w_wheel_pert_dot=(t_turb_pert/(R_DRIVE * r_gear) - t-wheel> ;
  F[5][5] = 1. + (w-wheel-pert-dot - w_wheel_dot)/dww*dt;
/* Sixth column of F-matrix */
  alpha_pert=alpha+dalf;
  engine(alpha_pert,ma,w_eng,t_pump,we_table,p_table,tnet_table,mao_table,
        we_table_size,p_table_size,tc_table,tc_table_size,&ma_pert_dot,
        &w_eng_pert_dot);
  F[1][6] = (ma-pert-dot - ma_dot)/dalf*dt;
  F[2][6] = (w_eng_pert_dot - w_eng_dot)/dalf*dt;
  buf=(BOUND(alpha_c,MAX_THROTTLE,MIN_THROTTLE) - alpha_pert)/TAU_THROTTLE;
  alpha_pert_dot=BOUND(buf,MAX_THROTTLE_RATE,-(MAX_THROTTLE_RATE));
  F[6][6] = exp(-dt/TAU_THROTTLE);
/* Seventh column of F-matrix  */
  t_brake_pert = t-brake + dtb;
  /* time derivative of wheel speed (wheel ang. accel) */
  t-wheel-pert = Hr * f-n-active + t-brake-pert; /* wheel torque */
  w-wheel-pert-dot = (t_turb/(R_DRIVE*r_gear) - t_wheel_pert)/J_WHEEL;
  F[5][7] = (w-wheel-pert-dot - w_wheel_dot)/dtb*dt;
  F[7][7] = exp(-dt/TAU_BRAKE);
  G[6][1] = 1.-exp(-dt/TAU_THROTTLE);
  G[7][2] = 1.-exp(-dt/TAU_BRAKE);
  G1[3] = 1.0;
  G1[4] = dt;
}
/* ************************************************************ */
/* ********************** SIMPLE_LONG ******************* */
/*     five state model(ma,w_eng,vel,pos,w_wheel) plus actuator dynamics

    input: t - current time in sec
           x - state vector
                ma          (mass of air in intake manifold in kg)
                w_eng               (engine speed in rad/s)
                vel_x       (car velocity in m/s)
                pos_x               (car position in m)
                w-wheel     (wheel speed in rad/s)
                alpha       (throttle angle)
                t-brake     (brake torque in Nm)
```

```
          u - control vector
               alpha-c       (commanded throttle angle)
               t-brake-c     (commanded brake torque)
               r_gear              (gear reduction ratio)
      output: dxdt - time derivative of state vector

      written by P. Patrick Yip 7/93
      VDL UC Berkeley
****/
#include "simple_long.h" /* defintions for the states and controls for the
                                simple longitudinal model */
#include "simple_car.h" /* predefined parameters for the car like mass, wheel
                               inertia, etc. */
/* #include "car_data.h" */
/* macro for bounding function: output=BOUND(input,upper_bound,lower_bound)*/
#define BOUND(x, y, z) ((x) > (y)) ? (y) : (((x) < (z)) ? (z) : (x))

/*  ******************************************************  */
/* Function: hiway                              */
/*   Purpose: This function determines the value of the slope   */
/*                 of the road given the location along the road.    */
/*  ******************************************************  */
void hiway(pos,alon)
  double pos,*alon;
{
  double dist[4],slope[4]; /* these are of dimension n-changes */
  double grav;
  int ih,n_changes;
  int      done;
            /* ******** here is hiway slope vs position */
  n-changes = 3;
  slope[0] = 0.02;
  slope[1] = -0.02;
  slope[2] = 0.01;
  slope[3] = -0.01;
  dist[0] = 0.0;
  dist[1] = 120.0;
  dist[2] = 240.0;
  dist[3] = 360.0;
  grav = 9.80;                 /* gravity in m/s**2 */
/*              determine location of car relative to
                locations on highway of slope changes      */
  done = 0;                    /* FALSE = 0 */
  ih = 3;
  while (!done)
    {
    if (pos > dist[ih])
      done = 1;                /* TRUE = 1 */
    else
      ih--;       /* ih=ih+ 1; */

    if (ih<0)
      {
      done = 1;                     /*** TRUE = 1 ***/
      ih = 0;
      }
    }

   *alon = -grav*slope[ih];
) /* End of function hiway */
/* ************************************************************ */
void  simple-long(t,x,u,xdot)
double t, *x, *u, *xdot;
{
  double slip,f_tractive; /* n-active force */
  double t-wheel; /* wheel torque */
  double t_pump,t_turb; /* torque converter pum and turbine torques */
  double r_gear; /* gear speed reduction ratio */
  double buf; /* temporary storage */
  int i;
  int j;
  void torque_converter(), engine();
  double fabs();
  double alon;  /* Acceleration due to longitudinal slope */

  /* global table lookup data arrays */
  extern double **tq_table1; /* torque converter data */
  extern double **tq_table2;
  extern int tq_table1_size;
  extern int tq_table2_size;
  extern double **tc_table; /* throttle characterisitic data */
  extern int tc_table_size;
  extern double **p-table;      /* engine map */
  extem double **met_table;
  extern double **mao table;
  extern double *we-table;               /* data */
  extern int *p_table_size; /* pressure table size for each we data pt */
  extern int we-table-size;

  /* assign gear reduction ratio according to what gear is engaged */
  switch ((int) gear-engaged) {
  case 1:
    r_gear = FIRST-GEAR-RATIO,
    break;
  case 2:
    r_gear = SECOND_GEAR_RATIO;
    break,
  case 3:
    r_gear = THIRD_GEAR_RATIO;
    break,
  case 4:
    r_gear = FOURTH-GEAR-RATIO,
  }
  /* calc. pump and turbine torque for torque converter */
  torque_converter(w_eng,w_wheel,R_DRIVE,r_gear,tq_table1,tq_table2,
                   tq_table1_size,tq_table2_size,&t_pump,&t_turb);

  /* time derivatives of engine states: ma-dot and w_eng_dot */
```

```
 engine(alpha,ma,w_eng,t_pump,
         we_table,p_table,tnet_table,mao_table,
         we_table_size,p_table_size,tc_table,tc_table_size,
         &ma_dot, &w_eng_dot);

 /* tire forces */
 slip = 1.0 - vel_x/(Hr * w-wheel); /* Hr = wheel radius */
 if ((buf=fabs((double) slip)) > 0.15) slip = 0.15 * slip/buf;
 f_tractive = Kt * slip; /* linear tractive force model */

 hiway(pos_x,&alon);
 vel_x_dot = (f-tractive - Ca * vel-x * vel-x - F_roll_frict + alon)/MASS;
 pos_x_dot = vel-x;
 /* time derivative of wheel speed (wheel ang. accel) */
 t-wheel = Hr * f_tractive + t-brake; /* wheel torque */
 w-wheel-dot = (t_turb/(R_DRIVE * r-gear) - t_wheel)/J_WHEEL;

 /* throttle actuator dynamics */
 /* constraint throttle angle command to physical limits */
 buf = (BOUND(alpha_c,MAX_THROTTLE,MIN_THROTTLE) - alpha)/TAU_THROTTLE;
 /* limit throttle rate to +/- MAX_THROTTLE_RATE */
 alpha-dot = BOUND(buf,MAX_THROTTLE_RATE,-(MAX_THROTTLE_RATE));
 /* brake dynamics */
 t-brake-dot = (t-brake-c - t_brake)/TAU_BRAKE;
}
/* ************************************************************ */
/* ******************** READ_COV_DATA   ************************ */
#/include <stdio.h>
#include <math.h>
#include <string.h>
#include "nrutil.h"

#include "stat.h"        /* Holds structure for control parameters       */

/* lookup table sizes */
#include "car_data.h"

/* global table lookup data arrays */

void Read_cov_data(covariance_pre,std_dev_data,uu,H,seed)
 double        covariance_pre[NUM_CAR][NUM_STATE+1][NUM_STATE+1];
 double        std_dev_data[NUM_DATA+1];
 double        uu[NUM_CTRL+1][NUM_CTRL+1];
 double        H[NUM_DATA+1][NUM_STATE+1];
 int           *seed;
{
 FILE                 *fp;
 int           i,j,k;           /* Generic incrementation variables */
 char          strholder[100]; /* Text throw away string holder    */

 fp=fopen("initial_data.dat","r"); /* File of initial setting of parameters*/

 fscanf(fp,"%d",seed);
 fscanf(fp,"%s\n",strholder);
 for(i=1;i<NUM_DATA+1;i++)
   for(j=1;j<NUM_STATE+1;j++)
      fscanf(fp,"%lf\t",&(H[i][j]));

 fscanf(fp,"%s\n",strholder);
 for(i=1;i<NUM_CTRL+1;i++)
   for(j=1;j<NUM_CTRL+1;j++)
      fscanf(fp,"%lf\t",&(uu[i][j]));

 fscanf(fp,"%s\n",strholder);
 for(i=1;i<NUM_CAR+1;i++)
   {
   for(j=1;j<NUM_STATE+1;j++)
     for(k= 1 ;k<NUM_STATE+ 1 ;k++)
        fscanf(fp,"%lf\t",&(covariance_pre[i][j][k]));
   fscanf(fp,"%s\n",strholder);
   }

 for(i=1;i<NUM_DATA+1;i++)
      fscanf(fp,"%lf\t",&(std_dev_data[i]));

 fclose(fp);

} /* end of Read_cov_data function */
/* ************************************************************ */
pa**************************      OUTPUT    ************************* */
/***** output: print out simulation results to file
        given the state vector and state derivative vector

                written by P. Patrick Yip 7/93
                VDL UC Berkeley

                                        ******/
#include <stdio.h>

#/include "simple_long.h"

void output(car,n,space,length,t,x,dx,std_dev,u,pos,xlead,fp)
double space, length; /* desired spacing between adjacent cars and car
                        length */
double t;               /* simulation time */
double *x, *u;                  /* state and control vector of current car */

double *dx,*std_dev;    /*  added  for  plot  purposes  */

double *pos, xlead;     /* position of cars and position of lead car */
int car;                /* current car id number in platoon */
int n;                  /* total number in platoon (excluding the lead car) */
PILE *fp;               /* output file */
{
        double xprev,error;
/*      if (t==O && car==l)    */
        if(car==l)
                xprev=xlead,
        else
                xprev=pos[car-1];

        error=pos_x - xprev + space + length;
        if(car==l) {
        fprintf(fp,"\n time = %6.2f\n",t);
        }
        if (car== 1)
        {
        fprintf(fp,"\n e p s   m - a i r  w_eng  v - c a r   x - c a r  w_wh A_acc T_brk \n");
        fprintf(fp,"          dx[1]  dx[2]  dx[3]  dx[4]  dx[5]  dx[6]  dx[7]  \n");
        fprintf(fp,"          sd[1]  sd[2]  sd[3]  sd[4]  sd[5]  sd[6]  sd[7]   \n\n");
        }
        /* print out spacing error, manifold air mass, throttle, brake
        torque, (replace with desired quantity as appropriate) */

      fprintf(fp,"%8.3lf %8.3lf %8.3lf %8.3lf %8.3lf %8.3lf %8.3lf %8.3lf \n",
        error,x[1],x[2],x[3],x[4],x[5],x[6],x[7]);

        fprintf(fp,"        %8.4lf %8.4lf %8.4lf %8.4lf %8.4lf %8.4lf %8.4lf \n",
        dx[1],dx[2],dx[3],dx[4],dx[5],dx[6],dx[7]);

        fprintf(fp,"        %8.4lf %8.4lf %8.4lf %8.4lf %8.4lf %8.4lf %8.4lf \n",
        std_dev[1],std_dev[2],std_dev[3],std_dev[4],std_dev[5],std_dev[6],std_dev[7]);

        fprintf(fp,"\n");
}
* ************************************************************ */
/* ************************     LONG_SIM     ******************** */
/**** top level template of the modularized path simulation
       long_sim: platoon simulation with simple longitudinal model
       usage: long-sim [-i init_cond_file] specify data file for init cond
                        [-o output_file] specify file name for sim. results
 ****/

#include <stdio.h>
#include <math.h>
#include “nrutil. h"
#define         NEW-LARGE (65535 * 32768)
#include "stat.h"       /* Holds structure for control parameters       */
#include "car_data.h"
/* global table lookup data arrays */
double **tq_table1, **tq_table2; /* torque converter data */
int tq_table1_size, tq_table2_size; /* torque converter table sizes */
int ij;
double **tc_table; /* throttle characterisitic data */
int tc_table_size; /* throttle table size */
double integ[4];   /* integral of error for each vehicle */
double  **p-table,**met_table,**mao_table; /* engine map */
double  *we-table;                 /* data */
int we-table-size; /* number of we data pt for engine map table */
int *p-table-size; /* pressure table size for each we data pt */
double F[NUM_STATE+1][NUM_STATE+1],G[NUM_STATE+1][NUM_CTRL+1];
double G1[NUM_STATE+1];
double ident[NUM_STATE+1][NUM_STATE+1];
double uu[NUM_CTRL+1][NUM_CTRL+1];
int   seed:       /* Used to initiate a pseudo-random sequence */
BUCKET values;

#define ORIG_MODE 0 /* Flag to read our own initial data (dt, tf,
                         and wrt_freq., ctrl_freq */

extern double dot();
extern double Nran();
/*      The following functions are used by main()           */
        void mateq(a,b)
        double a[NUM_STATE+1][NUM_STATE+1], b[NUM_STATE+1][NUM_STATE+1];
{
        int ij;
        for (i=0; i<=NUM_STATE; ++i)
        {
                for (j=0; j<=NUM_STATE; ++j)
                        a[i][j] = b[i][j];
        }
}

/* *************    PICK ROW i FROM MATRIX H **************** */
        void   matrix-row(irow,H,hrow)
        int irow;
        double H[NUM_DATA+1][NUM_STATE+1],hrow[NUM_STATE+1];
{
        intj;

        for (j=0; j<=NUM_STATE; j++)
        hrow[j] = H[irow][j];
}
/*      ********************************************** */
/* *********  DATA UPDATE OF COVARIANCE & ESTIMATE  ********* */
/* ********************************************************** */
  v o i d data_update(icar,t,H,ident,std_dev_data,x,xhat,cov2_pre,cov2_post)
        int icar;
        double t;
        double
x[NUM_STATE+1],xhat[NUM_STATE+1],ident[NUM_STATE+1][NUM_STATE+1];
        double H[NUM_DATA+ 1 ] [NUM_STATE+ 1] ,std_dev_data[NUM_DATA+ 1];
        double
cov2_pre[NUM_STATE+1][NUM_STATE+1],cov2_post[NUM_STATE+1][NUM_STATE+1];
{
        double fgain[NUM_STATE+1];
        double data_noise[NUM_DATA+1], gh[NUM_STATE+1][NUM_STATE+1];
        double vmhrow[NUM_STATE+1],hrow[NUM_STATE+1];

        double dx[NUM_STATE+1], egh[NUM_STATE+1][NUM_STATE+1],dy,y,yhat,hvhr;
        int i,j,irow;
      double xx;
      int ii;      /* Generic Incrementation Variable */
        double ires,jres; /* Temp variables used in modified random generator */
/*  Here are the scalar updates of the state estimates based on relative
        goodness of a priori estimate and data                     */
for( i= 0; i < NUM_STATE+1; i++ )
{
  fgain[i]= (double)0.0;
}

  for (i=l; i<= NUM_DATA; i++)    /* data update of xhat with y */
  {
                irow=i;

                matrix-row(irow,H,hrow);
                y=dot(hrow,x,NUM_STATE+1);
        if (seed c 0)
                {
                dy = Nran((double)O.O, std_dev_data[irow] );
                }
        else
                {
                xx = (double)(-6.0);
        for(ii=0;ii< 12;ii++)
                  {
                  ires = rand();
                  jres = rand();
                  xx += ((double)(jres)+((double)(ires)/NEW_LARGE))/NEW_LARGE;
                  }
                dy = std_dev_data[irow]*xx+0.0;
                printf("\n NOISE = %9.5lf \n",dy);
           }
                y=y + dy;
                yhat=dot(hrow,xhat,NUM_STATE+1); /* current estimated data */
                dy=y-yhat;
        mat_vec_product(cov2_pre,hrow,vmhrow,NUM_STATE+1,NUM_STATE+1);
        hvhr=dot(vmhrow,hrow,NUM_STATE+1)+std_dev_data[irow]*std_dev_data[irow];
                for (j = 1; j<=NUM_STATE; j++ )
                {
                        fgain[j] = vmhrow[j]/hvhr;
                        dx[j]=fgain[j]*dy;
                }
        outer_product(fgain,hrow,gh,NUM_STATE+1);
        matsubtract(ident,gh,egh,NUM_STATE+1,NUM_STATE+1);
/*   data-update of covariance:                        */
        matmultiply(egh,cov2_pre,cov2_post,NUM_STATE+1,NUM_STATE+1,NUM_STATE
+1);
        mateq(cov2_pre,cov2_post);
/*   data-update of state estimate, xhat */
        vecadd( xhat, dx, xhat, NUM_STATE+1);
  }
}
/* ************************************************************ */
/* ***************   TIME UPDATE OF COVARIANCE ************ */
        void cov_timeup(icar,uu,sig_bump,FF,GG,GG1,cov2_post,cov2_pre)
        double uu[NUM_CTRL+1][NUM_CTRL+1],
             FF[NUM_STATE+1][NUM_STATE+1];
        double GG[NUM_STATE+1][NUM_CTRL+1],
            GG1[NUM_STATE+1];
        double sig_bump;
        double cov2_post[NUM_STATE+1][NUM_STATE+1],
            cov2_pre[NUM_STATE+1][NUM_STATE+1];
        int icar;
  {
        double GT[NUM_CTRL+1][NUM_STATE+1],G1T[NUM_STATE+1];
        double g1ugt[NUM_STATE+1][NUM_STATE+1];
        double
FT[NUM_STATE+1][NUM_STATE+1],fvft[NUM_STATE+1][NUM_STATE+1];
        double fc[NUM_STATE+1][NUM_STATE+1],fcft[NUM_STATE+1][NUM_STATE+1];
        double
guu[NUM_STATE+1][NUM_CTRL+1],gugt[NUM_STATE+1][NUM_STATE+1];

        transpose(FF,FT,NUM_STATE+1,NUM_STATE+1);
        transpose(GG,GT,NUM_STATE+1,NUM_CTRL+1);

        matmultiply(FF,cov2_post,fc,NUM_STATE+1,NUM_STATE+1,NUM_STATE+1);
        matmultiply(fc,FT,fcft,NUM_STATE+1,NUM_STATE+1,NUM_STATE+1);
        matmultiply(GG,uu,guu,NUM_STATE+1,NUM_CTRL+1,NUM_CTRL+1);
        matmultiply(guu,GT,gugt,NUM_STATE+1,NUM_CTRL+1,NUM_STATE+1);
        matadd(fcft,gugt,cov2_pre,NUM_STATE+1,NUM_STATE+1);
        GG1[3]=GG1[3]*sig_bump;
        GG 1[4]=GG1[4]*sig_bump; /* longitudinal velocities sigbump */
        outer_product(GG1,GG1,g1ugt,NUM_STATE+1,NUM_STATE+1);
        matadd(cov2_pre,g1ugt,cov2_pre,NUM_STATE+1,NUM_STATE+1);
}
/* ************************************************************ */
/* *****************    MAIN PROGRAM (LONG_SIM) *************** */
main(argc,argv)
int argc;
char *argv[];
{
 'FILE  *dave;
  /* Declare the Xwindow pointer for graph display. */
  char display_name[80];

  double **x,**u; /* 2d matrices holding
                     state and control vectors for each car */
  double **xhat,*xhat_dot;
  double *xdot,*xout; /* 1d vectors holding xdot and updated state
                         vector for current car */
/* 7 LINES ADDED TO ORIGINAL? */
  double xhat_out[NUM_STATE+1],dx_out[NUM_STATE+1];
  double H[NUM_DATA+1][NUM_STATE+1],ident[NUM_STATE+1][NUM_STATE+1];
  double
std_dev_data[NUM_DATA+1],covariance_pre[NUM_CAR+1][NUM_STATE+1][NUM_STATE
+1];
  double
covariance_post[NUM_CAR+1][NUM_STATE+1][NUM_STATE+1],std_dev[NUM_STATE+1
];
  double
cov2_pre[NUM_STATE+1][NUM_STATE+1],cov2_post[NUM_STATE+1][NUM_STATE+1];
```

```
 double qpos,sumx[4],sumxsq[4],xavg[4],xrms[4],xms;
 int npts,icheck=0;
 double dd, dt, t=0.0,time,tf,dt_ctrl;
 double alead,vlead,xlead; /* accel, vel and position of lead car */
 double *acc,*vel,*pos; /* 1d vectors for accel, vel and postion
                           of each car in the platoon */
 double eps,sig_bump;
 double  aprev,vprev,xprev;
 int ctrl_flag=0, wrt_flag=0;
 int icar, i, j, k,
 int ctrl_freq,wrt_freq,iprint; /* number of dt for controller and data logging
                           update */
 double *mades_old,*tb_old;

 double space,spacing_error; /* desired distance between adjacent cars and
                                spacing error */
 double car_length;
 int auto_init=1;
 /* command line options */
  char in='n',out='n',in_file[LINELEN],out_file[LINELEN];

 FILE  *fp_in, *fp_out, *fopen(), *fp;
 void readdat(),measure(),rk4();
 void profile(),init_long(),gear_shift();
 void command(),output();
 double mair_manifold();
 int      covtosd();
 double simple_long();
 void transition();              /*   addition for F, G, G1 computation */
 char dummy_str[101];
 integ[0]=integ[1]=integ[2]=integ[3]=0.0;
 xrms[0]=xrms[1]=xrms[2]=xrms[3]=0.0;
 xavg[0]=xavg[1]=xavg[2]=xavg[3]=0.0;
 sumx[0]=sumx[1]=sumx[2]=sumx[3]=0.0;
 sumxsq[0]=sumxsq[1]=sumxsq[2]=sumxsq[3]=0.0;
 npts=O;
 xms=0.0;
 for (i=O; i<= NUMSTATE; i++)
   {
   for (j=0; j<=NUM_DATA; j++)
     H[j] [i]=O.O;
   for (j=0; j<=NUM_STATE; j++)
     {
       cov2_pre[i][j]=0.0;
       cov2_post[i] [j]=0.0;
       ident[i] [j]=0.0;
     }
   }
sig_bump=1.O;               /* fwd rms acceleration, m/s**2      */

 Read_cov_data(covariance_pre,std_dev_data,uu,H,&seed);
/* initial_data(&seed,H,uu,covariance_pre,std_dev_data);   */
 srand(seed);
 ident[0][0]=1.0;
```

```
  ident[1][1]=1.0;
  ident[2][2]=1.0;
  ident[3][3]=1.0;
  ident[4][4]=1.0;
  ident[5][5]=1.0;
  ident[6][6]=1.0;
  ident[7][7]=1.0;
 fp=fopen("outfile.dat","w"); /* the output file is a write-file */
#if ORIG_MODE
  x=dmatrix(1,NUM_CAR,1,NUM_STATE);
  u=dmatrix(1,NUM_CAR,1,NUM_CTRL);
  acc=dvector(1,NUM_CAR);
  vel=dvector(1,NUM_CAR);
  pos=dvector(1,NUM_CAR);
  mades_old=dvector(1,NUM_CAR);
  tb_old=dvector(1,NUM_CAR);
#endif
  xdot=dvector(1,NUM_STATE);
  xhat_dot=dvector(1,NUM_STATE);
  xout=dvector(1,NUM_STATE);
  /* allocate storage for table lookup data */
  tq_table1_size=TQ1_SIZE;
  tq_table2_size=TQ2_SIZE;
  tc_table_size=TC_SIZE;
  we_table_size=WE_SIZE;
  tq_table1=dmatrix(1,4,1,tq_table1_size);
  tq_table2=dmatrix(1,3,1,tq_table2_size);
  tc_table=dmatrix(1,2,1,tc_table_size);
  p_table=dmatrix(1,we_table_size,1,P_SIZE);
  tnet_table=dmatrix(1,we_table_size,1,P_SIZE);
  mao_table=dmatrix(1,we_table_size,1,P_SIZE);
  we_table=dvector(1,we_table_size);
  p_table_size=ivector(1,we_table_size);
  /* read in table lookup data for engine, torque converter
     and throttle characteristics */

  readdat(tq_table1,tq_table2,tq_table1_size,tq_table2_size,
           tc_table,tc_table_size,we_table,p_table,
           tnet_table,mao_table,we_table_size,p_table_size);
  space = 2.0;          /* desired spacing between adjacent cars in m */
  car-length = 4.0;     /* car length in m */

  /* process command line arguments */
  command(argc,argv,&in,&out,in_file,out_file);
  if (in == 'y') auto_init = 0;
  /** read in initial conditions for the states **/
  if (auto_init)
  {
   profile(t,&alead,&vlead,&xlead); /* lead car acc, vel, pos at t=O */
   /* initialize the states of the simple long. model */
   for(i=l; i<=NUM_CAR; i++)
   {
    init_long(i,vlead,xlead,space,car_length,x[i],u[i]);
        for (j=0; j<=NUM_STATE; j++)
```

```
          xhat[i][j]=x[i][j];
     }
   }
   else {
     if (in == 'y')
       fp_in=fopen(in_file,"r");
     else
       fp_in=fopen("ic.dat","r");
     fscanf(fp_in, "%*s %lf ', &dt);
     fscanf(fp_in, "%*s %lf", &tf);
     fscanf(fp_in, "%*s %d", &wrt_freq);
     fscanf(fp_in, "%*s %d", &ctrl_freq);
     fscanf(fp_in, "%*s %d", &iprint);
printf("\n Ncars = %d, dt = %lf, tf = %lf, wrt_freq = %d, ctrl_freq = %d\n", NUM_CAR, dt, tf,
wrt_freq, ctrl_freq);
   x=dmatrix( 1,NUM_CAR, 1,NUM_STATE);
   xhat=dmatrix( 1,NUM_CAR, 1,NUM_STATE);
   u=dmatrix( 1,NUM_CAR, 1,NUM_CTRL);

   acc=dvector( 1,NUM_CAR);
   vel=dvector(1,NUM_CAR);
   pos=dvector( 1,NUM_CAR);
   mades_old=dvector( 1,NUM_CAR);
   tb_old=dvector( 1,NUM_CAR);

     for (i=l ; i<=NUM_CAR; i++)
       for (j=1; j<=NUM_STATE; j++)
            fscanf(fp_in,"%lf ',&(x[i][j]));

     for (i=l; i<=NUM_CAR; i++)
       for (j=1; j<=NUM_STATE; j++)
            fscanf(fp_in,"%lf",&(xhat[i][j]));
     fclose(fp_in);
   }

  /* initialize controller variables */
   for (i=l; i<=NUM_CAR; i++)
 {
     mades_ old[i]=mair_manifold(x[i]);
     tb_old[i] =O.O;
   }
  /** perform other initializations **/
#if ORIG_MODE
  dt = 0.001; /* integration time step in sec */
  wrt_freq = 50;
  tf=30.0; /* simulation end time in sec */
  ctrl_freq=50; /* change control every 50 dt's */
#endif
  dt_ctrl = dt * ctrl_freq; /* controller update period */
   if (out == 'y')
     fp_out = fopen(out_file,"w");
  else
     fp_out=fopen("sim.out","w");
  dd=(double) 0.01;
```

```
/*** loop through time with time increment dt
    updating the states every dt and the controls every n*dt ***/

  while (t <= tf)
{                              /* B E G I N  TIME LOOP /*

   profile(t,&alead,&vlead,&xlead);

   /** loop through the number of cars updating
     the states and controls of each individual car **/

   for (icar=1; icar<=NUM_CAR; ++icar)
  {                                   /* B E G I N  CAR LOOP /*
    /* calculate control input for each car every
          ctrl_freq*dt */
     if (ctrl_flag == 0)
    {
          if (icar==1) {
            /* for first car, previous car is just the lead car */
            aprev = alead;
            vprev = vlead;
            xprev = xlead,
          }
          if (icar > 1) {
            /* get info for previous car   */
            aprev = acc[icar-1];
            vprev = vel[icar-1];
            xprev = pos[icar-1];
          }
          if (icheck == 1) printf("\n icar = %d, xprev = %6.2lf \n",icar,xprev);

          gear-shift(x[icar],u[icar]);  /* gear shifting */

/********       This is the data update of the filter   ***************/
          mateq( cov2_pre, covariance_pre[icar] );
          data_update(icar,t,H,ident,std_dev_data,x[icar],xhat[icar],cov2_pre,cov2_post);
          vecsubtract(x[icar],xhat[icar],dx_out,NUM_STATE+1);
          mateq(covariance_post[icar],cov2_post );
#define DEBUG 1
#if DEBUG
#endif
          covtosd(cov2_post,std_dev,NUM_STATE+1);
/****** USE XHAT, NOT X, TO COMPUTE CONTROL U        */
          if (icheck ==  1) printf("\n controller input, icar = %d, ctrl-flag = %d \n",icar,ctrl_flag);
          if (icheck ==  1) printf("\n xhat = %5.2lf, xprev = %5.2lf \n",xhat[icar][4],xprev);
          controller(icar,dt_ctrl,space,aprev,vprev,xprev,
                      alead,vlead,xlead,
                      &mades_old[icar],&tb_old[icar], xhat[icar],u[icar]);

    }       /*   end of control loop    */

     simple-long(t,x[icar],u[icar],xdot);  /* xdot at current time */
     measure(x[icar],xdot,&acc[icar],&vel[icar],&pos[icar]);
    /* output desired states and control info for cars of interest
```

```
        every wrt_freq*dt */
    if (wrt_flag == 0) (
        output(icar,NUM_CAR,space,car_length,t,
             x[icar],dx_out,std_dev,u[icar],pos,xlead,fp_out);
        eps=pos[icar]-xprev+space+car_length;
    for (j=1; j<=NUM_STATE; ++j) xout[j] = x[icar][j];
          if (icar == 1)
            {
            npts = npts + 1;
            }
          sumx[icar]=sumx[icar]+eps;
          xavg[icar]=sumx[icar]/npts;
          sumxsq[icar]=sumxsq[icar]+eps*eps;
          if (npts > 1)
            {
            xms=(sumxsq[icar] -  sumx[icar]*sumx[icar]/npts)/(npts-1);
            xrms[icar]=sqrt(xms);
            }
/*       Prints values for car number 1    */
         if (icar == 1)
        {
                if (iprint == 1)
                {
 printf("=====================================================");
          printf("\n Time = %5.2lf\n",t);
          if (icheck == 1) printf("\n icar=l, xprev,xlead = %7.3lf %7.3lf \n",xprev,xlead);
          printf( 'In        m - a  w_eng vel_x pos_x w_wh  a c c e l  t_brk  e p s " ) ;

          printf("\n xl = %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.3lf\n",
                  xout[1]  ,xout[2]  ,xout[3]  ,xout[4]    ,
                  xout[5] ,xout[6] ,xout[7]  , e p s ) ;
          printf(" dx = %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf\n",
                  dx_out[ 1] ,dx_out[2] ,dx_out[3] ,dx_out[4] ,
                  dx_out[5] ,dx_out[6] ,dx_out[7]);
          printf(" sd = %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf\n",
                  std_dev[1],std_dev[2],std_dev[3],std_dev[4],
                  std_dev[5],std_dev[6],std_dev[7]);
                }
/* FPRINTF  LOADING  FOR  PLOTS    */
          fprintf(fp," %5.2lf\n",t);
          fprintf(fp,
                " %7.2lf %7.2lf %7.2lf %7.2lf %7.3lf\n",
                  xout[3] ,xout[4] ,xout[6] ,xout[7]  , e p s ) ;
          fprintf(fp,
                " %7.3lf %7.3lf %7.3lf %7.3lf\n",
                  dx_out[3] ,dx_out[4],dx_out[6] ,dx_out[7]);
          fprintf(fp,
                " %7.3lf %7.3lf %7.3lf %7.3lf \n",
                  std_dev[3],std_dev[4],std_dev[6],std_dev[7]);
        }
        /* Prints values for car number 2 */
        if (icar == 2)
        {
                if (iprint == 1)
            {
      if (icheck == 1) printf("\n icar=2, xprev,xlead = %7.3lf %7.3lf \n",xprev,xlead);
      printf("\n x2 = %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.3lf \n",
              xout[1] ,xout[2] ,xout[3] ,xout[4] ,
              xout[5] ,xout[6] ,xout[7] , e p s ) ;
      printf(" dx = %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf\n",
              dx_out[ 1] ,dx_out[2] ,dx_out[3] ,dx_out[4] ,
              dx_out[5] ,dx_out[6] ,dx_out[7]);
      printf(" sd = %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf\n",
              std_dev[1],std_dev[2],std_dev[3],std_dev[4],
              std_dev[5],std_dev[6],std_dev[7]);
            }
      fprintf(fp,
          " %7.2lf %7.2lf %7.2lf %7.2lf %7.3lf\n",
              xout[3] ,xout[4] ,xout[6] ,xout[7] , e p s ) ;
      fprintf(fp,
          " %7.3lf %7.3lf %7.3lf %7.3lf\n",
              dx_out[3] ,dx_out[4] ,dx_out[6] ,dx_out[7]);
      fprintf(fp,
          " %7.3lf %7.3lf %7.3lf %7.3lf\n",
              std_dev[3],std_dev[4],std_dev[6],std_dev[7]);
    }

    /* Prints values for car number 3 */
    if (icar == 3)
      {
            if (iprint == 1)
            {
      if (icheck == 1) printf("\n icar=3, xprev,xlead = %7.3lf %7.3lf \n",xprev,xlead);
      printf("\n x3 = %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.2lf %7.3lf\n",
              x o u t [ l ] ,xout[2] ,xout[3] ,xout[4] ,
              xout[5] ,xout[6] ,xout[7] , e p s ) ;
      printf(" dx = %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf\n",
              dx_out[ 1] ,dx_out[2] ,dx_out[3] ,dx_out[4] ,
              dx_out[5] ,dx_out[6] ,dx_out[7]);
      printf(" sd = %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf %7.3lf\n",
              std_dev[1]*3.,std_dev[2],std_dev[3],std_dev[4],
              std_dev[5],std_dev[6],std_dev[7]);
            }
      fprintf(fp,
        " %7.2lf %7.2lf %7.2lf %7.2lf %7.3lf\n",
              xout[3] ,xout[4] ,xout[6] ,xout[7] , e p s ) ;
      fprintf(fp,
        " %7.3lf %7.3lf %7.3lf %7.3lf\n",
              dx_out[3] ,dx_out[4] ,dx_out[6] ,dx_out[7]);
      fprintf(fp,
        " %7.3lf %7.3lf %7.3lf %7.3lf\n",
              std_dev[3],std_dev[4],std_dev[6],std_dev[7]);
    }

  }
    time=t;
/*  printf("\n INTEGRATE THE STATE VECTOR, time = %5.3lf\n",time);   */
  rk4(x[icar],xdot,NUM_STATE,u[icar],t,dt,xout,simple_long);
```

```
    simple_long(t,xhat[icar],u[icar],xhat_dot);/* xhatdot current time */
        t=time;         /*      set time back to value before earlier rk4 */

        rk4(xhat[icar],xhat_dot,NUM_STATE,u[icar],t,dt,xhat_out,simple_long);

    for (j=1; j<=NUM_STATE; ++j) x[icar][j] = xout[j];
    for (j=1; j<=NUM_STATE; ++j) xhat[icar][j] = xhat_out[j];
/*      printf("Integrate estimate of car%d ,time = %5.2lf s\n",icar,t); */
/*              Now advance the Covariance to the next data time */
        mateq(cov2_post,covariance_post[icar]);

        transition(dd,t,
                x[icar],
                u[icar],
                xhat_dot);

        cov_timeup(icar,uu,sig_bump,F,G,G1,cov2_post,cov2_pre);
        mateq(covariance_pre[icar],cov2_pre);
    /* update other variables for next time step */

    }   /* end of car loop */

/* update variables and control flags for next time step */
  if (ctrl_flag == 0)
    ctrl_flag = ctrl_freq - 1;
  else
    --ctrl_flag;

  if (wrt_flag == 0)
    wrt_flag = wrt_freq - 1;
  else
    --wrt_flag;

  t += dt;
  } /* end of time loop */

          printf("\n  eps-avg,  eps_rms[1] = %8.3lf %8.3lf \n",xavg[1],xrms[1]);
          printf("  eps-avg,  eps_rms[2] = %8.3lf %8.3lf \n",xavg[2],xrms[2]);
          printf("  eps-avg,  eps_rms[3] = %8.3lf %8.3lf \n",xavg[3],xrms[3]);
        printf("\n sig(x_rel,v_rel,m_air,w_e)  =  %5.1lf,%5.1lf,%5.1lf,%5.1lf
\n",std_dev_data[1],std_dev_data[2],std_dev_data[3],std_dev_data[4]);

        printf(" sig(x_car,v_car,w_w,Acc,T_b)  =  %5.1lf,%5.1lf,%5.1lf,%6.1lf,%6.1lf
\n",std_dev_data[5],std_dev_data[6],std_dev_data[7],std_dev_data[8],std_dev_data[9]);
          printf("\n[s**2+(q1+q3)*s+(q2+cp)+q4/s]x = [ka*s**2+q1s+q2]xp +
[cf*s**2+q3*s+cp]xL\n");
          printf("    q1,q2,q3,q4 = %8.3lf %8.3lf %8.3lf %8.3lf \n",
                values.q1_dump,
                values.q2_dump,
                values.q3_dump,
                values.q4_dump);
        printf( "       ka,cf,cp = %8.3lf %8.3lf %8.3lf \n",
                values.ka_dump,
                        values.cf_dump,
                        values.cp_dump);

/* clean up chores before exiting program */
   fclose(fp_out);
   fclose(fp);
} /* end of simulation program */
```

## APPENDIX 3 - ESTIMATED DYNAMIC PARAMETERS FROM COMPUTER CODE

The C-code simple-long was written by Patrick Yip from **Fortran** programs originated by D. Swaroop and V. Garg. Its purpose is to **find** the time-derivatives of the seven state elements of a platoon vehicle. It uses two numerical tables, for the following computations:

1. Pump torque and turbine torque as tabular functions of engine speed, wheel speed and gear-reduction ratio

2. Engine mass-flow rate and acceleration as tabular functions of accelerator displacement, air mass, engine speed and pump torque

In addition, analytical functions are used to calculate three other accelerations:

1. Vehicle acceleration as a nonlinear analytical function of tire forces, air drag and rolling friction forces,

2. Wheel acceleration as a nonlinear function of tire forces and brake torque,

3. Rates of accelerator and brake torque as nonlinear functions of command values, limit values and time-constants.

The development of the linear dynamic equations is equivalent to the task of finding the elements of the transition and control matrices in the difference equation,

$$x(n+1) = F\,x(n) + G\,u(n)\,,$$

where x is the seven-dimensional longitudinal statae, and $u$ is the three-dimensional control. The computation of $F$ and G is shown in the function **transition.c** described in Appendix 2. A typical run determines the matrices at intervals of 0.1 s, typical variations of which are shown on the following page.

Time = 10.00

$$F = \begin{vmatrix} .491 & -.005 & 0 & 0 & 0 & .001 & 0 \\ 3.633 & .722 & 0 & 0 & .679 & 0 & 0 \\ 0 & 0 & .984 & 0 & .005 & 0 & 0 \\ 0 & 0 & .010 & 1.000 & 0 & 0 & 0 \\ 0 & .093 & 4.003 & 0 & .760 & 0 & -.004 \\ 0 & 0 & 0 & 0 & 0 & .403 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .875 \end{vmatrix}$$

Time = 10.10

$$F = \begin{vmatrix} .486 & -.005 & 0 & 0 & 0 & .001 & 0 \\ 3.632 & .716 & 0 & 0 & .695 & 0 & 0 \\ 0 & 0 & .986 & 0 & .005 & 0 & 0 \\ 0 & 0 & .010 & 1.000 & 0 & 0 & 0 \\ 0 & .095 & 3.975 & 0 & .755 & 0 & -.004 \\ 0 & 0 & 0 & 0 & 0 & .403 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & .875 \end{vmatrix}$$

$$G = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ .597 & 0 & 0 \\ 0 & 125 & 0 \end{vmatrix}$$

## REFERENCES

Chang, K. S. et al., "Automated Highway System Experiments in the PATH Program," IVHS Journal, Vol. 1, No. 1, 1993, pp 63-87.

Finch, J. R., and Smith, J. P., "Multidisciplinary Investigations to Determine Relationship Between Vehicle Defects, Failures and Vehicle Crashes," Report DOT-HS-800 550, June 1971

Grandel, J., "Investigation of the Technical Defects Causing Motor Vehicle Accidents," SAE International Congress and Exposition, Detroit, MI, 1985.

Hatch, W. and DeArmon, J., "Analysis of On-Road Failure Data," Final Rept. ASGI-TR-77-36, DOT-HS-802 360, 1977.

Hedrick, J. K., et al., "Longitudinal Control Development for IVHS Fully Automated and Semi-Automated Systems - Phase I," Final Report, Dept. of Mechanical Engineering, University of California, Berkeley, January 31, 1994.

McMahon, D. H., Narendran, V. K., Swaroop, D., and Hedrick, J. K., "Longitudinal Vehicle Controllers for IVHS: Theory and Experiment," Automatic Control Conference, 1992.

Schmidt, D. N., Raley, W. L., Long, W. R., and Holter, L. C., "Vehicle Disablement Study, Executive Summary," Report No. DOT-HS-261-3-771, Traffic Safety Corp., Palo Alto, CA, January 1974.

Treat, J. R., et al., "Tri-Level Study of the Causes of Traffic Accidents - Executive Summary," DOT HS-805 099, Final Report, Institute for Research in Public Safety, Indiana University, May 1979.