This paper has been mechanically scanned. Some errors may have been inadvertently introduced.
Regulation Layer Software Integration

Akash R. Deshpande

California PATH Research Report
UCB-ITS-PRR-99-34

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.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California. This report does not constitute a standard, specification, or regulation.

Report for MOU 335

November 1999
ISSN 1055-1425
Regulation Layer Software Integration

MOU 335 Final Report
California PATH
University of California at Berkeley

Contact: Dr. Akash R. Deshpande
akash@path.berkeley.edu

1 Introduction

California PATH has developed vehicle control algorithms for lateral and longitudinal control. This technology enables fully automated multi-car platoon operation with vehicles traveling at close spacing at freeway speeds with computer-controlled throttle, brake and steering. This development effort culminated in the highly successful Demo 97 in San Diego in August 1997.

Significant additional demands are already being placed on the regulation layer functionality. The themes underlying these requirements are:

- Higher levels of system integration
- More flexible, event-oriented functionality, and
- More modular, structured implementations

The regulation layer software integration work funded by CALTRANS under MOU 335 in year the 1997-98 is a natural extension of previous work done at PATH. It can be viewed as a process of maturing the vehicle control technology developed at PATH into the second generation by refining and restructuring it to meet the future demands on regulation layer functionality. While the first generation development methodology can be said to be experience-based, the second generation development has endeavored to combine this experience with formal methods-based approaches for the design and implementation of the real-time control systems. In particular, we have used the Teja tool set for real-time control system development and deployment.

Some of the benefits derived from using Teja are:

1. Simplified logical structure of the controller code
2. Simplified real-time structure of the controllers
3. Simplified interaction structure of the controllers
4. Flexible system architecture and easy system integration
5. Visual, object-oriented, real-time modeling

We believe that our work has overcome some of the drawbacks of the previous approach such as added costs due to hardware duplication, lack of integration of the different controllers, and lack of information access for higher level planning and supervisory controllers. This work has introduced additional benefits such as tighter system integration, event-oriented system behavior, intermediate or partial automation functionality features, fault management capabilities, and easy plug-in and plug-out of different subsystems. The learning curve for new users of the software will be short, and the software will be easier to modify and maintain.

1.1 Background

This project was funded under the CALTRANS/California PATH Memorandum of understanding No. 335 during the period 1 September 1997 through 31 August 1993.

The project was led by Dr. Akash R. Deshpande. Dr. Rajesh Rajamani, Dr. Han-Shue Tan, Aniruddha Pant, Paul Matacz, Marco Zandonadi, and Marius Bozga were on the project team.

The project reuses the sensor and actuator interface software developed by California PATH for the DEN1097 automated highway system demonstration on Highway I17 in San Diego during August 1997. This software is provided on the QNX operating system. It interfaces with the magnetometers, radars, and the various internal sensors such as wheel speed, engine speed, steering angle, etc, as well as with the actuators such as throttle, brake, and steering.

In this project, we wrote an interface process that communicates between the driver process described above and the controller process described below. The interface process is required because the driver process works in a blocking communication mode while the controller process works in a non-blocking communication mode.

The controller process is provided in three flavors: lateral control only, longitudinal control only, and integrated lateral and longitudinal control.

On the lateral control side, the lateral sensor processors use the magnetometer readings to estimate earth magnetic field, correct for earth, and estimate the car's lateral displacement as well as the lateral speed. Two lateral sensors are provided. One works with the left and center magnetometers and the other works with the center and right magnetometers. The lateral controller uses the estimated lateral position and speed from both sensor processors and to compute the desired steering angle using a proportional-derivative controller.

On the longitudinal control side, the car is permitted to be either the lead car of a platoon or a follower car in the platoon. In the case of a lead car, the desired longitudinal trajectory is provided in a file. In the case of a follower car, the desired longitudinal trajectory is computed using the range and range rate to the preceding car and the lead car in the platoon. All sensor measurements such as engine speed, wheel speed, intake manifold...
pressure, brake pressure, radar range and range rate, are filtered using sensor processors. The upper controller determines the desired acceleration to be applied at each instant. The lower controller uses sliding mode feedback algorithms to regulate the car to the desired behavior.

In each, monitors are used for in-memory logging of the system performance.

1.2 Vehicle Configuration

Figure 1 shows the schematic of the car used in the testing. The schematic shows the magnetometer sensors at the front and rear, and the radar sensor at the front of the car. (The magnetometer sensors on the rear of the car were not used in this project.) The car is equipped with a communication tranceiever, and throttle, brake and steering actuators.

![Figure 1: Schematic of car used in the project](image)

2 Performance

Regulation layer software is logically divided into two main parts, longitudinal control and lateral control. Both types of control systems were tackled in the current project and effort was made to produce a single software unit with tight coupling between the two parts with the ability to access information across them.

2.1 Longitudinal Control

Two main experiments were performed for testing the longitudinal part of the regulation layer software. One was with higher cruise speed and is called fast trajectory in this report. The other was with a lesser cruise speed and is called slow trajectory in this report. The input to the software in both cases for generating trajectories is the cruise speed and the time in which to attain that speed. This enables the software, specifically the Trajectory component, to calculate initial acceleration required. The time for maintaining cruise speed
and the time to decelerate to zero speed is specified as well, and the *Trajectory* component generates the required acceleration profile.

The results obtained in these experiments are shown in the following subsections.

### 2.1.1 Fast Trajectory

In the fast trajectory experiment, the cruise speed was specified as 17 m/s, acceleration time as 15 s, cruise time as 10 s and deceleration time as 10 s.

Figure 2 shows the graph of desired speed and actual speed with respect to time. It can be seen that the car follows the desired trajectory closely. The two curves differ when gear change takes place at the 10 s mark.

![Figure 2: Fast Trajectory](image)

Figure 3 shows the graph of desired acceleration and computed acceleration with respect to time. It can be seen that the computed acceleration curve has a dip when gear change takes place. Subsequently, the car cannot provide the desired acceleration and significant error shows up.

Figure 4 shows the throttle angle provided for achieving the desired trajectory. It can be seen that when the demanded acceleration reaches a peak around 15 s mark, the throttle angle saturates.

Figure 5 shows the desired torque for achieving the trajectory. Note the similarity to the acceleration profile of Figure 3.

Figure 6 shows the comparison of actual brake pressure and desired brake pressure for the trajectory. It can be seen that brake control comes into picture mainly at the end of the trajectory, when the car is decelerating, as it should be.

Figures 7 and 8 show the engine speed in rpm and times when gear changes take place for the fast trajectory respectively. The slope of the engine speed profile in Figure 7 indicates the acceleration the car is moving with. It can be seen that the profile has positive slope in the initial acceleration phase, flat, profile for cruise time and negative slope profile for deceleration range.
2.1.2 Slow Trajectory

In the slow trajectory experiment, the cruise speed was specified as 10 m/s, acceleration time as 20 s, cruise time as 20 s and deceleration time as 10 s. Following figures give the graphs for slow trajectory, similar to the fast trajectory.

Figure 9 shows the comparison of the actual speed of the car and desired speed for the trajectory. It can be seen that the car follows commanded speed closely.

Figure 10 shows the comparison of computed acceleration and desired acceleration. Figure 11 shows the throttle angle of the vehicle for slow trajectory. It can be seen that the maximum magnitude of the throttle angle is far less in slow trajectory case than fast trajectory case 4. Figure 12 shows the desired torque for the slow trajectory. Figure 13 shows the comparison of actual and desired brake pressure for slow trajectory. Figure 14 shows the engine speed. Figure 15 shows the time at which the gear change occurs for the slow trajectory.
2.2 Lateral Control

The function of lateral control is produce steering action so that the car follows the required trajectory along the lane. In the case of magnet marker systems, the trajectory is the curve formed by magnets embedded on the centerline of the lane. The car is equipped with three magnetometer sensors on the front bumper. Lateral control works by using the relative strength of magnetic fields due to the markers at different sensor locations. One of the hurdles of magnetic sensing is that the earth magnetic field is not small compared to the magnetic field of the markers. So an earth estimation procedure is required for predicting the lateral position of the car using magnetic markers. The main aspects of lateral control can be summarized as:

1. Lateral sensing

2. Lateral sensor processing, including earth estimation and computation of lateral position of the car with respect to magnetic marker line
2.2.1 Lateral Sensing

There are three magnetometers on the front and three magnetometers on the rear of the car. In the current project only the front magnetometers were used. In this report the magnetometers will be called Left, Center and Right, seen from the point of view of the driver.

2.2.2 Earth Estimation

The earth estimation algorithm is adopted from [1]. One can see that the magnitude of the vertical earth magnetic field component, \(H_{e\text{arth}}\), and the horizontal field component \(H_{h\text{e\text{arth}}}\) =
\[ \sqrt{x_{\text{earth}}^2 + y_{\text{earth}}^2}, \] where \( x_{\text{earth}} \) is the magnetic field in x-direction and \( y_{\text{earth}} \) is the magnetic field in y-direction is almost constant in a small geographical region. The expressions for earth estimation are given by [1]. For earth field in x-direction,

\[ x_{\text{earth}} = \bar{x}_{1} \tag{1} \]

where, \( \bar{x}_{1} \) denotes the moving average of the sensor readings in x-direction.

For earth field in y-direction,

\[ y_{\text{earth}} = \frac{1}{2} (\bar{y}_{1} + \bar{y}_{2}) \tag{2} \]

where, \( \bar{y}_{i}, \ i = 1, 2 \) denote the moving averages of the two sensor readings in y-direction. Here the suffix 1 denotes the left sensor and suffix 2 right sensor in the sensor processor.

Using the estimated earth magnetic fields, the sensor readings are corrected on-line before computing the lateral displacement of the car from the magnet’s centerline. Thus,

\[ x_{\text{corrected}} = x_{1} - x_{\text{earth}} \tag{3} \]
For simplicity of notation just \( x_1 \) and \( y_1 \) are used in this report, but they should be taken to mean the corrected magnetometer readings.

Figures 16, 17 and 18 show sensor readings in the left, center and right magnetometers respectively. Note that when the experiment was performed, the car was positioned between left and center magnetometers and so right magnetometer readings are almost constant at earth values. Figures 19 and 20 show the estimated earth magnetic fields using left-center magnetometers and center-right magnetometers, respectively.

### 2.2.3 Displacement Computation

The lateral distance between the magnet and the centerline of the magnetometer pair under consideration is computed using the magnet sensor reading corrected for earth magnetic
field. The expressions for this distance are given by (see [1]),

$$d_y = \frac{L}{2} \left( \frac{-\tan \beta_1 + \tan \beta_2}{\tan \beta_1 + \tan \beta_2} \right)$$

(5)

where,

$$\cos \beta_i = \cos \alpha_i \frac{y_i}{\sqrt{x_i^2 + y_i^2}}$$

(6)

$$\tan \alpha_i = \frac{3\eta_i + \sqrt{9\eta_i^2 + 8}}{4}$$

(7)

$$\eta_i = \frac{|z_i|}{\sqrt{x_i^2 + y_i^2}}$$

(8)

In the above equations, L is the distance between two magnetometers and $x_i$, $y_i$ and $z_i$ are the corrected magnetic fields for the two sensor processors, for $i = 1, 2$. Figure 13 shows
the lateral displacements of the centerline of the car with respect to the magnetic marker line. Note that a particular sensor processor is active only when the magnet is in between its two magnetometers. Thus, if left-center sensor processor is active then center-right sensor processor is not. Figure 23 shows the computed displacement. This is filtered by averaging the displacement over last 10 readings, and then used in the controller for computing the steering angle required.

Figure 24 shows the filtered lateral speed. The expression for filtering is given by equation 18, see [1].

### 2.2.4 Steering Angle Computation

A PD controller has been used in the controller. The stability of this algorithm has been proved in [1].

\[
\begin{align*}
\dot{\gamma} &= \frac{J}{\mu} \\
\dot{\gamma} &= \frac{m}{\mu} \\
q_0 &= \frac{C_f}{\dot{m}v} + \frac{C_f l_f l_f}{\dot{j}v} \\
q_1 &= \frac{C_f C_r (l_r + i_s)(l_r + l_f)}{\dot{m}v} \\
q_2 &= \frac{C_f C_r (l_r + l_f)}{\dot{m}v} \\
p_1 &= \frac{(C_r + C_f)}{\dot{m}v} + \frac{(C_r l_r + C_f l_f l_f)}{\dot{j}v} \\
p_2 &= \frac{(C_r l_r - C_f l_f)}{\dot{j}} + \frac{(C_r C_f (l_r l_r + l_f l_f) + 2C_r C_f l_r l_f)}{\dot{m}v} 
\end{align*}
\]
In the above equations, $J$ is normalized inertia of the car. $m$ is the normalized mass. $C_f$, $C_r$ are the cornering stiffnesses of front and rear tyre respectively. $l_r$, $l_f$ are distances to rear and front tyres from center of gravity of the car respectively. $\mu$ is the road adhesion factor.

To take into account the unmodeled dynamics of the car a parameter adaptation scheme [1] and integral control term is being tested. The derivative term in the control expression requires the computation of lateral velocity of the car, but the numerical derivative of the lateral displacement is too noisy to be directly used in the controller. So an observer has been designed [1],

$$d_s = -\nu \delta_{syn} + k_s \left( d_s - \dot{d}_s \right).$$

Here, $k_s$ is a positive parameter.

Figure 25 shows the computed steering angle.
Figure 17: Center Magnetometer Sensor Readings

Figure 18: Right Magnetometer Sensor Readings
Figure 19: Estimated earth magnetic field for left-center magnetometer sensor processor

Figure 20: Estimated earth magnetic field for center-right magnetometer sensor processor
Figure 21: Corrected magnetic field readings

Figure 22: Lateral displacement as computed by both magnetometers
Figure 23: Filtered lateral displacement used in controller

Figure 24: Filtered lateral speed

Figure 25: Steering Angle
3 Application Software

This section explains the structure of the control software designed in the Teja environment. For detailed explanation of procedures and basic structure of the Teja modeling environment, refer to the Teja reference manual [2].

3.1 Package Design

Figure 26 shows different packages defined in the application.

3.2 Class Design

Figure 27 shows the schematic structure of the classes in the software and the information flow between different component classes. The components are packaged in various packages, namely,

1. **Sensors**: These are the components representing the corresponding physical sensor components in the car. They take the value of a variable needed in the regulation layer software from the sensor database in the car, through an interface component called as DatabaseReader.

2. **Comm**: Currently this class contains only one component called CommunicatedVariables. This provides the values of variables which are communicated between different cars.

3. **Sensor processors**: This package contains all the sensor processors which filter and process the data acquired by various sensors for further use in controllers. Examples of components in this package are, LateralSensorProcessor and RadarSensorProcessor.

4. **Controllers**: This package contains different controllers used in the regulation layer software for longitudinal as well as lateral control. Examples of the components in this class are LateralController, LowerController, UpperController.

5. **Monitors**: The components in this class are used for keeping track of the values of variables of interest in the program, which can be used for plotting the results or further debugging.

3.3 System Architecture Design

Figure 28 shows the different servers used in the software structure. Server *lateral* has the component instantiations used for lateral control. For exploded view of the lateral server refer to Figure 29. The server *longitudinal* has the component instantiations used for longitudinal control. For exploded view of the longitudinal server refer to Figure 30. The server *reglayer* is used to integrate these two types of controllers together to have a single regulation layer control software program.
run online on car or in simulation mode on a desktop. The next layer of components is of sensors which read various variables from the data reader and pass them to the next layer components in the information flow diagram, i.e., to the sensor processors. The component layer after sensor processors is of controllers. They use the information from the sensor processors and compute commands for actuators. The component trajectory generates the reference trajectory for the car, and the information from this component is used by the other controllers in the server. The component Monitor saves the required data in memory and then dumps it on a file at the end of the run.

3.5 Models

In Teja designer, models of different components are designed using the hybrid state machines approach. A component state is represented by a discrete state and the states are changed by means of events. State transitions can be of three types: proaction, reaction and response. Proaction happens when some logical or real time guard within the same component becomes
true. Response occurs when some event by some other component is sent to the component in which response occurs. Reaction is similar to response but it occurs across different servers.

Structure of various components used in the regulation layer software is shown in figures given below. Brief explanation of the state machine dynamics is given as well.

3.5.1 Sensors

Figure 31 shows the state machine setup for brake sensor component. When read long event happens in the data reader component, brake pressure value is copied on the variable brake pressure in this component which will be further used in sensor processors. After copying the variable the component comes back to idle state on the event new data.

Figure 32 shows the state machine setup for car sensor component. When read long event happens in the data reader component, platoon position and car speed values are copied on the variables in this component. After copying the variable the component comes back to idle state on the event new data.

Figure 33 shows the model structure of the component data file reader. On event read mag the models reads in the sensor data from magnetometers and outputs an event called read mag which induces reaction in other components. Similarly lateral and longitudinal variables are read on the corresponding events.

Figure 34 shows the model structure of the component database reader. The component goes to busy state from idle state when it receives event read-long from the data reader. It
copies the specified variables from data reader and immediately comes back to idle state on the proaction new-data.

The component LateralSensor 36 reads the magnetic data on the transition read_mag and transmits the event new_data.

Radar copies radar range and range rate from the data reader shown in Figure 37.

The state machine structure of component Transmission is shown in Figure 38. The component changes state in response to event read_long in data reader. If there has been gear change then it transmits event gear-change.

3.5.2 Filters

In this section hybrid state machine structure of sensor processors in the reglayer applications are described in short.

Figure 39 shows structure of engine speed filter. This component smoothes out the engine speed sensor readings.

Lateral sensor processor is shown in Figure 40. On the event, new_data magnetometer sensor readings are copied on arrays defined in the model. This particular event occurs every 2 ms. The other event estimate is a response to a transition in Lateral Controller. This runs every 20 ms. The periods are variables and can be varied on the car to tune the software. On the event estimate main bulk of the work is done.

- Earth estimation
Figure 30: Longitudinal control process component architecture.

- Apply correction to the sensor readings
- Estimation of lateral position

Figure 41 shows structure of engine manifold pressure filter. It filters the engine manifold pressure reading to smoothen them out for further use in the software.

Figure 42 shows the state machine structure of radar sensor processor. This component filters the noisy radar reading to give more quiet radar range and range rate. When radar sensor gives zero reading it is considered to be in potential fault. In potential fault state the filtering is done using stored radar reading. If the zero radar reading persists for more than some time, say 1 second, then a fault is declared. Further fault management is needed, which is not included here.

### 3.5.3 Controllers

Figure 44 shows state machine structure of lateral controller. The model goes from idle to busy state on the event control and it transmits event estimate which causes a reaction in lateral sensor processor and lateral sensor processor estimates lateral displacement. From state busy the model goes to state idle on the event estimate which computes the control action and transmits event lateral command which causes database to be written by new computed value of control action.

Figure 45 shows the hybrid state machine diagram of the component Trajectory. The component is designed so that when user gives cruise speed, acceleration time and cruise time and deceleration time, it calculates the smooth acceleration that should be applied to the car to get the required trajectory.
Figure 46 shows the state machine diagram of the component \textit{Upper controller}. The function of this component is to calculate the required acceleration of the vehicle, using lead car speed, lead car acceleration, preceding car speed and preceding car acceleration. It takes this information from the component, communicated variables.

It can be seen that the component starts working when it receives the event begin trajectory from the component trajectory. Also, note that there are two discrete states corresponding to computation of acceleration in normal mode and the other is computation of acceleration when gear change is occurring. When the gear change occurs the engines torque drops by some amount and this needs to be considered when acceleration is being computed.

Figure 47 shows the structure of the component \textit{Lower controller}. The function of the lower controller is to compute required brake pressure and the throttle angle to achieve the acceleration computed by upper controller. These variables are computed on the event \textit{controller poll} and they are written to the database of the car on the same event.

### 3.5.4 Monitors

Figure 48 shows the state machine diagram of the component monitor. The function of this component is to store the specified variable values in memory while the real time experiment is being conducted and dump all the stored data onto a file at the end of the run. The proaction \textit{initialize} initializes the component while taking it from the state \textit{start} to \textit{idle}. In the idle state, lateral variables are stored on the response \textit{lateral command} and longitudinal variables are stored on the response \textit{controller poll}.
3.5.5 Communication

Figure 49 shows the component communicated variables. The function of this component is to provide the values of communicated variables to the regulation layer software. The experiments for this project were carried out using only one car so this component was not used in its full functionality, however in future when the regulation layer software will be tested on platoons of cars, this component will have more functionality.

4 Conclusions

The objectives of the project have been adequately met, namely, to re-implement and integrate the Demo97 software functionality using visual tools that provide an object-oriented software framework and flexible, real-time event-driven operation. The learning curve for new users of the software will be short, and the software will be easier to modify and maintain.
Figure 33: Data file reader model.

References


Figure 34: Database reader model.
Figure 35: Engine sensor model.
Figure 36: Lateral sensor model.
Figure 37: Radar model.
Figure 38: Transmission model.
Figure 39: Engine speed filter model.
Figure 40: Later sensor processor model.
Figure 41: Manifold pressure filter model.
Figure 42: Radar sensor processor model.
Figure 43: Wheel speed filter model.
Figure 44: Lateral controller model.
Figure 45: Trajectory model.
Figure 46: Upper controller model.

Figure 47: Lower controller model.
Figure 48: Monitor model.

Figure 49: Communicated variables model.
Interfacing Legacy Car Control Software with Teja

Introduction

PATH automated vehicles are driven by a set of concurrent processes, running on the QNX platform (a detailed description of the software structure can be found in [1]). They get input from sensors and write outputs to actuators. The means by which processes communicate data to/from each other are given by a special process, called database. The database manages an area of shared memory in which the I/O variables can be stored and retrieved by processes. It can also alert a process when a variable has changed. This feature allows the control software to respond to changes as soon as possible.

The purpose of this project is to replace the existing lateral and longitudinal controller software (which are currently implemented in C as two separate processes), with a Teja server (a detailed description of the Teja real-time design and implementation tools can be found in [2]). Teja assists in developing software from the early-design phase to the real-time implementation phase. The software architecture can be described using a graphic paradigm, which eases the learning curve of those who want to study the system.

Interfacing with Existing Software

Processes can read/write values from/to the database, using an API based on the following QNX inter-process communication primitives: Receive(), Send() and Reply(). They implement blocking communication. The Receive() function waits until a message is received from a remote process. The Send() function sends a message to a remote process. It waits until the remote process acknowledges the message with the Reply() function.

Blocking communication should not be used in Teja programs, because it interferes with the semantics of the system. Every Teja process is structured as a set of concurrent components. Each component should be able to respond to events as soon as possible. Each action in a component should be as brief and fast as possible, so that other actions can be executed in a timely manner. Using Receive() or Send() to interface directly with the database would inhibit this behavior.

The solution to this problem is to introduce an interface program behaving as a bridge between the Teja server and the database. On one hand the interface can connect to the database and receive triggers when variables change, using the blocking primitives. On the other hand it can communicate with the server using software signals. Teja processes support both TCP/IP communication and signals/interrupts. In this case TCP/IP can not be used because it is requires too much CPU time and therefore it’s not suitable for real-time applications.

The problem with signals is that they don’t carry information other than the signal itself. They can be used to warn the Teja server that some variable has changed in the database, but they cannot carry the name or the value of the variable. To overcome this problem the following technique is used: after sending the signal, the interface uses Send() to communicate the name of the changed variable to the Teja server. The server issues a Receive() in the action of the transition responding to the signal.
Immediately after that it sends a Reply() to unblock the Send() in the interface.

Using QNX blocking primitives in this way doesn’t interfere with Teja semantics. The Teja server does not wait in Receive() for a long time and the interface doesn’t block in Send() because the whole communication process is synchronized through the signal: Send(), Receive() and Reply() happen one immediately after the other because both processes are ready to communicate.

The relationships among interface, database and server are shown below.

The interface is triggered by the database when a variable it is interested in changes. The name of the variable is communicated via a special trigger message. After being triggered, the interface sends a signal to the server and communicates the name of the changed variable. The server closes the loop by reading the right value from the database.

Writing to the database does not require any synchronization.

**Implementation of the Interface Process**

The interface between the server and the database is a pure C program and it has the following implementation (important code sections have been italicized and extensive comments have been added to explain the code):

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/kernel.h>
#include <sys/proxy.h>
#include <sys/name.h>
#include <signal.h>
#include <setjmp.h>
#include <local.h>
```
#include <sys_rt.h>
#include <sys_list.h>

#include "db_comm.h"
#include "db-clt.h"
#include "track.h"
#include "veh_iols.h"
#include "clt_vars.h"

#include <errno.h>

static int sig_list[] =
{
    SIGINT,
    SIGQUIT,
    SIGTERM,
    ERROR
};

static jmp_buf exit-env;

static void sig_hand(int code);

void Error(char* message)
{
    fprintf(stderr, "Interface: %s
", message);
    exit(1);
}

void main(int argc, char *argv[])
{
    char hostname[MAXHOSTNAMELEN+1]; /* host name */
    db_clt_typ *pclt; /* database client pointer */
    pid_t readersid; /* diadem application pid */

    printf("initializing diadem interface...");
    fflush(stdout);

    /* Attaching the interface name on the QNX process name table. This * allows the server to get the interface pid by calling * qnx_name_locate() with "diadem-database-interface" argument. */
    
    if (qnx-name-attach(0, "diadem-database-interface") == -1) {
        Error("cannot attach diadem-database-interface");
    }

    /* At every second it tries to get the server pid by calling * qnx_name_locate() with "diadem_database_reader" argument. * The loop is exited when the server process issues a
* `qnx_name_attach()` with 'diadem-database-reader' argument.
*/

```c
for (;;) {
    readersid = qnx-name-locate(0, "diadem-database-reader", 0, NULL
    if (readersid != -1)
        break;
    sleep(1);
}
```

/* Log in to the database (shared global memory). */
```c
sprintf( hostname, "%lu", getnid() );
if(! ( pclt = clt_login( argv[0], hostname, DEFAULT-SERVICE, 
    COMM_QNX_XPORT)) == NULL )
    Error("cannot login into the database");
```

/* Setting up triggers, so that the database can notify the interface 
* whenever one of these variables changes 
*/
```c
if( clt_trig_set(pclt, DB_LAT_INPUT_SENSORS_VAR, 
    DB_LAT_INPUT_SENSORS_TYPE) == FALSE ) ||
    clt_trig_set(pclt, DB_LAT_INPUT_MAG_VAR, 
    DB_LAT_INPUT_MAG_TYPE) == FALSE ) ||
    clt_trig_set(pclt, DB_LONG_INPUT_VAR, 
    DB_LONG_INPUT_TYPE) == FALSE ) {
    clt_logout( pclt );
    Error("cannot set triggers");
}
```

/* Catch the SIGINT, SIGQUIT, and SIGTERM signals. When one of these signals 
* is caught, a clean exit is performed, by logging out of the database. 
*/
```c
if( setjmp( exit-env ) != 0 ) {
    if (pclt != NULL)
        clt_logout( pclt );
    exit( EXIT-SUCCESS );
} else
    sig_ign( sig_list, sig_hand );
```

printf("done.
");

/* Main loop */
```c
for (;;) {
    int dummy = 0;
    trig-info-typ trig-msg;
```
int snd;

// Waiting for a trigger from the database
Receive ( 0, &trig-msg, sizeof( trig-msg ) );

// After getting a trigger, a signal is sent to the Teja server.
kill( readersid, SIGUSR1 );

// The trigger message is sent to the Teja server. The server is ready to
// Receive() the message and to Reply() to it (it's part of the response to
// the signal). For this reason this Send() returns almost immediately.
snd = Send( readersid, &trig-msg, &dummy, sizeof( trig-msg ), sizeof(dummy) );
if (snd == -1)
    Error("cannot send triggers");

}

//Signal handler for the SIGINT, SIGQUIT, and SIGTERM signals.
static void sig_hand( int code ) {
    longjmp( exit-env, code );
}

Implementation of the Teja Server-Interface Component

The DatabaseReader class in the server has the purpose of reacting to interface-generated signals, that are
of type SIGUSR1, as defined in <signal.h>.

In the server source text the following interrupt handler is defined:

void
UsrlInterruptHandler(int signal)
{
    // Generation of an Alert and its registration in the server, so that
    // the server can react as if a normal Alert has been received, instead
    // of a signal. isrAlert() maps interrupts into Alerts.
    DEM::isrAlert( newAlert( get-trigger, 0) );

    // Registration of the user defined interrupt handler is refreshed to
    // avoid the default handler from being restored.
    DEM::registerInterruptHandler(signal, UsrlInterruptHandler);
}

DatabaseReader is instantiated once in the server and its constructor function contains all the needed
initialization code for communicating both with the interface and with the database:

DatabaseReader::DatabaseReader() {
The first transition is defined as follows:
char hostname[MAXHOSTNAMELEN + 1];

printf("initializing database reader... ");
fflush(stdout);

/* Registering the USR1 interrupt handler */
DEM::registerInterruptHandler(SIGUSR1, UsrlInterruptHandler);

/* Attaching the server name on the QNX process name table. This
 * allows the interface to get the server pid by calling
 * qnx_name_locate() with "diadem-database-reader"argument.
 */
if (qnx-name-attach (0, "diadem-database-reader") == -1)
    Error("DatabaseReader",
            "cannot attach diadem-database-reader");

/* At every second it tries to get the interface pid by calling
 * qnx_name_locate() with "diadem-database-interface"argument.
 * The loop is exited when the interface process issues a
 * qnx_name_attach() with "diadem-database-interface"argument.
 */
for(;;){
    interface_pid = qnx-name-locate (0, "diadem-database-interface",0, NULL);
    if (interface_pid != -1)
        break;
    sleep(1);
}

/* Login into the database */
sprintf( hostname, "%lu", getnid() );
if (( pclt = clt_login( "diadem_database_client",
                hostname,
                DEFAULT-SERVICE,
                COMM_QNX_XPORT)) == NULL)
    Error("DatabaseReader",
            "cannot login into the database");

/* initialize the global DB client pointer */
DBClient = pclt;

printf("done.\n");
}

The Teja model for the DatabaseReader is shown below
It is a reaction and it is taken upon reception of an Alert named get-trigger (just like the one that was defined in the interrupt handler). This transition closes the interrupt management loop: when a signal is received by the server, the handler responds to it and converts it to an Alert that triggers this transition. The guard is always true, therefore the action is executed unconditionally: the receive-trigger() function is called. The function body is:

```c
int dummy = 0;
int rep;
pid_t rec;

/* When a signal-handler generated alert is received from the interface, *
 * it means that the interface is ready to Send() the name of the changed *
 * variable name. *
 * It is therefore safe to Receive(): the message will arrive almost *
 * immediately and will make Receive() return.
```
/ rec = Receive(interface_pid, &trigger, sizeof(trigger));
  if (rec != interface_pid);
    Warning("DatabaseReader",
        "unexpected trigger received");

/* Reply() is needed to unblock the Send() in the interface */
  rep = Reply(interface_pid, &dummy, sizeof(dummy));
  if (rep == -1)
    Warning("DatabaseReader",
        "trigger reply error");

The next three transitions are proactions outputting respectively the read-long, read-lat and read-mag events. The purpose of these transitions is to read the variable that has changed (as communicated by the interface). Each transition reads one variable.

The read-long transition is defined as:
The guard is true when the value returned by the `test-long-var()` is true. The function is implemented as:

```c
// If the variable name communicated by the interface is DB-LONG-INPUT-VAR then
// 1 is returned, otherwise 0 is returned.
return (DB_TRIG_VAR(&trigger) == DB-LONG-INPUT-VAR);
```

The associated action is the function `read-long-var()`, which is implemented as:

```c
db_data_typ data;
long_input_typ* p;

// Reads the DB-LONG-INPUT-VAR variable from the database using the predefined // API.
if (clt_read( pclt,
DB—LONG—INPUT—VAR,
DB—LONG—INPUT—TYPE,
&data ), == .FALSE.

Error ("DatabaseReader," 
  "cannot read DB—LONG—INPUT—VAR") :

P = (long-input-typ *) data.value.user;

// Each database variable is actually a C struct. Each value from the struct // is copied in local variables (wich are defined in the superclass of // DatabaseReader).

brake_press = p->brake_press;

eng_rpm = p->eng_rpm;
eng_press = p->eng_press;

radar_range = p->radar_data.radar_range;
radar_rangerate = p->radar_data.radar_rangerate;

gear = p->gear;
gear_ratio = p->gear-ratio;

platoon_pos = p->platoon_pos;

The transition has an output event called read-long, which is used to communicate other components in the server that a fresh value of the DB—LONG—INPUT—VAR variable is available in the local variables of the DatabaseReader class.

The two proactions outputting read-lat and read-mag have exactly the same structure so they won’t be described.

The last transition is taken when neither read-long, nor read-lat nor read-mag are triggered.
This case should never happen and the associated action is null.

**Bibliography**

The reglayer Application

This report has been produced automatically by the TEJA Modeling Environment.

Table of Contents

I. Packages
   1. default
   2. Controllers
   3. Monitors
   4. Comm
   5. Sensors
   6. SensorProcessors
II. Application Models
    A. Component Models
       1. Brakesensor
       2. CarSensor
       3. CommunicatedVariables
       4. DataReader
       5. DatabaseReader
       6. DatafileReader
       7. Enginesensor
       8. EngineSpeedFilter
       9. LateralController
      10. LateralSensor
      11. LateralSensorProcessor
      12. LowerController
      13. ManifoldPressureFilter
      14. Monitor
      15. Radar
      16. RadarSensorProcessor
      17. Sensor
      18. Trajectory
      19. Transmission
      20. UpperController
      21. WheelSpeedFilter
    B. Event Models
    C. Alert Models

III. System Architecture
    A. Servers
       1. lateral
       2. longitudinal
       3. reglayer
    B. Clients

The reglayer application contains the Regulation Layer Software Integration system models for the automatic lateral and longitudinal control of Buick Le Sabre vehicles. This project was funded under the CALTRANS/California PATH Memorandum of understanding No. 335 during the period 1 September 1997 through 31 August 1998.

The project was led by Dr. Akash R. Deshpande. Dr. Rajesh Rajamani, Dr. Han-Shue Tan, Aniruddha Pant, Paul Kretz, Marco Zandonadi, and Marius Bozga were on the project team.

The project reuses the sensor and actuator interface software developed by California PATH for the
DEMO97 automated highway system demonstration on Highway I17 in San Diego during August 1997. This software is provided on the QNX operating system. It interfaces with the magnetometers, radars, and the various internal sensors such as wheel speed, engine speed, steering angle, etc, as well as with the actuators such as throttle, brake, and steering.

In this project, we wrote an interface process that communicates between the driver process described above and the controller process described below. The interface process is required because the driver process works in a blocking communication mode while the controller process works in a non-blocking communication mode. The interface process is described here.

The controller process is provided in three flavors: lateral control only, longitudinal control only, and integrated lateral and longitudinal control.

On the lateral control side, the lateral sensor processors use the magnetometer readings to estimate earth magnetic field, correct for earth, and estimate the car's lateral displacement as well as the lateral speed. Two lateral sensors are provided. One works with the left and center magnetometers and the other works with the center and right magnetometers. The lateral controller uses the estimated lateral position and speed from both sensor processors and to compute the desired steering angle using a proportional-derivative controller.

On the longitudinal control side, the car is permitted to be either the lead car of a platoon or a follower car in the platoon. In the case of a lead car, the desired longitudinal trajectory is provided in a file. In the case of a follower car, the desired longitudinal trajectory is computed using the range and range rate to the preceding car and the lead car in the platoon. All sensor measurements such as engine speed, wheel speed, intake manifold pressure, brake pressure, radar range and range rate, are filtered using sensor processors. The upper controller determines the desired acceleration to be applied at each instant. The lower controller uses sliding mode feedback algorithms to regulate the car to the desired behavior.

In each monitors are used for in-memory logging of the system performance.

---

**Packages**

**default**

Default Application Package

Contains classes Component Event Alert

**Controllers**

Contains classes LateralController UpperController LowerController Trajectory

**Monitors**

Contains classes Monitor

**Comm**

Contains classes CommunicatedVariables

**Sensors**
Contains classes DataReader DatafileReader DatabaseReader Sensor Lateralsensor Carsensor EngineSensor Brakesensor Radar Transmission

SensorProcessors

Contains classes LateralSensorProcessor RadarSensorProcessor EngineSpeedFilter ManifoldPressureFilter WheelSpeedFilter

Application Models

Components of the reglayer Application

Class Brakesensor

Reads brake pressure from DataReader.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Superclass of Brakesensor</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>brake-pressure</td>
<td>double</td>
<td>Brake Pressure [psi]</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
</tr>
<tr>
<td>busy</td>
<td>Discrete state</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_long</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {1}

Action

(ResetVariable brake_pressure | reader->brake_pressure)
Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>new_data</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {1}

Action

Output Event

Include Files and other Preprocessor Directives

Header Code

Brakesensor();

C++ Code

BrakeSensor::BrakeSensor()
{
}

Class Carsensor

This component reads speed of the car, current steering angle and platoon position of the car from DataReader.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Superclass of Carsensor</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:car-speed</td>
<td>double</td>
<td>speed of the car. [m/s]</td>
</tr>
<tr>
<td>:steer_angle</td>
<td>double</td>
<td>current steering angle. [degrees]</td>
</tr>
<tr>
<td>:platoon_position</td>
<td>int</td>
<td>number indicating the position of the car in the platoon.</td>
</tr>
</tbody>
</table>

Discrete States
### Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_lat</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

```c
function { if - true(1) ; }
```

**Action**

- **Reset Variable** `steer_angle` `reader->steer_angle`
- **Output Event**

### Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>new_data</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

```c
boolean {1}
```

**Action**

- **Output Event**

### Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_long</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

```c
boolean {1}
```

**Action**

- **Reset Variable** `car_speed` 
  
  `(reader->wheel-speed>39.5) ? 0.0 : reader->wheel-speed`
- **Reset Variable** `platoon_position` `reader->platoon_pos`
- **Output Event**

---

Include Files and other Preprocessor Directives
Header Code

Carsensor();

C++ Code

CarSensor::CarSensor()
{
}

Class CommunicatedVariables

The communicated variables component provides the simulated communication inputs to the controller. It supplies the speed and acceleration of the preceding and the lead vehicles.

In the physical system, this component is replaced by the actual communication receiver.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Superclass of CommunicatedVariables</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>preceding_speed</td>
<td>double</td>
<td>speed of the preceding car.[m/s]</td>
</tr>
<tr>
<td>preceding_acccl</td>
<td>double</td>
<td>acceleration of preceding car.[m/s^2]</td>
</tr>
<tr>
<td>lead_speed</td>
<td>double</td>
<td>Speed of lead car.[m/s]</td>
</tr>
<tr>
<td>lead_acccl</td>
<td>double</td>
<td>Acceleration of lead car.[m/s^2]</td>
</tr>
</tbody>
</table>

Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trajectory</td>
<td>Trajectory*</td>
<td>Link to the component Trajectory.</td>
</tr>
<tr>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>Link to component car sensor</td>
</tr>
</tbody>
</table>

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>/LinkType</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>platoon_position</td>
<td>int</td>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>platoon_position</td>
</tr>
</tbody>
</table>

Continuous States
Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>Discrete state</td>
<td></td>
<td>timer' = 1</td>
</tr>
<tr>
<td>busy</td>
<td>Discrete state</td>
<td></td>
<td>timer' = 1</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>new_data</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {1}

Action

Output Event

Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_comm</td>
<td>proaction</td>
<td>/Read communicated variables from the trajectory. Works only for the leader car</td>
</tr>
</tbody>
</table>

Guard

function {if (platoon-position() == 1)
  return when-gt (timer(), 0.02, timer dot());
else
  return -1;
}

Action

<table>
<thead>
<tr>
<th>Reset Continuous State</th>
<th>Description</th>
<th>reset</th>
<th>timer</th>
<th>timer() - 0.02</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Variable</td>
<td>preceding_speed</td>
<td>trajectory-&gt;desired_speed()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset Variable</td>
<td>preceding_accel</td>
<td>trajectory-&gt;desired_acceleration()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset Variable</td>
<td>lead_speed</td>
<td>trajectory-&gt;desired_speed()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset Variable</td>
<td>lead_accel</td>
<td>trajectory-&gt;desired_acceleration()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output Event

Transition 2
<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_comm</td>
<td>response</td>
<td>Read communicated variables from the database. Works for other than the leader car. [not yet implemented]</td>
</tr>
</tbody>
</table>

Guard

boolean {platoon_position() != 1};

Action

Output Event

Include Files and other Preprocessor Directives

Header Code

```
CommunicatedVariables();
```

C++ Code

```
CommunicatedVariables::CommunicatedVariables()
{
}
```

Class DataReader

This component is a superclass for DatafileReader which reads the data from a file in simulation mode and DatabaseReader which reads data from database in the car while running in real time mode.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of DataReader</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mag_fl-x</td>
<td>double</td>
<td>voltage reading of the front left magnetometer's Z axis</td>
</tr>
<tr>
<td>mag_fl_y</td>
<td>double</td>
<td>voltage reading of the front left magnetometer's Y axis</td>
</tr>
<tr>
<td>mag_fl_z</td>
<td>double</td>
<td>voltage reading of the front left magnetometer's Z axis</td>
</tr>
<tr>
<td>mag_fc_x</td>
<td>double</td>
<td>voltage reading of the front center magnetometer's X axis</td>
</tr>
<tr>
<td>mag_fc_y</td>
<td>double</td>
<td>voltage reading of the front center magnetometer's Y axis</td>
</tr>
<tr>
<td>mag_fc_z</td>
<td>double</td>
<td>voltage reading of the front center magnetometer's Z axis</td>
</tr>
<tr>
<td>mag_fr_x</td>
<td>double</td>
<td>voltage reading of the front right magnetometer's X axis</td>
</tr>
<tr>
<td>mag_fr_y</td>
<td>double</td>
<td>voltage reading of the front right magnetometer's Y axis</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>mag_fr_z</td>
<td>double</td>
<td>Voltage reading of the front right magnetometer's z axis</td>
</tr>
<tr>
<td>mag_bl_x</td>
<td>double</td>
<td>Voltage reading of the back left magnetometer's x axis</td>
</tr>
<tr>
<td>mag_bl_y</td>
<td>double</td>
<td>Voltage reading of the back left magnetometer's y axis</td>
</tr>
<tr>
<td>mag_bl_z</td>
<td>double</td>
<td>Voltage reading of the back left magnetometer's z axis</td>
</tr>
<tr>
<td>mag_bc_x</td>
<td>double</td>
<td>Voltage reading of the back center magnetometer's x axis</td>
</tr>
<tr>
<td>mag_bc_y</td>
<td>double</td>
<td>Voltage reading of the back center magnetometer's y axis</td>
</tr>
<tr>
<td>mag_bc_z</td>
<td>double</td>
<td>Voltage reading of the back center magnetometer's z axis</td>
</tr>
<tr>
<td>mag_br_x</td>
<td>double</td>
<td>Voltage reading of the back right magnetometer's x axis</td>
</tr>
<tr>
<td>mag_br_y</td>
<td>double</td>
<td>Voltage reading of the back right magnetometer's y axis</td>
</tr>
<tr>
<td>mag_br_z</td>
<td>double</td>
<td>Voltage reading of the back right magnetometer's z axis</td>
</tr>
<tr>
<td>wheel_speed</td>
<td>double</td>
<td>Longitudinal velocity [m/s]</td>
</tr>
<tr>
<td>steer_angle</td>
<td>double</td>
<td>Steering angle of handwheel [degrees]</td>
</tr>
<tr>
<td>lat_yaw</td>
<td>double</td>
<td>Yaw rate [rad/s]</td>
</tr>
<tr>
<td>lat_yacc</td>
<td>double</td>
<td>Lateral acceleration [m/s^2]</td>
</tr>
<tr>
<td>speed_cnt</td>
<td>int</td>
<td>Longitudinal velocity count [# of clock pulses between two gear teeth]</td>
</tr>
<tr>
<td>brake_press</td>
<td>double</td>
<td>Master cylinder pressure [psi]</td>
</tr>
<tr>
<td>eng_rpm</td>
<td>double</td>
<td>Engine speed [rpm]</td>
</tr>
<tr>
<td>eng_press</td>
<td>double</td>
<td>Measured manifold pressure [kpa]</td>
</tr>
<tr>
<td>radar_range</td>
<td>double</td>
<td>Range to nearest object [m]</td>
</tr>
<tr>
<td>radar_rangerate</td>
<td>double</td>
<td>Range rate of the nearest object [m]</td>
</tr>
<tr>
<td>gear</td>
<td>int</td>
<td>Decoded transmission position</td>
</tr>
<tr>
<td>gear_ratio</td>
<td>double</td>
<td>Overall transmission ratio</td>
</tr>
<tr>
<td>/platoon-pos</td>
<td>int</td>
<td>Platoon position [1, 2, 3, 4]</td>
</tr>
</tbody>
</table>

**Discrete States**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
</tr>
</tbody>
</table>
Include Files and other Preprocessor Directives

Header Code

DataReader();

C++ Code

DataReader::DataReader()
{
}

Class DatabaseReader

This component reads data from vehicle database to be used in the reglayer application

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataReader</td>
<td>Superclass of DatabaseReader</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_mag_var</td>
<td>void</td>
<td></td>
<td>Retrieve data from the variable DB_LAT_INPUT_MAG_VAR.</td>
</tr>
</tbody>
</table>

#ifdef QNX

db_data_typ data;
lat_input_mag_typ* p;

if ( clt_read( pclt,
    DB_LAT_INPUT_MAG_VAR,
    &data ) == FALSE )
    Error("DatabaseReader",
        "cannot read DB_LAT_INPUT_MAG_VAR");

p = (lat_input_mag_typ *) data.value.user;

/*
 * take into account different coordinate systems
 * on sensors and program
 */

mag_fr_x = p->mag_fr_z;
mag_fr_y = p->mag_fr_x;
mag_fr_z = p->mag_fr_y;
mag_fc_x = p->mag_fc_z;
mag_fc_y = p->mag_fc_x;
mag_fc_z = p->mag_fc_y;
mag_fl_x = p->mag_fl_z;
mag_fl_y = p->mag_fl_x;
mag_fl_z = p->mag_fl_y;
mag_br_x = p->mag_br_z;
mag_br_y = p->mag_br_x;
mag_br_z = p->mag_br_y;
mag_bc_x = p->mag_bc_z;
mag_bc_y = p->mag_bc_x;
mag_bc_z = p->mag_bc_y;
mag_bl_x = p->mag_bl_z;
mag_bl_y = p->mag_bl_x;
mag_bl_z = p->mag_bl_y;

#endif // ifdef QNX

/Name | Type | Parameters | Description
---|------|------------|----------------
/read-lat-var | void | | Retrieve data from the variable DB_LAT_INPUT_SENSORS_VAR.

#ifdef QNX

db_data_typ data;
lat_input_sensors_typ* p;
if (clt_read(pclt,
    DB_LAT_INPUT_SENSORS_VAR,
    &data) == FALSE)
    Error("DatabaseReader",
        "cannot read DB_LAT_INPUT_SENSORS_VAR");
p = (lat_input_sensors_typ*) data.value.user;

// wheel-speed = p->wheel-speed;
steer-angle = p->steer-angle;
lat_yaw = p->lat_yaw;
lat_yacc = p->lat_yacc;
speed_cnt = p->speed_cnt;
#endif // ifdef QNX

/Name | Type | Parameters | Description
---|------|------------|----------------
/read-long-var | void | | Retrieve data from the variable DB_LONG_INPUT_VAR.

#ifdef QNX

float v1, v2, v3, v4, v5, v6, V1, V2;

db_data_typ data;
long_input_typ* p;
if (clt_read(pclt,
    DB_LONG_INPUT_VAR,
    &data) == FALSE)
    Error("DatabaseReader",
        "cannot read DB_LONG_INPUT_VAR");
p = (long-input-typ*) data.value.user;
brake-press = p->brake-press;
eng-rpm = p->eng_rpm;
eng_press = p->eng_press;
radar-range = p->radar_data.radar_range;
radar-rangerate = p->radar_data.radar_rangerate;
gear = p->gear;
gear-ratio = p->gear_ratio;
platoon_pos = p->platoon_pos;

/* estimates wheel speed */

v1 = (p->speed1 >= 39.5) ? 0 : p->speed1;
v2 = (p->speed2 >= 39.5) ? 0 : p->speed2;
v3 = (p->speed3 >= 39.5) ? 0 : p->speed3;
v4 = (p->speed4 >= 39.5) ? 0 : p->speed4;
v5 = (p->speed5 >= 39.5) ? 0 : p->speed5;
v6 = (p->speed6 >= 39.5) ? 0 : p->speed6;

V1 = (v1 + v2 + v3 + v4) / 4.0;
V2 = (v5 + v6) / 2.0;

if (V1 >= 7.0)
    wheel_speed = V1;
else if (V1 >= 4.0)
    wheel_speed = (V1 * (V1 - 4.0) + V2 * (7.0 - V1)) / 3.0;
else
    wheel_speed = V2;

#endif // #ifdef QNX

receive_trigger  void  Parameters  Description
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Function</td>
</tr>
</tbody>
</table>

#ifdef QNX
int dummy = 0;
int rep;
pid_t rec;

rec = Receive(interface_pid, &trigger, sizeof(trigger));
if (rec != interface_pid)
    Warning("DatabaseReader",
    "unexpected trigger received");

rep = Reply(interface_pid, &dummy, sizeof(dummy));
if (rep == -1)
    Warning("DatabaseReader",
    "trigger reply error");
#endif

#ifdef QNX
return (DB_TRIG_VAR(&trigger) == DB_LAT_INPUT_MAG_VAR);
#else
return 1;
#endif
```c
#ifdef QNX
    return (DB_TRIG_VAR(var) == DB_LAT_INPUT_SENSORS_VAR);
#else
    return 1;
#endif

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>Discrete state</td>
</tr>
<tr>
<td>read</td>
<td>Discrete state</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>idle</td>
<td>read_lat</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {test_lat_var()}

Action

'Code 'read_lat_var();'

Output Event

| Event   | read_lat |

Transition 1

<table>
<thead>
<tr>
<th>From/To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>rcd, get-triger</td>
<td>reaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {1};
Action
```c
Code receive-trigger();
```

Output Event

Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>idle</td>
<td>read_mag</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
```c
boolean {test_mag_var()}
```

Action
```c
Code {read_mag_var();}
```

Output Event
```c
Event read_mag
```

Transition 3

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>idle</td>
<td>read_long</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
```c
boolean {test_long_var()}
```

Action
```c
Code {read_long_var();}
```

Output Event
```c
Event read_long
```

Transition 4

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>idle</td>
<td>ignore</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
```c
boolean {!(test_mag_var() || test_lat_var() || test_long_var())}
```

Action

Output Event
```c

```
Include Files and other Preprocessor Directives

/*
 * Include files.
 */

#ifdef QNX

#ifndef DATABASE INCLUDED
#define DATABASE_INCLUDED

#ifdef __cplusplus
extern "C" {
#endif

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/name.h>
#include <sys/kernel.h>
#include <sys/proxy.h>
#include <signal.h>
#include <setjmp.h>
#include <local.h>
#include <sys_list.h>
#include "db comm.h"
#include "db-clt.h"
#include "track.h"
#include "veh_iols.h"
#include "clt_vars.h"

#ifdef __cplusplus
}
#endif

extern db-clt-typ* DBClient;
#endif
#endif // ifndef DATABASE_INCLUDED
#endif // ifndef QNX

#endif // ifndef QNX

Header Code

#include QNX

db_clt_typ* pclt;
trig_info_typ trigger;
pid_t interface_pid;
#endif

DatabaseReader();
~DatabaseReader();

C++ Code

#ifdef QNX
/
 */
 * Global database access pointer.
 */
db_clt_typ* DBClient = NULL;

/*
 * DatabaseReader Construction / Destruction
 */

DatabaseReader::DatabaseReader() {
    char hostname[MAXHOSTNAMELEN + 1];
    printf("initializing database reader... ");
    fflush(stdout);

    /* register the USR1 interrupt handler */
    DEM::registerInterruptHandler(SIGUSR1, UsrlInterruptHandler);

    /* attach the database reader name on the QNX system */
    if (qnx_name_attach(0, "diadem_database_reader") == -1)
        Error("DatabaseReader",
              "cannot attach diadem_database_reader\n");

    /* locate the database interface */
    for (;;) {
        interface_pid =
            qnx_name_locate(0, "diadem-database-interface", 0, NULL);
        if (interface_pid != -1)
            break;
        sleep(1);
    }

    /* login into the database */
    sprintf( hostname, "%lu", getnid() );
    if (( pclt = clt_login("diadem database_client", hostname,
                            DEFAULT_SERVICE,
                            COMM_QNX_XPORT)) == NULL)
        Error("DatabaseReader",
              "cannot login into the database");

    /* initialize the global DB client pointer */
    DBClient = pclt;
    printf("done.\n");
}

DatabaseReader::~DatabaseReader() {
    /* logout from the database */
    clt_logout(pclt);
}

#endif /* ifdef QNX */

DatabaseReader::DatabaseReader(){
}

DatabaseReader::~DatabaseReader(){
}
#endif /* ifdef QNX */
Class DatafileReader

This component reads data from a datafile in simulation mode.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataReader</td>
<td>Superclass of DatafileReader</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>FILE*</td>
<td>data file</td>
</tr>
<tr>
<td>filename</td>
<td>char*</td>
<td>Name of the file data is to be read from.</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(read mag-var void)</td>
<td></td>
<td></td>
<td>This function reads magnetometer fields</td>
</tr>
</tbody>
</table>

```c
if (fscanf(file, "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
    &mag_fl_x, &mag_fl_y, &mag_fl_z, &mag_fc_x, &mag_fc_y, &mag_fc_z, &mag_fr_x, &mag_fr_y, &mag_fr_z) != 9) {
    fscanf (file, "%lf %lf %lf %lf %lf %lf %lf %lf %lf",
        &mag_bl_x, &mag_bl_y, &mag_bl_z, &mag_bc_x, &mag_bc_y, &mag_bc_z, &mag_br_x, &mag_br_y, &mag_br_z) != 9)
    Error ("DatafileReader", "cannot read data file");
}
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>readlat_var</td>
<td>void</td>
<td></td>
<td>This function reads lateral variables.</td>
</tr>
</tbody>
</table>

```c
if (fscanf(file, "%lf %lf %lf %lf %d",
    &wheel-speed, &steer-angle,
```
&lat_yaw, &lat_yacc, &speed_cnt) != 5)
Error("DatafileReader", "cannot read data file");

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_long-var</td>
<td>void</td>
<td>void</td>
<td>This function reads longitudinal variables</td>
</tr>
</tbody>
</table>

if (fscanf(file, "%lf %lf %lf %lf %d \n",
&brake_press, &eng_rpm, &eng_press, &radar-range, &gear, &gear_ratio, &platoon_pos) != 8)
Error("DatafileReader", "cannot read data file");

Continuous States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer</td>
<td>Continuous state variable</td>
<td>1.</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computation</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
<td>timer' = 1</td>
<td></td>
</tr>
<tr>
<td>read_1</td>
<td>Discrete state</td>
<td>timer' = 1</td>
<td></td>
</tr>
<tr>
<td>read_2</td>
<td>Discrete state</td>
<td>timer' = 1</td>
<td></td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>read_1</td>
<td>read_mag</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

threshold {timer >= 0.002}

Action

'Reset Continuous State [timer[0]]

| Code | readmag_var(); |

Output Event

<table>
<thead>
<tr>
<th>Event</th>
<th>read_mag</th>
</tr>
</thead>
</table>
Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_1</td>
<td>read_2</td>
<td>read_lat</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
boolean {1}

Action
Code [readlat_var();]

Output Event
Event [read_lat]

Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read_2</td>
<td>idle</td>
<td>read_long</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
boolean {1}

Action
Code [read_long var();]

Output Event
Event [read_long]

Include Files and other Preprocessor Directives

Header Code

DatafileReader();
~DatafileReader();

C++ Code

DatafileReader::DatafileReader() {
        /* open data file */
        if ( (file = fopen("/home/porsche2/varaiya/pant/car_run/field_data/xxx.dat", "r")
                Error ("DatafileReader",
                        "datafile opening error") ;
        }
}

DatafileReader::~DatafileReader() {
        fclose(file);
}
Class Enginesensor

This component provides engine speed, engine manifold pressure.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Superclass of EngineSensor</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine - speed</td>
<td>double</td>
<td>engine speed [rpm]</td>
</tr>
<tr>
<td>engine_manifold_pressure</td>
<td>double</td>
<td>Engine Manifold Pressure [psi]</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
</tr>
<tr>
<td>busy</td>
<td>Discrete state</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_long</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

Guard
boolean {1}

Action
- reset Variable engine_speed | reader->eng_rpm
- reset Variable engine_manifold_pressure | reader->eng_press

Output Event
- 

Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>new_data</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
boolean {1}

Action
Output Event

Include Files and other Preprocessor Directives

Header Code

Enginesensor();

C++ Code

EngineSensor::EngineSensor()
{
}

Class EngineSpeedFilter

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of EngineSpeedFilter</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>we_old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>we_flt</td>
<td>double</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/engine-sensor</td>
<td>EngineSensor</td>
<td>Link</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/filtered_engine_speed</td>
<td>double</td>
<td></td>
<td>Function</td>
</tr>
</tbody>
</table>

double we = engine_speed();

if(timer() < 0.07) {
    we_flt = we;
    we_old = we;
} else {
    if (timer() <= 1.5
        || gear_change_timer() < 0.8
        || gear_change_timer() > 1.5) {
        if (we - we_old > 5.0)
            we = we_old + 5.0;
        else if (we - we_old < -5.0)
            we = we_old - 5.0;
else {
  if (we - we_old > 20.0)
    we = we_old + 20.0;
  else if (we - we_old < -20.0)
    we = we_old - 20.0;
}

we_old = we;
we_flt = we*0.3 + we_flt*0.7;
}
return we_flt;

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>Link Type</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>/engine_speed</td>
<td>double</td>
<td>engine_sensor</td>
<td>EngineSensor*</td>
<td>engine_speed</td>
</tr>
</tbody>
</table>

### Continuous States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
<tr>
<td>gear_change_timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
</tbody>
</table>

### Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
<td></td>
<td>timer' = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>gear change timer' = 1</td>
</tr>
</tbody>
</table>

### Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>idle</td>
<td>trj_begin</td>
<td>response</td>
</tr>
</tbody>
</table>

**Guard**

boolean {1}

**Action**

| Reset Continuous State | timer [0] |

### Output Event

---

### Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>idle</td>
<td>gear_change</td>
<td>response</td>
</tr>
</tbody>
</table>

---
Guard

boolean {1}

Action

Reset Continuous State

Output Event

Include Files and other Preprocessor Directives

Header Code

EngineSpeedFilter();

C++ Code

EngineSpeedFilter::EngineSpeedFilter()
{
}

Class LateralController

This component computes the lateral control action which steering angle in degrees.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of LateralController</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_f</td>
<td>double</td>
<td>Front Wheel cornering stiffness</td>
</tr>
<tr>
<td>C_r</td>
<td>double</td>
<td>Rear wheel cornering stiffness</td>
</tr>
<tr>
<td>( \mu )</td>
<td>double</td>
<td>Road Adhesion Factor. Value guessed</td>
</tr>
<tr>
<td>l_f</td>
<td>double</td>
<td>Distance from center of gravity to front wheels.</td>
</tr>
<tr>
<td>v</td>
<td>double</td>
<td>Velocity of car</td>
</tr>
<tr>
<td>l_r</td>
<td>double</td>
<td>Distance from center of gravity to rear wheels</td>
</tr>
<tr>
<td>( \delta ) f</td>
<td>double</td>
<td>Front wheel steering angle</td>
</tr>
<tr>
<td>( \phi )</td>
<td>double</td>
<td>side slip angle</td>
</tr>
<tr>
<td>r</td>
<td>double</td>
<td>Yaw rate</td>
</tr>
<tr>
<td>( \tilde{J} )</td>
<td>double</td>
<td>Normalised angular momentum of vehicle</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>n_tilda</td>
<td>double</td>
<td>Normalised mass</td>
</tr>
<tr>
<td>l_s</td>
<td>double</td>
<td>Distance from center of gravity to front sensors</td>
</tr>
<tr>
<td>q_0</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>q_1</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>q_2</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>p_1</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>p_2</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>delta_syn</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>zeta</td>
<td>double</td>
<td>Damping Coefficient for controller</td>
</tr>
<tr>
<td>omega_n</td>
<td>double</td>
<td>natural frequency of the controller</td>
</tr>
<tr>
<td>period</td>
<td>double</td>
<td>Rate at which control computation is taking place.</td>
</tr>
<tr>
<td>delta_0</td>
<td>double</td>
<td>Parameter which is estimated by using adaptive scheme, to take into account model inaccuracies.</td>
</tr>
<tr>
<td>k_s</td>
<td>double</td>
<td>Gain for lateral velocity observer</td>
</tr>
<tr>
<td>d_shat_ddot</td>
<td>double</td>
<td>Estimated derivative of lateral velocity.</td>
</tr>
<tr>
<td>d_shat_dot</td>
<td>double</td>
<td>Estimated lateral velocity.</td>
</tr>
<tr>
<td>m</td>
<td>double</td>
<td>normalised mass</td>
</tr>
<tr>
<td>J</td>
<td>double</td>
<td>normalised angular momentum</td>
</tr>
<tr>
<td>d_s_estim</td>
<td>double</td>
<td>Estimated lateral displacement, using the simulation of the model equations.</td>
</tr>
<tr>
<td>d_s</td>
<td>double</td>
<td>Lateral displacement.</td>
</tr>
<tr>
<td>d_sdot</td>
<td>double</td>
<td>Lateral velocity.</td>
</tr>
<tr>
<td>speed_threshold</td>
<td>double</td>
<td>Speed below which the controller stops computing control action.</td>
</tr>
<tr>
<td>k_a</td>
<td>double</td>
<td>Gain for parameter adaptation.</td>
</tr>
<tr>
<td>close_loop</td>
<td>int</td>
<td>Variable to indicate whether reglayer is to be run in close loop mode or open loop mode.</td>
</tr>
<tr>
<td>d_sdot_threshold</td>
<td>double</td>
<td>Threshold for lateral velocity.</td>
</tr>
<tr>
<td>control_type</td>
<td>int</td>
<td>Variable to indicate which of the control action is to be used.</td>
</tr>
<tr>
<td>control_to</td>
<td>double</td>
<td>The position to which the car is to be controlled.</td>
</tr>
</tbody>
</table>
Variable to indicate whether the earth estimation is to be done in lateral sensor processor.

Variable

0 - use both left-center and center-right l.s.p 1 - use only the left-center l.s.p 2 - use only the center-right l.s.p.

Variable

Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>Link to car sensor for speed of the car.</td>
</tr>
<tr>
<td>cr_lsp</td>
<td>LateralSensorProcessor*</td>
<td>Link to CR lateral sensor processor.</td>
</tr>
<tr>
<td>lc_lsp</td>
<td>LateralSensorProcessor*</td>
<td>Link to LC lateral sensor processor.</td>
</tr>
</tbody>
</table>

Functions

double lambda = 0;
lambda = (zeta+sqrt(zeta*zeta-1))*omega_n;
delta-0 = delta_0+(-k_s*speed()*(d_shat_ddot+
  lambda_d_s))*period;

// trying to control to a non center position
double d_s_c = d_s - control_to;

v = speed();

d_shat_ddot = -v*(delta_syn + delta 0) + k_s +
  (d_sdot - d_shat_dot);

d_shat_dot = d_shat_dot + d_shat_ddot*period;

d_shat_dot = Saturate(d_shat_dot,d_sdt_threshold);

delta-syn = 1/v*(2*zeta*omega_n*d_shat_dot +
  omega_n*omega_n*d_s c)-delta_0;

double x1, x2, step;
v = speed();

J_tilda = J/mu;
m_tilda = m/mu;

\[ q_0 = \frac{C_f}{m_tilda*v} + \frac{C_f*1_f*1_e}{(C_tilda*v)}; \]

\[ q_1 = \frac{C_f*(1_r + 1_e)*(1_r + 1_f)/(m_tilda*J_tilda*v*v)}; \]

\[ q_2 = \frac{C_f*1_r * 1_f}{(m_tilda*J_tilda*v)}; \]

\[ p_1 = \frac{(C_r + C_f)/m_tilda*v}{(C_r*1_r + 1_f + C_f*1_f*1_f)/(J_tilda*v)}; \]

\[ p_2 = \frac{(C_r*1_r - C_f*1_f)/J_tilda + (C_r*1_r + 1_f*1_f + 2*C_r*1_r*1_f)/(m_tilda*J_tilda*v*v)}; \]

// Integrate the state space equation
step = period/100;
x1 = x2 = 0.0;
for(int i = 0; i < 100; i++){
  x1 = x1 + x2*step;
  x2 = x2 + ((-q_2/q_0)*x1 + (-q_1/q_0)*x2 + delta_syn)*step;
}

// Compute delta-f
delta_f = 0.7 * delta_f +
  1.3 * ((p_2/q_0-q_2/(q_0*q_0))*x1 +
         (p_1/q_0-q_1/(q_0*q_0))*x2 +
         1/q_0*delta_syn);

// Convert to degrees
control_angle = delta_f*180.0/PI;

// Convert to handwheel angle
control_angle = control_angle*17;

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat Disp Estimator</td>
<td>void</td>
<td></td>
<td>Function estimates ds when sensor readings are not reliable.</td>
</tr>
</tbody>
</table>

double r_der = 0, q_der = 0, theta = 0;

J_tilda = J/mu;
m_tilda = m/mu;

v = speed();

r_der = ((C_r*1_r - C_f*1_f)/J_tilda)*q - ((C_r*1_r*1_r + C_f*1_f*1_f)/(J_tilda*v))*r +
        (C_f*1_f/J_tilda)*delta_f;

r = r + r_der*period;

q_der = -r - ((C_f + C_r)/(m_tilda*v))*q +
        ((C_r*1_r - C_f*1_f)/(m_tilda*v*v))*r +
        (C_f/(m_tilda*v))*delta_f;

q = q + q_der*period;

theta = theta + r*period;
\[
d_s_{-\text{estim}} = -v(q + \theta) - l_s r;
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg_of_lat_dis</td>
<td>void</td>
<td></td>
<td>Calculates the average lateral displacement from all lsp's. A constant is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>added to control the car between center right magnetometers.</td>
</tr>
</tbody>
</table>

\[
d\text{valid-avg} = l_c \text{valid}_y() + cr \text{valid}_y();
\]

\[
\text{if (valid_avg }!= \text{ 0)}{ \begin{align*}
\text{d}_s &= 0.7 \text{ * (c}_r \text{d}_s) + 0.3 \text{ * (l}_c \text{d}_s) \text{ / valid-avg; }
\text{d}_s\text{dot} &= 0.7 \text{ * d}_s\text{dot} + 0.3 \text{ * (c}_r \text{d}_s\text{dot}) \text{ / valid-avg; }
\text{d}_s\text{dot} &= \text{Saturate(d}_s\text{dot, d}_s\text{dot_threshold);}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>control_comput_2</td>
<td>void</td>
<td></td>
<td>Function</td>
</tr>
</tbody>
</table>

\[
\text{if (d}_s < -0.1)\text{ delta}_f = 15;
\text{else if (d}_s > 0.1)\text{ delta}_f = -15;
\text{else}
\text{ delta}_f = 0;
\text{control-angle} = \text{delta}_f;
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write_command</td>
<td>void</td>
<td></td>
<td>Function</td>
</tr>
</tbody>
</table>

```c
#ifdef QNX

\text{if (close-loop)}{ \begin{align*}
\text{db_data_typ data;} \\
\text{lat_output_typ* p;} \\
\text{/* retrieve the current info from the database */} \\
\text{if (clt_read(DBClient,} \\
\text{DB_LAT_OUTPUT_VAR,} \\
\text{DB_LAT_OUTPUT_TYPE,} \\
\text{&data)} == \text{FALSE} \} \\
\text{Error("LateralController",} \\
\text{"cannot read DB_LAT_OUTPUT_VAR");} \\
\text{p = (lat_output_typ*) data.value.user;} \\
\text{/* modify the steer angle */} \\
\text{p->steer_ctrl = control-angle;}\end{align*}
```
/* update the database variable */
if (clt_update(DBClient,
    DB_LAT_OUTPUT_VAR,
    DB_LAT_OUTPUT_TYPE,
    sizeof(lat_output_type),
    (void*)data.value.user) == FALSE)
    Error("LateralController",
        "cannot update DB_LAT_OUTPUT_VAR");
#endif

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>Link Type</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>lc_d_s</td>
<td>double</td>
<td>lc_lsp</td>
<td>LateralSensorProcessor*</td>
<td>y_act</td>
</tr>
<tr>
<td>cr_d_s</td>
<td>double</td>
<td>cr_lsp</td>
<td>LateralSensorProcessor*</td>
<td>y_act</td>
</tr>
<tr>
<td>lc_d_sdot</td>
<td>double</td>
<td>lc_lsp</td>
<td>LateralSensorProcessor*</td>
<td>y_der</td>
</tr>
<tr>
<td>cr_d_sdot</td>
<td>double</td>
<td>cr_lsp</td>
<td>LateralSensorProcessor*</td>
<td>y_der</td>
</tr>
<tr>
<td>lc_valid_y</td>
<td>double</td>
<td>lc_lsp</td>
<td>LateralSensorProcessor*</td>
<td>valid_y_output_avg</td>
</tr>
<tr>
<td>cr_valid_y</td>
<td>double</td>
<td>cr_lsp</td>
<td>LateralSensorProcessor*</td>
<td>valid_y_output_avg</td>
</tr>
<tr>
<td>speed</td>
<td>double</td>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>car_speed</td>
</tr>
</tbody>
</table>

Continuous States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>!control-timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
<td></td>
<td>control_timer' = 1</td>
</tr>
<tr>
<td>busy</td>
<td>Discrete state</td>
<td></td>
<td>control_timer' = 1</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>estimate</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean (1)
Action

```
if (speed() > speed_threshold) {
  avg_of_lat_disp();
  param_adapt();
  observer();
  if (control-type == 1)
    control_comput_1();
  else
    control_comput_2();

  if(fabs(d_s) >= 0.02)
    write_command();
}
```

Output Event

```
Event lateral_command
```

Transition 1

```
<table>
<thead>
<tr>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>control</td>
<td>proaction</td>
</tr>
</tbody>
</table>
```

Guard

threshold {control-timer >= period}

Action

```
Reset Continuous State control_timer
control_timer() - period
```

Output Event

```
Event estimate
```

Include Files and other Preprocessor Directives

```
/*
 * Include files.
 */

#ifdef QNX

#ifndef DATABASE INCLUDED
#define DATABASE INCLUDED
#endif

#ifdef __cplusplus
extern "C" {
#endif

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/name.h>
#include <sys/kernel.h>
#include <sys/proxy.h>
#include <signal.h>
#include <setjmp.h>
```

```
#include <local.h>
#include <sys_list.h>

#include "db_comm.h".
#include "db-clt.h"
#include "track.h"
#include "veh_iols.h"
#include "clt_vars.h"

#ifndefcplusplus

extern db_clt_typ* DBClient;
#endif

#endif // ifndef DATABASE_INCLUDED

#endif // ifndef QNX

Header Code

LateralController();

C++ Code

LateralController::LateralController()
{

    FILE* file;

    /* Initialize lateral controller parameters
    * at some default values. [You must 'omit' their
    * initialization from the server architecture
    * diagram, otherwise, they will be overwritten.]
    */

    omega_n = 0.05;
    period = 0.02;
    k_s = 0.05;
    k_a = 0.05;
    d_sdot_threshold = 0.5;
    close_loop = 1;
    control_type = 1;
    control_to = 0.15;
    d_s = control-to;
    earth_estimate_flag = 0;
    zeta = 1.0;
    lsp_use = 0;
    d_threshold = 0.1;

    /* Attempt to read some other user-specific values for
    * the parameters from the external file 'lateral.ini'
    * The format of this file is simple:
    * parameter-name parameter-value
    * parameter-name parameter-value
    * The initialization ends when the reading from
    * the file fails. No checks about the parameters
    * values are done. If the initialization file is
    * missing or unknown parameter names are used,
    * the user is signalled. */

}
if ( (file = fopen("lateral.ini", "rrr")) != NULL) {
    char name[661];
    float value;
    while (fscanf(file, "%s %f", &name, &value) == 2) {
        if (!strcmp(name, "omega nu"))
            omega_n = value;
        else if (!strcmp(name, "period"))
            period = value;
        else if (!strcmp(name, "k s"))
            k_s = value;
        else if (!strcmp(name, "k a"))
            k_a = value;
        else if (!strcmp(name, "d sdot threshold"))
            d_sdot_threshold = value;
        else if (!strcmp(name, "close loop"))
            close_loop = (int) value;
        else if (!strcmp(name, "control type"))
            control_type = (int) value;
        else if (!strcmp(name, "control to"))
            d = control_to = value;
        else if (!strcmp(name, "earth estimate flag"))
            earth_estimate_flag = (int) value;
        else if (!strcmp(name, "zeta"))
            zeta = value;
        else if (!strcmp(name, "lsp use"))
            lsp_use = (int) value;
        else if (!strcmp(name, "d threshold"))
            d_threshold = value;
        else
            fprintf(stderr, "LateralController: invalid parameter '%s'!\n", name);
    }
}
else
    fprintf(stderr, "LateralController: missing initialization file!\n");

Class LateralSensor

This component provides the model of the lateral sensor. It provides sinusoidal inputs for the x, y and z magnetic fields.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Superclass of LateralSensor</td>
</tr>
</tbody>
</table>

Variables
### Name | Type | Description
--- | --- | ---
\(x\)-measurement | double | Field measurement in the longitudinal direction.
\(y\)-measurement | double | Field measurement in the lateral direction.
\(z\)-measurement | double | Field measurement in the vertical direction.
position | int | Variable indicating the position of the magnetic sensor. Left Center or Right.

### Functions

| Name | Parameters | Type | Description |
--- | --- | --- | ---
read_front-mag | | void | This function reads the magnet sensor data from DataReader.

```
switch (position) {
  case LEFT:
    x_measurement = reader->mag_fl_x;
    y_measurement = reader->mag_fl_y;
    z_measurement = reader->mag_fl_z;
    break;
  case CENTER:
    x_measurement = reader->mag_fc_x;
    y_measurement = reader->mag_fc_y;
    z_measurement = reader->mag_fc_z;
    break;
  case RIGHT:
    x_measurement = reader->mag_fr_x;
    y_measurement = reader->mag_fr_y;
    z_measurement = reader->mag_fr_z;
    break;
}
```

### Discrete States

| Name | Description |
--- | ---
idle | Discrete state
busy | Discrete state

### Transition 0

| From | To | Event | Type | Description |
--- | --- | --- | --- | ---
idle | busy | read_mag | response | }

### Guard

```
function {if true(1);}
```

### Action
Include Files and other Preprocessor Directives

#define LEFT -1
#define CENTER 0
#define RIGHT 1

Header Code

LateralSensor();

C++ Code

LateralSensor::LateralSensor()
{
}

Class LateralSensorProcessor

This component implements the lateral sensor processor algorithm documented in the paper "The Design of a Look-Down Feedback Adaptive Controller for the Lateral Control of Front-Wheel-Steering Autonomous Highway Vehicles" by Seibum B. Choi, Proceedings of the 1997 American Control Conference. It outputs the lateral position (dy) of the vehicle with respect to the reference trajectory of the magnetic markers. It also outputs the longitudinal position (dx) of the vehicle between the magnetic markers.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of LateralSensorProcessor</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>magnetometer-spacing</td>
<td>double</td>
<td>The distance between the two magnetometers on the car.</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>x_earth</td>
<td>double</td>
<td>Earth's magnetic field estimate in the car's longitudinal direction.</td>
</tr>
<tr>
<td>y_earth</td>
<td>double</td>
<td>Earth's magnetic field estimate in the car's lateral direction.</td>
</tr>
<tr>
<td>z_earth</td>
<td>double</td>
<td>Earth's magnetic field estimate in the vertical direction. The estimate measured a priori in the general location of the experiment.</td>
</tr>
<tr>
<td>input_sample_period</td>
<td>double</td>
<td>Time between the input samples.</td>
</tr>
<tr>
<td>n_magnets</td>
<td>int</td>
<td>The number of magnets over which the moving average is taken.</td>
</tr>
<tr>
<td>magnet_spacing</td>
<td>double</td>
<td>Distance between successive magnets [m].</td>
</tr>
<tr>
<td>n_samples</td>
<td>int</td>
<td>Number of samples used in the estimation of the earth field.</td>
</tr>
<tr>
<td>dx</td>
<td>double</td>
<td>Longitudinal distance to the next magnet.</td>
</tr>
<tr>
<td>dy</td>
<td>double</td>
<td>Lateral displacement from the line of magnets.</td>
</tr>
<tr>
<td>left_x_sample_array</td>
<td>double*</td>
<td>Array storing the history of left magnetometer x readings.</td>
</tr>
<tr>
<td>left_y_sample_array</td>
<td>double*</td>
<td>Array storing the history of left magnetometer y readings.</td>
</tr>
<tr>
<td>left_z_sample_array</td>
<td>double*</td>
<td>Array storing the history of left magnetometer z readings.</td>
</tr>
<tr>
<td>right_x_sample_array</td>
<td>double*</td>
<td>Array storing the history of right magnetometer x readings.</td>
</tr>
<tr>
<td>right_y_sample_array</td>
<td>double*</td>
<td>Array storing the history of right magnetometer y readings.</td>
</tr>
<tr>
<td>right_z_sample_array</td>
<td>double*</td>
<td>Array storing the history of right magnetometer z readings.</td>
</tr>
<tr>
<td>array_position</td>
<td>int</td>
<td>Array position tells the current position in the sample arrays.</td>
</tr>
<tr>
<td>cx1</td>
<td>double</td>
<td>x1 reading corrected for earth field.</td>
</tr>
<tr>
<td>cy1</td>
<td>double</td>
<td>y1 reading corrected for earth field.</td>
</tr>
<tr>
<td>cz1</td>
<td>double</td>
<td>z1 reading corrected for earth field.</td>
</tr>
<tr>
<td>cx2</td>
<td>double</td>
<td>x2 reading corrected for earth field.</td>
</tr>
<tr>
<td>cy2</td>
<td>double</td>
<td>y2 reading corrected for earth field.</td>
</tr>
<tr>
<td>cz2</td>
<td>double</td>
<td>z2 reading corrected for earth field.</td>
</tr>
<tr>
<td>magnet_sample_ratio</td>
<td>double</td>
<td>This ratio is used to find the n-samples so that the value will be using n_magnet=2 and input_sample_period 2ms.</td>
</tr>
<tr>
<td>valid_y_output</td>
<td>int</td>
<td>Variable which indicates whether the magnetometer readings in valid.</td>
</tr>
<tr>
<td>valid_x_output</td>
<td>int</td>
<td>Variable which indicates whether the magnetometer readings in valid.</td>
</tr>
</tbody>
</table>
### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>field_threshold</td>
<td>double</td>
<td>This variable stops the calculation of dx and/or dy if the value of field drops below the specified value.</td>
</tr>
<tr>
<td>position</td>
<td>int</td>
<td>Variable to indicate which of the magnetometers are used for processor. Types are left-center (LC), right-center (RC), and left.</td>
</tr>
<tr>
<td>y_act</td>
<td>double</td>
<td>The actual position of the centerline of the car with respect to the magnets.</td>
</tr>
<tr>
<td>y_der</td>
<td>double</td>
<td>Lateral velocity [m/s]</td>
</tr>
<tr>
<td>y_act_prev</td>
<td>double</td>
<td>Value of y_act from previous calculation.</td>
</tr>
<tr>
<td>speed_threshold</td>
<td>double</td>
<td>The longitudinal speed below which earth estimation is stopped.</td>
</tr>
<tr>
<td>valid_y_output_array</td>
<td>int*</td>
<td>Variable</td>
</tr>
<tr>
<td>valid_y_output_position</td>
<td>int</td>
<td>Variable</td>
</tr>
<tr>
<td>valid_y_output_avg</td>
<td>double</td>
<td>Variable</td>
</tr>
</tbody>
</table>

### Links

<table>
<thead>
<tr>
<th>[Name]</th>
<th>[Type]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left_magnetometer</td>
<td>LateralSensor*</td>
<td>Link to left lateral sensor.</td>
</tr>
<tr>
<td>right_magnetometer</td>
<td>LateralSensor*</td>
<td>Link to right lateral sensor.</td>
</tr>
<tr>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>Link to car sensor for access to car speed</td>
</tr>
<tr>
<td>lateral_controller</td>
<td>LateralController*</td>
<td>Link to lateral controller for access to earth estimate flag.</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:long_position</td>
<td></td>
<td>void</td>
<td>This function calculates the longitudinal position of the centerline of the magnetometers with respect to the next magnet.</td>
</tr>
</tbody>
</table>

```c
if (fabs(cx1*cy2 - cx2*cy1) < d_threshold; ||
    fabs(cx1) < field_threshold ||
    fabs(cx2) < field_threshold ||
    fabs(cy1) < field-threshold ||
    fabs(cy2) < field-threshold ||
    cyl*cy2 > 0) {

    valid_x-output = 0;

}

else {

    valid_x-output = 1;

    dx = magnetometer-spacing
        * (cx1 + cx2 / (cx1 + cy2 - cx2 * cy1));
```

double etal, eta2, alpha1, alpha2, beta1, beta2;

if (position == LC && lsp_use() == 2) {
    position = cr && lsp_use() == 1);

    valid_y_output = 0;
}
else
{
    valid_y_output = 0;
}
else {

valid_y_output = 1;

etal = fabs(cx1)*cx1 + cy1 * cy1) / sqrt(cx1 * cx1 + cy1 * cy1);
eta2 = fabs(cx2)*cx2 + cy2 * cy2);

alpha1 = atan((3 * eta1 + sqrt(9 * eta1 * eta1 + 8)) / 4);
alpha2 = atan((3 * eta2 + sqrt(9 * eta2 * eta2 + 8)) / 4);

beta1 = acos(-cos(alpha1) * cy1 / sqrt(cx1 * cx1 + cy1 * cy1));
beta2 = acos(cos(alpha1) * cy2 / sqrt(cx2 * cy2 + cy2 * cy2));

dy = 0.0 * dy + 1.0 * (0.5 * magnetometer_spacing * 
(sin(beta2-betal) / sin(beta2 + betal)));

// dy = 0.7 * dy + 0.3 * (0.5 * magnetometer_spacing * 
// (-tan(betal) + tan(beta2)) / 
// (tan(betal) + tan(beta2));

} // Average the valid_y_output values

valid_y_output_array[valid_y_output_position] = 
valid_y_output;
valid_y_output_position = 
(valid_y_output_position + 1) % 20;

valid_y_output_avg = 0.0;
for(int i = 0; i < 20; i++)
valid_y_output_avg += valid_y_output_array[i];
valid_y_output_avg /= 20.0;
This function computes earth magnetic field in vehicle longitudinal direction and lateral direction by using moving average scheme.

double sum_x = 0.0, sum_y1 = 0.0, sum_y2 = 0.0;
if(earth_estimate_flag() == 0)
  return;
if (speed() < speed_threshold)
  return;

// magnet_sampling_ratio is assumed to be constant
// and found by using 2 magnets and 2ms as sample period
// n_magnets = magnet_sample_ratio * input_sample_period;

n_samples = (int) (n_magnets * magnet_spacing
  / speed() / input_sample_period);

for(int i = 0; i <= n_samples; i++)
{
  int index = array_position - i;
  if (index < 0)
    index = SAMPLE_ARRAY_SIZE + index;

  sum_x += left_x_sample_array[index];
  sum_y1 += left_y_sample_array[index];
  sum_y2 += right_y_sample_array[index];
}

x_earth = sum_x / n_samples;
y_earth = (sum_y1 + sum_y2) / (2.0 * n_samples);

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_earth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y_earth</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_correction</td>
<td>void</td>
<td></td>
<td>Updates the corrected reading variables (c_{xy})[12]</td>
</tr>
</tbody>
</table>

\[
cxl = \text{left}_x() - x_{\text{earth}}; \\
cy1 = \text{left}_y() - y_{\text{earth}}; \\
cz1 = \text{left}_z() - z_{\text{earth}}; \\
cx2 = \text{right}_x() - x_{\text{earth}}; \\
cy2 = \text{right}_y() - y_{\text{earth}}; \\
cz2 = \text{right}_z() - z_{\text{earth}};
\]

if (valid_y_output) {
  switch (position) {
    case LC:
      y_act = -(magnetometer_spacing/2.0 - dy);
      break;
    case CR:
      y_act = (magnetometer_spacing/2.0 + dy);
      break;
  }
}
case LR:
    \[ y_{-}\text{act} = dy; \]
    break;
\]

\[ y_{-}\text{der} = (y_{\text{act}}-y_{\text{act prev}})/\text{period}(); \]
\[ y_{\text{act prev}} = y_{\text{act}}; \]

\]

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>[Link Type]</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>left_x</td>
<td>double</td>
<td>left_magnetometer</td>
<td>LateralSensor*</td>
<td>x_measurement</td>
</tr>
<tr>
<td>left_y</td>
<td>double</td>
<td>left_magnetometer</td>
<td>LateralSensor*</td>
<td>y_measurement</td>
</tr>
<tr>
<td>left_z</td>
<td>double</td>
<td>left_magnetometer</td>
<td>LateralSensor*</td>
<td>z_measurement</td>
</tr>
<tr>
<td>right_x</td>
<td>double</td>
<td>right_magnetometer</td>
<td>LateralSensor*</td>
<td>x_measurement</td>
</tr>
<tr>
<td>right_y</td>
<td>double</td>
<td>right_magnetometer</td>
<td>LateralSensor*</td>
<td>y_measurement</td>
</tr>
<tr>
<td>right_z</td>
<td>double</td>
<td>right_magnetometer</td>
<td>LateralSensor*</td>
<td>z_measurement</td>
</tr>
<tr>
<td>speed</td>
<td>double</td>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>car_speed</td>
</tr>
<tr>
<td>earth_estimate_flag</td>
<td>int</td>
<td>lateral_controller</td>
<td>LateralController*</td>
<td>earth_estimate_flag</td>
</tr>
<tr>
<td>period</td>
<td>double</td>
<td>lateral_controller</td>
<td>LateralController*</td>
<td>period</td>
</tr>
<tr>
<td>lsp_use</td>
<td>int</td>
<td>lateral_controller</td>
<td>LateralController*</td>
<td>lsp_use</td>
</tr>
<tr>
<td>d_threshold</td>
<td>double</td>
<td>lateral_controller</td>
<td>LateralController*</td>
<td>d_threshold</td>
</tr>
</tbody>
</table>

**Discrete States**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>Discrete state</td>
</tr>
</tbody>
</table>

**Transition 0**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>go</td>
<td>new_data</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean :1}

**Action**
Code:

```c
left_x_sample_array[array_position] = left_x();
left_y_sample_array[array_position] = left_y();
left_z_sample_array[array_position] = left_z();
right_x_sample_array[array_position] = right_x();
right_y_sample_array[array_position] = right_y();
right_z_sample_array[array_position] = right_z();
array_position = (array_position + 1) % SAMPLE_ARRAY_SIZE;
```

Output Event

---

Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>go</td>
<td>go</td>
<td>estimate</td>
<td>response</td>
<td>This transition is fired when n_magnets number of magnets will be crossed at the vehicle's current speed.</td>
</tr>
</tbody>
</table>

Guard

function {if_true(1);} |

Action

```c
earth_estimate();
apply_correction();
long_position();
lateral_position();
lateral_disp();
```

Output Event

---

Include Files and other Preprocessor Directives

```c
#include <math.h>
/*
 * We are assuming that the minimum longitudinal speed is
 * 1.2 m/s
 */
#define SAMPLE_ARRAY_SIZE 1000
```

Header Code

LateralSensorProcessor();

C++ Code

```c
LateralSensorProcessor::LateralSensorProcessor()
{ int i;
  left_x_sample_array = new double[SAMPLE_ARRAY_SIZE];
  left_y_sample_array = new double[SAMPLE_ARRAY_SIZE];
  left_z_sample_array = new double[SAMPLE_ARRAY_SIZE];
  right_x_sample_array = new double[SAMPLE_ARRAY_SIZE];
  right_y_sample_array = new double[SAMPLE_ARRAY_SIZE];
  right_z_sample_array = new double[SAMPLE_ARRAY_SIZE];
  for(i = 0; i < SAMPLE_ARRAY_SIZE; i++) {
```
```java
left_x_sample_array[i] = 0;
left_y_sample_array[i] = 0;
left_z_sample_array[i] = 0;
right_x_sample_array[i] = 0;
right_y_sample_array[i] = 0;
right_z_sample_array[i] = 0;
```

```java
valid_y_output_array = new int[20];
for(i = 0; i < 20; i++)
    valid_y_output_array[i] = 0;
}
```

### Class LowerController

#### Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Component is superclass of LowerController</td>
</tr>
</tbody>
</table>

#### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>throttle_angle</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>brake_pressure</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>lambda_des</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>sampling_time</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>tmain</td>
<td>double</td>
<td>Temperature in the manifold</td>
</tr>
<tr>
<td>i_eng</td>
<td>double</td>
<td>Engine inertia</td>
</tr>
<tr>
<td>i_whl</td>
<td>double</td>
<td>Wheel inertia</td>
</tr>
<tr>
<td>mass</td>
<td>double</td>
<td>Vehicle mass</td>
</tr>
<tr>
<td>hr</td>
<td>double</td>
<td>Tire radius</td>
</tr>
<tr>
<td>cd_a</td>
<td>double</td>
<td>Air drag coefficient</td>
</tr>
<tr>
<td>if_f</td>
<td>double</td>
<td>Rolling moment of the wheel (resistance due to bearings)</td>
</tr>
<tr>
<td>rd</td>
<td>double</td>
<td>Drive ratio</td>
</tr>
<tr>
<td>palm</td>
<td>double</td>
<td>Atmospheric pressure in kilo pascals</td>
</tr>
<tr>
<td>tmtopm</td>
<td>double</td>
<td>Ideal gas law constant (converts temperature to pressure)</td>
</tr>
<tr>
<td>idling_speed</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>holding_torque</td>
<td>double</td>
<td>Torque below idling speed to keep car stationary (in psi)</td>
</tr>
<tr>
<td>accel_to_psi_factor</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>tnet_des</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>tnet_min</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>alp_des</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>tnet_a</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>tnet_b</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>mades_old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>macd_old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>s2_old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>alp_des_old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>brake_sw_old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>r_star</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>r_starh</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>j_eff_in</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>j_beta</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>j_f-b</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>jpb_des</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>stopped_brake_press</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>jpb_des-old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>alp_gain</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>maneuver_id</td>
<td>int</td>
<td>Variable</td>
</tr>
<tr>
<td>jpb_temp</td>
<td>double</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Links
double desired torque =
   beta * synthetic_accel()
 + c_A * r_starh * wheel_speed() * wheel_speed()
 + r_starh * f_f;

if (wheel_speed() <= idling_speed) {
   double temp = min (300 * (idling-speed - wheel_speed())/idling_speed, holding-torque) - beta*temp/(accel_to.psi_factor*1.2);
}

return desired-torque;

double alp desd;
double lambda2 = 20.0;
double dt = sampling_time;
double time-filter = timer();
double tg = gear-change-timer();

double ma = pm_flt(t)/ (tmtopm*tman);
double pm des = eng_tbl (we_flt(),tnet_a,1,3);
double mades = pm des / (tmtopm*tman);
mades = mades*0.2 + makes old*0.8;
double madesd = (mades - makes old) / dt;
double pmdesd = madesd * (tmtopm*tmon);
double ma_old = ma;

mades old = mades;
double maod = eng_tbl (we_flt(),pm_flt(1,3,2));

if (time filter < 0.03) {
   maod old = maod;
   alp gain = 0.0;
} else ;
maod = maod*0.2 + maod_old*0.8;
maod_old = maod;
alp_gain = sampling-time + 1.6*(1 - alp_gain);

double s2 = ma - mades;
if (time_filter < 0.03)
    s2_old = s2;
s2_old = s2;  // is this correct????!!!
madesd = min (madesd, 0.7);
madesd = max (madesd, -0.7);
double buf = maod + madesd*2.5 - lambda2*s2;
alp_des = alp_tbl(pm_des, buf);
if (time_filter < 1.6 || tg>1.6) {
alp-des = alp_gain*alp_des + (1.0 - alp_gain)*alp_des_old;
alp_desd = (alp-des - alp_des_old)/dt;
alp_des_old = alp_des;
} else {
    double templ = 0.6*sin(2.0*PI/2.4*tg);
alp des = 0.2*(alp des-templ) + 0.8*alp des_old;
alp-desd = (alp-des - alp_des_old)/dt;
alp-des_old = alp-des;
} else if (maneuver_id == 0) || (maneuver-id == 10)) alp-des = 0.0;
alp.des = min (alp.desd, 1.5);
alp.desd = max(alp.desd, 1.5);
alp.des = max(alp.des, 0.0);
alp-des = min(alp-des, 25.);
return alp-des;  // is this correct?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/compute_brake_pressure</td>
<td>double</td>
<td></td>
<td>Function</td>
</tr>
</tbody>
</table>

double brake_sw;
double time_filter = timer();
if (time_filter < 0.07) {
    brake_sw = 0;
    double pb_flt = pb();
}
brake_sw_old = brake_sw;
if (brake_sw == 0 && tnet_des <= 20.0) {
    brake = 1;
} else if (brake_sw == 1 && tnet_des >= 25.0)
    brake_sw = 0;
if (tnet << 0.0) {
    double abrk_des = - tnet_b/beta;
pb_des = f*abrk*accel_to_psi_factor*1.5;
    /* g-force to psi */
alp-des = 0.0;
} else if (brake_sw == 0)
pb des = 0.0; /* no brake pressure */
else -
    // pb des = 20.0;
/* fill up the brake line with pressure:stand-by mode */
pb des = 0.0; /* fill up the brake line with pressure:stand-by mode */

if (t ctrl <(t wait-0.04*0.0))
    if (maneuver-id == 0)
        pb des = stopped-brake press;
if (maneuver id != 10) {
    // Do we need a filter on desired brake ?????
    // pb des = pb des + (-pb des + pb des old) * sampling_time/0.01 ;
    pb des old = pb des;
    pb temp = 0.0;
}
else {
    pb temp +=
        (- pb temp + (stopped-brake-press - pb des old) ) / (0.5
        pb des = pb temp + pb des old;
    alp des = 0.0;
}

if ( pb des > 800.0)
    pb des = 800.0;

return pb des; //is this correct ?

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alp tbl</td>
<td>double</td>
<td>float pp, float air des</td>
<td></td>
</tr>
</tbody>
</table>

float a, b, p, q, r, s, t, p1, p2, p3, p4;
int i, j;

static float pm index[NUM AIR VALS] =
    { 15.0, 21.5, 28.1, 34.6, 41.2, 47.7, 54.2, 60.8, 67.3, 73.8, 80.4, 86.9, 93.5, 100.0}; /* kPa */

static float air index[NUM AIR INDEX] =
    { 2.0, 12.6, 23.1, 33.7, 44.3, 54.9, 65.4, 76.0, 86.6, 97.1, 107.7, 118.3, 128.9, 139.4, 150.0}; /* g/s */

static float air value[NUM AIR INDEX][NUM AIR VALS] =
    { 0.00, 0.00, 0.00, 0.00, 0.16, 2.97, 2.04, 2.88, 2.50, 2.70, 4.73, 5.46, 10.7, 16.9, 24.3, 0.79, 3.71, 7.57, 8.16, 8.25, 8.11, 8.49, 8.86, 9.28, 10.7, 11.2, 11.1, 11.5, 22.8, 30.9, 6.28, 8.53, 12.0, 12.2, 12.2, 12.1, 11.2, 12.4, 12.5, 12.4, 13.3, 14.2, 17.6, 26.7, 38.3, 11.8, 11.6, 14.5, 15.1, 14.8, 14.2, 14.1, 14.8, 15.0, 15.6, 16.3, 17.9, 21.9, 33.5, 49.5, 16.0, 14.5, 16.0, 16.9, 17.5, 17.5, 16.9, 17.5, 18.8, 21.0, 21.9, 22.0, 27.3, 41.3, 61.4, 18.9, 17.0, 16.9, 16.8, 19.8, 19.0, 19.4, 19.4, 19.7, 21.3, 24.8, 29.0, 34.6, 46.9, 76.4, 20.9, 18.8, 18.1, 18.8, 20.3, 20.9, 21.9, 21.2, 21.5, 23.1, 26.5, 33.1, 40.5, 51.5, 76.5, 21.8, 19.9, 19.2, 19.3, 20.5, 21.9, 22.9, 23.1, 24.5, 28.1, 35.1, 44.0, 55.2, 83.0, 22.2, 20.5, 19.1, 19.8, 21.0, 22.8, 23.9, 24.8, 25.7, 29.6, 39.5, 45.7, 58.3, 81.7, 22.1, 20.6, 20.1, 20.3, 21.5, 23.5, 25.5, 27.4, 27.4, 29.7, 36.0, 46.7, 61.1, 79.7, 21.6, 20.4, 19.9, 20.5, 21.9, 24.0, 26.6, 29.7, 31.8, 33.4, 40.8, 50.4, 63.1, 83.2,
20.7, 19.2, 19.6, 20.3, 22.0, 24.4, 27.5, 31.3, 35.4, 39.4, 44.9, 53.7, 64.9, 85.0,
19.5, 18.7, 18.8, 19.8, 21.8, 24.4, 28.0, 32.3, 37.2, 42.7, 49.1, 56.8, 68.0, 85.0,
17.8, 17.3, 17.7, 19.0, 21.2, 24.1, 28.0, 32.8, 37.2, 44.3, 51.6, 60.2, 72.2, 85.0,
15.9, 15.5, 16.2, 17.8, 20.2, 23.6, 27.8, 32.6, 38.4, 45.2, 53.0, 62.8, 76.3, 85.0

}; /* degree */

if( air_des < air_index[0] )
   i = 0;
else if( air_index[NUM_AIR_INDEX - 13] <= air_des )
   i = NUM_AIR_INDEX - 2; /* NUM_AIR_VALS - 2 in Leon's */
else
   for (i = 0; i < NUM_AIR_INDEX - 1; i++)
      if( (air_index[i] <= air_des) && (air_des <
          air_index[i + 13] )
          break;
}

if( pp < pm_index[0] )
   j = 0;
else if( pm_index[NUM_AIR_VALS - 13] <= pp )
   j = NUM_AIR_VALS - 2;
else
   for (j = 0; j < NUM_AIR_VALS - 1; j++)
      if( (pm_index[j] <= pp) && (pp < pm_index[j + 1]) )
          break;
}

p = air_des - air_index[i];
g = air_index[i + 1] - air_des;
p1 = air_value[i][j];
p2 = air_value[i + 1][j];
p3 = air_value[i][j + 1];
p4 = air_value[i + 1][j + 1];
a = (p1*q + p2*p) / (p + q);
b = (p3*q + p4*p) / (p + q);
s = pp - pm_index[j];
t = pm_index[j + 13] - pp;
r = (a*t + b*s) / (s + t);

r = (r - 1.0)*0.75; /* modified again: 11/1/96 */

if( r < 0.0 )
   r = 0.0;
else if( r > 85.0 )
   r = 85.0;
return( r );

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eng_tbl</td>
<td>double</td>
<td>float we, float in_value, int in_index, int out_index</td>
<td>Function</td>
</tr>
</tbody>
</table>

/* New engine map of Le Saber modified based upon the original data */
/* 8/6/96, Seibum B. Choi */
/* We [rad/s] Tnet [N-m] maod [g/s] Pm [kPa] alp [deg] */

float dat[255][5] = {
52.00, -63.58, 4.07, 10.00, 0.00,
52.00, -51.62, 2.22, 16.43, 0.00,
52.00, -28.53, 1.96, 22.86, 0.00,
52.00, -1.15, 2.78, 29.29, 1.43,
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Width</th>
<th>Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>52.00, 29.42</td>
<td>2.88</td>
<td>35.71</td>
<td>2.45</td>
</tr>
<tr>
<td>52.00, 57.09</td>
<td>3.87</td>
<td>42.14</td>
<td>3.19</td>
</tr>
<tr>
<td>52.00, 76.47</td>
<td>4.34</td>
<td>48.57</td>
<td>3.81</td>
</tr>
<tr>
<td>52.00, 92.72</td>
<td>4.23</td>
<td>55.00</td>
<td>4.47</td>
</tr>
<tr>
<td>52.00, 112.44</td>
<td>4.68</td>
<td>61.43</td>
<td>5.08</td>
</tr>
<tr>
<td>52.00, 134.69</td>
<td>5.80</td>
<td>67.86</td>
<td>5.91</td>
</tr>
<tr>
<td>52.00, 156.38</td>
<td>6.92</td>
<td>74.29</td>
<td>6.70</td>
</tr>
<tr>
<td>52.00, 179.84</td>
<td>8.55</td>
<td>80.71</td>
<td>7.16</td>
</tr>
<tr>
<td>52.00, 206.36</td>
<td>11.09</td>
<td>87.14</td>
<td>10.54</td>
</tr>
<tr>
<td>52.00, 229.02</td>
<td>12.41</td>
<td>93.57</td>
<td>25.23</td>
</tr>
<tr>
<td>52.00, 255.31</td>
<td>14.87</td>
<td>100.00</td>
<td>64.59</td>
</tr>
<tr>
<td>85.38, -64.91</td>
<td>2.89</td>
<td>10.00</td>
<td>0.00</td>
</tr>
<tr>
<td>85.38, -54.70</td>
<td>1.85</td>
<td>16.43</td>
<td>0.00</td>
</tr>
<tr>
<td>85.38, -25.76</td>
<td>3.17</td>
<td>22.86</td>
<td>0.00</td>
</tr>
<tr>
<td>85.38, 6.27</td>
<td>4.22</td>
<td>29.29</td>
<td>2.85</td>
</tr>
<tr>
<td>85.38, 34.48</td>
<td>4.97</td>
<td>35.71</td>
<td>3.81</td>
</tr>
<tr>
<td>85.38, 58.36</td>
<td>6.17</td>
<td>42.14</td>
<td>4.69</td>
</tr>
<tr>
<td>85.38, 77.27</td>
<td>7.19</td>
<td>48.57</td>
<td>5.43</td>
</tr>
<tr>
<td>85.38, 93.43</td>
<td>7.64</td>
<td>55.00</td>
<td>6.21</td>
</tr>
<tr>
<td>85.38, 113.25</td>
<td>8.63</td>
<td>61.43</td>
<td>6.99</td>
</tr>
<tr>
<td>85.38, 134.49</td>
<td>9.99</td>
<td>67.86</td>
<td>7.97</td>
</tr>
<tr>
<td>85.38, 156.51</td>
<td>11.83</td>
<td>74.29</td>
<td>9.06</td>
</tr>
<tr>
<td>85.38, 181.37</td>
<td>14.18</td>
<td>80.71</td>
<td>10.36</td>
</tr>
<tr>
<td>85.38, 207.77</td>
<td>17.25</td>
<td>87.14</td>
<td>12.31</td>
</tr>
<tr>
<td>85.38, 236.27</td>
<td>20.89</td>
<td>93.57</td>
<td>26.50</td>
</tr>
<tr>
<td>85.38, 265.19</td>
<td>24.71</td>
<td>100.00</td>
<td>74.22</td>
</tr>
<tr>
<td>118.75, -64.28</td>
<td>2.14</td>
<td>10.00</td>
<td>0.00</td>
</tr>
<tr>
<td>118.75, -51.81</td>
<td>2.14</td>
<td>16.43</td>
<td>0.00</td>
</tr>
<tr>
<td>118.75, -23.09</td>
<td>4.07</td>
<td>22.86</td>
<td>0.00</td>
</tr>
<tr>
<td>118.75, 7.30</td>
<td>5.59</td>
<td>29.29</td>
<td>3.93</td>
</tr>
<tr>
<td>118.75, 35.50</td>
<td>7.19</td>
<td>35.71</td>
<td>5.18</td>
</tr>
<tr>
<td>118.75, 58.19</td>
<td>8.77</td>
<td>42.14</td>
<td>6.01</td>
</tr>
<tr>
<td>118.75, 75.98</td>
<td>9.98</td>
<td>48.57</td>
<td>6.99</td>
</tr>
<tr>
<td>118.75, 91.56</td>
<td>10.71</td>
<td>55.00</td>
<td>7.86</td>
</tr>
<tr>
<td>118.75, 109.74</td>
<td>12.26</td>
<td>61.43</td>
<td>8.83</td>
</tr>
<tr>
<td>118.75, 129.16</td>
<td>13.84</td>
<td>67.86</td>
<td>9.88</td>
</tr>
<tr>
<td>118.75, 151.33</td>
<td>15.89</td>
<td>74.29</td>
<td>11.15</td>
</tr>
<tr>
<td>118.75, 176.45</td>
<td>18.39</td>
<td>80.71</td>
<td>12.96</td>
</tr>
<tr>
<td>118.75, 204.81</td>
<td>21.99</td>
<td>87.14</td>
<td>14.51</td>
</tr>
<tr>
<td>118.75, 236.23</td>
<td>27.36</td>
<td>93.57</td>
<td>28.79</td>
</tr>
<tr>
<td>118.75, 269.02</td>
<td>32.99</td>
<td>100.00</td>
<td>85.00</td>
</tr>
<tr>
<td>152.13, -61.53</td>
<td>1.83</td>
<td>10.00</td>
<td>0.00</td>
</tr>
<tr>
<td>152.13, -47.24</td>
<td>2.61</td>
<td>16.43</td>
<td>0.00</td>
</tr>
<tr>
<td>152.13, -20.30</td>
<td>5.04</td>
<td>22.86</td>
<td>0.00</td>
</tr>
<tr>
<td>152.13, 9.84</td>
<td>7.34</td>
<td>29.29</td>
<td>5.08</td>
</tr>
<tr>
<td>152.13, 36.90</td>
<td>9.62</td>
<td>35.71</td>
<td>6.48</td>
</tr>
<tr>
<td>152.13, 57.34</td>
<td>11.45</td>
<td>42.14</td>
<td>7.30</td>
</tr>
<tr>
<td>152.13, 74.19</td>
<td>12.77</td>
<td>48.57</td>
<td>8.50</td>
</tr>
<tr>
<td>152.13, 90.89</td>
<td>14.17</td>
<td>55.00</td>
<td>9.48</td>
</tr>
<tr>
<td>152.13, 107.88</td>
<td>15.86</td>
<td>61.43</td>
<td>10.55</td>
</tr>
<tr>
<td>152.13, 127.91</td>
<td>17.99</td>
<td>67.86</td>
<td>11.70</td>
</tr>
<tr>
<td>152.13, 149.38</td>
<td>20.64</td>
<td>74.29</td>
<td>13.11</td>
</tr>
<tr>
<td>152.13, 176.06</td>
<td>24.02</td>
<td>80.71</td>
<td>15.19</td>
</tr>
<tr>
<td>152.13, 206.67</td>
<td>28.45</td>
<td>87.14</td>
<td>17.33</td>
</tr>
<tr>
<td>152.13, 239.09</td>
<td>34.45</td>
<td>93.57</td>
<td>31.45</td>
</tr>
<tr>
<td>152.13, 273.22</td>
<td>42.06</td>
<td>100.00</td>
<td>85.00</td>
</tr>
<tr>
<td>185.50, -59.58</td>
<td>1.62</td>
<td>10.00</td>
<td>0.00</td>
</tr>
<tr>
<td>185.50, -42.64</td>
<td>3.20</td>
<td>16.43</td>
<td>0.00</td>
</tr>
<tr>
<td>185.50, -15.45</td>
<td>6.41</td>
<td>22.86</td>
<td>1.23</td>
</tr>
<tr>
<td>185.50, 14.25</td>
<td>9.40</td>
<td>29.29</td>
<td>6.27</td>
</tr>
<tr>
<td>185.50, 38.75</td>
<td>12.12</td>
<td>35.71</td>
<td>7.67</td>
</tr>
<tr>
<td>185.50, 56.84</td>
<td>14.14</td>
<td>42.14</td>
<td>8.58</td>
</tr>
<tr>
<td>185.50, 74.66</td>
<td>16.01</td>
<td>48.51</td>
<td>8.97</td>
</tr>
<tr>
<td>185.50, 91.33</td>
<td>18.00</td>
<td>55.00</td>
<td>11.03</td>
</tr>
<tr>
<td>185.50, 108.50</td>
<td>20.15</td>
<td>61.43</td>
<td>12.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>185.50, 129.15,</td>
<td>22.95, 67.86,</td>
<td>13.45,</td>
<td></td>
</tr>
<tr>
<td>185.50, 151.70,</td>
<td>26.27, 74.29,</td>
<td>15.01,</td>
<td></td>
</tr>
<tr>
<td>185.50, 179.70,</td>
<td>30.60, 80.71,</td>
<td>17.32,</td>
<td></td>
</tr>
<tr>
<td>185.50, 211.18,</td>
<td>35.97, 87.14,</td>
<td>20.06,</td>
<td></td>
</tr>
<tr>
<td>185.50, 243.86,</td>
<td>42.76, 93.57,</td>
<td>32.85,</td>
<td></td>
</tr>
<tr>
<td>185.50, 277.58,</td>
<td>51.19, 100.00,</td>
<td>85.00,</td>
<td></td>
</tr>
<tr>
<td>218.88, -58.20,</td>
<td>1.46, 10.00,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>218.88, -38.65,</td>
<td>3.88, 16.43,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>218.88, -8.01,</td>
<td>8.00, 22.86,</td>
<td>2.96,</td>
<td></td>
</tr>
<tr>
<td>218.88, 20.26,</td>
<td>11.74, 29.29,</td>
<td>7.49,</td>
<td></td>
</tr>
<tr>
<td>218.88, 40.75,</td>
<td>14.59, 35.71,</td>
<td>8.84,</td>
<td></td>
</tr>
<tr>
<td>218.88, 57.89,</td>
<td>17.00, 42.14,</td>
<td>9.98,</td>
<td></td>
</tr>
<tr>
<td>218.88, 77.37,</td>
<td>19.91, 48.57,</td>
<td>11.36,</td>
<td></td>
</tr>
<tr>
<td>218.88, 96.05,</td>
<td>22.82, 55.00,</td>
<td>12.50,</td>
<td></td>
</tr>
<tr>
<td>218.88, 112.11,</td>
<td>25.26, 61.43,</td>
<td>13.78,</td>
<td></td>
</tr>
<tr>
<td>218.88, 134.23,</td>
<td>28.95, 67.86,</td>
<td>15.17,</td>
<td></td>
</tr>
<tr>
<td>218.88, 158.14,</td>
<td>33.09, 74.29,</td>
<td>16.88,</td>
<td></td>
</tr>
<tr>
<td>218.88, 185.95,</td>
<td>38.12, 80.71,</td>
<td>19.39,</td>
<td></td>
</tr>
<tr>
<td>218.88, 216.62,</td>
<td>44.28, 87.14,</td>
<td>22.75,</td>
<td></td>
</tr>
<tr>
<td>218.88, 248.72,</td>
<td>51.71, 93.57,</td>
<td>36.42,</td>
<td></td>
</tr>
<tr>
<td>218.88, 282.17,</td>
<td>60.47, 100.00,</td>
<td>85.00,</td>
<td></td>
</tr>
<tr>
<td>252.25, -55.53,</td>
<td>1.66, 10.00,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>252.25, -33.77,</td>
<td>4.82, 16.43,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>252.25, -1.97,</td>
<td>9.80, 22.86,</td>
<td>4.71,</td>
<td></td>
</tr>
<tr>
<td>252.25, 25.52,</td>
<td>14.19, 29.29,</td>
<td>8.61,</td>
<td></td>
</tr>
<tr>
<td>252.25, 44.60,</td>
<td>17.65, 35.71,</td>
<td>10.04,</td>
<td></td>
</tr>
<tr>
<td>252.25, 62.94,</td>
<td>20.96, 42.14,</td>
<td>11.32,</td>
<td></td>
</tr>
<tr>
<td>252.25, 82.45,</td>
<td>24.64, 48.57,</td>
<td>12.69,</td>
<td></td>
</tr>
<tr>
<td>252.25, 101.43,</td>
<td>28.20, 55.00,</td>
<td>13.95,</td>
<td></td>
</tr>
<tr>
<td>252.25, 120.94,</td>
<td>31.88, 61.43,</td>
<td>15.31,</td>
<td></td>
</tr>
<tr>
<td>252.25, 143.25,</td>
<td>36.28, 67.86,</td>
<td>16.84,</td>
<td></td>
</tr>
<tr>
<td>252.25, 167.18,</td>
<td>41.11, 74.29,</td>
<td>18.64,</td>
<td></td>
</tr>
<tr>
<td>252.25, 193.78,</td>
<td>46.63, 80.71,</td>
<td>21.30,</td>
<td></td>
</tr>
<tr>
<td>252.25, 233.14,</td>
<td>55.21, 87.14,</td>
<td>25.17,</td>
<td></td>
</tr>
<tr>
<td>252.25, 253.93,</td>
<td>61.09, 93.57,</td>
<td>40.52,</td>
<td></td>
</tr>
<tr>
<td>252.25, 286.73,</td>
<td>70.46, 100.00,</td>
<td>85.00,</td>
<td></td>
</tr>
<tr>
<td>285.63, -52.56,</td>
<td>2.21, 10.00,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>285.63, -28.72,</td>
<td>6.15, 16.43,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>285.63, 1.67,</td>
<td>11.86, 22.86,</td>
<td>6.13,</td>
<td></td>
</tr>
<tr>
<td>285.63, 28.53,</td>
<td>17.11, 29.29,</td>
<td>9.66,</td>
<td></td>
</tr>
<tr>
<td>285.63, 49.45,</td>
<td>21.56, 35.71,</td>
<td>11.21,</td>
<td></td>
</tr>
<tr>
<td>285.63, 69.22,</td>
<td>25.84, 42.14,</td>
<td>12.57,</td>
<td></td>
</tr>
<tr>
<td>285.63, 88.66,</td>
<td>30.08, 48.57,</td>
<td>13.94,</td>
<td></td>
</tr>
<tr>
<td>285.63, 108.92,</td>
<td>34.55, 55.00,</td>
<td>15.31,</td>
<td></td>
</tr>
<tr>
<td>285.63, 129.99,</td>
<td>39.20, 61.43,</td>
<td>16.79,</td>
<td></td>
</tr>
<tr>
<td>285.63, 152.49,</td>
<td>44.30, 67.86,</td>
<td>18.45,</td>
<td></td>
</tr>
<tr>
<td>285.63, 176.40,</td>
<td>49.78, 74.29,</td>
<td>20.40,</td>
<td></td>
</tr>
<tr>
<td>285.63, 202.29,</td>
<td>55.95, 80.71,</td>
<td>23.08,</td>
<td></td>
</tr>
<tr>
<td>285.63, 230.06,</td>
<td>62.87, 87.14,</td>
<td>27.60,</td>
<td></td>
</tr>
<tr>
<td>285.63, 259.61,</td>
<td>71.18, 93.57,</td>
<td>43.78,</td>
<td></td>
</tr>
<tr>
<td>285.63, 291.11,</td>
<td>80.75, 100.00,</td>
<td>85.00,</td>
<td></td>
</tr>
<tr>
<td>319.00, -50.82,</td>
<td>2.91, 10.00,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>319.00, -25.65,</td>
<td>7.71, 16.43,</td>
<td>0.00,</td>
<td></td>
</tr>
<tr>
<td>319.00, 3.94,</td>
<td>14.17, 22.86,</td>
<td>7.38,</td>
<td></td>
</tr>
<tr>
<td>319.00, 31.10,</td>
<td>20.46, 29.29,</td>
<td>10.65,</td>
<td></td>
</tr>
<tr>
<td>319.00, 53.75,</td>
<td>25.88, 35.71,</td>
<td>12.32,</td>
<td></td>
</tr>
<tr>
<td>319.00, 74.27,</td>
<td>30.88, 42.14,</td>
<td>13.72,</td>
<td></td>
</tr>
<tr>
<td>319.00, 94.05,</td>
<td>35.80, 48.57,</td>
<td>15.14,</td>
<td></td>
</tr>
<tr>
<td>319.00, 115.10,</td>
<td>40.99, 55.00,</td>
<td>16.60,</td>
<td></td>
</tr>
<tr>
<td>319.00, 137.04,</td>
<td>46.45, 61.43,</td>
<td>18.19,</td>
<td></td>
</tr>
<tr>
<td>319.00, 159.96,</td>
<td>52.24, 67.86,</td>
<td>19.98,</td>
<td></td>
</tr>
<tr>
<td>319.00, 183.92,</td>
<td>58.51, 74.29,</td>
<td>22.09,</td>
<td></td>
</tr>
<tr>
<td>319.00, 209.54,</td>
<td>65.37, 80.71,</td>
<td>24.69,</td>
<td></td>
</tr>
<tr>
<td>319.00, 236.53,</td>
<td>72.88, 87.14,</td>
<td>30.19,</td>
<td></td>
</tr>
<tr>
<td>319.00, 264.69,</td>
<td>81.56, 93.57,</td>
<td>46.56,</td>
<td></td>
</tr>
<tr>
<td>319.00, 294.07, 90.99, 100.00, 85.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, -50.70, 3.65, 10.00, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, -24.05, 9.60, 16.43, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 4.79, 16.48, 22.86, 8.71,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 31.83, 23.74, 29.29, 11.53,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 55.68, 30.10, 35.71, 13.34,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 77.58, 35.92, 42.14, 14.78,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 98.44, 41.55, 48.57, 16.26,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 120.34, 47.49, 55.00, 17.80,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 142.74, 53.65, 61.43, 19.51,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 165.91, 60.12, 67.86, 21.42,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 190.53, 67.13, 74.29, 23.57,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 215.31, 74.52, 80.71, 26.41,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 241.14, 82.64, 87.14, 32.82,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 268.15, 91.77, 93.57, 49.52,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>352.38, 295.54, 101.35, 100.00, 85.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, -51.34, 4.57, 10.00, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, -25.23, 11.27, 16.43, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 4.31, 18.80, 22.86, 9.56,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 31.03, 26.64, 29.29, 12.34,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 55.44, 33.98, 35.71, 14.29,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 79.16, 40.86, 42.14, 15.72,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 100.72, 46.98, 48.57, 17.27,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 123.38, 53.67, 55.00, 18.92,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 145.71, 60.34, 61.43, 20.74,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 169.99, 67.77, 67.86, 22.79,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 193.79, 75.24, 74.29, 25.06,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 218.33, 83.26, 80.71, 28.25,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 243.88, 91.99, 87.14, 35.73,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 269.30, 101.60, 93.57, 52.76,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>385.75, 295.46, 111.79, 100.00, 85.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, -54.35, 5.21, 10.00, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, -27.44, 13.26, 16.43, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 2.08, 21.22, -22.86, 10.46,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 28.79, 29.38, 29.29, 13.05,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 53.63, 37.45, 35.71, 15.15,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 78.04, 45.17, 42.14, 16.57,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 101.25, 52.22, 48.57, 18.20,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 123.84, 59.29, 55.00, 19.94,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 147.26, 66.90, 61.43, 21.88,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 170.61, 74.64, 67.86, 24.07,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 194.56, 82.81, 74.29, 26.61,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 218.15, 91.31, 80.71, 30.22,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 243.69, 100.63, 87.14, 38.55,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 267.62, 110.85, 93.57, 56.37,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>419.13, 293.18, 122.13, 100.00, 85.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, -58.31, 5.77, 10.00, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, -31.38, 15.02, 16.43, 0.17,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, -2.05, 23.49, 22.86, 10.68,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 25.25, 31.77, 29.29, 13.70,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 50.18, 40.51, 35.71, 15.91,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 74.94, 48.89, 42.14, 17.32,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 99.06, 56.80, 48.57, 19.04,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 122.24, 64.51, 55.00, 20.88,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 145.31, 72.52, 61.43, 22.93,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 168.98, 81.00, 67.86, 25.26,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 192.14, 89.64, 74.29, 28.10,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 215.82, 98.82, 80.71, 32.27,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 239.87, 108.55, 87.14, 41.62,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 263.52, 119.68, 93.57, 58.51,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>452.50, 287.34, 130.91, 100.00, 85.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>485.88, -63.53, 6.11, 10.00, 0.00,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>485.88, -36.33, 15.69, 16.43, 1.11,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>485.88, -8.09, 25.18, 22.86, 10.99,</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>485.88, 20.01, 33.60, 29.29, 14.27,</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
float ta, outa, tb, outb, a, b, r, ww;
int i, j, k;

ww = we * 2.0 * PI / 60.0; /* rpm to rad/s */

if ( ww < dat[0][0] )
{
  i = 0;
}
reglayer

```c
ww = dat[0][0];

} else if( ww >= dat[254][0] )
{
    i = 9;
    ww = dat[254][0];
}
else
{
    for( i = 0; i < 16; i++ )
    {
        if( ww >= dat[i*15][0] && ww < dat[(i+1) * 15][0] )
            break;
    }

    if( in_value < dat[i*15] [in_index] )
    {
        j = 0;
        /* in_value = dat[i+15][in_index];  /**/
    }
    else if( in_value > dat[(i+1)*15] [in_index] )
    {
        j = 13;
        /* in_value = dat[i*15 + 14][in_index];  */
    }
    else
    {
        for( j = 0; j < 14; j++ )
        {
            if( in_value >= dat[i*15 + j] [in_index] 
               && in_value < dat[i*15 + j + 11] [in_index] )
                break;
        }

        if( in_value < dat[(i + 1)*15] [in_index] )
        {
            k = 0;
            /* in_value = dat[(i+1)*15] [in_index];  */
        }
    else if( in_value > dat[(i + 1)*15 + 14] [in_index] )
    {
        k = 13;
        /* in_value = dat[(i + 1)*15 + 14] [in_index];  */
    }
    else
    {
        for( k = 0; k < 14; k++ )
        {
            if( in_value >= dat[ (i + 1)*15 + k ] [in_index] 
               && in_value < dat[(i + 1)*15 - k +1] [in_index] )
                break;
        }
    }

    a = ww - dat[i*15][0];
    b = dat[(i + 1)*15][0] - ww;

    ta = (dat[ (i + 1)*15 + k ] [in_index] * a 
          + dat[i*15 + j] [in_index] * b) / (a + b);

    outa = (dat[ (i + 1)*15 + k ] [out_index] * a 
            + dat[i*15 + j] [out_index] * b) / (a + b);

    tb = (dat[ (i + 1)*15 + k + 1 ] [in_index] * a 
          + dat[i*15 + j + 1] [in_index] * b) / (a + b);

    outb = (dat[ (i + 1)*15 + k + 13 ] [out_index] * a 
            + dat[i*15 + j + 1] [out_index] * b) / (a + b);
```
\[a = \text{in-value} - \text{ta};\]
\[b = \text{tb} - \text{in-value};\]
\[r = \frac{\text{outb} \times a + \text{outa} \times b}{a + b};\]
\[\text{return}(\ r\ );\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write_command</td>
<td>void</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
#ifdef QNX

db data typ data;
long_output_typ* p;

/* retrieve the current info from the database */
if (clt_read(DBClient, DB_LONG_OUTPUT_VAR, DB_LONG_OUTPUT_TYPE, &data) == FALSE)
    Error("LowerController", "cannot read DB_LONG_OUTPUT_VAR");

p = (long_output_typ*) data.value.user;

/* modify the throttle angle and the brake pressure */
p->throttle = throttle angle;
p->brake = brake-pressure;

/* update the database variable */
if (clt_update(DBClient, DB_LONG_OUTPUT_VAR, DB_LONG_OUTPUT_TYPE, sizeof(long_output_typ), (void*) data.value.user) == FALSE)
    Error("LowerController", "cannot update DB_LONG_OUTPUT_VAR");
#endif
```

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>Link Type</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>synthetic_accel</td>
<td>double</td>
<td>upper_controller</td>
<td>UpperController*</td>
<td>synthetic_accel</td>
</tr>
<tr>
<td>gear_ratio</td>
<td>double</td>
<td>transmission</td>
<td>Transmission*</td>
<td>gear_ratio</td>
</tr>
<tr>
<td>we_flt</td>
<td>double</td>
<td>engine_speed_filter</td>
<td>EngineSpeedFilter*</td>
<td>we_flt</td>
</tr>
<tr>
<td>pm_flt</td>
<td>double</td>
<td>manifold-pressure_filter</td>
<td>ManifoldPressureFilter*</td>
<td>pm_flt</td>
</tr>
<tr>
<td>wheel_speed</td>
<td>double</td>
<td>car_sensor</td>
<td>O2Sensor*</td>
<td>car_speed</td>
</tr>
<tr>
<td>pb</td>
<td>double</td>
<td>Delphi_brakes</td>
<td>BrakeSensor*</td>
<td>brake_pressure</td>
</tr>
</tbody>
</table>

**Continuous States**
### Continuous State Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
<tr>
<td>gear_change_timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
</tbody>
</table>

### Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
<td>timer' = 1</td>
<td>gear_change_timer' = 1</td>
</tr>
<tr>
<td>start</td>
<td>Discrete state</td>
<td>timer' = 0</td>
<td>gear_change_timer' = 0</td>
</tr>
</tbody>
</table>

### Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>controller_poll</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

```java
boolean {};
```

**Action**

```java
engine_speed_filter->filtered_engine_speed();
manifold_pressure_filter->filtered_manifold_pressure();
```

**Reset Variable**

```java
if (tnet_des >= tnet_min)
  {
    tnet_a = tnet_des;
    tnet_b = 0.0;
  }
else if (tnet_des < tnet_min)
  {
    tnet_a = tnet_min;
    tnet_b = tnet_des - tnet_min;
  }
else
  {
    tnet_a = tnet_des*(tnet_des - tnet_min)/(10.0 - tnet_min);
    tnet_b = tnet_des*(-10.0 - tnet_des)/(10.0 - tnet_min);
  }
```

**Reset Variable**

```java
/throttle_angle /compute_throttle_angle()  
/throttle_angle /compute_brake_pressure() 
```

**Output Event**
### Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>trj_begin</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean {1}

**Action**

- Reset Continuous State: timer = 0
- Reset Variable: maneuver_id = 1

**Output Event**

### Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>gear_change</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean {1}

**Action**

- Reset Continuous State: gear_change_timer = 0

**Output Event**

### Transition 3

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>idle</td>
<td>initialize</td>
<td>bproaction</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean {1}

**Action**

- Reset Variable: r_star = gear_ratio() * rd;
- Reset Variable: r-starh = gear_ratio() * rd * hr
- Reset Variable: eff_in = (mass * hr * hr + i_whl) * r_star * r_star + 1_eng
- Reset Variable: beta = eff_in / r_starh

**Output Event**

### Transition 4
<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>trj_end</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {1}

Action

Reset Variable maneuver_id[10]

Output Event

Include Files and other Preprocessor Directives

/
  * Include files.
  */

#ifdef QNX

#ifdef DATABASE_INCLUDED
#define DATABASE_INCLUDED

#ifdef __cplusplus
extern "C"
{
#endif

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/name.h>
#include <sys/kernel.h>
#include <sys/proxy.h>
#include <signal.h>
#include <setjmp.h>
#include <local.h>
#include <sys_list.h>

#include "db_comm.h"
#include "db_clt.h"
#include "track.h"
#include "veh_iols.h"
#include "clt-vars.h"

#ifdef __cplusplus
}
#endif

extern db_clt_typ* DBClient;
#endif // ifdef DATABASE_INCLUDED

#endif // ifdef QNX

Header Code
LowerController();

C++ Code

LowerController: :LowerController()
{
}

Class ManifoldPressureFilter

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of ManifoldPressureFilter</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pm_old</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>pm_flt</td>
<td>double</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine_sensor</td>
<td>EngineSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>lower_controller</td>
<td>LowerController*</td>
<td>Link</td>
</tr>
<tr>
<td>upper_controller</td>
<td>UpperController*</td>
<td>Link</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(filtered manifold pressure)</td>
<td>double</td>
<td></td>
<td>Function</td>
</tr>
</tbody>
</table>

double pm = engine manifold pressure;

if(timer() < 0.01) {
    pm-old = pm;
    pm_flt = pm;
} else {
    if (timer() <= 1.5
        || gear-change timer() < 0.8
        || gear_change_timer() > 1.5) {
        if (pm - pm_old > 0.2)
            pm = pm_old + 0.2;
        else if (pm - pm_old < -0.2)
            pm = pm-old - 0.2;
    } else {
        if (pm - pm_old > 0.8)
pm = pm_old + 0.8;
else if (pm - pm_old < -0.8)
    pm = pm_old - 0.8;
}
pm_old = pm;
double templ;
if (timer() < 5.0)
    templ = (-5.0 - timer())*1.0 + 10.0;
else
    templ = 10.0;
double temp = zeta() + sqrt(fabs(zeta() * zeta() - 1));

return pm_flt;

Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>Link Type</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>engine_manifold_pressure</td>
<td>double</td>
<td>engine_sensor</td>
<td>EngineSensor*</td>
<td>engine_manifold_pressure</td>
</tr>
<tr>
<td>pmdes_dot</td>
<td>double</td>
<td>lower_controller</td>
<td>LowerController*</td>
<td>pmdes_dot</td>
</tr>
<tr>
<td>sampling_time</td>
<td>double</td>
<td>lower_controller</td>
<td>LowerController*</td>
<td>Fling-time</td>
</tr>
<tr>
<td>zeta</td>
<td>double</td>
<td>upper_controller</td>
<td>UpperController*</td>
<td>zeta</td>
</tr>
</tbody>
</table>

Continuous States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>.timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
<tr>
<td>gear-change-time</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
<td></td>
<td>timer' = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>gear-change-timer'  = 1</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>trj_begin</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {1}

Action

Reset Continuous State !timer[0]

Output Event
Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>gear_change</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean [1]

Action

Reset Continuous State|gear_change_timer[0]

Output Event

Include Files and other Preprocessor Directives

Header Code

ManifoldPressureFilter();

C++ Code

ManifoldPressureFilter::ManifoldPressureFilter()
{
}

Class Monitor

This component "monitors" all the other components in the real time mode and saves the specified variable values in the run, for further analysis.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of Monitor</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshots</td>
<td>Set&lt;Snapshot*&gt;*</td>
<td>The set of monitor snapshots.</td>
</tr>
</tbody>
</table>

Links
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>brake_sensor</td>
<td>BrakeSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>engine_sensor</td>
<td>EngineSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>transmission</td>
<td>Transmission*</td>
<td>Link</td>
</tr>
<tr>
<td>radar</td>
<td>Radar*</td>
<td>Link</td>
</tr>
<tr>
<td>lateral_controller</td>
<td>LateralController*</td>
<td>Link</td>
</tr>
<tr>
<td>lower_controller</td>
<td>LowerController*</td>
<td>Link</td>
</tr>
<tr>
<td>upper_controller</td>
<td>UpperController*</td>
<td>Link</td>
</tr>
<tr>
<td>trajectory</td>
<td>Trajectory*</td>
<td>Link</td>
</tr>
<tr>
<td>lc_lsp</td>
<td>LateralSensorProcessor*</td>
<td>Link</td>
</tr>
<tr>
<td>cx_lsp</td>
<td>LateralSensorProcessor*</td>
<td>Link</td>
</tr>
<tr>
<td>engine_speed_filter</td>
<td>EngineSpeedFilter*</td>
<td>Link</td>
</tr>
<tr>
<td>man_pressure_filter</td>
<td>ManifoldPressureFilter*</td>
<td>Link</td>
</tr>
<tr>
<td>left</td>
<td>LateralSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>center</td>
<td>LateralSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>right</td>
<td>LateralSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>radar_sp</td>
<td>RadarSensorProcessor*</td>
<td>Link</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/idle</td>
<td>Discrete state</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>/From/To</th>
<th>Event</th>
<th>/Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/idle /idle</td>
<td>lateral_command</td>
<td>response</td>
<td>Create and store a lateral snapshot, every time when receive the lateral-command event from the lateral controller [every 20ms].</td>
</tr>
</tbody>
</table>
Guard

boolean {l}

Action

```java
LateralSnapshot* s = new LateralSnapshot;

// store all interesting values
s->car_speed = car_sensor->car_speed;
s->steer_angle = car_sensor->steer_angle;

s->lc_cx1 = lc_lsp->cx1;
s->lc_cx2 = lc_lsp->cx2;
s->lc_cy1 = lc_lsp->cy1;
s->lc_cy2 = lc_lsp->cy2;
s->lc_cz1 = lc_lsp->cz1;
s->lc_cz2 = lc_lsp->cz2;
s->lc_x_earth = lc_lsp->x_earth;
s->lc_y_earth = lc_lsp->y_earth;
s->lc_z_earth = lc_lsp->z_earth;
s->lc_y_act = lc_lsp->y_act;
s->lc_y_der = lc_lsp->y_der;
s->lc_valid_y_output = lc_lsp->valid_y_output_avg;

s->cr_cx1 = cr_lsp->cx1;
s->cr_cx2 = cr_lsp->cx2;
s->cr_cy1 = cr_lsp->cy1;
s->cr_cy2 = cr_lsp->cy2;
s->cr_cz1 = cr_lsp->cz1;
s->cr_cz2 = cr_lsp->cz2;
s->cr_x_earth = cr_lsp->x_earth;
s->cr_y_earth = cr_lsp->y_earth;
s->cr_z_earth = cr_lsp->z_earth;
s->cr_y_act = cr_lsp->y_act;
s->cr_y_der = cr_lsp->y_der;
s->cr_valid_y_output = cr_lsp->valid_y_output_avg;

s->d_s = lateral_controller->d_s;
s->d_sdot = lateral_controller->d_sdot;
s->d_shat_dot = lateral_controller->d_shat_dot;
s->delta_f = lateral_controller->delta_f;

s->mag_fl_x = left->x_measurement;
s->mag_fl_y = left->y_measurement;
s->mag_fl_z = left->z_measurement;
s->mag_fc_x = center->x_measurement;
s->mag_fc_y = center->y_measurement;
s->mag_fc_z = center->z_measurement;
s->mag_fr_x = right->x_measurement;
s->mag_fr_y = right->y_measurement;
s->mag_fr_z = right->z_measurement;

s->control_angle = lateral_controller->control_angle;
s->delta_0 = lateral_controller->delta_0;

// store the snapshot
snapshots->addNoDupCheck(s);
```

Output Event
## Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>idle</td>
<td>Initialize</td>
<td>proaction</td>
<td>This transition is added to initialize the global variable 'TheMonitor' to 'this' monitor. This global variable is needed in the end, when the execution terminates, to print (and then to delete) all the stored snapshots.</td>
</tr>
</tbody>
</table>

### Guard

boolean {1}

### Action

```c
extern Monitor* TheMonitor;
TheMonitor = this;
```

### Output Event

- 

## Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td>controller_poll</td>
<td>response</td>
<td>Create and store a longitudinal snapshot, on each (upper) controller-poll event [every 20ms].</td>
</tr>
</tbody>
</table>

### Guard

boolean {1}

### Action
```cpp
LongitudinalSnapshot* s = new LongitudinalSnapshot;

// store all interesting values
s->car_speed = car_sensor->car_speed;
s->platoon_position = car_sensor->platoon_position;
s->brake_pressure = brake_sensor->brake_pressure;
s->engine_speed = engine_sensor->engine_speed;
s->engine_manifold_pressure = engine_sensor->engine-manifold-pressure;
s->gear = transmission->gear;
s->gear_ratio = transmission->gear-ratio;
s->range = radar->range;
s->rangerate = radar->rangerate;
s->we_flt = engine_speed_filter->we_flt;
s->pm_flt = man_pressure_filter->pm_flt;
s->synthetic_accel = upper_controller->synthetic_accel;
s->throttle_angle = lower_controller->throttle_angle;
s->brake_pressure = lower_controller->brake_pressure;
s->tnet_des = lower_controller->tnet_des;
s->ma_des = lower_controller->ma_des;
s->a_des = trajectory->desired-acceleration();
s->v_des = trajectory->desired-speed();

// store the snapshot in the monitor list
snapshots->addNoDupCheck(s);
```

**Output Event**

---

**Include Files and other Preprocessor Directives**

```cpp
#include <stdio.h>

/* Snapshot Class Definition. 
 * A snapshot is basically a data structure containing 
 * the values of some meaningful variables defined in the model. The snapshots are stored in the main 
 * memory during the model execution and printed out 
 * when the execution terminates. Usually we stamp 
 * each snapshot with the time when it was taken. */

class Snapshot {
  public:
    float time;

  public:

    Snapshot() {
      time = DEM::elapsedTime();
    }
```
virtual ~Snapshot() {} 
virtual void Print(FILE* file) = 0;

/* LateralSnapshot Class Definition.

* This is the snapshot used for debugging the lateral controller. It stores values of variables from the sensors (car-speed, steer-angle, magnetometers measurements) and lateral controller (d_s, delta-f...).
*/

class LateralSnapshot : public Snapshot {

public:
float car-speed; /* car sensor */
float steer-angle;
float lc_cx1, lc_cx2; /* left-center */
float lc_cy1, lc_cy2;
float lc_cz1, lc_cz2;
float lc_x_earth, lc_y_earth, lc_z_earth;
float lc_y_act, lc_y_der;
float lc_valid_y_output;
float cr_cx1, cr_cx2; /* center-right */
float cr_cy1, cr_cy2;
float cr_cz1, cr_cz2;
float cr_x_earth, cr_y_earth, cr_z_earth;
float cr_y_act, cr_y_der;
float cr_valid_y_output;
float d_s, d_sdot; /* lateral controller */
float d_shat_dot;
float delta_f;
float mag_fl_x; /* magnetometers */
float mag_fl_y;
float mag_fl_z;
float mag_fc_x;
float mag_fc_y;
float mag_fc_z;
float mag_fr_x;
float mag_fr_y;
float mag_fr_z;
float control_angle, delta_0; /* lateral controller */

public:
LateralSnapshot() {}
virtual ~LateralSnapshot() {} 
virtual void Print(FILE* file);

};

/* LongitudinalSnapshot Class Definition.*/
* This is the snapshot used for debugging the longitudinal controller. It stores variables from sensors (car, brake, engine,...), filters (manifold-pressure, engine-speed), controllers, and the desired trajectory.

```cpp
class LongitudinalSnapshot : public Snapshot {

public:
    float car_speed;  // car sensor
    int platoon_position;
    float brake_pressure;  // brake sensor
    float engine_speed;  // engine sensor
    float engine_manifold_pressure;
    int gear;  // transmission
    float gear_ratio;
    float range;  // radar
    float rangerate;
    float we_flit;  // filters
    float pm_flit;
    float synthetic_accel;  // upper controller
    float throttle_angle;  // lower controller
    float brake_pressure;
    float tnet_des;
    float ma_des;
    float a_des;  // lead long trajectory
    float v_des;
    float estimated_range;  // radar sensor processor

public:
    LongitudinalSnapshot();
    virtual ~LongitudinalSnapshot();
    virtual void Print(FILE*);

};
```

**Header Code**

Monitor();

void Print(FILE*);

**C++ Code**

/* Printing method for the Lateralsnapshot. */

```cpp
void LateralSnapshot::Print(FILE* file)
{
    fprintf(file, "%f\t\t",
            time);

    fprintf(file, "%f\t%f\t\t",
```
car_speed, steer_angle);

fprintf(file, "$f\ t\ f\ n$" ,
lc_xl, lc cx2);
fprintf(file, "$f\ t\ f\ n$",
lc_yl, lc cy2);
fprintf(file, "$f\ t\ f\ n$",
lc_zl, lc cz2);
fprintf(file, "$f\ t\ f\ n$",
lc_x_earth, lc y_earth, lc_z_earth);
fprintf(file, "$f\ t\ f\ n$",
lc_y_act, lc y_der);
fprintf(file, "$f\ t\ f\ n$",
lc_valid y_output);

fprintf(file, "$f\ t\ f\ n$",
cr_xl, cr cx2);
fprintf(file, "$f\ t\ f\ n$",
 cr_yl, cr cy2);
fprintf(file, "$f\ t\ f\ n$",
 cr_zl, cr cz2);
fprintf(file, "$f\ t\ f\ n$",
 cr_x_earth, cr y_earth, cr_z_earth);
fprintf(file, "$f\ t\ f\ n$",
 cr_y_act, cr y_der);
fprintf(file, "$f\ t\ f\ n$",
 cr_valid y_output);

fprintf(file, "$f\ t\ f\ n$",
d s, d sdot, d shat dot, delta f);

fprintf(file, "$f\ t\ f\ n$",
mag fl_x, mag fl_y, mag fl_z);
fprintf(file, "$f\ t\ f\ n$",
mag fc x, mag fc_y, mag fc z);
fprintf(file, "$f\ t\ f\ n$",
mag fr x, mag fr_y, mag fr z);

fprintf(file, "$f\ t\ f\ n$",
control_angle, delta_0);

/* Printing method for the LongitudinalSnapshot. */

void LongitudinalSnapshot::Print(FILE* file){
    fprintf(file, "$f\ t\ t$",
time);
    fprintf(file, "$f\ t\ t$",
car_speed, platoon_position);
    fprintf(file, "$f\ t\ t$",
brake_pressure);
    fprintf(file, "$f\ t\ t$",
engine_speed, engine_manifold_pressure);
```c
fprintf(file, "%d\t%f\t\t\t", gear, gear_ratio);
fprintf(file, "%f\t%f\t\t", range, rangerate);
fprintf(file, "%f\t%f\t\t", we_flt, pm_flt);
fprintf(file, "%f\t\t", synthetic_accel);
fprintf(file, "%f\t%f\t%f\t%f\t\t", throttle_angle, brake_pressure, tnet_des, ma_des);
fprintf(file, "%f\t%f\t\t", a_des, v_des);
fprintf(file, "%f\n", estimated_range);
}

/* Monitor implementation. */

Monitor::Monitor(){
    snapshots = new Set<Snapshot*>;
}

void Monitor::Print(FILE* file){
    for(int i = 0; i < snapshots->size; i++)
        snapshots->elements[i]->Print(file);
}

Class Radar

The radar component simulates the physical radar. It produces a random range input that varies between 3.7m and 4.3m. In the physical system, it is replaced by the radar sensor device.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Superclass of Radar</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>rangerate</td>
<td>double</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Discrete States

65 of 94
### Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_long</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean {1}

**Action**

- Reset Variable: range
- Reset Variable: rangerate
- reader->radar_range
- reader->radar_rangerate

**Output Event**

---

### Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>new_data</td>
<td>function</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean {1}

**Action**

**Output Event**

---

### Include Files and other Preprocessor Directives

### Header Code

Radar();

### C++ Code

Radar::Radar()
{
}

### Class RadarSensorProcessor

This component takes in the raw radar range reading, the measured wheel speed of the car and the communicated wheel speed of the car in front. It outputs the estimated range. It checks for and signals error in case the input is stuck at zero.

### Superclasses
### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>observer_gain</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>T</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>poll_period</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>fault_timeout</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>fault_flag</td>
<td>int</td>
<td>Variable</td>
</tr>
<tr>
<td>stored_radar</td>
<td>double</td>
<td>Variable</td>
</tr>
</tbody>
</table>

### Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iradar</td>
<td>Radar+</td>
<td>Link</td>
</tr>
<tr>
<td>/car-sensor</td>
<td>CarSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>comm</td>
<td>CommunicatedVariables*</td>
<td>Link</td>
</tr>
</tbody>
</table>

### Functions

```
return estimated_range();
```

### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>Link Type</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_radar</td>
<td>double</td>
<td>radar</td>
<td>Radar*</td>
<td>range</td>
</tr>
<tr>
<td>/speed</td>
<td>double</td>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>car_speed</td>
</tr>
<tr>
<td>/preceding-speed</td>
<td>double</td>
<td>comm</td>
<td>CommunicatedVariables*</td>
<td>preceding_speed</td>
</tr>
<tr>
<td>platoon_position</td>
<td>int</td>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>platoon_position</td>
</tr>
</tbody>
</table>

### Continuous States
## Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
</tr>
</thead>
</table>
| 'normal       | Discrete state       | filtered_radar' = 
(1/T)*filtered_radar() + 
(1/T)*input_radar()                                                                                                                                   |
|               |                      | estimated_range' = preceding-speed() - speed() + (platoon-position() == 1 ? 0 : observer_gain) * (filtered_radar() - estimated_range())          |
|               |                      | poll_timer' = 1                                                                                                                                         |
|               |                      | fault_timer' = 0                                                                                                                                         |
| potential_fault| Discrete state       | filtered_radar' = 
(1/T)*filtered_radar() + 
(1/T)*stored_radar                                                                                                                                     |
|               |                      | estimated_range' = preceding-speed() - speed() + (platoon-position() == 1 ? 0 : observer_gain) * (filtered_radar() - estimated_range())          |
|               |                      | poll_timer' = 1                                                                                                                                         |
|               |                      | fault_timer' = 1                                                                                                                                         |
| fault         | Discrete state       | filtered_radar' = 0                                                                                                                                 |
|               |                      | estimated_range' = 0                                                                                                                                    |
|               |                      | poll_timer' = 0                                                                                                                                         |
|               |                      | fault_timer' = 0                                                                                                                                         |

### Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>normal</td>
<td>radar_poll</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

threshold (poll_timer >= poll_period)

**Action**
<table>
<thead>
<tr>
<th>Transition 1</th>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>potential_fault</td>
<td>potential_fault</td>
<td>radar_poll</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

threshold \( \text{poll\_timer} \geq \text{poll\_period} \)

Action

<table>
<thead>
<tr>
<th>Reset Continuous State</th>
<th>\text{poll_timer}</th>
<th>\text{poll_timer()} - \text{poll_period}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Variable</td>
<td>\text{fault_flag}</td>
<td>input_radar() == 0 ? 1 : 0</td>
</tr>
</tbody>
</table>

Output Event

<table>
<thead>
<tr>
<th>Transition 2</th>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\text{normal}</td>
<td>potential_fault</td>
<td>fault_start</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean \{ \text{fault\_flag} == 1 \}

Action

| Reset Continuous State | \text{Fault\_timer} | 0 |

Output Event

<table>
<thead>
<tr>
<th>Transition 3</th>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>potential_fault</td>
<td>normal</td>
<td>fault_clear</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean \{ \text{fault\_flag} == 0 \}

Action

Output Event

<table>
<thead>
<tr>
<th>Transition 4</th>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
Guard
threshold {fault_timer >= fault-timeout}

Action
Output Event

Include Files and other Preprocessor Directives

Header Code

RadarSensorProcessor();

C++ Code

RadarSensorProcessor::RadarSensorProcessor()
{
}

Class Sensor

This is the superclass of all the sensor components, which get data from DataReader.

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of Sensor</td>
</tr>
</tbody>
</table>

Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!reader</td>
<td>DataReader*</td>
<td>Link to DataReader to access the data</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
</tr>
</tbody>
</table>

Include Files and other Preprocessor Directives

Header Code

Sensor();

C++ Code
Sensor::Sensor()
{
}

Class Trajectory

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of Trajectory</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ispeed</td>
<td>double</td>
<td>the current desired speed.</td>
</tr>
<tr>
<td>acceleration_rate</td>
<td>double</td>
<td></td>
</tr>
<tr>
<td>speed_tbl</td>
<td>set&lt;double&gt;</td>
<td>Trajectory speed table.</td>
</tr>
<tr>
<td>time_tbl</td>
<td>set&lt;double&gt;</td>
<td>Trajectory times table.</td>
</tr>
<tr>
<td>step</td>
<td>int</td>
<td>The current step of trajectory.</td>
</tr>
<tr>
<td>/speed_poll_period</td>
<td>double</td>
<td>speed poll period parameter [constant].</td>
</tr>
<tr>
<td>beta</td>
<td>double</td>
<td>acceleration rate control parameter [constant].</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desired_speed</td>
<td>double</td>
<td>return speed;</td>
</tr>
<tr>
<td>desired_acceleration</td>
<td>double</td>
<td>return acceleration();</td>
</tr>
</tbody>
</table>

Continuous States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>acceleration</td>
<td>The current desired acceleration.</td>
<td>0</td>
</tr>
<tr>
<td>trajectory_step_timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
<tr>
<td>/speed_poll_timer</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
</tbody>
</table>
Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
<th>Flows</th>
</tr>
</thead>
</table>
| run  | Discrete state | | acceleration' = 0  
       |              | trajectory_step_timer' = 1  
       |              | speed_poll_timer' = 1 |
| up   | Discrete state | | [acceleration' = acceleration-rate  
       |              | trajectory-step-timer' = 1  
       |              | speed_poll_timer' = 1 |
| cruise | Discrete state | | acceleration' = 0  
              |               | trajectory-step-timer' = 1  
              |               | speed_poll_timer' = 1 |
| down | Discrete state | | acceleration' = -acceleration-rate  
       |              | trajectory_step_timer' = 1  
       |              | speed_poll_timer' = 1 |
| start | Discrete state | | acceleration' = 0  
           |               | trajectory_step_timer' = 1  
           |               | speed_poll_timer' = 1 |
| wait  | Discrete state | | acceleration' = 0  
           |               | trajectory_step_timer' = 1  
           |               | speed_poll_timer' = 1 |
| end   | Discrete state | | acceleration' = 0  
           |               | trajectory_step_timer' = 1  
           |               | speed_poll_timer' = 1 |

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>up</td>
<td>trj_accel</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {step < speed_tbl->size}

Action

|----------------|-------------------|------------------------------------------------|

Reset Continuous

<table>
<thead>
<tr>
<th>State</th>
<th>trajectory_step_timer'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Output Event

 Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
<td>cruise</td>
<td>trj_cruise</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

threshold {trajectory_step.timer >= {beta*time_tbl->elements[step]}}
Action
Output Event

Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cruise</td>
<td>down</td>
<td>trj_decel</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
threshold \((\text{trajectory-step}\_\text{timer} \geq (1.0 - \beta) \times \text{time} - \text{tbl-}->\text{elements}\{\text{step}\}))\)

Action
Output Event

Transition 3

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>down</td>
<td>run</td>
<td>trj_next</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard
threshold \((\text{trajectory-step}\_\text{timer} \leq (\text{time} - \text{tbl-}->\text{elements}\{\text{step}\}))\)

Action
Output Event

Transition 4

<table>
<thead>
<tr>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
<td>up</td>
<td>trj_poll</td>
</tr>
</tbody>
</table>

Guard
threshold \((\text{speed-poll}\_\text{timer} \geq \text{speed}\_\text{poll}\_\text{period})\)

Action

| Reset Variable       | speed     | speed + acceleration() - speed\_\text{poll}\_\text{period} |
| Reset Continuous State| speed\_\text{poll}\_\text{timer} | speed\_\text{poll}\_\text{timer} - speed\_\text{poll}\_\text{period} |

Output Event

Transition 5
Guard
threshold \( \text{speed\_poll\_timer} \geq \text{speed\_poll\_period} \)

Action
- \[ \text{Reset Variable} \hspace{1cm} \text{speed} \leftarrow \text{speed} + \text{acceleration()} \times \text{speed\_poll\_period} \]
- \[ \text{Reset Continuous State} \hspace{1cm} \text{speed\_poll\_timer} \leftarrow \text{speed\_poll\_timer} - \text{speed\_poll\_period} \]

Output Event

Transition 6

Guard
threshold \( \text{speed\_poll\_timer} \geq \text{speed\_poll\_period} \)

Action
- \[ \text{Reset Variable} \hspace{1cm} \text{speed} \leftarrow \text{speed} + \text{acceleration()} \times \text{speed\_poll\_period} \]
- \[ \text{Reset Continuous State} \hspace{1cm} \text{speed\_poll\_timer} \leftarrow \text{speed\_poll\_timer} - \text{speed\_poll\_period} \]

Output Event

Transition 7

Guard
\[ \text{boolean} \{1\} \]

Action
```c
FILE* file;

/* remove (previous) trajectory tables */
speed_tbl->removeAll();
time_tbl->removeAll();

/* read the (current) trajectory */
file = fopen("trajectory.ini", "r");
if (file != NULL) {
    float speed, time;
    while (fscanf(file, "%f %f", &speed, &time) == 2) {
        speed_tbl->addNoDupCheck(speed);
        time_tbl->addNoDupCheck(time);
    }
    /* test trajectory feasibility */
    /* ... */
    fclose(file);
} else
    DEM::shutdown();
```

---

## Continuous State Transition 8

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run</td>
<td>end</td>
<td>trj_end</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean (step >= speed_tbl->size)

**Action**

**Output Event**

<table>
<thead>
<tr>
<th>Event</th>
<th>trj_end</th>
</tr>
</thead>
</table>

---

## Continuous State Transition 9

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait</td>
<td>run</td>
<td>trj</td>
<td>proaction</td>
</tr>
</tbody>
</table>

**Guard**

threshold (trajectory_step_timer >= time_tbl->elements[step])

**Action**

<table>
<thead>
<tr>
<th>Reset Variable</th>
<th>step</th>
<th>step + 1</th>
</tr>
</thead>
</table>

**Output Event**
Include Files and other Preprocessor Directives

Header Code

Trajectory();

C++ Code

Trajectory: :Trajectory()
{
}

Class Transmission

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor</td>
<td>Superclass of Transmission</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gear</td>
<td>int</td>
<td>Variable</td>
</tr>
<tr>
<td>gear_ratio</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>prev_gear</td>
<td>int</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td></td>
</tr>
<tr>
<td>busy</td>
<td>Discrete state</td>
</tr>
</tbody>
</table>

Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>busy</td>
<td>read_long</td>
<td></td>
<td>response</td>
</tr>
</tbody>
</table>

Guard

boolean :l

Action
Output Event

---

Transition 1

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>new_data</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {gear == prev-gear}

Action

Output Event

---

Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>idle</td>
<td>new_data</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

boolean {gear != prev-gear}

Action

Output Event

Event [gear-change]

Include Files and other Preprocessor Directives

Header Code

Transmission();

C++ Code

Transmission::Transmission()
{
}

Class UpperController

The upper controller component determines the desired or "synthetic" acceleration for each car in the platoon. It uses inputs from the communicated variables, radar sensor processor, transmission and wheel speed sensor. During gear change, it modifies the controller gains until the gear change is completed.

Superclasses
### Superclass | Description
---|---
Component | Superclass of UpperController

### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega_multiplier</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>GearChangeTime</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>pcll_period</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>gear_des_time</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>/synthetic_accel</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>parameter1</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>/zeta</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>/lead_ratio</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>'omega-m</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>des_follow_dist</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>gear</td>
<td>int</td>
<td>Variable</td>
</tr>
<tr>
<td>omega</td>
<td>double</td>
<td>Variable</td>
</tr>
</tbody>
</table>

### Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comm</td>
<td>CommunicatedVariables*</td>
<td>Link</td>
</tr>
<tr>
<td>rsp</td>
<td>&quot;RadarSensorProcessor&quot;</td>
<td>Link</td>
</tr>
<tr>
<td>transmission</td>
<td>Transmission*</td>
<td>Link</td>
</tr>
<tr>
<td>'wheel_speed_filter</td>
<td>WheelSpeedFilter*</td>
<td>Link</td>
</tr>
</tbody>
</table>

### Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>;synthetic_input</td>
<td>double</td>
<td>function</td>
<td></td>
</tr>
</tbody>
</table>

```java
return(synthetic_accel);
```
### Inputs

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>Link Type</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>double</td>
<td>rsp</td>
<td>RadarSensorProcessor*</td>
<td>range_output()</td>
</tr>
<tr>
<td>gear_des</td>
<td>int</td>
<td>transmission</td>
<td>Transmission*</td>
<td>gear</td>
</tr>
<tr>
<td>preceding_speed</td>
<td>double</td>
<td>comm</td>
<td>CommunicatedVariables*</td>
<td>preceding_speed</td>
</tr>
<tr>
<td>preceding_accel</td>
<td>double</td>
<td>comm</td>
<td>CommunicatedVariables*</td>
<td>preceding_accel</td>
</tr>
<tr>
<td>lead_speed</td>
<td>double</td>
<td>comm</td>
<td>CommunicatedVariables*</td>
<td>lead_speed</td>
</tr>
<tr>
<td>lead_accel</td>
<td>double</td>
<td>comm</td>
<td>CommunicatedVariables*</td>
<td>lead_accel</td>
</tr>
<tr>
<td>speed</td>
<td>double</td>
<td>wheel_speed_filter</td>
<td>WheelSpeedFilter*</td>
<td>wr_flt</td>
</tr>
</tbody>
</table>

### Continuous States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>poll-timer</td>
<td>Continuous state variable</td>
<td></td>
</tr>
</tbody>
</table>

### Discrete States

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Common Computations</th>
<th>Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>Discrete state</td>
<td></td>
<td>poll_timer = 1</td>
</tr>
<tr>
<td>gearTransition</td>
<td>Discrete state</td>
<td></td>
<td>poll_timer = 1</td>
</tr>
<tr>
<td>start</td>
<td>Discrete state</td>
<td></td>
<td>poll_timer = 1</td>
</tr>
</tbody>
</table>

### Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>gearTransition</td>
<td>gear_change</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

boolean (gear-des() > gear)

**Action**

- Reset Variable gear = gear_des()
- Reset Variable gear_des_time = 0
- Reset Variable jomega_multiplier = 0.5

**Output Event**

---

### Transition 1
<table>
<thead>
<tr>
<th>From</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gearTransition</td>
<td>normal</td>
<td>gear_change_complete</td>
<td>proaction</td>
</tr>
</tbody>
</table>

Guard

boolean {gear_des_time > GearChangeTime}

Action

<table>
<thead>
<tr>
<th>Reset Variable</th>
<th>omega_multiplier</th>
</tr>
</thead>
</table>

Output Event

Transition 2

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gearTransition</td>
<td>gearTransition</td>
<td>controller_poll</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

threshold {poll_timer >= poll_period}

Action

<table>
<thead>
<tr>
<th>Reset Continuous State</th>
<th>poll_timer</th>
<th>poll_period</th>
</tr>
</thead>
</table>

/Code wheel_speed_filter -> filtered_wheel_speed();

<table>
<thead>
<tr>
<th>Reset Variable</th>
<th>gear_des_time</th>
<th>gear_des_time + poll_period</th>
</tr>
</thead>
<tbody>
<tr>
<td>omega</td>
<td>omega</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reset Variable</th>
<th>synthetic_accel</th>
<th>lead_ratio * lead_accel + (1 - lead_ratio) * preceding_accel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>omega * (speed() - lead_accel() - (2 * zeta * lead_ratio * parameter1)) - omega * (speed() - preceding_accel)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>omega * (speed() - range()) - des_follow_dist</td>
</tr>
</tbody>
</table>

Output Event

<table>
<thead>
<tr>
<th>Event</th>
<th>controller_poll</th>
</tr>
</thead>
</table>

Transition 3

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>normal</td>
<td>controller_poll</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

Guard

threshold {poll_timer >= poll_period}

Action

<table>
<thead>
<tr>
<th>Reset Continuous State</th>
<th>poll_timer</th>
<th>poll_period</th>
</tr>
</thead>
</table>

file:///C:/Akash/reglayer.html
reglayer

```c
reset_variable omega-m /omega * omega_multiplier
```

```
reset_variable synthetic_accel
```

**Output Event**

```
event controller_poll
```

**Transition 4**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>normal</td>
<td>trj_begin</td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

```c
boolean {1}
```

**Action**

```c
reset_variable parameter1 zeta + sqrt(fabs(zeta*zeta - 1.0));
```

**Output Event**

```
```

**Transition 5**

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>start</td>
<td>controller-poll</td>
<td>proaction</td>
<td></td>
</tr>
</tbody>
</table>

**Guard**

```c
threshold {poll_timer >= poll_period}
```

**Action**

```c
reset_continuous_state poll_timer poll_timer() - poll_period
```

**Output Event**

```
```

```
```

**Include Files and other Preprocessor Directives**

**Header Code**

```c
UpperController();
```

**C++ Code**
UpperController::UpperController()

Class WheelSpeedFilter

Superclasses

<table>
<thead>
<tr>
<th>Superclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>Superclass of WheelSpeedFilter</td>
</tr>
</tbody>
</table>

Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wr_flt</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>wr_dum</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>wr_raw</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>wr</td>
<td>double</td>
<td>Variable</td>
</tr>
<tr>
<td>stay</td>
<td>int</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Links

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>Link</td>
</tr>
<tr>
<td>lower_controller</td>
<td>LowerController*</td>
<td>Link</td>
</tr>
</tbody>
</table>

Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filtered_wheel_speed</td>
<td>double</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```cpp
wr = car-speed();

if (maneuver_id() == 0 || (maneuver_id() == 10))

  if (wr_dum == wr;
      wr = 0;
      stay = 1;
  }

if (maneuver_id() == 1 && stay == 1)

  if (wr == wr_dum)
      wr = 0.0;
  else
      stay = 0;
```
if (time < 0.03)
{
    wr_raw = wr;
    wr_flt = wr;
}
else
{
    if (wr - wr_raw < 0.1)
        wr_raw += 0.1;
    else if (wr - wr_raw < -0.1)
        wr_raw += -0.1;
    else
        wr_raw = wr;

    if (maneuver_id() != 1)
        { wr_flt = 0.3*wr_raw+0.7*wr_flt;
        }
    else
        { wr_flt = 1.0*wr_raw+0.0*wr_flt;
        }
}

if (time < 0.07)
{
    /* control starting routine initialization: begin */
    trans = 1;
    /* control starting routine initialization: end */
    wr_raw = wr;
    wr_flt = wr;
    /*
    */
}

return wr_flt;

**Inputs**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Link</th>
<th>Link Type</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_speed</td>
<td>double</td>
<td>car_sensor</td>
<td>CarSensor*</td>
<td>car_speed</td>
</tr>
<tr>
<td>maneuver_id</td>
<td>int</td>
<td>lower_controller</td>
<td>LowerController*</td>
<td>maneuver_id</td>
</tr>
</tbody>
</table>

**Continuous States**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_filter</td>
<td>Continuous state variable</td>
<td>1</td>
</tr>
</tbody>
</table>

**Discrete States**
### Name | Description | Common Computations | Flows
--- | --- | --- | ---
idle |  |  | `time_filter' = 1`

#### Transition 0

<table>
<thead>
<tr>
<th>From</th>
<th>To</th>
<th>Event</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>idle</td>
<td><code>trj_begin</code></td>
<td>response</td>
<td></td>
</tr>
</tbody>
</table>

#### Guard

```java
boolean {1}
```

#### Action

- Reset Continuous State

#### Output Event

---

### Include Files and other Preprocessor Directives

### Header Code

```c
WheelSpeedFilter();
```

### C++ Code

```cpp
WheelSpeedFilter::WheelSpeedFilter()
{
}
```

---

### Events of the reglayer Application

---

### Alerts of the reglayer Application

---

### System Architecture

### Servers
<table>
<thead>
<tr>
<th>Process Name</th>
<th>lateral</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name</td>
<td>localhost</td>
</tr>
<tr>
<td>Port</td>
<td>5234</td>
</tr>
<tr>
<td>Realtime</td>
<td>enabled</td>
</tr>
<tr>
<td>Logging</td>
<td>enabled</td>
</tr>
<tr>
<td>Databasing</td>
<td>disabled</td>
</tr>
<tr>
<td>Networking</td>
<td>disabled</td>
</tr>
</tbody>
</table>

**Server Initialization Code**

**Components**

DatabaseReader reader

| pclt | omit |
| trigger | omit |
| interface_pid | omit |
| discrete | idle |

Lateralsensor left-sensor

| position/LEFT | reader | reader | discrete | idle |

Lateralsensor center-sensor

| reader | reader | discrete | idle |

Lateralsensor right-sensor

| position/RIGHT | reader | reader | discrete | idle |

Monitor monitor
<table>
<thead>
<tr>
<th>snapshots</th>
<th>omit</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_sensor</td>
<td>car_sensor</td>
</tr>
<tr>
<td>brake_sensor</td>
<td>omit</td>
</tr>
<tr>
<td>engine_sensor</td>
<td>omit</td>
</tr>
<tr>
<td>transmission</td>
<td>omit</td>
</tr>
<tr>
<td>radar</td>
<td>omit</td>
</tr>
<tr>
<td>lateral_controller</td>
<td>lateral_controller</td>
</tr>
<tr>
<td>lower_controller</td>
<td>omit</td>
</tr>
<tr>
<td>upper_controller</td>
<td>omit</td>
</tr>
<tr>
<td>trajectory</td>
<td>omit</td>
</tr>
<tr>
<td>lc_lsp</td>
<td>lc_lsp</td>
</tr>
<tr>
<td>cr_lsp</td>
<td>cr_lsp</td>
</tr>
<tr>
<td>engine_speed_filter</td>
<td>omit</td>
</tr>
<tr>
<td>man_pressure_filter</td>
<td>omit</td>
</tr>
<tr>
<td>left</td>
<td>left_sensor</td>
</tr>
<tr>
<td>center</td>
<td>center_sensor</td>
</tr>
<tr>
<td>right</td>
<td>right_sensor</td>
</tr>
<tr>
<td>radar_sp</td>
<td>omit</td>
</tr>
<tr>
<td>discrete</td>
<td>start</td>
</tr>
</tbody>
</table>

**LateralSensorProcessor lc_lsp**

| x_earth | 0.75 |
| y_earth | -0.06 |
| z_earth | 1 |
| left_x_sample_array | omit |
| left_y_sample_array | omit |
| left_z_sample_array | omit |
| right_x_sample_array | omit |
| right_y_sample_array | omit |
| right_z_sample_array | omit |
| valid_y_output_array | omit |
| left_magnetometer | /left_sensor |
| right_magnetometer | /center_sensor |
| car_sensor | /car_sensor |
| lateral_controller | /lateral_controller |
| discrete | /go |

**LateralSensorProcessor cr-lsp**
| x-earth      | 10.58  |
| y-earth     | 1.06   |
| z-earth     | -1.1   |
| left_x_sample_array | omit |
| left_y_sample_array | omit |
| left_z_sample_array | omit |
| right_x_sample_array | omit |
| right_y_sample_array | omit |
| right_z_sample_array | omit |
| position    | cr     |
| /valid_y_output_array | omit |
| /left_magnetometer | center_sensor |
| /right_magnetometer | right_sensor |
| car_sensor  | car_sensor |
| lateral_controller | lateral_controller |
| discrete    | go     |

**LateralController lateral-controller**

| omega_n     | omit     |
| period      | omit     |
| k_s         | omit     |
| d_s         | omit     |
| k_a         | omit     |
| close_loop  | omit     |
| d_sdot_threshold | omit |
| /control-type | omit |
| /control_to | omit     |
| earth_estimate_flag | omit |
| control_angle | omit     |
| lsp_use     | omit     |
| d_threshold | omit     |
| /car-sensor | car_sensor |
| cr_lsp      | cr_lsp   |
| lc_lsp      | lc_lsp   |
| discrete    | idle     |

**CarSensor car-sensor**

| reader | reader |
| discrete | idle |

**Event Propagation Dependencies**
read-mag reader -> left_sensor
read-mag reader -> center_sensor
read-mag reader . -> right_sensor
read-lat reader -> car_sensor
read-long reader -> car_sensor
new-data left_sensor -> lc_lsp
new-data center_sensor -> cr_lsp
estimate lateral_controller -> lc_lsp
estimate lateral_controller -> cr_lsp
lateral_command lateral_controller -> monitor

Server Shutdown Code

FILE* file;
char filename[66];
time_t tm;

// block SIGUSR1
sigset_t set, oset;
sigemptyset(&set);
sigaddset(&set, SIGUSR1);
sigprocmask(SIG_BLOCK, &set, &oset);

// create an unique file name
time(&tm);
sprintf(filename, "Log %s.dat", ctime(&tm));
for(int i = 0; filename[i]; i++)
  if (filename[i] == '/' || filename[i] == '\n')
    filename[i] = '-';

// write the snapshot file
if ((file = fopen( "lat.log", "w")) == NULL)
  file = stdout;
TheMonitor->Print(file);
fclose(file);

<table>
<thead>
<tr>
<th>Process Name</th>
<th>[longitudinal]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host Name</td>
<td>localhost</td>
</tr>
<tr>
<td>Port</td>
<td>5234</td>
</tr>
<tr>
<td>Realtime</td>
<td>enabled</td>
</tr>
<tr>
<td>Logging</td>
<td>enabled</td>
</tr>
<tr>
<td>Databasing</td>
<td>disabled</td>
</tr>
<tr>
<td>Networking</td>
<td>disabled</td>
</tr>
</tbody>
</table>

Server Initialization Code

Components

DatabaseReader reader
Brakesensor brake

Carsensor car–sensor

CommunicatedVariables comm_var

Enginesensor engine

Radar radar

Transmission transmission

EngineSpeedFilter engine–speed–filter

ManifoldPressureFilter man–pressure–filter
RadarSensorProcessor radar-sp

| /radar | radar |
| /car-sensor | car_sensor |
| /comm | comm_var |

LowerController lower_controller

| /upper_controller | upper_controller |
| engine-speed-filter | engine-speed-filter |
| manifoldsqure-pressure-filter | man_pressure_filter |
| /car-sensor | car_sensor |

| Delphi_brakes | brake |
| /transmission | transmission |
| /discrete | start |

UpperController upper_controller

| /comm | comm_var |
| /radar_sp | radar_sp |
| /transmission | transmission |
| /wheel_speed-filter | wheel_speed_filter |
| /discrete | start |

Monitor monitor
<table>
<thead>
<tr>
<th>snapshots</th>
<th>omit</th>
</tr>
</thead>
<tbody>
<tr>
<td>car_sensor</td>
<td>car_sensor</td>
</tr>
<tr>
<td>brake_sensor</td>
<td>brake</td>
</tr>
<tr>
<td>engine_sensor</td>
<td>engine</td>
</tr>
<tr>
<td>transmission</td>
<td>transmission</td>
</tr>
<tr>
<td>radar</td>
<td>!radar</td>
</tr>
<tr>
<td>!lateral_controller</td>
<td>omit</td>
</tr>
<tr>
<td>/lower_controller</td>
<td>lower_controller</td>
</tr>
<tr>
<td>/upper_controller</td>
<td>upper_controller</td>
</tr>
<tr>
<td>trajectory</td>
<td>trajectory</td>
</tr>
<tr>
<td>fc_lsp</td>
<td>omit</td>
</tr>
<tr>
<td>cr_lsp</td>
<td>omit</td>
</tr>
<tr>
<td>engine_speed_filter</td>
<td>engine_speed_filter</td>
</tr>
<tr>
<td>man_pressure_filter</td>
<td>man_pressure_filter</td>
</tr>
<tr>
<td>left</td>
<td>omit</td>
</tr>
<tr>
<td>center</td>
<td>omit</td>
</tr>
<tr>
<td>right</td>
<td>omit</td>
</tr>
<tr>
<td>radar_sp</td>
<td>radar_sp</td>
</tr>
<tr>
<td>discrete</td>
<td>start</td>
</tr>
</tbody>
</table>

**WheelSpeedFilter wheel-speed-filter**

<table>
<thead>
<tr>
<th>:car-sensor</th>
<th>car_sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>lower_controller</td>
<td>lower_controller</td>
</tr>
<tr>
<td>discrete</td>
<td>idle</td>
</tr>
</tbody>
</table>

**Trajectory trajectory**

| discrete | start |

**Event Propagation Dependencies**
read_long reader -> brake
read_long reader -> engine
read_long reader -> transmission
read_long reader -> radar
read_long reader -> car_sensor
read_long reader -> car_sensor
gear-change transmission -> engine_speed_filter
gear-change transmission -> man_pressure_filter
gear-change transmission -> lower_controller
gear-change transmission -> upper_controller
controllerqoll upper_controller -> lower_controller
controllerqoll upper_controller -> monitor
trj-begin trajectory -> engine_speed_filter
trj-begin trajectory -> manqressure_filter
trj_begin trajectory -> wheel-speed-filter
trj_begin trajectory -> upper_controller
trj_begin trajectory -> lower-controller
trj_end trajectory -> lower-controller

Server Shutdown Code

```c
FILE* file;
char filename[66];
time_t tm;

// blocks SIGUSR1
sigset_t set, oset;
sigemptyset(&set);
sigaddset(&set, SIGUSR1);
sigprocmask(SIG_BLOCK, &set, &oset);

// create an unique file name
time(&tm);
sprintf(filename, "Log %s.dat", ctime(&tm));
for (int i = 0; filename[i]; i++)
    if (filename[i] == '"' || filename[i] == '\n')
        filename[i] = '-';

// write the snapshot file
if ((file = fopen( "long.log", "w")) == NULL)
    file = stdout;
TheMonitor->Print(file);
fclose(file);
```

92 of 94 10/30/98 1:45 PM
Server Initialization Code

Components

Event Propagation Dependencies

Server Shutdown Code

Clients

Application Include Files and Preprocessor Directives

/*
 * Constants and Macros.
 */

#define LC -1
#define cr 1
#define LR 0

#define NUM AIR VALS 14
#define NUMAIR_INDEX 15

#define max(a,b) ((a) > (b) ? (a) : (b))
#define min(a,b) ((a) < (b) ? (a) : (b))

#ifndef PI
#define PI 3.14159265358979L
#endif

/*
 * Global Functions Prototypes.
 */

void UsrlInterruptHandler(int);
void Warning(char*, char”);
void Error(char*, char”);

double Saturate(double, double);

Application C++ Code

/*
 * Global Variables
Monitor* TheMonitor = NULL;

/*
 * Global Functions.
 */

double Saturate(double v, double t)
{
    if (t < 0)
        t = -t;
    return (v > t) ? t : (v < -t) ? -t : v;
}

void UsrlInterruptHandler(int signal)
{
    DEM::isrAlert(new Alert(get_trigger, 0));
    DEM::registerInterruptHandler(signal, UsrlInterruptHandler);
}

void Warning(char* comp, char* message){
    fprintf(stderr, "%s: %s! (ignored)\n", comp, message);
}

void Error(char* comp, char* message){
    fprintf(stderr, "%s: %s! (exit)\n", comp, message);
    DEM::shutdown();
}