Real-Time Data Services for Automotive Applications

Gurulingesh Raravi, Krithi Ramamritham and Neera Sharma

Indian Institute of Technology Bombay

Email: {guru}@it.iitb.ac.in {krithi, neeras}@cse.iitb.ac.in

Abstract

In the recent years, the amount of data that needs to be sensed, stored and processed has increased significantly in an automobile with the rapid growth of electronics and control software applications. The number of microprocessors required to implement these applications have also gone up significantly. These applications deal with safety-enhancing vehicle functions and hence with critical data having stringent requirements on freshness and involving deadline bound computations. Therefore, there is a need for effective real-time data services to handle the data without affecting the performance and safety of these applications and also minimizing the computational resource requirements. Using computational resources efficiently is very important which helps in reducing the additional cost of the car manufacturer incurred due to electronics and control software. These applications generally being distributed across several communicating Electronic Control Units(ECUs) increases the problem complexity. Our work studies one such critical application namely, Adaptive Cruise Control(ACC). We take data+task centric approach in providing real-time data services, designing and implementing this application.

As our contributions we have, (i) identified the data related real-time issues in automotive applications, (ii) identified the data and task characteristics of ACC and shown how to map them on a real-world (robotic) platform, (iii) facilitated two real-time data services for designing ACC and hence similar vehicular critical functions for effective handling of real-time data and to reduce computational resource requirements and (iv) provided the scheduling strategies to meet the timing requirements of the tasks. Experiments demonstrate that the real-time data services are effective in handling the real-time data and reducing the computational resource requirement without compromising the real-time guarantees for safety-enhancing applications.

Introduction

Computer controlled functions in a car are increasing at a rapid rate and today's high end vehicles have up to 80-100 microprocessors implementing and controlling various parts of the functionalities [1]. Some of the features currently controlled by microprocessors include electronic door control, air-conditioning, cruise control, adaptive cruise control, anti-lock braking system, etc. It is predicted that 80% of automobile innovation will be in the area of electronics, controls and software. Automobile industries are aiming towards 360 degrees of sensing around the vehicle leaving no blind zones for the driver. Sophisticated features like collision-avoidance, lane-keeping and *by-wire* systems (steer-by-wire, brake-by-wire) are on the verge of becoming a reality. The number of sensors required to sense the environment for all these sophisticated applications is also increasing at a rapid rate and so is the amount of real-time data that need to be managed.

Features like *Collision Avoidance or Adaptive Cruise Control* are safety enhancers which deal with critical data having stringent timing requirements on freshness and involving deadline bound computations [2]. Therefore, there is a need for effective real-time data services to handle the data to provide real-time guarantees for such features.

The practice of implementing each feature as an independent *black-box* excludes any possibility of sharing the applications among the microprocessors. Increasing the microprocessors in relation to individual features will be a difficult proposition both in terms of cost and integration complexity. Hence, the microprocessors must be effectively used to make *fully electronically controlled vehicle* a reality.

Minimizing the number of microprocessors in an automobile by effectively using them and providing real-time guarantees to the safety-enhancing applications by providing better real-time data services is our goal. Most automotive applications that are safety-enhancing tend to be control applications (a particular target area being that of by-wire controls). These are typically developed using high-level functional modeling languages like Simulink and Stateflow (SL/SF). These models are interconnected networks of computational blocks that exchange information using signals. One of the challenging steps is the refinement of the high-level functional models into computational tasks that can be executed on a distributed platform. This involves the following steps:

1. Decomposition of functional model into tasks
2. Distribution of tasks to different computing nodes in the system
3. Generation of schedules for each of the computing nodes for executing their assigned tasks
4. Assignment of slots on the communication bus to communicating tasks

Three important requirements for such high integrity applications are:

- Correctness (both functional and timing) of the model to be preserved across the refinement from functional models to computational tasks

- Efficient utilization of resources
- Stability and performance analysis of the controller

The focus of this paper is on efficient utilization of resources by providing real-time data services without affecting the performance and safety of such applications. To this end, we discover efficient methods to allow sharing of multiple functions among the processors and at the same time guarantee real-time responses. This would be a critical need, for the cost-effective development of fully electronically controlled vehicle in the future and, in this paper, we have attempted to address this need. The refinement of the high-level functional models into computational tasks and stability analysis of the designed controller is planned in the future work. We have chosen ACC, a safety-enhancing application as the case-study for our approach.

In this paper we have (i) identified the data related issues in automotive applications, (ii) identified the data and task characteristics of ACC and shown how to map them on a real-world (robotic vehicle) platform, (ii) facilitated a real-time approach towards designing the ACC features by the application of mode-change and real-time data repository concepts for reducing the CPU capacity requirement and (iii) provided the scheduling strategies to meet the timing requirements of the tasks. The experiments carried out show that CPU requirements can be effectively reduced without compromising the real-time guaranties for safety-enhancing applications.

The rest of the paper is structured as follows. Section 1 briefs about the real-time data issues in automotive applications. Section 2 briefs about the features of ACC and the current practice followed to develop this application. The issues that need to be addressed to provide computational support for the ACC features are described in Section 3. The mainstay of our approach, namely, a two-level real-time data repository and mode-change design are detailed in Section 4 and 5, respectively. The experimental setup is described in Section 6 and the results of the evaluations from the prototype model is shown under Section 7. Section 8 presents the related work followed by conclusions and future work.

1 Real-Time Data Issues in Automotive Applications

There is a strong need in the automotive industry for better tool support in order to handle the increasing functionality and complexity of computer-controlled software. The amount of data to be managed in a vehicle has increased drastically due to increasing law regulations, diagnosis requirements and complexity of electronics and control software. There are various ways to characterize data items that automotive applications deal with depending on:

- methods of their update
  - *Continuous data items*: They reflect values from an entity, such as a sensor, that changes continuously in the external environment.
  - *Discrete data items*: They are updated at discrete intervals, for example cruise control set speed is updated when driver increases or decreases it using a manual switch.

- source of origin
  - *Base data items*: They are sensor values reflecting an external environment and can be of type either *continuous or discrete*.
  - *Derived data items*: They are actuator values or intermediate values used by several computations, for example, a vehicle's longitudinal speed based on wheel sensor inputs.
- their timing requirements
  - *Safety-critical*: The access and processing of these data items will have timing requirements whose violation might lead to catastrophe, e.g., leading vehicle distance from subject vehicle in ACC.
  - *Non-safety critical*: They will either have no timing requirements or soft timing requirements which when failed to meet will not lead to a catastrophe but meeting them will add some value to the system. For example, logging driver actions and vehicle state by event-data recorder for off-line processing to rate the driver.

Different preventive safety functions are now introduced on road vehicles to assist the driver and to reduce the risk of accidents. The stringent timing requirements on the data items handled by such applications make the problem very complex. Hence the time has come for automotive researchers to apply much of the research carried out in real-time (database) systems to provide efficient data services to effectively handle the data.

In providing real-time data services to guarantee the required timeliness of data and tasks, there are various issues that must be considered. Below is a discussion of some of the concerns that have been the subject of research in real-time systems that automotive researchers are looking into from their own perspective.

Data, task and system characteristics. The need to maintain coherency between the state of the environment and its reflection in the database or data repository leads to the notion of temporal consistency. This arises from the need to keep the controlling system's view of the state of the environment consistent with the actual state of the environment. Since many vehicular functionalities such as lane changing ACC, collision avoidance, etc. are safety enhancers, a real-time solution must maintain not only the logical consistency of data and the tasks handling them, it must also satisfy the task timing properties as well as data temporal consistency. *Task timing constraints* include deadlines, earliest start times and latest start times. Tasks must be scheduled such that these constraints are met. *Data temporal consistency* imposes constraints on how old a data item can be and still be considered valid.

Task scheduling. Task scheduling techniques that consider task and data characteristics have been major part of research in real-time (database) systems. Real-Time Data Base researchers have considered the inherent trade-offs between quality of

data vs. timeliness of processing. If logical consistency is chosen, then there is the possibility that a data item may become old, or that a task may miss a deadline. If, on the other hand, temporal consistency is chosen, the consistency of the data or of the task involved may be compromised. It is often not possible to maintain the logical consistency of a database and the temporal consistency as well. Therefore, there must be a trade-off made to decide which is more important. In dynamic systems, where it is not possible to know and control the real-time performance of the database, it may be necessary to accept levels of service that are lower than optimal. It may also be necessary to trade off the quality of the data in order to meet specified timing constraints.

Tracking dynamically varying data. Dynamic data is characterized by rapid changes and the unpredictability of the changes, which makes it hard to effectively track at predetermined times considering the computational resource limitations in automobiles. The validity interval of such data items is not necessarily fixed during the execution of the system. To have a fixed sampling time as in existing systems requires a worse-case design, leading to over-sampled data and ineffective use of the computational resource like processor.

Timely updates of derived data. The data derivation should be complete before the derived items read set becomes invalid. The derived item needs to be updated only when one or more data items from its read set changes more than a threshold value. For example, the host velocity is derived only when the angular velocity of wheels changes more than the threshold value. In existing systems, the derived values are updated periodically causing unnecessary updates, leading to over sampling and hence inefficient use of the processing power.

Handling of mode-specific data and tasks. Processes controlled by hard real-time computer systems typically exhibit mutually exclusive phases/modes of operation and control [3]. The change in system's mode of operating might change the task sets and the data processed by them, affecting task characteristics such as their timing requirements. A design process that requires all modes of a system to be considered and designed once leads to poor utilization of computational resources, scheduling overhead. For these reasons supporting different modes of operation is of major concern.

Distribution of data and processing. Several applications are obviously distributed in nature to satisfy fault tolerance requirements and localized availability of data. Sensing, computation and actuation tasks are not located on a single computing resource due to the nature of distribution of physical entities that carry out these activities. Rather, they are distributed, and may require that the real-time data be distributed as well. Issues involved with distributing data include data replication, replication consistency, and distributed concurrency control. When real-time requirements are added, these embedded systems become much more complex.

Dynamic Data Dissemination. With the technological advancements in automotive electronics, infotainment and mobile communication technology many data-intensive applications are emerging, e.g., Vehicle-to-Vehicle (V2V) communication like Co-operative Adaptive Cruise Control (CACC), Vehicle-to-Infrastructure (V2I) like real-time navigation, etc. Many of these applications deal with large amount of real-time data sensed to capture the environment status. The access to these data items and their processing might be associated with soft or even hard deadlines depending on the application under consideration. For example, vehicles equipped with CACC communicate messages to each other continuously to avoid probable collision at blind crossings and a real-time traffic navigation system should continuously track the vehicle till it reaches its destination after receiving a request for the best path for its destination. The data involved in these kind of applications will have timing requirements and need to be satisfied to provide better service and to even avoid catastrophes in certain cases. Hence, there is a need for effective data dissemination methods for meeting the timing requirements and to provide accurate information in both V2V and V2I applications.

Sensor Data Fusion. The automobile industry is moving toward a large number of sensors connected through a network to achieve the goal of 360 degree sensing leaving no blind zones for the driver around the host vehicle. Different kind of sensing techniques are under consideration: sensors, radars, vision system and the combination of these. The challenge here is environment recognition and reconstruction so as to be able to alert the driver and/or prevent collisions. The need for such systems exists in many applications involving tasks in which timely delivery of information streams is of critical importance to avoid possible collisions. The obstacle or possible threat recognition techniques requires a great deal of accuracy and reliability since an erroneous application of emergency braking caused by false alarms make it difficult for driver to accept such software controlled safety applications. Automotive researchers wont deny that such actions caused by false alarms might even result in a catastrophe. Hence, there is a need to study accurate and *real-time* fusion techniques to fuse the heterogeneous data coming from different sensors distributed across the vehicle to identify the correct potential threats well in time.

Supervisory Control Module. Computer controlled functions in a car are increasing at a rapid rate. Some of the features currently controlled by microprocessors include electronic door control, air-conditioning, cruise control, adaptive cruise control, anti-lock braking system, etc. Sophisticated features like collision-avoidance, lane-keeping, Curve speed control and *by-wire* systems(steer-by-wire, brake-by-wire) are on the verge of becoming a reality. These features will use the sensor data to compute the desired actions and issue command s to brake, throttle and steering subsystems to carry out them. Its very difficult to (i) avoid conflicts in commands issued by different features (ii) properly distribute commands between these subsystems (iii) coordinate each feature to improve overall vehicle performance and (iv) properly deal with signal noise, latency,

multiple dynamical rates at which different features issue commands. Hence, it is necessary to have a supervisory control module to handle these issues effectively to enhance vehicle performance without affecting the safety requirements.

Fault Tolerant Architecture. Without doubt, fault-tolerance is one of the major challenges that must be met for the design of dependable or safety critical electronic systems, such as *drive-by-wire* applications. These applications are obviously distributed in nature to satisfy fault tolerance requirements and localized availability of data. A reliable architecture (such as Time Triggered Architecture (TTA) [4]) is required to implement such fully fault-tolerant real-time systems. These systems also require a communication service (such as Time Triggered Protocol (TTP) [5] or FlexRay, *www.flexray.com*) that transports messages from one node to another node with nearly constant delay and minimal jitter.

2 Adaptive Cruise Control: An Overview

Adaptive Cruise Control (ACC) is an intelligent feature that automatically adjusts vehicle speed to maintain a *safe distance* from the vehicle moving ahead in the same lane (a.k.a. *leading vehicle*). When there is no vehicle ahead, it tries to maintain the *safe speed* set by the driver. The safe *Distance of Separation (DoS)* that needs to be maintained from the leading vehicle is a function of *host vehicle velocity* and a driver specified parameter, *timegap* (a.k.a. time separation) and it is given by [6]:

$$V_h * T_g + \delta \tag{1}$$

where V_h is the host vehicle speed and T_g is the timegap and δ is the separation distance when the leading vehicle is at standstill. This ensures that safety distance requirement is not violated under the extreme stopping condition where the leading vehicle can come to a halt instantaneously.

ACC, also called next generation Cruise Control, uses forward-looking radar to detect the speed and distance of the leading vehicle. It continuously monitors the host vehicle's speed, status of cruise control, driver's inputs, and leading vehicle's speed and distance. Using all these data, ACC does intelligent computations to determine the desired acceleration of the host vehicle and achieves the safe DoS with the help of throttle and braking system. Actions taken by the controller are displayed on the driver console to keep driver informed about the same.

The various components of the ACC system are shown in Figure 1 and are described below:

Sensors and Fusion Algorithms: Wheel sensors are used to measure the angular velocity of the wheels. The road-tire friction is also estimated using this data. Sensors are also used to know the current position of the throttle and brake pedals. The sensor fusion algorithms help to get the relevant and accurate information from the sensors. For instance, they calculate the vehicle speed by fusing data coming from individual wheel sensors. Rotational speed is converted to longitudinal speed.

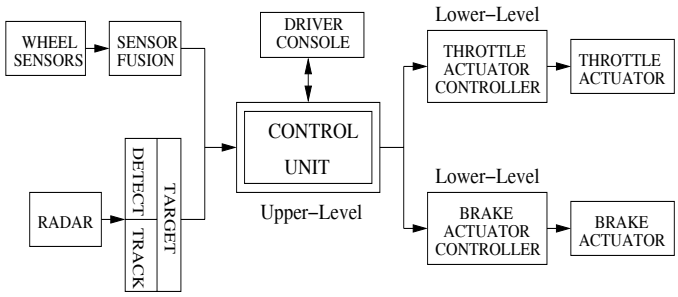

Figure 1. Block diagram of ACC components

Radar and Filter Algorithms: Radar is used to detect the vehicles ahead and measure the relative velocity and distance between the leading vehicle and the host vehicle. The algorithms used here help to identify the most relevant vehicle from the set of vehicles detected by the radar and helps to keep track of it. It helps the controller to compute necessary action by providing the DoS and speed of the relevant leading vehicle.

Driver Console: This unit takes care of the communication to and from the driver. The driver should be aware of the environment and action being taken by the ACC feature.

Controller: The controller maintains up-to-date information about the environment with the help of sensors and radar. It performs necessary control computations, in order to take appropriate actions to maintain the safe distance from the leading vehicle. It has a collection of subtasks that run on an embedded Real-Time Operating Systems (RTOS) like RTLinux, OSEK, etc. These subtasks have information about the environment. The subtasks also interact with other subsystems, and decide actions to be taken, and commands the actuators to perform the required actions. The controller is typically designed in a hierarchical manner consisting of an *upper-level controller* and a *lower-level controller* . The upper-level controller computes the desired acceleration based on the values of the current velocity, acceleration and measured DoS. The lower-level controller in the system is a sub-loop controller that manipulates brake and engine outputs to achieve the desired acceleration/deceleration. At this lower level, an adaptive control algorithm is required that closely monitors the brake and engine control units to achieve the desired acceleration value calculated by upper level controller. The upper-level controller has been the focus point for most of the research work in ACC.

The upper-level controller can operate in one of the two basic modes:

- **Distance Control mode:** When there is a leading vehicle ahead, controller tracks its DoS and speed and maintains the safe DoS as set by the driver.
- **Speed Control mode:** When there is no leading vehicle, controller maintains the safe speed set by the driver.

We exploit the difference between two modes for efficiency of processing requirement.

Actuators: The vehicle speed is controlled by brake and throttle actuators. They are electronically controlled components driven by the commands from the lower level controllers for the requested acceleration computed by the high-level controller.

One of the challenges faced when trying to maintain safe distance is that both the vehicles in the environment are moving. The task of the control system installed in the host vehicle (a.k.a. follower) is to continuously track the leading vehicle (a.k.a. leader) and adapt its velocity accordingly. The tracking rate should be fast enough to have accurate information about the leading vehicle and slow enough to ensure that the system is not overloaded with redundant operations, i.e., operations that do not lead to any change of control actions. The goal is to achieve correct functionality and meet timing requirements, while using the resources optimally. However, there are several factors which can make it very difficult to achieve these requirements. These include the rate at which vehicles are approaching or separating, vehicles cutting in from adjacent lanes to the host cars lane, and weather conditions. To illustrate the timing requirements in the system, consider a host vehicle moving with a velocity of 20 m/s and it detects a vehicle ahead moving with an uniform velocity of 15 m/s, at a distance of 50 m. Let us assume that the driver has set time gap to be maintained as 2s. ACC has to adapt its velocity from 20 m/s to 15 m/s by the time the DoS reduces to 30m. The calculation of desired speed, acceleration and the adjustment of throttle or brake should be completed within this time to maintain the safe distance. This practical scenario shows that the system must be able to work with timing constraints to guarantee that all the data collection and its processing, as well as subsequent decisions and actions happen in time.

In current ACC systems, all the sensors sense the data (leading vehicle speed, distance and host vehicle velocity) *periodically*. Also, these sensors provide the *raw data* which should be processed to convert them to application specific data (or meaningful information) known as *derived data*. The raw items reflect the external environment and can either change continuously e.g., wheel speed sensor, or change at discrete points in time e.g., selecting safe speed. The derived items are derived from raw items and/or other derived items i.e., each derived item *'d'* has a read set denoted *R(d)* that can have members from both the raw and the derived set [7]. For instance, host velocity is derived from the angular velocity of the wheels obtained from four wheel sensors. In this case, angular velocity is raw data item and host velocity is derived data item. The tasks which process raw data to get derived items are also run periodically in the existing systems. The periodicities of all the tasks in the system are set to handle the worst-case scenario.

3 Our Goals and Our Approach

Our goal is to provide real-time data services that address the following:

Effective tracking of dynamically varying data. A data item reflects the status of an entity only for a limited amount of time. When this time expires, the data item is considered to be stale and not valid anymore. This validity interval of a data item is not necessarily fixed during the execution of the system. For instance, consider a leading vehicle that accelerates for

a certain amount of time and then travels with uniform velocity. The validity interval (and hence the sampling period) for the data item *leading distance* will be small when the leading vehicle is accelerating and it can be large when it is moving with a uniform velocity. To have a fixed sampling time as in existing systems requires a worse-case design, leading to over-sampled data and ineffective use of the processor.

Timely updates of derived data. The data derivation should be complete before the derived item's read set becomes invalid. The derived item needs to be updated only when one or more data items from its read set changes more than a threshold value. For example, the host velocity is derived only when the angular velocity of wheels changes more than the threshold value. In existing systems, the derived values are updated periodically causing unnecessary updates. This will again lead to over sampling and hence inefficient use of the processing power.

Handling mode specific task sets. Processes controlled by hard real-time computer systems typically exhibit mutually exclusive phases of operation and control [3]. ACC system performs different tasks while following a close vehicle when compared to following a vehicle which is far away. For instance, we can have a task that determines how much time is left before the safety criteria are violated when the DoS is small, to help the controller to take alternate actions like warning the driver or changing the lane. Similarly, we can have a task that can adjust the driver-set parameters - safe speed and timegap depending on the weather and road conditions when the DoS is large. The task characteristics like periodicity may also vary in different modes. Such changes in the modes of operation affect the task timing requirements, their dependencies, and execution times. In current approaches, all the tasks are executed through out. This leads to poor CPU utilization, scheduling overhead and contradicts the principles of modularization.

Our approach to address the above mentioned issues by providing two real-time data services by exploiting two well known design techniques from real-time system domain: mode-change protocol and real-time update protocols . Both the approaches help to design the application that leads to effective utilization of the CPU capacity by understanding the needs of the system's task and data characteristics. The mode-change protocol is a *task-centric* approach that allows the designer to vary the task sets and characteristics over a period of time. At any point of time the system will have and schedule only the necessary tasks without wasting the CPU capacity on unnecessary tasks. The real-time data-repository model is a *data-centric* approach that decides the task characteristics from the freshness requirements of base and derived data items. Certain periodic tasks from the mode-change approach are made aperiodic to facilitate the *on-demand updates to data items*.

4 Specifics of the Dual Mode System

Processes controlled by hard real-time computer systems typically exhibit mutually exclusive phases/modes of operation and control. A mode change will typically lead to either:

- adding/deleting a task or
- increasing/decreasing the execution time of a task or
- increasing/decreasing the frequency of execution of a task

For instance, ACC performs different tasks while *following* a *close* leading vehicle compared to one that is *far*. In different modes, we can have the sensing tasks execute at different frequencies to deal with dynamically varying data and we can have different set of tasks active in different modes. Hence, we do not need to have all the tasks active at all the times. The design of ACC application with this approach requires answers to the following questions: (i) How many modes, should the design have? (ii) What condition/event should trigger mode change in the system? (iii) When can we switch modes and how much time can mode change operation take? and (iv) How should the tasks be scheduled to meet their timing requirements? We have explained below how these issues are handled while designing ACC application.

(i)Two mutually exclusive phases of operation for ACC.

Non-Critical Mode (NC Mode): In this mode, the environment status does not change rapidly. For instance, when the host vehicle is following a leading vehicle at uniform velocity, the parameters like DoS, host velocity, leading vehicle velocity do not change rapidly. The rate at which the parameters of the system change decides the periodicity of the tasks. The sensor tasks in this mode can execute less frequently.

Safety-Critical Mode (SC Mode): In contrast to NC mode, here the system parameters vary rapidly. For example, consider the case when the leading vehicle is applying maximum brake (say $4.9m/s^2$) and host vehicle under ACC is decelerating (say $2m/s^2$). In this case the DoS between the two vehicles is reducing at a rapid rate. Hence, the radar task which senses this separation should begin to execute more frequently to give the controller fresh data, helping it to determine the right decision at the right time. The system is classified into these two modes of operation based on following parameters:

- Distance between the two vehicles (can take the values FAR, NEAR, FOLLOW).
- Rate of change of Distance - RoD (can take the values Incr-Fast, Incr-Slow, Decr-Fast, Decr-Slow).

Task sets in different modes are shown in Table 4. The tasks in NC-mode namely FrictionT, AdaptT, EDrT perform non-critical operations such as determining road-tire friction, adapting the driver set parameters depending on weather conditions and logging the data for off-line processing, respectively. Similarly, the tasks in SC-mode, namely, TimeLeftT, AdjLaneT and SuggestT carry out critical operations such as determining the time left to avoid collision, sensing adjacent lanes to determine if the lane can be changed to avoid collision and suggesting the driver to take some alternative actions when it becomes difficult for ACC to take control of the situation respectively. The common tasks WheelT and SpeedT are sensor related tasks and RadarT, DistT and LeadVelT are radar tasks. The SpeedT task determines the host vehicle's speed using the data provided by the wheel sensor task, WheelT. The tasks DistT and LeadVelT calculate the leading vehicle DoS and velocity respectively making use of the data provided by the radar task, RadarT. The CruiseT, AccT and BrakeT, ThrottleT are upper-level and lower-level controller tasks and SwitchT is the task that performs mode-change. DriverT monitors the driver inputs and ExceptionT is the task to handle exceptions in the system. All these tasks are *periodic* in nature. The regions FAR, FOLLOW and NEAR are characterized in Figure 2. The radar detection range is given by the dotted curve. $Safe\_Dist$

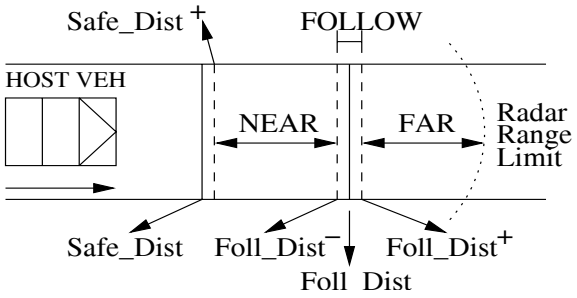

Figure 2. Environment partitioned into regions

is calculated as per the Equation 1 and $Follow\_Dist = Safe\_Dist + \Delta$, where Δ is additional gap for improved safety margins. $Safe\_Dist^{+}$ is given by: $Safe\_Dist + \delta$ which gives enough time for the controller to raise the alarm and for the driver to take over the control. The vehicle will lock-on to a leading vehicle and follow it when the DoS is equal to the $Follow\_Dist$. The regions NEAR and FAR are used to determine the mode of operation (SC vs. NC) as shown in Table 4. The $Foll\_Dist^{+}$ and $Foll\_Dist^{-}$ are used to create the FOLLOW region whose importance is explained when switching condition is described.

Mode	Task Set
NC Only	WeatherT, FrictionT, AdaptT, EDrT
SC only	TimeLeftT, SuggestT, AdjLaneT
Both Modes	WheelT, SpeedT, CruiseT, AccT, RadarT, LeadVelT, DistT, DriverT, BrakeT, ThrottleT, SwitchT, ExceptionT

Table 1. Task sets in different modes of ACC system

(ii) Details of the modes. Intuitively, we should switch from NC to SC mode, the moment the leading vehicle enters the NEAR region. The controller should resume NC mode as soon as the leading vehicle accelerates to the FOLLOW region.

The switching condition is satisfied more frequently leading to rapid mode switching, a phenomenon called *chattering* . Since the controller should have fresh information about the environment, considering only the DoS for switching modes will not yield good results e.g., when the leading vehicle is in the FAR region and decelerating fast. To tackle this case, we also consider the Rate of change of Distance (RoD) as one of the parameters while deciding mode switch condition. Finally, to avoid the chattering phenomenon, FOLLOW region is used as hysteresis. The decision on the current mode of the system is taken based on two parameters DoS and the RoD and change in their values triggers the mode-change phenomenon. Table 4 shows all the possible situations for the system to operate in each of the modes.

LeadDist	**RoD**	**mode**
FAR	Decr-Fast	SC
FAR	Incr-Fast	NC
FAR	Decr-Slow	NC
FAR	Incr-Slow	NC
NEAR	—	SC
FOLLOW	—	Retain Mode

Table 2. The state of the system in different modes

The system should satisfy the following conditions depending on its mode of operation.

In Safety-Critical mode:

$$(Safe\_Dist^{+} < Curr\_Dist \leq Follow\_Dist^{-})||$$

$$(Follow\_Dist^{+} < Curr\_Dist \leq Radar\_Dist \&\& RoD == Decr\_Fast)||$$

$$(Follow\_Dist < Curr\_Dist \leq Follow\_Dist^{+} \&\& Curr\_Mode == SC)$$

In Non Safety-Critical mode:

$$(Follow\_Dist^{+} < Curr\_Dist \leq Radar\_Dist \&\& RoD \neq Decr\_Fast)||$$

$$(Follow\_Dist < Curr\_Dist \leq Follow\_Dist^{+} \&\& Curr\_Mode == NC)$$

(iii) Switching modes. A mode change request is generated from either the radar task, when DoS parameter satisfies the condition for mode change or another periodic task, which tracks the RoD to satisfy the condition. Once the mode-switch condition is satisfied, mode change needs to be carried out by a task. This process involves deleting the tasks in the current mode, allocating stack to the new tasks and creating the new tasks ensuring schedulability at any given point of time throughout this process. The periodic task *SwitchT* performs these operations in our implementation. The mode-switch operation is initiated upon the termination of the task that makes the mode switch condition true.

Mode change delay, defined as the delay between the time at which mode-change is requested and the time at which all the tasks that need to be deleted have been deleted and their allocated processor capacity becomes available, should be small.

As we initiate the mode-change operation once the current task finishes its execution, the delay in reclaiming the processor capacity is bounded by the period of low priority task in the system. In Rate Monotonic Algorithm (RMA) , this is given by longest period [8].

(iv) Scheduling tasks in different modes. The modes in a system are characterized by number of active tasks. The tasks and their characteristics (periodicity, WCET) are known a priori. Hence, static priority scheduling is the obvious choice for system with different modes. Rate Monotonic Algorithm is used to schedule the tasks in our implementation.

5 Specifics of the Real-Time Data Repository

In this section we describe our real-time data repository model, which consists of two levels of data store. The concept of two levels of data store is motivated by: (a) the presence of raw and derived data items and (b) the fact that a small change in raw data i.e. sensor values might not affect the action of the ACC controller. As we discussed in Section 4, the wheel sensor task *WheelT* periodically senses the angular velocity of the wheels which is used by another periodic task *SpeedT* to determine the linear velocity of the vehicle. We realize that the periodic execution of the task *SpeedT* may be skipped in some cases. For instance, in a case where the host vehicle is following a leading vehicle moving with a uniform velocity maintaining the safe DoS, the host vehicle will travel with the same velocity until the leading vehicle starts accelerating or decelerating. In such cases we can choose to skip the execution of *speedT*, until we observe a considerable change in the value sensed by the task *WheelT*. Similarly, the tasks *DistT* and *LeadVelT*, which derive the DoS and leading vehicle velocity from the raw data provided by the task *RadarT* need not execute periodically. These tasks should execute only when significant changes are observed in data collected by *RadarT*. This approach of avoiding the unnecessary updates in the system would result in a best utilization of CPU capacity. Our real-time data repository model is a data centric approach, in which we explore the possibility of making some of the periodic tasks in our task set aperiodic, to reduce the number of unnecessary updates in the system. In this approach we describe the temporal characteristics of the data items in the system, which help us decide the temporal characteristics of the updating tasks associated with the data item.

We use two levels of data store in our approach: *Environment Data Repository (EDR)* and *Derived Data Repository (DDR)* as shown in Figure 2. The upper-level and lower-level controller in the Figure 2 corresponds to the controllers shown in Figure 1. EDR is an active entity, storing the data pertaining to the controlled ACC system. EDR contains base data items and the procedures for data derivation task. If smart sensors are employed in the vehicle, the process of data derivation can be done at these sensors themselves, causing some of the procedures implemented in EDR to be omitted. Collected sensor values might contain some noise. Techniques such as in-network aggregation and averaging may be implemented in EDR to filter noisy values. The second repository DDR in the model, acts as a global database for the system. ACC controller communicates with DDR to get the current values of vehicle parameters.

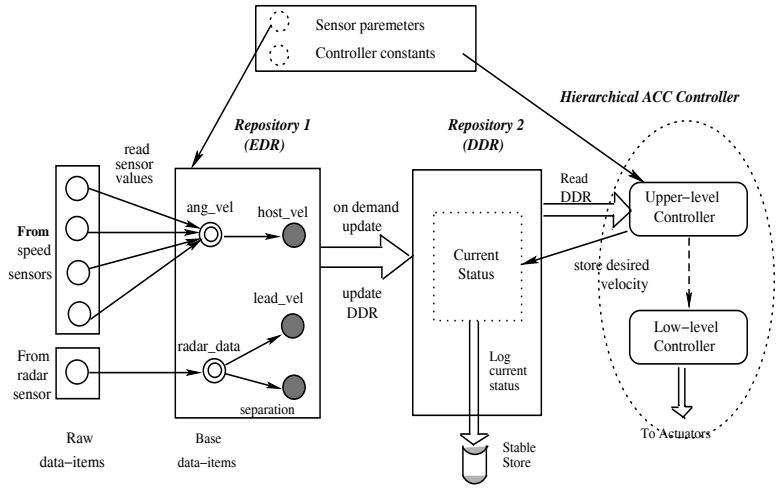

Figure 3. Real-time data repository for Adaptive Cruise Control

5.0.1 Task Models

As shown in Figure 2, circular nodes represent data items and arrows acting on these nodes represent tasks operating on data items. Effectively, each data item is associated with a value and a task manipulating the value. We identify the following classes of tasks in the system:

- Sensor Reading (SR) Tasks: Data collected from sensors are *temporally consistent*. They are valid if:

$$CurrentTime - TS(d_i) \leq VI(d_i) \tag{2}$$

  where, $TS(d_i)$ and $VI(d_i)$ denote time stamp and validity interval of data object d_i, respectively. The tasks *WheelT* and *RadarT* read the values from wheel sensors and radar periodically and update EDR. These tasks have a known period T_i, a computation time C_i, and a relative deadline D_i equal to its period. The period of these tasks is determined by the validity interval of data objects associated with them.

- On-demand Update (OD) Tasks: The derived data items are calculated and updated in DDR only when one of the data items from the read set, $R(d)$ changes more than the threshold value, δ_{th}. The tasks $SpeedT$, $DistT$ and $LeadVelT$ fall under this category. The validity condition of data item d_i in DDR can be written as:

$$|v_i(inEDR) - v_i(inDDR)| \leq \delta_{th} \tag{3}$$

  where v_i is the value of the data item d_i and δ_{th} is the threshold value which determines the frequency at which on-demand update tasks in the system are invoked.

- Lower Level Controller (LC) Tasks: They give command to the mechanical system of the vehicle to achieve the desired velocity. These tasks are performed by the hardware of our prototype model.
- Other Tasks: They are the lower priority tasks such as WeatherT and FrictionT for monitoring weather and road conditions. These tasks are executed only when there are no critical tasks pending in the system.

5.0.2 Scheduling of On-Demand tasks

The second repository update task and upper level controller tasks are modeled as on-demand aperiodic tasks. A well known conceptual framework to guarantee the service on aperiodic tasks is the *aperiodic server technique*. This models the behavior of aperiodic tasks by reserving a share of processor bandwidth for each of the aperiodic tasks and associates a controlling server thread to maintain each reserved bandwidth. We use Constant Bandwidth Server (CBS) [9] for scheduling aperiodic tasks, since it permits hard real-time guarantees. A CBS ($S = (C_s, T_s, B_s)$) is characterized by three parameters: maximum capacity (C_s), period (T_s) and bandwidth ($B_s = C_s/T_s$). Since we do not know the execution time (bandwidth requirement) of on-demand aperiodic (e.g., SpeedT) task a priori, we implement CBS which adapts bandwidth dynamically using feedback control: We start with a high value for bandwidth, which reduces at each step and finally stabilizes to a constant value. The stabilized value for bandwidth is kept more than the actual required value. At each step we calculate CBS scheduling error:

$$\epsilon = CBS\_Deadline - Task\_Deadline$$

where, $CBS\_Deadline\ d_i$ is set to d_{i-1}+ *server period*. $Task\_Deadline$ is calculated using task rate. If R_i is the task rate and $a_{i,j}$ is the arrival time of j^{th} instance of i^{th} aperiodic task, task deadline will be: $1/R_i + a_{i,j}$. At each step we adjust CBS bandwidth by changing server capacity C_s using the following feedback control law [10]:

$$\delta C_s = \frac{\epsilon}{T_s} * C_s \tag{4}$$

When ϵ is 0, the aperiodic task is guaranteed to respect its deadline. The objective of the system is to maintain ϵ near 0.

6 Robotic Vehicle Control: Experimental Setup

This section describes the implementation details of both hardware and software used to demonstrate the concept. Since our aim of this implementation is to prove the concept, we have implemented the essential parts of the system, abstracting out some real world factors. However, the implementation is scalable so that these factors can be accommodated as and when required. The robot on which ACC was implemented is shown in Figure 4 and had the following features:

- Obstacle detection (i.e., radar) range: $2m$.

- Maximum speed: $0.50cm/s$.
- Maintains path through white-line following.
- Closed-loop controller(s).

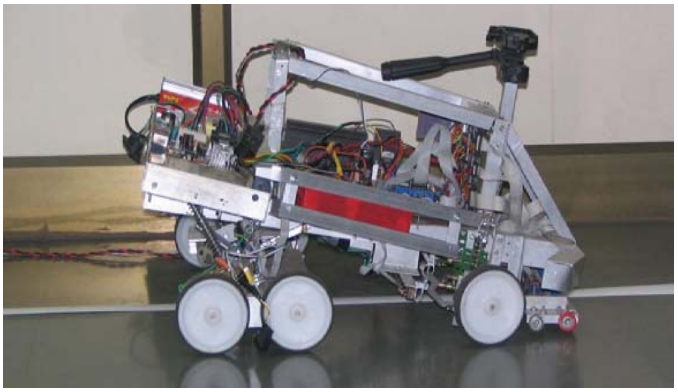

Figure 4. Experimental Robotic Vehicle Platform

The leading vehicle was simulated by manually moving an obstacle ahead of the robot and making the robot maintain the safe DoS from it. The experiments were performed using this manually controlled obstacle, maintaining constant DoS simulates the scenario of leading vehicle moving with uniform velocity. The case where leading vehicle varies its velocity over a period of time is simulated by increasing and/or decreasing the DoS. The robot was controlled by a PC running RTLinux . The PC performed all the computations and issued commands to the robot through the printer port. The standard printer port is used as the communication medium. The software was written using C on RTLinux platform. The logging functions were also written in C. The controller polled the data values from different sensors and performed computations to decide the action to be taken and drove the actuators to carry out the appropriate actions. The sensors were used to measure the host vehicle speed and leading vehicle distance. The task structure and data items in real-time repository are shown in Figure 5. The data items are represented by rectangular boxes and the tasks by circular boxes. The tasks updating EDR are periodic and those updating DDR are aperiodic in nature, ensuring *on-demand updates*.

The Constant Bandwidth Server (CBS) is used to guarantee task completion before its deadline and we used CBS patch for RTLinux developed by by G. Lipari et. al. [11] for this purpose. The controller task (AccT) that reads the data items from DDR and computes the desired speed of the host vehicle also executes aperiodically for the above stated reason. In SC mode and NC mode, the tasks and data items listed in Section 4 exist (common tasks) along with an additional task $SwitchT$ to carry out mode change operation. The tasks shown in Table 4 that exist in one of the two modes are implemented as dummy tasks with different periodicities. In the SC mode, the common tasks execute at double the speed as compared to NC mode.

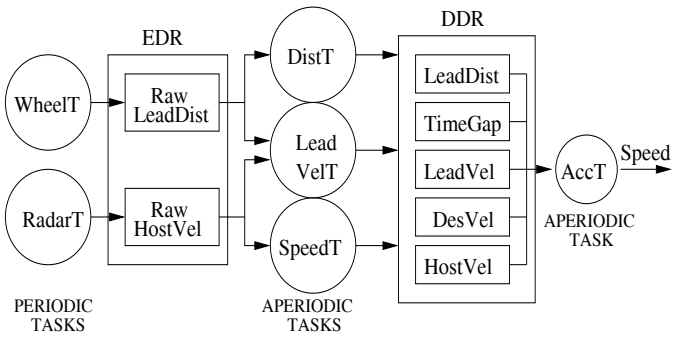

Figure 5. Task and data structure in real-time repository implementation

7 Results and Observations

Experiments were conducted in three stages. Initial experiments were meant to observe whether the robot was achieving the desired functionalities of the ACC system. This basic implementation did not incorporate either the mode-change or the real-time repository concepts. Secondly, experiments were conducted to test the behavior of the two-level real-time data repository design. These experiments were used to observe the reduction in the number of update tasks executed. The second level tasks were executed only when necessary, as opposed to periodic execution in the initial design without the two-level repository. This saves CPU processing power which can be effectively utilized to execute other lower priority tasks in the system, if any. Finally, experiments were conducted to study the system behavior under mode-change design.

7.1 Basic Experiments

Four different tests were carried out to observe the system behavior by logging the host velocity and DoS, with time. Figure 6 shows the velocity response of the host vehicle when there was no leading vehicle ahead and safe speed was set to 25 cm/s. We can observe that the controller is able to maintain the velocity in the range: 22 ± 3cm/s.

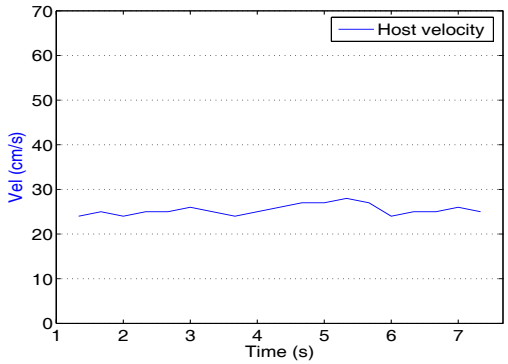

Figure 6. Host Velocity: Cruise Control case, set velocity = 25cm/s

The velocity response of the host vehicle and time separation maintained are shown in Figure 7 and 8 respectively when the leading vehicle is at a constant DoS (i.e., moving at a uniform velocity). The variation in the velocity response of the

host vehicle in the time window 4.5-5.0s (in Figure 7) is due to the variation in DoS in the time interval 4.0-4.5s. This delay of about 0.5s in the velocity response of the host vehicle to the changes in the environment can be attributed to its physical characteristics and control loop delay.

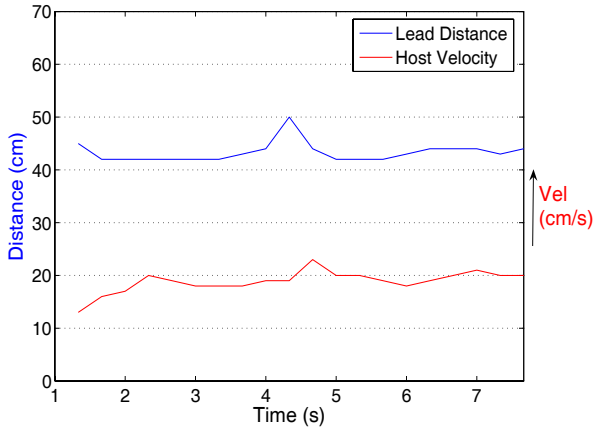

Figure 7. Host Velocity: Leading vehicle with uniform velocity

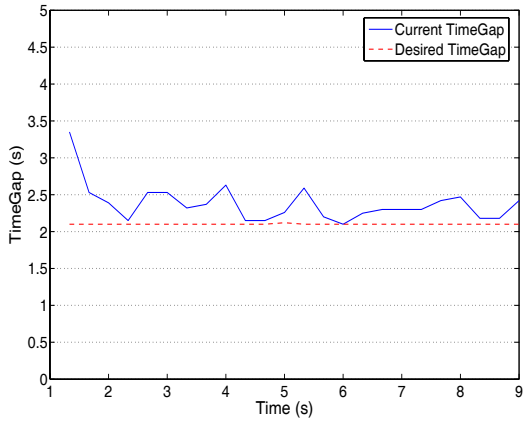

Figure 8. Timegap: Leading vehicle with uniform velocity

The next experiment simulated a scenario, where leading vehicle exhibits varying velocity. The DoS was increased gradually between time interval 1-6s, kept constant between 6-8s, then gradually decreased from 8-12s and again kept constant between 12-14s. Figure 9 shows the velocity response and Figure 10 shows the time separation maintained in this case.

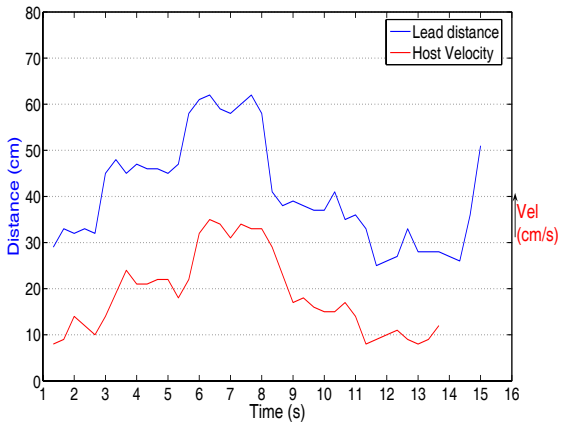

Figure 9. Host Velocity: Leading vehicle with varying velocity

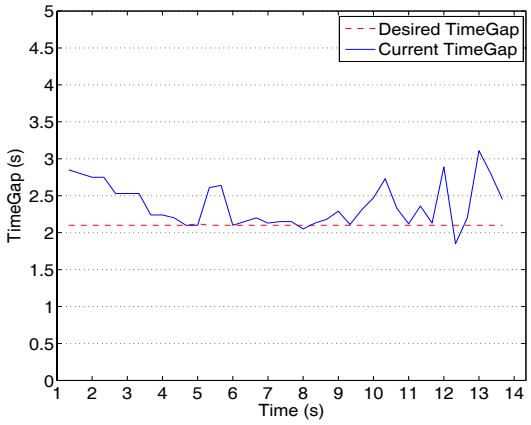

Figure 10. Timegap: Leading vehicle with varying velocity

7.2 Two-level Real-Time Data Repository Experiments

The second set of experiments tested the two-level real-time data repository design. These experiments captured the system behavior when the data update tasks of the two-level repository design are executed only when necessary.

The velocity response of the host vehicle with the use of the real-time repository when the leading vehicle is moving at a constant distance is shown in Figure 11. Invocations of the task updating the DoS (DistT) in DDR are observed (stems in the graph indicate the time of invocation). This task is invoked only when the DoS changes by a preset threshold value, 5cm. The DoS is calculated once at the beginning of the experiment and twice in the time interval 10-12s when the distance is decreased by 5cm and then set to the original value. The task deriving the DoS and updating the DDR executes only when necessary. Similar results can be observed in Figure 12 where the leading distance increases in time intervals 0-5s and 6-7s, kept constant between 5-6s and 7-8s and then gradually reduced from 8s to 12s. We can observe from the graph that during the time interval 5-6s and 7-8s when the DoS was constant, the on-demand task is not invoked and during other time intervals, it is invoked whenever the DoS changes by threshold value. Table 7.2 shows the DistT task's invocation in both the models (with and without two-level real-time repository model) over different time windows. The periodicity of the task was set to 0.3s in the experiments discussed in Section 7.1. We can observe that this approach requires less processing power compared to the conventional approach.

		DistT Task Invocation	
Lead Dist	**Time Window**	**with 2-level**	**without 2-level**
Const	0 to 12	3	40
Incr-Decr	0 to 12	16	40

Table 3. Number of invocations of lead-dist updating task in the two models

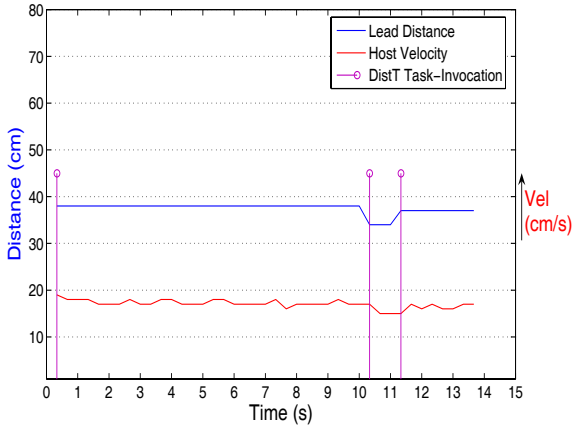

Figure 11. Host Velocity: Constant leading distance case - *on-demand updates*

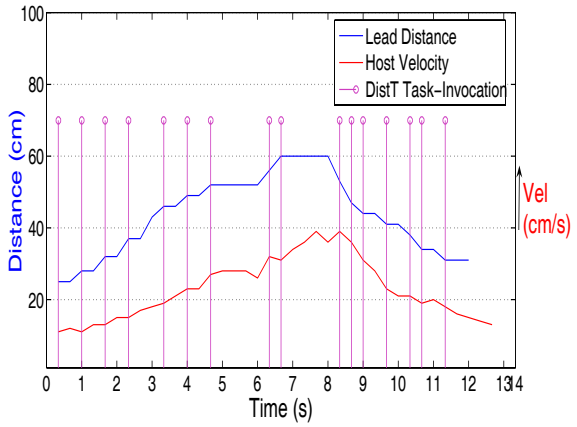

Figure 12. Host Velocity: Leading vehicle with varying velocity case - *on-demand updates*

7.3 Mode-Change Experiments

Finally, we study the effect of mode-change delay on data freshness and safety of the system in terms of maintaining the desired time separation. The velocity response of the host vehicle and time separation maintained with the mode-change implementation, when leading vehicle is moving with varying velocity are shown in Figure 13 and 14 respectively. The DoS is kept as the criterion for changing the mode i.e., if *leading distance* $\leq$ *65cm* enter SC mode else NC mode. The frequency of execution of tasks in NC mode is half that of the frequency in SC mode: the periodicity of the tasks was set to 0.3s in SC mode and 0.6s in NC mode. We can observe from the graph that the system is operating in NC-mode between time interval 8-12s, where the distance is gradually decreasing from 90 to 65cm and then it enters SC-mode at time 13s and again switches back to NC-mode at 19s. The desired timegap is violated couple of times in Figure 14 which can be attributed to the inertia of the vehicle and mode-change delay. This suggests that a conservative approach should be taken while deciding the safe DoS or time separation by taking these two factors into account. In this approach too, we can observe that half the CPU capacity is saved when the system operates in NC-mode compared to conventional approach.

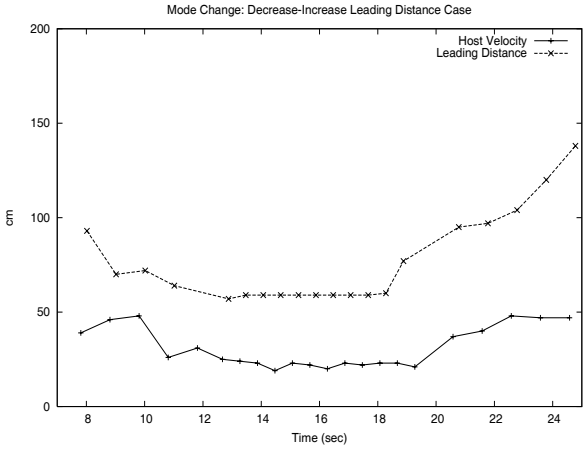

Figure 13. Host Velocity Response: Mode-change case, Leading vehicle with varying velocity

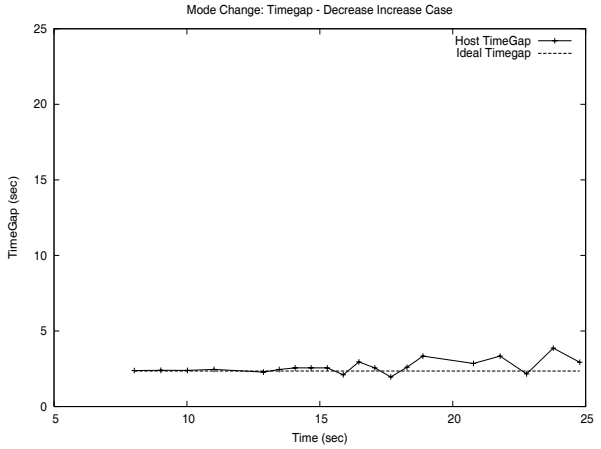

Figure 14. Timegap Response: Mode-change case, Timegap in Leading vehicle with varying velocity

8 Related Work

Adaptive Cruise Control is a well studied research topic in control systems. The design techniques and simulation results for ACC equipped vehicles are reported in [12, 13]. The paper discusses a design of an autonomous intelligent cruise control and a constant time-headway rule for a worst-case stopping scenario. Prior work discusses control aspects of the application not the real-time aspects. A datacentric approach to the architectural design of performance critical vehicular applications has been examined before in [14] and [15]. In particular, Gustafsson and Hansson [14] address the issues in the design and

implementation of an active realtime database system for EECU (Electronic Engine Control Unit) software. A set of ondemand updating algorithms: OnDemand Depth First Traversal (ODDFT) and OnDemand Top Bottom (ODTB) are presented in [7]. These algorithms optimistically skip unnecessary updates and hence provide increased performance. Methods for the specification and runtime treatment of mode changes are discussed in [3]. An approach to handle mode changes in the time triggered, strictly periodic, pre run-time scheduled system MARS, is studied in [16]. Sha et al [8] attempt to address the problem of analyzing a priori a single processor system, scheduled according to the rate monotonic scheduling policy, with tasks able to lock and unlock semaphores according to the priority ceiling protocol. The mode change protocol they propose is one, where tasks are either deleted or added across a mode change.

9 Conclusions and Further Work

In this paper, we have discussed the real-time data issues involved in developing automotive applications. We have also presented the issues involved in developing real-time support for ACC and shown the effectiveness of two real-time data services in handling the data and optimally utilizing computational resources. By using a two-level real-time data repository model to update the derived data only when necessary and designing ACC with different modes, each containing different task sets with different characteristics, we have utilized processor capacity effectively, compared to existing approaches. We have shown that these approaches can enable system designers and developers to build a safe and predictable system making effective use of the CPU capacity even if there are demanding validity requirements to be satisfied by the applications.

We are working on possible extensions to the research described here. First, more analysis of the system design is being carried out with different conditions for mode switching, periodicity of the tasks in different modes and conditions for triggering second level update tasks. Second, application needs are being mapped to a distributed platform (as it is the case in the real-world) and the real-time communication issues between the processors are being studied using FlexRay and CAN like communication infrastructures. Third, the impact of mode-change and real-time data repository design concepts on the controller's stability and performance from control theoretical perspective is being studied. Fourth, we are also looking at other real-time data services to address issues described in this paper such as Sensor data fusion, efficient dissemination of data in V2V and V2I communication applications, etc.

10 Acknowledgements

Our special thanks are due to Sachitanand Malewar for designing and realizing the required hardware.

References

[1] H. Kopetz, "Automotive electronics," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, June 1992, pp. 132–140.

[2] R. Gurulingesh, N. Sharma, K. Ramamritham, and S. Malewar, "Efficient real-time support for automotive applications: A case study," *IEEE Conference on Real-Time Computing Systems and Applications*, Aug 2006.

[3] G. Fohler, "Flexibility in statically scheduled hard real-time systems," Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 1994.

[4] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Oct. 2001.

[5] H. Kopetz and G. Grünsteidl, "Ttp - a protocol for fault-tolerant real-time systems." *IEEE Computer*, vol. 27, no. 1, pp. 14–23, 1994.

[6] J. Zhou and H. Peng, "Range policy of adaptive cruise control vehicles for improved flow stability and string stability," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, June 2005, pp. 229–237.

[7] T. Gustafsson and J. Hansson, "Dynamic on-demand updating of data in real-time database systems," in *Proceedings of the ACM Symposium on Applied computing*, NY, USA, 2004, pp. 846–853.

[8] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," Amherst, MA, USA, Tech. Rep., 1989.

[9] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, vol. 19. Washington, DC, USA: IEEE Computer Society, 2-4 Dec. 1998, pp. 4–13.

[10] L. Abeni and G. Buttazo, "Adaptive bandwidth reservation for multimedia computing," *Sixth International Conference on Real-Time Computing Systems and Applications*, pp. 70 – 77, December 1999.

[11] L. P. Giuseppe Lipari and L. Marzario, "Wp4 resource management components," OCERA Consortium, Tech. Rep., April 2003.

[12] P. A. Ioannou and C.-C. Chien, "Autonomous intelligent cruise control," in *IEEE Trans. on Vehicular Technology*, June 1993, pp. 657–672.

[13] U. Palmquist, "Intelligent cruise control and roadside information," *IEEE Micro*, pp. 20–28, 1993.

[14] T. Gustafsson and J. Hansson, “Data management in real-time systems: a case of on-demand updates in vehicle control systems.” in *Proceedings of the 10th IEEE RTAS*, 2004, pp. 182–191.

[15] D. Nyström, A. Tesanovic, C. Norström, J. Hansson, and N.-E. Bånkestad, “Data management issues in vehicle control systems: A case study.” in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, 2002, pp. 249–256.

[16] G. Fohler, “Realizing changes of operational modes with pre run-time scheduled hard real-time systems,” in *Proceedings of the Second International Workshop on Responsive Computer Systems*. Saitama, Japan: Springer Verlag, October 1992.

[17] K. Ramamritham, “Real-time databases,” *Distributed and Parallel Databases*, vol. 1, no. 2, pp. 199–226, 1993.

[18] K. Ramamritham, S. H. Son, and L. C. DiPippo, “Real-time databases and data services.” *Real-Time Systems*, vol. 28, no. 2-3, pp. 179–215, 2004.

[19] B. Adelberg, H. Garcia-Molina, and B. Kao, “Applying update streams in a soft real-time database system,” in *SIGMOD ’95: Proceedings of the international conference on Management of data*. New York, NY, USA: ACM Press, 1995, pp. 245–256.

[20] B. Kao, K.-Y. Lam, B. Adelberg, R. Cheng, and T. Lee, “Maintaining temporal consistency of discrete objects in soft real-time database systems,” *IEEE Trans. Comput.*, vol. 52, no. 3, pp. 373–389, 2003.

Index