Merge-by-Wire: Algorithms and System Support

Gurulingesh Raravi, Vipul Shingde, Ashish Gudhe, Krithi Ramamritham
Indian Institute of Technology Bombay

Abstract

Automakers are trying to make vehicles more intelligent and safe by embedding processors which can be used to implement by-wire applications for taking smart decisions on the road or assisting the driver in doing the same. Given this proliferation, there is a need to minimize the computing power required without affecting the performance and safety of the applications. The latter is especially important since these by-wire applications are distributed and real-time in nature and hence deal with critical data and involve deadline bound computations on data gathered from the environment. These applications have stringent requirements on the freshness of data items and completion time of the tasks. Our work studies one such safety-related application namely, Automatic Merge Control (AMC) which ensures safe vehicle maneuver in the region where two or more roads intersect.

As our contributions, we (i) propose two merge algorithms for AMC: Head of the Lane (HoL) and All Feasible Sequences (AFS) (ii) demonstrate how DSRC-based wireless communication protocol can be leveraged for the development of AMC (iii) present a real-time approach towards designing AMC by integrating mode-change and real-time repository concepts for reducing the processing power requirements and (iv) identify task characteristics of AMC and provide a scheduling strategy to meet their timing requirements. Simulations demonstrate the advantages of using our approach for constructing merge-by-wire systems.

1. Introduction

It is believed that automation of vehicles will improve safety, reduce accidents, increase traffic flow, and enhance comfort for drivers. Automakers are trying to achieve automation by embedding more processors, known as Electronic Control Units (ECUs) and sensors into vehicles which help to enhance their intelligence. As a result, computer-controlled functions in modern cars have increased 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]. The features currently governed by ECS applications range from a simple door locking module to *adaptive cruise control*, *anti-lock braking systems* and *hybrid power-train management*. Sophisticated features like *collision-avoidance systems* and *by-wire* systems are on the verge of becoming a reality.

In the past few years, there is an endeavor to incorporate the safety-related electronic systems in vehicles that are directly responsible for active and passive vehicle safety. Typically, these systems are distributed and real-time in nature. However, the current practice of implementing each application as an independent *black-box* excludes any possibility of sharing them among the microprocessors. Increasing the microprocessors in relation to individual applications will be a difficult proposition both in terms of cost and integration complexity. Hence, the microprocessors must be effectively used to progress in this area to make fully electronically controlled vehicle a reality. Also, all these safety-related systems have stringent timing requirements apart from having specific functional requirements. Hence, it is necessary to provide real-time guarantees for such systems. With a larger goal of understanding the requirements of and designing by-wire applications and hence a fully autonomous vehicle, we studied one such safety related application, namely Automatic Merge Control (AMC), which ensures safe vehicle maneuvering in the region where two or more roads intersect. This application is distributed in nature, requiring multiple vehicles to communicate and take a collective decision to ensure safety of the whole system, i.e., all the relevant vehicles on the road (a.k.a. *global safety*).

1.1. Our Goals and Our Approach

Our goal is to develop algorithms and provide system support for the AMC application to:

1. Ensure safe maneuvering of vehicles at intersections.
2. Achieve an optimization goal such as minimizing the average Driving-Time-To-Intersection (DTTI).
3. Maintain a safe separation distance between two successive vehicles on each lane.
4. Efficiently utilize both computational and communicational resources.
5. Take away autonomy of vehicles only when necessary.
6. Provide real-time support so that all the tasks meet their deadlines to guarantee safety of the system.

To this end, we (i) propose two merge algorithms and (ii) provide system support for AMC.

As part of the AMC system support, we (i) present high-level modular design in which AMC operates in different states, (ii) present system design for efficient management of task and data items and (iii) show how a DSRC-based wireless communication protocol can be leveraged for the development of AMC for inter-vehicle communication.

The bird's eye view of the entire paper is shown in Figure 1 with section numbers in square brackets.

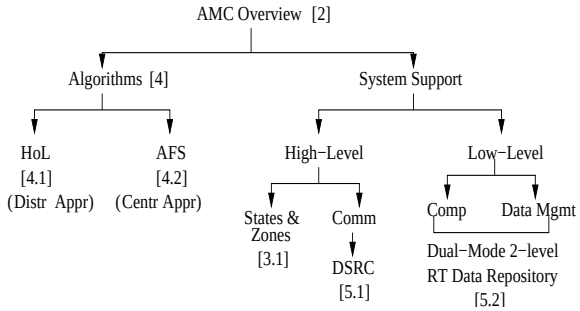

Figure 1. Overview of our contributions

The rest of this paper is organized as follows. Section 2 gives an overview of the AMC application. The overview of our approach and design are discussed in Section 3. The AMC algorithms and the system design are discussed in Section 4 and 5 respectively. The experiments that are carried out and their results are discussed in Section 6 and 7 respectively. Section 8 presents the related work followed by conclusions and future work. In this paper, terms *profile* and *behavior* of a vehicle are used interchangeably. Also, terms *lane* and *road* are used interchangeably since as a first step, algorithms assume that roads are of single lane which we intend to relax in further iterations of the algorithms.

2. Automatic Merge Control System: An Overview

In this section, we briefly describe the Automatic Merge Control application.

The Automatic Merge Control (AMC) is a distributed intelligent control system that ensures safe vehicle maneuver at road intersections. The system ensures that no two vehicles coming from different roads collide or interfere at the intersection region. It ensures that the time taken by any two vehicles to cross the intersection region is separated by at least δ (which depends on the length of the intersection region and permitted velocity of vehicles while at the intersection), by giving commands to adapt their velocities appropriately. This system involves following sub-problems: (**S1**) maintaining safe separation distance between vehicles (**S2**) ensuring safe maneuvering of vehicles at intersection region and hence determining the Merge Sequence (MS) i.e., the order in which vehicles cross the intersection region and (**S3**) minimizing the time taken by vehicles to cross the merge region, for example, minimizing the average *Driving Time To Intersection (DTTI)*. *DTTI* of a vehicle is the time it takes from AoI boundary to cross the intersection region. Maintaining a safe separation distance between vehicles not only ensures that collision does not occur in the area before and after the merge region but also simplifies the task of achieving safe maneuvering. To ensure safe maneuvering, the AMC takes over the autonomy of vehicles as described in the following sections. The AMC taking over the autonomy of vehicles is a simplification of the system which we intend to relax in the further iterations of our algorithms. Whereas the AMC system can be applied to *n-road* merge scenarios, for ease of exposition, the AMC system for *2-road* intersection is discussed in the rest of the paper.

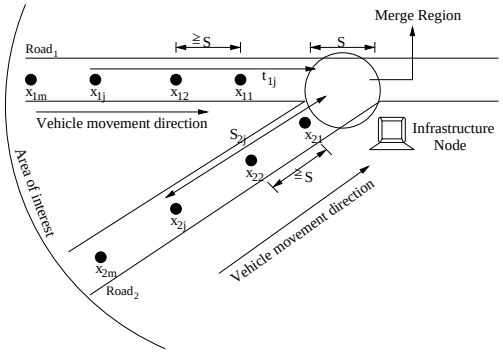

Figure 2. Automatic Merge Control System

Figure 2 shows an intersection of $Road_1$ and $Road_2$, where vehicles (denoted as x_{ij}) are atleast S distance apart from each other and represented by points. It is assumed that $Road_i$ contains m_i vehicles, where $1 \leq i \leq 2$ (road index) and $1 \leq j \leq m_i$ (vehicle index).

- **Precedence Constraint:** ensures no vehicle overtakes its leading vehicle.
- **Mutual Exclusion Constraint:** guarantees that no two vehicles are present in the intersection region at any given instant of time.
- **Safety Constraint:** ensures that safe distance is always maintained between consecutive vehicles on the same road, before they enter the merge region.
- **Other Constraints:** imposes limits on the velocity and acceleration range of vehicles.

3. Overview of Our Approach and Design

In this section, we give the overview of our (i) two algorithms developed for safe maneuvering of autonomous vehicles at road intersections: Head of the Lane (HoL) and All Feasible Sequences (AFS) towards realizing goals 1-5 listed in Section 1.1 and (ii) system support provided for AMC which comprises of (a) a high-level zone-based design where AMC operates in one of the two abstract states, (b) DSRC-based communication protocol for vehicles to communicate with each other and (c) a real-time system support towards achieving goals 3, 4 and 6.

Our algorithms determine the safe merging sequence and profiles (velocity, acceleration, etc.) for every vehicle to follow so as to achieve safe maneuvering at intersections. The profiles of vehicles not only ensure safety distance between vehicles within each lane but also between vehicles of different lanes in the merge region. Among our two algorithms, the HoL algorithm is a distributed solution that considers only the head or lead vehicle on each of the lanes for determining the merge sequence. The AFS algorithm is a centralized solution which looks at a snapshot of vehicles along with their current profiles and determines the merge sequence and new profiles of all the vehicles at once and communicates the same to every vehicle.

Since the algorithms take away the autonomy of vehicles till they cross the merge region, they should try to minimize this duration thereby allowing vehicles to enjoy the autonomous state as much as possible. The abstract level modular design of AMC allows it to operate in different states performing different activities. This helps the designer to tackle sub-problems of AMC independently and integrate them later. We have designed AMC (and hence the vehicles under its influence) to operate in one of the two possible states depending on the distance of vehicles from the merge region.

Our low-level system design exploits two well known design techniques from real-time domain namely, mode-change protocol [2] and real-time data repository [3]. Both the approaches lead 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 tasks necessary in the current mode 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. As part of the system design, we also give the details of DSRC-based wireless communication protocol used for inter-vehicle communication.

3.1. High-Level Design of AMC: Dual State, Triple Zone Approach

In this section, we discuss the high level design of AMC system that operates in different *states* in different *zones*.

To tackle the earlier mentioned sub-problems, S1 and S2, we have abstracted the AMC system to operate in one of the two states, i.e., either in *safe distance* (SD) state where vehicles maintain safe separation distance from each other, or in *safe merge* (SM) state trying to ensure safe merging of vehicles. Hence, all the vehicles which are part of AMC also operate in one of the two states. When in SD state, vehicles continuously monitor their own speed, cruise control settings, and leading vehicle's speed and distance. Using this data, the controller determines the desired acceleration for achieving the safe *Distance of Separation* (DoS) constraint and achieves it. This subsystem is similar to the Adaptive Cruise Control found in automotive literature. When the system is in SM state, the objective is to ensure safe maneuvering of vehicles. Here, vehicles are controlled by some external entity as opposed to autonomous state of vehicles in SD state. The external entity in our case is the merge algorithm which determines the new profiles that the vehicles have to follow till they cross the merge region. To draw a boundary on the section of the road that comes under AMC and to easily understand the system behavior, we have defined an imaginary area known as Area of Interest (AoI). Those vehicles which are inside this area are under the control of the AMC system. The AoI is divided into three zones described below and is shown in Figure 3.

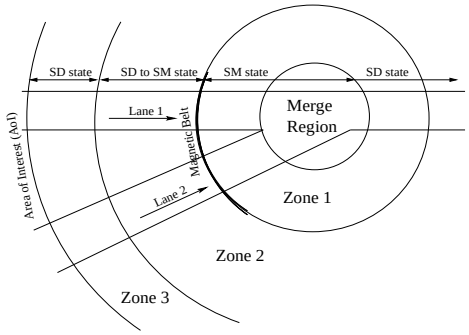

Figure 3. Zone-based partitioning of AOI region with system's state of operation

- **Zone3 (Z_3):** The vehicles enter this zone with random profiles and they need to maintain a safe separation distance from their immediately following vehicles, i.e., the system operates in SD state in this zone.
- **Zone2 (Z_2):** The vehicles present here are considered by AMC algorithms for determining the merge sequence and hence for computing profiles of each vehicle. In this zone, the system transitions from SD to SM state. The exact time of transition of each vehicle from SD to SM state depends on the merge algorithm.
- **Zone1 (Z_1):** The vehicles enter this zone with new profiles (set by AMC) and are left undisturbed i.e., once their profiles are decided by AMC in Z_2, they are never reconsidered for computation. The system operates in SM state here. As soon as a vehicle crosses the merge region, it switches to SD state.

This partitioning of AoI into multiple zones also serves following purposes:

1. **Stability:** The vehicles being left undisturbed in Z_1 add to the stability of the system, as these vehicles are very close to the merge region and have very little flexibility to adapt to any changes in their profile.
2. **Continuous stream of vehicles:** This zonal partitioning also helps algorithms to deal with continuous

streaming of vehicles as described in Section 4.

3. **Modular design:** As described earlier, this has helped to design the system for two exclusive states namely, SD and SM thereby allowing us to look at different sub-problems in different states.

After briefing about the system design, we discuss the three merge algorithms along with a brief description of vehicle to vehicle communication in each of them.

4. AMC Algorithms

We have made following assumptions while developing algorithms.

- The vehicles are assumed to be autonomous and follow the profile decided by AMC.
- An intelligent (communication + computation capable) infrastructure node is situated roadside near the intersection region. It performs all the computations and determines profiles (acceleration, velocity, etc) to be followed by each vehicle in AFS algorithm and is also useful to keep track of the profile of the recent vehicle whose order in the merge sequence was decided.
- There exists a magnetic belt between Z_1 and Z_2 which helps in identifying the leader on each road.
- Only those vehicles which are in AoI are considered by the AMC system.
- All the vehicles execute the same algorithm.

4.1. Head of the Lane Algorithm

This algorithm is a distributed solution that determines the merge sequence in an iterative manner. This approach is motivated by the way drivers in manually driven vehicles resolve the conflict at an intersection in practice. The drivers who are closest to the merge region on each road decide among themselves the order in which they will pass through the region (based on some criteria, say *First Come First Serve*). This algorithm achieves the goal of safe maneuvering by considering the foremost vehicles (a.k.a. Head of the Lane (HoL) vehicle) on each lane for determining the merge sequence. This approach postpones taking over the autonomy from vehicles as long as possible and easily maps to the way merging happen in real-world scenarios where vehicles are not automated. The algorithm is explained in detail below for the scenario as depicted in Figure 2.

Algorithmic Steps:

1. A new vehicle (whose profile is not yet decided) on any of the roads reaching the magnetic belt declares itself as HoL of that road (say x_{11} on $Road_1$). This event triggers HoL to be elected on the other road ($Road_2$).
2. The vehicle which is nearest to the merge region on $Road_2$ and whose profile is not yet determined by AMC is elected as the HoL, say x_{21}. If the other road is empty then the HoL which triggered this action elects itself as the winner with a place in the MS and new profile. Control goes to step 1.
3. Now, x_{21}, sends its profile to x_{11} which performs the computation to determine the winner i.e., the HoL that gets inserted into the MS along with a new behavior that the winner has to follow till it crosses the merge region.
4. x_{11} broadcasts the computed behavior of the winner and the vehicle is inserted into the MS. The winner is elected on the basis of minimizing the DTTI of the vehicles. The algorithm also considers the profile of the vehicle which was recently inserted into the MS so as to respect the safety criteria. If x_{11} itself is not the winner in step 3, it repeats steps 2-4 by declaring itself as the leader on $Road_1$ till it gets inserted into the MS.
5. Steps 1-5 are repeated in a continuous loop.

Once the winner is decided (i.e., x_{11}) in step-4, we have two options for electing the new HoL on that road to proceed with the computation: (i) we determine the next HoL, i.e., x_{12}, immediately after the winner is decided and proceed with the computation along with x_{21}. This can be continued till the profiles of all the vehicles in Z_2 is decided or (ii) we delay the selection of new HoL and hence computation of new winner till one of the vehicles reaches the magnetic belt. We decided to take the latter option since it allows the vehicles to enjoy the autonomous state as long as possible without compromising on the global safety. The algorithm is sporadically executed whenever a vehicle reaches the magnetic belt.

The fact that the HoL algorithm considers only two (head) vehicles at a time for computation makes it easy to deploy in real-world. But, due to the same fact, it might fail to work at higher traffic density since, it does not consider the effect of its decision on the DTTI of following vehicles. In other words, the local decision made might affect the global scenario (simulation results in Section 7 confirm this). To overcome this drawback, we propose another algorithm namely, All Feasible Sequences (AFS) algorithm in next section.

4.2. All Feasible Sequences Algorithm

This algorithm is a centralized solution that considers all the relevant vehicles from a snapshot, i.e., vehicles in Z_2, at a time and determines the merge sequence and profiles of these vehicles. The primary aim of this algorithm is to guarantee safe maneuvering of vehicles even at higher traffic density. The secondary aim is to minimize the average DTTI and to delay taking over the autonomy from vehicles as long as possible. The algorithm is initiated by a new vehicle sending the *MergeInitiate* message upon reaching the magnetic belt on one of the roads. The infrastructure node located adjacent to the merge region collects the profiles of all the vehicles present in Z_2 and tries all possible combina-

tions except those which are eliminated by the constraints specified in Section 2 for determining the solution. The newly computed profiles are communicated back to all the vehicles. We had two options for deciding who should collect all the profiles and perform computation: (i) one of the HoL vehicles or (ii) infrastructure node. We chose the latter option so that HoL vehicle need not perform such a computational and communicational intensive job since it might affect other applications executing on that vehicle.

Since this algorithm works for a static snapshot of vehicles, to make this algorithm work for continuous stream of vehicles, following questions need to be answered: (i) how often should the algorithm be run or the snapshots be captured? and (ii) in the new snapshot, how to handle the vehicles from previous snapshot which have not crossed Z_2.
(i) Frequency of Algorithm: The algorithm is repeated when Z_2 on one of the lanes is full of new vehicles. The algorithm can also be run for every new vehicle or N vehicles entering Z_2 which increases the computational overhead since the new snapshot will mostly consist of vehicles from the old snapshot which leads to redundant computations and also frequent changes in their profiles.
(ii) Handling previous vehicles: Whenever the algorithm is run again with completely new vehicles on one of the lanes, only the relevant old (snapshot) vehicles from the other lane in Z_2 are reconsidered for computation. The DTTI of the first new vehicle say, $DTTI_N$ on the lane (which is completely filled with new vehicles) is determined with the help of its current profile and then only those vehicles from other lane whose DTTI is greater than $DTTI_N$ are reconsidered for new computation.

5. System Support for AMC

After discussing high-level modular design of AMC followed by two algorithms, in this section, we describe the rest of our system support, i.e., (i) communication support for inter-vehicle communication and (ii) real-time system support for ensuring deadline guarantees and for reducing processing power requirements.

5.1. Overview of Inter-Vehicle Communication

In all our algorithms, vehicles communicate using Dedicated Short Range Communication (DSRC) based wireless protocol with each other for exchanging profiles and other information. This is motivated by the fact that IEEE has adopted the DSRC, a variant of 802.11a technology for Vehicle-Vehicle (V-V) and Roadside-Vehicle (R-V) communication. In particular, we have used Asynchronous Fixed Repetitions with Carrier Sensing (AFR-CS) protocol [4] for communication since it has less probability of reception failure and has smaller Channel Busy Time compared to other protocols. The protocol randomly selects 'k' distinct slots among the total 'n' possible slots (equal length intervals) during the lifetime of a message and the message is always repeated a fixed 'k' number of times.

The *MergeInitiate* message initiates inter-vehicle communication, *Profile* message carries profile of a vehicle. Note that in all the algorithms described in this section, once a *MergeInitiate* message is received by vehicles, all future *MergeInitiate* messages are ignored till they receive *MergeStop* message.

5.1.1. Communication Mechanism

Now we describe the communication sequence that takes place between vehicles in a single iteration. In the procedure described below, (i) Initiator refers to the vehicle which initiates communication, i.e., the vehicle that reaches the magnetic belt, (ii) Computing Node refers to the node which executes the algorithm using the data communicated to it. In HoL, the Initiator itself is the computing node, while in case of AFS it is the Road-side Infrastructure node.

1. Initiator sends *MergeInitiate* message.
2. Depending upon the algorithm being used, relevant vehicles send their profiles to the computing node. In HoL, the head vehicle on the other lane sends its profile, while in AFS all vehicles in Z_2 whose behavior has not been decided send their profiles.
3. Computing node uses this information to compute the behavior of relevant vehicles.

   HoL: Behavior of the winner among the two head vehicles is computed.

   AFS: Behavior of the relevant vehicles in Z_2 as described in 4.2 are computed.
4. The computed behavior(s) are sent to the respective vehicle(s).
5. Computing node sends a *MergeStop* message to temporarily terminate the algorithm till another vehicle whose behavior has not been decided reaches the magnetic belt.

   HoL: Above procedure is repeated till the initiator gets inserted into the MS.

Now to complete the system design, we extend the high-level design discussed in Section 3.1 by providing the low-level design details such as task and data management.

5.2. Real-Time Support: Dual-Mode, Two-Level Data Repository Approach

This section describes the low level design of the AMC system for providing real-time support for guaranteeing deadlines and reducing the processing power requirements by integrating mode-change and real-time data repository protocols. Our goal is to develop solutions that address the following issues:

- **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, the validity interval (and hence the sampling period) for the data item *host velocity* will be small when the vehicle is accelerating and it will be large when it is moving with a uniform velocity. To have a fixed sampling time as in existing systems requires a worst-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 [3]. 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 mode specific task sets.** Consider AMC operating in SD state which 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. 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 at all the times. This leads to unnecessary processing power consumption and scheduling overhead.

Our approach to address the above mentioned issues exploits two well known design techniques from real-time system domain: mode-change protocol and real-time data update protocols. The mode-change protocol and real-time data repository approaches are briefly described here before getting into the details of the hybrid design.

Mode-change protocol: Processes controlled by real-time computer systems typically exhibit mutually exclusive phases/modes of operation and control [2]. 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. 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.

Data Repository protocol: All the sensors embedded in an automobile generally sense periodically providing the *raw data* from the environment which should be processed to convert them to application specific data known as *derived data*. The raw items reflect the external environment and 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 [3]. For instance, while maintaining safe distance in SD state, host velocity is derived from the angular velocity of the wheels obtained from four wheel sensors. The data repository approach provides a facility for *on-demand update* of derived data thereby updating the derived data only when necessary.

Now, we proceed to give the details of low-level design of AMC responsible for execution of tasks and to effectively track the dynamically varying data to achieve efficient resource management and provide real-time guarantees.

5.3. Specifics of Dual-Mode Two-Level Real-Time Data Repository System

The concept of mode-change is motivated by the need to carry out different tasks when the vehicle is following a *close* leading vehicle compared to one that is *far*. The concept of data repository 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 controller. The design of the system 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? (iv) How many levels of data repository levels should the design have? (v) What should be the task model and when should the derived data be updated? and (vi) How should the tasks be scheduled to meet their timing requirements? We have explained below how these issues are handled while designing AMC application.

(i) Two mutually exclusive phases of operation: Non-Critical (NC) Mode: In NC 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, 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 allowing other lower priority tasks in the system to be executed. **Safety-Critical (SC) Mode:** In SC mode, the environment status varies rapidly and hence tasks execute more frequently to get as accurate data as possible about environment conditions. Also, the lower priority and not so relevant tasks are discarded in this mode.

(ii) Details of the modes: The decision on the current mode of the system is taken based on two parameters, Distance of Separation (DoS) and the Rate of change of separation distance (RoD) and change in their values triggers the mode-change phenomenon. Table 1 shows all the possible situations for the system to operate in each of the modes. The

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 1. The system state in different modes

characterization of the terms FAR, FOLLOW, NEAR and the details of the conditions to be met by the system in each of the modes and technique to avoid chattering phenomenon are skipped here due to space constraint.

(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, the mode change process involves deleting the tasks in the current mode and creating the new tasks ensuring schedulability at any given point of time throughout this process.

(iv) Two-level real-time data repository: The system comprises of two levels of data store: *Environment Data Repository (EDR)* and *Derived Data Repository (DDR)* as shown in Figure 4. EDR is an active entity, storing the data pertaining to the environment. EDR contains base data items and the procedures for data derivation task. The second repository DDR in the model acts as a global database for the system. As shown in Figure 4, circular nodes represent data items and arrows acting on these nodes represent tasks operating on data items.

(v) Task models and derived data update: The system has four types of task models: (**T1**) periodic Sensor Reading (SR) tasks for collecting data from sensors and updating EDR and periodic *communication receive task*, (**T2**) sporadic On-Demand update (OD) tasks for calculating the derived data and updating DDR, sporadic *communication transmit task* and controller task for maintaining safe distance and ensuring safe merging, (**T3**) periodic Lower level Controller (LC) tasks for giving commands to the mechanical system of the vehicle and (**T4**) other lower priority tasks which are executed only when there are no pending critical tasks to be executed. The sporadicity of T2 tasks depends on the periodicity of relevant tasks in T1 which in turn depends on the mode of operation. 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. This objective is achieved by making the OD tasks sporadic in nature. The sporadic nature of these tasks also helps to provide guarantee to meet their deadlines as opposed to making them aperiodic.

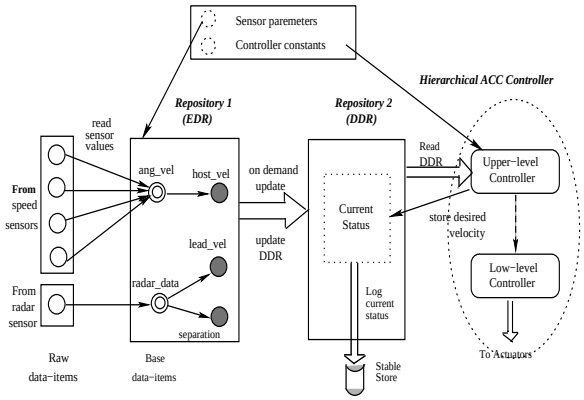

Figure 4. Real-time data repository for maintaining safe distance

Task sets in different modes are shown in Table 2. As described earlier, few tasks are periodic and few tasks are sporadic in nature. The tasks in NC mode perform non-critical operations whereas the tasks in SC mode carry out certain critical operations. The details of task sets are skipped here due to space constraint.

Mode	Task Set
NC Only	WeatherT, FrictionT, AdaptT, EDrT
SC only	TimeLeftT, SuggestT, AdjLaneT
Both Modes	WheelT, SpeedT, CruiseT, SDCtrlT, RadarT, LeadVelT, DistT, DriverT, BrakeT, ThrottleT, SwitchT, MagnetT, CommRxT, CommTxT, SMCtrlT

Table 2. Task sets in different modes of AMC

(vi) Scheduling tasks in different modes: The modes in a system are characterized by number of active tasks. The tasks and their characteristics (periodicity, minimum inter arrival time, WCET) are known a priori. Hence, an algorithm that provides offline guarantee to meet task deadlines is a good choice for our system. The presence of sporadic tasks makes our job of choosing an appropriate scheduling algorithm little non-trivial. Although sporadic tasks have minimum inter arrival time, their arrival is not guaranteed in that interval. The scheduling algorithm can exploit this fact to schedule some lower priority tasks when sporadic tasks fail to arrive. We found the algorithm proposed in [2, 5] as the best match for our task set and hence implemented the same on *RTLinux3.2-pre1* as a scheduler patch. The algorithm provides offline guarantee to both periodic and sporadic tasks and schedules them online to incorporate other low priority and aperiodic tasks whenever possible.

6. Experimental Setup

This section describes the implementation details of both hardware and software setup used to demonstrate the concepts. Java was used to simulate HoL and AFS algorithms along with their communication protocol. The low-level design of the system was implemented on a robotic vehicular platform built in our lab.

6.1. Simulation setup

We implemented both algorithms to operate in dual-state, triple-zone system along with the DSRC based AFR-CS protocol for inter-vehicle communication in Java.

The parameter settings for the Java based simulations were as follows: (i) Environmental settings: safe separation distance was set to $5m$, radius of Z_3, Z_2 and Z_1 from the merge region were set to $400m, 200m$ and $150m$ respectively (ii) Vehicles behavior: initial velocity of vehicles was uniformly distributed between 17 and $20m/s$, V_{MAX}, A_{MAX} and A_{MIN} were set to $30m/s, 4m/s^2 and -4m/s^2$ respectively. Vehicle generation rate per road was varied from $0.2 - 1.9$ $vehicle/s$. (iii) communication protocol settings: the packet size was set to $100bytes$ and its lifetime was set to $0.02s$, the transmission rate was set to $20Mbps$, and finally number of retransmission slots for every vehicle was varied between 1 to 20.

6.2. Vehicular platform setup

The hardware platforms were used to demonstrate the dual mode, two-level data repository concept. We could implement the dual-mode two-level data repository system on a robotic vehicular platform due to the dedicated effort of our hardware design team which successfully developed two robots within the short timeframe. Since it was difficult to replicate more robots in that duration, the effort of demonstrating the whole system on vehicular platforms is in progress.

The vehicular platforms used for conducting experiments is shown in Figure 5 and had following features:

- Obstacle detection range: $30cm$.
- Maximum speed: $50cm/s$.
- Maintains path through white-line following.
- Closed-loop controller(s).

The robot was controlled by a PC running on *RTLinux3.2-pre1* platform. The PC performed all the computations and issued commands to the robot. We implemented the scheduling algorithm discussed in [5, 2] for providing offline guarantee to both periodic and sporadic tasks in our system. The partial task structure and data items from our implementation is shown in Figure 6.

The DoS is kept as the criterion for changing the mode. The periodicity of the tasks in NC mode is set to half that of the frequency in SC mode, i.e., $0.2s$ in SC and $0.4s$ in NC mode. We observed the invocation of DistT on-demand

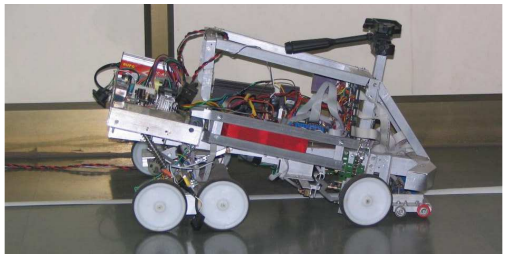

Figure 5. Experimental Vehicular Platform

update task (which derives the separation distance of leading vehicle). Its minimum inter arrival time was set to $0.2s$ in SC and $0.4s$ in NC mode and it gets invoked only when the raw data representing the separation distance of leading vehicle varies by a number which is equivalent to $1cm$.

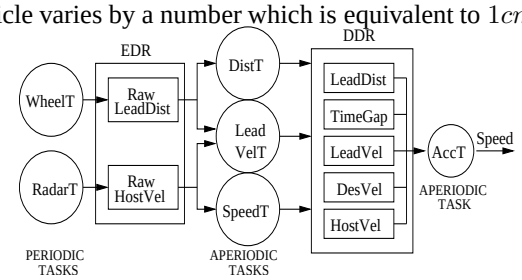

Figure 6. Task and data structure in real-time repository implementation

7. Results and Observations

In this section, we describe the results obtained from the AMC experiments and observations that were made in the dual mode two-level data repository approach.

Simulation Results: In the Java simulations, the algorithms were compared w.r.t. average duration of external control and average DTTI. The communication behavior for AFS algorithm under various traffic densities was also observed. Experiments were carried out at various traffic densities as shown in Figure 7 where average external control time and average DTTI are plotted against vehicle generation rate (λ). Higher value of λ indicates high traffic density. The infeasible regions of algorithms depicted in the graph indicate the λ value beyond which the algorithms fail to ensure safe merging. HoL fails to give solution for $\lambda > 1$, while AFS continues to give solution even at high values of λ, i.e $\lambda < 1.9$. This observation confirms our intuition about failure of HoL as discussed in section 4.1. At higher values of λ (0.5, 1) AFS has higher duration of external control as compared to HoL, which can be attributed to early determination of vehicle behavior in AFS. AFS and HoL show similar results w.r.t. DTTI. Also at lower λ, AFS behaves similar to HoL w.r.t. both duration of external control and DTTI, since the number of vehicles in Z_2 is very less(1-2). As a result AFS considers only 1-2 vehicles in every iteration which is similar to HoL.

Communication behavior of AFS algorithm is shown

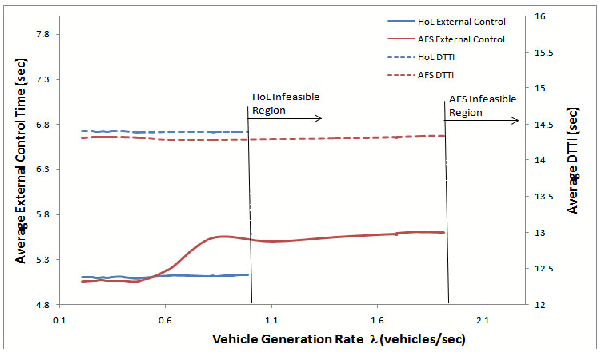

Figure 7. HoL v/s AFS: DTTI and external control duration

in Figure 8. Probability of Reception Failure (PRF) and Channel busy time(CBT) was observed for various values of retransmission number(K) of AFR-CS protocol for $\lambda = 1.5, 2.5, 3.7$. As we can see in the graph, since PRF stabilizes for $K \geq 4$, we chose $K = 4$ for performing experiments since it has the least CBT. The behavioral pattern observed is in accordance with those demonstrated in [4]. In HoL algorithm, since only two (head) vehicles communicate at any given point of time, changing the vehicle density will not have any impact on CBT and PRF.

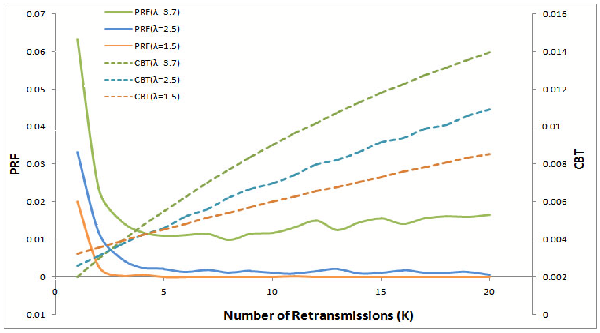

Figure 8. AFS performance: CBT and PRF

Vehicular Platform Results: An experiment was conducted on the robotic vehicular platform to observe the reduction in processing power requirement in dual-mode two-level data repository approach. The velocity response of the host vehicle along with the mode of operation and DistT task invocation is depicted in Figure 9 where the DoS increases in time intervals $0 - 9s$, remains constant between $9 - 12s$, decreases between $12 - 17s$, remains constant again between $17 - 21s$, increases again between $21 - 30s$ before decreasing again between $30 - 33s$.

We can observe from the graph that the system operates in SC mode between $0 - 2s$ and in NC mode between $2 - 14s$. It again enters SC mode at $14s$ and continues to be in that mode till it switches back to NC mode at $22s$. Since the tasks in NC mode operate at double the periodicity compared to SC mode, it is trivial to estimate that nearly 50% of the processing power requirements is reduced. Note that, the general practice of designing the system without mode-change protocol would have assigned the periodicities of all the tasks as $0.2s$ to be able to handle the worst-case scenario. We can also observe that during the time interval $9 - 13, 17 - 21$ and $26 - 29s$ when the DoS was constant, the on-demand update task, i.e., *DistT* is not invoked. Thus the task updating the DoS in DDR executes only when necessary. The *DistT* task is invoked only 17 times in our design whereas in generally followed design practice, it would have been invoked periodically 165 times. This significant difference in the number of times the task is executed leads to considerable amount of reduction in processing power requirement. As we can observe from the graph, the DoS maintained by the system is always greater than the desired DoS except only on one occasion, i.e., at $16s$, which can be attributed to the inertia of the vehicle. This suggests that a conservative approach should be taken while deciding the safe DoS by taking this factor into account.

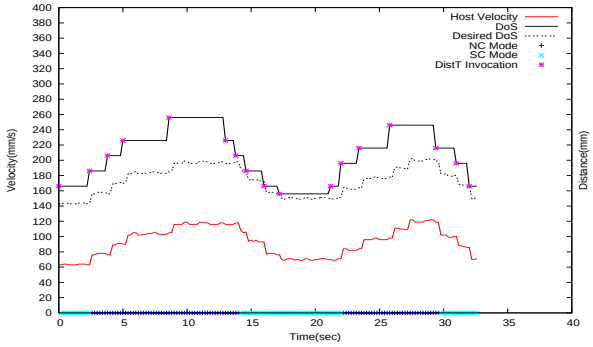

Figure 9. Dual-mode Two-level repository AMC system behavior: DoS maintained, DistT task invocation, mode of operation

8. Related Work

The merge control application with inter-vehicle communication is also studied in [6]. It uses the concept of virtual vehicle that is used to map vehicles on one lane onto the other lane (assuming a 2-lane merge) for ensuring safe distance criteria. But the algorithm for determining the merge order of vehicles is not provided. In [7], a reservation based multi-agent (reservation manager and driver agent) approach is proposed for designing the AMC system. The driver agents requests for space-time in the intersection which is either serviced or rejected depending on the availability. This work comes close to ours. We believe the main drawback of this approach is the process of repeated requests by the driver agent when its initial request is not met. The intersection manager should be more smart to make use of all the vehicles' information available with it and suggest or block an alternate space-time in the intersection instead of rejecting the request and wait for that driver agent to make another request. Also, the paper fails

to give communication support for the application. In [8], we had given HoL algorithm along with constraint based formulation for AMC. We have enhanced our earlier work by: extending HoL algorithm to delay the external control as long as possible, handling continuous stream of vehicles and providing communication support using DSRC-based protocol. A data centric approach to the architectural design of performance critical vehicular applications has been examined before in [3]. In particular, [9] addresses the issues in the design and implementation of an active real-time database system for Electronic Engine Control Unit software. A set of on-demand updating algorithms: On-Demand Depth First Traversal and On-Demand Top Bottom are presented in [3]. These algorithms optimistically skip unnecessary updates and hence provide increased performance. Methods for the specification and runtime treatment of mode changes are discussed in [2]. We have tailored these approaches to suit our AMC application. We had looked at real-time issues in maintaining safe distance between vehicles and had proposed two ways to provide real-time support for designing the system for efficient resource utilization, one using the dual-mode approach and the other using two level real-time data repository protocol in [10]. In this paper, we have enhanced our earlier work by integrating both the approaches: for achieving better resource utilization compared to both individual approaches and for providing deadline guarantees to on-demand update tasks by converting them to sporadic (from aperiodic).

9. Conclusions and Further Work

Given the increased intelligence being built into, and the resulting increase in the number of processors in modern automobiles, there is a need to minimize the computing power required without affecting the performance and safety of the applications. A systematic solution to the incumbent problems is important since these by-wire applications are distributed and real-time in nature and hence deal with critical functions. Our work studied one such safety-critical application namely, Automatic Merge Control (AMC) which ensures safe vehicle maneuver in the region where two or more roads intersect. We proposed two merge algorithms: Head of the Lane (HoL) and All Feasible Sequences (AFS) and also presented system support (both communication and computing infrastructure) for AMC. We also showed how DSRC-based wireless communication protocol can be leveraged for the development of AMC and how mode-change and real-time repository concepts can be integrated for reducing the processing power requirements. Experimental results demonstrated that HoL works only at lower traffic density whereas AFS continues to give solution even at higher densities while both the algorithms gave similar performance w.r.t. DTTI. Also, experiments demonstrated that the processing power requirement is reduced using our design without compromising the real-time guarantees.

In future, we plan to complete our setup for empirical studies involving the merge of mobile vehicles in intersections involving n merging roads. Also, we will be looking at reliability and safety issues arising from the practical need to allow driver control within the merge region.

References

[1] H. Kopetz, "Automotive electronics," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, York, England, 1999, pp. 132–140.

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

[3] 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.

[4] Q. Xu, T. Mak, J. Ko, and R. Sengupta, "Vehicle-to-vehicle safety messaging in DSRC," in *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, NY, USA, 2004, pp. 19–28.

[5] D. Isovic and G. Fohler, "Efficient scheduling of sporadic, aperiodic, and periodic tasks with complex constraints," in *Proceedings of the 21st International IEEE Real-Time Systems Symposium*, Walt Disney World, Orlando, Florida, USA, Nov 2000.

[6] T. Uno, A. Sakaguchi and S. Tsugawa, "A merging control algorithm based on inter-vehicle communication," in *IEEE International Conference on Intelligent Transportation Systems*, Japan, 1999, pp. 783–787.

[7] K. Dresner and P. Stone, "Multiagent traffic management: An improved intersection control mechanism," in *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, Eds. NY: ACM Press, Jul 2005.

[8] G. Raravi, V. Shingde, K. Ramamritham, and J. Bharadia, "Merge algorithms for intelligent vehicles," in *GM Workshop on Next Generation Design and Verification Methodologies for Distributed Embedded Control Systems*. India: Springer-Verlag, 2007.

[9] 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.

[10] G. Raravi, N. Sharma, K. Ramamritham, and S. Malewar, "Efficient real-time support for automotive applications: A case study," in *RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, 2006, pp. 335–341.