Merge-by-Wire: Algorithms and System Support

Vipul Shingde, Gurulingesh Raravi, Ashish Gudhe, Prakhar Goyal, Krithi Ramamritham
Embedded Real-Time Systems Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
{vipul.shingde,gurulingesh,ashish.gudhe}@gmail.com, {prakhargoyal,krithi}@cse.iitb.ac.in

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 involve deadline bound computations on critical 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 automotive 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) provide a scheduling strategy to meet AMC tasks' timing requirements. Simulations and implementation on robotic vehicular platforms 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 as many as 80 microprocessors [1]. The features currently governed by Electronic Control Systems (ECSs) 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.

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 the microprocessors. This trend of increasing the number of microprocessors in relation to individual applications will be difficult to maintain 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 vehicles 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 while deploying shared processing power.

With a larger goal of understanding the requirements of 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*).

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

1. Ensure safe maneuvering of vehicles at intersections.
2. Optimize average *Driving-Time-To-Intersection (DTTI)*. DTTI is the time taken by a vehicle to cross the merge region. It is further explained in Section 2.
3. Maintain a safe separation distance between two successive vehicles on each lane.
4. Efficiently utilize both computational and communicational resources.

1052-8725/08 $25.00 © 2008 IEEE
DOI 10.1109/RTSS.2008.51

5. Minimize the average *duration of external control* which is equal to the time during which vehicles lose their autonomy, i.e., are controlled by AMC system.
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. A bird's eye view of the entire paper is shown in Figure 1 (with section numbers in square brackets).

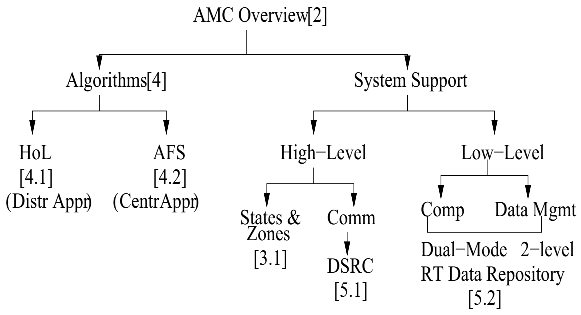

Figure 1. Overview of our contributions

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. Experiments and their results are discussed in Section 6 and 7 respectively. Section 8 presents the related work followed by conclusions and future work.

2. Automatic Merge Control System: An Overview

Automatic Merge Control (AMC) is a distributed intelligent control system that ensures safe vehicle maneuvering at road intersections. It ensures that the time instants at which two vehicles cross the merge region are separated by at least δ (which depends on the size of the merge region and permitted velocity of vehicles while at the intersection), by giving commands to adapt their velocities appropriately. This system involves the following sub-problems:

- **S1**: Maintain safe separation distance between vehicles.
- **S2**: Ensure safe maneuvering of vehicles at merge region and hence determine the Merge Sequence (MS) i.e., the order in which vehicles cross the merge region.
- **S3**: Minimize the time taken by vehicles to cross the merge region, for example, minimize the average Driving Time To Intersection (DTTI). DTTI of a vehicle is the time it takes from Area of Interest (AoI) boundary (refer Figure 2) to cross the merge region.

Figure 2 shows an intersection of $Lane_1$ and $Lane_2$, where vehicles (denoted as x_{ij}) are atleast S distance apart

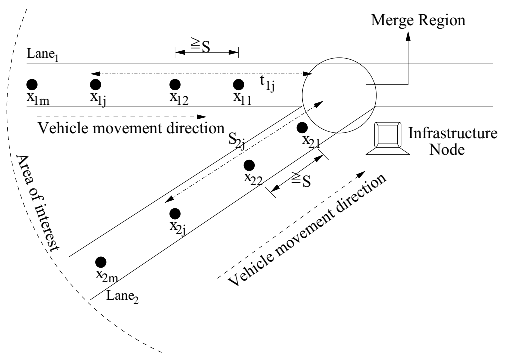

Figure 2. Automatic Merge Control System

from each other and represented by points. It is assumed that $Lane_i$ contains m_i vehicles, where $1 \le i \le 2$ (lane index) and $1 \le j \le m_i$ (vehicle index).

The AMC has to respect the following constraints at any given point of time:

- **Precedence Constraint:** No vehicle overtakes its leading vehicle.
- **Mutual Exclusion Constraint:** No two vehicles are present in the merge region at any given instant of time.
- **Safety Constraint:** Safe distance is always maintained between consecutive vehicles on the same lane, before they enter the merge region.
- **Other Constraints:** Limits on the velocity and acceleration range of vehicles must be respected.

3. Overview of Our Approach and Design

In this section, we give an overview of the 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. We also discuss system support provided for AMC which comprises of (a) a high-level dual-state triple-zone design (b) DSRC-based communication protocol [2] for vehicles to communicate with each other and (c) real-time system support towards achieving goals 3, 4 and 6 as defined in Section 1.

Our algorithms determine the safe merging sequence and *profile* (velocity, acceleration, etc.) for every vehicle to adhere to so as to achieve safe maneuvering at intersections. Among our two algorithms, HoL is a distributed solution that considers only the head or lead vehicle on each of the lanes for determining the merge sequence. Head or lead vehicle is defined as the vehicle closest to the merge region and whose profile has not yet been decided by AMC algorithm. AFS 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.

Our low-level system design exploits two well known design techniques from real-time domain namely, mode-change protocol [3] and real-time data repository [4]. 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 their characteristics (period, etc.) as the system moves from one mode to another. 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. In this paper, we have integrated both these approaches to leverage the advantages offered by the two methods. The DSRC-based wireless communication protocol used for inter-vehicle communication is explained in Section 5.2.

We now 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, a *safe distance* (SD) state where vehicles maintain safe separation distance from each other, or in *safe merge* (SM) state where safe merging of vehicles is ensured. 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 Area of Interest (AoI).The AoI is divided into three zones described below and is shown in Figure 3.

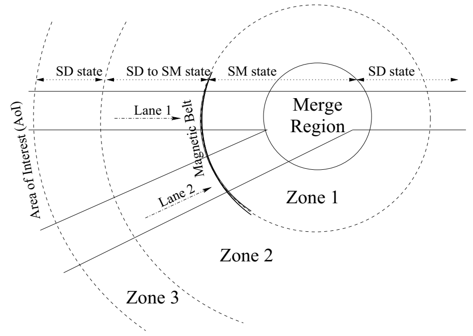

Figure 3. Zone-based partitioning of AOI region with system's state of operation

- **Zone3 (Z_3):** The vehicles enter this zone with random profiles but they maintain a safe separation distance from their immediately following vehicles, i.e., the system operates in SD state in this zone. Even though vehicles are required to operate in SD state in this zone, they are not directly under the control of AMC system, as each vehicle itself determines its own profile and follows it.
- **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 time or space to adapt to any new 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 helps 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 two merge algorithms along with a brief description of vehicle to vehicle communication in each of them.

4. AMC Algorithms

We have made the following assumptions while developing the algorithms.

- The vehicles are assumed to follow the profile decided by AMC.
- In AFS algorithm, an intelligent (communication + computation capable) infrastructure node is situated roadside near the merge region. It performs all the computations and determines profiles to be followed by the vehicles. It does this while maintaining the profiles of vehicles whose order in the merge sequence is already decided. Note that the infrastructure node, which might incur additional setup cost is required only by the AFS algorithm.
- There exists a mechanism to identify the lead vehicles on each lane (e.g., a magnetic belt).
- Only those vehicles which are in AoI are considered by the AMC system.
- All the vehicles execute the same algorithm.

4.1. Head of Lane Algorithm: HoL

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 lane decide among themselves the order in which they pass through the merge region (based on some criterion, say *First Come First Serve*). This algorithm achieves the goal of safe maneuvering by considering the lead vehicle (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 happens 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. When a lead vehicle on any of the roads reaches magnetic belt, it declares itself as HoL of that road (say x_{11} on $Lane_1$). This event triggers HoL to be ëlectedön the other road ($Lane_2$).
2. The vehicle which is nearest to the merge region on $Lane_2$ 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. 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 profile 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 $Lane_1$ till it gets inserted into the MS.
5. Steps 1-5 are repeated in a continuous loop.

When x_{11} is decided as the winner in step-4, the other option for electing the new HoL on that road (instead of waiting till one of the vehicles reaches the magnetic belt) is to immediately declare x_{12} as the new HoL. We decided to take the earlier 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 the real-world. But, due to the same fact, it might fail to work at a 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 the next section.

4.2. All Feasible Sequences Algorithm: AFS

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 densities. 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 combinations except those which are eliminated by the constraints specified in Section 2. For e.g., precedence constraint eliminates those combinations which don't retain the original order (the one determined by the ascending order of their distance from the merge region) for determining the solution. The newly computed profiles are communicated back to all the vehicles.

Since this algorithm works with a snapshot of vehicles (set of vehicles that are present in Z_2 at a given point of time), to make this algorithm work for continuous stream of vehicles, following issues need to be considered:

(i) Frequency of Invocation of Algorithm: The algorithm is invoked when Z_2 on one of the lanes contains new vehicles. The algorithm can also be run for every new vehicle or N vehicles entering Z_2, but this will increase the computational overhead since majority of vehicles in new snapshot will be from old snapshot (whose profiles have already been computed in previous iteration) which would lead to redundant computations and also frequent changes in their profiles.

(ii) Handling Previous Vehicles: Whenever the algorithm is executed again with completely new vehicles on one of the lanes (say $Lane_1$), other lane ($Lane_2$) might have old vehicles (vehicles considered in previous snapshot) in Z_2, whose profiles have already been computed in previous iteration. Apart from all new vehicles in Z_2, only the relevant vehicles from previous snapshot (i.e., only those vehicles whose profiles will be affected by the new vehicles) from the other lane ($Lane_2$) in Z_2 are reconsidered for the current iteration. In order to determine which old vehicles from the other lane ($Lane_2$) to reconsider, following analysis is done. Minimum DTTI (say, $DTTI_{1,min}$) of the first new vehicle on the lane which triggered the current iteration ($Lane_1$) is determined with the help of its current profile and profile of vehicle immediately in front of it. Only those old vehicles from other lane ($Lane_2$) whose DTTI (determined using the profile computed in previous iteration) is greater than

$DTTI_{1,min}$ are reconsidered for new computation.

5. System Support for AMC

In this section, we describe the rest of our system support, i.e., (i) real-time system support for ensuring deadline guarantees and for reducing processing power requirements and (ii) communication support for inter-vehicle communication.

5.1. 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 from a sensor reflects the status of the real-world 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 processing power.
- **Timely updates of derived data.** The data derivation should be complete before the derived item's read set becomes invalid [4]. The derived item needs to be updated only when one or more data items from its read set change more than a threshold value.
- **Handling mode-specific task sets.** AMC operating in SD state performs different tasks while following a close vehicle when compared to following a vehicle which is far away. In current approaches, all the tasks across multiple modes 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 repository protocols.
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.

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.

Mode-change protocol: Real-time applications typically exhibit mutually exclusive phases/modes of operation and control [3]. 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 all the time. The design of the system with this approach is explained in the following points:

(i) Two mutually exclusive modes 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 Distance of Separation (DoS) and leading vehicle velocity do not change rapidly.
- **Safety-Critical (SC) Mode:** In SC mode, the environment status varies rapidly and hence tasks execute more frequently to get as accurate a view as possible about the environment.

(ii) Details of the modes: The decision on the current mode of the system is taken based on two parameters, DoS and Rate of change of separation distance (RoD) and change in their values triggers the mode-change. Table 1 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 1. The system state in different modes

(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.

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.

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

Data Repository: 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 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 [4]. For instance, while maintaining safe distance in SD state, vehicle 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.

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.

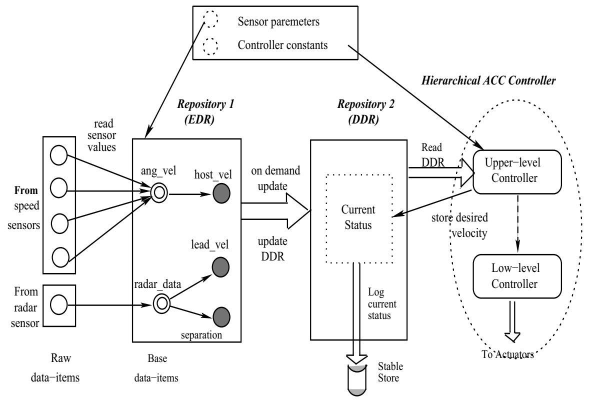

Figure 4. Real-time data repository for maintaining safe distance

Tasks and derived data update: The system has four types of tasks: (**T1**) periodic sensor reading tasks for collecting data from sensors and updating EDR and periodic *communication receive task*, (**T2**) sporadic on-demand update 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 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.

Scheduling tasks in different modes: The tasks and their characteristics (periodicity, minimum inter arrival time) 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 a little non-trivial. We found the algorithm proposed in [3, 5] as the best match for our task set. The algorithm provides offline guarantees to both periodic and sporadic tasks and schedules them online to incorporate other low priority and aperiodic tasks whenever possible.

5.2. Inter-Vehicle Communication Scheme

In all our algorithms, vehicles communicate using Dedicated Short Range Communication (DSRC) based wireless protocol [2] 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 [2] for communication since it has less probability of reception failure and 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.

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, specified vehicles send their profiles to the computing node.
   HoL: The head vehicle on the other lane sends its profile.
   AFS: All vehicles in Z_2 whose behavior has not been decided send their profiles.
3. Computing node uses this information to compute the profile of specified vehicles.
   HoL: Profile of the winner among the two head vehicles is computed.
   AFS: Profiles of the specified vehicles in Z_2 as described in 4.2 are computed.

4. The computed profile(s) are sent to the specified vehicle(s) using AFR-CS protocol.
5. Computing node sends a *MergeStop* message to temporarily terminate the algorithm till another vehicle whose profile has not been decided reaches the magnetic belt.
   HoL: Above procedure is repeated till the initiator gets inserted into the merge sequence.

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.

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 $5m$, radii 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 lane 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. Note that the length of Z_2 is $50m$ and with $5m$ as the separation distance, at any moment atmost 20 vehicles will lie in Z_2. AFS algorithm can easily handle such number of vehicles using low-end infrastructure node without compromising on the safety of vehicles.

The robotic platforms were used to demonstrate the dual mode, two-level data repository concept. The vehicular platforms used for conducting experiments are shown in Figure 5 and Figure 6 and had following features:

- Obstacle detection range: $30cm$.
- Maximum speed: $50cm/s$(Dexter), $10cm/s$(Firebird).
- Wireless radio module.
- Maintains path through white-line following.
- Closed-loop controller(s).

The robot in Figure 5 named Dexter 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 [3, 5] for providing offline guarantee to both periodic and sporadic tasks in our system. The robot in Figure 6 named FireBird was used to implement the AMC algorithms developed by us in addition to the algorithm discussed in [6].

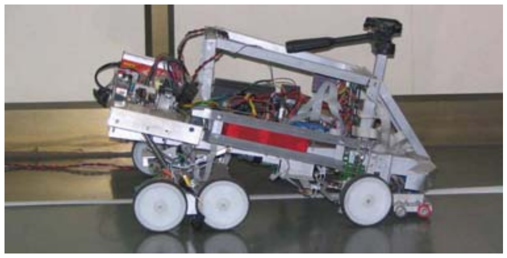

Figure 5. Experimental Vehicular Platform - Dexter

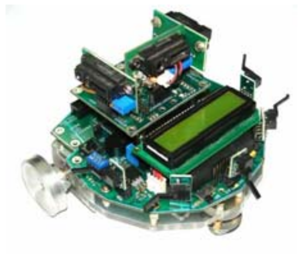

Figure 6. Experimental Vehicular Platform - FireBird

The partial task structure and data items from our implementation is shown in Figure 7. The DoS is kept as the criterion for changing the mode. The periodicity of the tasks in NC mode is set to twice that in SC mode, i.e., $0.2s$ in SC and $0.4s$ in NC mode. We observed the invocation pattern of DistT, an on-demand 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 threshold which is equivalent to $1cm$.

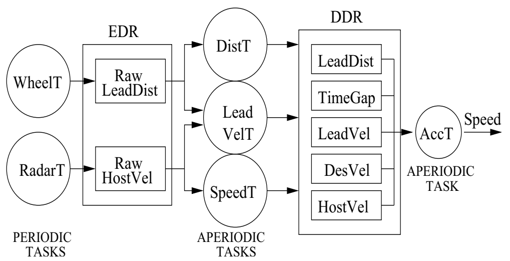

Figure 7. Task and data structure in real-time repository implementation

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.

7.1. Simulation Results

In the Java simulations, the algorithms were compared with respect to average duration of external control and average DTTI. The communication behavior for AFS algorithm under various traffic densities was also observed. Figure 8 plots average external control time and average DTTI against vehicle generation rate (λ). Higher value of λ (vehicles/sec) indicates higher 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 solutions for $\lambda > 1$, while AFS continues to give solutions even at high values of λ, upto $\lambda < 1.9$. This observation confirms our intuition about HoL as discussed in section 4.1. As λ varies from 0.5 to 1, AFS has greater 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 small (1-2). As a result AFS considers only 1-2 vehicles in every iteration, which is similar to HoL.

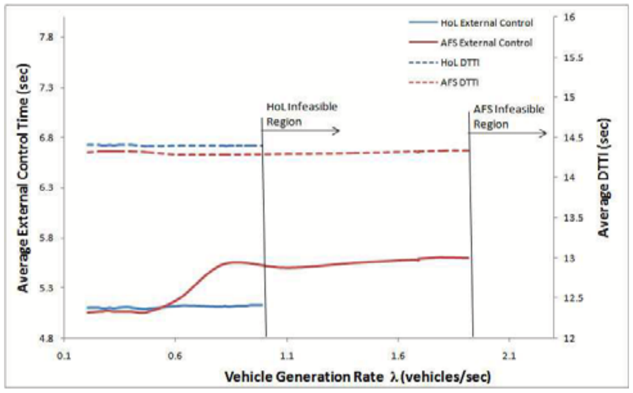

Figure 8. HoL and AFS: DTTI and external control duration

Communication behavior of AFS algorithm is shown in Figure 9. Probability of Reception Failure (PRF) and Channel Busy Time (CBT) were 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 [2]. In HoL, 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.

7.2. Vehicular Platform Results

Here we describe the results of the experiments on the two robotic vehicular platforms described in section 6.

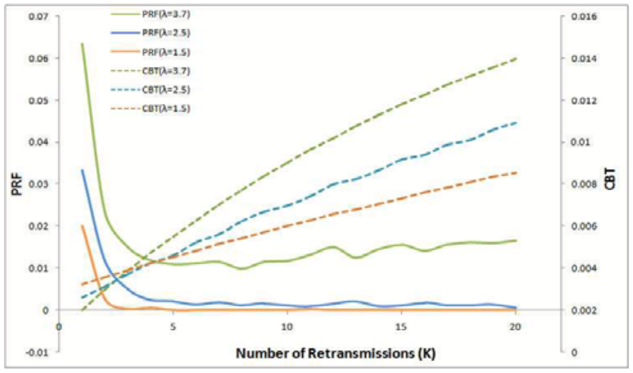

Figure 9. AFS performance: CBT and PRF

7.2.1. Reduction in processing power required

An experiment was conducted on the robotic vehicular platform shown in Figure 5 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 10 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 $30-33s$.

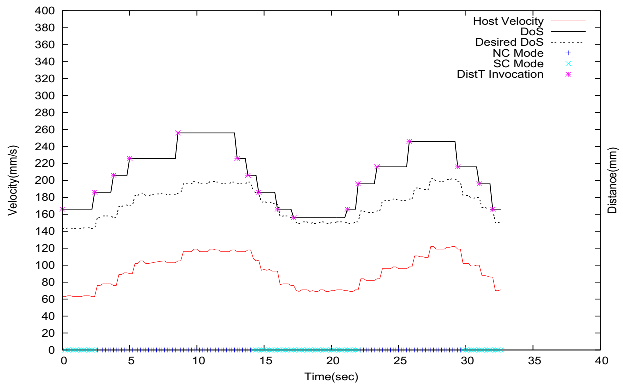

Figure 10. Dual-mode Two-level repository AMC system behavior: DoS maintained, DistT task invocation, mode of operation

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 the processing power requirements is reduced to 50%. 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 reduction in processing power requirement. As is observed from the graph, the DoS maintained by the system is always greater than the desired DoS except 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.

7.2.2. Implementation of AMC algorithms

The initial distance of the vehicles from the merge point was kept at 1200 mm. The desired safe distance to be maintained between two vehicles at the merge point was specified as 200 mm. The graph in Figure 11 shows the comparison of Distance of Separation (DoS) of AMC algorithms (HoL and AFS) with the virtual vehicle based work in [6]. In the implementation of virtual vehicle based algorithm, the vehicles maintained safe distance with the map of vehicles present on other lane. It is observed from the graph that AMC algorithms - HoL and AFS always maintain desired safe distance when the vehicles cross the merge point, whereas the virtual vehicle based algorithm gives inaccurate results under the same experimental conditions. AFS starts maintaining 200 mm distance even when the vehicles are at 600 mm from the merge point.

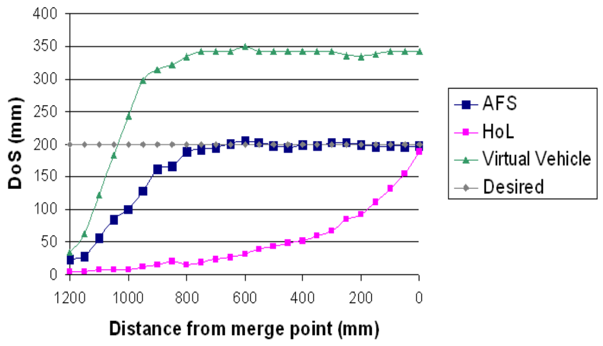

Figure 11. Distance of Separation comparison

The Average Driving-Time-To-Intersection (DTTI) comparison of the two algorithms with virtual vehicle based algorithm is shown in Figure 12. In this experiment, the number of vehicles in the system was increased from 1 to 4. The graph depicts that when there is one vehicle in the system, all the three algorithms deliver same results. As the number of vehicles increases from 1 to 4, AFS and HoL incur lower DTTI as compared to virtual vehicle based algorithm.

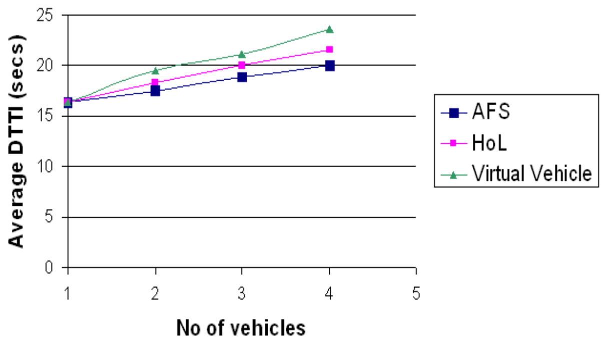

Figure 12. Average DTTI comparison

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. 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 smarter to make use of all the vehicles' information available with it and suggest 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 doesn't discuss 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, adding the handling of 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 in [4]. 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 [4]. 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]. 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 Future 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.

We realize that deploying such a system is not feasible in the current situation, as some of the required technology is still under development. Also the millions of non-collaborative or human-driven vehicles currently being used will be around in the future. Thus in future even with the development of appropriate technology and the required infrastructure, a sizeable fraction of vehicles would be non-collaborative and hence a system that handles mixture of both types of vehicles must be developed. Considering this, we are currently planning to work on enhancing our solution to handle such mixed traffic.

In the future, we plan to complete our setup for empirical studies involving the merge of mobile vehicles in intersections involving more than two merging lanes. 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] 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.

[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] 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.

[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*, 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 Real-Time and Embedded Technology and Applications Symposium*, 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 *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, 2006, pp. 335–341.