Vehicular Coordination via a Safety Kernel in the Gulliver Test-bed

António Casimiro
Faculdade de Ciências
da Universidade de Lisboa, Portugal
casim@di.fc.ul.pt

Oscar Morales-Ponce, Thomas Petig and Elad M. Schiller
Computer Science and Engineering
Chalmers University of Technology, Göteborg, Sweden
{mooscar, petig, elad}@chalmers.se

***Abstract*—Cooperative vehicular systems base their coordination on inherently uncertain inter-vehicle communications. If not conveniently managed, this uncertainty can ether lead to inefficient coordination solutions or to optimistic but unsafe ones. We consider that cooperative functions can be executed with several service levels and we use the system architectural concept of *safety kernel* for managing the service level in order to achieve the best possible performance while keeping the system safe.**

We use the Gulliver test-bed for demonstrating the safety kernel concept by means of a pilot system implementation on scaled vehicles with sensors and communication capabilities. The demonstrated architecture incorporates: (1) a *local dynamic map* (LDM) that uses local and remote sensory information for calculating the location of nearby objects, (2) a safety kernel to manage service levels, (3) a *cooperative level of service evaluator* that allows vehicles to reach agreement on a common service level and, finally, (4) a *driver manager* that executes in accordance to the cooperative level of service when determining how to calculate the trajectory. This paper explains how the different components considered in the architectural concept operate, and shows how it is possible to use (similar to existing) trajectory planning algorithms when implementing the concept.

I. Introduction

A step forward from autonomous to cooperative vehicles is coming, with smart vehicles that have not only automatic driving functions and advanced driver assistance systems, but which can also cooperate with other vehicles when realizing those functions. Some examples include vehicular platooning, coordinated intersection crossing, cooperative lane change, to name a few. These cooperative systems inherently depend on timed wireless communications to reach agreements on the actions that they have to perform to safely complete a maneuver. Hence, timely communication becomes a key component for road safety. However, it is well known that wireless communication is prone to failures due to external factors, such as interference or weather, and this can happen with a probability that cannot be neglected in safety critical applications. The KARYON project is aiming to provide a mechanism to operate vehicular coordination systems in a safe manner in spite of faults affecting communication or affecting the perception (through sensors) of the surrounding environment. The key concept is to consider that functions can be performed according to several levels of service (LoS), and having a safety kernel to select the highest possible, but still safe, LoS. Higher service levels require healthier systems than lower ones, but offer better performance (e.g., fuel consumption, traveling time). By selecting the appropriate LoS it becomes possible to improve performance while meeting safety objectives, despite faults and uncertainties affecting system.

This work demonstrates how the safety kernel facilitates the design and implementation of Cooperative Advanced Driver Assistance Systems (CADAS) that are safe in the presence of uncertainty. Our approach considers the Gulliver test-bed [6], [15], with miniature vehicles that allows the implementation of CADAS at low cost while provide insights of the systems in real vehicles.

The main problem in autonomous vehicle coordination is to make sure that cooperative functions are performed safely, i.e., that no accidents occur. Current driver-less vehicle implementations that operate on the road (e.g., Google car), do not consider cooperative path plans. As a result, the risk exclusion of causing severe damage with sufficient certainty requires longer headways. Recall that the *headway* is the distance, expressed as a time value, that the vehicle requires to perform operations safely. Recent developments in the automotive domain, as well as in communication networks, follow a more structured approach regarding information dissemination about road hazards, vehicle whereabouts and cooperative driving. We follow this structured- and cooperative-driving approach and consider a system in which accidents are avoided via periodic agreement on cooperative service level and recalculation of the path plans.

Vehicles use the wireless network for requesting information such as localization, speed, or intention of each other vehicle. Once all vehicles receive the information, each determines safe paths for all service levels that the vehicle supports before agreeing on a cooperative one. These paths ensure a safe driving for any coherent, yet possibly changing, service level. For example, in the case of cooperative functionalities such as adaptive cruise control and vehicle Platooning, the plan considers the safe distance headway for each of the possible cooperative service level.

Cooperative driving is not only dependent on the expected vehicle paths and the local sensory information but also on the data accuracy (data validity). Thus, these functionalities are based on periodic information exchange with nearby vehicles, and validity assessment before deciding on the cooperative LoS. It may be possible to consider systems in which vehicles select the LoS on a local basis, using only the locally available validity information, in contrast with agreeing on a cooperative service level. However, the safety analysis in this case must keep track of all the possible service level combinations of nearby vehicles, leading to less performing solutions. We use a cooperative LoS evaluator in a way that guarantees safe transition between a service level in which vehicles cooperate, and a service level in which they perform the functions autonomously (which is the only safe strategy when communication is not possible, or is untimely). And the solution ensures that all nearby vehicles coherently select a LoS. Note that although we consider an example with only two service levels, the architectural concept is applicable to systems with several levels of service.

A. Our Contribution and Related Work

This work considers the novel system architectural concept of *safety kernel*. We show how the safety kernel concept can facilitate the service level management according to observed timeliness and validity measures. We have implemented a pilot system in the Gulliver test-bed. The pilot demonstrates the safety kernel concept in a system architecture that incorporates a local dynamic map, and a driver manager that executes in accordance to the cooperative service level when determining how to calculate the trajectory. This paper shows how these components operate in a vehicular system that bases its trajectory planning on cooperative service level management according to observed timeliness and validity measures.

We have presented the architectural concept of safety kernel in [7], and its details in [10]. We the presented the Gulliver test-bed in [15], [6], and claimed it usefulness. This work is the first to show the complete picture. Due to the page limits, the test cases and their respective algorithms for trajectory planning appear in the Appendix.

II. System Overview

We illustrate the system software components and their interaction in Figure 1. The primary components are the safety kernel, which monitors data validity flows (Section IV), and the cooperative evaluator of service level that allows the vehicles to agree on the performing service level (Section V). Two other important components are the Local Dynamic Map (LDM) (Section III), which provides location information concerning all vehicles, and the Driving Manager (Section VI), whic determines the cooperative and autonomous trajectories.

The algorithms that we use for implementing the system in Figure 1, considers a system that includes a well-known set, *members*, of n identical vehicles with unique identifiers (network addresses) that are known to all. Each vehicle is an autonomous mobile entity with the capabilities for sensing, communicating, computing and controlling its motion speed and steering. All vehicles can access a common clock and they divide the time into *rounds* of exactly $ROUND$ time units, where $round_{now}$ denote the current round number. We assume (1) the existence of a *timely broadcast* protocol that delivers messages within a bounded time and that (2) all onboard processors do not crash.

III. Local Dynamic Map (LDM)

This component provides location information about the system objects, such a vehicles and roads. The aim here is not to provide an implementation of `ETSI TR 102 863`, rather we would like to have the most essential elements that are critical to the demonstration of the safety kernel. We do consider the position of intersections, lanes, and the vehicles and note that possible extensions can consider the complete environment, such as the weather, roadside units and obstacles. We note that the latter objects require periodic updates regarding their position. We assume that static elements of the map are preloaded with information about the entire driving task map that encode roads, lanes, intersections, to name a few. Using that task map, the position estimator can point out the vehicle whereabouts. The algorithms in Section A use this position estimation to discover the closest vertices on the task map and any additional information that is required for the path planning.

The position estimator implementation is divided into two parts: the local and the global position estimator. The local position estimator uses only onboard sensors for the position estimation of the vehicle and thus can be implemented in a real-time manner.

The abstract sensor combines the data of an (abstract) onboard range sensor and an (abstract) onboard odometry sensor. An ultra wide-band radio communication is used for indoor and outdoor localization. Such

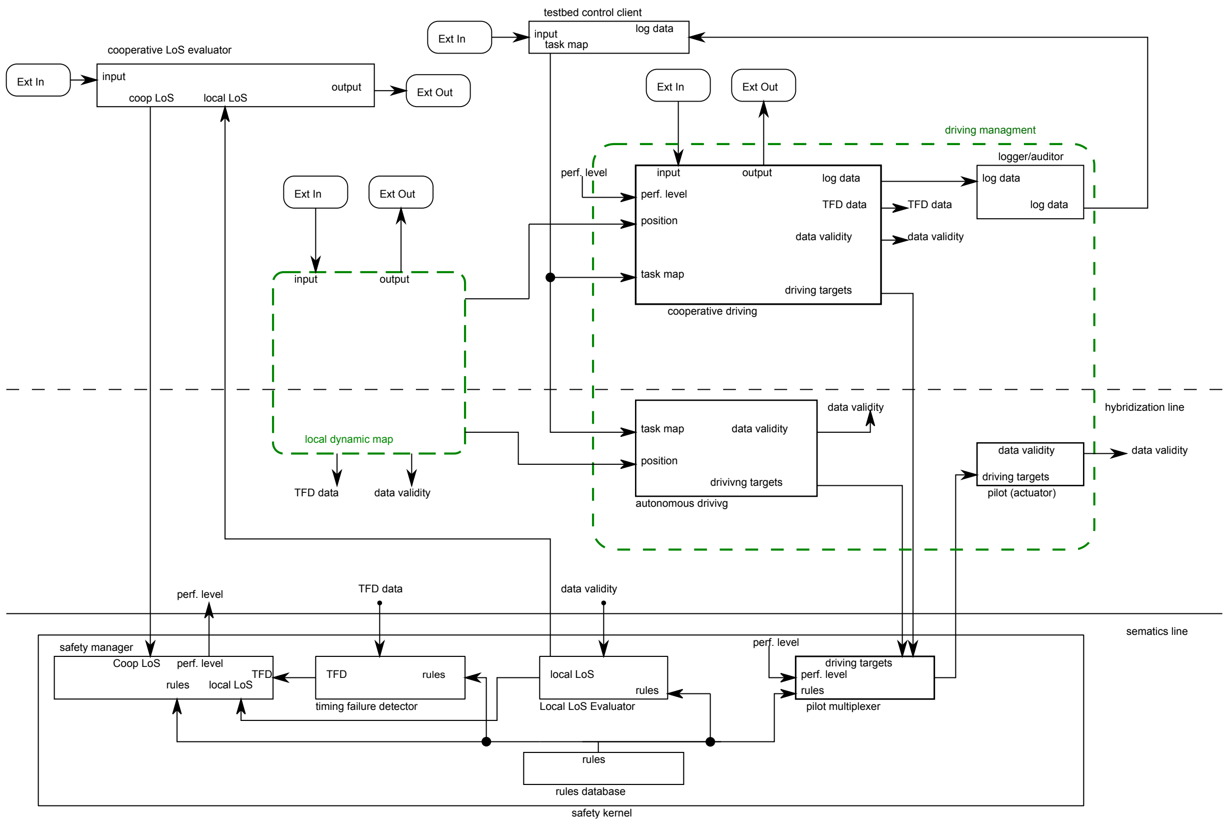

Fig. 1: System architecture for vehicular coordination with a safety kernel in the Gulliver test-bed.

a radio can determine the distance to any other radio in range with a ranging request. The test-bed uses three radios, called *anchors*, with known fixed positions. The (abstract) onboard range sensor sends ranging requests to the anchors in a periodic manner and returns the result, i.e., anchor id and distance, expected error and measurement time. Vehicles exchange their localization information, and use that for discovering the location of all nearby nodes.

The (abstract) onboard odometry sensor facilitates the motor controller to get information about speed and steering angle. Additionally, it can use an inertial measurement unit (IMU).

The Gulliver test-bed bases its localization mechanisms on external references (known anchors), and the exchange of this information among the vehicles for the sake of neighborhood discovery. The position database contains a position estimation for all vehicles from which sensor data has been received. The position database uses a Bayesian filter for each vehicle $v_i \in V$ to estimate the probability density function of v_i's position, speed and heading.

IV. The Safety Kernel and its Interfaces

The safety kernel [8] consists of a set of components that perform several safety-related task. These components must execute in a real-time environment, to ensure that LoS changes are performed timely and hence safely. In fact, the architectural concept defined in KARYON also requires that the components realizing the lowest level of service are also executed in a predictable way. The main tasks of the safety kernel include the collection of data validity information, the detection of timing failures in the execution of selected functional components, the ability to match the collected validity and timeliness information with predefined safety rules, the interaction with a cooperative service level evaluation component (which is not itself part of the safety kernel) and, finally, the activation of configuration changes whenever necessary, that is, whenever the level of service of some function has changed. In what follows we describe the data validity flow and the interactions between cooperative driving components (in Gulliver vehicles) and the safety kernel. These interactions are based on well-defined interface primitives that we list in Table I.

constants:	
$Validity \in \{Low, High\}$	
$Channel \in \{Pilot, Position\}$	
$PilotId \in \{CoopDrive, AutoDrive\}$	
$PositionId \in \{NetBasedId, NetFree\}$	
interface:	
$Write\_TFD(id)$	Sends to SK a heartbeat of module *id*
$Write\_validity(id, validity)$	Sends to SK the *validity* of module *id*
$Read\_validity(id)$	Reads module *id*'s validity
$Write\_performance\_level(los)$	Sends the effective service level *LoS*
$Read\_performance\_level()$	Returns the current effective service level
$Write\_coop\_los(los)$	Sends to SK the cooperative *LoS*
$Read\_coop\_los()$	Returns the cooperative service level
$Write\_local\_max\_los(los)$	Sends to SK the local maximum *LoS*
$Read\_local\_max\_los()$	Returns the local maximum *LoS*

TABLE I: The Safety Kernel (SK) interface.

The local level of service (LoS) evaluator receives sensor data validity from onboard sensors as well as from functional components that also produce relevant data, such as the cooperative LoS evaluator component. The safety kernel supports these operations using the primitives $Write\_validity$ and $Read\_validity$ for allowing components to send their validity data to be checked and evaluated by the safety kernel, and respectively, for the safety kernel to receive this information.

The local LoS evaluator determines the maximum possible level of service, as allowed by the locally collected validity information. This local level of service is used as an input to the cooperative LoS evaluator. The safety kernel supports these operations using the primitives $Write\_local\_max\_los$ and $Read\_local\_max\_los$ for allowing the safety kernel to send its local service level, and then for the cooperative LoS evaluator to receive it. The safety manager is the safety kernel component that receives the output from the cooperative LoS evaluator, that is, the agreed cooperative service level. This is done using the primitives $Write\_coop\_los$ (from the cooperative LoS evaluator side) and $Read\_coop\_los$ (from the safety kernel side). The safety manager periodically adjusts the performance level of functional components, depending on the agreed upon cooperative LoS. Note that the cooperative LoS evaluator may fail to reach agreement on the cooperative LoS, namely when communication is not possible. In this case, a timing failure will be detected by the safety kernel (using the TFD component), and the Local LoS Evaluator will determine the lowest LoS as the only possible. This will be fed to the safety manager, and a change to the lowest LoS will be performed, even without receiving any input from the cooperative LoS evaluator.

In order to adjust the performance level of the functional components, the safety manager uses the primitive $Write\_performance\_level$. The functional components, on the other hand, receive the performance level indication using the $Read\_performance\_level$ primitive.

The information flow of data validity also includes notifications on component timing failures. This specifically includes all network-centric mechanisms, such as localization, neighborhood discovery and cooperative LoS evaluator (as above-mentioned). The safety kernel architecture requires each of these components to periodically, and at a predefined minimal rate, send a heartbeat by calling the primitive $Write\_TFD$ and informing the safety kernel that the components are active, or have been able to complete the execution of some periodic task (like reaching agreement on the cooperative service level). For each of these components, the timing failure detector (TFD) component at the safety kernel checks the reception of the heartbeat timely.

Finally, within the safety kernel it is possible to include data multiplexers. These are necessary when a certain function is implemented by means of two distinct, and alternative, functional components. In this case, only one of the components must be used at any time, although both of them may be actively executing. The idea is to select the output from one of the alternative functional components, depending on the LoS that has to be imposed. In the considered example (see Figure 1), we use a multiplexer that switches the *PilotChannel* and has the inputs *CoopDrive* and *AutoDrive*. This multiplexer takes care of switching from the cooperative driving to the autonomous driving, ensuring that when the LoS degrades to the lowest one (autonomous driving mode), the pilot will consider the commands from the autonomous driving component rather than from the cooperative driving one. and this switch will be done in a timely, predictable way, after some timing failure or decreased data validity is observed.

V. Cooperative Evaluator of Service Level

The *cooperative evaluator of service level* is a fault tolerant distributed component that allows that vehicles to agree on a common service level according to which they are to operate with respect to their cooperative vehicular applications. At each round, vehicles propose their maximum service level that they support and let the cooperative evaluator of service level to jointly decide on the performance service level that they have to apply during the next round. The implementation therefore has to reach such a decision within a bounded time over failure-prone communication links.

We consider a set of local service levels, $\mathcal{MLOS}$, that is a totally ordered, where the minimum service level, *Autonomous_LoS*, allows autonomous driving. Moreover, we look into systems that has to agree on the highest service level that all vehicles can perform in during the next road. Thus, we define the cooperative service level as the minimum among the maximum local service level that each vehicle can support, i.e., $CLoS = \min_{\forall v \in member}\{\max\{\mathcal{MLOS}_v\}\}$.

We require the cooperative evaluator of service level to implement: (Agreement) all *members* decide on the same value $CLoS = \min_{\forall v \in member}\{\max\{\mathcal{MLOS}_v\}\}$, (Validity) the value $CLoS$ that is decided was proposed by at least one vehicle v, such that $CLoS \leq \max\{\mathcal{MLOS}_v\}$, as well as (Termination) all nodes decide (or abort) within a bounded time. (Recall that we assume that the system has access to a common clock and that all onboard processors do not crash.)

VI. The Driving Manager

The key objective of the safety kernel is facilitate predictable and safe coordination of smart vehicles that autonomously cooperate and interact. This system component manages the vehicle driving. It provides instructions for the pilot actuator, whether the vehicle follow autonomous or cooperative driving.

In cooperative driving, vehicle must be aware of the localization, speed, acceleration, intention, etc., of each other within a given horizon. We base our vehicular coordination on a variant of the model proposed by Ando *et al.* [4] that includes consensus. In each round, the vehicles: (1) *observe* the environment for a fixed period, (2) *compute* a local plan according to a deterministic algorithm, (3) *agree* on the cooperative service level and (4) *move*. We let the safety kernel determine the system perform level and select accordingly which output should be direct to the pilot actuator. Figure 2 depicts the interaction over a round of the cooperative and autonomous driving with the safety kernel and the local dynamic map (LDM).

In the *observation* phase, the vehicles timely collect information from the *local dynamic map* (LDM). The information includes the id, localization, speed, etc, of every member (when available). At the beginning of the *observation* phase, the vehicles write in the *timely failure detector* a heartbeat.

In the *computation* phase, the vehicles deterministically determine the plans for $round_{now}+1$ with the information collected by the current round. The plans include the target (or trajectory) for all service levels that the vehicle supports. We require a special trajectory, called *transition trajectory*, that guarantees a safe transition from a higher service level to the lowest service level in at most $transitionTime$ rounds. This trajectory is used in case that either the system fails due to communication, lack of agreements and/or low data validity, in the next round and, the safety kernel changes the performance level to the lowest one. The transition trajectory is intended for the $round_{now}+2$. Thus, to obtain the transition trajectory for the worst case scenario, we assume that the vehicles will perform the highest service level in $round_{now}+1$. For simplicity, we assume that the vehicles are able to follow the road and plans with high certainty.

In the *agreement* phase, the vehicles exchange the local maximum level of service so far in order to reach agreement on the cooperative LoS. We define the cooperative level of service as the minimum among the maximum local service levels. The driver manager indicates a low validity whenever on round $round_{now}$ vehicles have not reached an agreement (within $AGREE\_TIME$) on the cooperative level of service for $round_{now}+1$. We note this low validity can cause the safety kernel to lower the performance level for round $round_{now}+1$.

In the *movement* phase, all the trajectories are written in the *multiplexer pilot.* The safety kernel decides the trajectory that corresponds to the performance level according to the data validity. We require to the *autonomous driving* that when the performance level is lowered to the lowest, it sends to the *pilot multiplexer* the transition trajectory for $transitionTime$ rounds.

VII. Conclusion

Advances in the vehicular automotive have resulted in advance cooperative vehicular systems to improve the performance and comfort. These pose new safety-related challenges due to communication failures. The KARYON project has been defining solutions to simplify the design and maintain the operational safety in spite of communication failures, among others. We proposed an architecture that incorporates key components for the demonstration of the KARYON concept and describe the components in detail. We also proposed three test cases for validating the architecture and the KARYON concept.

Acknowledgments

This work was partially supported by the European UnionŠs Seventh Programme for research, technological development and demonstration, through project KARYON, under grant agreement No. 288195.

References

[1] Samer Ammoun and Fawzi Nashashibi. Design and efficiency measurement of cooperative driver assistance system based on wireless communication devices. *Transportation Research Part C: Emerging Technologies*, 18(3):408 – 428, 2010.

[2] Samer Ammoun and Fawzi Nashashibi. Design and efficiency measurement of cooperative driver assistance system based on wireless communication devices. *Transportation research part C: emerging technologies*, 18(3):408–428, 2010.

[3] Samer Ammoun, Fawzi Nashashibi, and Claude Laurgeau. An analysis of the lane changing manoeuvre on roads: the contribution of inter-vehicle cooperation via communication. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 1095–1100. IEEE, 2007.

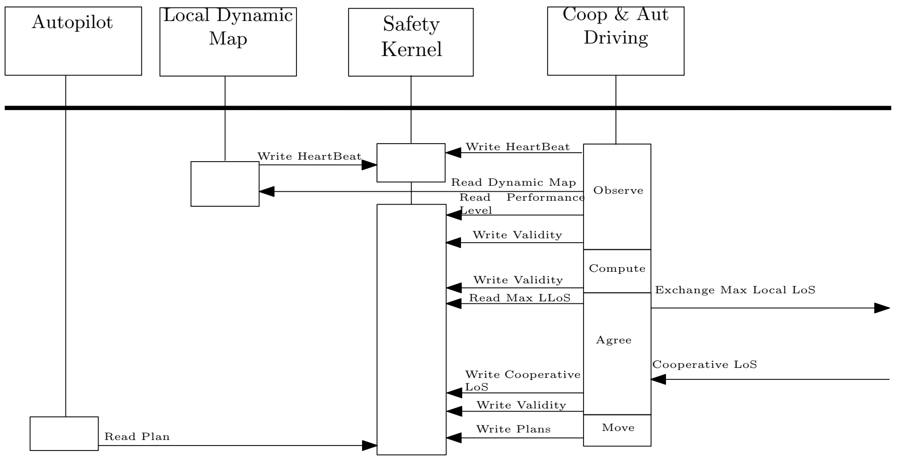

Fig. 2: Interaction of the cooperative and autonomous driving module with the other modules. (Sequence Diagram.)

[4] Hideki Ando, Yoshinobu Oasa, Ichiro Suzuki, and Masafumi Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *Robotics and Automation, IEEE Transactions on*, 15(5):818–828, 1999.

[5] Reza Azimi, Gaurav Bhatia, R Rajkumar, and Priyantha Mudalige. Intersection management using vehicular networks. In *Society for Automotive Engineers (SAE) World Congress*, 2012.

[6] Christian Berger, Erik Dahlgren, Johan Grunden, Daniel Gunnarsson, Nadia Holtryd, Anmar Khazal, Mohamed Mustafa, Marina Papatriantafilou, Elad M Schiller, Christoph Steup, et al. Bridging physical and digital traffic system simulations with the gulliver test-bed. In *Communication Technologies for Vehicles*, pages 169–184. Springer, 2013.

[7] Antonio Casimiro, Jörg Kaiser, Johan Karlsson, Elad Michael Schiller, Philippas Tsigas, Pedro Costa, José Parizi, Rolf Johansson, and Renato Librino. Brief announcement: Karyon: Towards safety kernels for cooperative vehicular systems. In Andréa W. Richa and Christian Scheideler, editors, *SSS*, volume 7596 of *Lecture Notes in Computer Science*, pages 232–235. Springer, 2012.

[8] António Casimiro, Jorg Kaiser, Elad M Schiller, Pedro Costa, José Parizi, Rolf Johansson, and Renato Librino. The karyon project: Predictable and safe coordination in cooperative vehicular systems. In *Dependable Systems and Networks Workshop (DSN-W), 2013 43rd Annual IEEE/IFIP Conference on*, pages 1–12. IEEE, 2013.

[9] Eric Chan, Peter Gilhead, Pavel Jelínek, and Petr Krejei. Sartre cooperative control of fully automated platoon vehicles. In *18th ITS World Congress*, 2011.

[10] Pedro Nóbrega Da Costa, João Craveiro, Antonio Casimiro, and José Rufino. Safety kernel for cooperative sensor-based systems. In Henrik Lönn and Elad Michael Schiller, editors, *ASCoMS@SAFECOMP*. HAL, 2013.

[11] J Alexander Fax and Richard M Murray. Information flow and cooperative control of vehicle formations. *Automatic Control, IEEE Transactions on*, 49(9):1465–1476, 2004.

[12] Javier Ibanez-Guzman, Stephanie Lefevre, Abdelkader Mokkadem, and Sylvain Rodhaim. Vehicle to vehicle communications applied to road intersection safety, field results. In *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*, pages 192–197. IEEE, 2010.

[13] Hossein Jula, Elias B Kosmatopoulos, and Petros A Ioannou. Collision avoidance analysis for lane changing and merging. *Vehicular Technology, IEEE Transactions on*, 49(6):2295–2308, 2000.

[14] Maziar E Khatir and Edward J Davison. Decentralized control of a large platoon of vehicles using non-identical controllers. In *American Control Conference, 2004. Proceedings of the 2004*, volume 3, pages 2769–2776. IEEE, 2004.

[15] Mitra Pahlavan, Marina Papatriantafilou, and Elad M Schiller. Gulliver: a test-bed for developing, demonstrating and prototyping vehicular systems. In *Proceedings of the 9th ACM international symposium on Mobility management and wireless access*, pages 1–8. ACM, 2011.

[16] Iakovos Papadimitriou and Masayoshi Tomizuka. Fast lane changing computations using polynomials. In *American Control Conference, 2003. Proceedings of the 2003*, volume 1, pages 48–53. IEEE, 2003.

[17] Gabriel Rodrigues de Campos, Paolo Falcone, and Jonas Sjöberg. Autonomous cooperative driving: a velocity-based negotiation approach for intersections crossing. In *16th International IEEE Conference on Intelligent Transportation Systems*, 2013.

[18] Steven E Shladover. Longitudinal control of automotive vehicles in close-formation platoons. *Advanced automotive technologies, 1989*, 1989.

[19] Steven E Shladover, Charles A Desoer, J Karl Hedrick, Masayoshi Tomizuka, Jean Walrand, W-B Zhang, Donn H McMahon, Huei Peng, Shahab Sheikholeslam, and Nick McKeown. Automated vehicle control developments in the path program. *Vehicular Technology, IEEE Transactions on*, 40(1):114–130, 1991.

[20] Srdjan S Stankovic, Milorad J Stanojevic, and Dragoslav D Siljak. Decentralized overlapping control of a platoon of vehicles. *Control Systems Technology, IEEE Transactions on*, 8(5):816–832, 2000.

[21] Youping Zhang, B Kosmatopoulos, Petros A Ioannou, and CC Chien. Using front and back information for tight vehicle following maneuvers. *Vehicular Technology, IEEE Transactions on*, 48(1):319–328, 1999.

Appendix

In this section we demonstrate the ability of the safety kernel to facilitate the implementation of cooperative advance driver assistance systems. We do not aim to design new cooperative advance driver assistance system, but show that the safety kernel can be used as a mechanism to achieve safe cooperative operation in spite of communication failures. We consider an autonomous and a fully cooperative service level in three test cases: 1) adaptive cruise control/vehicle platooning, 2) coordinated intersection crossing, and 3) coordinated lane-change.

The algorithm for each test case takes the input from the LDM and returns the trajectory plan for each service level. We compute the trajectory plan for both service levels based on the parameters depicted in Table II and assuming that the errors in the LDM are bounded, and let the safety kernel determine the performance level as well as the appropriate trajectory plan. The pseudo-code are structured using one part for the autonomous driving and one for cooperative driving.

Parameter	Description
maxAcceleration	Maximum vehicles' acceleration capability (bound)
maxDeceleration	Maximum vehicles' deceleration capability (bound)
maxSpeed	The road speed limit
cruisingSpeed	The cruising speed that all vehicles are aiming at. We assume that the vehicle's velocity may temporarily exceed their *cruisingSpeed* but never *maxSpeed*
length	Vehicle length
autonomousLoSHeadway	Minimum headway required in the autonomous level of service
fullyCooperativeLoSHeadway	Minimum headway required in the highest level of service
transitionTime	The transition time (number of rounds) between the highest service level and the lowest service level

TABLE II: Test-bed Parameters

A. Adaptive Cruise Control and Vehicle Platooning.

This application adjusts the headway between vehicles according to the data validity and the performance level. Namely, the safety kernel adjusts the inter-vehicle distance according to the performance level. The adaptive cruise control has been studied in [14], [20], [21]. The main pieces of information in this vehicular application are the relative distance to, and the speed of the vehicle ahead. We assume that the information that is coming from onboard sensors, such as the inter-vehicle distance, has sufficiently high quality. Vehicle platooning [9], [11], [18], [19] uses vehicle to vehicle communication for agreeing on the joint (cooperative) cruising speed.

In this test case, the highest service level corresponds to the cooperative application of vehicular platooning that is based in [19] and the lowest service level corresponds to the autonomous application of adaptive cruise control based on [14]. We refer to [14] and [19] for the formal proofs of these algorithms. For simplicity, we do not consider external factors, such as friction, mass, and road condition. In this context, the vehicular platooning depends only on the speed and relative position of each other vehicle. Meanwhile, the adaptive cruise control depends only on the speed and relative position of the vehicle in front. On a broader scope than KARYON, cooperative vehicle systems have the task of determining the participating vehicle set that is forming a platoon since different views may result in conflicting trajectories.

The autonomous trajectory (adaptive cruise control) in Algorithm 1 is computed in Line 16. Meanwhile, the cooperative trajectory (vehicular platooning) and transition trajectory are computed in Lines 5-15. For the vehicular platooning, the speeds are determined in the order in which they appear in the cluster starting with the leading vehicle.

One can show that Algorithm 1 is collision-free provided that all vehicles can follow the trajectories with a high level of precision. Indeed, the system is stable and safe if it is performing adaptive cruise control and continue in adaptive cruise control by [14]. Similarly, the system is safe and stable if it is performing platoon and continue performing platoon by [19]. Furthermore, the transition between lower service level to higher service level is safe as the vehicles have complete knowledge of the members. The risky part is the transition between the highest service level to the lowest service level. However, as they compute the transition trajectory at $round_{now}-1$ and they follow the transition trajectory for $transitionTime$ rounds with a high level of precision, the transition is safe.

The demonstration plan includes at most three vehicles and a single vehicle cluster or platoon. When considering a trajectory plan that includes several clusters, one must take into account their dependencies among them. However, these considerations are orthogonal to this demonstration of the safety kernel. As the communication can be lost, one has to consider the worst case scenario for the transition trajectories. Thus, the transition time becomes the fundamental issue for the correct design of vehicular systems. The problem is more evident when the number of platoons is greater than one. Indeed, one can maintain safety in the transition time by keeping a minimum distance between the leading vehicles of two platoons p_1, p_2 with p_1 in front of p_2 of at least $(|p_1|length)$ times the distance that corresponds to the lowest level of service where length is the largest vehicle length. This

Algorithm 1: Adaptive Cruise Control and Vehicle Platooning. (Each vehicle c executes it.)

```
Input: $member = \{u, v, w\}$: the set of the three vehicles in the domain.
Input: $vehicle = [\langle id, frontDistance, speed\rangle]_{id \in members}$ a vector of vehicles in which each record refers to the distance to
        the vehicle in front and speed at the beginning of $round_{now}$.
Output: $preliminaryPathPlan[autonomousLoS, fullyCooperativeLoS, transition] : speed$ a vector with the preliminary
        path plans of $c$.
Data: $speed[fullyCooperativeLoS, transition] : [\langle id, speed\rangle]$ speed of each vehicle.
1  function $non\_coop\_speed()$: determines the speed of $c$ given $autonomousLoSHeadway$, the distance to the vehicle in
   front and its speed, e.g., using the algorithm given in [14];
2  function $coop\_speed(v, frontVehicle, leadingVehicle)$: determines the speed of $v$ given $fullyCooperativeLoSHeadway$,
   the vehicle in front and leading vehicle in the platoon, e.g., using the algorithm given in [19];
3  function $tran\_speed(v, vehicleInFront, positionInPlatoon, speed)$: determines the speed of $v$ for the transition from
   $fullyCooperativeLoSHeadway$ to the $autonomousLoSHeadway$ in $transitionTime$ rounds given the vehicle in front and
   its position in the platoon. To obtain the transition trajectory for $round_{now} + 2$ of the worst case scenario we assume that
   all the vehicles will perform their trajectory $speed[fullyCooperativeLoS]$ for one round;
4  let $Platoon$ be the sorted set of vehicles on the lane where the vehicle with smallest id is the leading vehicle;
5  foreach $vehicle \in Platoon$ do
6      if $vehicle$ is the first in platoon then
7          let $speed[fullyCooperativeLoS][vehicle.id]$ be the $cruisingSpeed$ of $vehicle$;
           /* Transition trajectories maintain constant speed of the leading vehicles */
8          $speed[transition][vehicle.id] = speed[fullyCooperativeLoS][vehicle.id]$;
9      else
10         let $leading$ and $next$ be the leading and front of $vehicle$ in $Platoon$;
11         $speed[fullyCooperativeLoS][vehicle.id] = [\langle vehicle.id, coop\_speed(vehicle, next, leading)\rangle]$;
           /* The transition trajectory guarantees a safe transition */
12         let $position$ be the position of $vehicle$ in $platoon$;
13         $speed[transition][vehicle.id] = [\langle vehicle.id, tran\_speed(vehicle, next, leading, position, speed)\rangle]$;
14 $preliminaryPathPlan[fullyCooperativeLoS] = speed[fullyCooperativeLoS][c.id]$;
15 $preliminaryPathPlan[transition] = speed[transition][c.id]$;
16 $preliminaryPathPlan[autonomousLoS] = non\_coop\_speed()$;
17 return $preliminaryPathPlan$
```

distance allows the platoons to return to cruise control independently as no communication may be available. Figure 3 depicts the transition between highest service level and lowest service level. However, external events may create cascade effects.

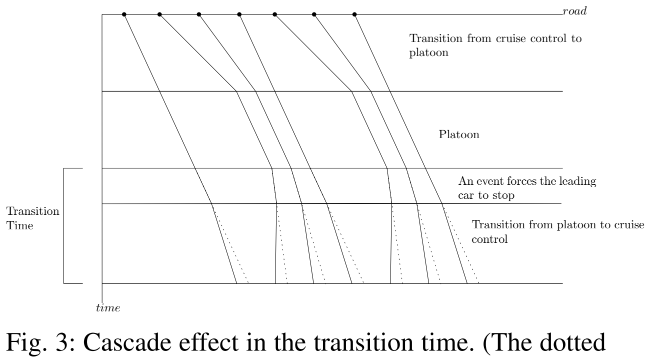
road
Transition from cruise control to platoon
Platoon
An event forces the leading car to stop
Transition from platoon to cruise control
Transition Time
time

Fig. 3: Cascade effect in the transition time. (The dotted points depicts the computed trajectories.)

B. Intersection Crossing

This application shows how the safety kernel allows vehicles to cross the intersection with minimal waiting time when the data validity is high and a conservative intersection crossing given the right to the right lane when the data validity is low. We use the first-come first served approach for deciding who crosses the intersection first. We break symmetry using the right hand rule, i.e., we assume that the roads have a given priority. Such symmetry breaking techniques are essential especially at low service levels for which there are no communication-based coordination guarantees. Thus, the highest service level is based on [17] and the lowest is based on the the right hand rule. We refer to [17] for the formal proof. We note that our approach is extendable to systems that further consider the details and optimality of trajectory planning [1], [5], [12], [17].

This test case considers an intersection of the roads that have a single conflicting directions. We define the *critical zone* as the area of high risk of collision. This area corresponds to the intersection, as well as, the road sections next to the intersection; see Figure 4. We assume that the road sections of the critical zone are at least the distance that every vehicle requires to stop completely without entering the intersection. We define the *negotiating zone* as the road sectors next to the critical zone that are in the communication range. We refer to these two zones as the vital zone. The pseudocode is depicted in Algorithm 2.

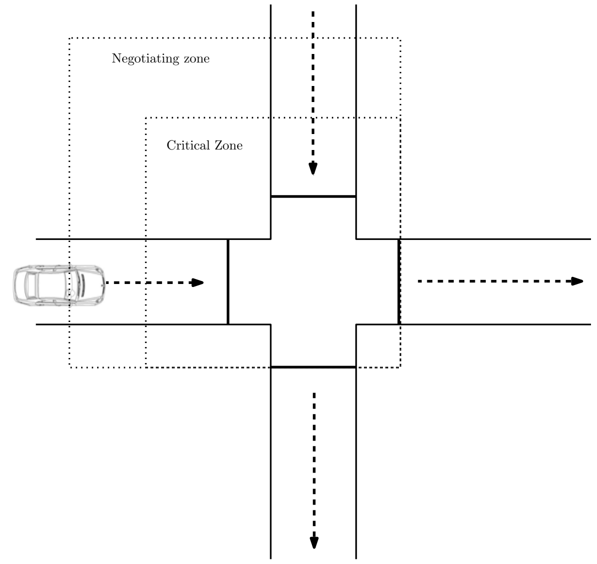

Fig. 4: Transition periods between highest service level and the lowest service level.

The algorithm considers the distance to the critical zone, speed and lane of the vehicles which is provided by the LDM. We note that the priority in the input is determined by the driving management and refers to the priority obtained in the previous round.

The main idea of Algorithm 2 is to maintain the intersection with at most one vehicle at every time. This is guaranteed by calculating consistent priorities; Lines 4 to 14. We give the highest priority to the vehicle in the right lane. One can show that the algorithm is safe by proving that exactly one vehicle has the highest priority and priorities are consistent between rounds and vehicles. Lines 16 and 18 computes the speed of c according to its priority. If c is in the critical zone and there is the need to lowered the level of service, c will cross the intersection at the cruise speed; line 21. However, if this happens when c is in the negotiating zone, it will stop before crossing the intersection in at most $transitionTime$ rounds; line 19. The autonomous trajectory is determined in lines 23-27. This trajectory lets the vehicle cross if there is not other vehicle in the vital zone, or when the vehicle is on the right lane inside the vital zone and the other vehicle is stopped completely in the vital zone.

Our demonstration plan includes two vehicles. When considering a trajectory plan that includes several vehicles, one must take into account their priorities. However, these considerations are orthogonal to this demonstration of the safety kernel.

C. Coordinated Lane-Change

This application schedules a safe lane-change maneuver by adjusting the inter-vehicle distances in the subject lane according to the data validity and the performance level that the safety kernel determines. The lane change maneuver has been studied in [13], [16] in the non-cooperative context and [2], [3] in the cooperative context.

We assume that the vehicles are able to follow the paths to change lanes and when, it needs to abort the maneuver, return to the original lane [16]. Algorithm 3 considers the accessibility of (primitive) applications for maintaining appropriate inter-vehicle distance, e.g., adaptive cruise control and vehicular platooning. The algorithm considers a vehicle that is changing lane to a *subject (target) lane*, as well as the vehicle projection on the subject lane. We define the *projection* as the intersection between the bisector ray of the vehicle position and the subject lane.

The autonomous lane change algorithm is based on [16] and the cooperative lane change algorithm is based on [3]. The algorithms consider speed and lane for each vehicle. The system computes this by using the vehicle positions and road map that LDM provides.

The main idea of the algorithm is the following: Consider a vehicle c that aims to change lanes. Assume that the two closest vehicles on the subject lane are c_1 and c_2 with c_2 following c_1. W.l.o.g, we may assume that the projection of c is in-between c_1 and c_2. At first, they start preparing to open a space by performing an adaptive cruise control or platooning with c_1, c_2 and the projection of c according to the information quality; line 4. They will start performing the maneuver if the space $|c_1c_2|$ is at least the distance that corresponds to $2 fullyCooperativeLosHeadway + length$ and the performance level of c_1 and c_2 is the highest; line 8. However, if the performance level of c_1 and c_2 is the lowest and $|c_1c_2|$ is at least the distance that corresponds to the $2 autonomousLosHeadway + length$, they will start performing the maneuver; line 10. While they are performing the maneuver, they check for hazardous situation. If there is one hazardous situation that can avoid completing the maneuver, they will abort it; line 14. Otherwise, they will finish it.

Our demonstration plans include three vehicles. This allows us to determine the vehicles involved in the maneuver. When considering an implementation that is more extensive than this pilot demonstration of the safety kernel one has to consider more conflict scenarios. For example, a conflicting situation may arise when two vehicles are aiming to change lanes simultaneously in the same spot. As a future direction, we propose to monitor the participating vehicle sets in the maneuver

Algorithm 2: Intersection Crossing. (Each vehicle c executes it.)

```
Input: $member = \{u, v\}$: the set of the two vehicles inside a vital zone.
Input: $vehicle = [\langle id, distanceToCriticalZone, speed, lane, priority \rangle]_{id \in members}$ a vector of vehicles in which each record
       refers to the distance to the critical zone, speed, lane and priority.
Output: $preliminaryPathPlan[autonomousLoS, fullyCooperativeLoS, transition]$ a vector of the vehicle speed so that the
        vehicle with highest priority safely cross before the vehicle with lowest priority.
1  function $conflictFreePath(priority)$ computes the path for $c$ as follows: If $priority = HighPriority$, $c$ will cross before,
   otherwise $c$ will cross after. The path guarantees that the vehicle with lower priority enters the critical zone after the
   vehicle with higher priority has left. The output of this function is the speed for the next round;
2  function $stopAtCriticalZone(r)$ computes the path for $c$ so that it stops just before the critical zone in at most $r$ rounds.
   The output of this function is the speed for the next round;
3  let $b = \{u, v\} \setminus \{c\}$;
4  let $priority = lowestPriority$;
5  if c is in critical zone and b is not in critical zone then
6      $priority = highestPriority$;
7  else if c is in critical zone and b is in critical zone then
8      $priority = emergencyStop$;
9  else if c is not in critical zone and b is in critical zone then
10     $priority = lowestPriority$;
11 else if c arrives at the intersection before b at the current speeds then
12     $priority = highestPriority$;
13 else if c arrives at the intersection at the same time as b at the current speeds and c is on the right lane then
14     $priority = highestPriority$;
   /* Computes the fully Cooperative LoS trajectory */
15 if $priority = emergencyStop$ then
16     $preliminaryPathPlan[fullyCooperativeLoS] = stopAtCriticalZone(0)$;
17 else
18     $preliminaryPathPlan[fullyCooperativeLoS] = conflictFreePath(priority)$;
   /* Computes the transition trajectory */
19 if c is in the negotiating zone then $preliminaryPathPlan[transition] = stopAtCriticalZone(transitionTime)$;
20 ;
21 else $preliminaryPathPlan[transition] = preliminaryPathPlan[fullyCooperativeLoS]$;
22 ;
   /* Computes the autonomous LoS trajectory */
23 if c is in critical zone and b is in critical zone then $preliminaryPathPlan[autonomousLoS] = stopAtCriticalZone(0)$;
24 ;
25 else if c is in critical zone and b is not in critical zone or c is on the right lane and b is stopped then
26     $preliminaryPathPlan[autonomousLoS] = conflictFreePath(highestPriority)$;
27 else $preliminaryPathPlan[autonomousLoS] = stopAtCriticalZone(\infty)$;
28 ;
29 return $preliminaryPathPlan$
```

and maintain the set consistencies, e.g., through state replication and consensus protocols.

As in the adaptive cruise control, the transitions from the highest service level to the lowest service level while they are performing is the most risky maneuver. This is specially true when the communication fails. For example, assume that the vehicles are performing the maneuver and there is the need to abort due to a hazardous situation. However, if communication broke just before aborting, the vehicle must detect the situation with its local sensory information.

Algorithm 3: Lane Change. (Each vehicle c executes it.)

```
Input: $member = \{u, v, w\}$: the set of the three vehicles in the domain.
Input: $vehicle = [\langle id, speed, lane, state \rangle]_{id \in members}$ a vector of vehicles in which each record refers speed, lane and state
        in the maneuver at the end of $round_{now} - 1$.
Output: $preliminaryPathPlan[autonomousLoS, fullyCooperativeLoS, transition]$ : a vector of $c$'s speed.
1  let $u$ be in front of $v$ and $w$ be the vehicle aiming to change lanes;
2  let $w'$ be the projection of $w$ on $u.lane$;
3  let $speedPlan$ be the preliminary path plan obtained from executing Algorithm 1 (Adaptive cruise control) with input
   $\{u, w', v\}$;
4  $preliminaryPathPlan = speedPlan$;
5  .
6  if $w'.lane \neq v.lane$ then
7      if the inter-vehicle distances between members in $\{u, w', v\}$ are at least $fullyCooperativeLoSHeadway$ and
       $v.LoS = fullyCooperativeLoS$ then
8          Execute fully cooperative LoS lane change maneuver;
9      if the inter-vehicle distances between members in $\{u, w', v\}$ are at least $autonomousLoSHeadway$ then
10         Execute autonomous LoS lane change maneuver;
11 else
12     if the maneuver is inclomplete completed and the level of service of v is lowered then
13         if it is not safe to complete the maneuver then
14             Abort the lane change maneuver to return to the original lane;
15 return $preliminaryPathPlan$;
```