

Cybercar cooperation for safe intersections

Laurent Bouraoui, Stéphane Petti, Anis Laouiti, Thierry Fraichard and Michel Parent

INRIA Rocquencourt

78153 Le Chesnay Cedex, France

firstname.lastname@inria.fr

Abstract—The paper addresses the problem of motion autonomy of Cybercars across a urban intersection. Cybercars are small electric city vehicles aimed at navigating autonomously. In the context of a crossing, the motion generation together with its safety are critical issues. The proposed approach to the problem lies in the coupling of perception and planning capabilities. A new car to car communication algorithm provides necessary information to a trajectory planner capable of iteratively generate safe trajectories within a dynamic environment in order to drive Cybercars safely through the intersection. The main contributions of this work are the development and integration of these modules into one single application, considering explicitly the constraints related to the environment and the system and to provide an original answer to the problem of intelligent crossing.

I. INTRODUCTION

In many urban environments, private automobile use has led to severe problems with respect to congestion, pollution, and safety. A large effort has been put in industrial countries into developing new types of transportation systems, the Cybercars as an answer to this problem [10]. Cybercars are city vehicles with fully automated driving capabilities. Such autonomous systems cannot be realized without using several capabilities designed to work together in a single application. Indeed, to safely navigate, the system will have to perceive its environment, plan its trajectory to the goal and finally execute it. There are several constraints the planning scheme must consider. At first, the dynamic nature of the urban environment (pedestrians, other cars,...) imposes on the navigation scheme, a real time constraint which is the time that the system has to take a decision. At second, a complex system as a Cybercar is constrained by its (nonholonomic) kinematics as well as its dynamics. The trajectory planning scheme must explicitly account for these different constraints in order to safely move the robot to its goal. As for the perception, there are situations where the embedded perceptive capabilities might not suffice. The interesting case of a urban intersection for instance, requires a cooperation between the Cybercars in order to safely cross the intersection. This problem lies at the heart of our paper. In this paper we present an original approach toward autonomous intersection crossing. Our approach lies in a Car to Car communication protocol allowing cooperation and sharing information between the two Cybercars. Furthermore an original planning algorithm is used and implemented on a Cybercar in order to safely

generate the control inputs that will allow both Cybercars to safely cross the intersection. Nowadays several projects have been promoted by the E.U. commission in order to increase road safety by using car to car communication (C2C) coupled with perception systems. We can found project focusing on intersections like INTERSAFE a Prevent project[5]. The main purpose of this project is to develop advanced sensor systems and algorithms to complement C2C communication, so it will be possible to warn the driver of potentially hazardous situations. Other projects focus on C2C Communication for road safety in general, we can found for example eSafety program [3], aims at accelerating the development, deployment and use of systems that use information and communication technologies to increase road safety. Finally, the Car2Car Communication Consortium [4] addresses similar problems, focussing on the creation of a standard for active safety applications. We detail in §II the planning scheme and §III the car-to-car communication system. In §IV we present the integration of both modules and the results of experiments performed on a Cybercar, the Cycab. Finally we draw some conclusions and discuss the future work in §V.

II. PLANNING IN DYNAMIC ENVIRONMENT

A. Introduction

Planning in an environment cluttered with moving obstacles implies to plan under a real time constraint. Indeed, a robotic system placed in a dynamic environment has a limited time only to compute the motion plan to be executed. If the execution of the plan could begin at an arbitrary time, there would not be any problem. This is however not the case. In a real dynamic environment, a robotic system cannot safely remain passive as it might be collided by a moving obstacle. This time the system has to make its decision is the *decision time constraint*, δ_d and is therefore a real-time constraint imposed by the environment.

Early work addressing the problem of navigation within dynamic environments, rely on *reactive* approaches. These methods consist in a local exploration of the velocity space, *i.e.* the set of all possible velocities of the robot, in order to find the proper velocity to be applied during the next time step. For robots controlled in speed and steering angle, the velocity output can be directly executed by the robot, which makes these techniques particularly efficient. Their local nature exhibit however strong limitations in terms of conver-

gence. Besides, complex kinematic or dynamic constraints are difficult to handle in a general way, without resorting to crude approximations. Recently, *deliberative* methods accounting for time constraints, have been also presented. Deliberative methods, also referred to as motion planning methods, consist in calculating a priori a complete motion plan to the goal. Some approaches based on improved dynamic programming techniques, have been presented [8]. These methods however are restricted to low dimension problems and cannot account for general kinematic or dynamic system's constraints. Recent random techniques have been presented with very fast and impressive results for higher dimension problems [2]. The real time constraint is however never explicitly considered and therefore no computation time upper bound can be guaranteed. Due to the complexity of the motion planning problem, sometimes referred to as "the curse of dimensionality", there is little hope that within an arbitrary bounded time, a complete plan to the goal might be found. Therefore, the proposed approach to the problem is a Partial Motion Planner (PMP) that guarantees a bounded computation time at the expense of its completeness, *i.e.* the guarantee to plan a complete trajectory to the goal.

B. Notations

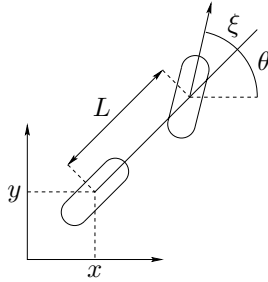


Fig. 1. The car-like vehicle \mathcal{A} (bicycle model).

Let \mathcal{A} denote the car-like robot placed in a workspace \mathcal{W} (fig. 1). The model of the car-like robot used in the planning strategy is described by the following differential equation :

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\xi} \end{bmatrix} = \begin{bmatrix} v_r \cos \theta \\ v_r \sin \theta \\ v_r \frac{\tan \phi}{L} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \alpha + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \gamma \quad (1)$$

This equation is of the form $\dot{s} = f(s, u)$ where $s \in \mathcal{S}$ is the state of the system, \dot{s} its time derivative and $u \in \mathcal{U}$ a control. \mathcal{S} is the state space and \mathcal{U} the control space of \mathcal{A} . A state of \mathcal{A} is defined by the 5-tuple $s = (x, y, \theta, v, \xi)$ where (x, y) are the coordinates of the rear wheel, θ is the main orientation of \mathcal{A} , v is the linear velocity of the rear wheel, and ξ is the orientation of the front wheels. A control of \mathcal{A} is defined by the couple

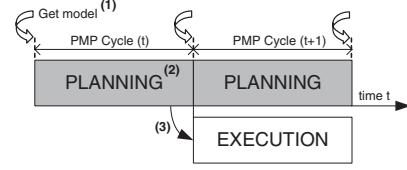


Fig. 2. Partial Motion Planning architecture.

(α, γ) where α is the rear wheel linear acceleration and γ the steering velocity, with $\alpha \in [\alpha_{min}, \alpha_{max}]$ (acceleration bounds), $\gamma \in [\gamma_{min}, \gamma_{max}]$ (steering velocity bounds), and $|\xi| \leq \xi_{max}$ (steering angle bounds). L is the wheelbase of \mathcal{A} , $\mathcal{A}(s)$ is the subset of \mathcal{W} occupied by \mathcal{A} at a state s . Let $\phi \in \Phi: [t_0, t_f] \mapsto \mathcal{U}$ denote a control input, *i.e.* a time-sequence of controls. Starting from an initial state s_0 , at time t_0 , and under the action of a control input ϕ , the state of the system \mathcal{A} at time t is denoted by $s(t) = \phi(s_0, t)$. An initial state and a control input define a trajectory for \mathcal{A} , *i.e.* a time sequence of states.

C. Partial Motion Planner (PMP) Algorithm

The partial motion planner (PMP) is a motion planning strategy that explicitly accounts for the real time constraint imposed by the environment. Besides, in a real environment, the model of the future can be predicted over a limited time only δ_v . Therefore, PMP is structured around a constant planning cycle (PMP cycle in fig. 2) of duration δ_c , in order to be able to regularly get an update of the model. This cycle duration must in fact fulfil the requirement that $\delta_c = \min(\delta_d, \frac{1}{2}\delta_v)$. The main cycle of PMP is described as follows, starting at time t_i :

- 1) Get an updated model of the future.
- 2) The state-time space of \mathcal{A} is searched using an incremental exploration method that builds a tree rooted at the state $s(t_{i+1})$ with $t_{i+1} = t_i + \delta_c$.
- 3) At time t_{i+1} , the current iteration is over, the best partial trajectory ϕ_i in the tree is selected according to given criteria (safety, metric) and is fed to the robot that will execute it from now on. ϕ_i is defined over $[t_{i+1}, t_{i+1} + \delta_{h_i}]$ with δ_{h_i} the trajectory duration.

After completion of a planning cycle, it is most likely the planned trajectory of time horizon δ_h is partial. Thus, the PMP algorithm iterates over a cycle of duration δ_c , as depicted in figure 2, until the goal is reached. The algorithm operates until the robot reaches a neighbourhood of the goal state. In case the planned trajectory has a duration $\delta_h < \delta_c$, the cycle of PMP can be set to this new lower bound or the navigation (safely) stopped. In practice however, the magnitude of δ_h is much higher than δ_c .

In our work, we use a sampling based incremental method. Sampling based methods avoid the complete space representation by probing the space by mean of a collision detection

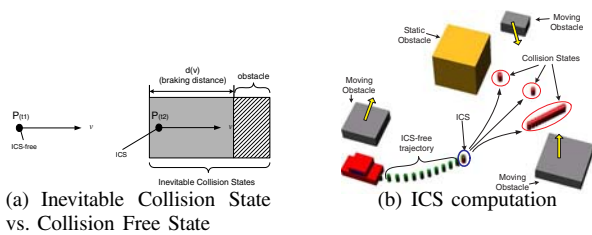


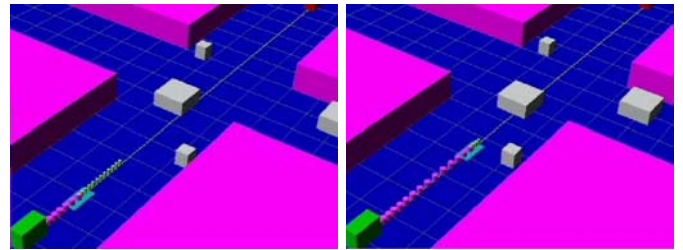
Fig. 3. Inevitable Collision States (ICS).

module. In our approach however, the usual geometric collision checker is replaced with an inevitable collision state checker described in the next part. This original module allows to deterministically extend the tree to the goal while insuring avoidance of static and dynamic obstacles. This method is incremental in order to be interrupted any time. The control space of our system is reduced to the set of bang bang controls $\mathcal{U}=(\alpha, \gamma)$ with $\alpha \in [\alpha_{min}, 0, \alpha_{max}]$ and $\gamma \in [\gamma_{max}, 0, \gamma_{min}]$. The exploration of the state-time space consists in building incrementally a tree as follows: the closest state s_c to the goal is selected. A control from \mathcal{U} is applied to the system during a fixed time (integration step). In case the new state s_n of the system is safe, this control is valid. The operation is repeated over all control inputs and finally the new state, safe and closest to the goal, is finally selected and added to the tree.

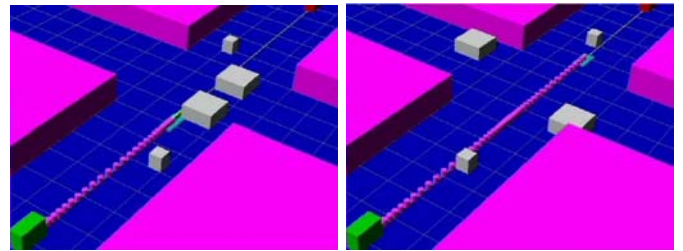
D. Safety Issues

Like every method that computes partial motion only, PMP has to face a safety issue: since PMP has no control over the duration of the partial trajectory that is computed, what guarantee do we have that \mathcal{A} will never end up in a critical situations yielding an inevitable collision? We need however to define the safety we consider. In figure 3(a) we consider a selected milestone of a point mass robot P with non zero velocity moving to the right (a state of P is therefore characterized by its position (x, y) and its speed v). Depending upon its state there is a region of states for which P , even though it is not in collision, will not have the time to brake and avoid the collision with the obstacle. As per [1], it is an Inevitable Collision State (ICS). In this paper, we refer to a safe state as ICS-free.

In general, computing ICS for a given system is an intricate problem since it requires to consider the set of all the possible future trajectories. To compute in practice the ICS for a system such as \mathcal{A} , it is taken advantage of the approximation property established in [1]. This property shows that a conservative approximation of the ICS can be obtained by considering only a finite subset \mathcal{I} of the whole set of possible future trajectories. For our application we consider the subset \mathcal{I} of braking trajectories obtained by applying respectively constant controls $(\alpha_{min}, \gamma_{max})$, $(\alpha_{min}, 0)$, $(\alpha_{min}, \gamma_{min})$ until the system has



(a) Cybercar approaching a crossing (b) Cybercar braking in order remain safe



(c) Cybercar stopped in front of a moving obstacle (d) Cybercar accelerating and crossing the intersection

Fig. 4. Autonomous intersection using PMP

stopped. Once it is still, it is checked to be collision free (*i.e.* over a trajectory obtained by applying constant $(0,0)$ controls) until the end of the PMP cycle. In the PMP algorithm, every new state is similarly checked to be an ICS or not over \mathcal{I} . In case all trajectories are in collision, this state is an ICS and is not selected (see fig. 3(b)).

A safe trajectory consists of safe states. However, a practical problem appears when safety has to be checked for the continuous sequence of states defining the trajectory. In order to solve this problem and further reduce the complexity of the PMP algorithm, we presented in [11] a property that simplifies the safety checking for a trajectory. This property is important since first, it proves a trajectory is continuously safe while the states safety is verified discretely only, and second it permits a practical computation of safe trajectories by integrating a dynamic collision detection module within existing incremental exploration algorithms, like A* or Rapidly-Exploring Random Tree (RRT).

In the context of a crossing, the constrained geometry is used in order to use PMP to generate safe longitudinal control inputs only. Simulation results (see fig. 4) show a 1D safe trajectory generation for a car crossing an intersection. The car is controlled in acceleration considering its own dynamics as well as a knowledge of the surrounding moving obstacles. In such an example therefore, the lateral control can easily rely on a specific method that will benefit from the environment structure, as a vision-based street border following algorithm.

III. PERCEPTION IN URBAN ENVIRONMENTS : THE OLSR APPROACH

In order to detect moving obstacles one may rely on the information given by the attached sensors (like radars, laser sensors). In this article we propose another approach based on the information exchange between vehicles. In fact, vehicles equipped with wireless medium in a crossing may form a mobile ad hoc network, and then may communicate and exchange their information. Several protocols were designed to enable communication for wireless ad hoc networks. One of them is the OLSR protocol. In the following we give a short description of the protocol then we describe the architecture we use to merge the PMP and the communication blocks.

This section describes the main features of the *OLSR* (Optimized Link State Routing) protocol.

OLSR is an optimization of a pure link state routing protocol. It is based on the concept of *multipoint relays* (*MPRs*) [12]. First, using *multipoint relays* reduces the size of the control messages: rather than declaring all links, a node declares only the set of links with its neighbours that are its “*multipoint relay selectors*”. The use of *MPRs* also minimizes flooding of control traffic. Indeed only *multipoint relays* forward control messages. This technique significantly reduces the number of retransmissions of broadcast control messages [6], [12]. The two main *OLSR* functionalities, Neighbour Discovery and Topology Dissemination, are now detailed. In our context the neighbours and nodes could be considered as vehicles or obstacles.

A. Neighbor Discovery

Each node must detect the neighbor nodes with which it has a direct link.

For this, each node periodically broadcasts *Hello* messages, containing the list of neighbours known to the node and their link status. The link status can be either *symmetric* (if communication is possible in both directions), *asymmetric* (if communication is only possible in one direction), *multipoint relay* (if the link is symmetric and the sender of the *Hello* message has selected this node as a *multipoint relay*), or *lost* (if the link has been lost). The *Hello* messages are received by all 1-hop neighbours, but are not forwarded. They are broadcasted once per refreshing period called the “*HELLO_INTERVAL*”. Thus, *Hello* messages enable each node to discover its 1-hop neighbours, as well as its 2-hop neighbours. This neighbourhood and 2-hop neighbourhood information has an associated holding time, the - “*NEIGHBOR_HOLD_TIME*”, after which it is no longer valid.

On the basis of this information, each node independently selects its own set of *multipoint relays* among its 1-hop neighbours in such a way all 2-hop neighbours of m have *symmetric* links with $MPR(m)$. This means that the *multipoint relays* cover (in terms of radio range) all 2-hop neighbours (Figure 5)

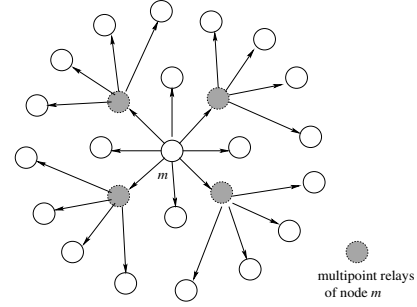


Fig. 5. Multipoint relays of node m

One possible algorithm for selecting these *MPRs* is described in [12]. The *multipoint relay* set is computed whenever a change in the 1-hop or 2-hop neighbourhood is detected. In addition, each node m maintains its “*MPR selector set*”. This set contains the nodes which have selected m as a *multipoint relay*. Node m only forwards broadcast messages received from one of its *MPR selectors*.

B. Topology Dissemination

Each node of the network maintains topological information about the network obtained by means of *TC* (*Topology control*) messages. Each node m selected as a *multipoint relay*, broadcasts a *TC* message at least every “*TC_INTERVAL*”. The *TC* message originated from node m declares the *MPR selectors* of m . If a change occurs in the *MPR selector* set, the next *TC* can be sent earlier. The *TC* messages are flooded to all nodes in the network and take advantage of *MPRs* to reduce the number of retransmissions. Thus, a node is reachable either directly or via its *MPRs*. This topological information collected in each node has an associated holding time “*TOP_HOLD_TIME*”, after which it is no longer valid.

The neighbour information and the topology information are refreshed periodically, and they enable each node to compute the routes to all known destinations. These routes are computed with Dijkstra’s shortest path algorithm [14]. Hence, they are optimal as concerns the number of hops. The routing table is computed whenever there is a change in neighbourhood or topology information.

C. OLSR adaptation to vehicle communications

The proactive behaviour of OLSR protocol is appropriate for vehicle communications. In fact, communicating car networks should be created quickly, and car appearance (i.e radio link appearance), or disappearance, should not affect, the global functioning of the network.

Moreover, OLSR is designed for multihop ad hoc networks, with a strong and efficient mechanism for data flooding to the entire network. Handling big networks may be inappropriate

for vehicle communications. [7], proposes a set of modifications to OLSR, in order to limit the relaying of topology information and rapidly build small ad hoc networks well suited for the dynamic context of vehicle communications in a crossing. Control information sent by a node (vehicle) is relayed within a restricted area in order to get the routing and topology knowledge of the nearby network. This information is enough to enable the broadcast and the relaying of data to a bigger area than the nearby known topology.

IV. EXPERIMENTALS

A. System architecture

With the Cycab vehicles the perception in an urban environment is handled by a central application that we call Taxi.

Taxi is a multi threaded C++ framework for developing robotic application using different sensors and actuators. It includes a set of C++ classes for building applications to be executed on the vehicle. Our architecture is composed of three components. The PMP block and its visualization module; the OLSR communication block which has the task of detecting new neighbours and broadcasting vehicle information; and the Taxi application block which coordinate the different information flows and has the vehicle low level control (see fig. 6).

PMP block runs the C++ implementation of the algorithms presented in §III and §II. The tracking of the generated trajectories is insured by a non-linear closed loop controller detailed in [13]. Actually, PMP block gives the calculated trajectory to the Taxi application block which has the control law to drive the Cycab.

Vehicles (moving obstacles) are detected by the reception of a set of information forwarded by OLSR block and containing useful data for PMP. This data could be a description of the vehicle like its GPS information, its speed, its dimensions, or its planned itinerary (such as turning left, or right in the next crossing).

Each vehicle must broadcast periodically this set of information to its neighbours in a crossing in order to refresh its characteristics, because a vehicle could change its direction or slow down its speed in any moment.

B. Scenario description

For our experiment we have used 2 Cybercars (One Cycab, a 4 wheeled electric vehicle and one AGV a Yamaha vehicle based on an electric golf car). Both vehicles were crossing at the same moment an intersection, and the PMP (Partial Motion Planner) was running on the Cycab. The Cycab is fully automated. The longitudinal control inputs are provided by the PMP module while the lateral control is insured by a vision based module which detects the road side. The vision algorithm is based on the Poppet (Position of Pivot Point Estimating Trajectory) algorithm [9]; it was controlling the steering. The PMP was controlling the acceleration and the

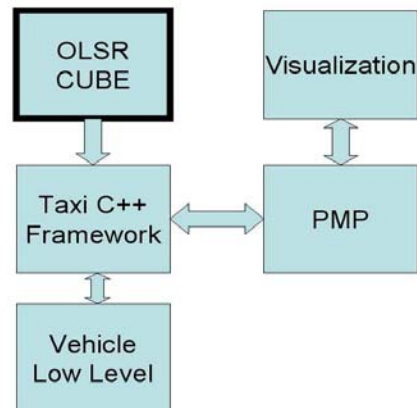


Fig. 6. Architecture of the embedded Taxi application



Fig. 7. 4G System Mesh Cube

laser scanner was used to stop the car if there was an obstacle in front.

The cars are linked via a mesh-network over WIFI using OLSR Ad-Hoc protocol. The entire communication task is embedded in a small MIPS Linux Box. (4G System Cube: see fig. 7). During the experiment, each car is flooding periodically its GPS information to the mesh network. A static wireless mesh cube has been added at the intersection in order to enable communication between vehicles by relaying the forwarded messages, when they were out of reach in terms of radio.

Generally, PMP needs a good perception to evaluate the obstacle around the vehicle, so it can plan a new trajectory and avoid it dynamically. For other experiments we might use a laser scanner, however in this application, only the data sent over the mesh network are used to localize the moving obstacles.



Fig. 8. The cycab and the AGV in a crossing

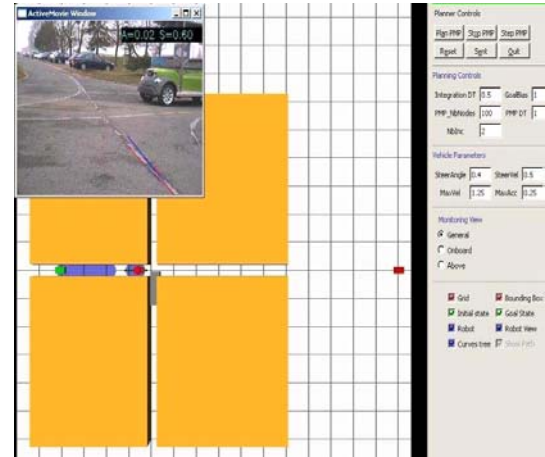


Fig. 9. The PMP interface

C. Results

As the vehicles approach the intersection (see fig. 8), the wireless links between the three nodes (the Cycab, the AGV, and the static mesh cube) is established. Hence, the two vehicles can communicate their GPS information. This information is initially relayed by the static node when the cars are out of reach. This is done by the MPR relaying technique of OLSR. When the two cars are in radio range, they exchange their information without any relaying.

Each received GPS information is processed by Taxi, and sent to PMP as an obstacle information (see fig. 9). The PMP calculates a new trajectory on the basis of each new obstacle information. When the cars are close to each other, the trajectory generated by PMP decelerates the car as the only possible way in this situation to remain safe and let the second car continuing its trajectory.

Regarding the bandwidth we noticed that we can guarantee around 500Kbits when the vehicle is communicating through the infrastructure node (the static mesh cube) and around 3Mbits with a direct connection.

V. CONCLUSION AND FUTURE WORKS

Automated guided vehicles need a strong collaboration between the motion planning unit and the C2C/C2I communication. It adds relevant information about the environment, and it can prevent the perception module from false detection. Intersections are dangerous zones, so several systems can be switched on to take control of the car or just warn the driver to prevent collision.

For our future works, we will use PMP for the full car control allowing to avoid unexpected obstacles coming into the intersection instead of only decelerating the car. To this mean, more accurate sensors like the laser Scanner will be needed to provide sufficient information to PMP.

REFERENCES

- [1] Thierry Fraichard and Hajime Asama. Inevitable collision states - a step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.
- [2] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, March 2002.
- [3] http://europa.eu.int/information_society/activities/esafety/index_en.htm.
- [4] <http://www.car-to-car.org>.
- [5] http://www.prevent-ip.org/en/prevent_subprojects/intersection_safety/intersafe.
- [6] P. Jacquet, P. Muhlethaler, P. Minet, A. Qayyum, A. Laouiti, T. Clausen, L. Viennot, and C. Adjih. Optimized link state routing protocol. In *IETF RFC3626*, October 2003.
- [7] A. Laouiti, L. Bouraoui, A. de la Fortelle. Olsr improvement for distributed traffic applications. In *Fourth Annual Mediterranean Ad Hoc Networking Workshop, MedHocNet'2005*, Porquerolles, France, June 2005.
- [8] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, June 2005.
- [9] S. Dickson, M.B. Wilson. Poppet: A robust road boundary detection and tracking algorithm. In *British machine vision conference 1999*, June 1999.
- [10] M. Parent. Automated public vehicles : A first step towards the automated highway. In *4th World Congress on Intelligent Transport Systems*, October 1997.
- [11] S. Petti and Th. Fraichard. Safe motion planning in dynamic environments. In *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, Edmonton, AB (CA), August 2005.
- [12] A. Qayyum, A. Laouiti, and L. Viennot. Multipoint relaying technique for flooding broadcast messages in mobile wireless networks. In *HICSS: Hawaii Int. Conference on System Sciences*, 2002.
- [13] P. Rives, S. Benhimane, E. Malis and J. R. Azinheira. Vision-based control for car platooning using homography decomposition. In *IEEE International Conference on Robotics and Automation*, pages 2173–2178, Barcelona, Spain, April 2005.
- [14] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 1996.