Trajectory Planning in a Crossroads for a Fleet of Driverless Vehicles

Olivier Mehani[1] and Arnaud de La Fortelle[1,2]

[1] Inria Rocquencourt, Imara Team, Domaine de Voluceau, BP 105, 78153 Le Chesnay Cedex, France, {olivier.mehani,arnaud.delafortelle}@inria.fr
[2] École des Mines de Paris, CAOR Research Centre, 60 boulevard Saint-Michel, 75272 Paris Cedex 06, France

Abstract. In the context of Intelligent Transportation Systems based on driverless vehicles, one important issue is the passing of a crossroads. This paper presents a supervised reservation system. Extending previous works in this direction, the proposed algorithm determines trajectories and speeds for all the vehicles willing to pass the intersection. The work is separated into tasks shared between the vehicles and an infrastructure in charge of the crossroads. This paper describes the characteristics of the algorithm, as well as considerations which led to it. Simulation results are given and support the argument that this algorithm is probably suboptimal but still performs well. Finally, this work shows there is place for ameliorations and hints to further improve the algorithm are given.

Key words: intelligent transportation system, cybercars, driverless vehicles, crossroads, time-space reservations, trajectory planning, speeds scheduling

Aknowledgments: This work has been partially funded by the EC project CyberCars-2 [1].

1 Introduction

In the context of ITS (Intelligent Transportation Systems) based on driverless road vehicles, one interesting and non-trivial problem is that of the passing of crossroads. At an intersection between two roads, the risk of collision is much higher due to the presence of other vehicles not going in the same direction, and potentially crossing each other's path.

It is necessary to find safe algorithms allowing every vehicle to pass this sensitive place without colliding. Moreover, it is important, while doing so, to keep efficiency in mind, for the vehicles not to wait forever before entering and passing the intersection. Every vehicle must pass the crossroads as fast as the safe possibilities allow.

An interesting approach to solve this issue is the use of reservations [2,3]. A supervising infrastructure is in charge of the intersection, and attributes space-time reservations of parts of the crossroads to the upcoming vehicles.

Basing our work on said previous litterature, we try to develop an algorithm usable for driverless cybercars which are parts of a complete ITS, and reasonably implementable in an actual system. In section 2, we formalize the problem and define the desired outputs of the algorithm. The proposed algorithm is then described in section 3. Section 4 describes the ad-hoc simulator developed to test this algorithm, as well as simulation results. The last section sets way for future works in order to improve and implement this algorithm in real hardware.

2 Problem Statement

2.1 Layered Control

In a transportation system with fully autonomous vehicles (or cybercars), a lot of tasks have to be executed to answer the demand of a customer to travel between two points. The framework of this work is an approach consisting in decomposing the planning into three levels, each of which using only a relevant subset of the information, thus reducing the complexity:

the macroscopic level, *e.g.* a city or a region;
the mesoscopic level, *e.g.* a city quarter;
the microscopic level, *i.e.* the surroundings of the cybercar.

At the macroscopic level, a *path* is computed. A path, is defined as a succession of edges (road segments) in the graph description of the road network. This is the level of fleet and roads management with a time scale ranging from half an hour to several hours.

At the mesoscopic level, paths are transmitted by the upper level and turned into *trajectories*. A trajectory is a precise space-time curve that the cybercar has to follow. Typical precisions are 10 cm and 1/10 s. The goal of this level is to produce trajectories for all the cybercars circulating in a given area. These trajectories have to be safe, mainly collision-free, but also efficient in terms of throughput (most notably deadlock-free).

At the microscopic level, the cybercar's control moves along the trajectory and ensures that none of these collisions which couldn't have been foreseen at the higher levels occur.

In this paper, the focus is put on algorithms for the mesoscopic level. More precisely, the study is presented on a simple crossroads (X junction). It has, however, been done with a much more general setting in mind which the developed concepts are supposed to deal with.

One key element of this type of algorithms is the *respect of the controllability constraints of the vehicles* (its dynamics, *e.g.* attainable speeds), for the actual vehicles to be able to follow the generated trajectories. A second key element is the *management of the shared resources*, *i.e.* the intersection.

More formally put, the problem is to attribute a shared resource (the crossroads) to each of the vehicles for a given time period, while distributing this resource the most efficiently possible in order not to block the vehicles or even reduce the speeds too much.

2.2 Desired Output

Whatever the algorithm used to determine the scheduling, the expected output must be usable by a vehicle control algorithm, that is by a microscopic level. The data provided by the considered layer should, thus, be composed of position and speed (or time) information.

It is also necessary to have this information *in advance* so that the microscopic level can plan the accelerations or decelerations that may be needed.

3 Reservation Algorithm

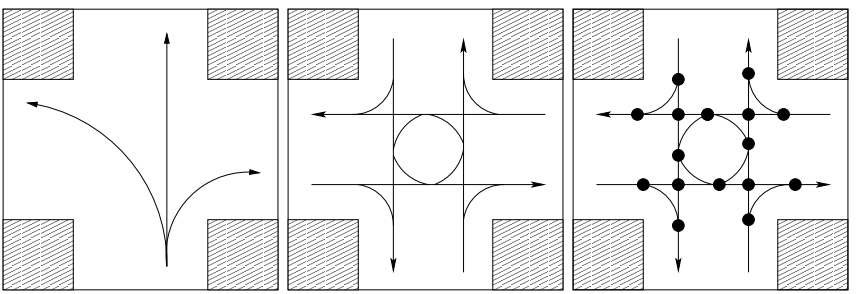

Fig. 1. A regular crossroads. Left: each vehicle entering it has three possibilities to get out. Center: trajectories that vehicles are able to, and *may*, follow. Right: only a small subset of critical points on the trajectories is where the risk of collision exists.

3.1 Simple Crossroads Model

A crossroads is basically the intersection of two roads. Considering the most regular one, a single vehicle already has the option of choosing between three directions (Fig. 1, left). Doing so, the vehicle will have to occupy various contiguous time-space places of the crossroads. When more than one vehicle are willing to pass the crossroads at the same time, it is increasingly hard to determine a safe schedule ensuring that there will be only one vehicle at a given time-point, taking into account its physical dimensions.

3.2 Reservation of Parts of the Intersection

In [2], Dresner and Stone proposed to use square *patches* as the parts of the road space to reserve. They have, moreover, shown that the smaller the squares were, the more vehicles could pass the crossroads in a given time period. This

is due to the fact that, with smaller patches, it is possible to reserve and free the needed space on the intersection more precisely, thus letting unneeded space available to other vehicles.

It is the authors' feeling, however, that this might not scale well to large crossroads or number of vehicles. Trying to get a better precision by reducing the reservable squares' size, thus increasing their number, would increase the charge on the supervisor, even if no collision could happen on some of the squares in normal operations (*e.g.* a part of the road where vehicle can only go in one direction). Reducing the set of reservable zones to those which actually need be reserved seems to be an appropriate way to increase the scalability of the system.

Given the dynamic constraints of the vehicles, it is possible to determine attainable 2-dimensional traces on the road (Fig. 1 on the previous page, center) that they are able to drive along to reach their target outgoing lane. One can see that only parts of the 2D traces are collision-prone. These are the places where a box (representing the vehicle) intersects another box placed anywhere on another trace. We further reduce the complexity by considering only the points where two or more traces intersect. This set is called the *critical points* (Fig. 1, left) and is proposed as the items to place reservations on in the intended algorithm. This simplifying assumption is justified by the simulations (using safety margins), where no collision occured. Of course a mathematical analysis should yields these safety margins from the geometry.

3.3 Outline of the Algorithm

Two types of entities are involved in the reservation process: (1) the vehicles, which place the reservations and adapt their speed in consequence, (2) the supervisor, which accepts or refuses a reservation request depending on the current schedule for the critical points. Each entity is supposed to be running a software agent able to execute the algorithm, which is, thus, separated into two distinct complementary parts.

Both intervenants retain useful information . The supervisor holds the geometry of the crossroads (*i.e.* the 2D traces) and maintains a list of all the current reservations on every critical point. The vehicle has information about its dimensions, and attainable speeds and accelerations. The agents need to share this information properly in order to successfully place a reservation.

The proposed method can be outlined as a three-steps algorithm (Fig. 2 on the facing page):

1. A vehicle arrives close to the crossroads and requests the crossroads geometry from the supervisor (*i.e.* the 2D traces and critical points);
2. According to its speed, it builds a reservation request which is sent back to the supervisor;
3. The supervisor decides whether the request is acceptable or not and informs the vehicle agent.

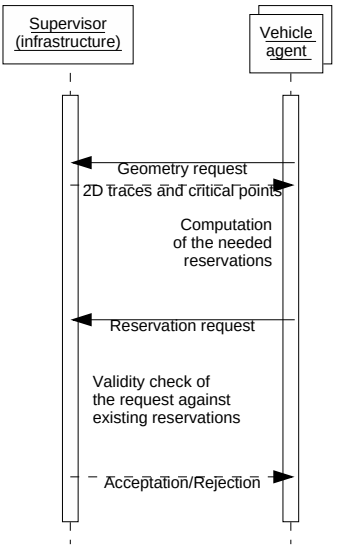

Fig. 2. A synopsis of the communications between a reservation agent and the crossroads supervisor. The proposed reservation scheme heavily relies on information exchanges between the supervisor and vehicles.

3.4 Building a Reservation Request

The following assumes a constant speed during one reservation *i.e.* , the vehicle is not accelerating, thus not integrating speed changes into the expected times computations. This assumption is made to simplify the problem solving, but removing it, in subsequent studies, may give more efficient results.

The vehicle arriving to an intersection has to compute the speed at which it will enter and pass it. To do so, it needs information about the crossroads' geometry. Moreover, it's the vehicle's role to try and place reservations on the critical points. The first step of the algorithm then consists of asking the infrastructure in charge of the crossroads to send these parameters.

Once all needed information is available, the *expected time of arrival* (ETA) to every critical point along the trajectory is computed. Knowing its current speed, the vehicle agent can also compute how long it will take to pass a critical point (*i.e.* the time period from when the front of the car starts being at the point to when its back enventually leaves it), the *expected time to pass* (ETP).

With both these values, a vehicle can determine the starting (1) and ending (2) times for each critical point it has to reserve.

$$t^{\text{start}} = ETA - \frac{sfETP}{2}, \tag{1}$$

$$t^{\text{end}} = ETA + \frac{sfETP}{2}. \tag{2}$$

The additional security factor $sf > 1$ is higly recommended in order to reserve a critical point a short time in advance, and free it a bit later than needed.

The reservation request is then built from these computations. It contains, for each critical point on the trace, the t^{start} and t^{end} parameters.

An improvement of this scheme, as derived from previous consideration about the construction of critical points, may be to reserve not only critical points on the trajectory, but also the *neighboring* ones. A vehicle is, indeed, not a punctual object and needs space around it to be able to do its maneuvers safely. A reasonable heuristic to determine whether a critical point can be considered as neighboring or not is to check its distance to the current on-trace critical point. If this distance is lower than the vehicle's half length, it can be considered as neighboring. In this situation, a reservation with the same timing values as the neighboring on-trace critical point should be requested at the same time.

3.5 Checking the Validity of a Request

The supervisor is in charge of accepting or rejecting a reservation. To do so, it keeps track of all the critical points of the crossroads and the times at which each is already reserved.

Upon an upcoming reservation request, it will sequentially check, for all of the critical points to be reserved, that the requested periods are free. In case *any* of them is not, the *whole* reservation is rejected. If no overlap is found, the reservation is accepted.

3.6 Behavior After a Request Has Been Answered

There are two outcomes to the reservation request: either it is accepted or refused. Depending on the supervisor's answer, the vehicle will behave differently:

the reservation is refused $\Rightarrow$ the vehicle slows down to stop *before* the first critical point while continuing to try and obtain a reservation;

the reservation is accepted $\Rightarrow$ the vehicle remains at a constant speed *or* tries to place new reservations assuming higher speeds in order to pass faster.

3.7 Comments About the Algorithm

It is important to note that, in the above algorithm, the work is clearly separated into three phases:

1. the vehicle agent's building of the reservation;
2. the supervisor's validation of the request;
3. communication between both entities.

This is very interesting as, as mentionned in [3], this means that the implementation of parts of this algorithm can be done in a fully separated way, as long as they respect the communication protocol. Taking the idea a bit further, this means that it is possible to implement, on either side, a completely different approach than those described above and, provided it is valid, still have a fully working system.

4 Simulation Results

In order to validate the algorithm and evaluate its efficiency, a simulator has been developed[3]. Written in Python, it simulates a crossroads and both vehicles and infrastructure cohabitating on it.

It has been developed as several modules, with the objective of making them easily interchangeable. The idea was to provide an algorithm-testing framework into which it would be easy to implement new crossroads scheduling policies to compare their efficiency. In order to test the performances of the proposed reservations algorithm, two other crossroads-passing policies have been implemented:

None which allows every vehicle to pass the intersection at full speed regardless of the collisions. It is supposed to be a good estimation of the lower bounds of the time needed to pass the crossroads. An algorithm getting time results close to those of this dummy policy could reasonably be considered a good one.

Polling treats the crossroads as a single atomic resource and only allows one vehicle at a time to pass it. This policy is 100% safe in terms of collision- and deadlock-freedom, but is intuitively not time-efficient.

Simulations of these two policies and the reservations based have been run for 100s[4] each. One can note (Table 1) that the dummy policy has a high throughput and relatively low passing times, while the polling policy only has one quarter of the throughput and much higher times, but no collision.

	Time (s)			Collisions	Vehicles
	min	max	avg		
None	5.28	10.80	6.21	458	422
Polling	5.28	92.02	47.37	0	108
Reservations	5.28	16.82	9.79	0	134

Table 1. The performances of the reservation algorithm compared to others. All results were obtained running the simulator for 100 (simulated) seconds with a 0.02s timestep.

The reservation algorithm, in comparison, behaves quite efficiently: no collisions have occured while the throughput was higher than that of the polling algorithm, and the times to pass were closer to those of the dummy policy.

This behavior is what was expected. The reservation algorithm detailed earlier was developed to provide a more efficient use of the crossroads space by distributing it between the vehicles in a much denser way than the polling algorithm does. Moreover, the fact that vehicles are *required* to have reserved a

[3] `https://gforge.inria.fr/frs/?group_id=424&release_id=951`

[4] in-simulation time, 0.02s timesteps

critical point before passing it is the key to avoid collisions on the managed space as long as the supervisor only accepts non-overlapping reservations.

In this respect, one can consider that this reservation algorithm, as simulated here, is quite efficient and may be interesting to study and implement further.

5 Conclusion and Future Works

After simulating the proposed algorithm, it turns out that it is, indeed, an interesting scheduling method to improve the crossroads management in the context of fully automated cybercars. It has been shown that better results were obtained than with polling methods similar to traffic lights, as the intersection can be used by several vehicles at the same time thanks to the separation of the main resource into critical points.

The preliminary simulations which have been run, however, were assuming a lot of simplifying conditions which should be, in future works on this topic, removed in order to obtain a much more realistic simulation and implement the algorithm in actual cybercars.

A good improvement to consider is to remove the "constant speed" assumption and compute reservations for an accelerating (or decelerating) trajectory instead of constantly trying to place reservation at increasing speeds once the first one has been granted. Other ideas would be to have the vehicles generate more than one reservation request at a time (vehicle agent modification only), or let the supervisor propose (for resubmission) an acceptable reservation in case it rejected that of a vehicle

One important thing to do would also be to formally prove the deadlock-freedom of the algorithm, in the case of a perfect microscopic level, and provide ways to avoid those which may arise in a more realistic case (*e.g.* if a cat crosses in front of a car, preventing it from being able to continue *on time*, on its trajectory).

References

1. European project CyberCars-2, http://www-c.inria.fr/cybercars2/.
2. Kurt Dresner and Peter Stone. Multiagent traffic management: A reservation-based intersection control mechanism. In *The Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 530–537, New York, New York, USA, July 2004.
3. Kurt Dresner and Peter Stone. Multiagent traffic management: An improved intersection control mechanism. In *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 471–477, Utrecht, The Netherlands, July 2005.