Thesis for the degree of Master of Science in
Complex Adaptive Systems

Mesoscopic Management of a Fleet of Cybercars at a Crossroads

Olivier Mehani

Department of Applied Physics
Chalmers University of Technology
Göteborg, Sweden, 2007

Mesoscopic Management of a Fleet of Cybercars at a Crossroads

OLIVIER MEHANI

Department of Applied Physics
Chalmers University of Technology
412 96 Göteborg, Sweden
Telephone: +46 (0)31 772 1000

Chalmers reproservice
Göteborg, Sweden, 2007

Abstract

In the context of driverless automatic road vehicles, one interesting and non-trivial problem is that of the passing of a 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 point 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 crossroad as fast as the safe possibilities allow.

This 6 months internship, done with the Imara team, tried to come up with an efficient algorithm for crossroads passing. A simulator has been written, giving the possibility to test various algorithms and efficiency comparisons have been made in order to determine which algorithm was the most promising, and where were the behaviors that could still be improved.

Finally, an algorithm based on a *reservation* of *critical points* has seemed to be the most interesting. Details about its inner working and implementation are given, as well as preliminary results.

Acknowledgment

I would like to thank Inria Rocquencourt for allowing me to work with the Imara team which enabled me to discover and solve some exciting problems and work in the fields I'm most interested in.

I am particularly grateful to Arnaud de la Fortelle, my supervisor in the team, who was always ready to answer my questions and gave me valuable pieces of advice concerning the way to steer my work, while still leaving me in control.

I also would like to thank Laurent Bouraoui, Armand Yvet and Rodrigo Benenson who, thanks to their availability and willingness to answer my questions, allowed me to quickly and fully integrate in the team.

Finally, I do not forget everybody else in the team whom I did not mention. It's been very interesting to work, for my internship, in this team, and I'm very happy to be able to continue working with them as an engineer.

Contents

1 Introduction **1**

2 Presentation of Imara **3**
2.1 Inria . . . 3
2.2 Short timeline . . . 4
2.2.1 Praxitèle . . . 4
2.2.2 Imara . . . 4
2.2.3 LaRA . . . 5
2.3 Research fields . . . 5
2.4 Platforms . . . 6
2.4.1 C3s . . . 6
2.4.2 CyCabs . . . 6

3 Crossroads passing algorithms **9**
3.1 Problem statement . . . 9
3.1.1 Layered control . . . 9
3.1.2 Crossroads . . . 10
3.1.3 Desired output . . . 10
3.2 Preexisting works . . . 11
3.2.1 Traffic lights control . . . 11
3.2.2 Multiagent-based reservations system . . . 11
3.3 Yet another reservation system . . . 12
3.3.1 2-dimensional traces on the crossroads . . . 12
3.3.2 Determining the resources to share . . . 12
3.3.3 Details of the algorithm . . . 13
3.3.4 Some comments about the algorithm . . . 17

4 Simulation and results **18**
4.1 Simulator . . . 18
4.1.1 Simplifying assumptions . . . 18
4.1.2 Technical choices . . . 18
4.1.3 Architecture . . . 18
4.2 Results . . . 23

5 Conclusion and future works **27**

A Model of a car and its trajectory **I**
A.1 Dynamic model of a cybercar I
A.2 Acceptable traces model I
A.2.1 Dubin's curves I
A.2.2 Clothoids I
A.2.3 Generating traces II

Chapter 1

Introduction

The work in the Imara team has been done in the context of driverless cybercars called CyCabs (Fig. 1.1). These fully automated vehicles are equipped with several types of sensors (cameras, lasers, ...) and communication devices (mostly WiFi links) which enables them to sense their environment and exchange information both with each other and with a potential intelligent road infrastructure.

Figure 1.1: Different types of cybercars: two CyCabs and one AGV. Reproduced with permission from Imara, courtesy of Rodrigo Benenson.

In this context, the situation of an intersection between two roads causes an increase in the difficulty to avoid collisions. In quite a dense traffic uniquely composed of CyCabs, the vehicles will need to pass on a part of the road which is shared between vehicles not going in the same direction (the crossroads).

It is necessary, for each vehicle, to be able to compute a trajectory (that is the speed to

achieve along a determined path) in order never to be at the same place as another vehicle. To do so, several algorithms have been thought of and compared to finally try and achieve the best efficiency while ensuring that no collision was possible.

In the following, the problem will be developed in much more details, then a survey of the pre-existing works in the field will be summed up, before going into depths about a reservation algorithm which proved to give the best results among those studied. Some simulation results will also be added to support the argument.

Chapter 2

Presentation of Imara

2.1 Inria

Inria, the French National Institute for Research in Computer Science and Control, has been created in Rocquencourt, near Paris, in 1967. It is a public scientific and technical research center managed by the *Ministère de l'Éducation Nationale* and the *Ministère de l'Industrie*.

Among the activities of the research unit are experimental systems, fundamental and applied research, technologies transferts, organization of international scientific exchanges and spreading of knowledge and know-how.

Computer scientists, mathematicians, and automaticians work on research projects about five main themes subdivided into several axes:

Comunicant systems

- Distributed systems;
- Networks and telecommunications;
- Embedded systems;
- Architecture and compilation.

Cognitive systems

- Statistical modelization and learning;
- Images and videos: perception, indexing and communication;
- Multimedia data: interpretation and human-machine interaction;
- Image synthesis and virtual reality.

Symbolic systems

- Software safety and reliability;
- Algebraic and geometric structures, algorithms;
- Contents and language organization.

Numerical systems

- Complex systems;

- Clusters and high-performance computing;
- Optimization and stochastic or highly dimensional problems;
- Modelisation, simulation and numerical analysis.

Biological systems

- Modelisation and simulation for biology and healthcare.

Since 1992, Inria is involved in important projects with industrial goals. These programs – usually planned to take 3 or 5 years - are led in partnerships with industrial actors and users of the information technologies. With several research teams participating in each of these actions, they create significant coordination and collaboration opportunities between research projects and technology companies.

Inria Rocquencourt is one out of six research units. 39 research teams, for a total of roughly 800 people, are hosted on the site. One of these projects is Imara (Informatique, Mathématiques et Automatique pour la Route Automatisée or, roughly, Computer Science, Mathematics and Automation for the Automated Road).

2.2 Short timeline

2.2.1 Praxitèle

The Praxitèle programme was started in 1993 as a partnership between Inria, Inrets and several companies like Renault, EDF, Dassault Electronique and CGFTE – a mass transit operator which was also the project leader. The aim was to develop and evaluate a new individual public transport based on self-service small electric vehicles.

In this project, Inria has been responsible for the modeling and the real time management of the system. Inria has also developed a new type of electric automated vehicle specific for this application: the CyCab. The vehicle can be operated safely, be it in semi or fully automatic mode.

The scientific director for the Inria team was Michel Parent.

2.2.2 Imara

At the end of the Praxitèle programme, in 1997, the Imara project was started, at Inria Rocquencourt, by Michel Parent, with the objective to continue the work which has been started with Praxitèle and provide safer, more efficient and more comfortable intelligent transportation systems, mostly via automatization.

Increased safety is based on four approaches: driver monitoring and warning, partial control of the vehicle in case of emergency, total control of some functions, or even of all the vehicle. This is achieved by techniques providing services ranging from driver aid to full driving automation.

Efficiency and comfort are increased by a better usage of the available resources (roads, vehicles and energy). Works aiming at reducing the congestion of existing infrastructures or proposing a multi-modal service where the classical automobile is only used wherever a mass-transportation service does not exist allow to improve these factors.

Energy savings, also, is not directly achieved as part of the research work, but mostly by promoting new and cleaner means of transportation.

Since 1997, the fields of interests of Imara have widened and the team now also focuses on non-fully automated vehicles, with the same global objective in mind.

2.2.3 LaRA

Since December 2005, Imara and the CAOR robotic lab of the École des Mines are associated in a joint research unit called *LaRA* – La Route Automatisée or "the Automated Road". Sharing the same goals in terms of integration and experimentation, these two labs – roughly 50 people – tightly collaborate.

The general philosophy is to assist the driver to improve road safety, comfort and efficiency of the road networks. The long term objective is to totally remove the driver from the control loop, at least in some specific maneuvers or infrastructures.

2.3 Research fields

Imara is a "horizontal" project at Inria. The project coordinates and transferts all the research done at Inria which can be applied to the the concepts of the Automated Road. In particular, the results of a number of Inria projects in the following domains are integrated:

Embedded systems

- command and control;
- safety and reliability.

Signal processing

- infrared or ultrasonic detection of obstacles;
- Laser reconstruction of the surrounding features;
- computer vision;
- etc.

Artificial intelligence

- decisions (potentially after negociation with peers);
- path and trajectory finding;
- navigation components.

Communications

- wireless links;
- mesh networks;
- dynamic topologies;
- IPv6 and mobility.

Figure 2.1: A CyCab (left) and a Citroën C3 (right), the two main platforms used at Imara. Reproduced with permission from Imara, courtesy of Laurent Bouraoui.

2.4 Platforms

One of the key strengths of Imara is that the team possesses several vehicles equipped with computer hardware. It is then possible to easily implement experiments and test both hardware and software in real conditions instead of just relying on simulation results. Imara currently uses two main development platforms (Fig. 2.1): Citroën C3 and CyCabs.

2.4.1 C3s

The C3 platform (Fig. 2.2 on the following page) is mostly used for applications aimed at transmitting information to the driver, the infrastructure or other vehicles.

The vehicles are equipped with several computers. Information about the vehicle status is obtained via the car's CAN bus. The computers are used to acquire and process data from the embedded sensing devices (cameras, Lasers, GPS receivers, ...), establish short- or long-distance network links using current technologies (wireless, GPRS,...) or run specific resource-intensive applications like Adas-RP (Advanced Driver Assistance System) or other navigation systems.

2.4.2 CyCabs

An original development of Inria's research teams, the CyCab is an example of cybercar. This electric vehicle is fully automated and equipped with two computers – one taking care of the low level control of the wheels and overall dynamics of the vehicle, the other in charge of the execution of higher level navigation algorithms.

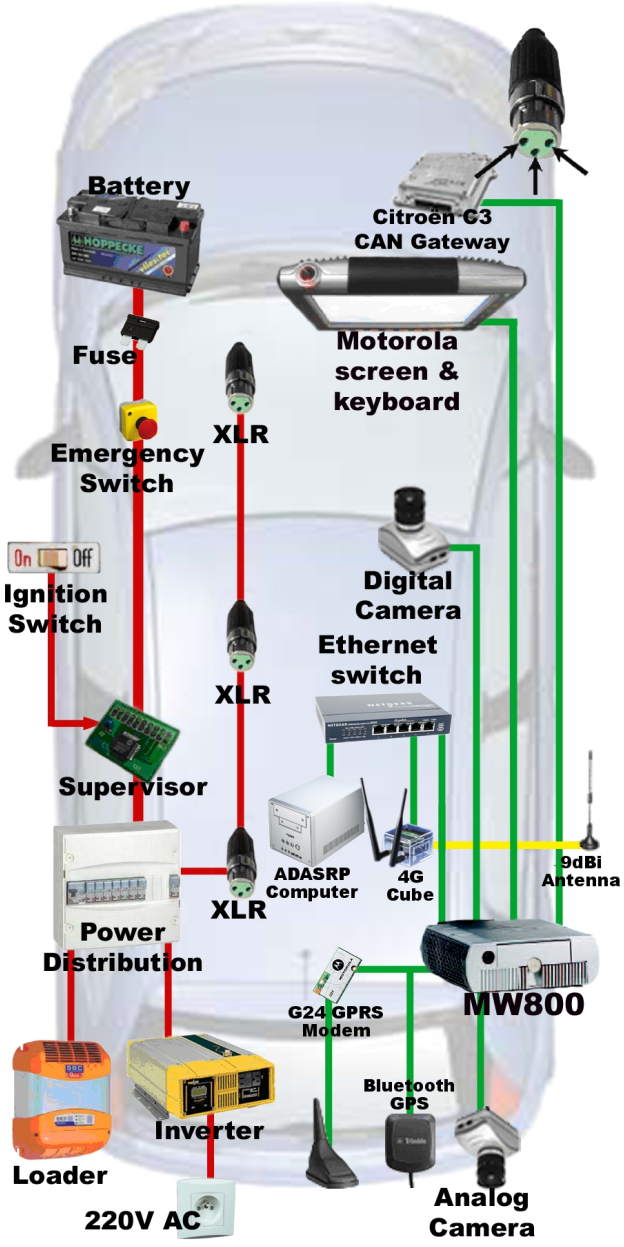

Figure 2.2: The C3s' hardware architecture. Reproduced with permission from Imara.

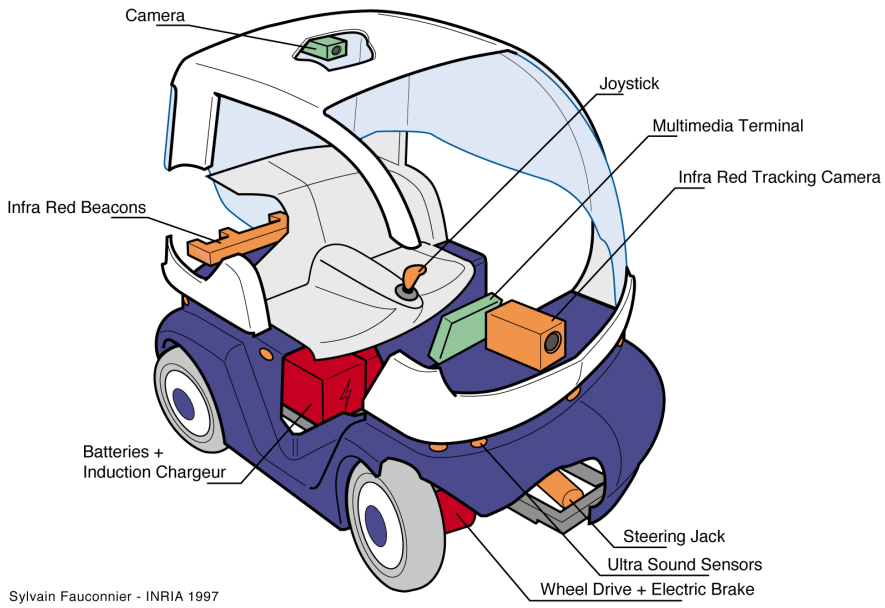

Figure 2.3: The CyCab's initial hardware architecture. Reproduced with permission from Imara.

Various types of sensors (ultrasonic devices, stereo-cameras, infrared devices,...) in order to sense their environment and provide the navigation algorithms with enough information to safely decide which movements to achieve.

This internship's motivations are mainly rooted with these vehicles in mind. However, it is not impossible to later use the produced algorithms with any type of vehicle.

Chapter 3

Crossroads passing algorithms

3.1 Problem statement

3.1.1 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, most notably collision-free, but also efficient, *i.e.* 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 thesis, the focus was put on 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 these 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.

3.1.2 Crossroads

A crossroads is basically the intersection of two routes. Considering the most regular one, a single vehicle already has the option of choosing between three directions (Fig. 3.1). Doing so, the vehicle will have to occupy various contiguous places of the time-space 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.

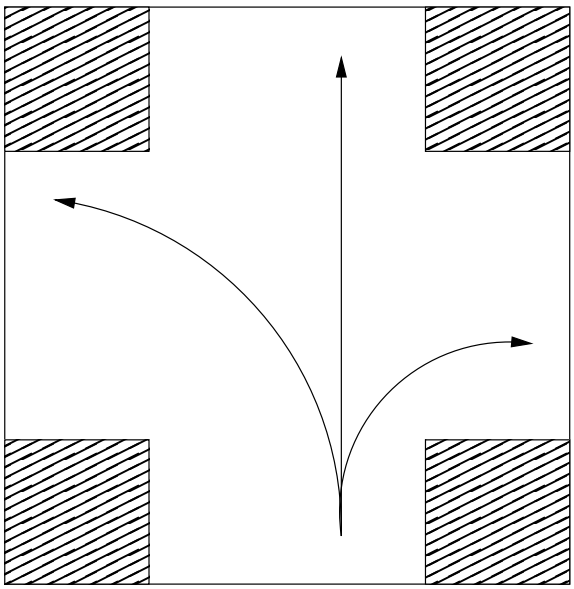

Figure 3.1: A regular crossroads. Each vehicle entering it has three possibilities to get out.

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 reduce the speeds too much or even block the vehicles.

3.1.3 Desired output

Whatever the algorithm used to determine the scheduling, the expected output must be usable by the cybercar, that is by its microscopic level. The data provided by this 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 acceleration or deceleration that may be needed.

The performance of the algorithms can be measured in terms of the average, minimum and maximum time it took a vehicle to passe on the crossroads. Of course, these have to be minimized. Another interesting criterion is the throughput of the crossroads, which would be interesting to maximize as much as possible.

3.2 Preexisting works

3.2.1 Traffic lights control

Scheduling with conflicts

In [1], Sandy Irani and Vitus Leung consider the traffic intersection control as the scheduling of competing jobs with limited resources. The conflicts are modelled by graphs with the nodes being the jobs to be accomplished (*i.e.* a vehicle or platoon of vehicles willing to pass the crossroads) and the edges, the conflicts existing between the jobs. From this formalized representation, the system infers the set of compatible jobs that can be run at the same time.

Using this scheduling algorithm, vehicles are allowed to pass the crossroads (*i.e.* the traffic lights are green) if they do not conflict with other vehicles or platoon already allowed to pass the crossroads. This scheme imposes physical constraints on the intersection as their must be specific lanes for turning vehicles (*e.g.* going from North to East) for the conflict graph not to have too few nodes and too much edges.

This approach is interesting because it garantees the absence of deadlock, thanks to the properties of the conflict graph. It does not, however, take the dynamics of the vehicles into account. Moreover, it is not trivial, from the traffic lights' state information, to determine in advance the speeds the vehicles have to achieve.

Self-organizing traffic lights

Carlos Gershenson proposed, in 2006, the concept of self-organizing traffic lights [2]. In the paper, he compared several traffic lights control method, both fixed and self-organized.

Fixed methods just change the traffic lights state at given intervals, as it is currently done in the Real World.

Self-organizing methods take into account the current upcoming traffic to adjust the lights' cycles (for example, not toggling the lights if there is no traffic from the other direction).

The simulations run by Gershenson in [2] showed that the self-organizing methods were performing better in terms of average speed of the vehicles, number of stopped vehicles and average time to pass an intersection. Some asumptions made in the simulations, however, limit the range and usability of the results:

- the vehicles were only coming from the northern and eastern part of the "world", thus not simulating vehicles on the same axis in opposite directions passing each other or trying and turn to the same lane ;
- once again, the dynamics of the vehicles was not taken into account and it was not possible to predict the desired speed in advance.

3.2.2 Multiagent-based reservations system

Kurt Dresner and Peter Stone suggested a multiagent traffic management system [3, 4] in which a software agent running in the vehicles communicates with the infrastructure in order to get a reservation on parts of the crossroads for a given period when it is expecting to pass.

In the first paper, the basics of the system are set with important limitations – for example, the vehicles aren't allowed to turn – but some interesting characteristics. Instead of considering the intersection between two roads as a single resource, it is possible to split the shared part of the road into *reservations tiles*. The vehicles then have to place several reservations, with various times and geographic positions. Whenever one tile is refused, the whole reservation request is dropped and the vehicle has to wait until it can get a valid reservation and enter the crossroads. The second paper removes the interdiction to turn, and gives a much more implementable version of the system.

This reservation approach brings a new important feature. Contrary to traffic lights methods seen above, it is here possible to deduce, from the obtained reservation, the speeds at which the vehicle should drive. Moreover, as the whole reservation is known *before* entering the crossroads, the speeds profiles are known in advance, eventually giving the microscopic level more information to decide how to adapt its moves.

Another interesting fact is the splitting of the resource in space and time. It becomes possible for two or more vehicles, with totally differing origins and/or destinations to access the crossroads at the same time while still being treated individually (as opposed to platoons, which are not always trivial, when possible, to form).

One may notice, however, a slight drawback depending on the desired granularity of the reservation tiles. With a very fine segmentation of the crossroads, there might be a large number of these patches to keep track of, which may be too computationally intensive.

3.3 Yet another reservation system

Following this quick review of preexisting works, it is obvious that traffic lights-based systems do not seem to easily provide the expected properties to reach the given goal. On the contrary, the reservation system of Dresner and Stone already has some desirable features. Based on these observations, the choice has been made to build a similar reservation system which could properly integrate in our levels-based framework.

3.3.1 2-dimensional traces on the crossroads

In the context of the classical 2×2 crossroads as shown on Figure 3.1 on page 10, and following the results of a previous internship concerning vehicles attainable moves (see Appendix A on page I for a summary), it is possible to generate all the 2D traces on the intersection that a vehicle may have to follow depending on its itinerary (Fig. 3.2 on the following page).

The curves are generated using clothoids, which are integrable functions guaranteeing the physical attainability of each trace to the vehicles. As the geometry of a given intersection does not change very often, it is reasonable to compute these traces once and store them in the infrastructure agent. This agent may, in turn, give away this information to its counterpart running in the vehicles, when requested.

3.3.2 Determining the resources to share

As already stated, the reservation algorithm of Dresner and Stone may not be very scalable due to its way to split the crossroads. It is necessary to determine which resources *actually* need management, and which ones can be ommitted without degradation of the performances of the algorithm.

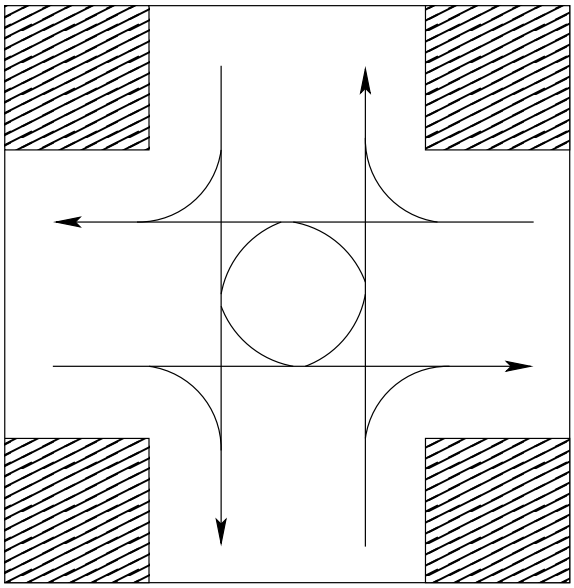

Figure 3.2: Example of 2-dimensional traces that a vehicle may follow on the intersection.

Information	Supervisor	Vehicle agent
2D traces	•	
Critical points	•	
Source lane		•
Destination lane		•
Current speed		•
Possible speed range		•
Possible acceleration range		•

Table 3.1: The information required for the algorithm to work and which actors originally knows each of them.

Given the traces on the road (Fig. 3.2), it is obvious that the risk of collision exists especially where two or more traces intersect (Fig. 3.3 on the following page). It is indeed possible for two vehicles with different sources and destination to be at the same trace intersection at the same time (which is to be avoided). These intersections will be refered to as *critical points*.

Restricting the reservations requests to the critical points allows to greatly reduce the number of reservable entities to manage, while keeping the desired functionnality of the algorithm unchanged.

3.3.3 Details of the algorithm

Once the resources to share have been determined and the information has been distributed between the actors (Table 3.1), it is fairly simple to determine the steps of the reservation algorithm.

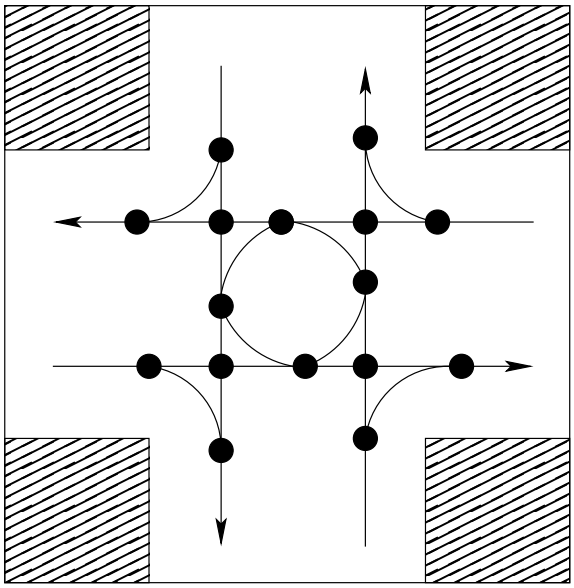

Figure 3.3: Some specific points, where two or more 2D traces intersect, are where the risk of collision exists. These points are called *critical points*.

CP_1	t_1^{start} t_1^{end}	CP_2	t_2^{start} t_2^{end}	$\cdots$	CP_n	t_n^{start} t_n^{end}

Table 3.2: An example reservation request.

Communication between the vehicle agent and the supervisor

In order to build a reservation request, the vehicle first needs to know which critical points exist on the crossroads, and the 2D traces it will have to follow. Then it has all the necessary information to compute a request. The algorithm (which UML sequence diagram can be seen on Figure 3.4 on the following page) can then be outlined as follows:

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.

Building the request

The request is composed of the list of critical points to reserve – that is, the critical points on the trace the vehicle has to follow to reach the exit of the interscetion. For every critical point, the starting and ending times of the reservation are embedded in the request. An example request diagram can be seen on Table 3.2.

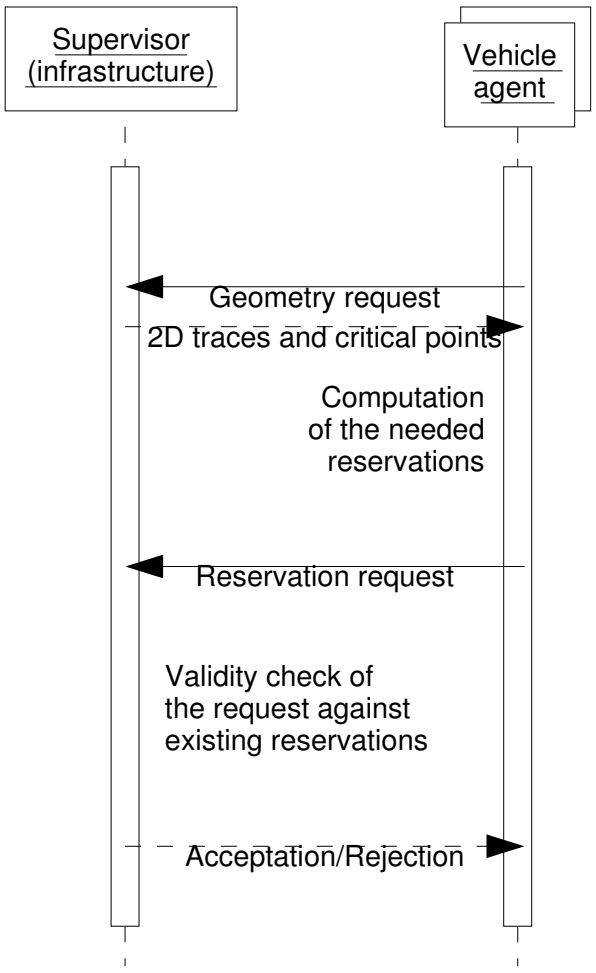

Figure 3.4: A synopsis of the communications between a reservation agent and the crossroads supervisor.

According to its current speed, the vehicle can compute its *expected time of arrival* (ETA) to each of the critical points. As the dimensions of the vehicle are also known by the agent, it is able to compute the *expected time to pass* (ETP) *i.e.* the time from the first instant the vehicle is "over" the critical point to the last instant it is (Fig. 3.5).

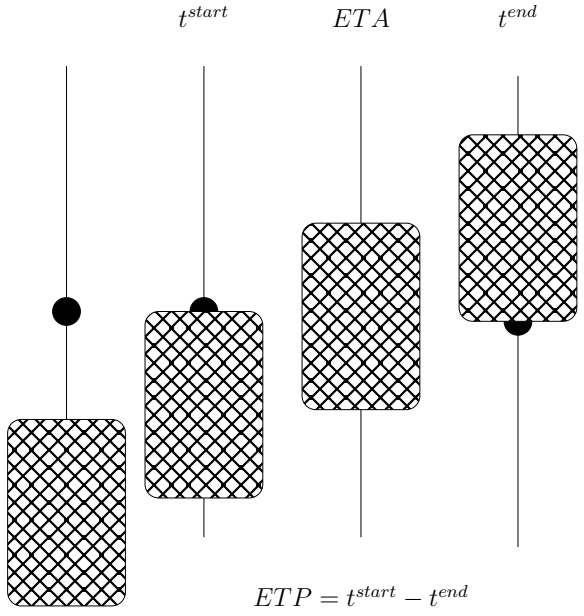

Figure 3.5: The relation between the time values used when building a reservation request.

To have a larger time window and ensure more security, the agent can also have a *security factor* sf (≥ 1) by which multiply the ETP. According to this scheme, the times at which a reservation starts and end can be expressed as

$$t^{start} = ETA - \frac{sfETP}{2}, \tag{3.1}$$

$$t^{end} = ETA + \frac{sfETP}{2}. \tag{3.2}$$

Assuming the reservation is made at a constant speed (*i.e.* the vehicle is not planning to change its speed while following the trajectory) s, d being the distance to the critical point *along the trajectory* and l the vehicle length, ETA and ETP are computed as

$$ETA = \frac{d}{s}, \tag{3.3}$$

$$ETP = \frac{l}{s}. \tag{3.4}$$

Another interesting – and seemingly more efficient in terms of performances – way to compute ETA and ETP would have been to take the vehicle's acceleration into account and integrate the speed so that the reservation can already reflect the expected acceleration along

the trajectory. This has not been the case in this study but may be interesting to keep in mind as future works.

It seems reasonable, also, to try and reserve *neighboring critical points*, *i.e.* those critical points not on the trace the vehicle has to follow but which may be too close to it.

Checking the validity of a reservation

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 periods 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 *anyone* of the period is not free, the *whole* reservation is rejected. If no overlapping is detected, the reservation is accepted.

Behavior after a reservation 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 agent 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.

3.3.4 Some 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 [4], 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.

Chapter 4

Simulation and results

4.1 Simulator

In order to test the algorithm presented in section 3.3 on page 12, a simulator (Fig. 4.1 on the following page) has been written. The source code is available[1] in Inria's GForge.

4.1.1 Simplifying assumptions

As the goal is to validate the proposed algorithm, several simplifying assumptions have been made:

perfect communication the transmission time between the agents is considered null and no packet is lost;

perfect microscopic level once the trajectory has been passed on by the mesoscopic level, the microscopic level *exactly* follows the plan given by the upper level;

homogeneous traffic only cybercars running a vehicle agent to place reservations are supposed to be on the road.

4.1.2 Technical choices

The simulator is supposed to be a proof-of-concept prototype. As noted earlier, several parts of the proposed algorithm, or even the full algorithm, may be replaced in order to test new methods or compare performances.

With this in mind, it has been decided to write the simulator in Python[2], a highly portable programming language with a fully object-oriented base.

4.1.3 Architecture

The idea was to provide an algorithm-testing framework into which it would be easy to implement new policies. Using and object-oriented approach allowed to design interfaces (Fig. 4.2 on page 20) with which the actual modules implementing the policies should conform in order to be directly simulable.

[1] browseable source: https://gforge.inria.fr/plugins/scmsvn/viewcvs.php/simulator/?root=mehani; downloadable package: https://gforge.inria.fr/frs/?group_id=424&release_id=951

[2] http://www.python.org/

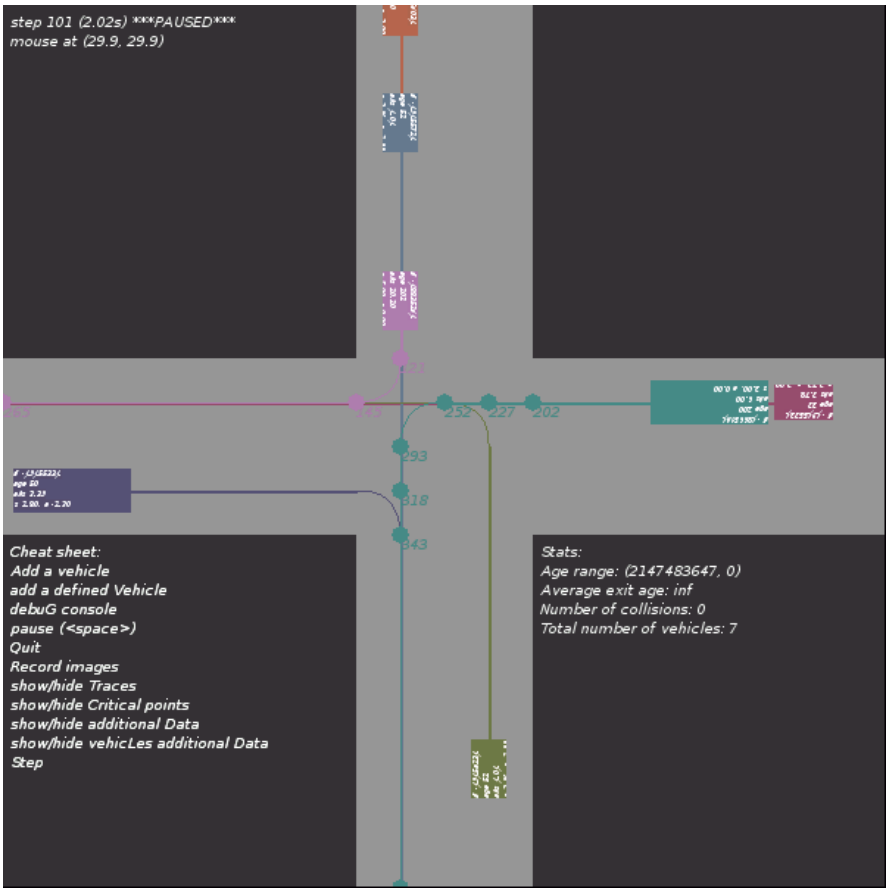

Figure 4.1: A screenshot of the crossroads simulator.

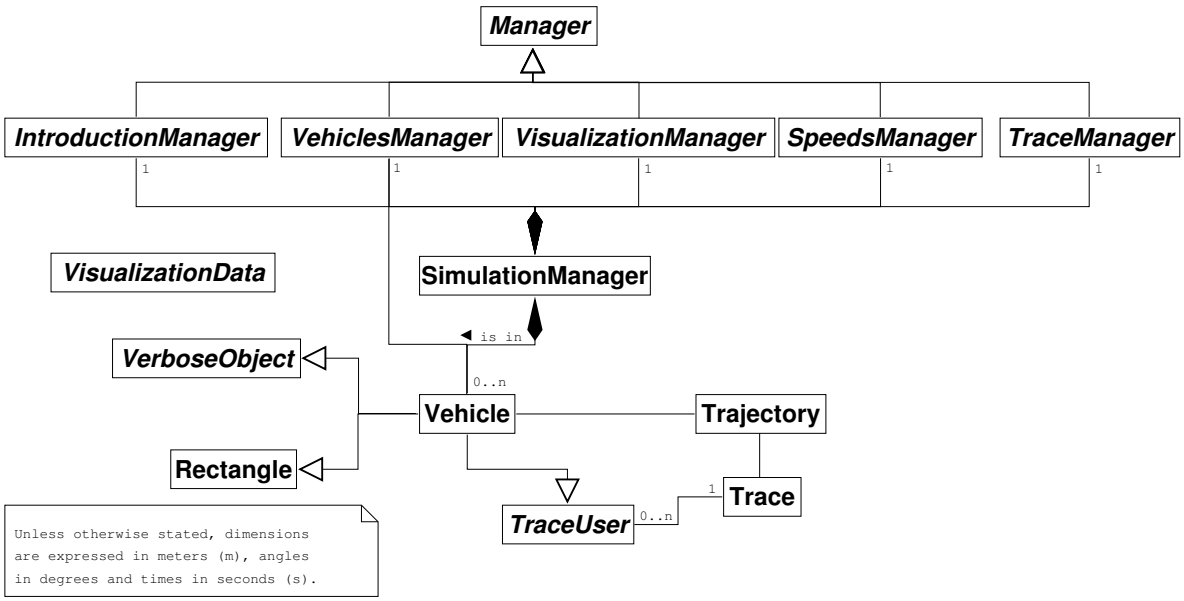

Figure 4.2: The main UML class diagram of the simulator.

Some of the classes shown in the main UML diagram (the `Managers`) are only abstract. It is up to the developer to code implementations of these according to the algorithm or functionnality he wants to add. For everyone of them, at least one default version has been implemented for the simulator to be able to work "out of the box".

Already implemented classes

Some classes of the simulator provide objects or functions which might be needed by other parts of the program, or are abstract classes which are already implemented with a default behavior satisfying most needs.

the SimulationManager (Fig. 4.3 on the following page) is the main part of the simulator. It initializes all its modules (*i.e.* the other managers) then runs an infinite loop calling the `step()` method of each at every timestep.

the IntroductionManager is in charge of holding the vehicles and introducing them on the crossroads when their is enough space left or according to some statistical laws. Currently, the most used is the `QuadPoissonIntroductionManager` which introduces the vehicles according to a Poisson law for each lane.

the VehiclesManager has the role of selecting the type of vehicle to be introduced in the crossroads (actually, to be introduced in the waiting queue of the `IntroductionManager`). The default implementation (`RouletteWheelVehiclesManager`) is based on a roulette wheel mechanism.

the VisualizationManager just provides a system to give feedback to the user. Currently, the `DefaultVisualizationManager` provides a simple Graphical User Interface as shown

Figure 4.3: The SimulationManager is the central class of the simulator.

on Figure 4.1 on page 19 while the `DummyVisualizationManager` just does nothing except printing a status line every 100 timesteps, for long simulations and results gathering purposes.

the TraceManager generates the 2D traces and finds the critical points. According to Appendix A on page I, the implementation, `ClothoidTraceManager`, uses clothoids to construct curves that the vehicles' dynamics can follow.

the Trace object (Fig. 4.4) is a representation of 2D traces and provides useful methods to ease their manipulation.

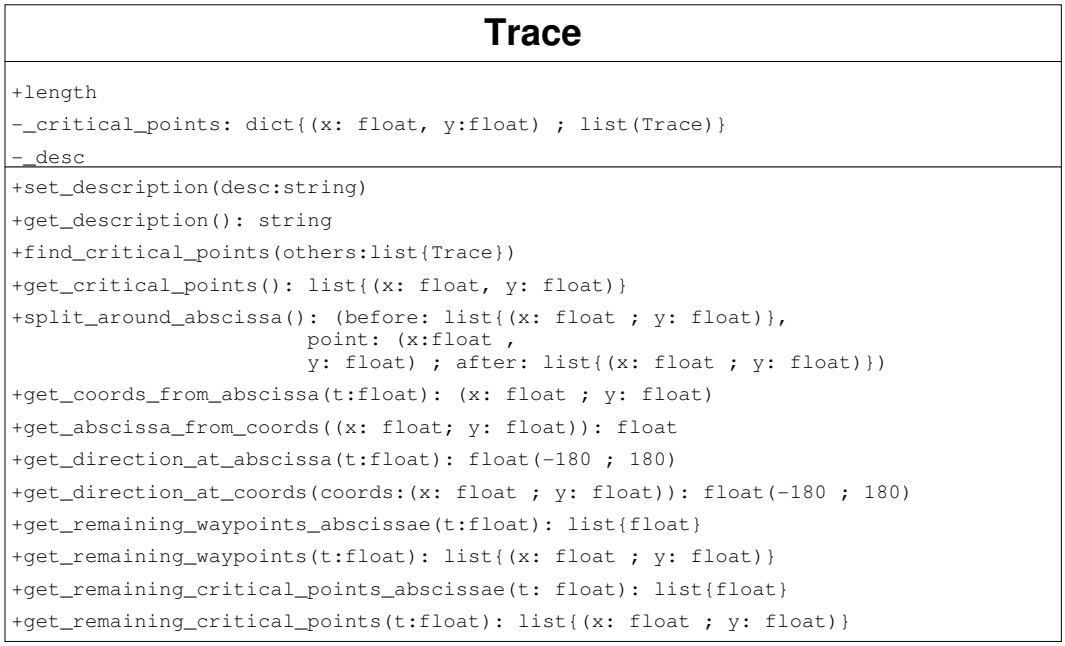

Figure 4.4: The Trace object exposes useful manipulation methods.

the TraceUser object (Fig. 4.5 on the following page provides a set of methods to help moving an object along a given trace.

the Trajectory object is used to add timing information to the `Traces`, in order to represent the full kynetics expected from the vehicle.

the Rectangle object (Fig. 4.6 on page 24) is a basic class providing geometric function. It is mostly intended to be used as a parent class for the `Vehicle`.

Classes important for the algorithm developer

There are two main classes that need to be in order to implement crossroads passing algorithms: `Vehicle` and `SpeedsManager`.

the SpeedsManager (Fig. 4.7 on page 24) is in charge of generating acceptable trajectories for the vehicles. It only has three interesting methods to keep track of the vehicles on the crossroads and generate the trajectories.

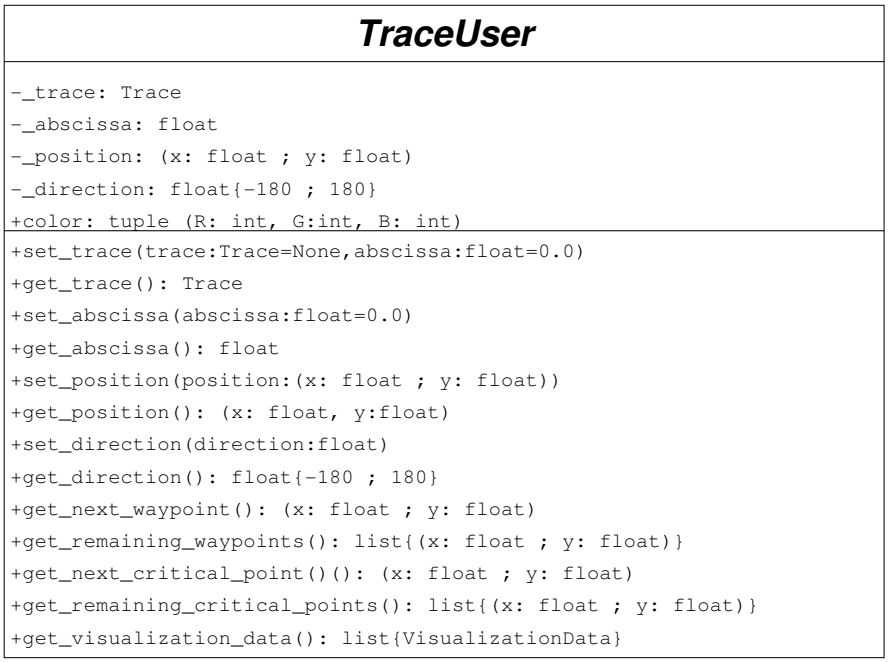

Figure 4.5: The TraceUser contains all the needed methods and properties to simulate the movement of an objet.

the Vehicle class (Fig. 4.8 on page 25) handles the movement of the cybercar along its trajectory. The default class can be used as-is, providing basic trajectory-following functions, but it may be necessary to extend the class in order to add algorithm-specific behaviors, like has been done in the `ReservationVehicle`.

4.2 Results

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 the 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[3] each[4]. One can note (Table 4.1 on page 26) that the dummy policy has a high throughput and

[3] in-simulation time
[4] 0.02s timestep

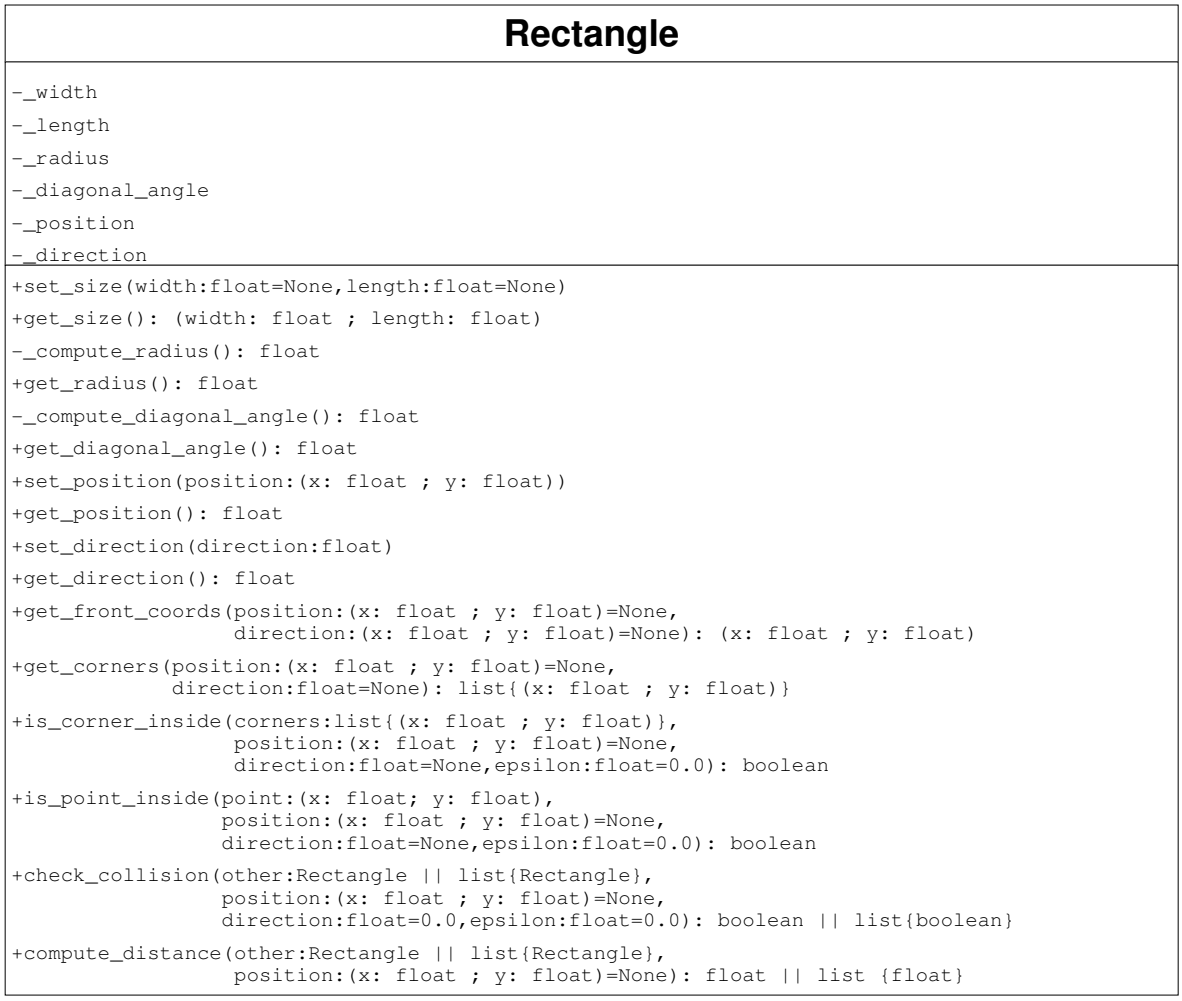

Figure 4.6: The Rectangle object provides useful geometric methods.

Figure 4.7: The SpeedsManager exposes only a few functions to propose trajectories to the vehicles.

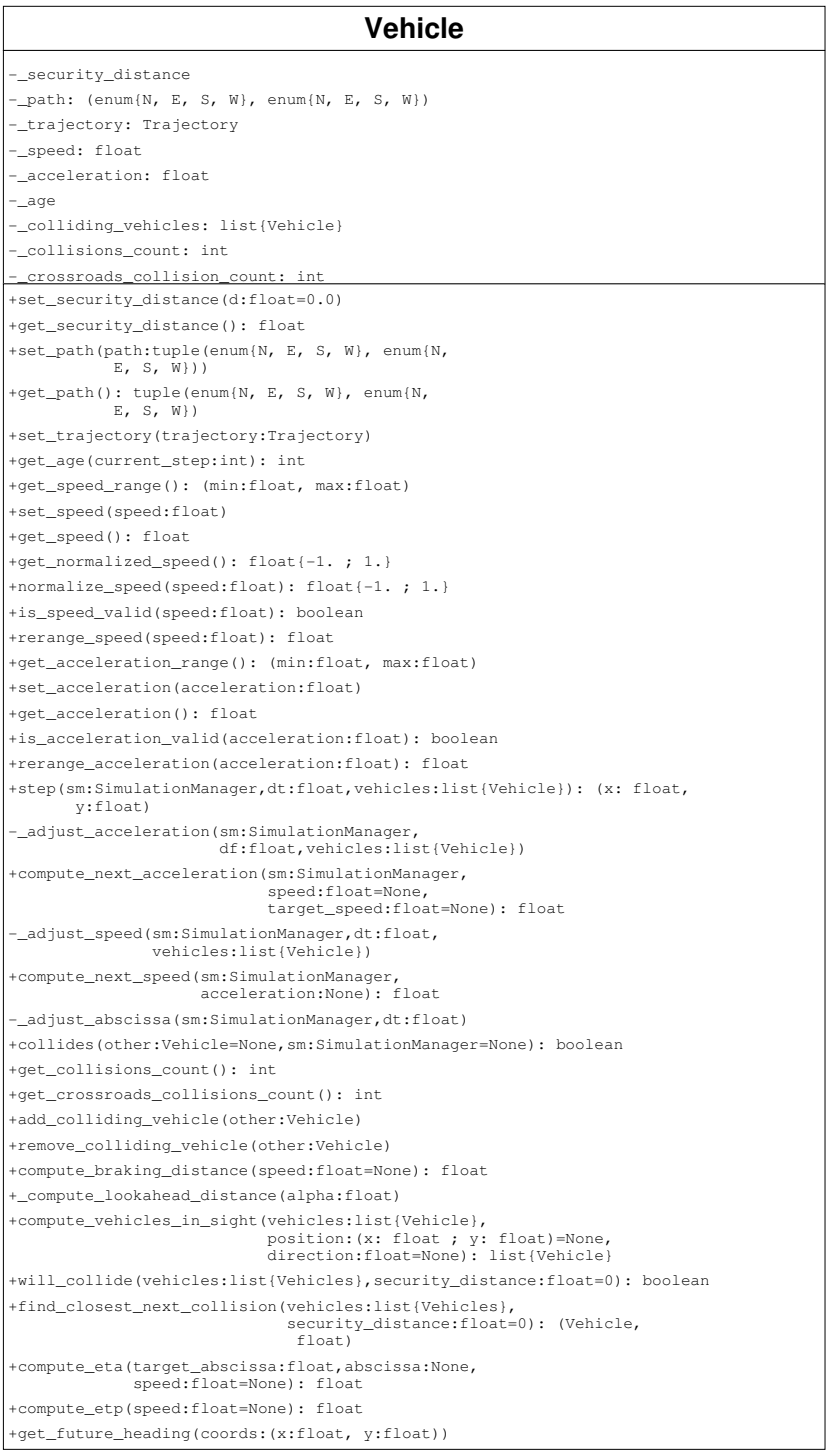

Figure 4.8: The Vehicle class already provides all the methods which may be needed to simulate a 4-wheeled vehicle. These may be used as-is or inherited in more algorithm-specific implementations.

	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 4.1: 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.

relatively low passing times, while the polling policy only has one quarter of the throughput and much higher times, but no collision.

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 critical point before passing it is the key to surely 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.

Chapter 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.

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).

As a conclusion, one can say that this thesis resulted in the first developments of quite an efficient algorithm, but some more work is still needed to take it out of the simulation and put it into Real World vehicles.

Bibliography

[1] Sandy Irani and Vitus Leung. Scheduling with conflicts, and applications to traffic signal control. In *SODA '96: Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 85–94, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[2] Carlos Gershenson. Self-organizing traffic lights. *COMPLEX SYSTEMS*, 16:29, 2004.

[3] 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.

[4] 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.

Appendix A

Model of a car and its trajectory

The following is a summary of the work of another intern at Imara, Hayet Ait Mebarek, who has been working on the modelization of a Cybercar and its attainable trajectories. This has served as a basis for the preliminary study of the 2D traces generator used in the simulator.

A.1 Dynamic model of a cybercar

An usual road vehicle is an *non-holonomic* vehicle. This means that its attainable moves are not only constrained by its position in time and space, but also by its velocity, momentum or direction.

For example, some of the constraints are expressed as non-integrable relations between the state of the vehicle and some of its derivatives.

A.2 Acceptable traces model

A.2.1 Dubin's curves

Dubins has proved that, for vehicles assimilable to bicycles (*bicycle model*, like the cars dealt with) with no obstacle in sight, the shortest path between two configurations is a curve. This curve can be generated by linking circles arcs by line segments. Such a curve is called a *Dubin's path*.

A.2.2 Clothoids

Clothoids (Fig. A.1 on page II) are a specific type of parametric curves which radius of curvature changes linearly. This is similar to what is obtained steering the wheel of a car.

These parametric curves can be expressed as

$$\left\{ \begin{array}{rcl} x(t) & = & a\sqrt{\pi} \cdot \mathrm{FresnelS}\left(\frac{t}{\sqrt{\pi}}\right) \\ y(t) & = & a\sqrt{\pi} \cdot \mathrm{FresnelC}\left(\frac{t}{\sqrt{\pi}}\right) \end{array} \right. , \tag{A.1}$$

where

$$\left\{ \begin{array}{rcl} \mathrm{FresnelS}(u) & = & \int_0^u \sin \frac{\pi t^2}{2} \, \mathrm{d}t \\ \mathrm{FresnelC}(u) & = & \int_0^u \cos \frac{\pi t^2}{2} \, \mathrm{d}t \end{array} \right. . \tag{A.2}$$

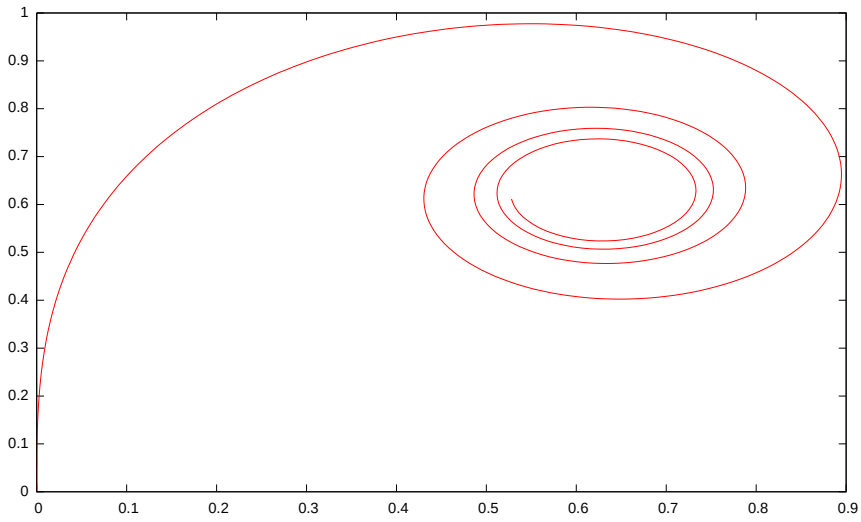

Figure A.1: An example of clothoid.

A.2.3 Generating traces

With these two elements, it is possible to construct traces adapted to what actual road vehicles can follow. Joining clothoids by segments (Fig. A.2) as proposed by Dubins allows to build complete paths on the studied itinerary (here, a crossroads).

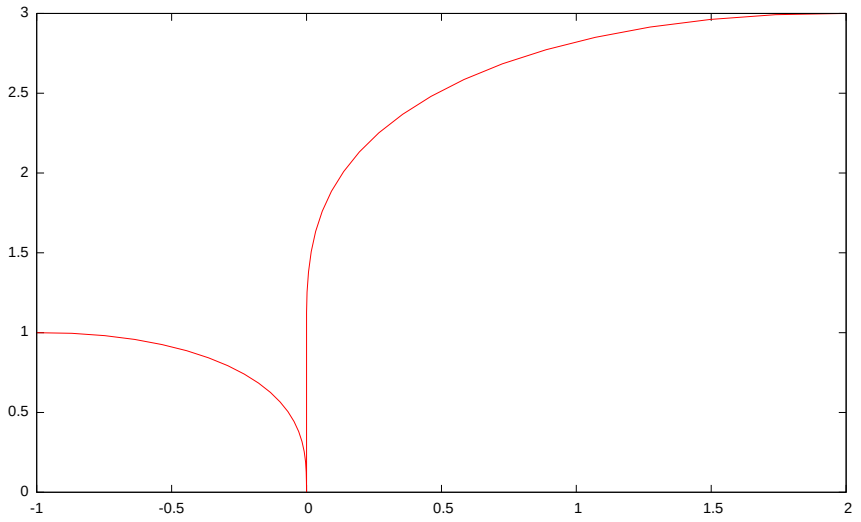

Figure A.2: Dubins' paths generated from clothoids.

List of Figures

1.1 Different types of cybercars 1

2.1 The two main platforms used at Imara. 6
2.2 The C3s' hardware architecture. 7
2.3 The CyCab's initial hardware architecture. 8

3.1 A regular crossroads. 10
3.2 2D traces on the crossroads. 13
3.3 The critical points on the regular crossroads. 14
3.4 A synopsis of the communications between a reservation agent and the crossroads supervisor. 15
3.5 The relation between the time values used when building a reservation request. 16

4.1 A screenshot of the crossroads simulator. 19
4.2 The main UML class diagram of the simulator. 20
4.3 SimulationManager UML description. 21
4.4 Trace object UML description. 22
4.5 TraceUser object UML description. 23
4.6 Rectangle object UML description. 24
4.7 SpeedsManager UML description. 24
4.8 Vehicle object UML description. 25

A.1 An example of clothoid. II
A.2 Dubins' paths generated from clothoids. II

List of Tables

3.1 The needed information pieces and their holders. 13
3.2 An example reservation request. 14

4.1 Reservation algorithm performances compared to others. 26

Index

Ait Mebarek, Hayet, I
Benenson, Rodrigo, i, 1
Bouraoui, Laurent, i, 6
La Fortelle (de), Arnaud, i
Parent, Michel, 4
Yvet, Armand, i
2D trace, I, 12, 14, 22

Adas-RP, 6
artificial intelligence, 5

bicycle model, I

camera, 6, 8
CAN bus, 6
CAOR, 5
CGFTE, 4
Citroën C3, 6
class
    IntroductionManager, 20
    Rectangle, 22, 24
    SimulationManager, 20, 21
    SpeedsManager, 22, 24
    Trace, 22
    TraceManager, 22
    TraceUser, 22, 23
    Trajectory, 22
    Vehicle, 22, 23, 25
    VehiclesManager, 20
    VisualizationManager, 20
clothoid, I, II, 12, 22
critical point, 13, 14, 16, 17, 27
crossroads, II, 1, 9–12, 14, 15, 17, 19, 22, 23, 27
cybercar, I, 1, 6, 9, 10, 18, 23, 27
CyCab, 1, 4, 6, 8

Dassault Electronique, 4
deadlock, 9

École des Mines, 5

EDF, 4
embedded systems, 3, 5
expected time of arrival, 16
expected time to pass, 16

framework, 9

GPRS, 6

Imara, i, I, 1, 3–8
infrared device, 8
infrastructure, 11
Inrets, 4
Inria, i, 3–6, 18
intelligent transportation systems, 4

joint research unit, 5

La Route Automatisée, 5
level, 9, 12
    macroscopic, 9
    mesoscopic, 9, 18
    microscopic, 9, 10, 12, 18, 27

multiagent traffic management system, 11

navigation, 6
network, 6
non-holonomic, I

path, I, II, 2, 5, 9
Praxitèle, 4

Renault, 4
reservation, 11–17, 23, 26
reservation system, 12

security factor, 16
signal processing, 5
simulator, I, 18–20
software agent, 11
supervisor, 13–15, 17

traffic lights, 11, 12
trajectory, 9

ultrasonic device, 8

vehicle agent, 13, 14, 16–18

wireless, 5, 6