Visual-Guided Planning and Control for a Non-Holonomic Robot

N.T.U. Technical Report

Alexis Scheuer, Research Fellow in
"Vision and Control" Strategic Research Program,
School of Mechanical and Production Engineering,
Nanyang Technological University (Singapore)

September 11, 1998

Abstract:

This technical report summarizes the work I have achieved in the "Vision and Control" Strategic Research Program, directed by Dr. Xie Ming, from February to September 1998. It is articulated in two parts, the first one describing my team work and the second presenting my research work.

In the first part, I explain how I worked to improve the collaboration between NTU and the French institute Inria, and to install some tools bringing the power of Unix (or Linux) to the PC of the "Vision and Control" Strategic Research Program. On another hand, the research part of this report proposes an algorithm to integrate, on a Nomad 200^{TM} robot, vision sensing with a control and planning system adapted to the robot. If practical results have not been as successful as anticipated, theoretical results and implementations should allow to achieve the aimed objectives soon.

Table of Contents

Introduction

Former student from the École Normale Supérieure de Lyon (one of the highest French "Grande École"), I did a Ph. D. thesis in the Sharp project, directed by Ch. Laugier, from the Gravir Laboratory and the Inria Rhône-Alpes [5]. This Ph. D. thesis deals with "Continuous-Curvature Path Planning for Non-Holonomic Mobile Robot": it addresses the problem of planning more accurately paths for car-like robots[1].

Classical path planners for car-like robots computes paths made of circular arcs tangentially connected by line segments. These paths have a discontinuous curvature profile, and therefore cannot be followed precisely without stopping at each curvature discontinuity, to reorient the directing wheels. If this is not a problem when the car is supposed to do backup maneuvres, it is not a good solution when the car goes only forward (without maneuvre).

My Ph. D. thesis defines a new problem, adding to the classical problem a curvature continuity constraint and a bound on the curvature's derivative. This last bound represents the maximum velocity of the directing wheels' reorientation, and it prevents to have solutions too close from the ones with discontinuous curvature. The existence of solutions to this problem and their nature are studied, and an algorithm to find a solution is presented **in the case without maneuvre** (*i.e.* when the robot moves only in the forward direction).

In this case, the continuous-curvature path planning algorithm is compared to the classical one (giving paths with a discontinuous curvature), with respect to their complexity, their computation time and how accurately they can be followed using a classical control method (namely a Kanayama method [3]). While the complexity and computation time of the two planners are similar, the following of the continuous-curvature paths is more than ten times better than the following of the discontinuous-curvature paths.

My research fellowship at Nanyang Technological University, as part of the collaboration with the French Inria institute, was an occasion to present this planning method to an other research group and to implement it on a new experimental platform (it has already been used in Inria for the electrical self-driving car). It was also an occasion for me to learn more about vision and control, and to work on IBM PC rather than on Unix workstations (Sun or Silicon Graphics), as I did for almost ten years.

[1] A non-holonomic robot is a robot respecting constraints that are not integrable, and thus cannot be eliminated by considering a special workspace. A car-like robot is a good (hard) example of non-holonomic mobile robot.

Overview of this Report

This report is divided in two main parts, one describing my team work (I tried to emulate the French–NTU collaboration, and to enhance the working utilities, see section 1) and the other presenting my research work (description of the experimental platform, and of the algorithms implemented, see section 2).

1 Team Work

As I worked in the French Inria institute during my Ph. D. thesis, I know the reseach and technical staff there and thus have been a natural correspondent between the "Vision and Control" Strategic Research Program and this institute (§ 1.1). On another hand, as I administrated Unix computers for a few years, I used my knowledge to enhance the installation of the computers in the new "Autonomous Vehicle Laboratory", adding a set of Unix-like tools under NT (§ 1.2) or installing a dual-boot with Linux on some of the PC (§ 1.3).

1.1 Collaboration NTU – Inria

My coming in the "Vision and Control" Strategic Research Program as a research fellow was a prelude to the NTU – Inria collaboration. It helped to tighten the relationship between Dr. Xie Ming's team and Inria Rhône-Alpes's staff (mainly the robotic staff and the Sharp project).

The "Vision and Control" Strategic Research Program will soon receive an experimental electrical vehicle called Cycab, similar to the one developed by the robotic staff at Inria Rhône-Alpes and used by the Sharp project. My first task was to engage a dialog between the robotic staff and Dr. Xie Ming, in order to choose a system architecture more adapted to the goals of the "Vision and Control" Strategic Research Program, and to identify the sensors then needed. This also lead me to take contact with RoboSoft, the French company that will deliver the Cycab.

On another hand, I kept numerous contacts with the Sharp project, from which a Master student (namely Frédéric Large) came for a training course in order to finish his engineer school (the French École Supérieure d'Ingénieurs de Chambéry, or ESIGEC). Mr. Large is working in the "Intelligent Vehicle Projet", a research group between the School of Mechanical and Production Engineering (vision subgroup of the project), the School of Applied Science (planning subgroup) and the School of Electrical and Electronic Engineering (control subgroup). Under the supervision of Michel Pasquier (from the School of Applied Science), he is developing a three-dimensional simulator which will be used by the "Intelligent Vehicle Projet". As I knew Christian

Laugier, Mr. Large's French research director, and as I already met Mr. Large, I helped to prepare his arrival and his settle down in Singapore.

1.2 Compilation under Windows NT

Dr. Xie Ming, directing the "Vision and Control" Strategic Research Program, asked me to compile my Ph. D. thesis work under NT, so that it can be used by other students of the team. This should not have been a problem, as my programs are in C++ language (which can be easily compile on many platforms), using as base a library called LEDA (which has also been compiled for a large set of platforms, including PC under NT).

However, it took me a long time to complete this stage. First of all, I needed to download the LEDA library compiled under NT. This has been a problem, as this library is quite large (the compressed archive is about 4 Megabytes) and the network bandwidth seems to be low in Singapore. Moreover, for an unknown reason, the downloading stage have been repeatedly stopped before its end (after a random time), and had to be done again. Thus the download stage, which could have taken only a few minutes, needed about a week to be completed.

I also had to install a new version of *Microsoft's Visual C++*, as the LEDA library had been compiled for a newer version than the one installed on my PC. It did not take me too long, as I have been able to find this version on a CD-Rom. Then, in order to compile the LEDA library's demonstration programs (to verify that the library was well compiled and installed), I needed a few Unix-like tools. Once again, it took me several days to download the 2 Megabytes compressed archive, but the harder was to configure NT in order for the tools to work well. I was more used to Unix operating system than to Windows NT, and I have not been able to find anybody to help me to get used to NT.

At last, once the LEDA library has been successfully installed, I tried to compile my Ph. D. thesis work. Then, I discovered that *Microsoft's Visual C++* does not accept some of the standards in C++: the suffixes I used (`.hh` and `.cc`) were not recognized (they are however standards), and thus the hierarchical dependence between the files was not understood; moreover, classical makefiles where not accepted ("they have not been generated by *Visual C++*")...

As a conclusion, I decided to compile my work using the `make` function and the compiler of *Visual C++*, but not the rest of this tool. It took me a few days to be able to modify my makefiles and my system settings in order to compile my Ph. D. thesis work. **Then**, I had another surprise with *Microsoft's Visual C++*: if the LEDA library's demonstration programs have been correctly compiled using the `make` function and the compiler of *Visual C++*, it is not the case with my programs. The source code of these programs is correct (it works under several Unix operating systems, *e.g.*

BSD, Solaris and Irix, and under Linux), but *Visual C++* tools are not able to compile it: the resulting application randomly does nothing, generates a `system error` or blocks the computer.

As this work was not of major interest, and as trying to find why *Visual C++* does not compile correctly my programs could be done by somebody more used to Microsoft's tools (in fact, it should be done by Microsoft itself), Dr. Xie Ming asked me to stop to work on this point and to focus on my research work.

1.3 Installation of Linux Operating System

However, my research work implies an implementation under Linux operating system. Thus, I had to install this operating system on the PC I was using, together with Windows NT operating system, using what is usually called a *dual-boot*.

Once again, this should not have been a problem. But, for another unknown reason, the dual-boot cannot be installed in the usual way on the computers bought for the new "Autonomous Vehicle Laboratory". Thus, it took me several weeks to find a way to install this dual-boot correctly, as nobody (neither in the "Robotic Reseach Center", the "Computer Center" nor in Inria) has ever heard of such a problem. At last, after a long search in the various guides and "how to...", I found a solution.

Since then, I installed a similar dual-boot on two other computers in the "Autonomous Vehicle Laboratory", using the same method (and verifying twice again that the classical method does not work). During the second installation, I explained the unusual steps of this method to a technician from the "Robotic Reseach Center" and to an other research fellow from the "Vision and Control" Strategic Research Program (namely Dr. Guo Dong), and I gave them the address of the HTML page which describe this method (`http://www.ntu.edu.sg/home/mascheuer/Tools/main-eng.html#Linux`). Both should now be able to repeat the installation if needed, without my help.

2 Research Work

My main work in the "Vision and Control" Strategic Research Program has been on "Visual-Guided Planning and Control for a Non-Holonomic Robot". I used a Nomad 200$^{\text{TM}}$ robot, which is a non-holonomic mobile robot, even if it can be considered for planning as a holonomic mobile robot (*i.e.* a robot moving without constraints). I will describe more precisely the Nomad 200$^{\text{TM}}$ in the next section (§ 2.1).

First of all, I studied the robot controller (defined by Nomadic Inc., the company selling the Nomad 200$^{\text{TM}}$) and used my Ph. D. thesis work in order to define a simple and user-friendly interface to move the robot (§ 2.2).

Then, a camera has been installed on this robot, along with a frame grabber board and programs. I developed an interface in order to manipulate easily the frame grabber program, to calibrate it and to identify obstacles in front of the Nomad 200^{TM} robot (§ 2.3). Integrating these two work will allow the robot to move from a position to an other, avoiding the obstacles that are in its way (§ 2.4).

2.1 Presentation of the Robot

Build by Nomadic Inc., the Nomad 200^{TM} is a roughly cylindrical mobile robot, mounted on three wheels. Each of these wheels can turn, but the three always remain parallel. Thus, the turning radius of this robot is unconstrained (the robot can turn without moving). As a consequence, even if the robot is non-holonomic (it can only translate in the direction of its wheels), it is highly maneuverable (it can follow any type of curve).

Figure 1: the Nomad 200^{TM}.

The lower part of the robot, around and just above the wheels, is fixed. Around this part, two pressure-sensitive bumpers ("the Sensus 100^{TM} Tactile System") allow to detect any collision. Above this part, the *turret* of the robot can turn around its vertical symmetry axis. The base of this turret contains 16 independent sonar units ("the Sensus 200^{TM} Sonar System") with an effective range from 15 to 647 cm. On the upper part of this turret, 16 independent analog infrared sensors ("the Sensus 300^{TM} Infrared System") can be used to detect obstacles nearer than 60 cm, with a 10 Hertz frequency. At last, a camera is mounted on top of the turret, pointing toward the ground at a distance of about two meters of the robot.

This camera is plugged, through a Matrox Meteor frame grabber, to a standard Linux-based PC (namely "the Nomad Control System^{TM}") which is, in turn, connected to the network through a wireless Ethernet link. On

this PC, Nomadic Inc. has developed a set of tools (the "Nomad Software Development System^{TM}") to manipulate the robot, including:

- a low-level controller which allows the user to command the translation of the robot, its steering and the orientation of its turret, through a language interface;
- a simulator;
- and a graphic interface to visualize the motion of the simulated robot or of the real one;

2.2 Control

In this section, I will describe the work I achieved in order to develop a high-level controller for the Nomad 200^{TM}. First of all, we will see how the robot is modeled (§ 2.2.1), then I will describe the robot's optimal motions (§ 2.2.3). After that, I will show how the low-level controller installed by Nomadic Inc. limits the motions that can be executed (§ 2.2.4). At last, I will present the C++ (high-level) interface I developed in order to control more easily the robot (§ 2.2.5).

2.2.1 Model of the Robot

As this robot will only move on a planar horizontal ground, the workspace $\mathcal{W}$ is represented by a bounded closed subset of the plane $\mathbb{R}^2$. A *geometric configuration* of the robot (*i.e.* the specification of the position of every point of this robot) can then be represented simply. The robot's symmetry axis (the axis of the cylinder) will remain vertical, and thus a position of the robot is given by the coordinates of the projection of this axis on the ground. A geometric configuration needs also to specify the orientation of the wheels of the robot (they remain parallel, *cf.* Fig. 2) and the orientation of the turret. However, **in our case**, as the robot will always look (*i.e.* direct its camera) in the direction of its motion (which is the direction of its wheels), **the orientation of the wheels and the one of the turret will remain equal**.

Thus, a geometric configuration of the Nomad 200^{TM} in the workspace $\mathcal{W}$ will be modeled by an oriented circle, noted $\mathcal{A}$, of fixed radius $R_{\mathcal{A}}$ (*cf.* Fig. 2). This oriented circle (and thus the geometric configuration of the robot) is represented by a three dimensional vector $(x, y, \theta) \in \mathbb{R}^2 \times \mathcal{S}^1$, the two first coordinates giving the position of the circle's center and the third being the orientation of the circle (*i.e.* the angle between the x-axis and the wheels' direction).

As we want the robot's wheels to roll without sliding, its motion has to respect a non-holonomic constraint: at a given geometric configuration, its

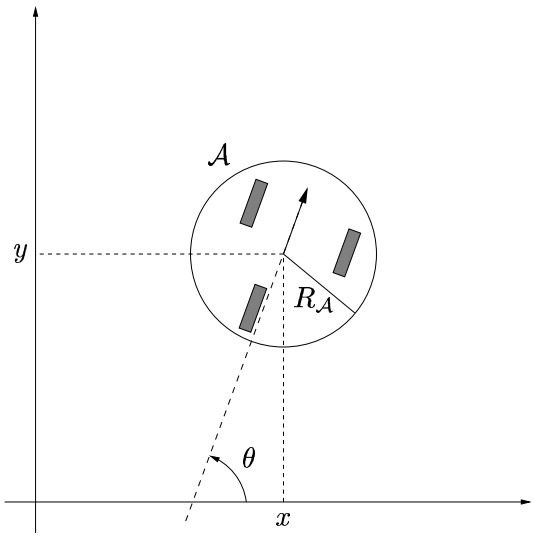

Figure 2: Geometric Configuration of the Nomad 200^{TM}.

instantaneous velocity is always parallel to its wheels' direction. Using the notations previously defined, this constraint can be written in the workspace $\mathcal{W}$ as:

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{1}$$

However, a sequence of geometric configurations of $\mathcal{A}$ is not enough to represent precisely the robot's motion. We are also interested in the robot's dynamics, and thus we have to add the translational, steering and turret's turning velocity to the configuration's parameters. Once again, as the turret's turning velocity will remain equal to the steering velocity (because the wheels and the turret keep the same orientation), it is ignored.

Let us call *dynamic configuration* the combination of the geometric configuration's parameters and of its derivatives (w.r.t. time). A dynamic configuration of $\mathcal{A}$ is thus a 5 dimensional vector $(x, y, \theta, v, \omega) \in \mathbb{R}^2 \times \mathcal{S}^1 \times \mathbb{R}^2$, where x and y are the coordinates of the robot's vertical symmetry axis, θ is the orientation of its wheels and turret, v is its translational velocity and ω is its steering velocity (and its turret rotating velocity).

The non-sliding constraint (1) can then be rewritten in the dynamic configuration space as:

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ y \\ \theta \\ v \\ \omega \end{pmatrix} = \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ a \\ \gamma \end{pmatrix} \tag{2}$$

where a and γ are respectively the translational and steering (or turret) accelerations, and are the control parameters. The robot has to respect an other constraint, giving its dynamic limitations (*i.e.* its maximum velocities and accelerations):

$$\begin{cases} |v| & \leq & v_{\max} \\ |\omega| & \leq & \omega_{\max} \\ |a| & \leq & a_{\max} \\ |\gamma| & \leq & \gamma_{\max} \end{cases} \tag{3}$$

2.2.2 Statement of the planning problem

We just saw that the motions of our robot $\mathcal{A}$ are limited by non-holonomic (*i.e.* non-integrable) constraints. Thus, to move from a configuration to an other one while avoiding obstacles, $\mathcal{A}$ can only follow a restricted set of paths. It will then need a *planner* to find among these paths the optimal one.

To define formally the *planning problem* we need to solve, let us denote $\mathcal{C}$ the configuration space, *i.e.* the subset of $\mathbb{R}^2 \times \mathcal{S}^1 \times \mathbb{R}^2$ containing all the configurations for which $\mathcal{A}$ is in the workspace $\mathcal{W}$. If q is a configuration in $\mathcal{C}$, let $\mathcal{A}(q)$ be the (circular) region occupied in $\mathcal{W}$ by $\mathcal{A}$ when it is in configuration q. Then, if $\mathcal{B}$ is an obstacle in $\mathcal{W}$, we say that q is *in collision* with $\mathcal{B}$ if, and only if, $\mathcal{A}(q) \cap \mathcal{B} \neq \emptyset$.

If the workspace $\mathcal{W}$ is clustered with a set of obstacles $\mathcal{B}_j$, $j \in \{1, ..., n_\mathcal{B}\}$, let us denote $\mathcal{C}_{\text{collision}}$ the set of all the configurations of $\mathcal{C}$ that are in collision with an obstacle $\mathcal{B}_j$, $j \in \{1, ..., n_\mathcal{B}\}$, and let $\mathcal{C}_{\text{free}}$ be its complement:

$$\mathcal{C}_{\text{collision}} = \{q \in \mathcal{C} \ / \ \exists\, j \in \{1, ..., n_\mathcal{B}\}, \mathcal{A}(q) \cap \mathcal{B}_j \neq \emptyset\}$$

$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{collision}} = \{q \in \mathcal{C} \ / \ \forall\, j \in \{1, ..., n_\mathcal{B}\}, \mathcal{A}(q) \cap \mathcal{B}_j = \emptyset\}$$

Moreover, if ε is a positive real value, $\mathcal{C}_{\text{free}}^{\varepsilon}$ is the set of all the configurations of $\mathcal{C}$ which distance to $\mathcal{C}_{\text{collision}}$ is greater than (or equal to) ε.

We call *path* a continuous curve in $\mathcal{C}$. A *collision-free path* is then a continuous curve in $\mathcal{C}_{\text{free}}$. Moreover, a path is said *feasible* for $\mathcal{A}$ if, and only if, it respects the motion's constraints of this robot, *i.e.* if, and only if, it respects the equations (2) and (3).

A formal statement of the planning problem we consider is then the following:

> Being given a starting configuration q_s, a goal configuration q_g, the set of obstacles $\mathcal{B}_j$, $j \in \{1, ..., n_\mathcal{B}\}$, and a real $\varepsilon > 0$, we search a path Π which:
>
> - connects q_s to q_g (Π is a curve in $\mathcal{C}$);
> - is feasible for $\mathcal{A}$, *i.e.* respects the equations (2) and (3);

- is a curve in $\mathcal{C}_{\text{free}}^{\varepsilon}$, *i.e.* whose distance to the obstacles remains greater than ε.

Remarks:

- ε can be interpreted as a "security distance" to the obstacles; the fact that the search is limited to $\mathcal{C}_{\text{free}}^{\varepsilon}$ (instead of only $\mathcal{C}_{\text{free}}$) will also ensure of the existence of a time-optimal solution (whenever a solution exists, see § 2.2.3);

- the search for a solution path is done in $\mathcal{C}$ instead of in $\mathcal{C}_{\text{free}}^{\varepsilon}$ (the projection of the obstacles $\mathcal{B}_j$, $j \in \{1, ..., n_{\mathcal{B}}\}$, in the configuration space $\mathcal{C}$ is not computed, as it is too long), the collision avoidance being verified in $\mathcal{W}$; it is however important to notice that, due to equation (3), $\mathcal{C}$ is now a subset of $\mathcal{W} \times \mathcal{S}^1 \times [-v_{\max}, v_{\max}] \times [-\omega_{\max}, \omega_{\max}]$ and is a compact set, as well as $\mathcal{C}_{\text{free}}^{\varepsilon}$ for all $\varepsilon > 0$, while $\mathcal{C}_{\text{free}}$ is an open set.

2.2.3 Controllability and optimality

Now that the planning problem we consider is formally stated, let us study some of its properties.

First of all, the robot $\mathcal{A}$ we modeled can move as a holonomic robot: $\mathcal{A}$ can turn without moving, and it can follow straight lines. It means that a solution path exists if, and only if, q_s and q_g are in the same connected part of $\mathcal{C}_{\text{free}}^{\varepsilon}$. However, the simplest paths (made of pure rotations and of straight line motions) are not *optimal* (*i.e.* the shortest in time).

When a solution path exists (*i.e.* as soon as the configurations to connect are in the same connected part of $\mathcal{C}_{\text{free}}$), it is interesting to find the best one. In our case, it will be the path whose execution time will be the smallest. This can be considered as a Lagrange optimization problem [1, Chap. 5], for which the conditions of the Filippov Theorem adapted to this class of problems [1, Th. 5.1.ii or 9.3.i] are respected (for more details, refer to § A.1 in appendix). As a consequence, whenever a solution path exists, a solution path with minimum execution time can be found. This is the case as the search is limited to $\mathcal{C}_{\text{free}}^{\varepsilon}$, which is a compact set, instead of $\mathcal{C}_{\text{free}}$, which is open: indeed, if the search would have been limited to $\mathcal{C}_{\text{free}}$, it would always have been possible to find a path shorter than any given one (simply by taking a path closer to the obstacles).

However, for a given problem, the characterization of the optimal solution path is difficult. The Pontryagin Maximum Principle [4] gives necessary conditions respected by the portions of the optimal paths that are inside the boundaries of the search space (here the configuration space), *i.e.* in our case when either the linear or the angular velocity (v or ω) remains maximum

(in absolute value). These conditions constrain a vector dual to the configuration vector, and are therefore difficult to clearly apprehend.

Using this principel, I have been able to prove that, along optimal paths, either the linear **or** the angular acceleration (*i.e.* a or γ) remains equal to its maximum (in absolute value), except when an obstacle's border is followed; details are given in § A.2 in appendix. Nevertheless, I have the intuition that both acceleration should be maximum (in absolute value) or null (accelerations are then said to be *bang-bang*): in the following, I will assume that this the case.

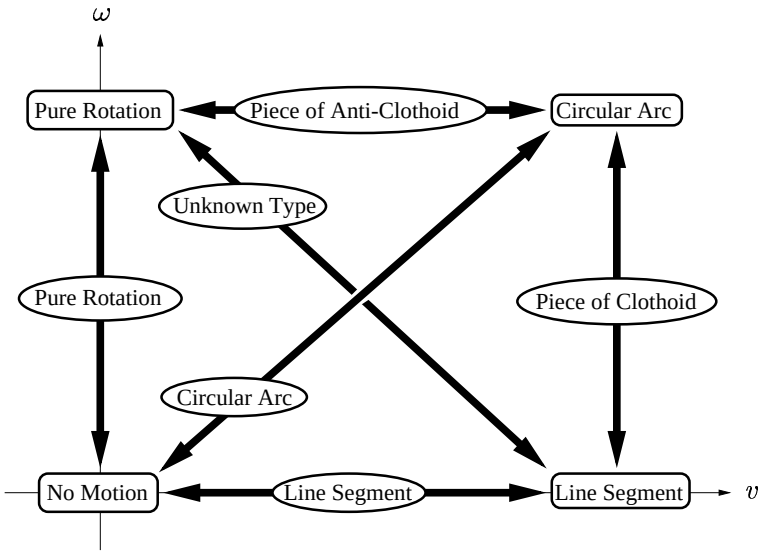

Figure 3: the Curves Followed with Optimal Control.

Fig. 3 indicates the curves followed in the plane by the projection of the vertical symmetry axis of the robot $\mathcal{A}$, when control is optimal (as defined in the previous paragraph). In this figure, rounded boxes correspond to motions for which both (linear and angular) accelerations are null (linear and angular velocities are constant), while ellipses correspond to motions with (at least) one maximum acceleration (in absolute value).

The nature of the optimal curves is computed in the following way:

- it is trivial when the linear velocity v is null;
- it is easily found considering that the curvature κ along the curves is (if s is the arc length):

$$\kappa = \frac{\mathrm{d}\theta}{\mathrm{d}s} = \frac{\mathrm{d}\theta}{\mathrm{d}t}.\frac{\mathrm{d}t}{\mathrm{d}s} = \frac{\omega}{v}$$

The line segments correspond to a null curvature, the circular arcs to a con-

stant (not null) curvature, the clothoid[2] pieces to a linear curvature (w.r.t. the time) with constant linear velocity and the anti-clothoid (or involutes of circles, as defined by Fleury *et al.* in [2]) pieces to a linear radius of curvature (the inverse of the curvature) with a constant angular velocity (see Fig. 3).

One of these optimal curves have not yet been identified: when the linear velocity v decreases (resp. increases) with maximum linear acceleration a (in absolute value) while the angular velocity ω increases (resp. decreases) with maximum angular acceleration γ (in absolute value), the coordinates of the curve followed by the robot $\mathcal{A}$ is a combination of the coordinates of a circle and of those of a clothoid. However, in the following, we will only use the curves of the lower right half of Fig. 3, *i.e.* linear segments, circular arcs and pieces of clothoids, as we want to maintain the linear velocity v to its maximum as long as possible.

2.2.4 Nomadic's Controller

As indicated in § 2.1, the Nomad 200$^{\text{TM}}$ has already a low-level controller, developed by Nomadic Inc. We call it a *low-level* controller as it only accepts low-level commands: it is possible to reach a desired (translational, turning or turret's) velocity or to move the robot (resp. turn the wheels or the turret) for a given distance (resp. angle), but it is not possible to ask to reach a precise configuration (or even position).

The definition of a *high-level* controller (*i.e.* a controller accepting the low-level commands as well as more sophisticated commands) can only be done using the low-level controller: this controller is the only way to access to the robot actuators. Thus, before defining the high-level controller, we need to study the low-level controller.

The study of this controller raised problems which are still unsolved:

- First of all, the simulator developed by Nomadic Inc. does not take into account the acceleration bounds, which can although be specified using the function `ac` (*cf.* the "User's Manual"). Indeed, velocity variations are instantaneous, which correspond to infinite accelerations.

- The controller on the real robot does not seem to have a better behaviour. If the velocity variations are not instantaneous, they correspond to extremely variable accelerations, which are about ten times the maximum fixed by the function `ac`.

- Among the two main control mode, the *velocity mode* (*i.e.* the function `vm`, fixing the translational, steering and turret velocities to reach) does work well. However, the *position relative mode* (through the

[2] A clothoid, or Cornu spiral, is a curve whose curvature is a linear function of its arc length.

function `pr`, which gives the distance to move, and the angles to turn the steering and the turret) does not have the supposed result.

The first two points prevent us from using the continuous-curvature path planner, as the continuous variations of the curvature cannot be planned. Nomadic is nowadays working to solve the first problem: the results presented in this report (mainly those presented in § 2.2.3) may interest their research team. However, they did not give us any explanation about the second point.

Moreover, as said in the third point, the *low-level* controller does not work as it is supposed to, and it can not be used **in any way** to control correctly the robot. Numerous experiments, to try to understand how the robot can be controlled (or how Nomadic's controller works, as its behaviour does not correspond to what the "Language Reference Manual" describes), have been driven.

Unfortunately, the DC/DC converter (which supply power from the batteries to the Linux-based PC) inexplicably burnt during one of those experiments. Procedures to replace this (vital) part are still in progress.

2.2.5 A Higher-Level Controller

During these numerous experiments, and in order to program them more easily, I developed in C++ a friendly-user interface. This interface mainly handle automatically the connection to the robot (through or without a server, using the old or the new version of the direct connection function), but transforms also the C functions (as defined in Nomadic's controller) in C++ functions with structured parameters (instead of lists of low-level type parameters), default values for these parameters, and save the important values (which where not accessible in Nomadic's controller) concerning the robot's control (maximum velocities and accelerations, robot identity number).

This interface has also been connected to the planners developed during my Ph. D. thesis, to define a higher-level controller which can compute how to reach a specified position or configuration while avoiding a set of obstacles. The low-level (*i.e.* Nomadic's) controller is then used to follow the planned path, while verifying that the distance from the robot to any obstacle remains greater than a specified value ε (using the infrared sensors) and that no bumper is hit. However, as long as Nomadic's controller does not react as it is supposed to (or as long as its real behaviour has not been understood and taken into account), this high-level controller does not work correctly (due to Nomadic's controller).

2.3 Vision

The controller we tried to implement in the previous section has to be connected to a vision system, to detect and localize obstacles and to select intermediate geometric configurations. Research and development concerning the vision part of this system has not been carried as far as for the planning and control part (presented in the previous section).

In this section, we will first describe the hardware use for the vision system (a Meteor frame grabber, see § 2.3.1), then we will present the mathematical model used to represent the projection on the images' plane (§ 2.3.2). At last, we will show how this system can be initialize (§ 2.3.3) and how obstacles can be detected and localized (§ 2.3.4).

2.3.1 The Meteor Frame Grabber

As described in § 2.1, the robot is equipped of a camera which is connected through a Meteor frame grabber to the computing system. However, only a rough interface (in C) has been developed to control this frame grabber. Moreover, the absence of comments in the source code of this interface made it difficult to use.

Thus, I implemented a cleaner and more user-friendly interface in C++, with numerous comments to explain how functions should be used and how they work (I also added comments to explain what I understood of the basic interface in C). The resulting class has various methods, from low-level ones (to control the frame grabber, which is accessed through a device, using the `ioctl` function, *i.e.* the input/output device control function) to high-level ones (connection and initialization of the Meteor frame grabber, modification of the geometry of the output images, etc...).

In addition, to display the images grabbed by the Meteor card (and later verify the image treatment), a simple implementation of the graphical X-windows was needed. Once again, a C++ class has been defined in order to interface more easily the classical C functions and to handle the needed variables.

Both of these steps have been an occasion for me to learn more about Unix (or Linux), and more precisely about device control function (`ioctl`) and basic X-windows managing functions. I hope that the comments written in my code will help the next person who will use the libraries I developed to learn more easily and faster how those functions work.

2.3.2 Model of the Vision

The transformation from the real world coordinates to the video camera images coordinates is approximated by the combination of a projection on the plane $\mathcal{P}$ (parallel to the images' plane $\mathcal{P}_I$, and thus perpendicular to the camera's main axis Δ), and of a homothetic reduction. This combination

is a linear approximation of the real transformation (which is non-linear), and remains correct as long as the original point (projected in the frame of the video camera images) is close enough of the plane $\mathcal{P}$. This can easily be assumed, as the camera does not have a wide view angle.

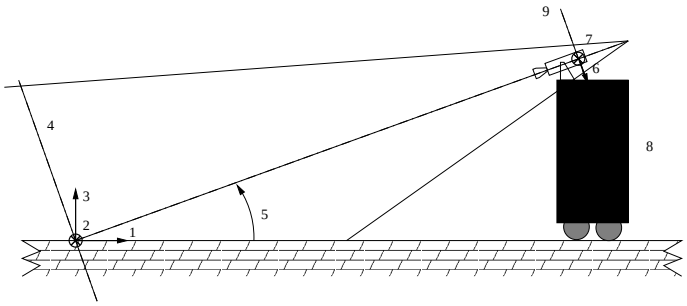

Figure 4: the Vision Transformations.

Let φ be the angle between the camera's main axis Δ and the ground floor $\mathcal{F}$ (considered as a plane), and let λ be the homothetic factor of the image reduction. We denote $\mathcal{R}$ the three-dimensional frame centered at the intersection between Δ and $\mathcal{F}$, which x-axis is the projection of Δ on $\mathcal{F}$ and z-axis is perpendicular to the ground $\mathcal{F}$ (*cf.* Fig. 4). The image frame $\mathcal{R}_I$ is chosen in the images' plane $\mathcal{P}_I$, centered on the camera's main axis Δ, with its x-axis being directed downward in the vertical plane and its y-axis being directed on the robot's right in the horizontal plane.

Thus, the transformation from the coordinates (x, y, z) in the real world's frame $\mathcal{R}$ to the coordinates (x_I, y_I) in the image frame $\mathcal{R}_I$ is:

$$\begin{pmatrix} x_I \\ y_I \end{pmatrix} = \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} \cdot \begin{pmatrix} \sin\varphi & 0 & -\cos\varphi \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{4}$$

2.3.3 Calibration of the System

The parameters λ and φ only depends of the camera and of its position on the robot (height from the ground and orientation). However, they can change (usually slightly) as the camera can be moved. Thus, it is important to be able to compute these parameters before using the vision model. This is what we call *calibration* of the visual system.

This is achieved using the image of a fixed-sized white cross, drawn on the floor. If the size of each line segment of the cross is l, and if these lines are projected on the video camera image as vectors of respective coordinates (x_A, y_A) and (x_B, y_B), we have:

$$\begin{cases} \lambda &= \frac{\sqrt{{y_A}^2+{y_B}^2}}{l} \\ \varphi &= \arcsin \frac{\sqrt{{x_A}^2+{x_B}^2}}{\sqrt{{y_A}^2+{y_B}^2}} \end{cases} \quad (5)$$

This calibration has not been implemented (it can be easily done), as the C++ interface for the Meteor frame grabber has not yet been debugged (because the DC/DC converter of the Nomad 200$^{\text{TM}}$ burnt).

2.3.4 Obstacle Detection

Once the previous steps will have been achieved, it will be possible to detect and to locate obstacles in front of the robot. This can be achieved in two ways.

The first way is simpler as it only uses the last image grabbed, but is of course less accurate. It consists in detecting edges (lines) in the image, and gathering these edges to find objects boundaries (this should eliminate isolated lines). If the lower boundary of each object is supposed to be on the ground level, the exact position of the object can be localized.

The second solution is more complicated, as it needs to correlate the points from two (or more) successive images. The vision system reacts then as a stereo-vision system, which allows us to compute the exact position of each point in the three-dimensional world. The coordinates in the real world of the end points of every line detected in the image are computed. The lines that are not in the ground plane ($z = 0$) are considered as obstacles' boundaries, and are gathered to build obstacles.

2.4 Achieving Vision-Guided Motion

Once the planning and control part (described in § 2.2) and the vision one (§ 2.3) will work, their integration will allow to achieve vision-guided motion for the Nomad 200$^{\text{TM}}$. In that case, it will be possible, after calibration, to give a goal (position or geometric configuration, or even dynamic configuration) to the robot and to have it reaching this goal while avoiding every obstacle which is in its way.

The formal algorithm would then be the following:

Vision-Guided Algorithm

Calibrate the vision system (§ 2.3.3)
Ask for the goal
while the goal is not reached **do**
    Detect and localize possible obstacles in front of the robot (§ 2.3.4)
    Add these obstacles to an obstacle database
    Plan a collision-free (w.r.t. the obstacle database) path from
        the current configuration to the goal

and follow this path (§ 2.2.5)

Let us recall that the high-level controller uses the infrared sensors and the tactile bumpers to verify that the robot avoids every obstacles (§ 2.2.5). In the same way, the sonar units (§ 2.1) can be used to detect and localize obstacles around the robot (first step of the loop in the previous algorithm).

Conclusion and Future Works

During these seven months in the "Vision and Control" Strategic Research Program, I achieved team works as well as research works. In the first case, I contribute to the collaboration between NTU and the French institute Inria, but also installed some tools to bring the power of Unix (or Linux) to the PC of the new "Autonomous Vehicle Laboratory"İf these works have been successful, their achievement took more time than it was first planned.

In the second case, unexpected problems and incidents prevented the research work to be fully implemented. However, major theoretical results have been obtained, and basic implementations have been conducted. Indeed, the planning and control problem has been formally stated, and existence and nature of optimal solutions to this problem have been studied. On another hand, a vision algorithm has been proposed, as well as an integration method. Implementations have not been totally fulfilled, due to technical (software malfunction) and practical (hardware failure) reasons.

The completion of this research work does not request as much qualification as it previously did: the low-level functions are already implemented (and commented to be easily used). Main directions for the future works are the following:

- repair the robot and understand the software malfunction, in order to achieve the planning and control part of the work; this work should be possible (at least partly) from the Inria Rhône-Alpes, where I will return after the end of my contract in NTU;
- implement the ideas presented in the vision part of the work; this should not be a problem as a lot of work has already been done in that domain in the "Vision and Control" Strategic Research Program;
- integrate the vision and control parts to conclude the work; once again, this is a very high-level implementation that should not be difficult.

Appendix

A Existence and Nature of Optimal Paths for our Robot

We proved that there exists a solution to the planning problem stated in § 2.2.2 if, and only if, the configurations to connect are in the same connected part of $\mathcal{C}^{\varepsilon}_{\text{free}}$. In this appendix, we will prove that, in that case, there also exists a time-optimal solution and we will show a few properties of this optimal solution.

First of all, let us recall the problem:

> Being given a starting configuration q_s, a goal configuration q_g, and the set of obstacles $\mathcal{B}_j$, $j \in \{1, ..., n_{\mathcal{B}}\}$, we search a path Π which:
>
> - connects q_s to q_g (Π is a curve in $\mathcal{C}$);
> - is feasible for $\mathcal{A}$, *i.e.* respects the equations (2) and (3);
> - is a curve in $\mathcal{C}^{\varepsilon}_{\text{free}}$, *i.e.* whose distance to the obstacles remains greater than ε.

The path Π can be considered as a mapping from time to configurations, *i.e.* as a function $t \mapsto q(t)$ from $[0, T]$ to $\mathcal{C}^{\varepsilon}_{\text{free}}$. To find the time optimal solution path, we want to optimize

$$I[\Pi, u] = g(0, q(0), T, q(T)) + \int_0^T f_0(t, q(t), u(t))dt,$$

where $g = 0$, $f_0 = 1$ and the functions q (the path) and u (its associated control function) verify:

- the differential system

$$\frac{\mathrm{d}q}{\mathrm{d}t} = f(t, q(t), u(t)) = (v(t)\cos\theta(t), v(t)\sin\theta(t), \omega(t), u_0(t), u_1(t))$$

- the limit conditions

$$(0, q(0), T, q(T)) \in B = \{0\} \times \{q_s\} \times [0, T_{\max}] \times \{q_g\}$$

- and the constraints

$$\begin{cases} (t, q(t)) & \in & A = [0, T_{\max}] \times \mathcal{C}^{\varepsilon}_{\text{free}} \\ u(t) & \in & U = [-a_{\max}, a_{\max}] \times [-\gamma_{\max}, \gamma_{\max}] \end{cases}$$

A.1 Existence of Optimal Paths

Before giving Filippov's theorem, which proves the existence of time-optimal solutions to our problem, let us define, for all (t, q) in A:

$$\begin{aligned} \widetilde{Q}(t,q) &= \left\{(z^0, z)/\exists u \in U, z^0 \geq f_0(t,q,u), z = f(t,q,u)\right\} \\ &= \left\{(z^0, v\cos\theta, v\sin\theta, \omega, u), z^0 \in [1, +\infty[, u \in U\right\} \end{aligned}$$

Theorem 1 (Filippov) *Let us assume that A and U are compact sets, B is a closed set, f_0 and f are continuous on $M = A \times U$, g is continuous on B and $\widetilde{Q}(t,q)$ is a convex set for all (t,q) in A.*
Then, if there exists solutions to the considered problem, $I[\Pi, u]$ reaches an absolute minimum on the set of solutions.

Corollary 1 *If q_s and q_g are in the same connected part of $\mathcal{C}^{\varepsilon}_{\text{free}}$, there exists a time optimal solution to the planning problem we consider.*

Proof: It is easy to verify that A and U are compact sets (let us recall that $\mathcal{C}^{\varepsilon}_{\text{free}}$ is a compact set), that B is a closed set, and that the functions f_0, f and g are continuous.

Moreover, $\widetilde{Q}(t,q)$ is a convex, as it is similar to $[-a_{\max}, a_{\max}]\times[-\gamma_{\max}, \gamma_{\max}]\times [1, +\infty[$ for all (t,q) in A. □

A.2 Nature of Optimal Paths

To show some properties of the optimal paths solution of our problem, we will use the Pontryagin Maximum Principle as stated by Cesari [1, Chap. 4]. This principle can only be applied to the paths contained in the inner part of $\mathcal{C}^{\varepsilon}_{\text{free}}$. However, if the workspace $\mathcal{W}$ is chosen wide enough, the pieces of the optimal paths that are on the boundaries of $\mathcal{C}^{\varepsilon}_{\text{free}}$ either follow the borders of an obstacle (at a distance ε) or correspond to a maximum (in absolute value) linear or angular velocity.

Nothing can be said in the first case (the nature of the path depends on the geometric shape of the obstacles). In the two other cases, we can prove that the accelerations are bang-bang (*i.e.* null or maximum in absolute value). When the linear velocity v is maximum (in absolute value), the optimal path is the one with the shortest length. The problem is then similar to the one considered in my Ph. D. thesis [5], and optimal paths correspond to a bang-bang angular acceleration γ with a null linear acceleration a. When the angular velocity ω is maximum (in absolute value), the possible paths have a fixed geometric shape, and the optimal one correspond to a bang-bang linear acceleration a (maximum acceleration, possibly maximum velocity, maximum deceleration) with a null angular acceleration γ.

Let us now consider the pieces of the optimal paths that are in the inner part of $\mathcal{C}^{\varepsilon}_{\text{free}}$. For those pieces, the necessary conditions of the Pontryagin Maximum Principle [1, Chap. 4, cond. (a)-(d)] are verified:

(a) the existence of an optimal path has been proved using Filippov's theorem (*cf.* § A.1);

(b) the curve $(t, q(t))$ corresponding to the concerned piece of optimal path remains in the inner part of A;

(c) U is a bounded and closed set of $\mathbb{R}^2$;

(d) B has a linear tangent variety B' whose vectors are $h = (0, 0, \tau, 0), \tau \in \mathbb{R}$ (the elements of B have constant coordinates, except for the third which remains in a interval).

When $\eta = (\eta_0, \eta_1, \eta_2, \eta_3, \eta_4, \eta_5) \in \mathbb{R}^6$, the Hamiltonian function is defined as:

$$\begin{aligned} H(t, q, u, \eta) &= \eta . \begin{pmatrix} f_0(t, q, u) \\ f(t, q, u) \end{pmatrix} \\ &= \eta_0 + \eta_1 v \cos\theta + \eta_2 v \sin\theta + \eta_3 \omega + \eta_4 a + \eta_5 \gamma \end{aligned}$$

where $a = u_0$ and $\gamma = u_1$ are the control parameters (*i.e.* the linear and angular accelerations). Let M be its minimum for $u \in U$:

$$M(t, q, \eta) = \min_{u \in U} H(t, q, u, \eta)$$

The Pontryagin Maximum Principle then implies that the optimal path $(q(t), u(t))$, for $t \in [0, T]$, verifies the necessary following conditions:

(P1) there exists a function η absolutely continuous on $[0, T]$, which is never zero on this interval, whose first coordinate η_0 is a positive constant and which verifies almost everywhere[3]:

$$\begin{aligned} \dot{\eta}(t) &= \begin{pmatrix} \dot{\eta}_0(t) &=& 0 \\ \dot{\eta}_1(t) &=& -\frac{\partial H}{\partial x}(t, q(t), u(t), \eta(t)) \\ \dot{\eta}_2(t) &=& -\frac{\partial H}{\partial y}(t, q(t), u(t), \eta(t)) \\ \dot{\eta}_3(t) &=& -\frac{\partial H}{\partial \theta}(t, q(t), u(t), \eta(t)) \\ \dot{\eta}_4(t) &=& -\frac{\partial H}{\partial v}(t, q(t), u(t), \eta(t)) \\ \dot{\eta}_5(t) &=& -\frac{\partial H}{\partial \omega}(t, q(t), u(t), \eta(t)) \end{pmatrix} \\ &= \begin{pmatrix} \dot{\eta}_0(t) &=& 0 \\ \dot{\eta}_1(t) &=& 0 \\ \dot{\eta}_2(t) &=& 0 \\ \dot{\eta}_3(t) &=& \eta_1 v \sin\theta - \eta_2 v \cos\theta \\ \dot{\eta}_4(t) &=& -\eta_1 \cos\theta - \eta_2 \sin\theta \\ \dot{\eta}_5(t) &=& -\eta_3 \end{pmatrix} \end{aligned}$$

[3] A condition is verified *almost everywhere* if, and only if, it is verified except in a finite number of case.

(P2) the minimum, for $u \in U$, of $H(t, q(t), u, \eta(t))$ is (almost everywhere) obtained for $u = u(t)$:

$$M(t, q(t), \eta(t)) = H(t, q(t), u(t), \eta(t))$$

(P3) the function $M(t) = M(t, q(t), \eta(t))$ is absolutely continuous (or, in fact, is almost everywhere equal to a fixed absolutely continuous function) on $[0, T]$, and its derivative is (almost everywhere):

$$\dot{M}(t) = \frac{\partial H}{\partial t}(t, q(t), u(t), \eta(t)) = 0$$

(P4) as g is constant and the vectors of B' are $h = (0, 0, \tau, 0), \tau \in \mathbb{R}$, the transversality relation is as simple as $M(T) = 0$.

These necessary conditions are given for the vectorial function η, which is adjoint to the path and control functions (resp. q and u), or for the Hamiltonian function H. The implications of these conditions w.r.t. the nature of the optimal paths still need to be pinpointed.

First of all, condition (P1) implies that the functions $\eta_1(t)$ and $\eta_2(t)$ are constant (they are continuous and their derivative is zero almost everywhere). We can then write:

$$\exists!\ (\mu, \psi) \in \mathbb{R}^+ \times \mathcal{S} \ / \ \forall t \in [0, T], \left\{ \begin{array}{rcl} \eta_1(t) & = & \mu \cos \psi \\ \eta_2(t) & = & \mu \sin \psi \end{array} \right. \tag{6}$$

The differential equation of condition (P1) can now be rewritten as:

$$\dot{\eta}(t) \quad = \quad \left(\begin{array}{rcl} \dot{\eta}_0(t) & = & 0 \\ \dot{\eta}_1(t) & = & 0 \\ \dot{\eta}_2(t) & = & 0 \\ \dot{\eta}_3(t) & = & \mu v \sin(\theta - \psi) \\ \dot{\eta}_4(t) & = & -\mu \cos(\theta - \psi) \\ \dot{\eta}_5(t) & = & -\eta_3 \end{array} \right)$$

On another hand, let us note that the function $M(t)$ defined in condition (P3) is continuous by definition, as the function η is itself continuous according to condition (P1). Condition (P3) also indicates that the derivative of the function $M(t)$ is zero almost everywhere, which implies that this function is constant (as it is continuous). Then, condition (P4) implies that this function is the zero function:

$$\forall t \in [0, T], M(t) = 0$$

Moreover, this function can be rewritten, using equation (6) as:

$$M(t) = \eta_0 + \mu v(t) \cos(\theta(t) - \psi) + \eta_3(t)\omega(t) + \eta_4(t)a(t) + \eta_5(t)\gamma(t)$$

Then, condition (P2) implies that, almost everywhere:

$$\eta_4(t)a(t) + \eta_5(t)\gamma(t) = \min_{u \in U} \left(\eta_4(t)u_0 + \eta_5(t)u_1\right)$$

or, as the two coordinates of the control vector u are independent:

$$\begin{cases} \eta_4(t)a(t) &= -|\eta_4(t)|\, a_{\max} \\ \eta_4(t)\gamma(t) &= -|\eta_4(t)|\, \gamma_{\max} \end{cases} \tag{7}$$

This induces that, almost everywhere, either the linear acceleration $a(t)$ is maximum (in absolute value) or $\eta_4(t)$ equals zero, and that either the angular acceleration $\gamma(t)$ is maximum or $\eta_5(t)$ equals zero.

If $\eta_4(t)$ (resp. $\eta_5(t)$) equals zero but η_4 (resp. η_5) is not null in a neighborhood of t, then $a(t)$ (resp. $\gamma(t)$) can be chosen maximum (it does not change the resulting path). The only problem may occur when η_4 (or $\eta_5(t)$) is null on an interval.

Let us first suppose that η_4 is null on an interval I, which is equivalent to say that a is not maximum on that interval. Then, this function derivative is null, and thus $\dot{\eta}_4(t) = -\mu\cos(\theta(t) - \psi) = 0$. It means that either μ equals zero or θ is constant on I. This second case is impossible: it would imply that both ω and γ are null on I, and $M(t)$ would then be equal to η_0 which is a positive constant according to (P1), but we just proved that M is the zero function. As a consequence, only $\mu = 0$ is possible and the function η_3 is constant ($\dot{\eta}_3(t) = 0$). This constant cannot be zero, otherwise we would again have $M(t) = \eta_0 = 0$, which is impossible due to (P1). In this case, $\eta_5(t)$ is not constant on I and γ is maximum (in absolute value) on I.

We just proved that, if a is not maximum on an interval, γ is maximum (in absolute value) on it. The symmetric demonstration can be done in a very similar way. We can thus conclude that, almost everywhere in $[0, T]$:

$$(a(t) = \pm a_{\max}) \textbf{ or } (\gamma(t) = \pm a_{\max})$$

Bibliographic References

[1] L. Cesari. *Optimization–theory and applications*, volume 17 of *Application of Mathematics*. Springer-Verlag, 1983.

[2] S. Fleury, Ph. Soures, J.-P. Laumond, and R. Chatila. Primitives for smoothing paths of mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics and Automation*, volume 1, pages 832–839, Atlanta GA (US), May 1993.

[3] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. A stable tracking control method for a non-holonomic mobile robot. In *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, volume 3, pages 1236–1241, Osaka (JP), November 1991.

[4] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The mathematical theory of optimal processes*. Interscience publishers, 1962.

[5] A. Scheuer. *Planification de chemins courbure continue pour robot mobile non-holonome*. Thèse de doctorat, Inst. Nat. Polytechnique de Grenoble, Grenoble (FR), January 1998.