Université de Liège
Faculté des Sciences Appliquées
Département d'Électricité, Électronique et Informatique

Efficient and Precise Trajectory Planning for Nonholonomic Mobile Robots

Stéphane Lens

Thèse présentée en vue de l'obtention du grade de
Docteur en Sciences de l'Ingénieur

Juillet 2015
(Dernière mise à jour: 31 août 2015)

Abstract

Trajectory planning is one of the fundamental problems in mobile robotics. A wide variety of approaches have been proposed over the years to deal with the various issues of this problem.

This thesis presents an original and complete solution to tackle the motion planning problem for nonholonomic mobile robots in two-dimensional space. Given a set of obstacles, an initial and a goal configuration, the problem consists in computing efficiently a physically feasible trajectory that reaches the specified target as fast as possible.

One of the original aspects of this work lies in the decomposition of the general problem into several simpler subproblems, for which very efficient solutions are developed. Their combination provides a complete trajectory planning approach that is one of the most computationally effective method suited for the motion of cylindrically shaped wheeled mobile robots in the presence of polygonal obstacles.

This complete solution consists of three main steps. The first one is aimed at finding a short path that avoids obstacles and manages to reach the destination, without taking into account nonholonomic constraints of the robot. Our path planning method relies on an original refinement procedure of a constrained Delaunay triangulation of the obstacles, that outperforms other existing planning techniques.

The second step consists in interpolating paths into smooth curves that can be followed by a real robot without slowing down excessively. By joining only two arcs of clothoids for moving from one curvature to another, our approach is simpler and also computationally cheaper than other interpolation methods.

Finally, thanks to the introduction of an original discretization scheme, an efficient algorithm for computing a time-optimal speed profile for arbitrary paths is presented. The speed profile that results from this procedure not only allows the robot to follow the synthesized path as fast as possible while taking into account a broad class of velocity and accelerations constraints, but also provides the accurate advance information necessary to implementing coordinated actions during the displacement of the robot (e.g., between the locomotion system and other actuators).

Résumé

La planification de trajectoires est un des problèmes fondamentaux de la robotique mobile. Ces dernières années, une grande variété de solutions ont donc été proposées afin de répondre au mieux aux différents aspects de ce problème.

Cette thèse s'inscrit dans cette démarche et présente une solution complète et originale pour résoudre le problème de planification de mouvements pour des robots mobiles non-holonomes se déplaçant dans un espace à deux dimensions. Étant donné un robot, un ensemble d'obstacles, et une configuration de départ et d'arrivée, le problème consiste à calculer une trajectoire qui est non seulement réalisable par ce robot, mais qui de plus lui permet d'atteindre sa destination le plus rapidement possible.

Une des originalités de la méthode présentée dans cette thèse réside dans la décomposition du problème en plusieurs sous-problèmes, pour lesquels, des solutions très efficaces sont développées. La combinaison de ces différentes solutions permet de répondre au problème posé et forme, à notre connaissance, une des méthodes les plus efficace dédiée à la planification de mouvements pour des robots cylindriques à roues se déplaçant parmi des obstacles polygonaux.

Cette solution complète comporte trois grande étapes. La première consiste à trouver un chemin court qui évite les obstacles et qui permet d'atteindre la destination, sans tenir compte des contraintes de non-holonomie du robot. Notre méthode repose sur une procédure originale de raffinement d'une triangulation de Delaunay contrainte, qui surpasse la plupart des autres méthodes existantes.

La deuxième étape consiste en une interpolation des chemins en des courbes lisses pouvant être suivies plus rapidement par un robot car demandant de moins ralentir lors des changements d'orientation. En utilisant seulement des paires de clothoïdes pour passer d'une courbure à une autre, notre méthode est plus simple et plus rapide que d'autres méthodes d'interpolation.

Finalement, grâce à l'introduction d'une procédure de discrétisation originale, une méthode efficace de calcul d'un profil de vitesse optimal en temps est présentée. Celui-ci permet au robot de suivre le chemin calculé le plus rapidement possible tout en tenant compte d'une large gamme de contraintes de vitesse et d'accélération. Il fourni aussi les informations sur les configurations futures du robot qui sont nécessaires à l'élaboration d'actions coordonnées (e.g., entre le système de locomotion et d'autres actuateurs).

Acknowledgements

I wish to express my personal gratitude to all of those who directly or indirectly contributed to the realization of this work.

First and foremost, I would like to sincerely thank my advisor, Bernard Boigelot, for the confidence he has placed in me, his encouragements, and his constant and passionate investment in our work. I am also grateful to him for the Eurobot project we started at the Montefiore Institute several years ago. This project has been a great opportunity to confront theory to practice, but also to improve my skills in various fields, like digital and power electronics, mechanics, or even management.

I would also like to thank all the other people who have been involved in this project, not only for their incredible work, but also for all the long nights that we have spent in the lab and all the great moments that we have shared together in (sometimes unexpected) foreign countries. I think, in particular, about Guy Lejeune, Etienne Michel and Vincent Pierlot who have participated to this project during numerous years, but also about Laurent Poirrier, Maxime Urbin-Choffray, Sébastien Pierard and Matthieu Remacle.

I am warmly thankful to Pascal Gribomont, who gave me the opportunity to work at the University. He always showed an interest in my work, even when it drifted away from his field of expertise. I am also very grateful to him for the confidence he has always placed in me, allowing me to assist him in several teaching duties.

I would also like to express my genuine appreciation to all the members of the jury, for devoting time and interest to the reading and evaluation of this manuscript.

I thank all my colleagues at the Montefiore Institute for the positive working atmosphere. Special thanks to Matthieu Philippe. We have come through our last year of thesis together; it was long, but we did it!

I also acknowledge all the people who never missed our traditional and enjoyable "Boulets du Vendredi" lunch times. I think in particular about Denis de Marneffe, Guy Lejeune and Etienne Michel, but also about Gauthier Becker, Sébastien Hoffay, Christophe Rutten and Matthieu Philippe.

The “Kfet du Vendredi”, what a great way to end the week! Many thanks go to Denis de Marneffe, François Reip, Régis Pirard, Simon François and to everyone else for all the good times and also all the drinks we have shared.

Finally, I thank warmly all my family and friends, for their incredible support during all these years. I address all my affection to my parents, and to my (not so) little brother Cédric.

Last but not least, I cannot conclude without thanking Anne, who, in spite of her own challenges, never stopped supporting and encouraging me throughout this work. With all my heart, thank you!

Contents

Notations

The following notations are used throughout the thesis.

Sets

$\emptyset$	The empty set.
$\mathbb{Z}$	The set of all integers.
$\mathbb{N}$	The set of all natural numbers.
$\mathbb{N}_{>0}$	The set of all strictly positive natural numbers.
$\mathbb{R}$	The set of all real numbers.
$\mathbb{R}_{\geq 0}$	The set of all non-negative real numbers.
$\mathbb{R}_{>0}$	The set of all strictly positive real numbers.
$\mathbb{R}_{<0}$	The set of all strictly negative real numbers.
$\mathbb{R}^2$	The set of all ordered pairs of real numbers.
$\mathbb{R}^3$	The set of all ordered triples of real numbers.
$\mathbb{R}^n$	The set of all ordered n-tuples of real numbers.
$\{A_1, A_2, \ldots, A_p\}$	The set of all the elements A_1 to A_p.
$A \times B$	The Cartesian product of two sets, $A \times B = \{(a,b) \mid a \in A,\ b \in B\}$.
$[a,b]$	The real interval $\{x \mid a \leq x \leq b\}$.
$(a,b]$	The real interval $\{x \mid a < x \leq b\}$.
$[a,b)$	The real interval $\{x \mid a \leq x < b\}$.
(a,b)	The real interval $\{x \mid a < x < b\}$.

Manifolds

$\mathcal{M}$	An arbitrary manifold.
$\mathbb{S}^1$	The unit sphere in $\mathbb{R}^2$ (the unit circle), the set of unit norm vectors in $\mathbb{R}^2$.
$\mathrm{SO}(3)$	The special orthogonal group in $\mathbb{R}^3$, the set of 3×3 rotation matrices.
$\mathrm{SE}(3)$	The special Euclidean group of rigid body displacements in three-dimensions.

Computational complexity

$O(g(n))$	Big-O notation, $f(n) = O(g(n))$ means $\exists c, k \in \mathbb{N}_{>0} : n \geq k \Rightarrow 0 \leq f(n) \leq cg(n)$.

Geometry

$[\alpha]$	The angle belonging to $(-\pi, \pi]$ and such that $[\alpha] = k\alpha$ with $k \in \mathbb{Z}$.
$\arctan(\alpha,\ \beta)$	The principal value of the arctangent of $\frac{\alpha}{\beta}$ (handles $\beta = 0$, cf. Eq. 3.1).
$[P_1P_2]$	The line segment that joins P_1 to P_2.
$\lvert P_1P_2 \rvert$	The length of $[P_1P_2]$.
$\lvert P_1 - P_2 \rvert$	The distance between P_1 and P_2.
$\widehat{ABC}$	The angle between the two segments $[AB]$ and $[BC]$.
ABC	The triangle ABC.

Logical connectives

$\neg$	Negation ($\neg A$ is true iff A is false).
$\wedge$	Logical conjunction ($A \wedge B$ is true iff A and B are both true).
$\vee$	Logical disjunction ($A \vee B$ is false iff A and B are both false).
$\Rightarrow$	Material implication ($A \Rightarrow B$ is false iff A is true and B is false).
$\forall$	Universal quantification ($\forall x : P(x)$ means $P(x)$ is true for all x).
$\exists$	Existential quantification ($\exists x : P(x)$ means $P(x)$ is true for at least one x).
$\nexists$	Negation of the existential quantification.

Mobile robots

$(x,\ y)$	Coordinates of the robot.
θ	Orientation of the robot's chassis.
θ_S	Steering angle.
$e,\ e'$	Axle width and wheelbase of the robot.
$\dot\psi_L,\ \dot\psi_R,\ \dot\psi_S$	Angular speeds of the left, right, and steering wheels.
$v_L,\ v_R,\ v_S$	Speeds of the left, right, and steering wheels.
v	Longitudinal speed of the robot.
ω	Instantaneous speed of rotation of the robot.
$a_L,\ a_R,\ a_S$	Tangential accelerations of the left, right and steering wheel.
$a_C,\ a_T$	Tangential and radial accelerations of the robot.

Miscellaneous

κ	Instantaneous curvature.
C	Configuration space.
C_{obs}	The obstacle region.
C_{free}	The free space (the set of configurations that do not conflict with obstacles).

CHAPTER 1

Introduction

"Setting a goal is not the main thing.
It is deciding how you will go about achieving it
and staying with that plan."

— Tom Landry

Robots are everywhere!

Industrial manipulator arms, reconnaissance drones, robot vacuum cleaners, space rovers, surgery robots, or more recently, self-driving cars are becoming increasingly common. In the not-too-distant future, we can even expect to see humanoid robots assisting humans in responding to natural or man-made disasters, based, for example, on the recent achievements of the DRC[1]. With all those robots around, one of the most important challenges that we are facing, is how to make them as autonomous as possible, so they can safely perform their tasks without further human intervention.

Most interactions between a robot and its environment are carried out by moving actuators. Except in a limited and fully controlled environment (as it can sometimes be the case, for example, in assembly plants), it is very difficult and unreasonable to describe explicitly all of these possible motions. Even simple daily tasks performed by a human being, like moving into a house without colliding with furniture, or simply grabbing an object, can

[1] DARPA Robotics Challenge (http://www.theroboticschallenge.org/)

become quite complex for an autonomous robot. Indeed, the human brain is well suited for carrying to carry out such actions, but programming a robot to perform them amounts to solving many problems: detecting and reacting to obstacles, computing sequences of actions to be carried out, interacting with the environment, ...

Motion planning plays a key role in robotics, and in other domains of engineering that rely on it, allowing robots to automatically decide which move to make in order to reach a goal while taking into account environmental constraints. This problem is even more crucial for mobile robots that can move in relatively large areas in the presence of dynamic obstacles.

1.1 Motivation

There are lots of different types of robots: some have wheels, tracks or legs, some can fly or dive underwater, some are articulated, and all of them have to deal with their physical constraints (such as limited speed or acceleration of their actuators). Motion planning, in general, is therefore a complex problem.

Even if it may look simple, the *generalized mover's problem*, which is to decide whether a robot, consisting of a set of polyhedra freely linked together at certain vertices, can move without collisions from an initial to a final configuration in an environment constrained by fixed polyhedral obstacles, is PSPACE-hard [Rei79, Can88].

In this thesis, we narrow our scope to motion planning algorithms for autonomous mobile robots, and more particularly, for nonholonomic[2] wheeled ones. This restriction leads to simplifications with regard to the more general problem and, hopefully, to adequate solutions. Moreover, with the advent of unmanned vehicles, the scope is still very large and considerable room remains for improvement.

For many applications, the ability of the robot to react quickly to any situation is crucial, and a strong emphasis must be placed on the efficiency of motion planning, which often has to be carried out with the limited amount of computing power available on the robot. Our objective will then not to find an optimal solution to the planning problem, but to develop a method that can very quickly synthesize a trajectory for reaching the target in acceptable time, and that is consistent with the physical limitations of the robot (e.g., sufficient clearance is ensured with respect to obstacles, speed and acceleration bounds are respected at all times).

[2] A robot is nonholonomic when it has constraints on its velocity that are not derivable from position constraints; a car, for example, is nonholonomic, since it can move only in the direction imposed by its steering wheels.

This work has primarily been motivated by the Eurobot[3] contest, in which small autonomous mobile robots have to compete against opponents in a $2 \times 3\,\mathrm{m}^2$ area strewn with obstacles. In this contest, as in other robotics competitions like RoboCup Soccer[4], it is essential to plan paths in real-time due to the dynamic nature of obstacles, which requires a method with very low computational cost, and synthesizing paths that can be followed without slowing down in the vicinity of obstacles can provide a definite competitive advantage. In addition, obtaining accurate advance information about the locations that will be visited by the robot and their associated timestamps is essential to implementing coordinated actions between several robots, or between the locomotion system and other actuators (e.g., picking up objects while moving).

As it can be expected given its importance, the motion planning problem for mobile robots in two-dimensional space, has already been widely studied. (See, e.g., the survey paper by Schwartz and Sharir [SS88] and the classical books by Latombe [Lat91] and by Laumond [Lau98] for the basic concepts related to robot motion planning. Other more recent books like the ones by Choset et al [CLH+05], by LaValle [LaV06] and by Siciliano et al [SSVO09] cover most of the latest methods in the field.) In spite of all this, none of the existing methods, to the best of our knowledge, was able to fulfill directly and in a satisfactory manner all our requirements for Eurobot.

Several methods, such as cell decomposition [BLP85], roadmap [BG08], rapidly exploring random trees [LaV98, KWP+11], and potential field techniques [GC02], generally produce paths that are expressed as sequences of straight line segments (with sometimes the addition of circle arcs), between the initial and final configurations. Those methods are usually efficient, but do not take into account velocity or acceleration constraints of the actuators. This results in paths that cannot be followed by a mobile robot without stopping at the junction points between adjacent segments in order to change its orientation, which is inefficient.

Other methods, such as curvature-constrained planning (e.g., [Dub57, JC93, RW98, BK08]), tackle this problem and produce smooth paths, but are more complex, and less efficient in terms of computation time. This is mainly due to the fact that these methods merely try to generate optimal trajectories (shortest, fastest, ...), which is an inherently difficult problem[5].

One general idea in order to improve efficiency is to divide the motion problem into subproblems, first computing a collision-free trajectory without taking into account nonholonomic constraints, and then transforming it to compute the desired trajectory. This

[3] http://www.eurobot.org

[4] http://wiki.robocup.org/wiki/Middle_Size_League

[5] The curvature-constrained shortest-path problem is known to be NP-hard, when the obstacles are polygons with a total of N vertices and the vertex positions are given within $O(N^2)$ bits of precision [RW98].

approach is also followed in, e.g., [LJTM94, LL98], but, since they also try to meet optimality criteria, these methods are still often complex and time-consuming.

We can also mention, among others, reactive approaches (e.g., [Kha86, FBT97]) or methods suited for autonomous car navigation prompted by the DARPA Grand and Urban Challenge[6] (e.g., [TMD⁺06, CE07, UAB⁺08, KKT⁺09]). Those methods are usually efficient, but either have a reactive nature that prevents planning coordinated actions, or cannot provide a guarantee of finding a feasible path if one exists. These issues rarely happen for regular on-road planning, which is the primary use of these techniques, but this can be different for complex situations or other environments.

In order to be competitive against various opponents in the context of a robotics competition, the ability to quickly plan actions is decisive, and since we also have to implement our solutions on low-cost embedded processors, the computation time of the complete motion planning method then becomes a priority. Generating trajectories to reach targets in the shortest possible time is also important, but, because of environmental uncertainties (drift, sensor errors, . . .), obtaining global optimality matters less.

This thesis concentrates on the design of a complete workflow that can be and has been used on real mobile robots, like those participating to the Eurobot contest or the RoboCup Soccer Middle Size League, assembling existing efficient solutions when possible, and optimizing or developing others when necessary, while keeping in mind our important concern for efficiency.

We can summarize the different requirements for our motion planning problem as follows:

- generate trajectories that avoid a given set of obstacles with sufficient clearance,
- generate trajectories consistent with the physical constraints of the robot,
- generate trajectories that allow to reach the target as fast as possible,
- limit as much as possible the computation time, which should remain negligible with respect to time constants of actuators,
- deal efficiently with a changing environment (avoid the need to pre-compute trajectories, . . .),
- provide accurate advance information about trajectories, such as future locations with associated timestamps (to make it possible to plan coordinated actions),
- remain general when possible (with respect to different kind of wheeled robots, shape of obstacles, . . .).

[6] http://archive.darpa.mil/grandchallenge/

1.2 Contributions

This thesis presents an original and complete method to tackle the motion planning problem for a nonholonomic mobile robot in two-dimensional space. Given a set of polygonal obstacles, an initial and a target configuration, the problem consists in computing efficiently a fast and physically feasible trajectory that reaches the specified destination.

Our approach consists in dividing the problem into three main subproblems. The first one is aimed at finding a path that avoids obstacles and manages to reach the target. Such a path takes the form of a sequence of straight line segments that clears the obstacles at a specified safety distance. The second step is to smooth this path, in order to obtain a path that can be travelled faster than the broken line. The third step is to compute a time-optimal speed profile for this path, taking into account the physical constraints of the robot.

In addition to this general workflow, we can highlight other major contributions:

- An original data structure for representing the spatial properties of any path that can be followed by a nonholonomic mobile robot (Section 3.2), that allows efficient and accurate interpolation, as well as resampling (Section 3.2.6).
- An algorithm for computing an optimal speed profile[7] for such represented paths, while taking into account a broad class of physical constraints of the robot, e.g., bounds on the admissible velocity or acceleration of its wheels or its center of mass, the aim being to minimize the total time needed for following the path (Chapter 4).
- A new algorithm for efficiently interpolating a broken line, i.e., a sequence of connected straight line segments, into a smooth curve, while remaining at a safe distance from obstacles (Chapter 5). The goal is to obtain a path that can be travelled much faster than the original broken line.
- An original approach to efficiently generating paths with arbitrary clearance to polygonal obstacles (Chapter 7).

Some of these contributions have been submitted for publication [LB15a, LB15b].

The results presented in this thesis have been successfully implemented in the robots built for Eurobot at the University of Liège. In this framework, the very low computational cost of this approach made it possible to perform near-optimal trajectory planning in real-time, which provided a significant advantage over other methods.

[7] A speed profile associates each point of the path with a timestamp that provides the instant at which it will be visited by the robot.

1.3 Outline of the Thesis

Chapter 2 introduces notions that are useful for the rest of the manuscript. It first describes locomotion models of different wheeled mobile robots and then describes basic concepts related to motion planning. It also gives some details about the Eurobot contest framework.

Chapter 3 discusses spatial and temporal properties of paths and trajectories for mobile robots, and presents an original data structure for discretely representing trajectories that can be followed by nonholonomic wheeled mobile robots. It also discusses different properties of this structure, such as efficient arbitrary resampling.

Chapter 4 introduces the notion of speed profile for the path representation described in Chapter 3. It then presents an algorithm for computing a time-optimal speed profile that minimizes the total time needed by a robot for following such path representations while being consistent with the physical constraints of the robot. This algorithm has a linear computational cost (in the number of path steps).

Chapter 5 presents an algorithm for interpolating a collision-free path expressed as a sequence of straight line segments into a smooth path that can be precisely followed by a nonholonomic robot without slowing down excessively. The aim is to reduce the time needed for the robot to reach the destination, while still avoiding obstacles.

Chapter 6 describes a method for computing a sequence of line segments that leads from an initial configuration of the robot to a target location, while avoiding some set of point obstacles with a given clearance. This path can be used as an input to the algorithm described in Chapter 5.

Chapter 7 presents an original generalization of the method described in Chapter 6 for sets of obstacles that include line segments in addition to individual points, which is useful for efficiently delineating polygonal obstacles without prior discretization, and speeds up the procedure.

Chapter 8 first describes some of the experiments performed in order to validate the different algorithms introduced in previous chapters. It considers case studies from the Eurobot contest, but also discusses performance for larger problems. Then it summarizes the different results and discusses the main contributions of this thesis. Finally, it gives a non-exhaustive list of improvements or perspectives than this work has opened.

"Begin at the beginning," the King said gravely,
"and go on till you come to the end: then stop."

— Lewis Carroll, *Alice in Wonderland*

CHAPTER 2

Mobile Robots

This chapter introduces some notions that are useful for the rest of this manuscript. First, it presents the locomotion models of the most common wheeled mobiles platforms, and discusses and compares their structural and dynamical properties. According to this discussion, it shows that most of these platforms have restricted mobility due to nonholonomic constraints resulting from the pure rolling condition between the wheels and the ground. Then, it introduces some basic concepts related to the environment. It shows, for example, that odometry, using wheel sensors, can be used as a first step to estimate the position of a wheeled robot in its workspace. It also addresses the concepts of configuration space, collision-free path, path clearance or admissible trajectory. Finally, it introduces the Eurobot contest framework, which has first motivated this thesis and wherein its results have been successfully implemented.

2.1 Locomotion Models

Ground locomotion of man-made vehicles usually relies on wheels, and this is also the case for mobile robots. This design is much simpler than using tracks or legs and is easier to build, program, and control in relatively flat environments. Wheeled robots, however, have difficulties navigating in rocky or slippery terrain, which is not our concern in the scope of this thesis. There are several classes of wheeled robots, determined mainly by the number and the position of wheels used. This section presents common wheeled mobile robot platforms and discusses their properties in a flat environment.

2.1.1 Differential Drive

A differential drive robot, as illustrated for example in Figure 2.1, is one of the most popular wheeled robot design and is constituted by two fixed, identical, and parallel driving wheels (W_L and W_R), each of which is controlled by an independent motor. Additional passive support elements (such as ball casters, depicted as double circles in the figure) ensure the stability of the platform. They are often mounted on springs, such that at any time, only one of them pushes the ground with non negligible force. In other words, the weight of the robot is always distributed over the locomotion wheels and one of the casters.

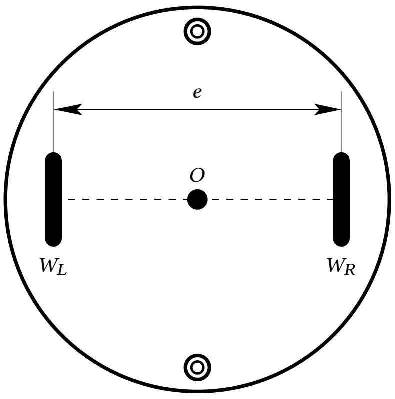


Figure 2.1 – Differential drive platform.

This choice of locomotion platform constrains the possible movements of the robot. Assuming perfect friction, as we will throughout this thesis, the kinematic constraints imply that each wheel is only free to move along its current orientation, and that the velocity of its center is parallel to the plane of the ground (nonslip condition) and is proportional to its angular speed (pure rolling condition). These constraints also imply that each wheel can only be steered around its contact patch with the ground, which we assume to be a single point. This means that each wheel is constantly tangent to the trajectory it follows or, equivalently, that the instantaneous center of rotation of such a trajectory is always located on that wheel's axle (i.e., the line orthogonal to the plane of the wheel that passes through its center).

As a consequence, the trajectory followed by the robot is entirely characterized by the angular speeds $\dot{\psi}_L$ and $\dot{\psi}_R$ of its two wheels (assuming that the geometry of the robot is known). For simplicity sake, we assume that the directions of rotation of the wheels are measured in such a way that a positive value of $\dot{\psi}_L$ or $\dot{\psi}_R$ moves the robot forwards. When both wheels rotate at constant speed, the robot either follows a straight line (if $\dot{\psi}_L = \dot{\psi}_R$) or a circular arc (if $\dot{\psi}_L \neq \dot{\psi}_R$). In the latter case, the center of the circular arc is located on the common axle of the wheels. The situation is illustrated in Figure 2.2, where e denotes the axle width (i.e., the distance between the wheel contact patches).

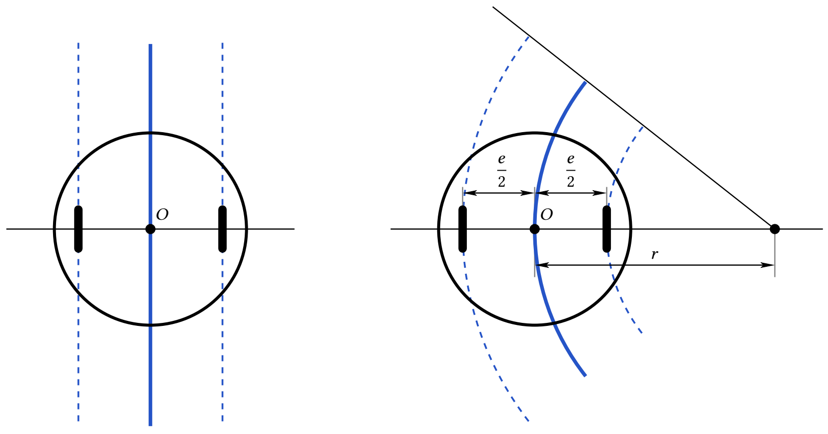


Figure 2.2 – Trajectories at constant wheel speed.

Since the locomotion platform is assumed to be rigid, the trajectories followed by the robot can be described by the movement of a single reference point O, a natural choice for which is the midpoint of the two driving wheels.

Let r_W denotes the (common) radius of the wheels. If both wheels rotate at a constant angular speed $\dot{\psi}_L = \dot{\psi}_R = \dot{\psi}$, then O follows a straight line parallel to the wheels, with the speed $v = r_W\dot{\psi}$. (A negative speed means that the robot moves backwards.)

If, on the other hand, the wheels rotate at different angular speeds $\dot{\psi}_L$ and $\dot{\psi}_R$, then the velocities of the left and right wheels are respectively given by

$$v_L = r_W\,\dot{\psi}_L, \tag{2.1}$$
$$v_R = r_W\,\dot{\psi}_R. \tag{2.2}$$

The radius r of the circle followed by the reference point O satisfies

$$v_R\left(r - \frac{e}{2}\right) = v_L\left(r + \frac{e}{2}\right), \tag{2.3}$$

where a positive (resp. negative) value of r means that the center of the circle is located on the left (resp. right) of O. This yields

$$r = \frac{e}{2}\cdot\frac{v_R + v_L}{v_R - v_L} = \frac{e}{2}\cdot\frac{\dot{\psi}_R + \dot{\psi}_L}{\dot{\psi}_R - \dot{\psi}_L}. \tag{2.4}$$

The longitudinal velocity and the instantaneous velocity of rotation of the robot are then directly related to the angular speed of the two driving wheels.

For the longitudinal velocity v of the robot, we have

$$v = \frac{v_R + v_L}{2} = \frac{r_W}{2}\left(\dot{\psi}_R + \dot{\psi}_L\right), \tag{2.5}$$

and for the instantaneous velocity of rotation ω of the robot, we have

$$\omega = \frac{v_R - v_L}{e} = \frac{r_W}{e}\left(\dot{\psi}_R - \dot{\psi}_L\right). \tag{2.6}$$

2.1.2 Tricycle Robot

A tricycle robot is a locomotion platform that is a bit more complex. It is composed of two wheels W_L and W_R that are parallel, fixed, and free (in the sense that they are not driven), augmented with a third wheel W_S that is driven and steered. This platform is illustrated in Figure 2.3. It has the advantage of providing a larger support polygon than the differential drive configuration. Furthermore, it allows to mount odometry feedback sensors (see Section 2.2.1) on the two free wheels, which are less likely to skid than a driven wheel.

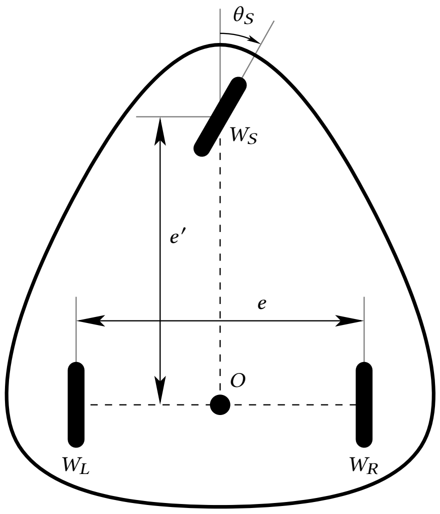


Figure 2.3 – Tricycle platform.

In this model, the trajectory followed by the robot is entirely characterized by the angular speed $\dot{\psi}_S$ of the steering wheel, as well as by the steering angle θ_S (assuming that the geometry of the robot is known). Like in Section 2.1.1, we consider that a positive value of $\dot{\psi}_S$ corresponds to the robot moving forwards. Moreover, as depicted in Figure 2.3, the steering angle is defined such that $\theta_S = 0$ corresponds to a straight trajectory, and $\theta_S > 0$ (resp. $\theta_S < 0$) to a left (resp. right) turn. It can be assumed w.l.o.g. that the steering angle always satisfies $-\frac{\pi}{2} \leq \theta_S \leq \frac{\pi}{2}$.

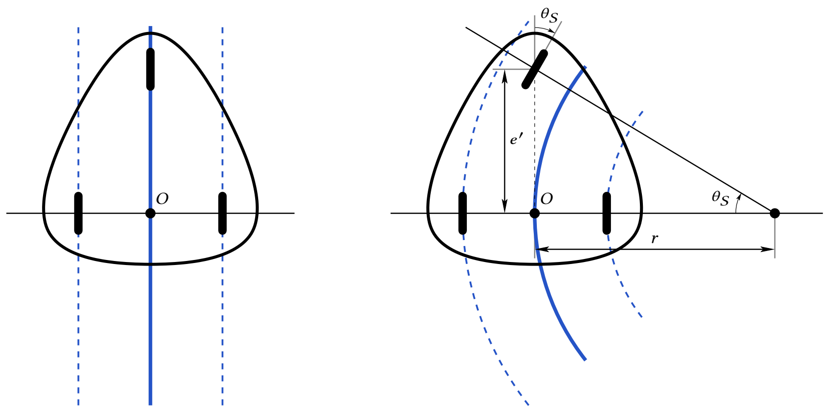


Figure 2.4 – Trajectories at constant wheel speed.

If the steering angle θ_S is constant, then the path followed by the robot is either a straight line if $\theta_S = 0$, or a circular arc if $\theta_S \neq 0$. The center of this circular arc is located at the intersection of the common axle of the free wheels and the axle of the steering wheel. This is illustrated in Figure 2.4, where e' denotes the distance between the contact patch of the steering wheel and the rear axle, also known as the *wheelbase* of the robot.

The trajectories followed by the robot can also be described by the movement of a single reference point O, assuming, as in the case of a differential drive platform, that the locomotion platform is rigid. A natural choice is to place the reference point O of the robot at the midpoint of the free wheels axle.

Let r_W denotes the radius of the steering wheel. Assume that the steering wheel rotates at a constant speed $\dot{\psi}_S$. The speed of the steering wheel is given by

$$v_S = r_W \, \dot{\psi}_S, \tag{2.7}$$

If $\theta_S = 0$, then O follows a straight line parallel to the free wheels, with the velocity $v = r_W \dot{\psi}_S$ identical to the speed v_S of the steering wheel. (A negative speed means that the robot moves backwards.)

If $\theta_S \neq 0$, then the radius r of the circle followed by O is given by

$$r = e' \cot \theta_S. \tag{2.8}$$

As in Section 2.1.1, a positive (resp. negative) value of r means that the center of the circle is located on the left (resp. right) of O.

The longitudinal velocity of the robot becomes

$$v = v_S \cos\theta_S = r_W\, \dot{\psi}_S \cos\theta_S, \tag{2.9}$$

and the instantaneous velocity of rotation ω of the robot becomes

$$\omega = \frac{v_S}{e'} \sin\theta_S = \frac{r_W}{e'}\, \dot{\psi}_S \sin\theta_S. \tag{2.10}$$

2.1.3 Car-Like Robot

A popular choice for a four-wheel robot is the car-like platform. This platform is illustrated in Figure 2.5 and is composed of two rear wheels (W_{rL} and W_{rR}) that are parallel and fixed, and of two front wheels (W_{fL} and W_{fR}) that are steerable. The two steering angles θ_L and θ_R of the front wheels should be synchronized to keep the same instantaneous center of rotation. Thus, these two wheels are kinematically equivalent to a single virtual steerable wheel placed at their midpoint, with a steering angle of θ_S, and the car-like robot has, therefore, similar properties to the tricycle robot (see Section 2.1.2).

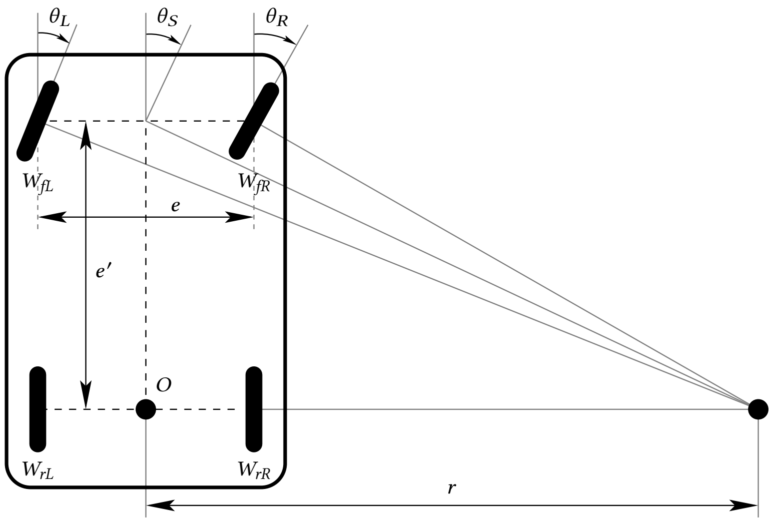


Figure 2.5 – Car-like platform.

As in Section 2.1.2, let e denotes the axle width and e' the wheelbase. A natural choice is to place the reference point O of the robot at the midpoint of the rear wheels axle. Assume that the wheels rotate at a constant speed, if O follows a straight line parallel to the wheels, we have

$$\theta_L = \theta_R = \theta_S = 0, \tag{2.11}$$

and if O follows a circle of radius r, we have

$$\tan\theta_S = \frac{e'}{r}, \tag{2.12}$$

$$\tan\theta_L = \frac{e'}{r - \frac{e}{2}}, \tag{2.13}$$

$$\tan\theta_R = \frac{e'}{r + \frac{e}{2}}. \tag{2.14}$$

A major advantage of a car-like robot is that it is far more stable (than a tricycle) in high-speed turns, since the center of gravity has to remain inside a rectangle instead of a triangle. However, it requires a slightly complicated steering mechanism (e.g., an Ackermann steering geometry [Jaz14]), and it also requires, in most cases, a differential gear and suspensions on the wheels in order to obtain pure rolling of the driven wheels during turns and to keep all the wheels in the ground when the terrain is not perfectly flat.

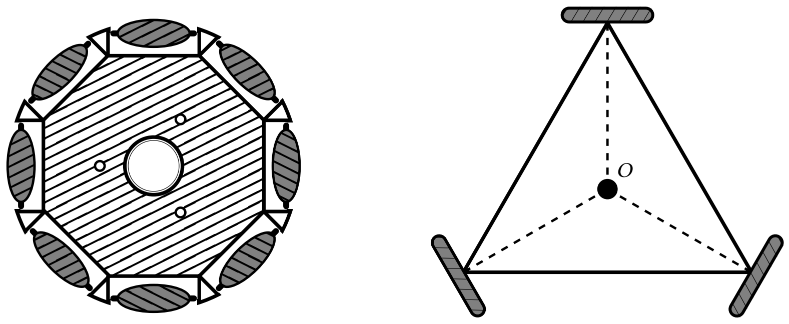
Because of the complicated steering geometry, it is in general complex to build a car-like platform that can achieve a turning radius of zero. Motion planning and control can, therefore, become difficult in the presence of obstacles.

2.1.4 Other Wheeled Robots

There are many other alternative designs for wheeled mobile robots. Robots can in particular be designed with only one wheel (e.g., unicycle robot) or two (e.g., bicycle robot, inverted pendulum, …). Those robots have the advantage to be compact, but they require dynamic control in order to maintain their balance. Three wheels are sufficient for static and dynamic balance, but additional wheels can also be added in order to increase stability. However, if the terrain is not flat, then nontrivial mechanisms are needed to keep all the wheels in contact with the ground.

All the robots presented so far are nonholonomic since they have restricted mobility, due to the fact that the wheels are assumed to roll without slipping. In other words, the allowable direction of movement of a wheel center is limited to the possible orientations of the wheel, and then, for example, a differential drive robot can not move in a direction perpendicular to its fixed wheels.

There exist wheeled robots that are omnimobile (i.e., able to move in any direction) and holonomic (i.e., that do not need any reorientation of the wheels before arbitrary motion). One of the most common example is the omnimobile robot with Swedish wheels. The Swedish wheel, as illustrated for example in Figure 2.6(a), is a special wheel designed to ensure three degrees of freedom for plane motion. In order to eliminate the nonholonomic constraint, small rollers, that are free to rotate, are placed along the outer rim of the wheel allowing free lateral motion of the wheel. This results in a wheel that can be actively driven in its main direction of travel, but for which the lateral velocity is passively determined by the motion of the other wheels.



(a) Swedish wheel. (b) Omnimobile platform.

Figure 2.6 – Omnimobile robot with Swedish wheels.

Three Swedish wheels are necessary and sufficient to get the three degrees of freedom needed to build an holonomic omnidirectional robot (see Figure 2.6(b)). The platform itself is very simple, but the design of a satisfying Swedish wheel is quite complex. There is almost inevitably discontinuities between the rollers, that can cause vibrations and result in poor grip. These wheels are also more fragile than conventional ones, and are not recommended for heavy payloads or with non-flat environment. In this thesis, we do not consider holonomic mobile robots, but he who can do more can do less, and all the trajectories that can be followed by a nonholonomic wheeled mobile robot, can also be followed by an holonomic one.

2.2 Environment

2.2.1 Localization

Assuming nonslip and pure rolling condition of all wheels, it is possible to develop a model of the configuration (position and orientation) of a nonholonomic wheeled robot as a function of time.

Using an external reference frame as illustrated in Figure 2.7, if $x(t)$ and $y(t)$ represent the coordinates of the robot reference point O, and $\theta(t)$ characterizes the orientation of the

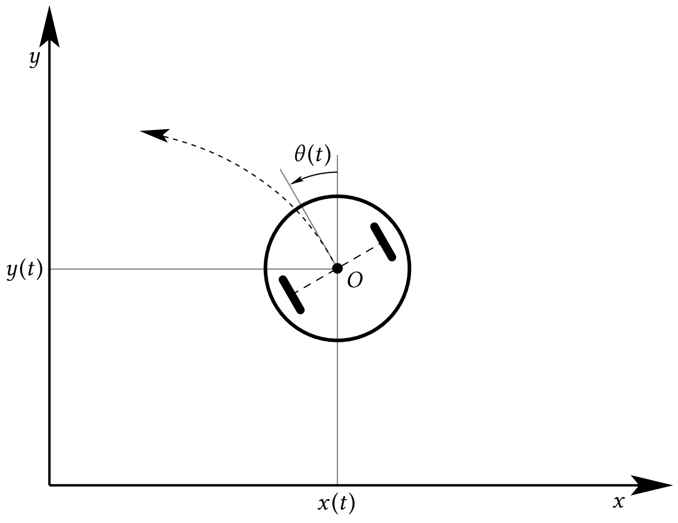


Figure 2.7 – Configuration variables.

robot's chassis[1], we have

$$\dot{x} = v \cos\theta \tag{2.15}$$

$$\dot{y} = v \sin\theta \tag{2.16}$$

$$\dot{\theta} = \omega \tag{2.17}$$

where v denotes the longitudinal velocity and ω the instantaneous velocity of rotation of the reference point.

The configuration at time t of the robot can then be obtained by integration:

$$x(t) = \int_0^t \dot{x}(\tau)\, d\tau + x_0 = \int_0^t v(\tau) \cos\theta(\tau)\, d\tau + x_0 \tag{2.18}$$

$$y(t) = \int_0^t \dot{y}(\tau)\, d\tau + y_0 = \int_0^t v(\tau) \sin\theta(\tau)\, d\tau + y_0 \tag{2.19}$$

$$\theta(t) = \int_0^t \dot{\theta}(\tau)\, d\tau + \theta_0 = \int_0^t \omega(\tau)\, d\tau + \theta_0 \tag{2.20}$$

We see in Section 2.1, that v and ω depend only on the angular speeds $\dot{\psi}_L$ and $\dot{\psi}_R$ of the driving wheels for a differential drive platform, and on the angular speed $\dot{\psi}_S$ of the steering wheel as well on the steering angle θ_S for tricycle and car-like platforms. Thus, given these

[1] The orientation of the robot is arbitrarily chosen such that $\theta(t) = 0$ corresponds to a direction of motion that follows the y-axis.

two control inputs and an initial configuration, it is now possible to estimate the state of an idealized robot using this motion model at any time t. This procedure is called *odometry.*

Given such a model and assuming perfect knowledge of the control inputs, we should, in principle, be able to accurately estimate the configuration of a nonholonomic wheeled robot at any time. Unfortunately, the world is not perfect, and there are a lot of elements that can introduce imprecisions between the reckoning and the true motion of the robot.

First, motors and motor controllers are not ideal, and errors may arise between desired wheel speed and true speed. For differential drive, this problem can be alleviated using sensors (e.g., encoders) on the driving wheels. For tricycle or car-like platform, sensors can be added on the steering mechanism, but this solution is more complicated and somehow less precise than for the differential wheeled robot (discretization errors on the steering angle are more problematic than the ones on the rotation of the wheels[2]). But, as discussed in Section 2.1, the trajectories suited for tricycle or car-like platforms correspond to the ones obtained for a differential drive, which implies that using encoders on the two free fixed wheels is sufficient to estimate the overall motion of the robot.

Next, imprecisions in the modeling of the robot (such as errors on the measured axle width, on the wheel radius, ...) as well as those due to physical assumptions (slippage, wheel compaction, default of ground planarity, ...) are unavoidable. These errors are difficult to predict, and in order to have a good long term configuration estimation, other sensor systems will be required. It is worth noting that the encoders on the free wheels on a tricycle platform are less vulnerable to slippage than on a differential drive, since a free wheel is less likely to skid than a driven one.

In practice, the odometry processing will of course be discretized. Sensor values are sampled at regular intervals, assuming, if the time intervals are sufficiently short, that the wheel speeds are constant between two successive measurements.

As already mentioned, when both wheels rotate at a constant speed, the robot either follows a straight line or a circular arc, and the estimation of the robot configuration can be updated as follows:

$$d_r = r_W\, p_r,\; d_l = r_W\, p_l \tag{2.21}$$

where r_W denotes the radius of the wheels equipped with sensors and p_r, p_l are the measured angular displacements (between time t_n and t_{n+1}) of these wheels.

[2] Encoder discretization errors for a differential drive robot that follows a straight line can lead to position errors; but discretization errors on the steering angle for a tricycle robot that follows the same line can lead to position and orientation errors.

If $d_r \simeq d_l$:

$$\theta(t_{n+1}) = \theta(t_n), \tag{2.22}$$

$$x(t_{n+1}) = x(t_n) - d_l \sin\theta(t_n), \tag{2.23}$$

$$y(t_{n+1}) = y(t_n) + d_l \cos\theta(t_n). \tag{2.24}$$

If $d_r \neq d_l$:

$$d\theta = \frac{d_r - d_l}{e}, \tag{2.25}$$

$$a_1 = \sin\theta(t_n) \cdot \frac{e}{2} \cdot \frac{d_l + d_r}{d_r - d_l}, \tag{2.26}$$

$$a_2 = \cos\theta(t_n) \cdot \frac{e}{2} \cdot \frac{d_l + d_r}{d_r - d_l}, \tag{2.27}$$

where e denotes the distance between the two wheels, and

$$\theta(t_{n+1}) = \theta(t_n) + d\theta, \tag{2.28}$$

$$x(t_{n+1}) = x(t_n) - a_1 \sin d\theta + a_2 \cos d\theta - a_2, \tag{2.29}$$

$$y(t_{n+1}) = y(t_n) + a_1 \cos d\theta + a_2 \sin d\theta - a_1, \tag{2.30}$$

2.2.2 Configuration Space

The workspace (or world), denoted as $\mathcal{W}$, is the space in which the robot moves. It can be modeled either in $\mathbb{R}^2$ or $\mathbb{R}^3$. In the case of wheeled mobile robots moving in a relatively flat environment, a two-dimensional space is sufficient to describe the planning problem as the robot will be considered to be in permanent contact with the ground. This space contains a set of obstacles. Let the *obstacle region* $\mathcal{O}$ denotes the set of all points in $\mathcal{W}$ that are covered by those obstacles; hence, $\mathcal{O} \subseteq \mathcal{W}$.

A general idea is to represent the position and the orientation of the mobile robot as a point in a properly modeled space and to map the workspace obstacles in this space. This space is called the *configuration space* or $\mathcal{C}$*-space* and is denoted by $\mathcal{C}$. The number n of degrees of freedom (DOF) of a mobile robot $\mathcal{A}$ in $\mathcal{W}$ is the number of independent parameters that define its configuration. The *configuration space* or $\mathcal{C}$*-space*, denoted by $\mathcal{C}$, is then defined as the n-dimensional space of all possible configurations of $\mathcal{A}$.

Any configuration of $\mathcal{A}$ in $\mathcal{W}$ should correspond to only one point in $\mathcal{C}$*-space* and a continuous motion in $\mathcal{W}$ should correspond to a continuous motion in $\mathcal{C}$*-space*. The $\mathcal{C}$*-space* of a particular mobile robot is then described as a composition of $\mathcal{C}$*-spaces* that correspond to each of its degree of freedom. This mapping transforms the motion planning problem of an object in $\mathcal{W}$ into the problem of planning the motion of a point in $\mathcal{C}$*-space*.

The reason that we could do so was that these configuration spaces have local topological and differential structures that are identical to those of $\mathbb{R}^n$, such spaces are called *smooth manifolds*. A subset of $\mathbb{R}^m$ is a smooth (or infinitely differentiable) n-dimensional manifold if and only if it is locally diffeomorphic to $\mathbb{R}^n$. Lines or circles are examples of one-dimensional smooth manifolds; planes, spheres or cylinders are examples of two-dimensional smooth manifolds.

For a rigid mobile platform which can translate and rotate in a two-dimensional space, the configuration space[3] is the three-dimensional smooth manifold $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$. First, the two DOFs resulting from translation yield to the smooth manifold $\mathcal{M}_1 = \mathbb{R}^2$. Next, any angle $\theta \in (-\pi, \pi]$ can be applied for the rotational DOF, but the rotation angle does not need to be limited, as for example $\theta = 0$ yields to the same rotation as $\theta = 2\pi$. This results in a "circular" manifold $\mathcal{M}_2 = \mathbb{S}^1$, where $\mathbb{S}^1$ can be defined as the set $(-\pi, \pi]$ in which the addition of two numbers a and b corresponds to $c = a+b+2k\pi$ where $k \in \mathbb{Z}$ is chosen in order to have $c \in (-\pi, \pi]$. The $\mathcal{C}$*-space* that corresponds to all possible motions of a rigid platform in two-dimensional workspace is then $\mathcal{C} = \mathcal{M}_1 \times \mathcal{M}_2 = \mathbb{R}^2 \times \mathbb{S}^1$. Similarly, the $\mathcal{C}$*-space* of a free-flying rigid object in three-dimensional workspace[4] is $\mathcal{C} = \mathbb{R}^3 \times \mathrm{SO}(3) = \mathrm{SE}(3)$. Note that, the concept of manifold is only introduced here in order to properly define the configuration space, it will not be used further in this document.

The $\mathcal{C}$*-space* is composed of two parts: the obstacle region and the free space. Let q denotes the configuration of the robot $\mathcal{A}$. The *obstacle space* $\mathcal{C}_{obs}$ is defined as

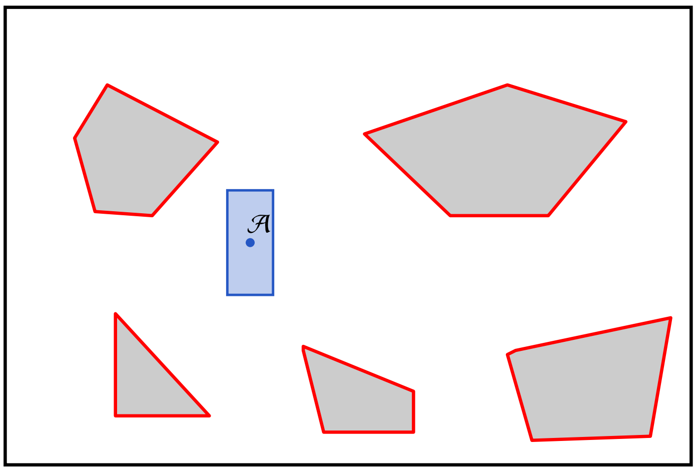
$$\mathcal{C}_{obs} = \{q \in \mathcal{C} \,|\, \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\}\,. \tag{2.31}$$

[3] N.B.: If $\mathcal{M}$ and $\mathcal{N}$ are smooth manifolds then the Cartesian product $\mathcal{M} \times \mathcal{N}$ is a smooth manifold. Note also that, to be exact, $\mathcal{C} \cong \mathbb{R}^2 \times \mathbb{S}^1$ should be used instead of $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}^1$ to indicate that $\mathcal{C}$ could be any space diffeomorphic to $\mathbb{R}^2 \times \mathbb{S}^1$. However, throughout this thesis, this notation will be avoided.

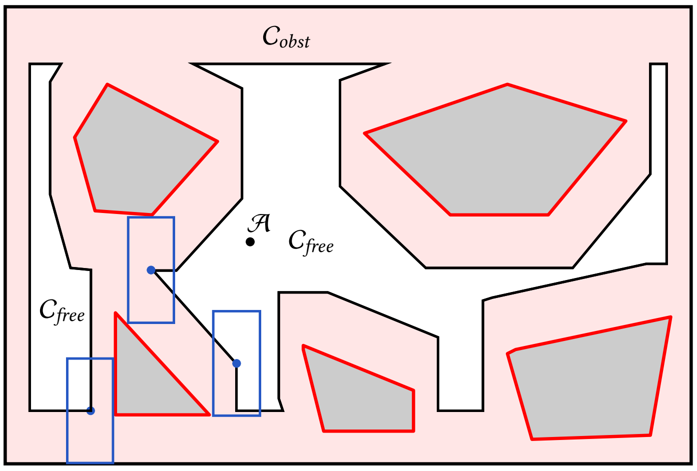
[4] To be exact, SO(3) and SE(3) are Lie groups (i.e. groups that are also smooth manifolds).

This space corresponds to the set of all configurations q, at which $\mathcal{A}(q)$, the *robot placement* in $\mathcal{W}$ (the set of points of $\mathcal{W}$ occupied by the robot in configuration q), touches or intersects the obstacle region $\mathcal{O}$. The set of configurations that are not in the obstacle space is called the *free space*, which is defined as

$$C_{free} = C \setminus C_{obs}. \tag{2.32}$$



(a) Workspace.



(b) *C-space.*

Figure 2.8 – Configuration space corresponding to the workspace of a rigid, translating rectangle platform $\mathcal{A}$. C is two-dimensional ($C = \mathbb{R}^2$). $\mathcal{A}$ is only a point in *C-space.*

2.2.3 Obstacles and Physical Constraints

A path Π for $\mathcal{A}$ is a continuous map $\Pi \in [0, 1] \rightarrow C$ which describes geometrically the motion of the robot $\mathcal{A}$ in the workspace. The points $\Pi(0)$ and $\Pi(1)$ are called respectively the *initial* and the *final configurations* of Π and we say Π is a path from $\Pi(0)$ to $\Pi(1)$. A path is called *collision-free* if it fully lies in C_{free}, i.e. $\forall t \in [0,1] : \Pi(t) \in C_{free}$. Any closed bounded interval $[a,b]$ where $a < b$ could of course replace the unit interval $[0,1]$, but, since $[a,b]$ is homeomorphic to $[0,1]$, this will not represent a gain of generality.

For many applications, in order for example to take into account modeling or localization errors, a minimum clearance needs to be ensured with respect to obstacles. A collision-free path Π for $\mathcal{A}$ is said to have a *path clearance* of c from the obstacles O if $c = \min_{t \in [0,1]} dist(\mathcal{A}(\Pi(t)),\ O)$, where *dist* denotes the Euclidean distance in the workspace.

This notion of clearance can also be used in some cases to simplify the motion planning problem. Consider a platform that has a cylindrical shape (i.e., a robot at a position O occupies a region of the space that corresponds to a disk centered in O). If this robot is in a safe configuration (the robot does not touch or intersect any obstacle), any configuration with the same position and a different orientation remains safe. Therefore, in order to find a collision-free path, it is not necessary to consider the orientation of the robot, and a robot consisting of the simple point O with a minimum clearance to the obstacles chosen to be equal to the radius of this disk will be equivalent to the cylindrically shaped one.

One remarks that a wheeled robot is often symmetrical and it is usually not necessary to consider its exact shape. For example, a differential drive robot is usually well approximated by its smallest covering disk, centered at the midpoint of the segment that links the two locomotion wheels (the reference point O). This approximation is safe, since a collision-free path for the covering disk is necessarily collision-free for the real robot, and it allows to use the point approximation discussed in the previous paragraph.

A trajectory τ for $\mathcal{A}$ is a continuous map $\tau \in [t_{MIN},\ t_{MAX}] \rightarrow C$ which describes the motion of the robot $\mathcal{A}$ in the workspace as a function of time (within a certain time interval $[t_{MIN},\ t_{MAX}]$). A trajectory is *admissible* (or *feasible*) for $\mathcal{A}$ if it complies with the different physical constraints of the robot. To be admissible, the trajectory must be collision-free (i.e. $\forall t \in [t_{MIN},\ t_{MAX}] : \tau(t) \in C_{free}$) and must be consistent with the kinematic model of the robot (nonholonomic constraints, bounds on the admissible velocity, acceleration of individual wheels, tangential accelerations of its center of mass, ...).

It is easy to understand that every collision-free path can not be mapped to an admissible trajectory, which is why motion planning for mobile robots, and for nonholonomic wheeled ones in particular, is a difficult problem.

2.2.4 Eurobot Framework

As mentioned previously, this work has primarily been motivated by our participation to the Eurobot[5] contest.

Eurobot is an international robotics competition created in 1998 and opened to teams of students or independent clubs. This event mainly takes place in Europe, but teams from all over the world are welcome to participate. Nowadays, more than 400 teams from about 30 countries are involved each year in this contest.

In each game, two teams composed of one or two[6] autonomous mobile robots, compete during 90 seconds in a $2 \times 3\,\mathrm{m}^2$ area strewn with obstacles. The winning team is the one who scores the most.

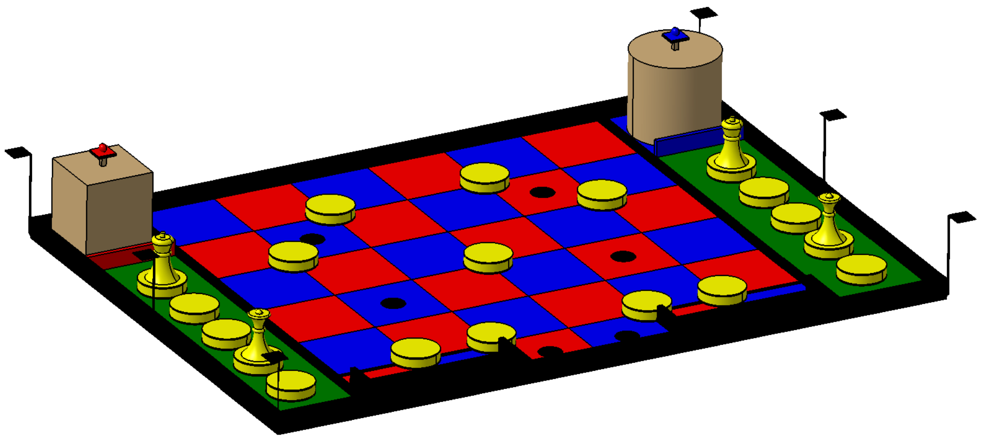


Figure 2.9 – Example of Eurobot playground. ©Eurobot — http://www.eurobot.org

The rules for counting points change every year. As an example, we briefly describe the 2011 version of the rules. As shown in Figure 2.9, the playground consists mostly of a 6 by 6 red and blue chessboard. Each team may use only one robot and has to move yellow pawns on the playground. One team plays in red and the other in blue. After 90 seconds, each yellow pawn fully on a square scores points for the team of the corresponding color. There are also four special pawns or figures that score more points, and if these figures are stacked on top of one or two regular pawns, they score even more. Finally, some squares on the playground contain a black dot, which give bonus points if pawns are located on it at the end of the game. It is also worth noting that the position of the yellow pawns at the beginning of the game is randomly selected among dozens of options and can not be communicated to the robots.

[5] http://www.eurobot.org

[6] Some years, the rules allow only one robot per team, as is the case in the ones presented in this section.

Taking into account the random initial configuration, and the fact that the game situation changes constantly and quickly, it is easy to understand that it is almost impossible to build an efficient autonomous robot by precomputing all possible trajectories. Some other significant points can also be highlighted. To perform well, robots should be able to react quickly to changing situations, to move quickly without disturbing well-placed pawns and without colliding with the opponent, and to catch pawns and construct towers while moving.

In order to participate to the contest, several differential and tricycle robotic platforms have been developed since 2008 at the University of Liège. Each platform is composed of a low-cost embedded computer running real-time Linux, together with several custom-made electronic cards. All those cards are used to pilot the actuators and the different sensors and are addressed 50 times per second. For example, some of those cards are used to efficiently speed control the locomotion motors. These motors are designed to achieve a robot speed of more than $1.5\,\mathrm{m\,s^{-1}}$ and acceleration of more than $3\,\mathrm{m\,s^{-2}}$.

The localization of the robot in the playground is estimated by using odometry (as seen in Section 2.2.1) and an original global positioning system [Pie13] based on a rotating turret and a set of three beacons that the Eurobot rules allow to place around the play field. Data are mixed together by an extended Kalman filter in order to obtain an accurate and reliable position of the robot at any time. The position of game elements and opponents are obtained with a certain degree of precision through, for example, the use of 3D cameras or ultrasound sensors. These positions are regularly updated at the rate of several tens of Hz.

In summary, for this contest, since the game situation changes quickly, it is essential to plan trajectories in real-time, employing a method with very low computational cost, given the limited amount of computing power available on the robot. Information about future locations that will be visited by the robot and their associated timestamps is essential to efficiently implement catching objects and building operations while moving (in order to coordinate the different actuators). Finally, the generated trajectories need to reach quickly their target while taking into account the obstacles and the physical constraints (limited speed or acceleration) of the robot.

The developments of this thesis have been successfully implemented and tested in this framework (for both differential and tricycle platforms), where they have proven to offer a clear competitive advantage over the designs of other teams.

CHAPTER 3

Paths and Trajectories

This chapter discusses the spatial and temporal properties of paths and trajectories suited for nonholonomic mobile robots. It shows how to extract useful information from them, such as, for example, the travel length, the speed and acceleration of the reference center of the robot, or the speed and acceleration of its individual wheels. This chapter then introduces an original data structure that can be used to represent such paths and trajectories using only a finite amount of information. This structure allows to efficiently extract accurate approximations of the configuration of the robot at all times. It also makes possible to reason about physical constraints, such as bounds on the speed or acceleration of the robot or its wheels, and will be used in Chapter 4 in order to compute an optimal speed profile.

As discussed in Section 2.2.3, a path Π describes geometrically how the robot moves from an initial to a final configuration with respect to the value of some parameter. When this parameter is time, this path is called a trajectory.

It is worth mentioning that, in this chapter, we merely study the geometrical properties of trajectories, without taking into account the physical constraints imposed on the speeds and accelerations of a robot following them. (The problem of computing the speed at which a given path can be followed will be addressed in Chapter 4.) In this perspective, time can be seen as an arbitrary parameter, the only requirement on which being that increasing its value makes the trajectory proceed further, and does not necessarily have to match physical time. For simplicity sake, at this point, we explicitly forbid the configuration $\tau(t)$ to remain constant over any interval of non-zero length[1], for every trajectory τ.

[1] This restriction will be lifted later in Section 3.2.5, in order to allow trajectories in which the robot pauses for some amount of time, for instance in order to perform some actions.

The coordinate system used for expressing configurations is identical to the one used in Section 2.2.1: the position of the reference point O of the robot is given by its Cartesian coordinates $(x(t), y(t))$, and its orientation by a function $\theta(t)$, such that $\theta(t) = 0$ corresponds to a direction of motion that follows the y-axis. In this setting, the configuration of the robot at time t is thus specified by the tuple $\tau(t) = (x(t), y(t), \theta(t))$.

Consider a trajectory τ, mapping each instant $t \in [t_{MIN}, t_{MAX}]$ onto a configuration $\tau(t) = (x(t), y(t), \theta(t))$. If this trajectory is physically feasible, then we can assume that the functions $x(t)$, $y(t)$ and $\theta(t)$ admit continuous first and second derivatives[2] over the interval $[t_{MIN}, t_{MAX}]$. For every function $f(t)$, we will use the notations $f'(t)$ and $f''(t)$ to denote respectively its first and second derivatives with respect to t.

3.1 Spatial and Temporal Properties

3.1.1 Spatial Properties

At each time t, the vector

$$\vec{v}(t) = \begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix}$$

is either equal to $\vec{0}$ or tangent to the trajectory. The first case can either correspond to a rotation of the robot around its reference point (a *local turn*), or to a robot stop (e.g., at the initial or final configuration). In the latter case, assuming a differential or tricycle platform moving forwards, the orientation of the robot at time t has to follow this vector (due to nonholonomic constraints, as seen in Section 2.1), which implies

$$\theta(t) = \arctan(-x'(t),\ y'(t)), \tag{3.1}$$

where $\arctan(\alpha,\ \beta)$ is the usual generalization of $\arctan \frac{\alpha}{\beta}$ that

- takes into account the signs of α and β for producing a value in $(-\pi, \pi]$, and
- handles the special case $\beta = 0$.

Over an infinitesimal delay dt, the robot moves by the distance

$$ds = \left\| \begin{pmatrix} x'(t) \\ y'(t) \end{pmatrix} \right\| dt = \sqrt{x'^2(t) + y'^2(t)}\, dt.$$

[2] The value of $\theta(t)$ is only actually known up to an integer multiple of 2π. In the following, we assume that the values of $\theta(t)$ are restricted to the interval $(-\pi, \pi]$, but nonetheless consider that this function admits continuous first and second derivatives.

It follows that, during the time interval $[a, b]$, the total linear distance $s_{[a,b]}$ covered by the robot can be expressed as

$$s_{[a,b]} = \int_a^b \sqrt{x'^2(t) + y'^2(t)}\, dt. \tag{3.2}$$

Moreover, during the delay dt, the orientation of the robot is increased by

$$d\theta = \theta'(t)\, dt.$$

If the robot is not performing a local turn at time t, introducing (3.1) in the expression of $d\theta$ gives

$$d\theta = \frac{x'(t)\, y''(t) - x''(t)\, y'(t)}{x'^2(t) + y'^2(t)}\, dt.$$

This expression can be used for computing the *curvature* $\kappa(t)$ of the trajectory at time t, defined as

$$\kappa(t) = \frac{d\theta}{ds},$$

which becomes

$$\kappa(t) = \frac{x'(t)\, y''(t) - x''(t)\, y'(t)}{\left(x'^2(t) + y'^2(t)\right)^{\frac{3}{2}}}. \tag{3.3}$$

Intuitively, a curvature $\kappa(t) = 0$ induces a constant orientation around time t, corresponding to a straight trajectory. On the other hand, a curvature $\kappa(t) \neq 0$ describes a trajectory that performs an instantaneous rotation of radius

$$r(t) = \frac{1}{\kappa(t)}, \tag{3.4}$$

the sign of which follows the convention introduced in Sections 2.1.1 and 2.1.2, a positive (resp. negative) value of r means that the center of the circle is located on the left (resp. right) of O. Abusing the notation, we write $\kappa(t) = +\infty$ (resp. $\kappa(t) = -\infty$) in the case of a local turn to the left (resp. right) at time t. Note that both cases of local turns correspond to $r(t) = 0$.

Recall that, in the case of a tricycle platform, the orientation $\theta_S(t)$ of the steering wheel satisfies

$$r(t) = e' \cot \theta_S(t),$$

if $\theta_S(t) \neq 0$ (where e' denotes the wheelbase of the robot), and $\kappa(t) = 0$ if $\theta_S(t) = 0$.

This yields

$$\theta_S(t) = \arctan(e'\kappa(t)), \tag{3.5}$$

with $\theta_S(t) = \frac{\pi}{2}$ if $\kappa(t) = +\infty$, and $\theta_S(t) = -\frac{\pi}{2}$ if $\kappa(t) = -\infty$ (these special cases both correspond to local turns).

3.1.2 Temporal Properties

In the case where the parameter t corresponds to physical time, one can extract from a trajectory specification the speed $v(t)$, angular speed $\dot{\theta}(t)$, and acceleration vector $\vec{a}(t)$ of the reference point O:

$$\begin{aligned} v(t) &= \frac{ds}{dt} \\ &= \sqrt{x'^2(t) + y'^2(t)}, \end{aligned} \tag{3.6}$$

$$\begin{aligned} \dot{\theta}(t) &= \frac{d\theta}{dt} \\ &= \frac{x'(t)\,y''(t) - x''(t)\,y'(t)}{x'^2(t) + y'^2(t)} \\ &= \kappa(t)\,v(t) \quad \text{if } \kappa(t) \notin \{-\infty, +\infty\}, \end{aligned} \tag{3.7}$$

$$\begin{aligned} \vec{a}(t) &= \frac{d^2}{dt^2}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} \\ &= \begin{pmatrix} x''(t) \\ y''(t) \end{pmatrix}. \end{aligned} \tag{3.8}$$

The acceleration vector $\vec{a}(t)$ can more conveniently be expressed with respect to a reference frame that is attached to the robot. This makes it possible, for example, to check whether the robot is sufficiently supported by its support polygon. The *tangential* acceleration $a_T(t)$ and the *radial (or centripetal)* acceleration $a_C(t)$ are respectively defined as

$$a_T(t) = \vec{a}(t) \cdot \begin{pmatrix} -\sin\theta(t) \\ \cos\theta(t) \end{pmatrix}$$

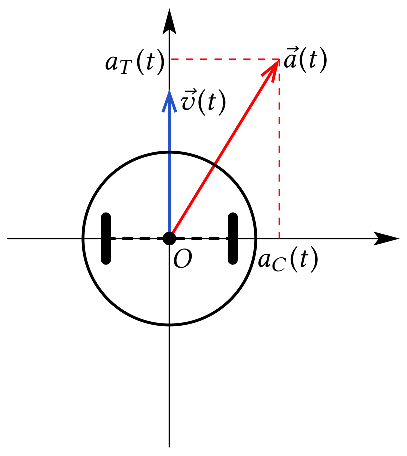
$$= -\sin\theta(t)\, x''(t) + \cos\theta(t)\, y''(t), \tag{3.9}$$

and

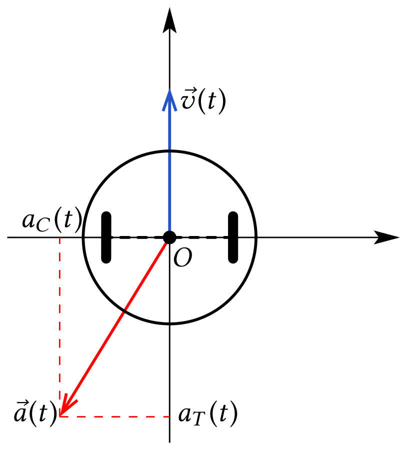
$$a_C(t) = \vec{a}(t) \cdot \begin{pmatrix} \cos\theta(t) \\ \sin\theta(t) \end{pmatrix}$$

$$= \cos\theta(t)\, x''(t) + \sin\theta(t)\, y''(t). \tag{3.10}$$

Intuitively, the tangential acceleration is the projection of $\vec{a}(t)$ onto an axis going from the rear to the front of the robot, whereas the radial acceleration corresponds to the projection of $\vec{a}(t)$ onto a left-to-right axis (see Figure 3.1).



(a) $a_T(t) > 0, a_C(t) > 0$.



(b) $a_T(t) < 0, a_C(t) < 0$.

Figure 3.1 – Examples of tangential and radial accelerations.

It is worth noting that:

- *If* $v(t) = 0$:

$$a_T(t) = a_C(t) = 0. \tag{3.11}$$

- *If* $v(t) \neq 0$:

As already mentioned, the orientation of a nonholonomic platform moving forwards at time t has to follow the vector $\vec{v}(t)$, and we have

$$\cos\theta(t)\, x'(t) + \sin\theta(t)\, y'(t) = 0.$$

This constraint entails

$$|a_T(t)| = \frac{|x'(t)\, x''(t) + y'(t)\, y''(t)|}{\sqrt{x'^2(t) + y'^2(t)}},$$

and

$$|a_C(t)| = \frac{|y'(t)\, x''(t) - x'(t)\, y''(t)|}{\sqrt{x'^2(t) + y'^2(t)}}.$$

It remains to determine the sign of $a_T(t)$ and $a_C(t)$. Since the robot is moving forwards, one can remark in Figure 3.1 that the tangential acceleration is positive (resp. negative) when the dot product $\vec{v}(t) \cdot \vec{a}(t) = x'(t)\, x''(t) + y'(t)\, y''(t)$ is positive (resp. negative). Then we have

$$a_T(t) = \frac{x'(t)\, x''(t) + y'(t)\, y''(t)}{\sqrt{x'^2(t) + y'^2(t)}} = \frac{d}{dt} v(t). \tag{3.12}$$

Let $\vec{v_p}(t)$ denotes the $\frac{\pi}{2}$ clockwise rotation of $\vec{v}(t)$.

$$\vec{v_p}(t) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \vec{v}(t) = \begin{pmatrix} y'(t) \\ -x'(t) \end{pmatrix}.$$

Then, in a similar way, since the robot is moving forwards, one can remark that the radial acceleration is positive (resp. negative) when the dot product $\vec{v_p}(t) \cdot \vec{a}(t) = y'(t)x''(t) - x'(t)y''(t)$ is positive (resp. negative). We thus have

$$a_C(t) = \frac{y'(t)\, x''(t) - x'(t)\, y''(t)}{\sqrt{x'^2(t) + y'^2(t)}} = -\kappa(t)\, v^2(t). \tag{3.13}$$

It can also be useful to extract the respective (linear) speeds $v_L(t)$ and $v_R(t)$ of the left and right wheels (in the case of a differential or tricycle platform):

- *If* $v(t) = 0$:

$$v_L(t) = -\frac{e}{2}\,\dot{\theta}(t) \tag{3.14}$$

$$v_R(t) = \frac{e}{2}\,\dot{\theta}(t). \tag{3.15}$$

- *If* $v(t) \neq 0$:

$$v_L(t) = \left(1 - \frac{e\kappa(t)}{2}\right) v(t) \tag{3.16}$$

$$v_R(t) = \left(1 + \frac{e\kappa(t)}{2}\right) v(t), \tag{3.17}$$

where e denotes the axle width of the robot.

Also, the respective tangential accelerations $a_L(t)$ and $a_R(t)$ at the left and right wheels can be computed, for example to estimate the torque transmitted between the wheels and the ground with a differential drive platform.

- *If* $v(t) = 0$:

$$a_L(t) = -\frac{e}{2} \cdot \frac{d}{dt}\dot{\theta}(t) \tag{3.18}$$

$$a_R(t) = -a_L(t). \tag{3.19}$$

- *If* $v(t) \neq 0$:

$$\begin{aligned} a_L(t) &= \frac{d}{dt}v_L(t) \\ &= \left(1 - \frac{e\kappa(t)}{2}\right)\frac{d}{dt}v(t) - \frac{e}{2}\,v(t)\,\frac{d}{dt}\kappa(t) \end{aligned} \tag{3.20}$$

$$\begin{aligned} a_R(t) &= \frac{d}{dt}v_R(t) \\ &= \left(1 + \frac{e\kappa(t)}{2}\right)\frac{d}{dt}v(t) + \frac{e}{2}\,v(t)\,\frac{d}{dt}\kappa(t). \end{aligned} \tag{3.21}$$

The next step is to express the linear speed $v_S(t)$ and the tangential acceleration $a_S(t)$ of the steering wheel, in the case of a tricycle platform.

We get:

- *If* $v(t) = 0$:

$$v_S(t) = e'|\dot{\theta}(t)| \tag{3.22}$$

$$\begin{aligned} a_S(t) &= \frac{d}{dt} v_S(t) \\ &= e' \frac{d}{dt} |\dot{\theta}(t)|. \end{aligned} \tag{3.23}$$

- *If* $v(t) \neq 0$:

$$\begin{aligned} v_S(t) &= \frac{v(t)}{\cos \theta_S(t)} \\ &= v(t)\sqrt{1 + \tan^2 \theta_S(t)} \\ &= v(t)\sqrt{1 + (e'\kappa(t))^2} \end{aligned} \tag{3.24}$$

$$\begin{aligned} a_S(t) &= \frac{d}{dt} v_S(t) \\ &= \sqrt{1 + (e'\kappa(t))^2}\, \frac{d}{dt} v(t) + \frac{\frac{d}{dt}\kappa(t)\, v(t)\, e'^2\, \kappa(t)}{\sqrt{1 + (e'\kappa(t))^2}}. \end{aligned} \tag{3.25}$$

where e' denotes the wheelbase of the robot.

Finally, the angular speed of steering $\dot{\theta}_S(t)$ (which may be useful for reasoning about constraints in the steering mechanism) is given by:

$$\begin{aligned} \dot{\theta}_S(t) &= \frac{d}{dt} \theta_S(t) \\ &= \frac{e'}{1 + (e'\kappa(t))^2}\, \frac{d}{dt} \kappa(t). \end{aligned} \tag{3.26}$$

3.1.3 Backward Motion

Up to this point, we have only considered trajectories in which the robot either moves forwards, or performs a local turn. In order to reason about backward moves, a simple solution consists in considering another robot that follows exactly the same trajectory, but with an opposite orientation, and to relate the trajectory variables to those of this robot.

Formally, for any function $f(t)$ describing the evolution of a trajectory variable for a robot moving forwards (such as a position, velocity, or acceleration component), $\overline{f}(t)$ will denote the corresponding function for a robot performing the same move backwards, i.e., reaching the same positions at the same time but with an opposite orientation. In order to normalize orientations, we also introduce the notation $[\alpha]$ to denote an angle belonging to the interval $(-\pi, \pi]$ that is equal to α up to an integer multiple of 2π.

For a configuration $(x(t), y(t), \theta(t))$, we thus have

$$\begin{aligned}\bar{x}(t) &= x(t)\\ \bar{y}(t) &= y(t)\\ \bar{\theta}(t) &= [\theta(t) + \pi]\\ &= \arctan(\bar{x}'(t),\ -\bar{y}'(t)).\end{aligned}$$

The infinitesimal distance $d\bar{s}$ and orientation increment $d\bar{\theta}$ during a delay dt are given by

$$\begin{aligned}d\bar{s} &= -ds\\ d\bar{\theta} &= d\theta.\end{aligned}$$

Note that we use a signed notion of distance. The value of $d\bar{s}$ corresponds to an increment of the linear position of the robot along its trajectory. A negative value of $d\bar{s}$ thus simply means that the robot is moving backwards, i.e., that its linear position is decreasing with time. This leads to

$$\begin{aligned}\bar{\kappa}(t) &= -\kappa(t)\\ \bar{r}(t) &= -r(t).\end{aligned}$$

Note that a positive (resp. negative) value of $\bar{\kappa}(t)$ or $\bar{r}(t)$ still means that the robot is performing a left (resp. right) turn, observed from the reference orientation of the robot. (Intuitively, with a tricycle robot, a left (resp. right) turn is one that can be followed, either forwards or backwards, by steering left (resp. right).) However, because of the backward motion, a positive (resp. negative value) of $\overline{\kappa}(t)$ now implies that the orientation of the robot is decreasing (resp. increasing) with time.

For a tricycle platform, the orientation of the steering wheel is given by

$$\overline{\theta_S}(t) = -\theta_S(t) = \arctan(e'\overline{\kappa}(t)) ,$$

with $\overline{\theta_S}(t) = \frac{\pi}{2}$ if $\overline{\kappa}(t) = +\infty$, and $\overline{\theta_S}(t) = -\frac{\pi}{2}$ if $\overline{\kappa}(t) = -\infty$.

Similarly, one obtains the speed, angular speed, and acceleration vector of the reference point of the robot moving backwards:

$$\begin{aligned}
\overline{v}(t) &= -v(t) \\
\overline{\dot{\theta}}(t) &= \dot{\theta}(t) \\
\overline{\vec{a}}(t) &= \vec{a}(t).
\end{aligned}$$

The tangential and radial components of $\overline{\vec{a}}(t)$, and the linear speeds and tangential accelerations of the left and right wheels can be obtained by reversing the geometry of the robot:

$$\begin{aligned}
\overline{a_T}(t) &= -a_T(t) \\
\overline{a_C}(t) &= -a_C(t) \\
\overline{v_L}(t) &= -v_R(t) \\
\overline{v_R}(t) &= -v_L(t) \\
\overline{a_L}(t) &= -a_R(t) \\
\overline{a_R}(t) &= -a_L(t).
\end{aligned}$$

Finally, for a tricycle platform, the linear speed, tangential acceleration, and angular speed of the steering wheel are given by

$$\begin{aligned}
\overline{v_S}(t) &= -v_S(t) \\
\overline{a_S}(t) &= -a_S(t) \\
\overline{\dot{\theta}_S}(t) &= -\dot{\theta}_S(t) \\
&= \frac{e'}{1 + (e'\kappa(t))^2} \frac{d}{dt}\overline{\kappa}(t).
\end{aligned}$$

3.2 Discretization

As mentioned in the previous section, a trajectory is defined as a function mapping continuous time onto robot configurations. This formalism is well suited for theoretical developments. However, in order to develop algorithmic tools for synthesizing, analyzing, and transforming trajectories, it is necessary to introduce a discretized representation system, in which trajectories are specified using only a finite amount of information.

Usually, either to extract the configuration of the robot at a specific time, to compute a speed profile or to reason in any other way about a trajectory, an analytical closed-form formulation of this trajectory is used [LJTM94, CC00, FS04, BP11b, ZT13]. If such a formulation exists, it actually represents a finite amount of information, but speed profile computation can quickly become difficult and time-consuming, and, since it requires the evaluation of a potentially complicated closed form (clothoid arcs, lemniscate curves, . . .), even the determination of the robot configuration at a specific time is not trivial.

For the sake of efficiency, an original discretization procedure, capable of representing arbitrary trajectories, is introduced. The trajectories thus discretized are used in Chapter 4 in order to develop a new efficient method for computing a time-optimal speed profile consistent with the physical constraints of the robot. Those discretized trajectories also allow easy evaluation of the robot configuration at all time, and therefore limit, for example, the operations required for trajectory tracking.

It is of course impossible to represent exactly arbitrary trajectories using such a discretized representation, but the goal is to make the approximation accurate enough for our intended applications. The main requirements are the following:

- From a discretized representation of a trajectory, it should be possible to extract an approximation of the configuration of the robot at all times, and this approximation should not significantly differ from its true configuration.
- Reasoning about constraints such as bounds on the speed or acceleration of the robot or its wheels at all times should be possible on the basis of a discretized representation of its trajectory.
- One should be able to discretize a trajectory independently from the speed at which it is to be followed. The aim is to make it possible, for example, to first generate a trajectory satisfying some given constraints, and afterwards compute the quickest speed profile at which this trajectory can be followed.

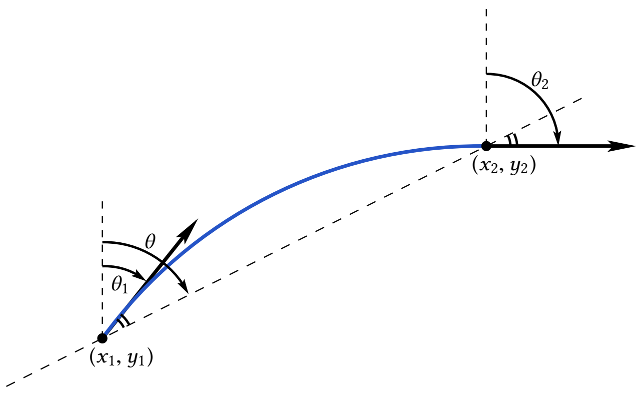


Figure 3.2 – Constant-curvature step.

3.2.1 Constant-Curvature Steps

Let τ be a trajectory defined over the time interval $[t_{MIN},\ t_{MAX}]$. A first approach to discretizing this trajectory consists in sampling its value at a finite number of instants $t_0, t_1, \ldots, t_m$. The trajectory is then represented by the finite sequence of configurations $\tau(t_0); \tau(t_1); \ldots; \tau(t_m)$.

The main problem of this solution is that moving from a configuration $\tau(t_i)$ to a configuration $\tau(t_{i+1})$ can generally not be achieved by a simple locomotion action, i.e., by moving with a constant steering angle, in the case of a tricycle platform. If the precise way in which such a move is performed is unknown, it becomes difficult to reason about the constraints satisfied by the trajectory, for instance inferring bounds on the speed and acceleration experienced between $\tau(t_i)$ and $\tau(t_{i+1})$ is not easy.

Let us study more precisely the problem of moving with a constant steering angle, or in other words with constant curvature from a configuration $\gamma_1 = (x_1, y_1, \theta_1)$ to another one $\gamma_2 = (x_2, y_2, \theta_2)$, which we call a *constant-curvature step*. For simplicity sake, we only consider forward moves, i.e., we assume that the speed of the reference point of the robot is non-negative. For the constant-curvature step to be feasible, there must exist a circular arc (or, as a degenerate case, a straight segment) to which γ_1 is tangent at (x_1, y_1) and γ_2 is tangent at (x_2, y_2). This situation is illustrated in Figure 3.2.

In the special case where (x_1, y_1) coincides with (x_2, y_2), a constant-curvature step is possible regardless of the values of θ_1 and θ_2, and takes the form of a local turn.

If, on the other hand, (x_1, y_1) is distinct from (x_2, y_2), then $\theta = \arctan(x_1 - x_2,\ y_2 - y_1)$ corresponds to the orientation of the segment linking (x_1, y_1) to (x_2, y_2). By the symmetry of the problem, the differences between the orientations θ_1 and θ and between θ and θ_2 have to be identical. This constraint can be expressed as

$$[\theta_1 - \theta] = [\theta - \theta_2]. \tag{3.27}$$

Our aim is to use constant-curvature steps as elementary building blocks for specifying a trajectory between two successive sampling points. It is thus reasonable to impose bounds on the value of $[\theta_1 - \theta]$, since a large difference between the overall direction of a step and its initial orientation can always be prevented by choosing sufficiently small sampling steps. In practice, we will impose

$$-\frac{\pi}{2} < [\theta_1 - \theta] < \frac{\pi}{2}.$$

This choice is motivated by the fact that, when backward trajectories will also be considered, the initial orientation of a constant-curvature step will uniquely characterize its direction of motion.

In summary, we obtain the following criterion:

Theorem 3.2.1. *For a differential or tricycle platform, there exists a forward constant-curvature step leading from the configuration (x_1, y_1, θ_1) to the configuration (x_2, y_2, θ_2) iff either*

$$(x_1, y_1) = (x_2, y_2),$$

or

$$(x_1, y_1) \neq (x_2, y_2) \text{ and } -\frac{\pi}{2} < [\theta_1 - \theta] = [\theta - \theta_2] < \frac{\pi}{2}$$

where $\theta = \arctan(x_1 - x_2,\ y_2 - y_1)$.

3.2.2 Approximated Trajectories

For arbitrary trajectories, the hypotheses of Theorem 3.2.1 are generally not satisfied by two successively sampled configurations. In order to obtain a discretized representation of those trajectories, a possible strategy could then be to consider only trajectories that can be expressed as a sequence of constant-curvature steps. In other words, a trajectory would be represented by a sequence of configurations $\tau(t_0); \tau(t_1); \ldots; \tau(t_m)$ such that for every $i \in [0, m-1]$, there exists a constant-curvature step leading from the configuration $\tau(t_i)$ to the configuration $\tau(t_{i+1})$.

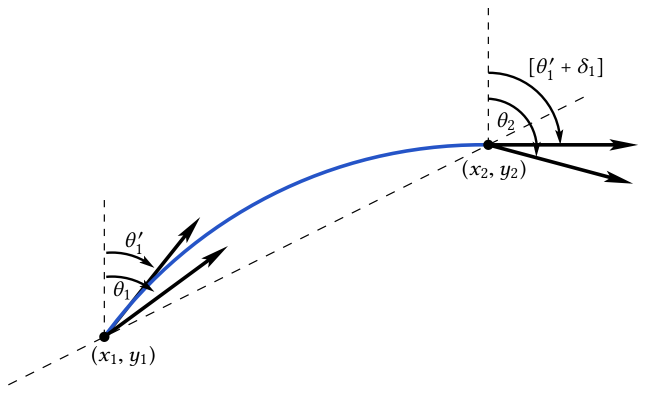


Figure 3.3 – Discretized trajectory.

This solution has the advantage of precisely specifying the trajectory followed by the robot between two samplings, which makes it possible to compute variables such as it speed and acceleration at all times. However, it has the drawback of imposing that the sampling points necessarily coincide with the origins of the constant-curvature steps. Indeed, with an arbitrary sampling of such a trajectory, the hypotheses of Theorem 3.2.1 are generally not satisfied.

In order to alleviate this drawback, one has to lift some restrictions on the allowable forms of trajectories. Our solution consists in expressing a discretized trajectory as a sequence of constant-curvature steps, but without requiring tangency between the circular arcs and the robot orientation at the sampled configurations. In other words, a discretized trajectory takes the form of a sequence of arbitrary configurations $\tau(t_0); \tau(t_1); \dots; \tau(t_m)$. Between a configuration $\tau(t_i) = (x_i, y_i, \theta_i)$ and its successor $\tau(t_{i+1}) = (x_{i+1}, y_{i+1}, \theta_{i+1})$, we consider that the robot moves along a circle arc from (x_i, y_i, θ_i') to $(x_{i+1}, y_{i+1}, [\theta_i' + \delta_i])$, increasing its orientation by the angle $\delta_i = [\theta_{i+1} - \theta_i]$, with orientations θ_i' and $[\theta_i' + \delta_i]$ that do not necessarily match θ_i and θ_{i+1}. This is illustrated in Figure 3.3.

Note that, in order to satisfy the hypotheses of Theorem 3.2.1, the value of θ_i' is uniquely determined from the values of x_i, y_i, x_{i+1}, y_{i+1} and δ_i, provided that $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$: we have in this case

$$\theta_i' = \left[\theta - \frac{\delta_i}{2}\right], \tag{3.28}$$

where $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$. The curvature κ_i of the arc followed by the constant-curvature step is given by

$$\kappa_i = \frac{2}{\lambda_i} \sin \frac{\delta_i}{2}, \tag{3.29}$$

where λ_i is the chord length

$$\lambda_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}. \tag{3.30}$$

3.2.3 Discretizing a Trajectory

We now address the problem of obtaining a discretized representation (x_0, y_0, θ_0); (x_1, y_1, θ_1); $\ldots$; (x_m, y_m, θ_m) of a trajectory τ defined over $[t_{MIN},\ t_{MAX}]$, with respect to the sampling times t_0, t_1, $\ldots$, t_m. Once again, we assume that the trajectory τ proceeds forwards.

For $i \in [0, m-1]$, they are several cases to consider:

- *If* $(x_i, y_i) = (x_{i+1}, y_{i+1})$. Then, a local turn is performed between t_i and t_{i+1}.

- *If* $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$. Then, we have to define a constant-curvature step between t_i and t_{i+1}. Let $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$ be the orientation of the segment linking (x_i, y_i) to (x_{i+1}, y_{i+1}). As discussed in Section 3.2.2, if the constraints

  $$-\frac{\pi}{2} < [\theta_i - \theta] < \frac{\pi}{2}$$

  and

  $$-\frac{\pi}{2} < [\theta - \theta_{i+1}] < \frac{\pi}{2}$$

  are not satisfied, then the sampling of τ is too coarse for the discretization.

  Otherwise, in the time interval $[t_i, t_{i+1}]$, the trajectory τ increases the orientation of the robot by the angle $\delta_i = [\theta_{i+1} - \theta_i]$. We then define a constant-curvature step that moves from (x_i, y_i) to (x_{i+1}, y_{i+1}) and increases the current orientation by δ_i. In other words, we choose the initial orientation θ'_i of the circular arc and its final one $[\theta'_i + \delta_i]$ so as to have

  $$[\theta - \theta'_i] = [\theta'_i + \delta_i - \theta] = \frac{\delta_i}{2},$$

  which yields (as in the previous section)

  $$\theta'_i = \left[\theta - \frac{\delta_i}{2}\right]. \tag{3.31}$$

It is worth mentioning that this choice of values is somehow arbitrary. We now motivate this choice, in particular against the alternative consisting in defining $\theta'_i = \theta_i$ for all i. As we have just mentioned, a first justification is that (3.31) implies the existence from the configuration (x_i, y_i, θ'_i) of a circular arc to $(x_{i+1}, y_{i+1}, [\theta'_i + \delta_i])$, i.e., one that reaches the correct subsequent position in the sampled trajectory and adds the correct increment to the orientation.

A second justification is provided by the following property. Consider two successive constant-curvature steps that are physically realizable by a robot, i.e., configurations $\gamma_A = (x_A, y_A, \theta_A)$, $\gamma_B = (x_B, y_B, \theta_B)$ and $\gamma_C = (x_C, y_C, \theta_C)$ such that there exist a constant-curvature step from γ_A to γ_B as well as one from γ_B to γ_C, with continuity of position and orientation ensured at γ_B. Assume that γ_A and γ_C are two successive samples, respectively in $t = t_i$ and $t = t_{i+1}$, of such a trajectory.

By the discretization process, one obtains

$$(x_i, y_i) = (x_A, y_A)$$

and

$$\theta'_i = \left[\theta - \frac{[\theta_C - \theta_A]}{2}\right],$$

where $\theta = \arctan(x_A - x_C,\ y_C - y_A)$ is the orientation of the segment linking (x_A, y_A) to (x_C, y_C). As previously explained, there exists a constant-curvature step moving from (x_A, y_A, θ'_i) to $(x_C, y_C, [\theta'_i + (\theta_C - \theta_A)])$. The property is that this constant-curvature step necessarily visits (x_B, y_B), which shows that our notion of discretized trajectories closely approximates the realistic ones.

Let us establish that the constant-curvature step that we have mentioned does indeed visit (x_B, y_B). The problem is depicted in Figure 3.4. We define the points $A : (x_A, y_A)$, $B : (x_B, y_B)$, $C : (x_C, y_C)$, and the angles θ_{AB} and θ_{BC} as the respective orientations of the segments linking A to B and B to C, that is,

$$\begin{aligned}\theta_{AB} &= \arctan(x_A - x_B,\ y_B - y_A)\\ \theta_{BC} &= \arctan(x_B - x_C,\ y_C - y_B).\end{aligned}$$

From Theorem 3.2.1, we get

$$\begin{aligned}\theta_{AB} &= \left[\frac{\theta_A + \theta_B}{2}\right]\\ \theta_{BC} &= \left[\frac{\theta_B + \theta_C}{2}\right].\end{aligned}$$

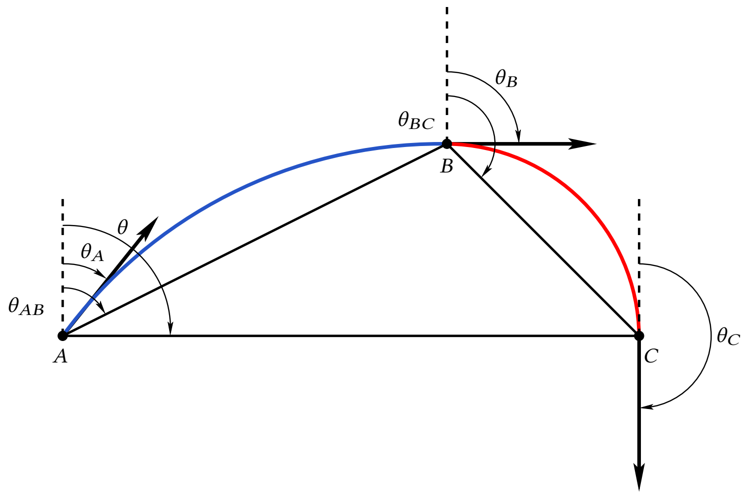


Figure 3.4 – Two successive constant-curvature steps.

We thus obtain

$$\begin{aligned}\widehat{ABC} &= [\pi + \theta_{BC} - \theta_{AB}] \\ &= \left[\pi + \frac{[\theta_C - \theta_A]}{2}\right],\end{aligned}$$

which is true if and only if B belongs to a circular arc that links A to C, and such that its tangent in A has the orientation

$$\left[\theta - \frac{[\theta_C - \theta_A]}{2}\right].$$

This circular arc corresponds precisely to the one from (x_A, y_A, θ'_i) to $(x_C, y_C, [\theta'_i + (\theta_C - \theta_A)])$ followed by the constant-curvature step, which proves the property.

3.2.4 Backward Motion

The discretized trajectories that we have defined so far take the form of sequences of constant-curvature steps, each of which can either be a local turn, or a forward circular arc. Precisely, a constant-curvature step originating in the configuration (x_i, y_i, θ_i) and leading to the position (x_{i+1}, y_{i+1}) is a local turn if $(x_i, y_i) = (x_{i+1}, y_{i+1})$, and a forward circular arc otherwise. In the latter case, the constraint

$$-\frac{\pi}{2} < [\theta_i - \theta] < \frac{\pi}{2},$$

has to be satisfied, where $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$. Intuitively, this constraint expresses that the orientation of the robot at the beginning of the constant-speed step is consistent with a forward direction of motion.

We now generalize this discretized representation of trajectories by allowing backward constant-curvature steps, in addition to forward ones and local turns. By the same reasoning as in Section 3.1.3, a backward constant-curvature step can be understood in terms of a forward step performed by another robot oriented in the opposite direction. It follows that a backward constant-curvature step originating in the configuration (x_i, y_i, θ_i) and leading to the position (x_{i+1}, y_{i+1}) is characterized by

$$-\pi < [\theta_i - \theta] < -\frac{\pi}{2} \text{ or } \frac{\pi}{2} < [\theta_i - \theta] \leq \pi,$$

where $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$.

Note that the situation $[\theta_i - \theta] = \pm\frac{\pi}{2}$ is ambiguous and explicitly forbidden. In actual implementations relying on a non-exact representations of real numbers, one can require the value of $|[\theta_i - \theta]|$ to be significantly distinct from $\frac{\pi}{2}$ (recall that this value can always be reduced by refining the trajectory sampling).

Finally, just like forward steps, the orientation at the beginning of the circular arc is uniquely determined from the values x_i, y_i, x_{i+1}, y_{i+1} and $\delta_i = [\theta_{i+1} - \theta_i]$: we have

$$\theta_i' = \left[\pi + \theta - \frac{\delta_i}{2}\right],$$

where $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$.

3.2.5 Static Steps

A particular case of constant-curvature step that has not yet been discussed is one during which the robot does not move, which we call a *static step*. Such a step is thus characterized by identical initial and final configurations

$$(x_i, y_i, \theta_i) = (x_{i+1}, y_{i+1}, \theta_{i+1}).$$

There are several motivations for including static steps in trajectories. First, a static step may represent an interval in a trajectory when the locomotion has to pause during the time needed for performing some action, for instance picking up an object at a specific location. Second, in the case of a tricycle platform, static steps must sometimes be inserted in order to allow sufficient time for orienting the steering wheel. This happens, in particular, between local turns (where the absolute steering angle is equal to $\frac{\pi}{2}$) and forward or backward steps (where this angle is strictly less than $\frac{\pi}{2}$).

3.2.6 Resampling

With this solution, the robot configurations are specified at all times, and arbitrary samplings of trajectories are allowed. However, the orientation function $\theta(t)$ defined by a discretized trajectory, corresponding to the direction of the tangent vector at all points, is generally not continuous. In order to alleviate this drawback, the orientation and the position of the robot are treated separately. The position during a forward or a backward constant-curvature step i follows the one of the defined circular arc, but the orientation will be assumed to change linearly with respect to the travelled distance from θ_i to θ_{i+1}.

With this assumption, the orientation function $\theta(t)$ is continuous, but does not necessarily match the true orientation of the circular arcs. This is not necessarily problematic, for the discretization process was expected to introduce approximations, but it remains to assess whether this approximation is accurate enough.

First, for a particular forward or backward constant-curvature step i, the maximal error between the considered orientation and the one of the circular arc is

$$|[\theta_i - \theta_i']| = \left|\left[\frac{\theta_i + \theta_{i+1}}{2} - \theta\right]\right|$$

(where $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$) and can be arbitrarily bounded by choosing sufficiently small sampling steps.

Next, for the same constant-curvature step, we can compute an upper bound of the position error. If the curvature of the trajectory is assumed to change monotonously between κ_a at time t_i and κ_b at time t_{i+1} (which is a weak assumption, since it can always be enforced by choosing sufficiently small sampling steps), the trajectory lies between the two circular arcs of curvature κ_a and κ_b leading from (x_i, y_i) to (x_{i+1}, y_{i+1}) (if one of this two circular arcs does not exist, then the sampling is too coarse and must be refined). By symmetry, an upper bound of the error is given by the distance between the midpoints of these two circular arcs, that is,

$$\left|\left(\frac{1}{\kappa_a} - \frac{1}{\kappa_b}\right) - \left(\sqrt{\left(\frac{1}{\kappa_a}\right)^2 - \frac{{\lambda_i}^2}{4}} - \sqrt{\left(\frac{1}{\kappa_b}\right)^2 - \frac{{\lambda_i}^2}{4}}\right)\right|,$$

where

$$\lambda_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}.$$

This bound can also be arbitrarily reduced by choosing sufficiently small sampling steps.

With this discretization procedure, arbitrary resampling between two sample points can be computed by similar developments as for odometry (see Section 2.2.1), and the configuration of a robot that has travelled a fraction $\alpha \in [0,\ 1]$ of the distance from the point i to the point $i+1$ is given by:

- if $\kappa_i = 0$ ($\theta_i = \theta_{i+1}$):

$$\begin{aligned} \theta_{i+\alpha} &= \theta_i, \\ x_{i+\alpha} &= x_i - ds \sin\theta_i, \\ y_{i+\alpha} &= y_i + ds \cos\theta_i, \end{aligned}$$

  where

$$ds = \alpha \cdot \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}.$$

- if $\kappa_i \neq 0$:

$$\begin{aligned} \theta_{i+\alpha} &= [\theta_i + d\theta], \\ x_{i+\alpha} &= x_i - a_1 \sin d\theta + a_2 \cos d\theta - a_2, \\ y_{i+\alpha} &= y_i + a_1 \cos d\theta + a_2 \sin d\theta - a_1, \end{aligned}$$

  where

$$\begin{aligned} d\theta &= \alpha \cdot [\theta_{i+1} - \theta_i], \\ a_1 &= \frac{\sin\theta_i}{\kappa_i}, \\ a_2 &= \frac{\cos\theta_i}{\kappa_i}. \end{aligned}$$

3.2.7 Data Structure

We are now ready to define the data structure that will be used for representing symbolically trajectories. It is worth recalling that, at this point, we are only interested in describing the spatial properties of trajectories (i.e., the corresponding path); temporal aspects such as the speed at which a path is to be followed will be addressed in Chapter 4 (leading to an extended symbolic representation).

Definition 3.2.2. *The* representation *of a path is a pair* (m, σ)*, where*

- $m \in \mathbb{N}$ *is a* number of steps*, and*
- σ *is a function that maps every* index $i \in [0, m]$ *onto a* sampled configuration $\sigma(i) = (x_i, y_i, \theta_i, \kappa_i)$*, where*

- $(x_i, y_i) \in \mathbb{R}^2$ *is a* position,
- $\theta_i \in (-\pi, \pi]$ *is an* orientation, *and*
- $\kappa_i \in \mathbb{R} \cup \{-\infty, +\infty\}$ *is a* curvature.

As discussed in Sections 3.2.3 and 3.2.4, representations of paths have to satisfy some integrity constraints in order to be valid.

Definition 3.2.3. *A path representation* (m, σ) *is* well-formed *iff for all* $i \in [0, m-1]$*, one of the following criteria is satisfied:*

- $(x_i, y_i) = (x_{i+1}, y_{i+1})$ *and* $\theta_{i+1} \neq \theta_i$ (local turn).

  In this case, one must have

  $$\kappa_i \in \{-\infty, +\infty\}.$$

- $(x_i, y_i) = (x_{i+1}, y_{i+1})$ *and* $\theta_{i+1} = \theta_i$ (static step).

  In this case, the value of κ_i *is undefined, and is by convention chosen as*

  $$\kappa_i = \begin{cases} \kappa_{i-1} & \textit{if } i > 0, \\ 0 & \textit{otherwise.} \end{cases}$$

- $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ *and* $-\frac{\pi}{2} < [\theta_i - \theta] < \frac{\pi}{2}$*, where* $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$ (forward step).

  In this case, one must have

  $$\kappa_i = \frac{2}{\lambda_i} \sin \frac{\delta_i}{2},$$

  where

  $$\begin{aligned} \lambda_i &= \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}, \\ \delta_i &= [\theta_{i+1} - \theta_i]. \end{aligned}$$

- $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ *and either* $-\pi < [\theta_i - \theta] < \frac{\pi}{2}$ *or* $\frac{\pi}{2} < [\theta_i - \theta] \leq \pi$, *where* θ *is as before* (backward step).

  Like in the previous case, one must have

  $$\kappa_i = \frac{2}{\lambda_i} \sin \frac{\delta_i}{2},$$

  where λ_i *and* δ_i *are as before.*

In order to simplify reasoning about path steps, we augment the path representation with a function $\nu : [0, m-1] \to \{L, R, F, B, S\}$ that associates each index with the *nature* of the corresponding step. Formally, we have for every $i \in [0, m-1]$:

- $\nu_i = L$ if $\kappa_i = +\infty$, i.e., the $(i+1)$-th step is a local turn to the left.

- $\nu_i = R$ if $\kappa_i = -\infty$, i.e., the $(i+1)$-th step is a local turn to the right.

- $\nu_i = F$ if $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ and $-\frac{\pi}{2} < [\theta_i - \theta] < \frac{\pi}{2}$, where $\theta = \arctan(x_i - x_{i+1},\ y_{i+1} - y_i)$, i.e., the $(i+1)$-th step is a forward step.

- $\nu_i = B$ if $(x_i, y_i) \neq (x_{i+1}, y_{i+1})$ and either $-\pi < [\theta_i - \theta] < \frac{\pi}{2}$ or $\frac{\pi}{2} < [\theta_i - \theta] \leq \pi$, where θ is as before, i.e., the $(i+1)$-th step is a backward step.

- $\nu_i = S$ if $(x_i, y_i) = (x_{i+1}, y_{i+1})$ and $\theta_i = \theta_{i+1}$, i.e., the $(i+1)$-th step is a static step.

It then becomes straightforward to describe the semantics associated to a well-formed path representation (m, σ). For each $i \in [0, m-1]$, the $(i+1)$-th step of the represented path originates from the configuration (x_i, y_i, θ_i), and

- in the case of a local turn ($\nu_i \in \{L, R\}$), changes the orientation of the robot without modifying its position (x_i, y_i), ending in the orientation θ_{i+1}. The turn is performed to the left (resp. right) if $\kappa_i = +\infty$ (resp. $\kappa_i = -\infty$).

- in the case of a forward or backward step ($\nu_i \in \{F, B\}$), follows a circular arc of curvature κ_i from (x_i, y_i) to (x_{i+1}, y_{i+1}) in the corresponding direction of motion, increasing its orientation by the angle $\delta_i = [\theta_{i+1} - \theta_i]$.

As discussed in Section 3.1.3, the sign of κ_i provides the turn direction. The distance $|s_i|$ travelled by the reference point of the robot during the step is given by

$$|s_i| = \begin{cases} \left|\dfrac{\delta_i}{\kappa_i}\right| & \text{if } \kappa_i \neq 0 \\ \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} & \text{if } \kappa_i = 0. \end{cases} \tag{3.32}$$

Following the convention introduced in Section 3.1.3, we consider that we have $s_i > 0$ if $v_i = F$, and $s_i < 0$ if $v_i = B$.

- in the case of a static step ($v_i = S$), stays at the configuration (x_i, y_i, θ_i).

CHAPTER 4

Speed Profile Computation

This chapter first introduces the notions of speed profile and of time-optimal speed profile for the path representation introduced in Chapter 3. It then presents an algorithm for computing a time-optimal speed profile that minimizes the total time needed by a robot for following such path representations while being consistent with various physical constraints of the robot. This algorithm has a linear computational cost (in the number of path steps) and is faster and able to take into account a broader class of physical constraints than other solutions, such as imposing tighter speed bounds in the vicinity of some obstacles, or expressing the admissible acceleration of a wheel as a function of its speed. Finally, the chapter briefly introduces the notions of trajectory resampling and of braking trajectories.

In Chapter 3, we have introduced an original data structure for discretizing paths to be followed by a nonholonomic mobile robot, which represents such paths by a finite sequence of robot configurations visited along the paths. We now address the problem of computing a *speed profile* for such paths, i.e., determining for each discrete step the speed at which it can be performed. For our intended applications (such as robotics competitions like Eurobot), it is crucial to move rapidly and since energy efficiency is not an issue, the goal is then to generate the speed profile that reaches as quickly as possible the destination of the path, while satisfying at all times physical constraints of the robot such as various speed or acceleration bounds (in order for the speed profile to be physically realizable).

4.1 Speed Profiles

4.1.1 Principles

In Section 3.2.7, discretized paths have been defined as sequences of elementary moves (local turns as well as forward, backward, and static steps) performed with constant curvature. A natural approach would then be to specify for each non-static step of a path the speed at which it shall be followed. This speed can be defined as the speed v of the reference point of the robot in the case of forward and backward steps, and as the angular speed $\dot{\theta}$ of its reference orientation for local turns. A simpler alternative, that has the advantage of handling uniformly both cases, consists in specifying for each step the speeds of the left and right wheels of the robot (in a way that is consistent with the properties of the step, as discussed in Section 3.1.2).

One could then consider that those speeds are constant during the time that the step is performed. This solution has the drawback of introducing speed discontinuities between successive steps. This is not necessarily problematic, since every robot is only practically able to update the reference speeds of its wheels at discrete time instants (usually separated by a predetermined *control period*). Nevertheless, the presence of speed discontinuities requires that the time needed for performing an elementary step does not exceed that control period, if moves such as driving straight segments with constant acceleration have to be described accurately.

Unfortunately, that a path sampling is sufficiently refined for satisfying this criterion can not always be inferred when the path is generated, since the speed at which it will be followed has not yet been computed at that time. Another problem is the difficulty of specifying precisely boundary conditions on the speed of the robot; for instance, starting a path with an initial speed equal to zero does not mean that the first path step has to be performed with zero speed.

In order to alleviate these problems, we refine our definition of non-static steps and consider that, in addition of being performed with *constant curvature*, each of them is carried out with *constant acceleration* of the wheels. According to the results of Section 3.1.2, the constant curvature requirement amounts only to imposing that the ratios between the linear speeds of the left, right, and steering wheels (if any), are constant throughout the step, and that the steering angle stays constant as well in the case of a tricycle platform. The constant acceleration requirement means that the speed of each wheel changes linearly with time when the step is performed, instead of staying constant within this step. In other words, the speed of the wheels is linearly interpolated from the speeds specified at the beginning and the end of each step.

A technical difficulty is the fact that the curvatures of two successive path steps generally differ, which implies that wheel speeds specified at the beginning of a step (consistent with the curvature of this step) do generally not comply with the geometric properties of the previous one. This means that, if wheel speeds are only provided at the boundaries between path steps, then continuity of linear speed can generally not be ensured simultaneously for both wheels.

We solve this problem in the following way. First, for the sake of simplicity, and in order to be able to easily chain paths together, we impose this curvature to be zero at the origin and endpoint of all paths (except, of course, for those starting or ending with a local-turn).

Second, one can recall from Chapter 3.2.7, that each step i of a well-formed path representation is associated with a value $\nu_i \in \{L, R, F, B, S\}$, that represents the *nature* of the corresponding move (i.e., respectively, a local turn to the left, a local turn to the right, a forward step, a backward step or a static step). For the same motivation of simplicity, at the boundary between two successive path steps that have different natures (i.e., different values of ν_i), the speed of both wheels will be required to be equal to zero.

This requirement is legitimate, since usually a robot must already come to a complete stop to change its direction of motion. The only cases where a robot does not need to stop, are for paths switching from a local-turn to forward or backward steps while decreasing (in absolute value) its curvature without discontinuities, or the reverse situation (i.e., switching from forward or backward steps to a local turn while increasing its curvature without discontinuities). Such spiral shaped paths are not really useful for our intended applications, and will then be explicitly forbidden. As a consequence from these two requirements, each maximal sequence of forward or backward steps of a path must start and end with zero curvature.

Third, between successive steps sharing the same nature, continuity of speed will only be required for the reference point of the robot in the case of forward and backward steps. (For local turns, continuity of speed can, naturally, be achieved for all wheels.)

Finally, we have to consider the static steps. As discussed in Section 3.2.5, those are mainly introduced for two reasons:

- describing periods during which locomotion is paused in order to perform other actions (*pausing static steps*);
- allowing sufficient time for orienting the steering wheel between successive steps with significantly different curvatures, in the case of a tricycle platform (*steering*

static steps).

In a speed profile, a static step can be handled by simply specifying its duration. If the goal is to compute the fastest speed profile for a given path, then only the steering static steps have to be considered (since the pausing ones systematically will be assigned a zero duration).

We can add the following definitions to the ones from Chapter 3.2.7:

Definition 4.1.1. *A path representation* (m, σ) *is* acceptable *iff it is well-formed, and*

- *if* $v_0 \in \{F, B\}$, *we have* $\kappa_0 = 0$,
- *if* $v_{m-1} \in \{F, B\}$, *we have* $\kappa_{m-1} = \kappa_m = 0$,
- *for all* $i \in [1, m-2]$ *such that* $v_i \in \{F, B\} \neq v_{i+1}$ *or* $v_i \in \{F, B\} \neq v_{i-1}$, *we have* $\kappa_i = 0$.

Definition 4.1.2. *A path representation* (m, σ) *is* steerable *iff it is acceptable, and for all* $i \in [0, m-2]$ *such that* $v_i \neq v_{i+1}$, *we have*

$$v_i = S, \text{ or } v_{i+1} = S.$$

In other words, a steerable path is one into which a static step has been inserted between two consecutive steps that do not share the same nature. Indeed, at this point, as a result from Definition 4.1.1, the robot must come to a complete stop.

In the following, for simplicity sake, we will consider only paths that are steerable, even with a differential drive platform (in which case in the absence of steering constraints, the duration of all static steps will end up being equal to 0).

Last but not least, in order to be able to accurately compute constraints, knowing the curvature of each steps will not be sufficient. Indeed, if we consider that this curvature changes linearly from κ_i at the beginning of the $(i+1)$-th step, to κ_{i+1} at the end, results will strongly depend from the length of the step (intuitively, κ_i is a measure of the mean curvature during this step). A better choice, that will be less influenced by the length of the discretization step, is to compute the *instantaneous curvature* $\tilde{\kappa}_i$ for each configuration i.

There are two ways of extracting $\tilde{\kappa}_i$ from the path representation. First, if the exact path is known, it can be computed without approximation during the discretization process. Another approach is to accurately approximate $\tilde{\kappa}_i$ from the discretized path as follows:

- $\tilde{\kappa}_0 = 0$

- $\tilde{\kappa}_m = 0$
- For every $i \in [1, m-1]$:
  - if $\kappa_i = 0$ or $\kappa_{i-1} = 0$ (beginning or end of a straight line step).
  $$\tilde{\kappa}_i = 0,$$
  - otherwise
  $$\tilde{\kappa}_i = \kappa_{i-1} + \frac{\kappa_i - \kappa_{i-1}}{|s_i| + |s_{i-1}|} \cdot |s_{i-1}|,$$
  where $|s_i|$ is the distance travelled by the robot during the $(i+1)$-th step (cf. Equation 3.32), and assuming that the curvature change linearly with the travelled distance, from κ_{i-1} to κ_i, between the middle points of the two steps.

4.1.2 Formalization

Formally, a speed profile for a differential drive or a tricycle robot is defined as follows:

Definition 4.1.3. *Given a steerable path representation (m, σ), a* speed profile *associated to this path is a pair (ρ, μ), where:*

- *The function $\rho : [0, m] \to \mathbb{R}^2$ maps every path index $i \in [0, m]$ onto a pair $\rho(i) = (v_{L_i}, v_{R_i})$ of speeds for the left and right wheels of the robot at that index.*
- *The function $\mu : \{i \in [0, m-1] \mid \nu_i = S\} \to \mathbb{R}_{\geq 0}$ maps every static step i onto the duration $\mu(i)$ of that step.*

In order to be valid, a speed profile has to satisfy some integrity constraints, expressed by the following definition.

Definition 4.1.4. *A speed profile (ρ, μ) associated to a path representation (m, σ) is* well-formed *iff for every $i \in [0, m-1]$:*

- *If $\nu_i = R$ (*right local turn*), then*
$$0 \leq v_{L_i} = -v_{R_i}.$$
- *If $\nu_i = L$ (*left local turn*), then*
$$v_{L_i} = -v_{R_i} \leq 0.$$

- *If* $v_i \in \{F, B\}$ *(* forward or backward step*), then*

  $$v_{L_i}\left(1 + \frac{e\tilde{\kappa}_i}{2}\right) = v_{R_i}\left(1 - \frac{e\tilde{\kappa}_i}{2}\right).$$

  where, as before, e denotes the axle width (i.e., the distance between the contact patches of the left and right wheels).

- *If* $v_i = S$ *(* static step*), then*

  $$v_{L_i} = v_{R_i} = 0$$

  $$v_{L_{i+1}} = v_{R_{i+1}} = 0.$$

4.1.3 Semantics

We are now ready to describe precisely the semantics of a well-formed speed profile (ρ, μ) associated to the representation (m, σ) of a path. For each $i \in [0, m-1]$, we have:

- *If* $v_i \in \{L, R\}$. Then, during the step, the left and right wheels move at the same speed but in opposite directions. Let us focus on the left wheel, and denote by $|s_{L_i}| > 0$ the (absolute) distance driven by this wheel during the $(i+1)$-th step, i.e., the length of the circle arc followed by the wheel when the robot moves from $\sigma(i)$ to $\sigma(i+1)$.

  We consider that the step is performed by accelerating linearly (with time) the left wheel from the speed v_{L_i} to the speed $v_{L_{i+1}}$. In other words, at the beginning of the step, the distance driven by the left wheel is zero and its speed is v_{L_i}. At the end of the step, the distance driven by the wheel is $|s_{L_i}|$ and its speed is $v_{L_{i+1}}$. It follows that the amount of time Δt_i needed for driving the step is given by

  $$\Delta t_i = \frac{2\,|s_{L_i}|}{|v_{L_i} + v_{L_{i+1}}|}.$$

  In between, after a time Δt such that $0 \leq \Delta t \leq \Delta t_i$, the speed v_L of the wheel is given by

  $$v_L = v_{L_i} + (v_{L_{i+1}} - v_{L_i})\frac{\Delta t}{\Delta t_i},$$

  and the distance $|s_L|$ driven by the wheel from the beginning of the step is equal to

  $$|s_L| = \left| v_{L_i} t + (v_{L_{i+1}} - v_{L_i})\frac{\Delta t^2}{2\Delta t_i} \right|.$$

From these results, it is immediate to derive the speed v_R of the right wheel after time Δ_t, and the distance $|s_R|$ driven by that wheel. We have

$$\begin{aligned} v_R &= -v_L \\ |s_R| &= |s_L|. \end{aligned}$$

- *If* $v_i \in \{F, B\}$. Then, since the curvatures of successive steps generally differ, continuity of speed can not be simultaneously achieved for both left and right wheels. We focus on the speed v of the reference point of the robot, and apply the same reasoning as in the case of local turns, i.e., we consider that the reference point accelerates linearly from its speed v_i at $\sigma(i)$ to its speed v_{i+1} at $\sigma(i+1)$. Let $|s_i|$ denote the distance driven by the reference point during the $(i+1)$-th step.

  We obtain

$$\begin{aligned} \Delta t_i &= \frac{2|s_i|}{|v_i + v_{i+1}|} \\ v &= v_i + (v_{i+1} - v_i)\frac{\Delta t}{\Delta t_i} \\ |s| &= \left| v_i t + (v_{i+1} - v_i)\frac{\Delta t^2}{2\Delta t_i} \right|, \end{aligned}$$

  where v is the speed reached after spending the time Δt into the step, and $|s|$ is the distance driven from the beginning of the step.

  Once the speed of the reference point has been established, the speed of each wheel can easily be computed using the formulas obtained in Section 3.1.2. Note that, with this scheme, the wheel speeds are generally discontinuous at the end of a step. In situations where a continuous approximation of the wheel speeds is needed (for instance, in order to reason about tangential accelerations), we will consider that the speed of each wheel increases with constant acceleration during the step. This approximation is not exact, since the curvature of the step then becomes inconsistent with the speed of the wheels. But, as discussed in Section 3.2.6, in order to keep the position error under a certain bound, the discretization process produces successive steps with curvatures that are sufficiently close to each other, and this approximation is usually quite precise.

- *If* $v_i = S$. Then, the robot keeps the configuration

$$(x_i, y_i, \theta_i)$$

during the duration

$$\Delta t_i = \mu(i).$$

4.1.4 Data Structure

We are now ready to describe the data structure that will be used for representing the properties of trajectories.

Definition 4.1.5. *The* representation *of a trajectory is a tuple* (m, σ, ρ, μ) *such that* (m, σ) *is a path representation, and* (ρ, μ) *is a corresponding speed profile. The representation* (m, σ, ρ, μ) *is well-formed is* (m, σ) *is steerable and* (ρ, μ) *is well-formed.*

In other words, a trajectory representation (m, σ, ρ, μ) maps every index $i \in [0, m]$ onto:

- a *position* $(x_i, y_i) \in \mathbb{R}$,
- an *orientation* $\theta_i \in (-\pi, \pi]$,
- a *curvature* $\kappa_i \in \mathbb{R} \cup \{-\infty, +\infty\}$,
- a *left-wheel speed* $v_{L_i} \in \mathbb{R}$,
- a *right-wheel speed* $v_{R_i} \in \mathbb{R}$,
- a *step duration* $\Delta t_i \in \mathbb{R}_{\geq 0}$ (for ($i \in [0, m-1]$).

In practice, in order to speed up some computations, one can also store in the trajectory representation additional derived data for each step, such as:

- its *nature* $v_i \in \{L, R, F, B, S\}$ (for $i \in [0, m-1]$),
- the *distance travelled by the reference point* $|s_i| \in \mathbb{R}_{\geq 0}$ (for $i \in [0, m-1]$),
- the *instantaneous curvature* $\tilde{\kappa}_i \in \mathbb{R} \cup \{-\infty, +\infty\}$,

- its *time instant*

  $$t_i = \sum_{0 \le j < i} \Delta t_j$$

  (for $i \in [0, m]$),

- the *steering-wheel orientation* $\theta_{S_i} \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ that corresponds to $\tilde{\kappa}_i$ (for $i \in [0, m]$),

- the *steering-wheel speed* $v_{S_i} \in \mathbb{R}$ (for $i \in [0, m]$).

Note that, in the case of a tricycle platform, the semantics introduced in Section 4.1.3 does not describe the behavior of the steering wheel during static steps, since it does not influence the movement of the robot. In practice, we will consider that the steering wheel moves with a constant angular rate during static steps.

4.1.5 Problem Statement

Let us now address the problem of computing a time-optimal speed profile (i.e., the speed profile that reaches as quickly as possible the destination) for a given path representation (m, σ). We first introduce temporal constraints on the initial and final configurations of such a trajectory:

- The initial speed of the robot, i.e., the speed v of its reference point at the origin $\sigma(0)$ of the trajectory, is given by v_0. Note that, if the first step of the trajectory is a local turn or a static step, then one must necessarily have $v_0 = 0$ in order for a speed profile to exist. If, on the other hand, the first step is a forward (resp. backward) step, then one must have $v_0 \geq 0$ (resp. $v_0 \leq 0$).

  The motivation behind imposing an initial speed is that trajectories are meant to be followed one after another: the initial configuration of a trajectory, including the speed of the robot, has to match the final configuration of the previous one, or the actual physical configuration of the robot.

- The initial angular speed of the robot, i.e., the angular speed $\dot{\theta}$ at $\sigma(0)$ is, for simplicity sake, required to be equal to 0. Note that, in order to be able to follow sequences of trajectories, a similar constraint will have to be imposed on final angular speeds. Moreover, if the first step of the trajectory (m, σ) is a forward or backward step, then this requirement implies $\kappa_0 = 0$ (as already required by Definition 4.1.1).

- The maximum final speed of the robot, i.e., the maximum (in absolute value) speed v of its reference point at the endpoint $\sigma(m)$ of the trajectory, is given by v_{fm}. As for the initial speed, if the last step of the trajectory is a local turn or a static step, then one must have $v_{fm} = 0$. In the case of a forward (resp. backward) step, the constraint becomes $v_{fm} \geq 0$ (resp. $v_{fm} \leq 0$).

  Note that, unlike the case of the initial speed, only an upper bound is imposed on the final speed of the robot.

- The final angular speed of the robot, i.e., the angular speed $\dot{\theta}$ at $\sigma(m)$, is required to be equal to 0, as already discussed in the context of the initial angular speed. If the last step of the trajectory is a forward or backward one, then this constraint implies $\kappa_m = 0$ (also already required by Definition 4.1.1).

In addition, we define some number of physical constraints that need to be satisfied at all times, such as lower and upper bounds on the velocity or acceleration of individual wheels, the speed, angular speed, tangential and radial accelerations measured at the center of mass or some other reference points, on the rate of variation of the steering angle for a tricycle robot, … Some of those constraints may be context-sensitive, such as imposing tighter speed bounds in the vicinity of some obstacles, or expressing the admissible angular velocity of the steering wheel as a function of the robot speed.

In this work, we consider 16 distinct physical constraints, which have proved to be the more relevant ones for the Eurobot case study, but other constraints can also be considered, as long as they comply with the requirements that will be expressed in Section 4.2.3.

First, we specify constraints on the speed and acceleration of the reference point of the robot, that must hold throughout the trajectory:

- Functions $v_{min} : [0, m] \to \mathbb{R}_{\leq 0}$ and $v_{max} : [0, m] \to \mathbb{R}_{\geq 0}$ associate to each index $i \in [0, m]$ of the path the minimum and maximum speeds $v_{min}(i)$ and $v_{max}(i)$ of the reference point. Note that, in order for a speed profile to exist, one must have $v_{min}(0) \leq v_0 \leq v_{max}(0)$.

  The reason why the minimum speed is not systematically chosen to be the opposite of the maximum one is to be able to cope with robots that have asymmetrical locomotion capabilities. Furthermore, the minimum and maximum speeds are expressed as functions instead of constant bounds in order to be able to specify more stringent constraints when the path brings the robot close to obstacles or opponents, and to relax them in open areas.

- A function $\dot{\theta}_{max} : [0, m] \rightarrow \mathbb{R}_{\geq 0}$ provides the maximum (absolute) angular speed $\dot{\theta}_{max}(i)$ for each index.

- Lower and upper bounds $a_{Tmin} \in \mathbb{R}_{<0}$ and $a_{Tmax} \in \mathbb{R}_{>0}$ are imposed on the tangential acceleration a_{T_i}, for every $i \in [0, m-1]$. Similarly, lower and upper bounds $a_{Cmin} \in \mathbb{R}_{<0}$ and $a_{Cmax} \in \mathbb{R}_{>0}$ are imposed on the radial acceleration a_{C_i} for all i. These bounds make it possible to reason about the stability of the robot with respect to its support polygon, for instance in order to insure that it does not tip over in tight turns.

Next, we define constraints related to the physical limitations of the locomotion mechanisms. First, the reachable speed of each driving wheel is bounded. Then, in order for such a wheel to develop traction, the force exchanged between the wheel and the ground must stay within specified bounds. This translates into constraints over the tangential acceleration at that wheel (assuming that when the wheel rotates, it displaces a constant mass).

Additionally, one must ensure that the torque that has to be developed at a wheel can be produced by the corresponding locomotion motor, which again amounts to imposing bounds on the tangential acceleration at the wheel. Since the torque developed by a motor may depend on its rotational speed, we choose to express those bounds as functions over the linear speed at the wheel. Furthermore, for the steering wheel of a tricycle platform, we impose bounds on the rate of variation of the steering angle.

We obtain the following constraints:

- In the case of a differential drive platform, lower and upper bounds $v_{LRmin} \in \mathbb{R}_{\leq 0}$ and $v_{LRmax} \in \mathbb{R}_{\geq 0}$ are imposed on the linear speeds v_{L_i} and v_{R_i} of the left and right wheels.

  In addition, a function $a_{LRmin} : [v_{LRmin}, v_{LRmax}] \rightarrow \mathbb{R}_{\leq 0}$ specifies for each possible value of the speed v_{L_i} or v_{R_i} the minimum value of the corresponding tangential acceleration a_{L_i} or a_{R_i}, for every $i \in [0, m-1]$. Similarly, a function $a_{LRmax} : [v_{LRmin}, v_{LRmax}] \rightarrow \mathbb{R}_{\geq 0}$ defines the maximum tangential acceleration with respect to the linear speed of the left or right wheel. Note that, for simplicity sake, we assume the constraints discussed here to by symmetrical over the left and right wheels.

- For a tricycle platform, lower and upper bounds $v_{Smin} \in \mathbb{R}_{<0}$ and $v_{Smax} \in \mathbb{R}_{>0}$ are imposed on the linear speed v_{S_i} of the steering wheel. An upper bound $\dot{\theta}_{Smax} \in \mathbb{R}_{>0}$ is imposed on the (absolute) rate of variation $\dot{\theta}_{S_i}$ of the steering angle, for all $i \in [0, m-1]$. In addition, two functions $a_{Smin} : [v_{Smin}, v_{Smax}] \rightarrow \mathbb{R}_{<0}$ and a_{Smax} :

$[v_{Smin}, v_{Smax}] \to \mathbb{R}_{>0}$ specify respectively the minimum and the maximum values of the tangential acceleration a_{S_i} at the steering wheel, with respect to the linear speed v_{S_i} at that wheel, for every $i \in [0, m-1]$.

We are now ready to precisely define the time-optimal speed profile computation problem. Given a discretized path representation (m, σ), parameters v_0, v_{fm}, v_{LRmin}, v_{LRmax}, v_{Smin}, v_{Smax}, a_{Tmin}, a_{Tmax} a_{Cmin}, a_{Cmax} and $\dot{\theta}_{Smax}$, and functions v_{min}, v_{max}, $\dot{\theta}_{max}$, a_{LRmin}, a_{LRmax}, a_{Smin} and a_{Smax}, the time-optimal speed profile computation problem consists in computing a speed profile that makes it possible to follow this path as fast as possible, while satisfying all the constraints that have been specified.

4.2 Solution

From Definitions 4.1.2 and 4.1.4, we know that the speeds v_{L_i} and v_{R_i} of the robot wheels must be equal to 0 between steps of different nature, i.e., when $i > 0$ and $\nu_i \neq \nu_{i-1}$. It follows that the problem of computing a time-optimal speed profile can be solved separately for each maximal sequence of left-turn, right-turn, forward, backward, and static steps composing the path (which we call a *segment* of this path).

We have the following boundary constraints:

- for the first segment, the initial values $v_{L_0} = v_{R_0}$ are imposed by the initial speed v_0;
- for the last segment, the final values $v_{L_m} = v_{R_m}$ are constrained by the final speed v_{fm};
- for any index i located at the boundary between two successive segments, i.e., such that $i > 0$ and $\nu_i \neq \nu_{i-1}$, one has $v_{L_i} = v_{R_i} = 0$.

From now on, we will thus address w.l.o.g. the problem of computing a time-optimal speed profile for a path composed of a single segment, i.e., one in which all the steps share the same nature $\nu \in \{L, R, F, B, S\}$.

The main strategy employed to solve our speed profile problem is identical for every kind of segment, but details differ. Those are fully developed in Appendix A for the 16 distinct physical constraints that we consider for the Eurobot application.

Solving the time-optimal speed profile computation problem for a well-formed non-static path segment representation (m, σ) amounts to computing for each $i \in [0, m]$ an optimal value of the speeds v_{L_i} and v_{R_i}. These speeds can not be chosen independently from each other; indeed, it follows from Definition 4.1.4 that their ratio is fixed, and depends only on the nature of the segment as well as on the instantaneous curvature of the path.

In fact, one can recall from Chapter 3.1.2 that, for differential-drive, tricycle or car-like robots, all the speeds of interest (i.e., measured at various locations on the robot) can be expressed as functions of a single parameter and the geometry of the path, the optimal value of which has to be computed. A natural choice would be to define this parameter as the speed of the robot measured at its center. This solution turns out to be problematic for reasoning about parts of paths where the absolute curvature is high, which intuitively corresponds to rotations of the robot around its center. In such a case, even though the robot is in motion, its center moves only slowly, or not at all. A second choice for the parameter could be the speed of the fastest wheel, but this approach leads to complex developments when this fastest wheel switches sides.

We choose instead to express all speeds of interest in terms of a parameter z_i that we call the *velocity* at the current index i of the path, defined as the quadratic mean of the respective speeds v_{L_i} and v_{R_i} of the left and right wheels:

$$z_i = \sqrt{\frac{v_{L_i}^2 + v_{R_i}^2}{2}}. \tag{4.1}$$

This parameter has the advantage of being always positive, and nonzero whenever the robot is not stationary. Knowing the geometry of the robot, one can easily compute the speeds at its center, center of mass, individual wheels, or other locations of interest, from the velocity z_i and the instantaneous curvature $\tilde{\kappa}_i$ at the current path index. When the curvature is zero, such as at the extremities of paths, all those speeds become equal to z_i.

For static path segments, the problem is different and consists in computing their duration. This problem will be addressed separately from the computation of the optimal values of the parameters z_i (see Section 4.2.6).

4.2.1 Problem Reduction

The time-optimal speed profile computation problem for non-static path segment can thus be restated as follows. The goal is to compute for all $i \in [0, m]$ the maximum value of $z_i \geq 0$ satisfying some given set of constraints.

Schematically, these constraints can be classified as follows:

- The initial value $z_0 = z_{init}$ is imposed by the initial speed of the robot.

- A first group of constraints that translate into an upper bound on the velocity z_i at a path index i. We call them *local state constraints.* Those are, for example, constraints imposed on the speed of the robot, its angular speed or the speed of its wheels.

Let us illustrate this situation with a simple example. Assume that the robot moves in the forward direction and that the speeds of its right and left wheels (v_{R_i} and v_{L_i}) are constrained to belong to the interval $[-v_{imax}, v_{imax}]$ at the path index i, with $v_{imax} > 0$.

From Equations 3.16 and 3.17, we can express the wheel speeds in term of z_i.
For each $i \in [0, m]$, we have

$$v_{L_i}\left(1 + \frac{e\tilde{\kappa}_i}{2}\right) = v_{R_i}\left(1 - \frac{e\tilde{\kappa}_i}{2}\right).$$

where e denotes, as previously, the axle width of the robot (i.e., the distance between the left and right wheels contact patches).

By defining

$$\omega_i = \arctan\left(1 + \frac{e\tilde{\kappa}_i}{2}, 1 - \frac{e\tilde{\kappa}_i}{2}\right), \tag{4.2}$$

we get

$$\begin{aligned} v_{L_i} &= z_i\sqrt{2}\cos\omega_i, \\ v_{R_i} &= z_i\sqrt{2}\sin\omega_i. \end{aligned}$$

Note that, from Equation 2.5 we have

$$v_i = \frac{v_{L_i} + v_{R_i}}{2} \geq 0$$

and thus

$$\sin\omega_i + \cos\omega_i > 0,$$

since we assume that the robot is moving forward.

As a consequence, we have

$$-\frac{\pi}{4} < \omega_i < \frac{3\pi}{4},$$

for every $i \in [0, m]$.

The relations between v_{Ri}, v_{Li}, z_i and ω_i are illustrated in Figure 4.1.

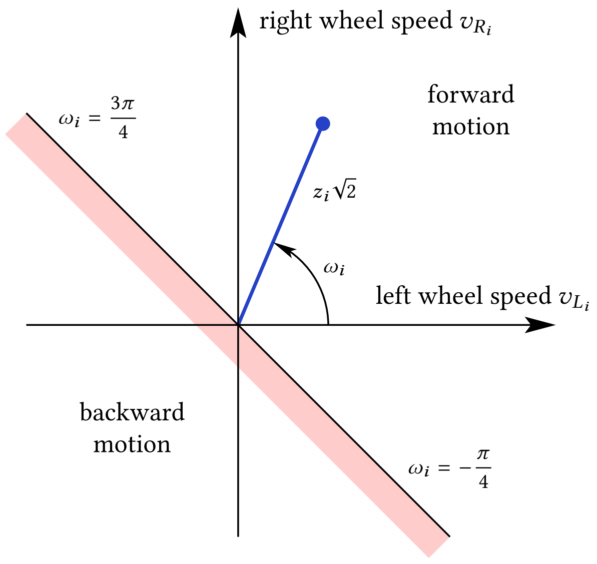

right wheel speed v_{R_i}
forward motion
$\omega_i = \frac{3\pi}{4}$
$z_i\sqrt{2}$
ω_i
left wheel speed v_{L_i}
backward motion
$\omega_i = -\frac{\pi}{4}$

Figure 4.1 – Relations between speeds.

Our constraint then gives out the upper bound

$$z_i \leq \frac{v_{i\,max}}{\sqrt{2}\cos\omega_i} \quad \text{if } -\frac{\pi}{4} < \omega_i \leq \frac{\pi}{4},$$
$$z_i \leq \frac{v_{i\,max}}{\sqrt{2}\sin\omega_i} \quad \text{if } \frac{\pi}{4} < \omega_i < \frac{3\pi}{4}.$$

Parameters v_{fm}, v_{LRmin}, v_{LRmax}, v_{Smin}, v_{Smax}, a_{Cmin}, a_{Cmax}, $\dot{\theta}_{Smax}$, and the functions v_{min}, v_{max} and $\dot{\theta}_{max}$ all translate into upper bounds on the velocity z_i, and if, z_{max_i} is defined as the smallest of these upper bounds at the path index i, the local state constraints can be resumed as

$$z_i \leq z_{max_i},$$

for all $i \in [0, m]$.

Of course, the initial value z_{init} must satisfy

$$z_{init} \leq z_{max_0}.$$

Note that, all the developments related to these parameters and functions are provided (for every kind of segment) in Appendix A.

- A second group of constraints, that we call *transition constraints*, contains those that jointly involve the velocity at successive path indices (e.g., constraints that involve the velocities z_i and z_{i+1} at two successive indices i and $i+1$ of the path). This group mainly includes constraints expressed in terms of acceleration, like constraints imposed on the tangential acceleration of the robot or the acceleration of its wheels.

Let us give an example. If, as explained in Section 4.1.1, we assume constant-acceleration motion during a step, then the amount of time Δt_i needed for travelling the distance s_i of the $(i+1)$-th step is given by

$$\Delta t_i = \frac{2s_i}{v_i + v_{i+1}}.$$

The tangential acceleration a_{T_i} of the reference point of the robot during this step can thus be estimated, from Equation 3.12, as

$$\begin{aligned} a_{T_i} &= \frac{v_{i+1} - v_i}{\Delta t_i} \\ &= \frac{v_{i+1}^2 - v_i^2}{2s_i}. \end{aligned}$$

Since we have

$$v_i = \frac{v_{Ri} + v_{Li}}{2}$$

for all i, this expression becomes

$$a_{Ti} = \frac{(\sin\omega_{i+1} + \cos\omega_{i+1})^2 z_{i+1}^2 - (\sin\omega_i + \cos\omega_i)^2 z_i^2}{4s_i},$$

where ω_i is defined as previously.

One easily sees that imposing bounds on this acceleration amounts to enforcing a constraint over both z_i and z_{i+1}. At all every path index $i \in [0, m-1]$, we denote by $\phi_i(z_i, z_{i+1})$ the conjunction of all physical constraints that jointly involve z_i and z_{i+1}.

For simplicity, we can divide the constraint $\phi_i(z_i, z_{i+1})$ in two parts. First, due to constraints on the acceleration of the robot, the value of z_i at the beginning of the $(i+1)$-th step determines an upper bound on the value of z_{i+1} at the end of this step (i.e., the largest value of z_{i+1} that makes the constraint $\phi_i(z_i, z_{i+1})$ satisfiable). If this upper bound is denoted by $z_{maxF_i}(z_i)$, we have

$$z_{i+1} \leq z_{maxF_i}(z_i),$$

where

$$z_{maxF_i} : [0, z_{max_i}] \to \mathbb{R}_{\geq 0},$$

for all $i \in [0, m-1]$.

Similarly, the value of z_i at the end of the i-th step determines an upper bound on the value of z_{i-1} at the beginning of this step, due to constraints on the deceleration of the robot. Let $z_{maxP_i}(z_i)$ denotes this upper bound, we then have

$$z_{i-1} \leq z_{maxP_i}(z_i),$$

where

$$z_{maxP_i}(z_i) : [0, z_{max_i}] \to \mathbb{R}_{\geq 0},$$

for all $i \in [1, m]$.

Note that, parameters a_{Tmin}, a_{Tmax}, and the functions a_{LRmin}, a_{LRmax} a_{Smin} and a_{Smax} all translate into constraints over the velocities at two successive path indices and therefore belong to the transition constraints group. As for local state constraints, all the developments related to these parameters and functions are provided (for every kind of segment) in Appendix A.

Unfortunately, one can remark that the constraint $\phi_i(z_i, z_{i+1})$ may not be satisfiable for every $z_i \in [0, z_{max_i}]$ (or similarly for every $z_{i+1} \in [0, z_{max_{i+1}}]$). Figure 4.2 illustrates an example of this issue. With a valid fixed value of $z_i \leq z_{max_i}$, it could be impossible, due to the geometry of the path, to assign a suitable value of z_{i+1} that takes into account both transition constraints on the right and left wheels.

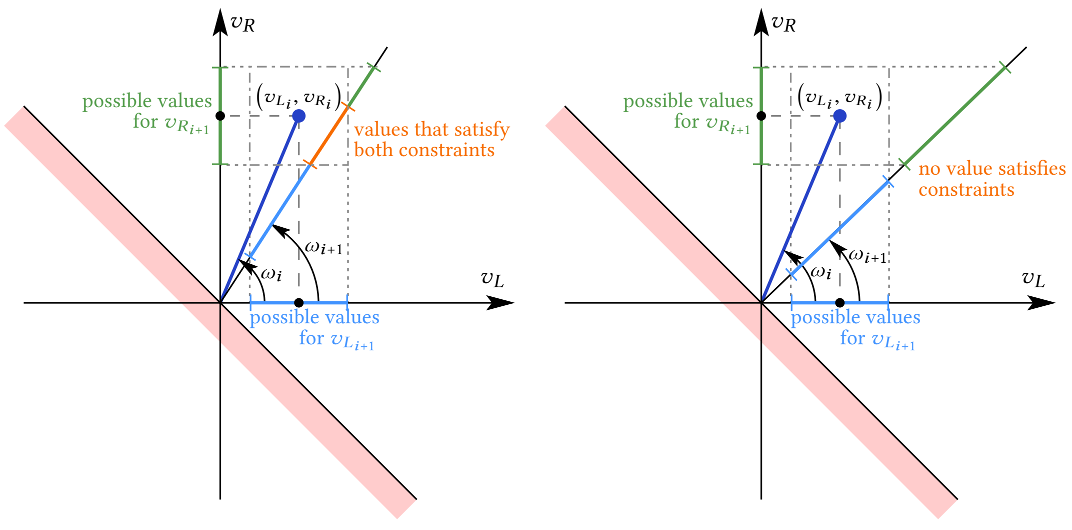
In Figure 4.2(a), constraints on both left and right wheels are consistent and $\phi_i(z_i, z_{i+1})$ is satisfiable, but in Figure 4.2(b), the intersection of the set of values of z_{i+1} that satisfy acceleration on the left wheel and the set of values of z_{i+1} that satisfy acceleration on the right one is empty and therefore $\phi_i(z_i, z_{i+1})$ is unsatisfiable.

In order to always be able to reduce the speed of the robot, it is then legitimate to require the constraint $\phi_i(z_i, z_{i+1})$ to have the following properties:

$$\forall i \in [0, m-1], \forall z_i, z_i', z_{i+1} : z_i' \leq z_i \wedge \phi_i(z_i, z_{i+1}) \Rightarrow \exists z_{i+1}' : z_{i+1}' \leq z_{i+1} \wedge \phi_i(z_i', z_{i+1}'),$$

$$\forall i \in [1, m], \forall z_i, z_i', z_{i-1} : z_i' \leq z_i \wedge \phi_{i-1}(z_{i-1}, z_i) \Rightarrow \exists z_{i-1}' : z_{i-1}' \leq z_{i-1} \wedge \phi_{i-1}(z_{i-1}', z_i').$$

Intuitively, these properties ensure that a value of z_i, that makes the constraints $\phi_i(z_i, z_{i+1})$ and $\phi_i(z_{i-1}, z_i)$ satisfiable, can always be lowered while still allowing the two constraints to be satisfiable (i.e., with $z_{i+1}' \leq z_{i+1}$ and $z_{i-1}' \leq z_{i-1}$).



v_R
possible values for $v_{R_{i+1}}$
(v_{L_i}, v_{R_i})
values that satisfy both constraints
ω_{i+1}
ω_i
v_L
possible values for $v_{L_{i+1}}$
v_R
possible values for $v_{R_{i+1}}$
(v_{L_i}, v_{R_i})
no value satisfies constraints
ω_{i+1}
ω_i
v_L
possible values for $v_{L_{i+1}}$

(a) Satisfiable. (b) Unsatisfiable.

Figure 4.2 – $\phi_i(z_i, z_{i+1})$ may be unsatisfiable.

4.2.2 Optimization Algorithm

We are now ready to describe our procedure for computing the fastest physically-feasible speed profile for a given path. This procedure has originally been introduced in [Len08]. It differs from [CC00, BP11b, KS12] by the much broader range of physical constraints that it supports, such as context-sensitive constraints, and constraints on individual wheels.

The problem of computing an optimal value for each $z_i \in [0, m]$ is solved by the algorithm given in Algorithm 4.1. If the initial value satisfies $z_{init} < z_{max_0}$, then this algorithm proceeds in three main stages:

1. Lines 3 – 8:

   It assigns for every index i, an initial value to each z_i corresponding to its upper bound z_{max_i} (i.e., the largest possible value allowed by the local state constraints at that point) or the initial value z_{init}.

   It is also essential to make sure that the constraints that belong to the second group (i.e., the transition constraints) remain satisfiable: if z_{max_i} is such that the constraint $\phi_i(z_{max_i}, z_{i+1})$ does not hold for at least one z_{i+1}, then z_{max_i} has to be lowered into the largest value that makes the constraint satisfiable. In the same way, z_{max_i} must be sufficiently small for the constraint $\phi_{i-1}(z_{i-1}, z_{max_i})$ to be satisfiable in z_{i-1}. These operations can be carried out numerically, a simple strategy being to perform a binary search until the required precision is reached.

```
1  Function Optimize-z($m$, $z_{init}$, $z_{max_i}[1:m]$, $z_{maxF_i}[0:m-1]$, $z_{maxP_i}[1:m]$)
2      if $z_{max_0} < z_{init}$ then fail;
3      $z_0 := z_{init}$;
4      for $i := 1, 2, \ldots, m$ do
5          $z_i := z_{max_i}$;
6          if $i < m$ and $z_{maxF_i}(z_i)$ fails then $z_i :=$ SearchLargestZiF($z_{maxF_i}$, $z_i$);
7          if $z_{maxP_i}(z_i)$ fails then $z_i :=$ SearchLargestZiP($z_{maxP_i}$, $z_i$);
8      end
9      for $i := 0, 1, \ldots, m-1$ do
10         $z := z_{maxF_i}(z_i)$;
11         if $z < z_{i+1}$ then $z_{i+1} := z$;
12     end
13     for $i := m, m-1, \ldots, 1$ do
14         $z := z_{maxP_i}(z_i)$;
15         if $z < z_{i-1}$ then $z_{i-1} := z$;
16     end
17     if $z_0 < z_{init}$ then fail;
18     else return $z_i[0:m]$;
```

Algorithm 4.1 – Algorithm for assigning optimal values to the parameters z_i.

2. Lines 9 – 12:
   It lowers the value of all z_{i+1} whenever the constraint
   $$z_{i+1} \leq z_{maxF_i}(z_i) \tag{4.3}$$
   is not satisfied. Since this constraint defines an upper bound on z_{i+1} that depends on the value of z_i, this procedure is carried out with i increasing from 0 to $m-1$.

3. Lines 13 – 16:
   It lowers the value of all z_{i-1} whenever the constraint
   $$z_{i-1} \leq z_{maxP_i}(z_i) \tag{4.4}$$
   is not satisfied. Since this constraint defines an upper bound on z_{i-1} that depends on the value of z_i, this procedure is carried out with i decreasing from m to 1.

After completing the third stage, the computed values of z_i at all path indices are such that constraints of both groups are satisfied, and it remains to check that the computed initial velocity z_0 corresponds to the initial speed of the robot (Line 17). In the case of a mismatch (meaning that z_0 had to be lowered during third stage), it is then impossible to follow the path with the specified initial speed while satisfying all physical constraints, and the speed profile computation returns an error.

Otherwise, one can straightforwardly compute, from the value of z_i at all indices and the geometry of the path, the instant t_i at which the corresponding configuration will be reached.

Indeed, in the case of a forward or backward path segment, the amount of time Δt_i needed for travelling the distance s_i of the $(i+1)$-th step is given by

$$\Delta t_i = \frac{2\,|s_i|}{|v_i + v_{i+1}|} = \frac{2\sqrt{2}\,|s_i|}{(\sin\omega_i + \cos\omega_i)\,z_i + (\sin\omega_{i+1} + \cos\omega_{i+1})\,z_{i+1}},$$

for all $i \in [0, m-1]$, where ω_i is defined as in Equation 4.2.

In the case of a local turn path segment, if θ_i denotes the orientation of the robot at the path index i, the length of the circular arc driven by the left wheel during the $(i+1)$-th is given by

$$|s_{L_i}| = \frac{e}{2}\,|[\theta_i - \theta_{i+1}]|\,,$$

and the amount of time Δt_i needed for performing this step is given by

$$\Delta t_i = \frac{2\,|s_{L_i}|}{|v_{L_i} + v_{L_{i+1}}|} = \frac{2\,|s_{L_i}|}{z_i + z_{i+1}}.$$

Let us remind that the instant t_i is given by

$$t_i = \sum_{0 \le j < i} \Delta t_j,$$

for all $i \in [0, m]$.

The computational cost of this algorithm is linear in the number of path steps, provided that all constraints can be solved in bounded time.

Figure 4.3 shows an example of the execution of this algorithm for a given discretized path and some set of physical constraints. This figure depicts:

- In blue: the velocities assigned after the first stage of the algorithm. They are such that they all satisfy the local state constraints.
- In green: the velocities assigned after the second stage. They now also satisfy the constraints $z_{i+1} \le z_{maxF_i}(z_i)$ (i.e., an upper bound on the acceleration of the robot).

- In red: the velocities assigned after the third stage. They now satisfy all the given physical constraints.

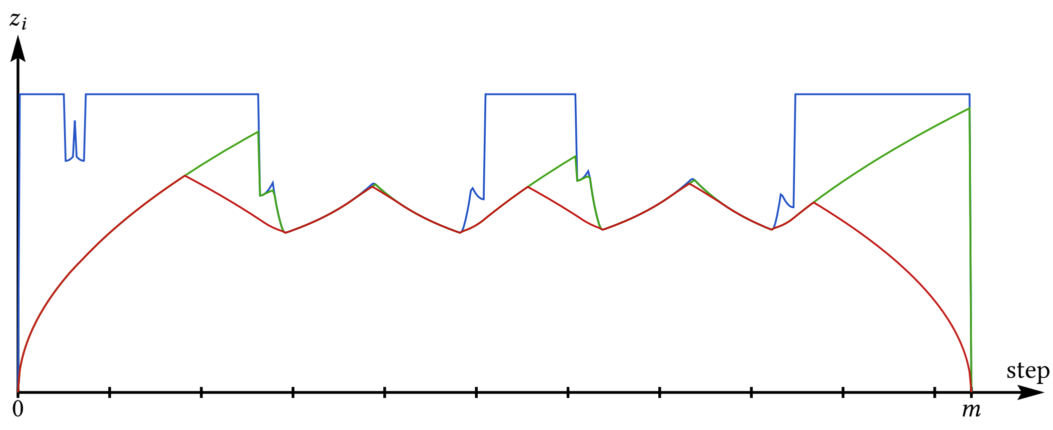


Figure 4.3 – Execution of the algorithm.

4.2.3 Correctness, Optimality and Completeness of the Algorithm

Constraints Properties

In order to prove that Algorithm 4.1 is correct and optimal, it is essential to define some properties that need to be fulfilled by our set of physical constraints. In Section 4.2.4, we will prove that the 16 physical constraints considered for the Eurobot application actually satisfy these properties.

1. Local state constraints (i.e., constraints that only involve the velocity z_i at a given path index i) should all translate into a positive upper bound on the velocity z_i. In other words, this means that a velocity of zero should satisfy the local state constraints for all $i \in [0, m]$.

   This implies, for example, that constraints that impose a minimal positive speed can not be processed by this algorithm, and therefore this method is not suited for computing an optimal speed profile for an airplane or other vehicles that need a minimum speed to operate properly.

2. As already discussed in Section 4.2.1, the conjunction of all transition constraints $\phi_i(z, z_{i+1})$ (i.e., constraints that involve the velocities z_i and z_{i+1} at two successive

indices i and $i+1$ of the path) should satisfy the following properties:

$$\forall i \in [0, m-1], \forall z_i, z'_i, z_{i+1} : z'_i \leq z_i \wedge \phi_i(z_i, z_{i+1}) \\ \Rightarrow \exists z'_{i+1} : z'_{i+1} \leq z_{i+1} \wedge \phi_i(z'_i, z'_{i+1}), \quad (4.5)$$

$$\forall i \in [1, m], \forall z_i, z'_i, z_{i-1} : z'_i \leq z_i \wedge \phi_{i-1}(z_{i-1}, z_i) \\ \Rightarrow \exists z'_{i-1} : z'_{i-1} \leq z_{i-1} \wedge \phi_{i-1}(z'_{i-1}, z'_i). \quad (4.6)$$

This means that, if z_i makes the constraints $\phi_i(z_i, z_{i+1})$ and $\phi_{i-1}(z_{i-1}, z_i)$ satisfiable, then this value can always be lowered while still allowing the two constraints to be satisfiable, which is physically sound.

3. As a legitimate requirement, we also impose that

$$\phi_i(0, 0) \quad (4.7)$$

holds for all $i \in [0, m]$.

This means that, with an infinite amount of time, it should always be possible to move from one configuration to another. This also means, for example, that this method can not handle physical constraints that impose a positive minimal acceleration when the robot is stationary.

4. In order to be able to prove the optimality of the algorithm, the conjunction of all transition constraints should also have these properties:

$$\forall i \in [0, m-1], \forall z_i, z_{i+1} : \phi_i(z_i, z_{i+1}) \wedge \nexists z'_{i+1} : z'_{i+1} > z_{i+1} \wedge \phi_i(z_i, z'_{i+1}) \\ \Rightarrow \nexists z'_i, z'_{i+1} : z'_i \leq z_i \wedge z'_{i+1} > z_{i+1} \wedge \phi_i(z'_i, z'_{i+1}), \quad (4.8)$$

$$\forall i \in [1, m], \forall z_i, z_{i-1} : \phi_{i-1}(z_{i-1}, z_i) \wedge \nexists z'_{i-1} : z'_{i-1} > z_{i-1} \wedge \phi_{i-1}(z'_{i-1}, z_i) \\ \Rightarrow \nexists z'_i, z'_{i-1} : z'_i \leq z_i \wedge z'_{i-1} > z_{i-1} \wedge \phi_{i-1}(z'_{i-1}, z'_i). \quad (4.9)$$

This means that if z_{i+1} (resp. z_{i-1}) is the largest achievable velocity from z_i, lowering z_i will never allow to achieve a velocity z'_{i+1} larger than z_{i+1} (resp. z'_{i-1} larger than z_{i-1}). In other words, if we assume, for example, that the robot is moving forward, lowering (resp. increasing) the speed at the beginning of a step can not allow to accelerate (resp. decelerate) to a larger (resp. slower) speed while travelling the step, which is physically sound.

As a consequence from all these requirements, we can also derive the following properties:

- For the local state constraints, we have
$$z_{max_i} \geq 0, \tag{4.10}$$
for all $i \in [0, m]$.

- If z_{maxL_i} denotes the velocity at the path index i after the first stage of the algorithm, from Property 4.7 and Property 4.10, we have
$$z_{maxL_i} \geq 0, \tag{4.11}$$
for all $i \in [0, m]$.

- If we recall that we have defined $z_{maxF_i}(z)$ as the largest value of z_{i+1} that makes the constraint $\phi_i(z, z_{i+1})$ satisfiable, and similarly $z_{maxP_i}(z)$ as the largest value of z_{i-1} that makes the constraint $\phi_{i-1}(z_{i-1}, z)$ satisfiable, as we know that after the first stage, z_{maxL_i} is such that $\phi_i(z_{maxL_i}, z_{i+1})$ and $\phi_{i-1}(z_{i-1}, z_{maxL_i})$ are satisfiable for at least one z_{i+1} and one z_{i-1}, by using Properties 4.5 and 4.6, we have
$$\forall z : 0 \leq z \leq z_{maxL_i} \Rightarrow z_{maxF_i}(z) \geq 0, \tag{4.12}$$
for all $i \in [0, m-1]$, and
$$\forall z : 0 \leq z \leq z_{maxL_i} \Rightarrow z_{maxP_i}(z) \geq 0, \tag{4.13}$$
for all $i \in [1, m]$.

- Also from the definitions of $z_{maxF_i}(z)$ and $z_{maxP_i}(z)$, and by combining Properties 4.8, 4.9, 4.12 and 4.13, we can derive:
$$\forall i \in [0, m-1], \forall z, z' : 0 \leq z' \leq z \leq z_{maxL_i} \Rightarrow z_{maxF_i}(z') \leq z_{maxF_i}(z), \tag{4.14}$$
$$\forall i \in [1, m], \forall z, z' : 0 \leq z' \leq z \leq z_{maxL_i} \Rightarrow z_{maxP_i}(z') \leq z_{maxP_i}(z). \tag{4.15}$$

- Note that, from the definitions of $z_{maxF_i}(z)$ and $z_{maxP_i}(z)$, we can also derive the following properties:
$$\forall i \in [0, m-1], \forall z : 0 \leq z \leq z_{maxL_i} \Rightarrow z \leq z_{maxP_{i+1}}(z_{maxF_i}(z)), \tag{4.16}$$
$$\forall i \in [1, m], \forall z : 0 \leq z \leq z_{maxL_i} \Rightarrow z \leq z_{maxF_{i-1}}(z_{maxP_i}(z)). \tag{4.17}$$

  Indeed, as $z_{maxP_i}(z)$ is the largest value of z_{i+1} that makes the constraint $\phi_i(z, z_{i+1})$ satisfiable, the largest value z' that makes the constraint $\phi_i(z', z_{maxP_i}(z))$ satisfiable is at least equal to z (because $\phi_i(z, z_{maxP_i}(z))$ is satisfiable), which prove the Property 4.16. A similar reasoning can be used to prove Property 4.17.

Note that, as discussed previously, constraints in both groups may be context-sensitive: The constraints involving z_i and z_j at two different locations i and j can differ, or be expressed with respect to different values of parameters. This makes it possible, for instance, to impose tighter speed limits in the close vicinity of obstacles, or to express the admissible angular velocity of a wheel as a function of the robot speed.

Correctness

To prove correctness of Algorithm 4.1, we show that after its three stages, the computed values of z_i at all path indices are such that all the physical constraints are satisfied.

In other words, we prove that the computed values of z_i satisfy:

$$\forall i \in [0, m] : z_i \leq z_{max_i} \tag{4.18}$$

$$\forall i \in [0, m-1] : z_{i+1} \leq z_{maxF_i}(z_i) \tag{4.19}$$

$$\forall i \in [1, m] : z_{i-1} \leq z_{maxP_i}(z_i). \tag{4.20}$$

After completing the first stage of the algorithm, the constraints $z_i \leq z_{max_i}$ are satisfied. Further operations performed by the algorithm will only be able to decrease the value of the variable z_i or to leave it unchanged, hence guaranteeing that the constraint $z_i \leq z_{max_i}$ will stay satisfied.

After the second stage, all the constraints $z_{i+1} \leq z_{maxF_i}(z_i)$ are satisfied.

At the end of the third stage, all the constraints $z_{i-1} \leq z_{maxP_i}(z_i)$ are satisfied. It is however not obvious that the constraint $z_i \leq z_{maxF_{i-1}}(z_{i-1})$, which was satisfied at the end of the second stage, still holds. We prove that the validity of this constraint is preserved during third stage by considering the two possible scenarios:

- If the value of z_{i-1} computed during the second stage has not been modified during the third stage, then the constraint $z_i \leq z_{maxF_{i-1}}(z_{i-1})$ still holds, since the value of z_i has not been increased during the third stage.

- If the value of z_{i-1} computed during the second stage has been lowered during the third stage, then the final value of z_{i-1} satisfies

  $$z_{i-1} = z_{maxP_i}(z_i).$$

  From Property 4.17, we have

  $$z_i \leq z_{maxF_{i-1}}(z_{maxP_i}(z_i)),$$

hence

$$z_i \leq z_{maxF_{i-1}}(z_{i-1}),$$

which establishes that the constraint 4.19 is still satisfied for all $i \in [0, m-1]$.

Optimality

In this section, we prove that this optimization algorithm yields a time-optimal speed profile. This can be done by showing that increasing the speed of the robot at any location on the path would lead to violating at least one physical constraint.

Indeed, after executing the three main stages of this algorithm, there is four possible cases for the computed values of z_i:

1. z_0 corresponds to the initial speed of the robot. This velocity can not be increased since this value can not be modified.
2. For all $i \in [1, m]$: z_i has been lowered for the last time in the first stage of the algorithm (i.e., in order to satisfy $z_i \leq z_{max_i}$ or to allow $\phi_i(z_i, z_{i+1})$ and $\phi_{i-1}(z_{i-1}, z_i)$ to be satisfiable). This velocity can not be increased, since increasing it would lead to violating at least one local state constraint, or preventing at least one transition constraint to be satisfiable.
3. For all $i \in [1, m]$: z_i has been lowered for the last time in the second stage (i.e., in order to satisfy $z_i \leq z_{maxF_{i-1}}(z_{i-1})$). From Property 4.14, we know that this velocity can only be increased if z_{i-1} is increased.
4. For all $i \in [0, m-1]$: z_i has been lowered for the last time in the third stage (i.e., in order to satisfy $z_i \leq z_{maxP_{i+1}}(z_{i+1})$). From Property 4.15, we know that this velocity can only be increased if z_{i+1} is increased.

Furthermore, if a value of z_i is lowered for the last time in the second stage, then z_i necessarily satisfies

$$z_i = z_{maxP_{i-1}}(z_{i-1}).$$

From Property 4.16, we have

$$z_{i-1} \leq z_{maxP_i}(z_{maxF_{i-1}}(z_{i-1})),$$

hence

$$z_{i-1} \leq z_{maxP_i}(z_i),$$

which establishes that z_{i-1} has not been lowered during the third stage, and thus that it will never exist cycle situations where z_i may only be increased if z_{i-1} is increased, and where at the same time z_{i-1} may only be increased if z_i is increased.

Therefore, for all $i \in [0, m]$, we can construct for z_i a list of dependencies that will always stop in z_0 or in a value that has been lowered for the last time in the first stage. Since this value can not be increased without violating a constraint, going through the list, no value can be increased. Hence, as a consequence, z_i can not be increased without violating at least one constraint. Therefore, the value of z_i at the end of the algorithm is optimal.

Completeness

Let, as previously, z_{maxL_i} denote the velocity at the path index i after the first stage of the algorithm. Since all other operations (i.e., after the first stage) performed by the algorithm will only be able to decrease the value of the variable z_i or leave it unchanged, we can prove from Properties 4.11, 4.12 and 4.13, that the different stages of the algorithm will never fail, and therefore that this algorithm will always terminate, provided that all constraints can be solved in bounded time.

There are, actually, four possible execution scenarios:

1. The algorithm fails before the first stage because $z_{max_0} < z_{init}$, and it is trivial to show that there is no speed profile that satisfy the constraints.

2. The algorithm fails after the third stage because $z_0 < z_{init}$, and because the algorithm is correct and time-optimal, we can show that z_0 can not be increased, hence there is no speed profile that satisfy the constraints.

3. The algorithm computes a speed profile where two or more consecutive velocities are equal to zero. In this case the robot need an infinite amount of time to reach the destination, and from the same arguments of correctness and optimality, we can show that there is no speed profile that satisfy the constraints and that reaches the destination in bounded time.

4. In all other cases, the algorithm computes a speed profile that is correct and time optimal.

4.2.4 Constraints Considered for the Eurobot Application

We now show that the 16 physical constraints (defined in Section 4.1.5), that we consider for the Eurobot application, comply with the requirements that we have defined in Section 4.2.3.

All the developments related to these constraints are provided (for every kind of segment) in Appendix A.

First, the parameters v_{fm}, v_{LRmin}, v_{LRmax}, v_{Smin}, v_{Smax}, a_{Cmin}, a_{Cmax}, $\dot{\theta}_{Smax}$, and the functions v_{min}, v_{max} and $\dot{\theta}_{max}$ all translate into a positive upper bounds on the velocity z_i, and therefore a velocity of zero satisfies all the local state constraints for all $i \in [0, m]$, which fulfills our first requirement.

Second, the parameters a_{Tmin}, a_{Tmax}, and the functions a_{LRmin}, a_{LRmax} a_{Smin} and a_{Smax} all translate into constraints over the velocities at two successive path indices.

For simplicity, we will only show here that these transition constraints comply with our requirements in the case of forward path segments. A similar reasoning straightforwardly applies to backward or local turns path segments.

From Appendix A, we have:

$$a_{T_i} = \frac{(\sin\omega_{i+1} + \cos\omega_{i+1})^2\, z_{i+1}^2 - (\sin\omega_i + \cos\omega_i)^2\, z_i^2}{4s_i}$$

$$a_{L_i} = \frac{(\cos\omega_{i+1}\, z_{i+1} - \cos\omega_i\, z_i)\,(sc_i\, z_i + sc_{i+1}\, z_{i+1})}{2s_i}$$

$$a_{R_i} = \frac{(\sin\omega_{i+1}\, z_{i+1} - \sin\omega_i\, z_i)\,(sc_i\, z_i + sc_{i+1}\, z_{i+1})}{2s_i}$$

$$a_{S_i} = \frac{\left(\dfrac{sc_{i+1}\, z_{i+1}}{\cos\theta_{S_{i+1}}} - \dfrac{sc_i\, z_i}{\cos\theta_{S_i}}\right)(sc_i\, z_i + sc_{i+1}\, z_{i+1})}{4s_i},$$

where

$$sc_i = \sin\omega_i + \cos\omega_i,$$

$$sc_{i+1} = \sin\omega_{i+1} + \cos\omega_{i+1},$$

a_{T_i}, a_{L_i}, a_{R_i} and a_{S_i} denote respectively the tangential acceleration of the robot, and the tangential acceleration of left, right and steering wheels during the $(i+1)$-th step of the trajectory, and where ω_i and θ_i are as previously defined.

These translate into the following transition constraints:

$$a_{T\,min} \leq a_{T_i} \leq a_{T\,max}$$
$$a_{LRmin} \leq a_{L_i} \leq a_{LRmax}$$
$$a_{LRmin} \leq a_{R_i} \leq a_{LRmax}$$
$$a_{S\,min} \leq a_{S_i} \leq a_{S\,max}.$$

From Section 4.1.5, we know that all the lower bounds are always lower or equal to zero, and similarly that all the upper bounds are always larger or equal to zero, then, all these constraints are satisfied with the velocities $z_i = 0$ and $z_{i+1} = 0$.
Therefore

$$\phi_i(0,0)$$

holds for all $i \in [0, m]$.

Constant Transition Constraints

For now, we will assume that the functions a_{LRmin}, a_{LRmax} $a_{S\,min}$ and $a_{S\,max}$, that describe the constraints, are constant (i.e., they do not depend on the wheel speeds).

We first show that if the transition constraints are satisfied with z_i and z_{i+1}, they are also satisfied with $z'_i = k\, z_i$ and $z'_{i+1} = k\, z_{i+1}$, for all $k \in [0, 1]$.

Indeed, for the tangential acceleration, we have

$$\begin{aligned} a'_{T_i} &= \frac{(\sin\omega_{i+1} + \cos\omega_{i+1})^2\, z'^2_{i+1} - (\sin\omega_i + \cos\omega_i)^2\, z'^2_i}{4s_i} \\ &= \frac{(\sin\omega_{i+1} + \cos\omega_{i+1})^2\, k^2\, z^2_{i+1} - (\sin\omega_i + \cos\omega_i)^2\, k^2\, z^2_i}{4s_i} \\ &= k^2\, a_{T_i}, \end{aligned}$$

and similarly, for the other accelerations, we have

$$\begin{aligned} a'_{L_i} &= k^2\, a_{L_i} \\ a'_{R_i} &= k^2\, a_{R_i} \\ a'_{S_i} &= k^2\, a_{S_i}. \end{aligned}$$

Since $k \in [0, 1]$, and since all the transition constraints are satisfied with z_i and z_{i+1}, the new transition constraints

$$a_{T\,min} \leq a'_{T_i} \leq a_{T\,max}$$

$$a_{LRmin} \leq a'_{L_i} \leq a_{LRmax}$$

$$a_{LRmin} \leq a'_{R_i} \leq a_{LRmax}$$

$$a_{S\,min} \leq a'_{S_i} \leq a_{S\,max}.$$

are necessarily satisfied, which proves by construction that the transition constraints comply with Properties 4.5 and 4.6.

It remains to prove that the transition constraints also comply with the two last Properties 4.8 and 4.9. We prove this in the following way: if z_{i+1} is the largest achievable velocity from z_i that satisfy the transition constraints, this means that at least one of the transition constraint is such that increasing z_{i+1} leads to increasing its corresponding acceleration in absolute value, and at the same time, leads to violating the constraint.

Let, for instance, this constraint be the one on the tangential acceleration of the robot, we have

$$|a_{T_i}| = \frac{\left|(\sin\omega_{i+1} + \cos\omega_{i+1})^2\, z_{i+1}^2 - (\sin\omega_i + \cos\omega_i)^2\, z_i^2\right|}{4s_i},$$

since we know that increasing z_{i+1} leads to increasing $|a_{T_i}|$, we know that

$$(\sin\omega_{i+1} + \cos\omega_{i+1})^2\, z_{i+1}^2 \geq (\sin\omega_i + \cos\omega_i)^2\, z_i^2,$$

and thus, whatever the geometry of the path, since $|a_{T_i}|$ can not be increased without violating the constraint, lowering z_i leads to a lower or equal value of z_{i+1}.

If this constraint is, on the other hand, for example, the one involving the tangential acceleration of the left wheel, this property becomes less obvious, but since this acceleration can also be estimated in the following way

$$\begin{aligned} |a_{L_i}| &= \frac{|v_{L_{i+1}} - v_{L_i}|}{\Delta t_i} \\ &\simeq \frac{\left|v_{L_{i+1}}^2 - v_{L_i}^2\right|}{2\,|s_{L_i}|} \\ &= \frac{\left|\cos^2\omega_{i+1}\, z_{i+1}^2 - \cos^2\omega_i\, z_i^2\right|}{|s_{L_i}|}, \end{aligned}$$

where $|s_{L_i}|$ is the distance[1] driven by the left wheel during the $(i+1)$-th step, this leads to the same conclusion.

With a similar reasoning for the right and for the steering wheels, we prove that if z_{i+1} is the largest achievable velocity from z_i that satisfies the transition constraints, lowering z_i will not allow to achieve a velocity z'_{i+1} that is larger than z_{i+1}, which proves that the transition constraints comply with the Property 4.8.

Symmetrically, If z_{i-1} is the largest velocity that allows to achieve z_i while satisfying the transition constraints, this means that at least one of the transition constraints is such that increasing z_{i-1} leads to increasing its corresponding acceleration in absolute value, and at the same time, leads to violating the constraint.

Let, for instance, consider the constraint involving the tangential acceleration of the robot. We have

$$|a_{T_{i-1}}| = \frac{\left|(\sin\omega_i + \cos\omega_i)^2 z_i^2 - (\sin\omega_{i-1} + \cos\omega_{i-1})^2 z_{i-1}^2\right|}{4s_{i-1}},$$

since we know that increasing z_{i-1} leads to increasing $|a_{T_{i-1}}|$, we know that

$$(\sin\omega_{i-1} + \cos\omega_{i-1})^2 z_{i-1}^2 \geq (\sin\omega_i + \cos\omega_i)^2 z_i^2,$$

and thus, whatever the geometry of the path, since this $|a_{T_{i-1}}|$ can not be increased without violating the constraint, lowering z_i leads to a lower or equal value of z_{i-1}.

Once again, we can follow a similar reasoning for the acceleration on the various wheels, which proves that the transition constraints also comply with the Property 4.9.

Transition Constraints That Depend on the Speed

If the minimum and maximum accelerations of a wheel depend of its speed, the problem of computing a time-optimal speed profile is more complicated. Indeed, as we assume constant-acceleration motion when we are travelling a step, the speed of this wheel usually changes throughout the step.

As a consequence, knowing the maximal and minimal acceleration of the wheel as a function of the speed is not enough, and we will need a way to estimate the maximal and minimal mean acceleration of that wheel as functions of its initial speed and of the distance

[1] Note that this distance can be equal to zero. In this case the corresponding transition constraint naturally always holds, and thus can not be the limiting one.

travelled by that wheel.

Figure 4.4 shows the typical curve for the maximal acceleration (as a function of the wheel speed v_w) of a wheel driven by a DC electrical motor. For low speeds, the acceleration is bounded by a_{max} and when the absolute value of the speed exceeds a certain limit, the acceleration decreases linearly when the absolute value of the speed increases.

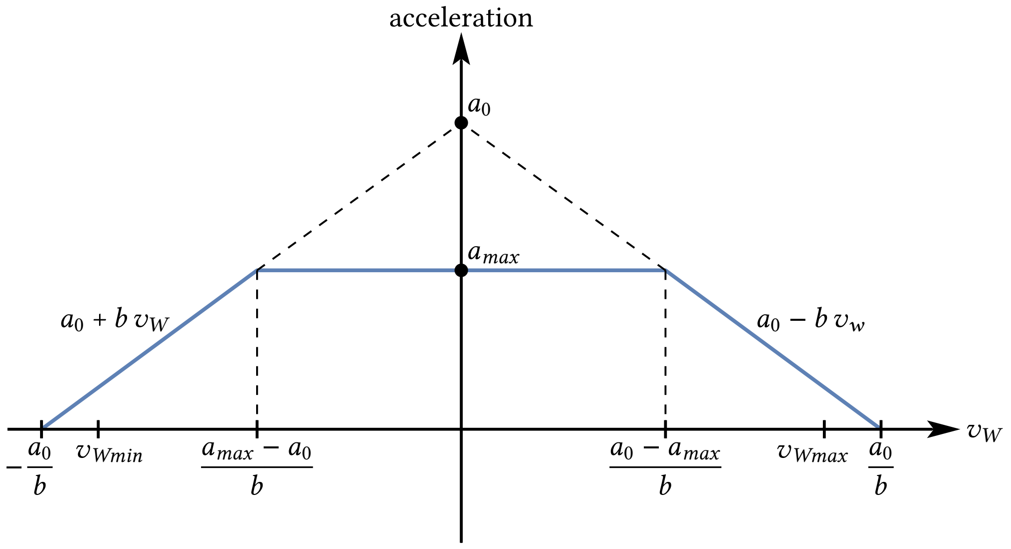


Figure 4.4 – Maximal acceleration in function of the wheel speed.

With this kind of acceleration curve, it is also possible to prove that the transitions constraints comply with Properties 4.5, 4.6, 4.8 and 4.9.

Assume, for instance, that the speed of the wheel at the beginning of the $(i + 1)$-th step is equal to v_{w_i}, and that s_{w_i} is the distance that must be driven by that wheel during this step.
If we have

$$\frac{a_0 - a_{max}}{b} \le v_{w_i} \le \frac{a_0}{b},$$

and if $v_{wmax_{i+1}}$ denotes the maximal speed of the wheel (i.e., satisfying the maximal acceleration curve) at the end of the step, then we also have

$$\frac{a_0 - a_{max}}{b} \le v_{wmax_{i+1}} \le \frac{a_0}{b}$$

We can then focus, on the positive part of the acceleration function that is not constant, that is

$$a(v_w) = a_0 - b\,v_w \tag{4.21}$$

with $a_0 > 0$ and $b > 0$.

Let us first express the travelled distance as a function of the wheel speed. Since the acceleration a of an object over a period of time is defined as its change in velocity divided by the duration of the period, we have

$$a = \frac{dv}{dt},$$

and, since the speed v of an object over a period of time is defined as its change in position divided by the duration of the period, we have

$$a = \frac{dv}{dt} \cdot \frac{ds}{ds} = \frac{ds}{dt} \cdot \frac{dv}{ds} = v\,\frac{dv}{ds},$$

thus, we obtain

$$ds = \frac{v}{a}\,dv,$$

$$\int ds = \int \frac{v}{a}\,dv.$$

Let $\hat{s}_i(v_w)$ denotes the function that compute the distance travelled by the wheel when the speed of that wheel reached $v_w \geq v_{w_i}$ with maximal acceleration, starting from an initial wheel speed of v_{w_i}. We have

$$\begin{aligned}\hat{s}_i(v_w) &= \int_{v_{w_i}}^{v_w} \frac{v}{a(v)} dv \\ &= \frac{b\,(v_{w_i} - v_w) + a_0 \log \frac{a_0 - b\,v_{w_i}}{a_0 - b\,v_w}}{b^2}.\end{aligned}$$

We can calculate (see Appendix A.5) the reciprocal of this function. Let $\hat{v}_i(s_w)$ denote the function that compute the speed reached by the wheel with maximal acceleration after travelling the distance s_w, starting from an initial wheel speed of v_{w_i}. We have

$$\hat{v}_i(s_w) = \frac{a_0}{b}\left(1 + W_0\left(\left(\frac{b}{a_0}v_{w_i} - 1\right)\mathrm{e}^{\left(\frac{b}{a_0}\left(v_{w_i} - b\,s_w\right) - 1\right)}\right)\right),$$

where W_0 is the main branch of the *Lambert W-function.*

The Lambert W-function[2] $W(x)$, depicted in Figure 4.5, is defined as the reciprocal function of $f(W) = W\mathrm{e}^W$. The Lambert W-function is defined for all $x \geq -\frac{1}{\mathrm{e}}$, and is double-valued for $x \in [-\frac{1}{\mathrm{e}}, 0[$. The constraint $W(x) \geq -1$ defines the function $W_0(x)$ (i.e., the main branch of the Lambert W-function), the constraint $W(x) \leq -1$ defines the lower branch $W_{-1}(x)$.

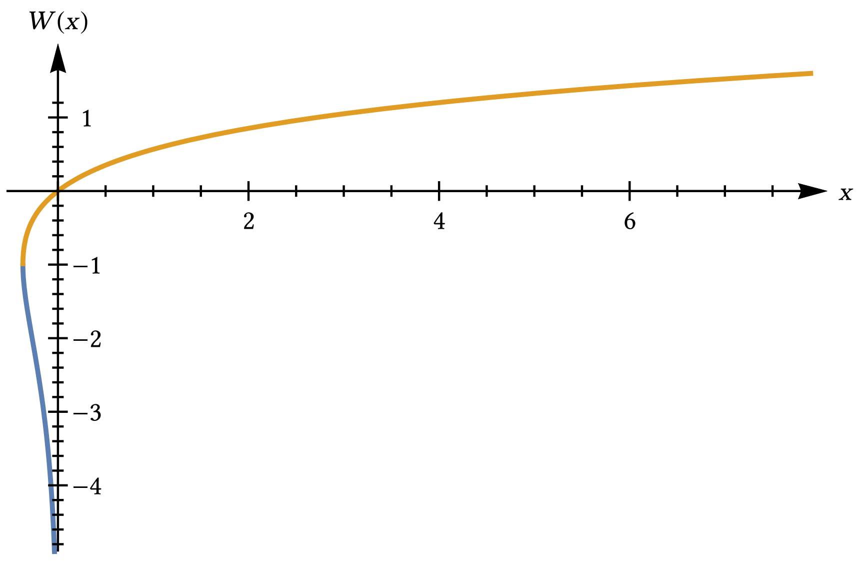


Figure 4.5 – The Lambert W-function $W(x)$.

Finally, the maximal mean acceleration of the wheel for the $(i+1)$-th step satisfying the acceleration curve, is given by

$$\hat{a}_{wmax}(v_{w_i}, s_{w_i}) = \frac{\hat{v}_i^2(s_{w_i}) - v_{w_i}^2}{2s_{w_i}} \tag{4.22}$$

where $v_{w_i} \geq \frac{a_0 - a_{max}}{b}$ is the speed of the wheel at the beginning of the step, and s_{w_i} is the distance that is driven by that wheel during this step.

The maximal acceleration constraint for that wheel becomes

$$a_{w_i} \leq \hat{a}_{wmax}(v_{w_i}, s_{w_i})$$

where a_{w_i} denotes the tangential acceleration of the wheel.

We can see[3] in Figure 4.6 that for a fixed value of s_{w_i}, $\hat{v}_i(s_{w_i})$ decreases monotonously

[2] Note that very efficient numerical approximations are known for computing the Lambert W-function [Fuk13], note also that functions for computing the two branches $W_0(x)$ and $W_{-1}(x)$ are available in the GNU Scientific Library (GSL).

[3] Knowing that $\frac{dW(x)}{dx} = \frac{W(x)}{x(1+W(x))}$, we can even prove, for all $a_0 > 0$, $b > 0$, $s_{w_i} > 0$ and $v_{w_i} \in [0, \frac{a_0}{b}]$, that $\frac{d\hat{v}_i(s_{w_i})}{dv_{w_i}} \in [0, 1]$, and thus that $\frac{d\hat{a}_{wmax}(v_{w_i}, s_{w_i})}{dv_{w_i}} \in \mathbb{R}_{\leq 0}$.

when v_{w_i} decreases, and that $\hat{a}_{wmax}(v_{w_i}, s_{w_i})$ increases monotonously when v_{w_i} decreases, which is exactly what is needed in order to comply with the required properties.

Indeed, first, if the constraint is satisfied with v_{w_i} and $v_{w_{i+1}}$, then it is also satisfied with $v'_{w_i} = k\, v_{w_i}$ and $v'_{w_{i+1}} = k\, v_{w_{i+1}}$, and second, since $\hat{v}_i(s_{w_i}) = v_{wmax_{i+1}}$ is the largest achievable velocity from v_{w_i} that satisfy the constraint, and since $\hat{v}_i(s_{w_i})$ decreases monotonously when v_{w_i} decreases, lowering v_{w_i} will not allow to achieve a velocity $v'_{w_{i+1}}$ that is larger than $v_{wmax_{i+1}}$.

Therefore, if we do similar calculations for $v_{w_i} < \frac{a_0 - a_{max}}{b}$ and for the minimal acceleration, dealing with the various transitions inside the curves, we can prove that the transitions constraints, in cases of acceleration curves like in Figure 4.4, also comply with Properties 4.5, 4.6, 4.8 and 4.9.

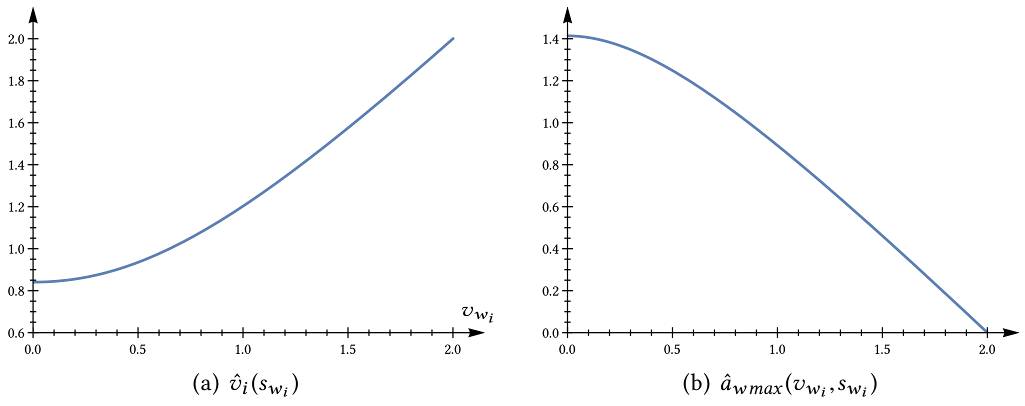


Figure 4.6 – $a_0 = 2$, $b = 1$, $s_{w_i} = 0.25$.

It is worth mentioning that similar results can be established for other acceleration curves, like, for example, the ones corresponding to internal combustion engines.

4.2.5 Improvements of the Optimization Algorithm

As already mentioned, provided that all constraints can be solved in bounded[4] time, the computational cost of our optimization algorithm is linear in the number of path steps. But as efficiency is one of our major concerns, we can further improve our algorithm to reduce its time cost in most of the cases.

[4] Which is, as we show in Appendix A, the case for the 16 physical constraints considered for the Eurobot case.

First, in Algorithm 4.2, we introduce two improvements that do not negatively affect the properties of correctness, optimality and completeness of Algorithm 4.1:

1. The first two stages of the previous algorithm can be mixed in the same loop, and since, for all $i \in [1, m-1]$, the values of z_i will now already satisfy the constraint $\phi_{i-1}(z_{i-1}, z_i)$, they will be, in most cases, smaller than z_{max_i}. Therefore, $z_{maxF_i}(z_i)$ is less likely to fail (i.e., $\phi_i(z_i, z_{i+1})$ is more likely to be satisfiable for at least one z_{i+1}) than in the first version of the algorithm, reducing the computation time.

   Note also that, since $\phi_{i-1}(z_{i-1}, z_i)$ is satisfied and since further operations will only be able to decrease z_i, from Property 4.6, we know now that $z_{maxP_i}(z_i)$ will never fail.

2. From the optimality proof, we know that if in the last stage $z_i = z_{maxP_{i-1}}(z_{i-1})$ (i.e., if z_i has been lowered by the transition constraints in the first loop), using Property 4.16, we have

   $$z_{i-1} \leq z_{maxP_i}(z_{maxF_{i-1}}(z_{i-1})),$$
   $$z_{i-1} \leq z_{maxP_i}(z_i),$$

   which establishes that z_{i-1} can not be lowered in the last stage, thus we can avoid computing $z_{maxP_i}(z_i)$ in this case, which will also likely reduce the computation time.

   Note that, we can even do better, and only compute $z_{maxP_i}(z_i)$ when z_i is smaller than the smallest z_i' that makes the constraint $\phi_{i-1}(z_{i-1}, z_i')$ satisfiable.

A second improvement strategy is to notice that, for transition constraints that depend on the speed, if the sampling steps are sufficiently small, then it is not necessary anymore to compute the maximal and minimal mean acceleration. Indeed, for sufficiently small steps (in regards to the derivative of the acceleration functions), the distances that are travelled by the wheels are small and we will have

$$\hat{a}_{wmax}(v_{w_i}, s_{w_i}) \simeq a_{wmax}(v_{w_i}),$$
$$\hat{a}_{wmin}(v_{w_i}, s_{w_i}) \simeq a_{wmin}(v_{w_i}).$$

One can remark that, if for $z_{maxF_i}(z_i)$ we use this approximation with z_i to compute the transitions constraints, and if for $z_{maxP_{i+1}}(z_{i+1})$ we use this approximation with z_{i+1}, then Property 4.6 may no longer be valid.

Indeed, for a wheel driven by an electrical motor, we see in Figure 4.4, that the acceleration decreases monotonously with the wheel speed, thus if $z_{i+1} > z_i$, $\phi_i(z_i, z_{i+1})$ could be satisfied when we approximate the constraints with z_i and in the same time not satisfied when we approximate the constraints with z_{i+1}.

```
Function Optimize2-z($m$, $z_{init}$, $z_{max_i}[1 : m]$, $z_{maxF_i}[0 : m-1]$, $z_{maxP_i}[1 : m]$)
    if $z_{max_0} < z_{init}$ then fail;
    $z_0 := z_{init}$;
    for $i := 0, 1, \ldots, m-1$ do
        $z_{i+1} := z_{max_{i+1}}$;
        $z := z_{maxF_i}(z_i)$;
        if $z_{maxF_i}(z_i)$ fails then
            $z_i :=$ SearchLargestZiF($z_{maxF_i}$, $z_i$);
            $reducedByFi[i] := false$;
            $z := z_{maxF_i}(z_i)$;
        end
        if $z < z_{i+1}$ then
            $z_{i+1} := z$;
            $reducedByFi[i+1] := true$;
        else
            $reducedByFi[i+1] := false$;
        end
    end
    for $i := m, m-1, \ldots, 1$ do
        if $reducedByFi[i]$ then
            $z := z_{maxP_i}(z_i)$;
            if $z < z_{i-1}$ then
                $z_{i-1} := z$;
                $reducedByFi[i-1] := false$;
            end
        end
    end
    if $z_0 < z_{init}$ then fail;
    else return $z_i[0 : m]$;
```

Algorithm 4.2 – Improved algorithm for assigning optimal values to z_i.

This problem can easily be solved: in the forward stage (i.e., when we compute z_{maxF_i}), one can approximate the constraints using the value of z_i and in the backward stage (i.e., when we compute $z_{maxP_{i+1}}$), to ensure satisfiability, use the minimum between z_{i+1} and z_i (i.e., the least restrictive velocity). Providing that the steps are sufficiently small, since the error induced by this approximation is at most equal to the effect of one step, the computed values remain close to the exact ones.

4.2.6 Static Steps

The case of static steps is much easier to handle than forward, backward or local turn path segments. Since the robot is stationary, we do not need to compute an optimal value for the velocity of the robot, but only the duration of the steps.

First, since a sequence of static steps is semantically equivalent to a single one, we can assume w.l.o.g. that every static segment is composed of a single static step (i.e., that the path segment (m, σ) under analysis is such that $m = 1$ and $v_0 = S$). The problem then reduces to computing the delay $\mu(0)$ associated to this step. There are two situations to consider:

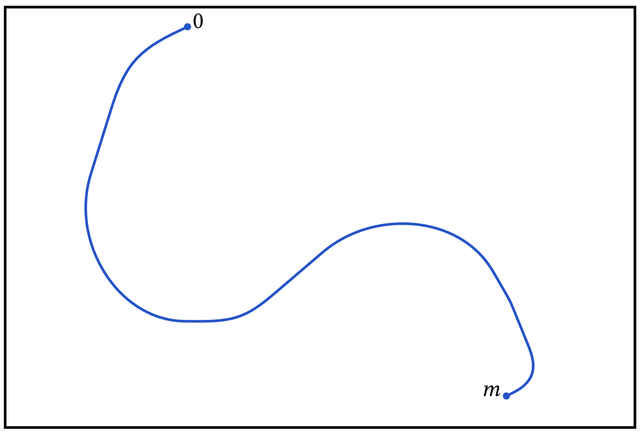
- *With a differential drive locomotion platform.* No specific constraint has to be satisfied, hence one has
$$\mu(0) = 0.$$
- *With a tricycle locomotion platform.* In this case, the maximum rate of variation of the steering angle must not be exceeded. The angle steered during the step is given by
$$[\theta_{S_1} - \theta_{S_0}],$$
where for each $i \in \{0, 1\}$, we have
$$\theta_{S_i} = \begin{cases} \arctan(e'\tilde{\kappa}_i) & \text{if } \tilde{\kappa}_i \neq \{-\infty, +\infty\} \\ \dfrac{\pi}{2} & \text{if } \tilde{\kappa}_i = +\infty \\ -\dfrac{\pi}{2} & \text{if } \tilde{\kappa}_i = -\infty. \end{cases}$$
The constraint
$$\frac{|[\theta_{S_1} - \theta_{S_0}]|}{\Delta t_0} \leq \dot{\theta}_{S\,max}$$
then yields
$$\mu(0) = \Delta t_0 = \frac{|[\theta_{S_1} - \theta_{S_0}]|}{\dot{\theta}_{S\,max}}.$$

4.2.7 Experimental Results

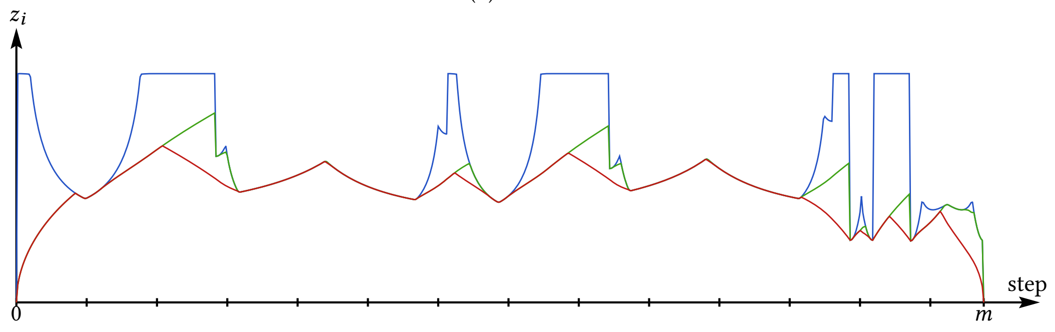
In order to illustrate the efficiency of our method, we report in Figure 4.7 the time needed for running the speed profile algorithm on a few sample trajectories experienced in the Eurobot application, considering 16 distinct physical constraints of robots with differential as well as tricycle drive. The computational cost typically amounts to less than quarter a millisecond of CPU time on an i5-460M processor running at 2.53 GHz, which is several orders of magnitude faster than techniques such as [SM85, CC00, VT08, BP11b, ZT13, KS12].

	Nb. of path steps	Path length	Computation time	Total time to travel
1	292	1.439 m	84.9 μs	2.543 s
2	520	2.570 m	156.0 μs	4.165 s
3	632	3.118 m	192.9 μs	4.967 s
4	656	3.215 m	209.8 μs	5.400 s
5	690	3.386 m	246.5 μs	6.478 s
6	1368	3.386 m	483.4 μs	6.480 s

Figure 4.7 – Experimental results.



(a) Path #5.



(b) Execution of the algorithm for path #5 (with $v_0 = v_{fm} = 0$).

Figure 4.8 – Example of Eurobot trajectory.

Note that the sixth path is the same as the fifth, but with smaller sampling steps. This path is depicted in Figure 4.8. As expected, the optimization algorithm returns very close results for those two inputs.

4.3 Trajectory Resampling

For each well-formed trajectory representation (Definition 4.1.5), arbitrary resampling between two sample points can be computed by similar developments as for path resampling (see Section 3.2.6). Therefore, for every t such that $0 \leq t < t_m$, we can estimate accurately the configuration of the robot (x_t, y_t, θ_t), the speeds of its left and right wheels (v_{L_t}, v_{R_t}) and, for a tricycle robot, the orientation and the speed of its steering wheel (θ_{S_t}, v_{S_t}).

If $t_i \leq t < t_{i+1}$, we already have travelled the $(i+1)$-th step during

$$dt = t - t_i.$$

And, as we assume constant acceleration motion during this step, we then have:

- If $\nu_i \in \{F, B\}$:

$$v_i = \frac{v_{L_i} + v_{R_i}}{2}, \quad v_{i+1} = \frac{v_{L_{i+1}} + v_{R_{i+1}}}{2},$$

$$a = \frac{v_{i+1} - v_i}{\Delta t_i},$$

$$v = v_i + a\,dt,$$

$$ds = dt\,v_i + a\frac{dt^2}{2},$$

  - if $\kappa_i = 0$ $(\theta_i = \theta_{i+1})$:

$$x_t = x_i - ds \sin\theta_i,$$

$$y_t = y_i + ds \cos\theta_i,$$

$$\theta_t = \theta_i, \quad \theta_{S_t} = \theta_{S_i},$$

$$v_{L_t} = v_{R_t} = v_{S_t} = v.$$

  - if $\kappa_i \neq 0$:

$$x_t = x_i - a_1 \sin d\theta + a_2 \cos d\theta - a_2,$$

$$y_t = y_i + a_1 \cos d\theta + a_2 \sin d\theta - a_1,$$

$$\theta_t = \theta_i + d\theta, \quad \theta_{S_t} = \theta_{S_i} + d\theta_S,$$

$$v_{L_t} = v\left(1 - \frac{e\tilde{\kappa}}{2}\right), \quad v_{R_t} = v\left(1 + \frac{e\tilde{\kappa}}{2}\right),$$

$$v_{S_t} = \frac{v}{\cos\theta_{S_t}},$$

    where

$$
\begin{aligned}
a_1 &= \frac{\sin\theta_i}{\kappa_i}, \;\; a_2 = \frac{\cos\theta_i}{\kappa_i},\\
d\theta &= \frac{ds}{s_i}\cdot[\theta_{i+1}-\theta_i]\,,\\
d\theta_S &= \frac{ds}{s_i}\cdot[\theta_{S_{i+1}}-\theta_{S_i}]\,,\\
\tilde{\kappa} &= \tilde{\kappa}_i + \frac{ds}{s_i}\cdot(\tilde{\kappa}_{i+1}-\tilde{\kappa}_i)\,.
\end{aligned}
$$

- If $v_i \in \{L,R\}$:

$$
\begin{aligned}
a &= \frac{v_{L_{i+1}}-v_{L_i}}{\Delta t_i},\\
v &= v_{L_i} + a\,dt,\\
ds &= dt\,v_{L_i} + a\frac{dt^2}{2},\\
ds_i &= \Delta t_i\,v_{L_i} + a\frac{\Delta t_i^2}{2},
\end{aligned}
$$

$$x_t = x_i,\;\; y_t = y_i,\;\; \theta_{S_t} = \theta_{S_i},\;\; \theta_t = \theta_i + d\theta,$$

$$
\begin{aligned}
v_{L_t} &= v,\;\; v_{R_t} = -v,\\
v_{S_t} &= \left|\frac{2e'}{e}v\right|,
\end{aligned}
$$

where

$$d\theta = \frac{ds}{ds_i}\cdot[\theta_{i+1}-\theta_i]\,.$$

- If $v_i = S$:

$$
\begin{aligned}
x_t &= x_i,\;\; y_t = y_i,\;\; \theta_t = \theta_i,\\
\theta_{S_t} &= \theta_{S_i} + d\theta_S,\\
v_{L_t} &= v_{R_t} = v_{S_t} = 0,
\end{aligned}
$$

where

$$d\theta_S = \frac{dt}{\Delta t_i}\cdot[\theta_{S_{i+1}}-\theta_{S_i}]\,.$$

4.4 Emergency Braking

In Section 4.2, we have shown how to solve the problem of computing a time-optimal speed-profile for a given path representation, but in practical applications, one also need

```
Function Braking-z($m$, $z_{init}$, $z_{minF_i}[0 : m-1]$)
    $z_0 := z_{init}$;
    for $i := 0, 1, \ldots, m-1$ do
        if $z_i == 0$ then
            $z_{i+1} := 0$;
        else
            $z_{i+1} := z_{minF_i}(z_i)$;
        end
    end
    return $z_i[0 : m]$;
```

Algorithm 4.3 – Algorithm to compute an optimal braking trajectory.

to tackle the problem of reaching as fast as possible a speed of zero on a given path.

For instance, suppose that we are following a previously computed time-optimal speed profile, and that, after some time and for some emergency reason, this trajectory has to be aborted. The problem then consists in stopping as fast as possible the robot while still satisfying its physical constraints.

Assuming that we have already computed a time-optimal speed profile for a path, we show in this section how to compute an optimal braking speed profile for that path.

We consider w.l.o.g. that we start to brake at the index 0 of a path segment with an initial velocity z_{init}[5].

Similarly to the definition of $z_{maxF_i}(z_i)$, due to constraints on the acceleration of the robot, the value of z_i at the beginning of the $(i+1)$-th step also determines a lower bound on the value of z_{i+1} at the end of this step (i.e., the smallest value of z_{i+1} that makes the constraint $\phi_i(z_i, z_{i+1})$ satisfiable). Let $z_{minF_i}(z_i)$ denote this lower bound.

The problem of computing an optimal braking profile[6] is then solved by the algorithm given in Algorithm 4.3.

[5] Note that from Definition 3.2.7, forward and backward path segments are supposed to have a curvature of zero at their origin. For the first path segment of a braking trajectory, since the initial velocity is fixed by the state of the robot, we will naturally drop this requirement.

[6] Note that, if the path is not long enough, it could be impossible to reach a velocity of zero at the end of the last path segment. (In this case, in the Eurobot application, we arbitrarily decide to continue to brake while following a straight line.)

Indeed, from Property 4.5, we have

$$\forall i \in [0, m-1],\ \forall z, z' : 0 \le z' \le z \le z_i \Rightarrow z_{minF_i}(z') \le z_{minF_i}(z), \tag{4.23}$$

and since we already have computed a time-optimal speed profile, we know that before computing our braking profile, all the velocities satisfy the physical constraints. Thus, before computing the braking profile, we also have

$$z_{i+1} \ge z_{minF_i}(z_i),$$

for all $i \in [0, m-1]$.

As a consequence, the new computed values of z_i will be smaller or equal to the previous ones, and thus will also satisfy the local state constraints, which prove the correctness of the algorithm. From the same Property 4.5, this also show that z_{minF_i} will never fail, which prove the completeness of the algorithm.

Additionally from Property 4.23, by following a similar reasoning as for the optimality of the time-optimal speed profile, we can also prove the optimality of this algorithm.

Note that, it is often reasonable to use less stringent physical constraints during the computation of an emergency braking profile. But he who can do more can do less, and this will not change the properties of correctness, optimality and completeness of the algorithm.

CHAPTER 5

Path Interpolation

This chapter addresses the problem of interpolating collision-free paths expressed as a sequence of straight line segments, such as those produced by planning algorithms, into smooth paths that can be precisely followed by nonholonomic robots without slowing down excessively. The solution that is presented in this chapter has the advantage of being simpler than other existing approaches, and has a low computational cost that allows a real-time implementation. It produces paths on which curvature and variation of curvature are bounded at all points, and preserves obstacle clearance. It aims at reducing the time needed for robots to reach the destination while still avoiding the different obstacles. Such paths are constructed out of clothoids, which are curves with a curvature that varies linearly with travelled distance.

5.1 Introduction

As we will see in Chapter 6, the general problem of moving a mobile robot from a configuration to another while avoiding a given set of obstacles can be tackled by several methods, such as cell decomposition [BLP85], roadmap [KWP$^+$11], rapidly-exploring random trees (RRT) [LKJ00], or potential field techniques [GC02].

In two-dimensional space, these path-planning algorithms usually produce obstacle-free paths that are expressed as broken lines, i.e., sequences of straight line segments, between the initial and final configurations.

A differential-drive robot, for example, cannot follow such a path without stopping at the junction points between adjacent line segments in order to change its orientation, which wastes time. In fact, path-planning methods focus on the obstacle avoiding problem, but usually do not consider the physical constraints of the robots.

This problem can, in particular, be alleviated by interpolating broken lines into smooth curves along which the orientation of the robot and the curvature remain continuous everywhere.

This chapter addresses the problem of computing such interpolations, so as to obtain paths that can be physically followed by a nonholonomic robot without the need for stopping or slowing down excessively. We develop an interpolation algorithm that guarantees that the obstacles cleared by a broken line are avoided as well by the resulting smoothed out path.

Note that, in this chapter, we consider only the case of differential-drive robots, but our results also apply to tricycle or car-like[1] platforms.

5.2 Problem Statement

The problem that we consider consists of smoothing out a path expressed as a broken line.

We define a *broken line path* as a sequence $p_0, p_1, \ldots p_n$ of points, with $n \geq 1$, such that

- each point p_i, with $i \in [0, n]$ is defined by its coordinates (x_i, y_i) in two-dimensional space, and
- each intermediate point p_i, with $i \in [1, n-1]$, is associated with a *clearance parameter* $c_i \in \mathbb{R}_{>0} \cup \{+\infty\}$. The goal of this parameter is to give information about the nearest obstacles of p_i. It will be defined formally in the next paragraphs.

The path is composed of the successive straight line segments $[p_0, p_1]$, $[p_1, p_2]$, $\ldots$, $[p_{n-1}, p_n]$.

In this chapter, in order to deal with obstacles, we assume that robots have zero measure, in other words, that a path clears a set of obstacles iff the intersection between its line segments and the union of all these obstacles is empty.

Note that the case of a robot with a cylindrical geometry of diameter d (i.e., a robot at a position P occupies a region of the plane that corresponds to a disk of diameter d centered

[1] For car-like platforms, since the curvature of the physically feasible paths is usually bounded, some restrictions may apply (see Section 5.4.4).

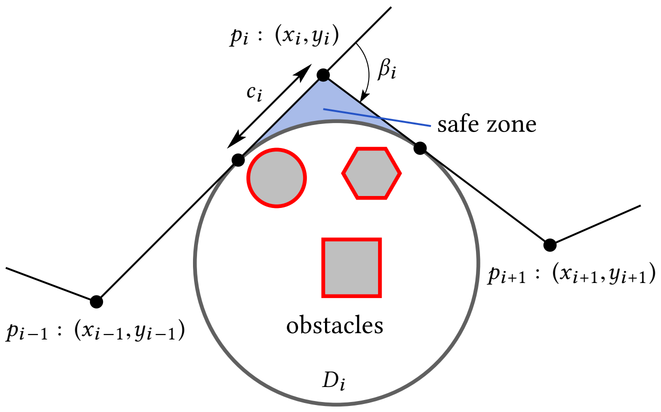

$p_i : (x_i, y_i)$
β_i
c_i
safe zone
$p_{i+1} : (x_{i+1}, y_{i+1})$
$p_{i-1} : (x_{i-1}, y_{i-1})$
obstacles
D_i

Figure 5.1 – Safe zone between adjacent segments.

in P, where d is a parameter of the robot) can straightforwardly be handled by growing obstacles by $d/2$ (i.e., by computing the Minkowski sum of the obstacles with a disk of diameter d).

The purpose of the clearance parameter c_i is to provide additional information about the location of the obstacles that are avoided when moving along the segments $[p_{i-1}, p_i]$ and $[p_i, p_{i+1}]$. This is achieved by considering a disk D_i that is tangent to both segments (which implies that its center belongs to the inner bisector of the angle formed by these segments), and that fully covers the obstacles cleared by the pair of segments.

This latter property precisely means that the area located between the disk D_i and the segments $[p_{i-1}, p_i]$ and $[p_i, p_{i+1}]$ is free from obstacles; we call this area the *safe zone* of this pair of segments[2]. The point is that any interpolation of the path that is confined to safe zones is guaranteed to avoid obstacles. The safe zone across p_i is characterized by the parameter c_i, defined as the distance between p_i and the points of tangency between D_i and the segments $[p_{i-1}, p_i]$ and $[p_i, p_{i+1}]$. An illustration is provided in Figure 5.1.

The parameter c_i may be left undefined ($c_i = +\infty$), in which case its value can be replaced by the smallest of the lengths of the two segments. (In other words, D_i is then the largest disk simultaneously tangent to both segments.)

For each $i \in [1, n-1]$, we define β_i as the angle between the vectors $\overrightarrow{p_{i-1}p_i}$ and $\overrightarrow{p_i p_{i+1}}$, which corresponds to the change in orientation of the robot when it moves from the segment $[p_{i-1}, p_i]$ to the segment $[p_i, p_{i+1}]$. Without loss of generality, we assume $\beta_i \neq 0$.

[2] By extension, we consider that the segments themselves also belong to their safe zone.

It is also natural to impose an upper bound on $|\beta_i|$. Indeed, with a large value of β_i, the segments $[p_{i-1}, p_i]$ and $[p_i, p_{i+1}]$ can be followed in opposite directions, and it is not always appropriate in such cases to interpolate the path into one that remains close to the segments. In this work, we arbitrarily impose the upper bound $|\beta_i| \leq \pi/2$ for all $i \in [1, n-1]$. In the case of adjacent segments forming an acute angle, it then becomes necessary to interleave an intermediate segment between them.

A differential-drive robot subject to physical constraints cannot follow a broken line path without stopping at junction points. As we have seen in Chapter 4, such constraints usually take the form of lower and upper bounds on the velocity and acceleration of the robot measured at specific locations, such as its individual wheels, center of mass, or other reference centers.

The speed that can be reached by the robot at some point of a path is then, among others, bounded by a function of the absolute curvature $|\kappa|$ at this point, as well as the rate of variation $\left|\frac{d\kappa}{ds}\right|$ of this curvature with respect to the linear travelled distance.

We are now ready to define precisely the interpolation problem. Given a broken line path, the goal is to compute a curve that leads from its origin to its endpoint staying within safe zones, and such that the absolute curvature and variation of curvature remain small at all points.

Note that, as already mentioned in previous chapters, for our Eurobot application, it is essential to be able to carry out this operation with a low computational cost.

5.3 Related Work

Various solutions were proposed to generate smooth feasible paths for nonholonomic robots. First, Dubins paths [Dub57] (paths composed of circular arc segments of maximum curvature and straight lines) are still commonly used for path smoothing. These paths are easy to compute but have curvature discontinuities which cause robots to slow down (and even stop) every time the curvature abruptly changes.

Other methods often use cubic splines [KH97], Bezier's curves [CCE08], B-splines [ESJ15] or NURBS (Non-Uniform Rational B-Splines) curves [AC05] to solve this issue, but they do not necessarily result in tight bounds on the variation of curvature of the resulting smoothed paths.

Clothoids, which are curves with a curvature that varies linearly with travelled distance, manage to avoid this problem. Existing solutions usually only interpolate paths between configurations with zero curvatures [KM85] or use complex constructions of three clothoids to interpolate between two configurations with different curvatures [SS90, BP11a]. Methods like [SF96, FS04], also rely on clothoids, but since they are, in particular, able to deal with complex maneuvers that require the change of the direction of motion, they are usually complex.

It is also worth mentioning that, in the majority of literature, it is proclaimed that the evaluation of clothoids is computationally expensive, which motivates using splines or Bezier curves. As we will see in Section 5.4.2, clothoids are expressed in term of Fresnel integrals, for which very efficient numerical approximations are known [Mie00]. Therefore, we consider this problem to be a non-issue (or, at least, no more an issue on modern hardware).

Another interesting property of clothoids is that they correspond to the time-optimal trajectories of differential-drive robots (i.e., the paths followed by differential-drive robots when their wheels are driven at respectively their minimum and maximum acceleration, as a result, for instance, of bang-bang control.). For these various reasons, we choose to construct our paths out of clothoids.

5.4 Solution

We solve the interpolation problem in two steps, the first one being aimed at producing a path in which the absolute curvature is bounded at all points, and the second one modifying this path in order to now bound the rate of variation of curvature. In both steps, the interpolated path has to stay within safe zones in order to clear obstacles.

5.4.1 Bounding Curvature

The first main problem is that at the junction point between two non collinear adjacent segments of a broken line path, the curvature is infinite, which cause the robot to stop at this point.

In order to bound the absolute curvature throughout the path, we build a curve composed of straight line segments (with zero curvature) and circle arcs (with constant curvature), connected in such a way that continuity of the tangent vector is ensured everywhere. On such a curve, the curvature can be expressed as a piecewise constant function with respect to travelled distance.

We construct such a curve by computing, for each pair of adjacent segments ($[p_{i-1},p_i]$, $[p_i,p_{i+1}]$) a value ℓ_i specifying the distance from p_i at which the curve transitions from the segments to a circle arc. In other words, ℓ_i corresponds to the distance between p_i and

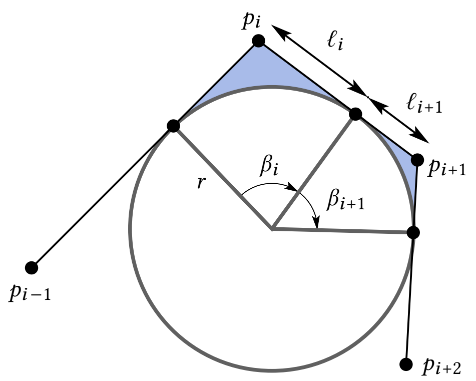


Figure 5.2 – Segments tangent to a common circle.

each point of tangency between that circle arc and the segments. Of course, in order to clear obstacles, it is necessary to have $\ell_i \leq c_i$ for all $i \in [1, n-1]$.

We compute ℓ_i by applying the following principle: If three or more consecutive segments are all tangent to a common circle, then the arcs that interpolate these segments must belong to that circle, provided that they are located within safe zones. This solution has the desirable property of keeping the curvature constant across two or more interpolation steps, which is likely to reduce the time needed to travel the path by a specific robot.

We now show how to carry out this computation. Consider a path in which all segments are tangent to a common circle of radius r. This situation is illustrated in Figure 5.2.

At the points p_i and p_{i+1}, one has respectively

$$\ell_i = r \left| \tan \frac{\beta_i}{2} \right|$$

and

$$\ell_{i+1} = r \left| \tan \frac{\beta_{i+1}}{2} \right|$$

Since, in this case, the constraint $\ell_i + \ell_{i+1} = |p_i p_{i+1}|$ is satisfied, we obtain

$$\ell_i = \frac{\tau_i \, |p_i p_{i+1}|}{\tau_i + \tau_{i+1}} \tag{5.1}$$

$$\ell_{i+1} = \frac{\tau_{i+1} \, |p_i p_{i+1}|}{\tau_i + \tau_{i+1}}, \tag{5.2}$$

where

$$\tau_i = \left|\tan \frac{\beta_i}{2}\right|$$

for all i.

Note that these expressions do not involve r, and that Equation 5.1 can be rewritten at the point p_i into

$$\ell_i = \frac{\tau_i \, |p_{i-1}p_i|}{\tau_{i-1} + \tau_i}. \tag{5.3}$$

For general paths, successive segments are not tangent to a common circle, and Equations 5.1 and 5.3 then provide different values for ℓ_i.

Our strategy is, for all $i \in [1, n-1]$, to define ℓ_i as the smallest value among those expressed by Equations 5.1 and 5.3, and the clearance parameter c_i.

This solution also applies to pairs of adjacent segments that turn in opposite directions; in such a case, small values of $|\beta_i|$ (which represent small changes of direction) lead to small circle arcs, and large values of $|\beta_i|$ to large arcs, which is geometrically sound.

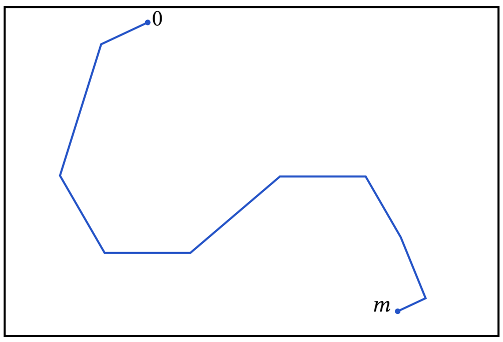
First Results

This bounding curvature step has two effects: it reduces the total length of the path, and it bounds, as most as possible, the curvature at all points of the path, while still clearing obstacles. These two effects naturally reduce the total time needed for following the path by a robot. This can be experimentally verified using the time-optimal speed profile developed in Chapter 4. Figures 5.3 and 5.4 show these results for one of the robots used in the context of the Eurobot contest.

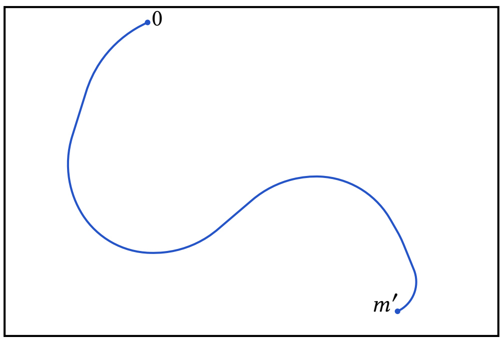
For this robot, due to its physical constraints, we need 22.5 s to follow the 3.57 m of the broken line, but only 11.53 s to follow the 3.34 m of the same path with bounded curvature. This is a great improvement, but we see on Figure 5.4 that the robot still does need to nearly stop every time the curvature abruptly changes, which is inefficient.

5.4.2 Bounding Curvature Variations

We now turn to the problem of modifying the path produced at the previous step, which has a curvature that is piecewise constant, into one in which the rate of variation of curvature with respect to travelled distance remains bounded. Clearing obstacles is achieved by constraining the interpolation to remain within safe zones. The resulting path must have a curvature that is continuous, bounded, and of bounded slope, at all points.

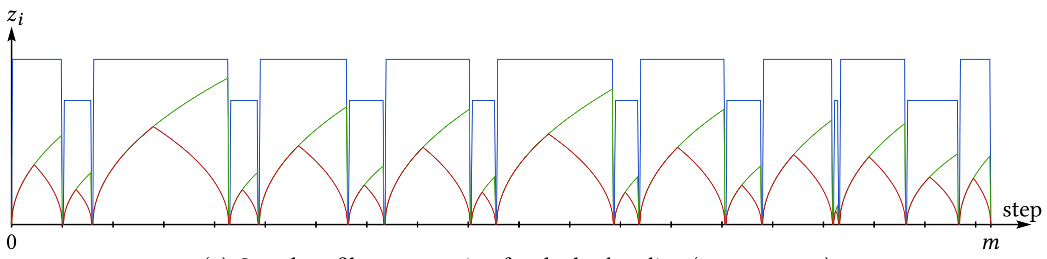


(a) Broken line path (3.57 m).

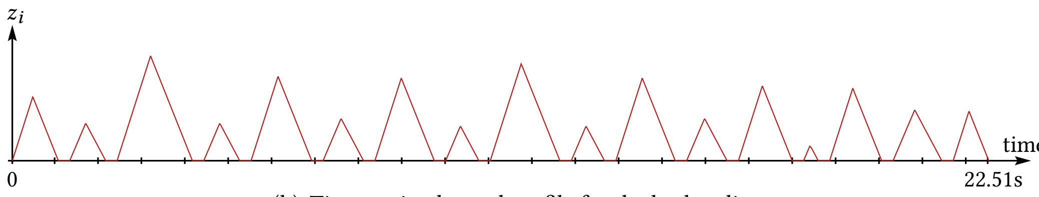


(b) Same path with bounded curvature (3.34 m).

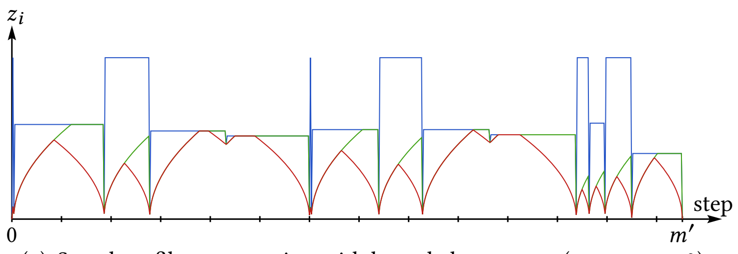
Figure 5.3 – Example of path with bounded curvature.



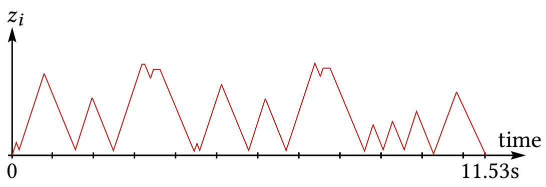
(a) Speed profile computation for the broken line ($v_0 = v_{fm} = 0$).



(b) Time-optimal speed profile for the broken line.



(c) Speed profile computation with bounded curvature ($v_0 = v_{fm} = 0$).



(d) Time-optimal speed profile with bounded curvature.

Figure 5.4 – Associated speed profiles.

As already discussed, we construct such curves out of *clothoids*, which, as already said, correspond to the time-optimal trajectories of differential-drive robots (i.e., the paths followed by differential-drive robots when their wheels are driven at respectively their minimum and maximum acceleration).

Clothoids

Clothoids are formally defined as curves with a curvature that varies linearly with travelled distance.

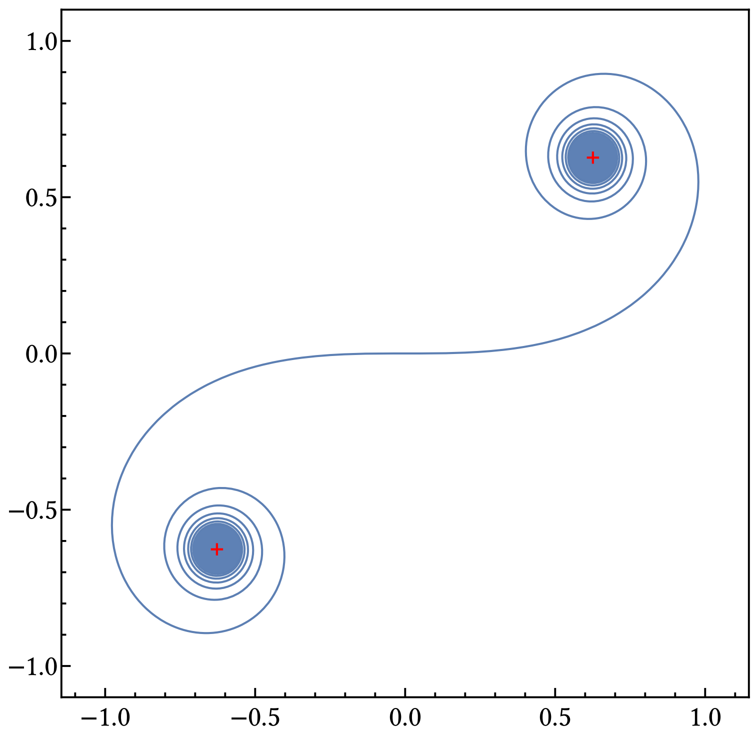


Figure 5.5 – The unit clothoid[3].

The curvature κ of the clothoid is given by

$$\kappa(s) = \kappa_0 + cs, \tag{5.4}$$

where κ_0 is the initial curvature, c is the clothoid *sharpness* (i.e., rate of change of curvature), and s denotes the travelled distance.

[3] The curve converges to the points marked with a red cross, as s tends to positive or negative infinity.

Recall from Chapter 3 that the curvature of a path is defined by

$$\kappa(t) = \frac{d\theta}{ds},$$

where $\theta(s)$ denotes the tangential angle of the path, we have

$$\theta(s) = \theta_0 + \kappa_0 s + \frac{1}{2}cs^2, \tag{5.5}$$

where θ_0 is the initial tangential angle.

The general[4] parametric expression of a clothoid is then

$$\begin{cases} x(s) = x_0 + \displaystyle\int_0^s \cos\left(\frac{\pi}{2} + \theta_0 + \kappa_0 u + \frac{1}{2}cu^2\right) du \\ y(s) = y_0 + \displaystyle\int_0^s \sin\left(\frac{\pi}{2} + \theta_0 + \kappa_0 u + \frac{1}{2}cu^2\right) du. \end{cases} \tag{5.6}$$

Figure 5.5 shows the unit clothoid ($x_0 = y_0 = \kappa_0 = 0,\ \theta_0 = -\frac{\pi}{2},\ c = 1$).

The coordinates of the points visited by a clothoid are thus expressed in terms of the integrals $S(x)$ and $C(x)$

$$S(x) = \int_0^x \sin t^2\, dt,$$

$$C(x) = \int_0^x \cos t^2\, dt,$$

which are known as the Fresnel integrals.

These integrals cannot be evaluated analytically, but, as already mentioned, can be very efficiently approximated [Mie00].

Note that, from Equations 2.5, 2.6, 2.18, 2.19 and 2.20, one can easily establish that a differential robot with its wheels driven at respectively their minimum and maximum acceleration actually follows a clothoid curve.

Computing a Pair of Clothoids

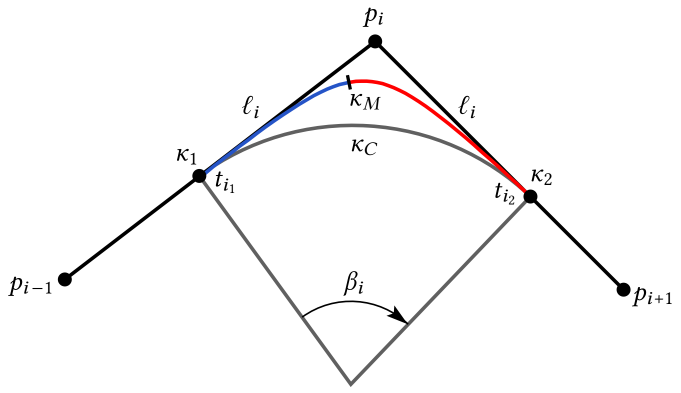
We now show how we use clothoids in order to construct a smooth path that clears obstacles. Consider a circle arc with curvature κ_C, interpolating two successive line segments $[p_{i-1}, p_i]$ and $[p_i, p_{i+1}]$ within their safe zone. Let t_{i_1} and t_{i_2} denote the points of tangency between

[4] Note that, as in the previous chapters, the orientation is chosen such that $\theta = 0$ corresponds to a direction that follows the y-axis.

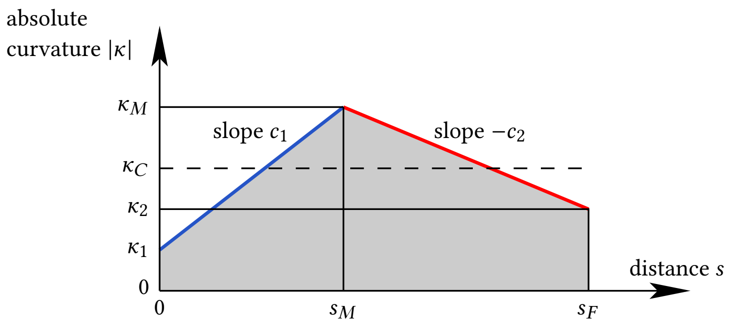
this circular arc and receptively the lines segment $[p_{i-1}, p_i]$ and $[p_i, p_{i+1}]$. Let also θ_1 and θ_2 denote the tangential angles at these two points. Assuming w.l.o.g. $\kappa_C > 0$ (the case $\kappa_C < 0$ is handled symmetrically), we have established the following result.

Theorem 5.4.1. *For every rotation angle $\beta_i \in]0, \frac{\pi}{2}]$ and for every pair of curvatures κ_1, κ_2 such that $0 \le \kappa_1 < \kappa_C$ and $0 \le \kappa_2 < \kappa_C$, there exist two clothoids arcs moving respectively from the curvatures κ_1 to κ_M and from κ_M to κ_2, with $\kappa_M > \kappa_C$, the concatenation of which interpolates the path from t_{i_1} to t_{i_2} within the safe zone, with initial and final tangential angles of respectively θ_1 and $\theta_2 = \theta_1 + \beta_i$ and with continuity of the tangent vector at the junction point between the two curves. The parameters of these two clothoids arcs are uniquely determined by κ_1, κ_2, κ_C, and the rotation angle β_i.*

Our method for characterizing the two clothoids arcs consists in reasoning on a diagram expressing the curvature of the interpolated path as a function of travelled distance. The problem is illustrated in Figure 5.6 (exaggerating the curvatures in order to make the interpolated path stand out from the circle arc). A proof of this theorem is provided in Appendix B.



(a) Path geometry.



(b) Curvature graph.

Figure 5.6 – Interpolation with two clothoid arcs.

Let s_M denote the distance travelled along the first arc, and s_F the total distance travelled over both arcs. The linear rate of variation of curvature for these arcs are respectively denoted by c_1 and $-c_2$. In the graph depicted in Figure 5.6(b), the grayed out area below the curvature line must be equal to β_i.

Note that κ_M, c_1 and c_2 can be expressed in terms of the other variables, since, from Equation 5.4, one has

$$\kappa_M = \kappa_1 + c_1 s_M = \kappa_2 + c_2 \left(s_F - s_M\right),$$

and from Equation 5.5, one has

$$\kappa_1 s_M + \frac{1}{2} c_1 s_M^2 = \beta_i - \kappa_2 \left(s_F - s_M\right) - \frac{1}{2} c_2 \left(s_F - s_M\right)^2 .$$

It thus remains to compute s_M and s_F given κ_1, κ_2, κ_C and β_i.

We solve this problem numerically, observing empirically that the initial estimates

$$\frac{s_F}{s_M} = 1 + \frac{\kappa_C - \kappa_1}{\kappa_C - \kappa_2}, \quad s_F = \frac{\beta_i}{\kappa_C}$$

lead to very fast convergence with Newton-Raphson's method[5].

In practice, we first perform a variable change operation by defining

$$a = \frac{c_1 - c_2}{c_1 + c_2}$$

and

$$c_m = \frac{c_1 + c_2}{2},$$

and then carry out the search over the variables a and s_F (while ensuring that $c_1 > 0$ and $c_2 > 0$ in order to always turn in the same direction).

Intuitively, a is a measure of the asymmetry between the two clothoids arcs, and remains small when κ_1 and κ_2 are reasonably balanced.

[5] By exhaustively computing all the possible cases using a reasonably small discretization (i.e., 10^{-3}rad for β_i and 10^{-4}m^{-1} for curvatures), we have established that for $\beta_i \in]0, \frac{\pi}{2}]$, $\kappa_C \in [0.01, 1000]$, $0 \leq \kappa_1 < \kappa_C$ and $0 \leq \kappa_2 < \kappa_C$, we need at most 4 iterations to converge with an absolute position error less than 10^{-8}m and an absolute orientation error less than 10^{-8}rad. Note that this even drops to at most 2 iterations for $\kappa_1 \leq 0.9\kappa_C$ and $\kappa_2 \leq 0.9\kappa_C$, which are always satisfied in our Eurobot application (see next subsection).

The value of s_F is confined to an interval with a lower bound

$$\frac{\beta_i}{\kappa_C}$$

equal to the distance travelled on the circle arc. Its upper bound corresponds to the minimum value among

$$\frac{2}{\kappa_C}\tan\frac{\beta_i}{2},$$

which is the combined length of the line segments, and

$$\frac{\beta_i}{\min(\kappa_1,\kappa_2)},$$

which is the largest distance that can be travelled without lowering the curvature below κ_1 and κ_2.

It is also worth mentioning that, in the particular case $\kappa_1 = \kappa_2 = 0$, this procedure can be simplified into one that does not rely on approximations [BP11a], except for evaluating Fresnel integrals.

Indeed, w.l.o.g., we can assume that for the first clothoid segment we have $\theta_0 = 0$. Because of symmetry, we also have $c_1 = c_2$ (i.e., $a = 0$) and the tangential angle at the end of the first clothoid segment is given by

$$\theta_M = \frac{\beta_i}{2}.$$

From Equation 5.5, for a clothoid with a rate of variation of curvature equal to 1, we have

$$s'_F = 2s'_M = 2\sqrt{\beta_i},$$

which we can use to compute the curvature κ'_C of the corresponding circular arc.

From Equations 5.5 and 5.6, we can observe that C defined as

$$C^2 = \frac{1}{c},$$

acts as a scaling factor for the clothoid.

Therefore, since we need a circular arc of κ_C instead of κ'_C, we finally have

$$c_m = \left(\frac{\kappa_C}{\kappa'_C}\right)^2, \quad s_F = 2\sqrt{\beta_i}\left(\frac{\kappa'_C}{\kappa_C}\right).$$

Choosing Curvature at the Extremities

Finally, in order to apply Theorem 5.4.1, it remains to choose values for κ_1 and κ_2 at the extremities of interpolated curves. We use the following strategy:

- At the junction between a straight line segment and an arc, or between two arcs turning in opposite directions, a natural choice is $\kappa_i = 0$.

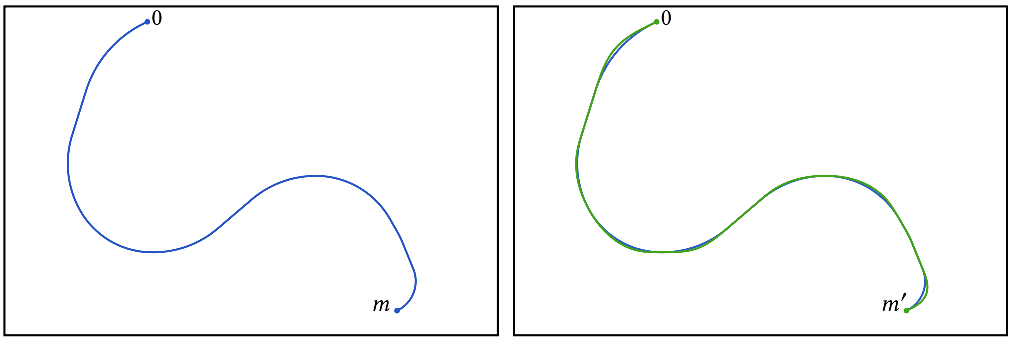
- If we need to connect two arcs turning in the same direction, with respective curvatures κ_{C1} and κ_{C2} which we assume w.l.o.g. to be positive, we have to choose a curvature κ_i that satisfies $\kappa_i < \min(\kappa_{C1}, \kappa_{C2})$.

  This can be achieved by defining $\kappa_i = f \min(\kappa_{C1}, \kappa_{C2})$, where $0 < f < 1$ is a reduction factor that can be arbitrarily chosen. For the Eurobot application, we have observed that selecting $f = 0.75$ leads to paths along which both the curvature and variation of curvature stay within acceptable bounds.

  In fact, we can observe that increasing f leads to shorter paths with smaller intermediate maximum absolute curvatures (i.e., κ_M), but with bigger maximum curvature variations (i.e., $\max(c_1, c_2)$). The real optimal value for f depends of the physical constraints of the robot, but selecting $f = 0.75$ seems to be a good trade-off for the various robots considered in the Eurobot application.

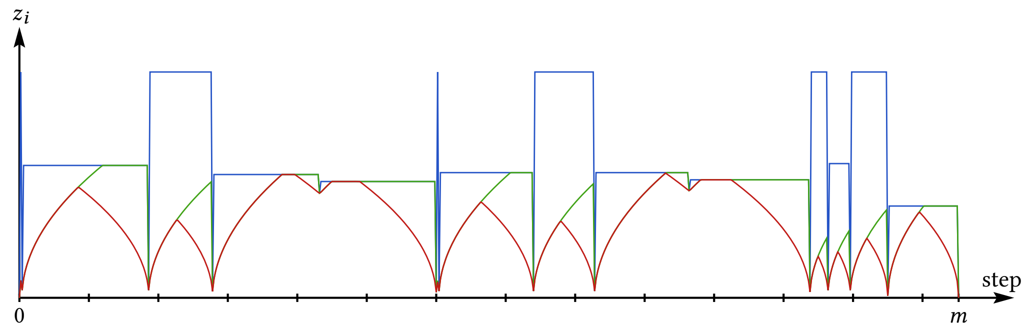
5.4.3 Bounding Curvature Variations Results

As for the bounding curvature step, one can experimentally verify, using the time-optimal speed profile developed in Chapter 4, that our method for interpolating paths with clothoids usually reduces the total time needed for following the paths by a robot (even if the interpolated path is slightly longer). Figures 5.7 and 5.8 show these results for one of the robots used in the context of Eurobot.

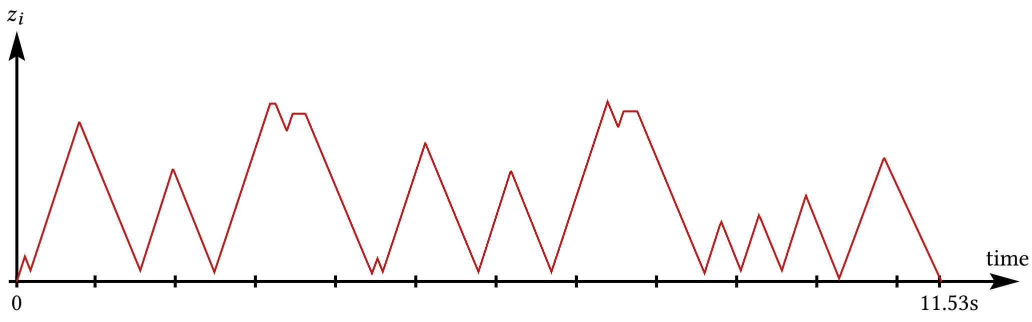


(a) Path with bounded curvature (3.34 m). (b) Same path (in green) with additional bounded curvature variation (3.39 m).

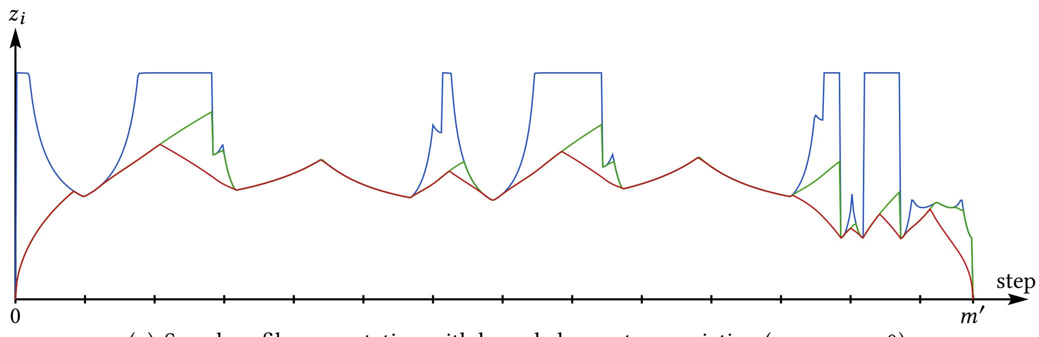
Figure 5.7 – Example of path with bounded curvature and bounded curvature variation.



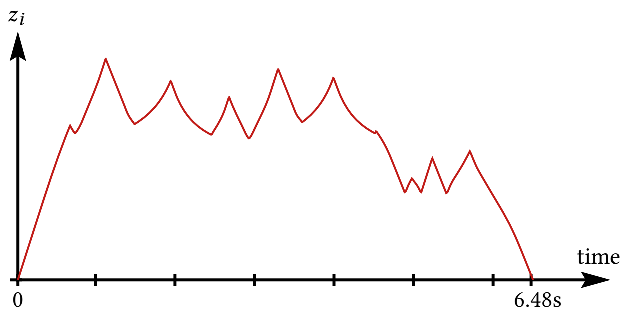
(a) Speed profile computation with bounded curvature ($v_0 = v_{fm} = 0$).



(b) Time-optimal speed profile with bounded curvature.



(c) Speed profile computation with bounded curvature variation ($v_0 = v_{fm} = 0$).



(d) Time-optimal speed profile with bounded curvature variation.

Figure 5.8 – Associated speed profiles.

For this robot, due to its physical constraints, we need (as in Section 5.4.1) 11.53 s to follow a path of length 3.34 m with bounded curvature (i.e., composed of straight lines and circle arcs), but only 6.48 s to follow the same path now smoothed out into one of length 3.39 m with bounded curvature variation, which is again a great improvement.

Note that this path was computed using $f = 0.75$, and that more precisely we need 6.478 s to follow its 3.386 m. If we use instead, for example, $f = 0.5$, we will need 6.510 s to follow the 3.394 m of this new interpolated path, and if we use $f = 0.9$, we will need 6.780 s to follow 3.380 m.

A large number of other tests have been performed on hundreds of trajectories. These tests show that, in order to reduce as most as possible the travel time in most of the cases, $f = 0.75$ is actually a good trade-off for the robots considered in our application.

Finally, it also worth noting that the computational cost of our complete interpolation procedure is linear in the number of segments in the broken line path.

5.4.4 Car-Like Platforms

For a car-like platform, as the curvature of the physically feasible paths is usually bounded, it could be impossible to bound the curvature of the broken line below this limit in the first step. In this case, as the first step aims at reducing as most as possible the curvature at all points of the path, no feasible path can be computed for the car-like platform (i.e., the clearance parameters are not sufficient to compute a feasible path).

In the second step, we need to ensure that k_M stays below the curvature limit imposed by the platform, this can be achieved, for instance, by increasing f and/or by interleaving intermediate segments when necessary.

5.5 Experimental Results

Compared with methods such as [SS90, FS04, BP11a] that also rely on clothoids for interpolating paths, our approach of joining only two arcs of clothoids for moving from one curvature to another has the advantage of being simpler and computationally cheaper. The generated curves are clearly not guaranteed to be optimal (i.e., the ones the correspond the fastest associated speed-profiles), but optimality is not guaranteed either by other methods.

In order to illustrate the efficiency of our method, we report in Figure 5.9 the time needed for running the interpolation algorithm on a few sample trajectories experienced

	Discretized points	Synthesis time	Discretization time	Total time
1	292	7.6 μs	30.1 μs	37.7 μs
2	521	11.8 μs	49.8 μs	61.6 μs
3	630	16.5 μs	63.9 μs	80.4 μs
4	656	12.8 μs	68.1 μs	80.9 μs
5	690	22.4 μs	85.0 μs	107.4 μs

Figure 5.9 – Experimental results.

in the Eurobot application. We distinguish the costs of the curve synthesis and path discretization (i.e., as described in Chapter 3). The total computational cost typically amounts to a hundred of microseconds of CPU time on an i5-460M processor running at 2.53 GHz, which is significantly faster than techniques such as [FS04, BP11a].

Note that the fifth path considered in this experiment is the same as the one in Figure 5.7.

CHAPTER 6

Path Planning

This chapter addresses path planning for nonholonomic robots moving in two-dimensional space. The problem consists in computing a sequence of line segments that leads from the current configuration of the robot to a target location, while avoiding a given set of obstacles. This chapter discusses the construction of such paths in the case of a set of point obstacles. The method relies on a search in a Voronoi diagram (or, more specifically, in its dual graph, the Delaunay triangulation) that characterizes the possible ways of moving around obstacles, followed by a string-pulling procedure, based on a generalization of the funnel algorithm, aimed at improving the resulting path. The resulting sequences of line segments can directly be used as inputs to the algorithm described in Chapter 5, in order to produce short smooth paths that avoid obstacles.

6.1 Introduction

We consider the general problem of planning the motion of an autonomous nonholonomic robot in two-dimensional space, the goal being to reach a given configuration while avoiding some set of obstacles. As already mentioned, this problem is difficult and often computationally expensive.

For many applications, such as Eurobot, the ability of the robot to react quickly is crucial, and a strong emphasis must be placed on the efficiency of motion planning, which often has to be carried out with the limited amount of computing power available on the robot. The objective is then not necessarily to find an optimal solution to the motion planning problem, but to develop a method that can very quickly synthesize a trajectory for reaching the target in acceptable time, and that is consistent with the physical limitations of the robot.

In order to develop an efficient method, the general idea that was followed in this work, was to divide the motion problem in two distinct subproblems. The first one is aimed at finding a path that avoids obstacles and manages to reach the destination, without taking into account nonholonomic constraints. Such a path takes the form of a sequence of straight line segments that clears the obstacles at a specified safety distance. The second step is to convert this path into a smooth trajectory that can be followed by the robot, taking into account its physical constraints.

Chapters 4 and 5 already discuss in details how to carry out efficiently the later step. In this chapter, we focus on the former, introducing a method for computing piecewise linear paths that lead from an origin to a destination while avoiding obstacles with a given clearance.

6.2 Related Work

This chapter studies path planning for robots moving in two-dimensional space in the presence of obstacles. There exist a large number of methods for solving this problem; these methods can broadly be classified into three major approaches: cell decomposition, roadmap and potential field [Lat91, CLH+05, LaV06, SSVO09]. They all have different pros and cons; none of them solves the problem completely, optimally, and efficiently.

6.2.1 Cell Decomposition

Cell decomposition methods consist in decomposing the set C_{free} of the configurations that stand clear from obstacles, into simpler regions called *cells*. If this decomposition can be made in such a way that paths can easily be generated between any two configurations in a cell, then motion planning reduces to finding paths in a graph that represents the connectivity between those cells.

The decomposition of C_{free} into cells can be performed exactly, for instance using the trapezoidal decomposition method [Cha87, BCKO08], suited for convex polygonal obstacles, or by introducing approximations, such as the quadtree decomposition algorithm described in [KD85].

It has been established that the exact cell decomposition of a set of disjoint convex polygonal obstacles can be computed in $O(n \log n)$ time, where n is the number of edges in the obstacles representation [BCKO08]. This method has the advantage of being complete, but it does not easily extend into one that takes into account the size of the robot. Approximate methods, on the other hand, may not be complete but, by increasing the grid resolution, can produce results that are nearly optimal at arbitrary precision levels, at the expense of a high computation time.

6.2.2 Roadmap

The main principle behind roadmap methods is to capture the connectivity between the configurations in C_{free} within a graph called the *roadmap*. The path planning problem then reduces to computing three subpaths: one that leads from the initial configuration to a node of the roadmap, another moving within the roadmap, and a final one connecting a node of the roadmap with the target configuration.

There exist several methods to build a roadmap. A first approach is to construct a *visibility graph*, the nodes of which correspond to points located at the boundaries of obstacles, with an edge connecting two nodes iff the line segment that connects the underlying points does not conflict with obstacles (i.e., is entirely contained in C_{free}). In the case of polygonal obstacles, the nodes correspond to their vertices.

For more general convex sets, one can define edges as the free bitangents between obstacles, and the vertices as the points of tangency. The visibility graph can be computed in $O(k + n \log n)$ time, where k is the number of edges in the graph and n the number of obstacles [PV95]. The main drawback of this approach is the fact that a visibility graph can admit as many as $O(n^2)$ edges.

If obstacles have to be cleared at a given safety distance (i.e., for instance, when the robot has non-zero measure), then the visibility graph can be constructed after dilating obstacles, but one then needs to check whether these are still disjoint from each other after the operation. A procedure is introduced in [WBH05] for constructing a roadmap that is complete and optimal (in terms of travelled distance), with respect of a given clearance. The computation time is $O(n^2 \log n)$ where n is the total number of vertices, which is not always efficient enough for our intended applications.

Another efficient strategy for obtaining a suitable roadmap is to build the *Voronoi diagram* of the set of obstacles [ÓY85, BCKO08]. In the case of a set of individual points (called *sites*), this diagram partitions the plane into convex cells that are each associated to one site, containing all the points that are closer (with respect to Euclidean distance) to this site than the others. Voronoi diagrams can be generalized to more complex types of sites such as line segments or polygons; in this case, borders of Voronoi cells are no longer necessarily line segments, but can also take the form of parabolic arcs.

Other names for methods based on Voronoi diagrams are *Retraction Methods* or *Maximum-Clearance Roadmaps*. In a Voronoi diagram, an edge separating two cells clears, by definition, the corresponding sites at the largest possible distance. As a consequence, Voronoi-based path planning methods are complete, and can easily be extended so as to take into account the size of the robot. On the other hand, paths extracted from roadmap graphs are generally not optimal, and need to be further refined.

Finally, there exist other strategies for obtaining roadmaps, such as Probabilistic Roadmaps [O+92, KSLO96, GO04] and Rapidly-exploring Random Trees [LaV98, KWP+11]. These methods are usually very efficient as their cost mainly depends on the complexity of the generated paths, but do not generally produce optimal paths, and are only probabilistically complete.

6.2.3 Potential Field

The idea behind this approach, first mentioned in [Kha86], is to view the robot as a particle moving under the influence of an artificial potential field. If the target location generates a strong attractive potential, and the obstacles produce a repulsive potential in order to avoid collisions, a path leading to the target can be found by following the direction of the fastest descent of the potential.

This reduces the path planning problem to an optimization problem and therefore we can use all the classical optimization methods to solve it (e.g., simulated annealing, genetic algorithm, ant colony, …). This approach can be very efficient and can sometimes be computed in real time. However, like other optimization techniques, those methods usually rely on parameters that must be finely tuned to the actual application [KB91].

6.3 Path Planning Algorithm

This chapter tackles the problem of generating piecewise linear paths for robots moving in a region of two-dimensional space constrained by a finite set of obstacles. We consider robots that have a cylindrical shape, i.e., a robot at a position P occupies a region of the plane that corresponds to a disk of diameter d centered in P, where d is a parameter of the robot.

The described method is based on the Voronoi diagram roadmap approach outlined in Section 6.2.2. As discussed in [For87, AK00], Voronoi diagrams can efficiently be computed in $O(n \log n)$ time and $O(n)$ space, where n is the number of sites.

Note that dealing with sites that are more complex than single points can be difficult in practice. For many applications, this problem can, for instance, be avoided by approximating such sites by finite sets of points. In Chapter 7, we will present an original approach to tackle this problem.

6.3.1 Voronoi Diagram

An edge separating two Voronoi cells clears the corresponding sites at the largest possible distance. Therefore, in order to reason about the possible moves of a robot around obstacles, we only have to check the minimal distance between those edges and the obstacles. The graph constructed from all the separating edges (i.e., the borders of all the Voronoi cells) then forms a complete and valid roadmap for our path planning problem [BCKO08].

6.3.1.1 Voronoi Diagram of a Set of Points

The simplest case is the Voronoi diagram of a finite set of points in an Euclidean plane. In this case, each site is simply a point, and its corresponding Voronoi cell is a convex polygon and contains all the points that are closer (with respect to Euclidean distance) to this site than the others.

An example of Voronoi diagram is depicted in Figure 6.1; the sites (i.e., the points) are represented in blue and the borders of the Voronoi cells in red.

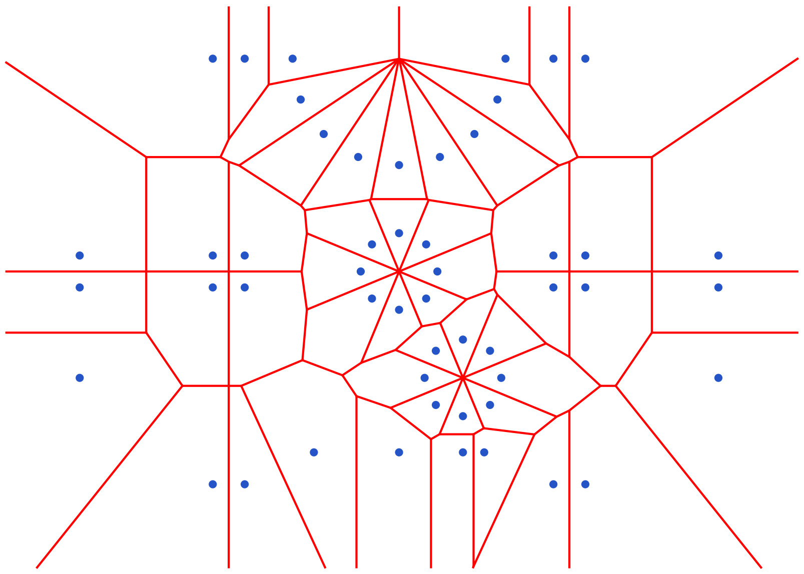


Figure 6.1 – Example of Voronoi diagram.

6.3.1.2 Approximation of Voronoi Diagram

Voronoi diagrams are easy to obtain for obstacles represented by single points. In the case of polygonal obstacles, the main drawback is that the edges of the Voronoi cells can take the form of parabolic arcs, that are more tricky to work with than line segments.

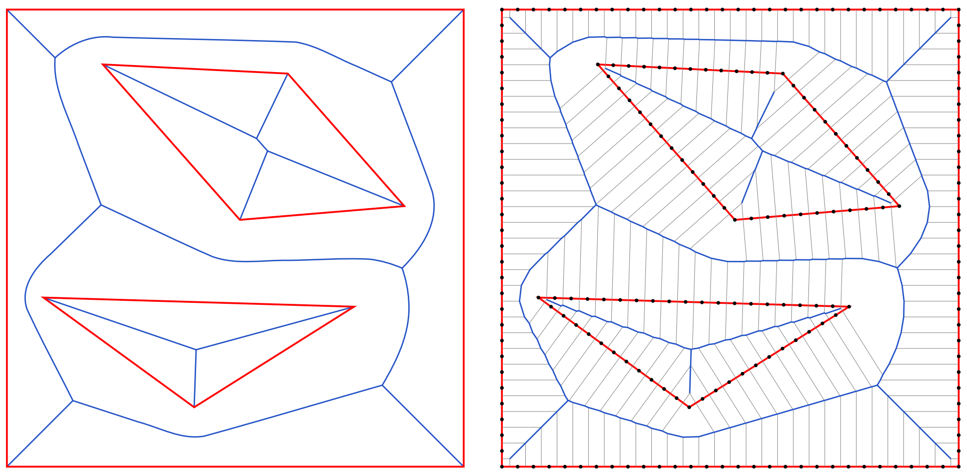

(a) Generalized Voronoi diagram. (b) Diagram approximation.

Figure 6.2 – Example of discretized Voronoi diagram.

A workaround consists in approximating the diagram by discretizing the obstacles. It is shown in [HIKL+99] that good approximations can be obtained by placing sites at the boundaries of obstacles, and then deleting the Voronoi edges that appear between different sites of a common obstacle (see Figure 6.2). The remaining issue is to approximate sufficiently precisely these shapes with finite sets of points.

In order to be able to compute efficiently a roadmap, the goal is to create as few sites as possible without sacrificing too much accuracy. Consider for instance, as in Figure 6.3(a), the case of two obstacles respectively represented by a straight line ℓ, and a point p that does not belong to ℓ. If the center of the robot needs to clear the obstacles at a distance c (which is equivalent to reasoning about a robot of diameter $d = 2c$ that can graze obstacles), there exists a path moving between this pair of obstacles provided that the distance between p and ℓ is at least equal to $2c$.

Assume now that the line is approximated by discrete points sampled at the resolution s. By remaining at a distance greater or equal to c from those points, it may now be possible to have a path that only clears p and ℓ at the distance $\frac{\sqrt{16c^2-s^2}}{4}$. This worst case, depicted in Figure 6.3(b), corresponds to p forming an isosceles triangle with the two nearest discrete points on ℓ.

It follows from this expression that choosing $s = \frac{c}{2}$ leads to a relative error that is less than 1%. This approximation is usually precise enough for a lot of practical applications, since an additional safety margin is usually required anyway to take into account positioning or sensor errors.

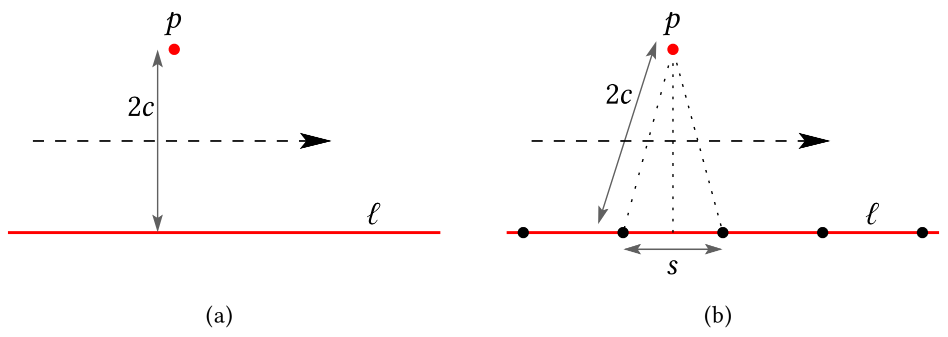


Figure 6.3 – Approximation error.

It is worth mentioning that the shape of the paths obtained with this discretization of obstacles does not always match those found in exact Voronoi diagrams. For instance, a path going between two parallel lines sampled at uniform resolution may take the form of a jagged line. This is not at all problematic for our path planning method, since such paths will be smoothed out later in the procedure, by the string-pulling operation that will be discussed in Section 6.3.5.

6.3.2 Delaunay Triangulation

Consider a finite set $Obst = \{A_1, A_2, \ldots, A_p\}$ of points representing obstacles that must be avoided. A *triangulation* of these points is a finite set $\{T_1, T_2, \ldots, T_q\}$ of triangles taking their vertices in *Obst*, such that their union exactly covers the convex hull of A_1, A_2, $\ldots A_p$, and the intersection between any pair of triangles can either be empty, equal to a single vertex A_i, or equal to a segment $[A_iA_j]$ linking two points in *Obst*.

In addition, such a triangulation is said to be *Delaunay* if, for every triangle T_i, all the points in *Obst* are located outside or on the circumcircle of T_i [Ber04].

It has long been known that the Delaunay triangulation of the finite set of points *Obst* corresponds[1] to the dual graph of the Voronoi diagram for *Obst* [BCKO08].

As depicted on Figure 6.4, the Delaunay Triangulation is obtained from the Voronoi diagram by connecting sites in *Obst* iff they are located in adjacent Voronoi cells. Reciprocally, the Voronoi diagram can be obtained from the Delaunay triangulation by connecting the centers of the circumcircles of the triangles that share a common edge.

[1] Providing that *Obst* contains at least three points that are not collinear. Note also that, if four or more points in *Obst* are cocyclic (i.e., lie on a same circle), then the Delaunay triangulation is not unique.

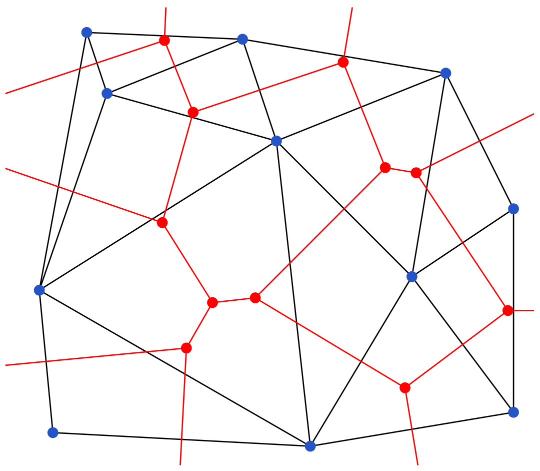


Figure 6.4 – Duality between the Voronoi diagram and the Delaunay triangulation. (Obstacles are represented by blue dots, the triangulation is represented in black, the centers of the circumcircles are represented by red dots and the Voronoi diagram is represented in red.)

Thanks to this duality argument, the exact same roadmap (as from the Voronoi diagram), can be constructed starting from a *Delaunay triangulation* of the set of points. Therefore, since efficient Voronoi (of points) implementations often first create the Delaunay triangulation and use it to generate the Voronoi diagram in $O(n)$ time (where n is the number of points) [GKS92, SLYD14], and since reasoning with triangles will turn out to be helpful for the rest of this work, from now on we will only discuss Delaunay triangulations.

6.3.3 Roadmap

As seen in the previous subsection, from a Delaunay triangulation of a set *Obst* of obstacles, one can extract a roadmap representing the possible paths around them. This is done by building a graph G whose nodes correspond to the triangles T_i (technically, to the center of their circumcircle), and in which two nodes are linked if and only if their underlying triangles T_i and T_j share a common edge.

An interesting property of this graph is that it can be used for reasoning about the possible moves of a cylindrical robot with an arbitrary diameter d, even though the graph is defined independently from the robot dimension. Let us define a *feasible position* P of the robot as a location (of the center of the robot) at which none of its interior points conflicts with points in *Obst*, in other words, such that $|P - A_i| \geq d/2$ for all i. A *feasible path* is defined as a path that only visits feasible positions. A *feasible triangle* is one that contains at least one feasible point.

Theorem 6.3.1. *There exists a path in G from a feasible triangle T_i to another one T_j that traverses only edges of length at least equal to d iff there exists a feasible path leading from a point in T_i to a point in T_j, for a robot of diameter d.*

This theorem intuitively expresses that the feasible paths that can be followed around the obstacles are exactly represented by the paths of G, being careful of only traversing edges that have a length consistent with the size[2] of the robot. The graph G thus represents a roadmap that can be searched for paths leading from a triangle to another, using for instance algorithms such as Dijkstra's or A* [WW07], and weighting, for example, the arcs of the graph with the distance between the centers of the circumcircles of the underlying triangles of their two nodes.

It is worth mentioning that a path of G that visits a sequence of triangles $T_{i_1}, T_{i_2}, T_{i_3}, \ldots$ does not always translate into a feasible path of the robot that exactly follows this sequence of triangles. It may indeed be the case that moving from T_{i_k} to $T_{i_{k+1}}$ requires to pass through an intermediate triangle that is not represented in the path, which is not problematic. However, the sequence of triangles successively visited by any feasible path must necessarily correspond to a path that exists in G.

6.3.4 Origin and Destination

The origin and the destination location do not, in general, correspond to the center of the circumcircle of triangles in the Delaunay triangulation. Therefore, in order to be able to use the roadmap that we have built for discovering paths with arbitrary origin and destination locations, a first step is to connect these locations to the nodes of the graph.

Obviously, both the origin and destination locations must clear the obstacles by a distance that is a least equal to $d/2$ (i.e., these two positions must be feasible), otherwise the path planning problem does not admit a solution. Recall that the edges of the roadmap graph correspond exactly to the borders of the Voronoi cells of the obstacles, hence we only need to check the distance between a position and the underlying obstacle of its cell to efficiently determine whether this position is feasible.

If the origin and destination are feasible, then we start by first identifying, for each of these two points, the triangle[3] of the triangulation to which they belong. This operation can be performed efficiently, for instance, by a local search in the triangulation [DPT02].

[2] Note that undesirable edges, generated by the approximation of obstacles by a finite set of points, have a length inferior or equal to $\frac{c}{2} = \frac{d}{4}$ (if we use the discretization step $s = \frac{c}{2}$), therefore they are naturally not considered during this step.

[3] It is natural to require that the robot remains confined in an area fully delineated by external obstacles. Hence, a feasible position necessarily belong to the convex hull of the obstacles, and thus to a triangle of the triangulation.

An arbitrary origin or destination point can then be taken into account by connecting this point to the node (i.e., the center of the circumcircle) that corresponds to the triangle in which it is located. Intuitively, since the center of the circumcircle of a triangle (in a Delaunay triangulation) locally maximizes the distance to its nearest obstacles, following a straight line from a feasible position located in a triangle to the center of its circumcircle can only increase the distance to the nearest obstacle and therefore will always only visit feasible positions.

6.3.5 String Pulling

As explained in Section 6.3.3, Delaunay-based methods guarantee that a feasible path will be obtained whenever one exists. But paths $T_{i_1}, T_{i_2}, \ldots, T_{i_m}$ that lead from an origin O in T_{i_1} to a destination D in T_{i_m} extracted from the roadmap G, for a robot of diameter d, are usually far from being optimal. Technically, such paths are sequences of line segments that start from O, then visit the center of the circumcircle of each triangle in the sequence, and end at D.

In addition to the artefacts that might have been introduced by the approximations discussed in Section 6.3.1.2, such a computed path may contain lots of unneeded direction changes, and usually clears obstacles at unnecessarily large distances. This is not at all surprising, since the edges of the roadmap are such that they maximize the clearance to obstacles. After having computed a path, we thus need an additional optimization step aimed at simplifying this path and lowering its length, which usually also reduces the time needed for following it.

This problem can be solved in the following way. The sequence[4] of triangles $T_{i_2}, T_{i_3}, \ldots, T_{i_{m-1}}$ as well as the origin and destination points (located respectively in T_{i_1} and T_{i_m}) represents a *channel*, i.e., a triangulated region of the plane in which the robot can move from a triangle T_{i_k} to its successor $T_{i_{k+1}}$ by traversing their common edge. The vertices of the interiors triangles (i.e., those that do not contain O or D) that compose a channel precisely represent the obstacles that must be cleared when the robot moves along this channel. Figure 6.5 shows an origin and destination point, a sequence of triangles, and the resulting channel.

Note that, since each of the interior edges (i.e., the edges separating the triangles of the channel) is traversed in a particular direction, its endpoints can be associated to the left or the right (relatively to this direction). All the left endpoints form the left side of the channel, and symmetrically, all the right endpoints form the right side of the channel.

[4] Note that, since the shortest sequence of triangles is extracted from the roadmap using algorithms such as Dijkstra's or A*, this sequence never contain multiple occurrences of triangles.

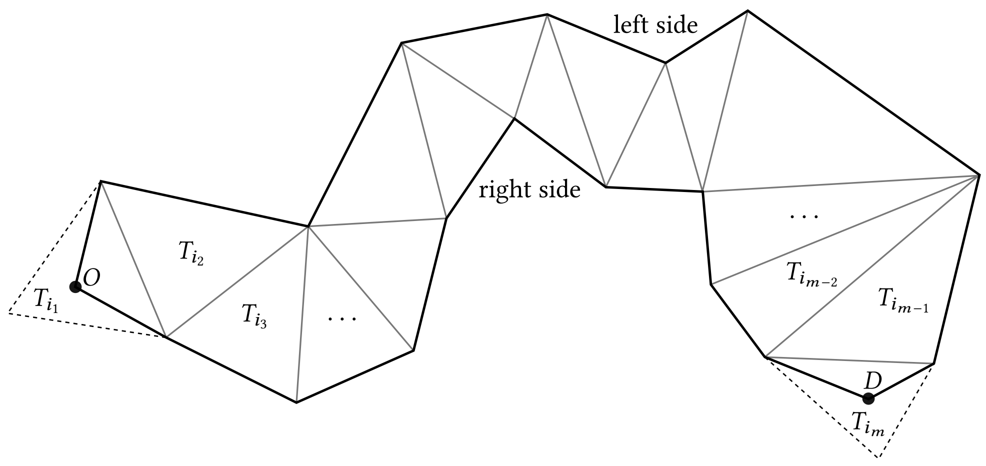


Figure 6.5 – Origin and destination locations, sequence of triangles and the resulting channel.

Given a channel obtained for a robot of diameter d, the shortest feasible path from origin to destination that follows this channel, i.e., such that each channel vertex is cleared at a distance at least equal to $d/2$, can be computed thanks to a simple generalization of the *funnel algorithm* [DB06, Dem07]. This operation conceptually consists in placing a solid disk of radius $d/2$ on each channel vertex, and of pulling an elastic cord taunt between these disks, from the origin to the destination.

6.3.5.1 Funnel Algorithm

The original funnel algorithm [Cha82, LP84, HS94] finds the shortest path within a channel composed of n triangles in $O(n)$ time. This algorithm considers that the moving object is a point (i.e., a robot with zero measure) and produces the shortest sequence of line segments from origin to destination that lies entirely in the channel.

The funnel algorithm works with three structures: the *path*, the *apex*, and the *funnel*. The path is the sequence of line segments forming the part of the shortest path currently known by the algorithm. The funnel consists of two series of line segments, one (on the right side) turning clockwise and one (on the left side) counterclockwise, which represent the area in which all shortest paths leading to the unprocessed part of the channel must lie. Finally, the apex is the point which joins the path and the funnel. These three structures are depicted in Figure 6.6.

The pseudocode for the funnel algorithm is given in Algorithm 6.1. At the start, the path is empty, the apex is set to the origin and the funnel corresponds to the two segments connecting the origin and the endpoints of the first interior edge.

```
Function FindShortestPath($m$, $edges[1 : m-1]$, $O$, $D$)
    if $m = 1$ then
        $path$.init($O$); $path$.append($D$); return $path$;
    end
    $path$.init($O$); $funnel$.init(); $funnel$.setApex($O$);
    ProcessVertex($path$, $funnel$, $edges[1]$.leftEndPoint(), $Left$);
    ProcessVertex($path$, $funnel$, $edges[1]$.rightEndPoint(), $Right$);
    for $i := 2, 3, \ldots, m-1$ do
        if $edges[i]$.leftEndPoint() = $edges[i-1]$.leftEndPoint() then
            ProcessVertex($path$, $funnel$, $edges[i]$.rightEndPoint(), $Right$);
        else
            ProcessVertex($path$, $funnel$, $edges[i]$.leftEndPoint(), $Left$);
        end
    end
    ProcessVertex($path$, $funnel$, $D$, $Left$);
    for each segment $[\overrightarrow{v_1 v_2}]$ in $funnel$ left side do
        $path$.append($v_2$);
    end
    return $path$;
Function ProcessVertex($path$, $funnel$, $v$, $side$)
    if $side = Left$ then
        loop
            $[\overrightarrow{v_1 v_2}]$ := last segment on $funnel$ left side;
            if $[\overrightarrow{v_1 v_2}]$ does not exist then
                $[\overrightarrow{v_1 v_2}]$ := first segment on $funnel$ right side;
                if $[\overrightarrow{v_1 v_2}]$ does not exist or $v$ is located on the left of $[\overrightarrow{v_1 v_2}]$ then
                    $funnel$.leftSide.append($v$);
                    break;
                else
                    $funnel$.rightSide.removeFirstVertex();
                    $funnel$.setApex($v_2$);
                    $path$.append($v_2$);
                end
            else
                if $v$ is located on the left of $[\overrightarrow{v_1 v_2}]$ then
                    $funnel$.leftSide.append($v$);
                    break;
                else
                    $funnel$.leftSide.removeLastVertex();
                end
            end
        end
    else if $side = Right$ then
        {symmetrical procedure}
    end
```

Algorithm 6.1 – Funnel algorithm.

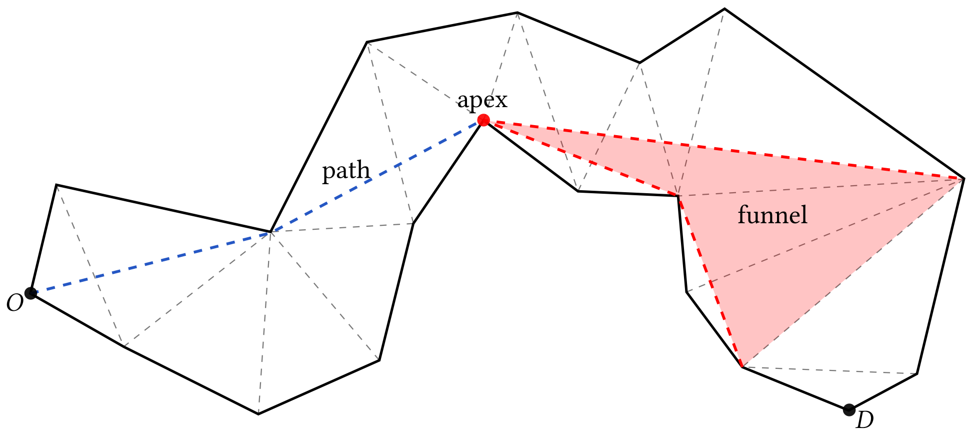


Figure 6.6 – Path, apex and funnel during the execution of the algorithm.

Then, each interior edge is processed one after the other. One endpoint of this edge corresponds to a vertex v that has not been processed yet, let us assume that this one is on the left side of the channel (the other case is handled symmetrically). Consider the last (oriented) segment on the left side of the funnel, v can be either on the left or on the right[5] of this segment.

If v is on the left, the current funnel is not going to be narrowed by v and we add v at the end of the left side of the funnel. But, if v is on the right, the current funnel is going to be narrowed by v. Thus, the last vertex on the left side of the funnel is removed, and we restart by checking on which side v is located with respect to the new last segment of the left side of the funnel.

If the left side of the funnel becomes empty (i.e., contains no more segment) during this operation, then, we consider the first segment on the right side of the funnel and check on which side v is located. If v is on its left, the funnel is going to be narrowed by v but we also know that the resulting channel will not be empty. Hence, we add v at the end of the left side of the funnel.

But, if the vertex is on its right, adding v to the left side of the funnel would result in an empty funnel. Thus, the first vertex (different from the apex) on the right side of the funnel is necessarily part of the shortest path. Therefore, it becomes the new apex, a segment connecting the old apex and the new one is added the path, and we restart by checking on which side v is located with respect to the new first segment of the right side of the funnel.

[5] If the vertex lies exactly on the line supported by this segment, we consider arbitrarily that the vertex is on the right of the segment.

Once the final interior edge of the channel has been processed, we proceed with the destination point, considering it as an edge of zero length and assuming arbitrarily, for instance, that this destination vertex is on the left side of the channel. Finally, we combine the path together with the left side of the funnel to form the shortest path.

It is straightforward to see that each time we check whether an unprocessed vertex is on the left or on the right of a segment, we add or remove one vertex to the funnel. As each vertex can not be added or removed more than once to the funnel, the number of operations of the algorithm is linear in the number of vertices, or equivalently to the number of triangles in the channel.

Figure 6.7 shows a representative example (i.e., that shows all the possibles cases) of the execution of the funnel algorithm.

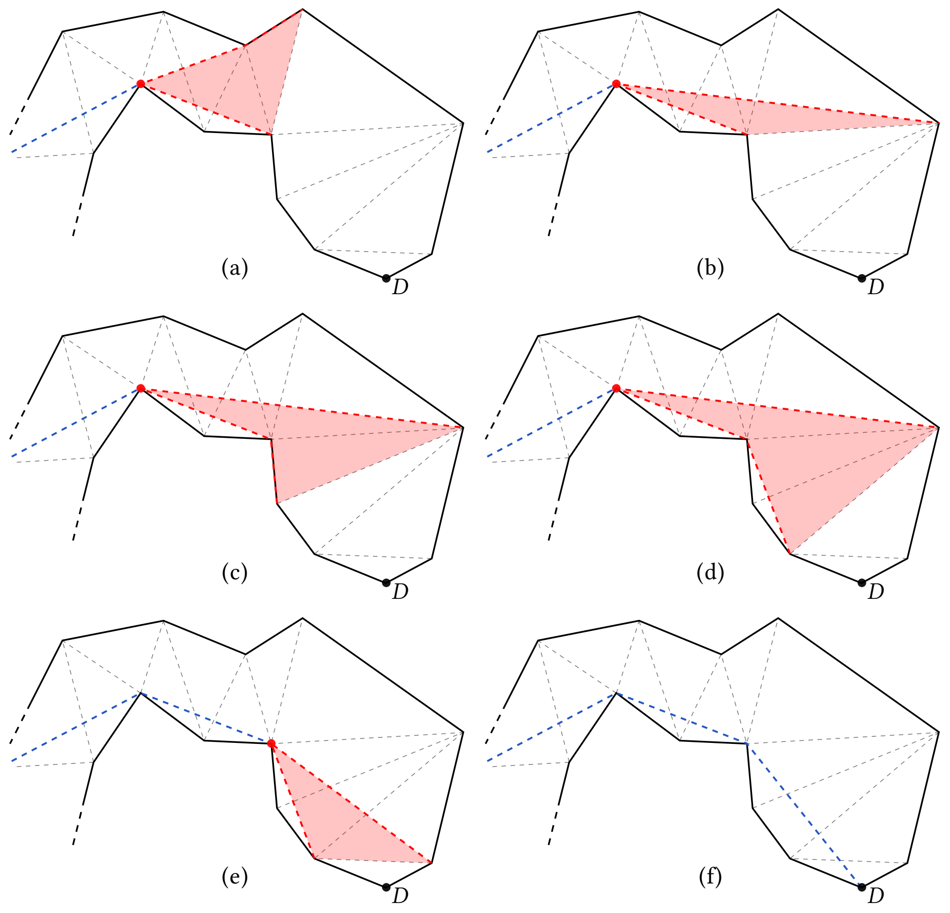


Figure 6.7 – Execution of the funnel algorithm.
(From (a) to (b), each transition corresponds to processing one new interior edge.)

Note that if the origin and destination points are located in the same triangle, then the number of edges to be traversed is obviously equal to zero, and the shortest path between the two is simply the segment that joins them.

6.3.5.2 Generalized Funnel Algorithm

We know now how to efficiently obtain the shortest path, between origin and destination within a channel, for a point robot (i.e., with a diameter $d = 0$). The more general case $d > 0$ can be handled by adapting the funnel algorithm. Intuitively, instead of searching for paths that connect vertices of the channel, one now needs to consider the segments that are tangent to disks of radius $d/2$ centered on these vertices (except for the origin and destination points, which still need to be considered as points). This generalization is illustrated in Figure 6.8.

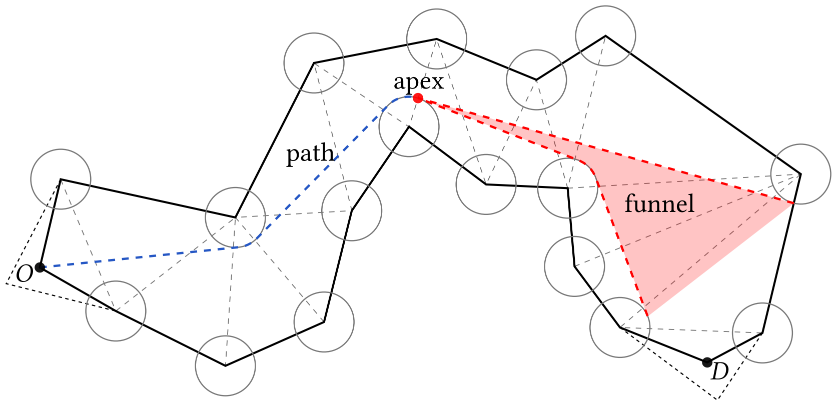


Figure 6.8 – Generalized funnel algorithm.

The adapted funnel algorithm is very similar to the original one, but instead of checking whether the next unprocessed vertex v is on the left or on the right of a particular segment $[\overrightarrow{v_1 v_2}]$, we need to check instead whether the segment (consistent[6] with the channel) that is tangent to the two disks respectively centered in v_1 and v is oriented CCW or CW from the segment tangent to the two disks centered in v_1 and v_2.

Since we now consider disks instead of points, an other modification should be made in order to make the algorithm correct. When the addition of the vertex v on one side of the funnel would result in an empty funnel, it is not necessarily anymore the vertex v_2 (i.e., the first vertex, different from the apex, on the opposite side of the funnel) that should become the new apex. Indeed, as illustrated in Figure 6.9, the segment tangent to the two disks centered in v_1 (i.e., the apex) and in v_2 can traverse the disk centered in v.

[6] There are at most four segments tangent to the two disks, but only one is consistent with the channel (i.e., the one that pass on the right of a vertex that is located on the left side of the channel and vice versa). From now on we will consider only this one. Note also that all the disks are of radius $d/2$, except for the origin and destination points which correspond to disks of zero radius.

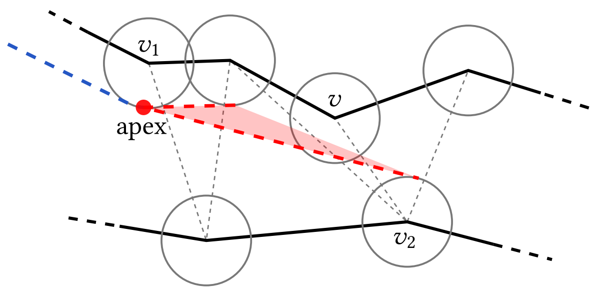


Figure 6.9 – Case where v should become the new apex instead of v_k.

This situation occurs whenever the segment tangent to the two disks centered in v_1 and in v is shorter than the segment tangent to the two disks centered in the v_1 and in v_2. In this case the shortest path must necessarily turn around v, and v thus becomes the new apex.

These modifications clearly do not affect the complexity of the algorithm and this generalized funnel algorithm thus also runs in $O(n)$ time in terms of the number n of triangles in the channel. Algorithm 6.2 presents the pseudocode for this generalized funnel algorithm.

```
Function FindShortestPath($m$, $edges[1 : m-1]$, $O$, $D$, $radius$)
    if $m = 1$ then
        $path$.init($O$); $path$.appendSegmentTowards($D$); return $path$;
    end
    $path$.init($O$); $funnel$.init(); $funnel$.setApex($O$, $-$);
    ProcessVertex($path$, $funnel$, $edges[1]$.leftEndPoint(), $Left$, $radius$);
    ProcessVertex($path$, $funnel$, $edges[1]$.rightEndPoint(), $Right$, $radius$);
    for $i := 2, 3, \ldots, m-1$ do
        if $edges[i]$.leftEndPoint() = $edges[i-1]$.leftEndPoint() then
            ProcessVertex($path$, $funnel$, $edges[i]$.rightEndPoint(), $Right$, $radius$);
        else
            ProcessVertex($path$, $funnel$, $edges[i]$.leftEndPoint(), $Left$, $radius$);
        end
    end
    ProcessVertex($path$, $funnel$, $D$, $Point$, $radius$);
    for each segment $[\overrightarrow{v_1 v_2}]$ in $funnel$ left side do
        $[\overrightarrow{t_1 t_2}]$ := consistent tangent to the two disks centered in $v_1$ and $v_2$;
        if $[\overrightarrow{v_1 v_2}]$ is the first segment on funnel left side then
            $path$.appendCircArcTowards($t_1$, $funnel$.appexDir() around $v_1$);
        else
            $path$.appendCircArcTowards($t_1$, CCW around $v_1$);
        end
        $path$.appendSegmentTowards($t_2$);
    end
    return $path$;
```

```
Function ProcessVertex(path, funnel, v, side, radius)
    if side = Left then
        loop
            [$\overrightarrow{v_1 v_2}$] := last segment on funnel left side;
            if [$\overrightarrow{v_1 v_2}$] does not exist then
                [$\overrightarrow{v_1 v_2}$] := first segment on funnel right side;
                if [$\overrightarrow{v_1 v_2}$] does not exist then
                    funnel.leftSide.append($v$);
                    break;
                end
                [$\overrightarrow{t_1 t_2}$] := consistent tangent to the two disks centered in $v_1$ and $v_2$;
                [$\overrightarrow{t'_1 t_v}$] := consistent tangent to the two disks centered in $v_1$ and $v$;
                if [$\overrightarrow{t'_1 t_v}$] is oriented CCW from [$\overrightarrow{t_1 t_2}$] then
                    funnel.leftSide.append($v$);
                    break;
                else
                    if distance($t'_1$, $t_v$) < distance($t_1$, $t_2$) then
                        funnel.setApex($v$, CCW);
                        path.appendCircArcTowards($t'_1$, funnel.appexDir() around $v_1$);
                        path.appendSegmentTowards($t_v$);
                        break;
                    else
                        funnel.rightSide.removeFirst();
                        funnel.setApex($v_2$, CW);
                        path.appendCircArcTowards($t_1$, funnel.appexDir() around $v_1$);
                        path.appendSegmentTowards($t_2$);
                    end
                end
            else
                [$\overrightarrow{t_1 t_2}$] := consistent tangent to the two disks centered in $v_1$ and $v_2$;
                [$\overrightarrow{t'_1 t_v}$] := consistent tangent to the two disks centered in $v_1$ and $v$;
                if [$\overrightarrow{t'_1 t_v}$] is oriented CCW from [$\overrightarrow{t_1 t_2}$] then
                    funnel.leftSide.append($v$);
                    break;
                else
                    funnel.leftSide.removeLast();
                end
            end
        end
    else if side = Right then
        {symmetrical procedure}
    else if side = Point then
        {same procedure as for Left but using a zero radius for the disks centered in $v$}
    end
```

Algorithm 6.2 – Generalized funnel algorithm.

It is worth mentioning that the shortest path resulting from the generalized funnel algorithm does not always stay entirely within the channel. As already mentioned in Section 6.3.3, it may be the case that moving from T_{i_k} to $T_{i_{k+1}}$ requires to pass through an intermediate triangle that is not represented in the path, which is not at all problematic.

6.3.5.3 Departure and Arrival Clearance

Since the two triangles at the extremities of a channel (i.e., those that contain O or D as vertex) are generally not triangles of the triangulation, they do not satisfy the Delaunay criterion. Therefore, even if this case is uncommon (and is actually not mentioned in [Dem07]), there may exist obstacles that could interfere with the shortest path at the beginning and the end of the channel (i.e., before the first and after the last interior edge).

In fact, due to the Delaunay criterion, all the possible segments (consistent with the channel) tangent to two disks centered in two interior vertices of a channel (extracted from G for a robot of diameter d) are feasible. But this may not be the case for the tangents passing by O and/or D. Figure 6.10 shows an example of this issue, where the robot in position O does not conflict with obstacles, but the path tangent to the disk centered in v_{R_1} and passing by O conflicts with v_p (which is not a vertex of the channel).

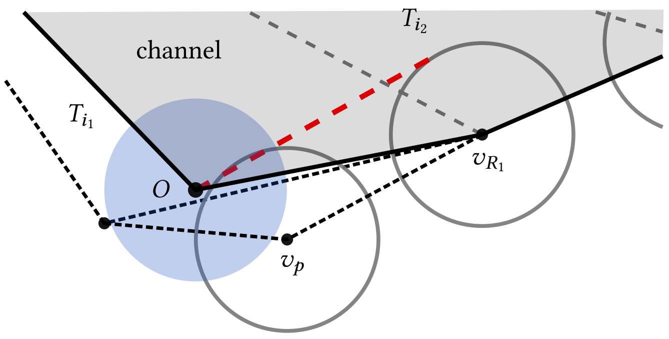


Figure 6.10 – Case where a vertex v_p, that is not part of the current channel, interferes with the shortest path before the first interior edge of the channel.

In such a case, a vertex is problematic (i.e., could interfere with the shortest path) for O (the case for D is handled symmetrically) iff it is located between the two segments tangent to the circle C_O of radius $d/2$ centered in O and passing respectively by the first vertex v_{R_1} on the right side and by the first vertex v_{L_1} on the left side of the channel, but outside of C_O (since O is feasible) and outside of the circumcircle of T_{i_1} (since T_{i_1} satisfies Delaunay's criterion). An illustration is provided in Figure 6.11.

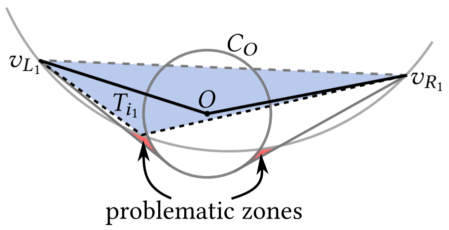


Figure 6.11 – Areas where the problematic vertices for O can be located.

This provides (at most) two small zones that can be searched for problematic vertices. Note that, due to the Delaunay criterion, if one such zone contains a problematic vertex, then the other one (if any) will necessarily be empty.

Assume that there exist some problematic vertices on one side of O. Since we know that there is at least one feasible path that avoid obstacles traversing the interior edges of the channel, this path should also traverse the edges[7] connecting each of the problematic vertices and the first interior vertex v_i on the other side of the channel. The solution is then to update the channel to take into account these edges by adding them before its first interior edge (in clockwise order if v_i is on the right side of the channel and in counter-clockwise order if v_i is on the left side).

Note that if the origin and destination points are located in the same triangle, then the area to be searched is the one between the two outer tangents to the two disks of radius $d/2$ centered in O and in D, but from which we exclude the two disks and the circumcircle of the triangle. One can then easily construct a channel to take into account these potential problematic vertices, if any[8].

The generalized funnel algorithm can then be applied to the updated channel (i.e., the one that take into account the potential problematic vertices for both O and D) to produce the shortest path in the channel from the origin to destination that clears all the obstacles in *Obst* with a distance at least equal to $d/2$.

6.3.5.4 Broken Line Generation

While the original funnel algorithm produces a sequence of lines segments between the origin and destination that connect several interior vertices of the channel, the generalized algorithm results in a path that consists in circular arcs of radius $d/2$ around these vertices and line segments tangent to them.

[7] Strictly speaking, the center of the robot do not necessarily cross these edges, but at least some points of the robot do. Thus, in the generalized Funnel algorithm, we will process these edges before the first interior edge of the channel, even if they are located on the other side of O from this first interior edge.

[8] If there is no problematic vertex, the shortest path between O and D is simply the segment that joins them.

The final step consists of replacing the circle arcs with straight line segments in order to obtain a path represented by a broken line. This operation is performed by ensuring that the deviations between circle arcs and the line segments that approximate them remain small. This can be achieved by imposing an upper bound on the absolute difference of orientation between adjacent segments. For Eurobot, we set this bound at $\frac{\pi}{2}$ in order to satisfy the requirements[9] of our path interpolation method, presented in Chapter 5.

Note that, in very narrow passages, the sequence of line segments, that replaces a circular arc turning around a vertex v with a given upper bound, could interfere with one of the opposite vertex of an interior edge to which v is attached. The easiest solution is then to locally reduce this upper bound.

6.3.6 Illustration

We illustrate all the successive steps of the presented path planning method in Figure 6.12, on a simplified case study taken from the Eurobot contest. Figure 6.12(a) shows the area with the obstacles, origin O and destination D locations (along with the required safety distance). Figure 6.12(b) shows how the obstacles and the borders of the area are approximated by a finite set of points and their Delaunay triangulation. Figure 6.12(c) illustrates the graph that corresponds to this triangulation. Note that, only the arcs of the graph that have sufficient clearance are drawn.

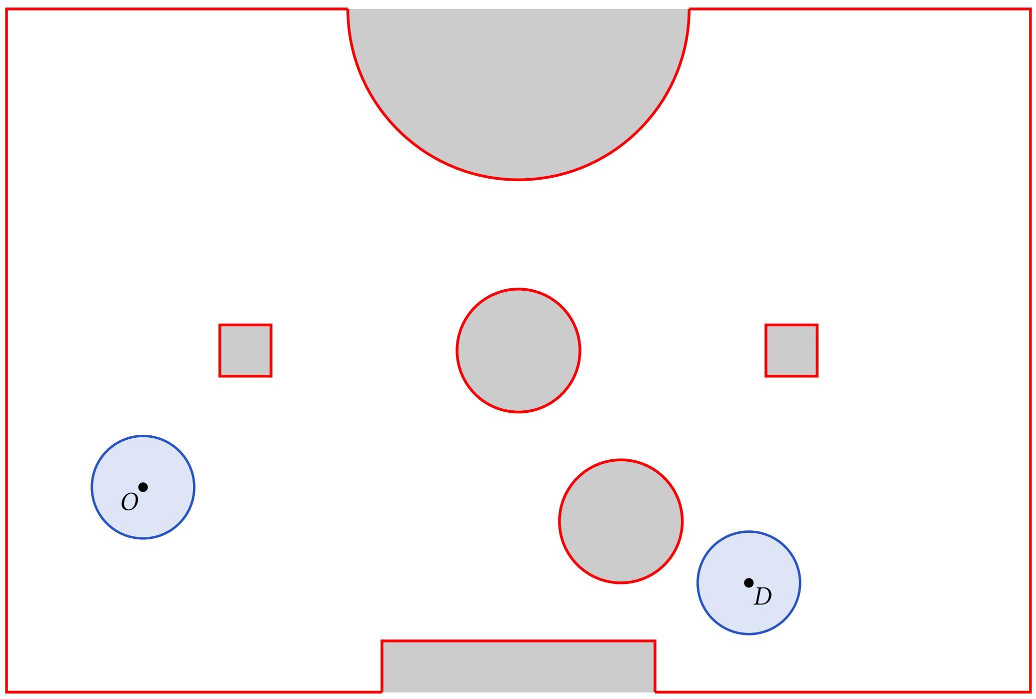
This graph is searched and the shortest path from origin to destination is highlighted in green. Figure 6.12(d) shows the channel that correspond to the shortest path in the graph. The result of the generalized funnel algorithm performed on this channel is illustrated in Figure 6.12(e), and the final sequence of line segments (i.e., the broken line) is shown in Figure 6.12(f). Figure 6.12(f) also shows the smoothed path produced by our interpolation method, described in Chapter 5.

6.4 Refinements

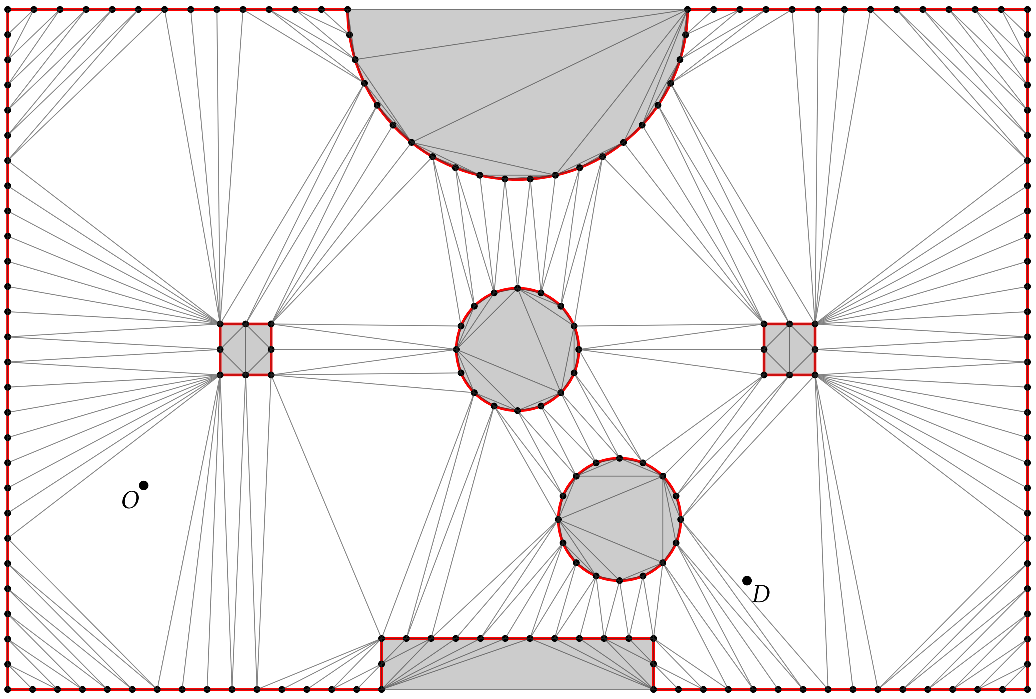
6.4.1 Misled Searches

Using a Delaunay-based (or Voronoi-based) method has many advantages, but since edges of the resulting roadmap are located at the largest possible distance from obstacles, this approach has the drawback of distorting the length of paths. This can sometimes mislead the search algorithm, since the shortest path in the roadmap does not necessarily translate into the optimal one after applying the funnel algorithm.

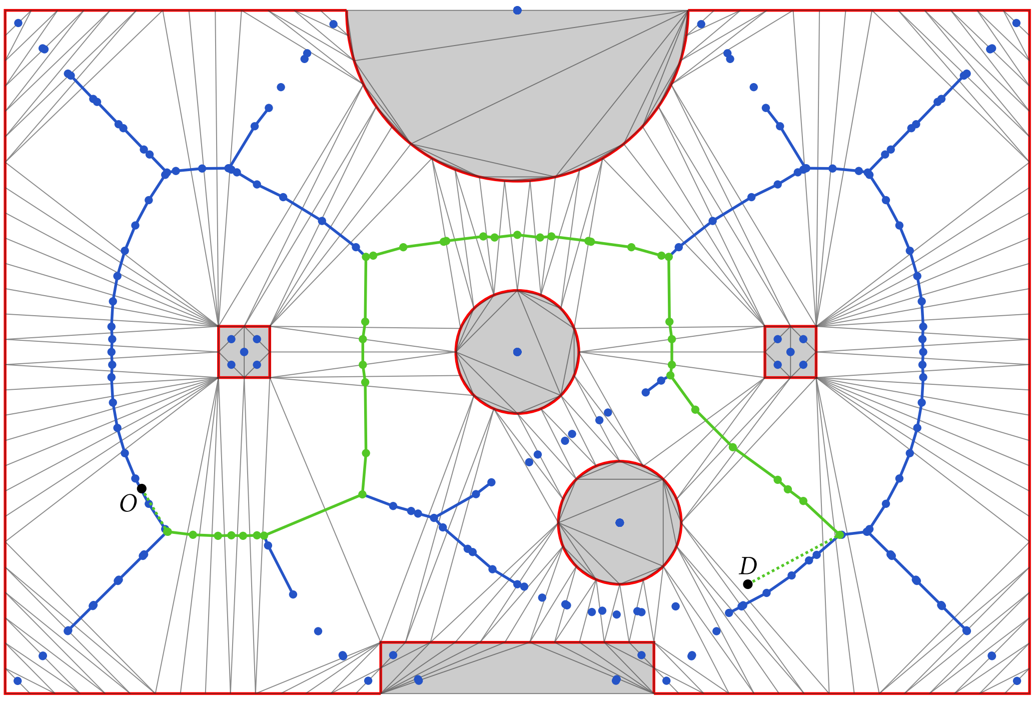
[9] The interpolation method, presented in Chapter 5, also requires that each point of the broken line provides a clearance parameter c_i. One can naturally easily compute these parameters from the circular arcs.



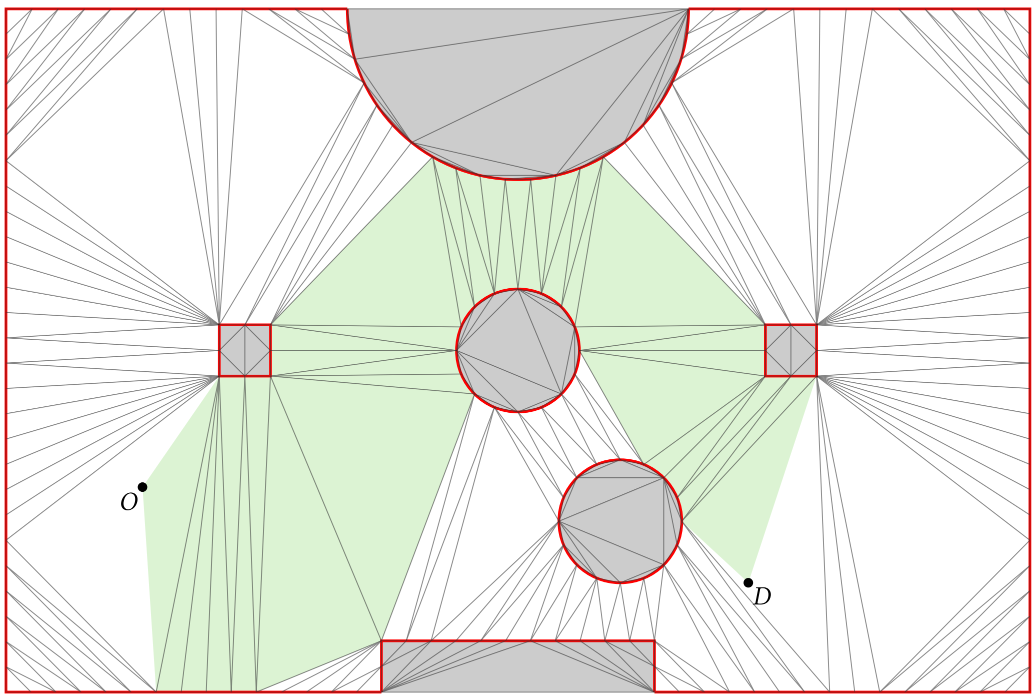
(a) Problem statement.



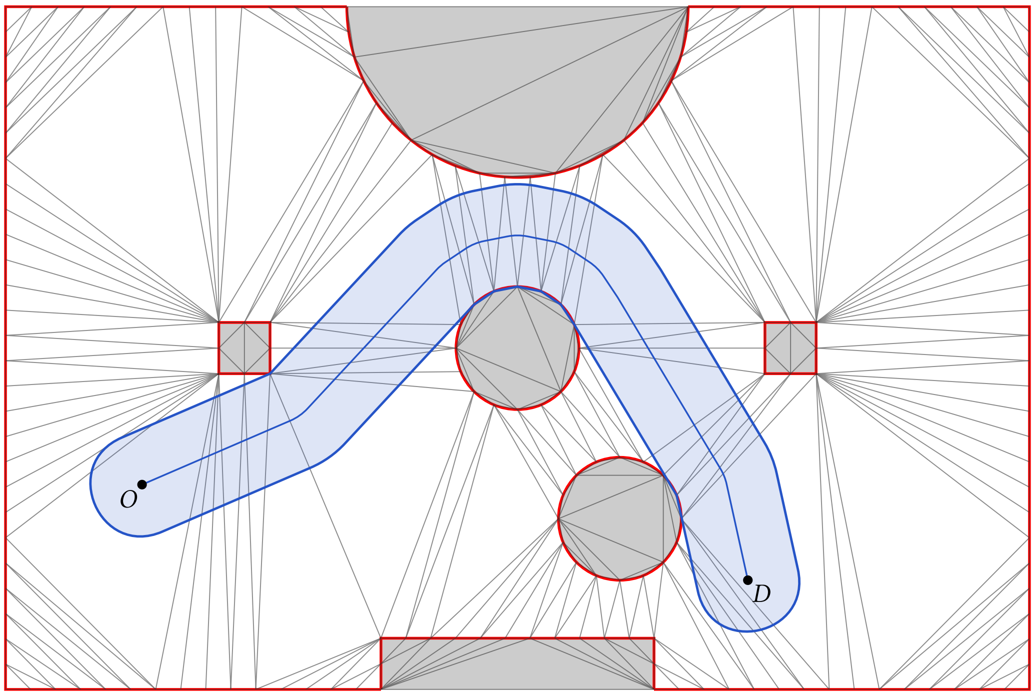
(b) Delaunay triangulation of the (discretized) obstacles.



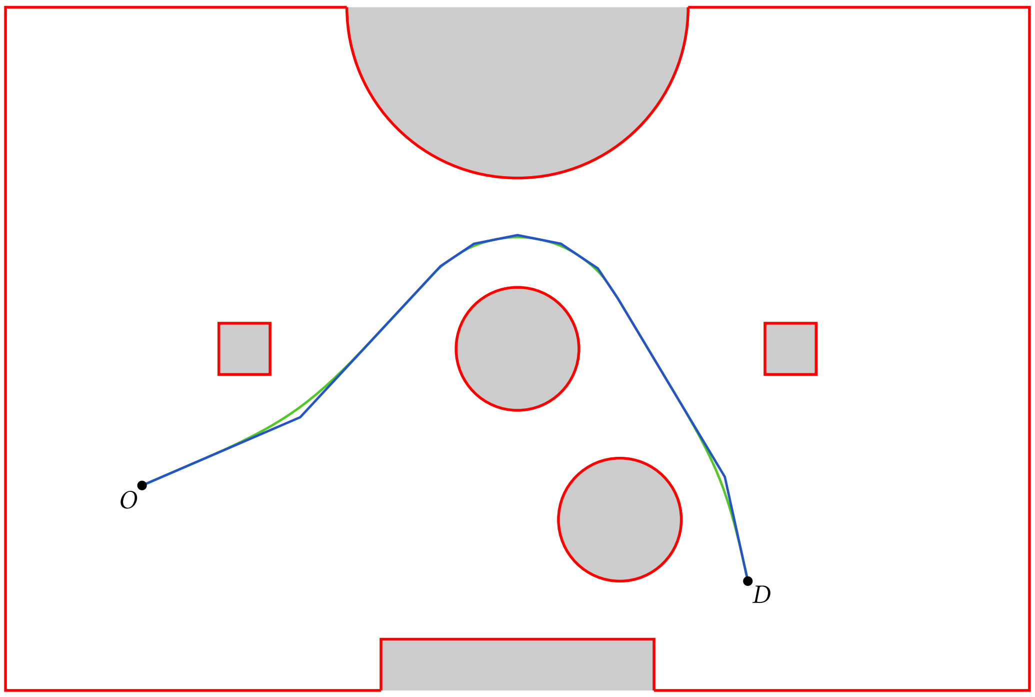
(c) Feasible arcs of the corresponding graph (shortest path highlighted in green).



(d) Extracted channel.



(e) Shortest path in the channel.



(f) Broken Line (in blue) and smoothed path (in green).

Figure 6.12 – Illustration of the path planning method.

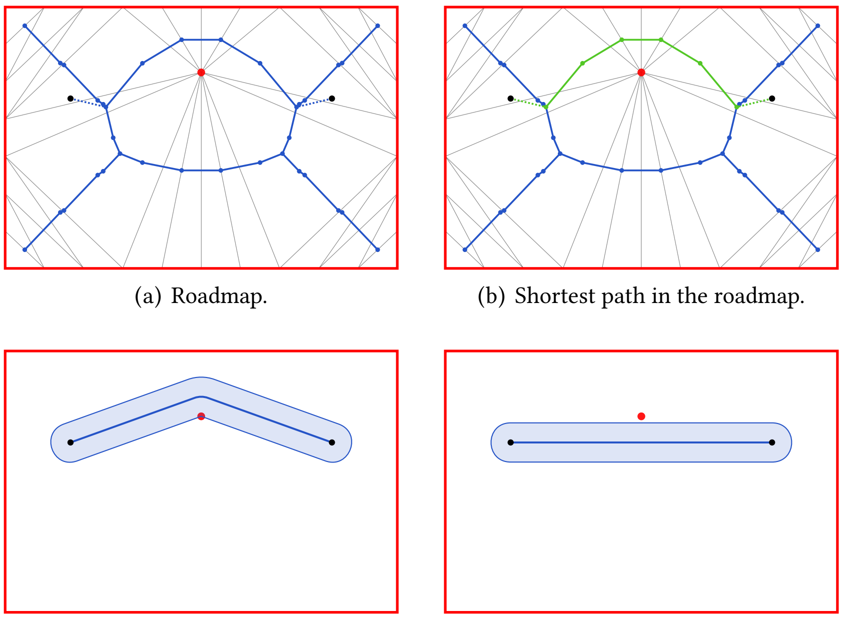

(a) Roadmap.
(b) Shortest path in the roadmap.
(c) Shortest path in the channel.
(d) Global shortest path

Figure 6.13 – Issue with the Delaunay-based search.

This issue is illustrated in Fig. 6.13, which shows an instance for which our path planning algorithm generates a path that is not optimal. This problem is especially noticeable when C_{free} contains large zones that are free from obstacles.

The optimal path (using various criteria) can be extracted from the roadmap by using, for instance, an augmented version of Dijkstra's shortest path algorithm which allows edge weights to be defined relative to the current path by using Markov-like states [KPA03]. The drawback is that this method is complex and computationally expensive.

In fact, since the global shortest path in the plane does not necessarily correspond to the fastest trajectory, it is usually sufficient to use an approximation. Indeed, reducing the length of a path usually reduce the time needed for following it, but can also lead to sharper turns where the robot will need to slow down, or other similar issues.

One simple improvement that we can make to our search algorithm is to tune it by using a heuristic that reduces the weight of the edges that are far from obstacles, since such edges are likely to benefit the most from reductions by the funnel algorithm. This solution has the advantage of being easy to implement, but, of course, does not guarantee optimality.

[Kal10] presents other metrics and heuristics to efficiently extract a short path from a roadmap. It also shows that even using the metric that we have already discussed (i.e., weighting the edges of the graph with the distance between the centers of the circumcircles of the underlying triangles) usually suffices for most applications.

6.4.2 Orientation

We have defined the origin O and destination D of a path as the locations initially and finally taken by the robot but, in most applications dealing with nonholonomic robots, one additionally needs to impose the orientation at that points. One can, of course, just start and end the trajectory with local turns in order to take into account these initial and final orientations, but this solution is usually not effective since it is likely to waste time.

For Eurobot, an efficient solution was obtained by constructing small line segments $[OO']$ and $[D'D]$ that are travelled in the appropriate directions, and that will respectively be prepended and appended to a broken line computed from O' to D'. The lengths of such segments are computed to be consistent with the physical constraints imposed on the robot, for instance, a reasonable strategy is to make $|OO'|$ at least equal to the braking distance of the robot at its initial speed. These segments must also clear obstacles. This approach has the advantage of generating paths with zero initial and final curvatures, which simplifies chaining successive paths.

6.4.3 Improved Smoothing

As illustrated in Figure 6.14(a), discretizing an obstacle, like a circle or a circular arc, by a set of points can sometimes result in a trajectory that is not well smoothed by our interpolation method. In fact, the funnel algorithm computes the shortest path that avoids the set of points (and not the circle), and thus the situation is as if the robot must avoid a regular polygon instead of a circle.

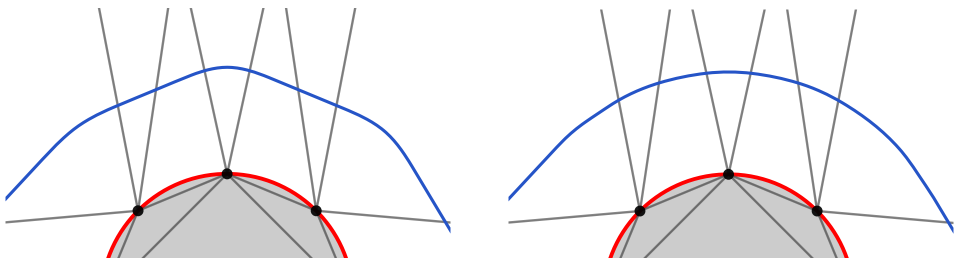

(a) Using exact clearance parameters. (b) Using increased clearance parameters.

Figure 6.14 – Interpolation.

The interpolation method smoothes the path as much as possible in order to satisfy the (exact) computed clearance parameters c_i, but this results in a path with undesirable straight line pieces. For Eurobot, we solve this small problem by judiciously increasing these clearance parameters in order to increase the flexibility of the interpolation algorithm. In this way, all the segments, that turn around a particular circular arc, will be allowed to be tangent to a common circle[10] in the first step of the interpolation method. Figure 6.14(b) shows the smoothed path that result from this refinement.

6.5 Experimental Results

The main advantage of the approach presented in this chapter is its efficiency: Triangulating a set of points and searching for the shortest path in a finite graph can be performed in $O(n \log n)$ time, where n denotes the number of obstacles, and the generalized funnel algorithm runs in $O(n)$ time. One drawback is the fact that the shortest path in the roadmap does not necessarily correspond to the shortest feasible one in the plane, but using a suitable metric, the approximation usually suffices for most applications [Kal10].

In order to illustrate the efficiency of this approach, we report in Figure 6.15 the time needed for running the complete path planning algorithm on a few sample cases experienced in the Eurobot application. We distinguish the costs of the Delaunay triangulation, the shortest-path search (along with the origin and destination insertion), and the generalized funnel algorithm. The total computational cost typically amounts to less than half a millisecond of CPU time on an i5-460M processor running at 2.53 GHz, which allows near-optimal path planning in real-time.

	Point obstacles	Triangles	Delaunay	Search	Funnel	Total time
1	194	274	127.9 μs	21.4 μs	7.4 μs	156.7 μs
2	206	298	144.4 μs	21.7 μs	11.5 μs	177.6 μs
3	291	422	202.2 μs	21.8 μs	6.7 μs	230.7 μs
4	366	522	239.9 μs	34.5 μs	7.7 μs	282.1 μs
5	493	752	359.7 μs	49.9 μs	10.4 μs	420.0 μs

Figure 6.15 – Experimental results.

Note that the first case considered in this experiment corresponds to the problem depicted in Figure 6.12.

[10] This common circle is smaller than the one we need to avoid (i.e., the real obstacle is the circumcircle of the points, but the obstacle that will be avoided instead is the inscribed circle of the convex hull of these points), but this is usually not a problem due to safety margins. If it was the case, all we need to do is to sufficiently dilate the obstacle.

CHAPTER 7

Path Planning with Polygonal Obstacles

Chapter 6 discussed the computation of a roadmap for a robot moving in two-dimensional space, in the presence of a finite set of point obstacles. This chapter addresses an original generalization of this method for sets of obstacles that include line segments in addition to individual points, which is useful for efficiently delineating polygonal obstacles without prior discretization. The method starts from a constrained Delaunay triangulation of the obstacles, and refines it by adding a carefully selected set of Steiner points. From the refined triangulation, one can straightforwardly extract a roadmap graph suited for motion planning, for cylindrically shaped robots of arbitrary diameter. The method does not require to store additional information in triangles, i.e., only the length of traversed edges has to be checked, which makes searching for paths very efficient. Compared to other solutions, our approach has the advantage of being simpler, as well as significantly faster.

7.1 Introduction

The path planning method presented in Chapter 6 runs in $O(n \log n)$ time, where n denotes the number of point obstacles. Therefore, as discussed in Section 6.5, this method is often efficient enough to allow near-optimal path planning for cylindrically shaped robots in real-time.

The only main drawback is that it can only process set of points obstacles. For more general obstacles, such as polygons, a straightforward workaround consists in discretizing obstacles into finitely many individual points. For small cases, this is easily feasible, but for larger ones, it can become difficult to find a good trade-off between a fine discretization level, which may yield an unnecessarily large roadmap graph, and a coarse one, which might result in an imprecise approximation.

Clearly, discretizing line segments into lots of points is not the most effective way to determine whether a robot can move safely between them. This chapter tackles this drawback, and shows how the procedure for building of a roadmap can be generalized to sets of obstacles that include line segments in addition to individual points, which allows to efficiently delineate polygonal obstacles.

As in the previous chapter, we consider robots that have a cylindrical shape, i.e., a robot at a position P occupies a region of the plane that corresponds to a disk of diameter d centered in P, where d is a parameter of the robot.

7.2 Related Work

As already discussed in Chapter 6, a first possible solution for building a roadmap with respect to polygonal obstacles is to compute a generalized Voronoi diagram, which partitions the points of the plane according to their nearest point or line obstacle [Ger10]. The main drawback is that the edges of such diagrams can take the form of parabolic arcs, that are more tricky to work with than line segments.

The most widely used workaround, presented in the previous chapter, consists in discretizing line obstacles into finitely many individual points [BG08], but this can lead to unnecessarily large roadmap graphs, which is inefficient.

A third strategy is to build a *visibility-Voronoi complex* [WBH05], which produces a graph from which a roadmap can easily be extracted, but this technique turns out to be costly, requiring a computation time that is more than quadratic in the number of obstacles.

By only constructing the relevant subgraph of the visibility graph for a given path query (i.e., for given origin and destination locations), the strategy presented in [KMM97] is often more efficient than the one based on the visibility-Voronoi complex, but still requires a computation time that is quadratic in the number of obstacles.

We can also cite methods based on the continuous Dijkstra paradigm [Mit93, HS99, IKM10], which simulate the expansion of a wavefront from a point source in the presence of polygonal obstacles. These methods are usually subquadratic in time, but are tricky to implement and, as they merely try to directly find the global shortest path in the plane (which is a more complex problem), are still computationally expensive in practice.

Another strategy is to start from a *constrained Delaunay triangulation*, in which the line obstacles are required to appear as edges of the triangles [Che89]. A roadmap can then be extracted from such a triangulation using the same method as for (classical) Delaunay ones. Unfortunately, exploiting such a roadmap for planning the paths of a robot of known size is not as straightforward as before: The existence of a path from a triangle to another that only traverses edges of length at least equal to d does not necessarily imply that a corresponding path can be followed by a robot of diameter d without colliding with obstacles.

In [Kal14], this problem is solved by augmenting the triangulation with parameters known as *clearance values* that have to be taken into account when the roadmap is searched for paths. In addition, some parts of the triangulation have to be refined in order for this mechanism to behave correctly.

An other approach is to determine the width of narrow passages by refining the constrained edges that have orthogonal projections of vertices on the opposite side of a passage [LD04]. But this method is only suited for simple situations.

The contribution of this work is to show that a roadmap graph can be derived from a constrained Delaunay triangulation by simply refining this triangulation according to a carefully selected set of additional *Steiner points*. The resulting graph has the property that the paths traversing edges of length at least equal to d exactly correspond to the feasible paths of a robot of diameter d, for arbitrary values of d. In other words, this roadmap is suited for motion planning of robots with an arbitrary size.

The method does not require to store additional information in triangles, i.e., only the length of traversed edges has to be checked, which makes searching for paths very efficient. Our algorithm has been implemented and experimentally evaluated on randomly generated sets of obstacles, obtaining significantly lower execution times than [BG08, Ger10, Kal14].

7.3 Point and Line Obstacles

Our aim is to generalize the results of the previous chapter (Section 6.3) to sets of obstacles that include line segments in addition to individual points. The motivation is to be able to deal with obstacles that have a polygonal shape. We first adapt triangulations to this setting, and then investigate how to build roadmaps out of them.

7.3.1 Constrained Delaunay Triangulations

Let us consider a set of obstacles $Obst = \{A_1, A_2, \dots A_p, L_1, L_2, \dots, L_q\}$ composed of a finite number of points A_i as well as a finite set of line segments L_i. We assume w.l.o.g. that each segment links two points A_i and A_j that belong to *Obst*, and that the intersection of any pair of segments is either empty, or limited to a point A_i belonging to *Obst*.

For such a set of obstacles *Obst*, a *constrained triangulation* is a triangulation in which every segment $L_i \in Obst$ (called a *constrained segment*) forms the edge of at least one triangle.

Note that, if the goal is to reason about the possible moves of a robot, the notion of triangulation can somehow be slightly extended. First, it is not mandatory for regions that cannot be reached by the robot, such as the interior of polygonal obstacles, to be covered by triangles. Second, since constrained segments cannot be traversed, we allow an edge of a triangle to be composed of a series of collinear constrained segments instead of a single one. An example of constrained triangulation is given in Figure 7.1, in which an unreachable region is grayed out, and constrained segments are represented in red.

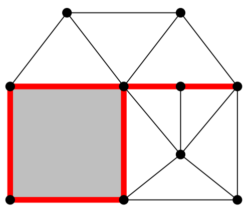


Figure 7.1 – Example of constrained triangulation.

The Delaunay criterion can readily be adapted to constrained triangulations [Ber04]. Given a set *Obst* of obstacles, a point P_2 is said to be *visible* from a point P_1 if the intersection between the segment $[P_1P_2]$ and every constrained segment $[A_iA_j] \in Obst$ is either empty, or is equal to one of its extremities P_1 or P_2. A constrained triangulation is then *Delaunay* if for each of its triangles $A_iA_jA_k$, all the points in *Obst* that are visible from A_i, A_j or A_k are located outside or on its circumcircle. It is easily shown that this criterion is equivalent to imposing that, for every pair $(A_iA_jA_k, A_\ell A_jA_i)$ of adjacent triangles, with $A_k \neq A_\ell$, either $[A_iA_j]$ is a constrained segment, or one has $\widehat{A_iA_kA_j} + \widehat{A_iA_\ell A_j} \leq \pi$. An example of constrained Delaunay triangulation is given in Figure 7.2.

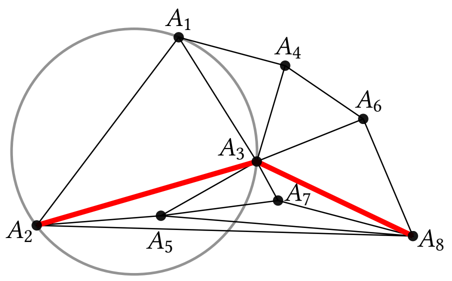


Figure 7.2 – Example of constrained Delaunay triangulation. A_5 and A_7 are not visible from A_1, but all the other points are.

It is also natural to require[1] that the robot remains confined in an area fully delineated by external obstacles. Hence, every potential position of the robot belongs to the convex hull of the obstacles, and thus to a triangle of the triangulation.

7.3.2 Refining Constrained Delaunay Triangulations

A roadmap can be derived from a constrained triangulation by a procedure similar to the one outlined in Section 6.3.3: One builds a graph whose nodes are associated to triangles, and links the nodes corresponding to pairs of adjacent triangles. However, in the presence of constrained segments, Theorem 6.3.1 does not hold anymore. The problem is illustrated in Figure 7.3(a), where a robot of diameter $d = \min(|A_1A_2|, |A_1A_3|)$ would not be able to traverse the gap between the point A_1 and the constrained segment $[A_2A_3]$.

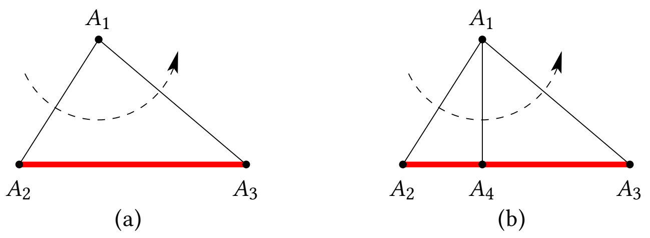


Figure 7.3 – Constrained triangulation refinement.

Our solution to this problem is to *refine* the constrained triangulation by inserting additional obstacle points known as *Steiner points*, until we obtain a constrained triangulation for which a result similar to Theorem 6.3.1 can be established. Adding a Steiner point amounts to splitting a constrained segment in two components connected at that point, fragmenting one of the two triangles adjacent to this segment.

For instance, Figure 7.3(b) shows that adding the orthogonal projection A_4 of A_1 onto $[A_2A_3]$ as a Steiner point[2] decomposes the triangle $A_1A_2A_3$ into $\{A_1A_2A_4, A_1A_4A_3\}$, yielding a channel for which Theorem 6.3.1 now applies. Indeed, a robot is able to traverse the sequence of edges $[A_1A_2]$; $[A_1A_4]$; $[A_1A_3]$ if and only if its diameter d is at most equal to $\min(|A_1A_2|, |A_1A_4|, |A_1A_3|)$.

Since inserting a Steiner point splits a triangle in two, this operation may result in a refined constrained triangulation that does not satisfy anymore Delaunay's criterion. It then becomes necessary to update the triangulation so as to restore this property, which will be essential to the correct computation of Steiner points.

[1] If this constraint is not satisfied, this can straightforwardly be achieved by adding a sufficiently large bounding box to all the points.

[2] We assume that the Steiner point A_4 added on the segment $[A_2A_3]$ is only visible from A_2, A_3 and the points that are on the same side of this segment as A_1.

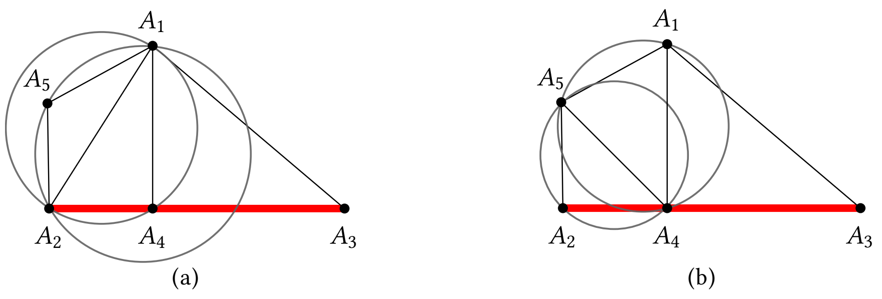


Figure 7.4 – Flipping procedure.

This update procedure can be carried out as follows. Consider a triangle $A_1A_2A_3$ that has been split into $\{A_1A_2A_4, A_1A_4A_3\}$ by the addition of a Steiner point A_4 inside the segment $[A_2A_3]$. If the segment $[A_1A_2]$ is unconstrained and joins two triangles $A_1A_2A_4$ and $A_1A_5A_2$, one has to check whether they satisfy Delaunay's criterion. If not, as illustrated in Figure 7.4(a), the quadrilateral $A_4A_1A_5A_2$ must be *flipped*, which consists in replacing, as depicted in Figure 7.4(b), the pair of triangles $\{A_1A_2A_4, A_1A_5A_2\}$ by $\{A_1A_5A_4, A_2A_4A_5\}$. The same operation must be performed for the segment $[A_1A_3]$, as well as for the remaining outside edges of flipped quadrilaterals. It is known that this procedure, borrowed from *incremental Delaunay triangulation* algorithms, has a low average cost, the number of triangles to be processed being usually small [BCKO08].

Finally, it should be stressed out that the refinement operation must be able to handle situations that are more complex than the one illustrated in Figure 7.3. For instance, even if $[A_2A_3]$ is unconstrained, the traversal of $[A_1A_2]$; $[A_1A_3]$ might be affected by constraints located beyond this segment. We address the problem of selecting a suitable set of Steiner points in the next two subsections.

7.3.3 Point/Point and Point/Line Constraints

Recall that the goal is to obtain a roadmap graph in which, for every $d > 0$, each path that only traverses edges of length greater than or equal to d represents a channel that can be passed by a robot of diameter d. In other words, one should be able to check whether a channel can be passed only by measuring the length of the traversed edges, similarly to the case of point obstacles discussed in Section 6.3.

A robot of diameter d that moves along a channel has to clear different configurations of obstacles. First, traversing an edge $[A_mA_n]$ between two vertices A_m and A_n is only possible if $d \leq |A_mA_n|$. We call this condition a *point/point constraint.* Second, if the orthogonal projection H of a vertex A_k on a constrained segment $[A_iA_j]$ is located inside this segment, moving between A_k and this segment (in other words, traversing $[A_kH]$) imposes $d \leq |A_kH|$, which forms a *point/line constraint.*

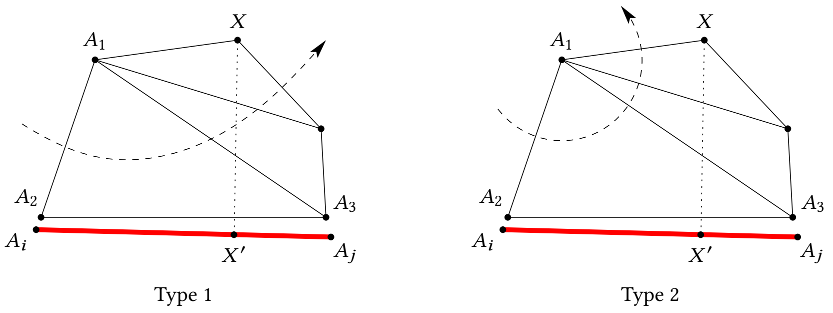


Figure 7.5 – Channel types.

In order to assess the possibility of passing a channel, it is sufficient to consider point/-point and point/line constraints. As discussed in [Kal14], other configurations such as moving between two constrained segments are systematically covered by the point/point and point/line constraints induced by their extremities.

It has been shown in Section 6.3 that point/point constraints are automatically dealt with by Delaunay triangulations: In the absence of constrained segments, Theorem 6.3.1 holds and the triangulation provides a suitable roadmap graph. The difficulty is thus to handle correctly point/line constraints.

Consider a triangle $A_1A_2A_3$ that the robot crosses from the edge $[A_1A_2]$ to the edge $[A_1A_3]$, or the other way around. We assume w.l.o.g. that this triangle satisfies $|A_1A_2| \leq |A_1A_3|$. We check whether there exists a vertex X of the triangulation located on the same side of A_2A_3 as A_1, involved in a point/line constraint with a constrained segment $[A_iA_j]$ located on the other side of this line (which precisely means that the orthogonal projection X' of X on A_iA_j belongs to the interior of the segment $[A_iA_j]$, and is such that $[XX']$ intersects $[A_2A_3]$).

A point/line constraint induced by such a vertex X is relevant only if it is more restrictive than the point/point constraints generated by the channel followed by the robot. We call a vertex X *problematic* for the traversed edges $[A_1A_2]$ and $[A_1A_3]$ if it satisfies the abovementioned conditions, and is such that $|XX'| < \min(|A_1A_2|, |XA_2|, |XA_3|)$.

Indeed, it is not necessary to consider vertices X for which $|XX'| \geq |A_1A_2|$, since the point/line constraint between X and $[A_iA_j]$ is then systematically satisfied as a consequence of the fact that the robot is able to traverse $[A_1A_2]$. The motivation for the conditions $|XX'| < |XA_2|$ and $|XX'| < |XA_3|$ is less obvious. With respect to X, there are actually two types of channels in which the triangle $A_1A_2A_3$ is traversed from $[A_1A_2]$ to $[A_1A_3]$: Those in which the robot needs to cross the segment $[XX']$ (*type 1*), and those in which it does not (*type 2*). Note that the segment $[XX']$ does not necessarily correspond to an edge of the triangulation. The two situations are illustrated in Figure 7.5.

For a channel of type 1, if $|XX'| > \min(|XA_2|, |XA_3|)$, the point/line constraint between X and $[A_iA_j]$ cannot be more restrictive than the point/point constraints induced by the channel, since it guarantees that the robot is able to traverse the segment $[XX']$. On the other hand, in a channel of type 2, the point/line constraint induced by X is irrelevant and does not need to be taken into account.

When studying the triangle $A_1A_2A_3$, one does not know whether the channel that will be followed after traversing $[A_1A_2]$ and $[A_1A_3]$ is of type 1 or 2, hence both possibilities have to be taken into account. This goal is achieved by considering only the point/line constraints induced by problematic vertices, according to our definition. Note that the notion of problematic vertices is similar to the concept of *disturbances* introduced in [Kal10], but our strategy for dealing with them, described in the next subsection, differs.

7.3.4 Steiner Points Placement

Our approach to selecting a suitable set of Steiner points is based on the following idea: For every pair of unconstrained edges $([A_1A_2], [A_1A_3])$ of every triangle $A_1A_2A_3$ of the constrained triangulation, if this pair of edges potentially admits a problematic vertex, then this triangle has to be refined. The refinement operation consists in inserting a Steiner point in $[A_2A_3]$ in such a way that, for the two resulting triangles, problematic vertices cannot exist anymore.

Checking for problematic vertices is done as follows. Recall that we assume w.l.o.g. $|A_1A_2| \leq |A_1A_3|$. First, it is easily established that the pair $([A_1A_2], [A_1A_3])$ can only admit a problematic vertex if the corresponding triangle is such that $\widehat{A_1A_2A_3} < \pi/2$. This property relies on the fact that the constrained triangulation satisfies Delaunay's criterion, from which it follows that all the points located outside or on the circumcircle of $A_1A_2A_3$ necessarily violate at least one of the conditions to be problematic.

If the angle $\widehat{A_1A_2A_3}$ is acute, which can be checked by testing whether the orthogonal projection of A_1 onto A_2A_3 is an interior point of the edge $[A_2A_3]$, then the next step is to check whether A_1 itself is problematic. This is equivalent to deciding whether there exists a constrained segment $[A_iA_j]$ that contains as an interior point the orthogonal projection A'_1 of A_1 onto A_iA_j, such that A'_1 is visible from A_1, $[A_1A'_1]$ crosses $[A_2A_3]$, and $|A_1A'_1| < |A_1A_2|$.

We will introduce a procedure for carrying out efficiently this check in Section 7.4. If this operation succeeds, then A'_1 is inserted as a Steiner point into the constrained segment $[A_iA_j]$, splitting the triangle that is adjacent to this segment, on the same side as A_1. The situation is illustrated in Figure 7.6(a).

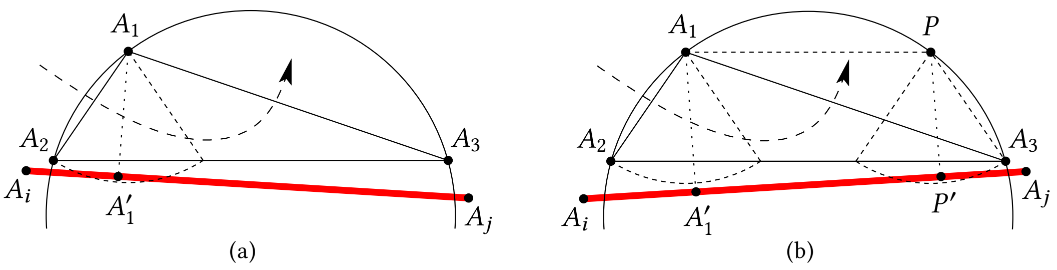


Figure 7.6 – Checking for problematic vertices.

When, on the other hand, A_1 turns out to be unproblematic, it remains to check whether there may exist other potential problematic vertices. This operation is performed in the following way. We compute the point P located at the intersection of the circumcircle of $A_1A_2A_3$ and the line parallel to A_2A_3 containing A_1. Then, we apply to P the same decision procedure as to A_1 in the previous step: We check whether P projects onto a constrained segment $[A_iA_j]$, its projection P' is visible from P, $[PP']$ crosses $[A_2A_3]$, and $|PP'| < |A_1A_2|$. In such a case, the projection A'_1 of A_1 onto $[A_iA_j]$ is necessarily interior to this segment. We then add A'_1 as a Steiner point, in the same way as before.

This case is illustrated in Figure 7.6(b). The motivation for selecting A'_1 instead of P' is twofold. First, P is generally not a vertex of the constrained triangulation, hence it does not always appear in the result of the refinement. Second, the goal of getting rid of triangles that may admit problematic vertices is correctly achieved with the choice of A'_1.

Indeed, after having refined a constrained triangulation using the procedure described in this section, every resulting triangle that is part of a channel includes a *safe zone* that cannot be crossed by constrained segments. This safe zone is illustrated in Figure 7.7. Every robot that is able to pass a channel can perform this operation while remaining confined in the safe zone of the successively visited triangles, which proves the following result.

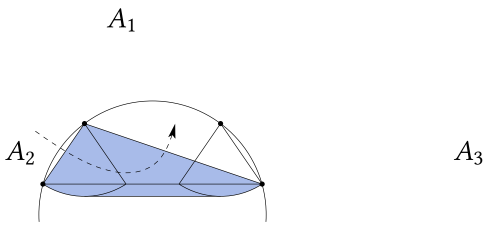


Figure 7.7 – Safe zone.

Theorem 7.3.1. *In a refined constrained triangulation, there exists a path in G from a feasible triangle T_i to another one T_j that traverses only unconstrained edges of length at least equal to d iff there exists a feasible path leading from a point in T_i to a point in T_j, for a robot of diameter d.*

Where a feasible triangle is defined, as in the previous chapter, as a triangle that contains a least one feasible point (i.e., a location of the center of the robot at which none of its interior points conflicts with obstacles in *Obst*).

Finally, it is worth mentioning that our refinement procedure always terminates, since the insertion of a Steiner point creates right triangles that will not be refined further.

7.4 Refinement Algorithm

As explained in Section 7.3.4, a constrained triangulation is refined by checking, for each pair $([A_1A_2], [A_1A_3])$ of unconstrained edges belonging to one of its triangles $A_1A_2A_3$, whether a Steiner point has to be created by projecting A_1 on some constrained segment. Assuming w.l.o.g. $|A_1A_2| \leq |A_1A_3|$, this check only has to be performed if $\widehat{A_1A_2A_3} < \pi/2$, and amounts to deciding whether there exists a constrained segment $[A_iA_j]$ onto which A_1 can be projected, with its projection A'_1 satisfying specific conditions. If the check does not succeed, a similar operation has to be carried out for another point P derived from A_1.

We now explain how to perform efficiently this check operation, which can be seen as a particular instance of the following problem: Given a point X, an edge $[A_kA_\ell]$ of the triangulation that contains the orthogonal projection X' of X, and a distance λ such that $|XX'| < \lambda \leq \min(|XA_k|, |XA_\ell|)$, decide whether there exists a point Y belonging to a constrained segment $[A_iA_j]$, such that $[XY]$ crosses $[A_kA_\ell]$ with $|XY| < \lambda$. If yes, return the corresponding segment $[A_iA_j]$ that is nearest to X.

In order to solve this problem, the first step is to check whether $[A_kA_\ell]$ itself is a constrained segment, in which case the procedure terminates with $[A_iA_j] = [A_kA_\ell]$. Otherwise, we explore the triangulation, starting with the triangle $A_kA_\ell A_q$ adjacent to $[A_kA_\ell]$ on the opposite side as X. It can be shown that, thanks to Delaunay's criterion, the edge $[A_kA_q]$ can contain a point Y such that $|XY| < \lambda$ only if $|A_kA_q| > |A_\ell A_q|$.

As a consequence, the search can be continued by repeating the same procedure with the longest edge among $[A_kA_q]$ and $[A_\ell A_q]$ replacing $[A_kA_\ell]$. The check terminates when a suitable constrained segment is found, the projection X' of X onto $[A_kA_\ell]$ does not exist, or this projection is such that $|XX'| \geq \lambda$.

The complete algorithm for refining a constrained triangulation is summarized in Algorithm 7.1.

```
Function Refine(constrained triangulation $Tr$)
    for all unconstrained $[A_1A_2]$, $[A_1A_3]$ such that $A_1A_2A_3 \in Tr$ and $|A_1A_2| \leq |A_1A_3|$ do
        $(ok, [A_iA_j])$ := Check($Tr$, $A_1$, $[A_2A_3]$, $|A_1A_2|$);
        if not $ok$ then
            $P$ := $circumcircle(A_1A_2A_3) \cap d$, with $d \parallel A_2A_3$, $A_1 \in d$;
            $(ok, [A_iA_j])$ := Check($Tr$, $P$, $[A_2A_3]$, $|A_1A_2|$);
        end
        if $ok$ then
            $A'_1$ := projection of $A_1$ onto $[A_iA_j]$;
            $A_k$ := vertex of $A_iA_jA_k \in Tr$, on same side of $A_iA_j$ as $A_1$;
            $Tr$ := $(Tr \setminus \{A_iA_jA_k\}) \cup \{A_kA_iA'_1, A_kA'_1A_j\}$;
            FlipIfNeeded($Tr$, $A'_1$, $[A_kA_i]$);
            FlipIfNeeded($Tr$, $A'_1$, $[A_kA_j]$);
        end
    end
    return $Tr$;

Function Check($Tr$, $X$, $[A_iA_j]$, $\lambda$)
    $X'$ := projection of $X$ onto interior of $[A_iA_j]$;
    if $X'$ does not exist or $|XX'| \geq \lambda$ then return $(false, -)$;
    if $[A_iA_j]$ is constrained then return $(true, [A_iA_j])$;
    $A_k$ := vertex of $A_iA_jA_k \in Tr$, on other side of $A_iA_j$ as $X$;
    if $|A_kA_i| > |A_kA_j|$ then
        return Check($Tr$, $X$, $[A_kA_i]$, $\lambda$);
    else
        return Check($Tr$, $X$, $[A_kA_j]$, $\lambda$);
    end

Function FlipIfNeeded($Tr$, $A_1$, $[A_2A_3]$)
    if $[A_2A_3]$ is unconstrained and adjacent to two triangles of $Tr$ then
        $A_4$ := vertex of $A_4A_2A_3 \in Tr$ such that $A_4 \neq A_1$;
        if $A_4$ is interior to $circumcircle(A_1A_2A_3)$ then
            FlipIfNeeded($Tr$, $A_1$, $[A_2A_4]$);
            FlipIfNeeded($Tr$, $A_1$, $[A_3A_4]$);
        end
    end
```

Algorithm 7.1 – Refinement algorithm.

7.5 Path Planning

7.5.1 Roadmap

As a consequence of Theorem 7.3.1, from a refined constrained Delaunay triangulation of *Obst*, one can extract a roadmap representing the possible ways of moving around those obstacles, by a similar procedure to the one outlined in Section 6.3.3.

This is done by building a graph G whose nodes correspond to the triangles of the triangulation, and in which two nodes are linked if and only if their underlying triangles share a common unconstrained edge.

As in the previous chapter, the feasible paths that can be followed around the obstacles are exactly represented by the paths of G, being careful of only traversing edges that have a length consistent with the size of the robot. Thus, G represents a roadmap that can be searched for paths leading from a triangle to another, using, for instance, Dijkstra's or A*.

It is worth mentioning that a sequence of triangles $T_{i_1}, T_{i_2}, T_{i_3}, \ldots$ extracted from G for a robot of diameter d does not anymore directly translate into a feasible path. Indeed, contrary to the case of a (classical) Delaunay triangulation, in a refined constrained Delaunay triangulation, the center of the circumcircle of a feasible triangle may be unfeasible, for instance, if it is too close from a constrained segment.

Hence, the path that visits the center of the circumcircle of each triangle in the sequence is not necessarily feasible[3], but as a consequence from Theorem 7.3.1, at least one feasible path exists in the corresponding channel. Therefore, we can still apply the generalized funnel algorithm to this channel to compute the shortest feasible path (in the channel).

7.5.2 Origin and Destination

As previously, both the origin and destination locations must clear the obstacles by a distance that is a least equal to $d/2$ (i.e., these two positions must be feasible), otherwise the path planning problem can not be solved.

An interesting property of Delaunay triangulations, is that a vertex is always connected by an edge to its nearest neighbor vertex (in other words, the nearest neighbor graph of a set of points is a subgraph of the Delaunay triangulation of those points [BCKO08]). Moreover, by definition, in a constrained triangulation refined according to our procedure, a vertex cannot be closer to a constrained edge than to all the other vertices. We can therefore use this property to efficiently determine whether a location is feasible or not.

[3] Even if such a path is not feasible, weighting the edges of the graph with the distance between the centers of the circumcircles of the underlying triangles still manages to find the near-shortest path in the plane.

Indeed, if we add the point P to a refined triangulation using, for instance, the efficient insertion[4] algorithm presented in [MGD03], and then apply again the refinement algorithm to the affected triangles, then, in order to determine whether P is feasible, we will only need to check the distance between P and all the vertices to which it is directly connected.

7.5.3 Departure and Arrival Clearance

From Theorem 7.3.1, we know that if a path that visits a sequence of triangles $T_{i_1}, T_{i_2}, \ldots, T_{i_m}$ is extracted from the roadmap G, for a robot of diameter d, then there exists a feasible path for this robot that lead from at least one point in T_{i_1} to at least one point in T_{i_m}. But we have no guarantee that the origin O located in T_{i_1} or the destination D located in T_{i_m} will correspond to one of those points.

In fact, in a classical Delaunay triangulation, if two points belonging to the same triangle are feasible, then there always exists a feasible path that joins them, but in a refined triangulation this property does not hold anymore, in particular, for obtuse triangles. This situation is depicted in Figure 7.8, where the distance between the constrained segment $[A_iA_j]$ and the vertex A_1 (that belongs to T_1) is inferior to d. Hence, a robot of diameter d cannot pass between them, while there exist feasible positions for this robot in T_1 on both sides of this passage.

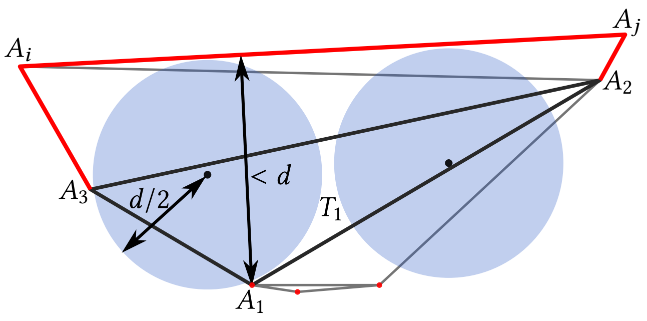


Figure 7.8 – Feasible positions in a triangle that cannot be connected by a feasible path. (As usual, obstacles are depicted in red.)

The point A_1 is then problematic for a path that starts or ends in T_1. The idea to solving this problem is the same as for our refinement procedure: we add additional Steiner points to our triangulation in such a way that this situation cannot exist anymore.

This is achieved, by adding the origin or the destination point to the refined constrained triangulation, in the exact same way as for the feasibility check. Then we can remove this

[4] Note that, in order to take into account our relaxed triangulation definition (i.e., the fact that we allow an edge of a triangle to be composed of a series of collinear constrained segments instead of a single one), the local search to determine the triangle in which the point should be inserted must of course be slightly adapted. In the same way, constrained edges cannot be flipped by the insertion algorithm.

point from the triangulation using, for instance, the efficient deletion[5] algorithm also presented in [MGD03], and, finally, apply again one more time the refinement algorithm to the affected triangles.

Figure 7.9 illustrates the result of the insertions and deletions of both the origin O and destination D locations to the same triangulation as the one depicted in Figure 7.8. This yields a triangulation for which Theorem 7.3.1 now applies for paths that lead from O to D.

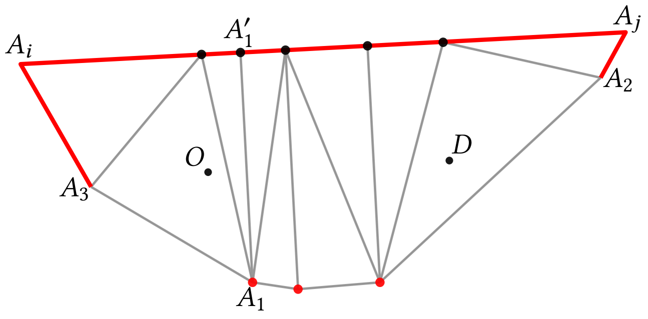


Figure 7.9 – Insertion and deletion of successively the origin and destination locations. The orthogonal projection A_1' of A_1 onto $[A_iA_j]$ has, for instance, been added as a Steiner point.

In fact, these local operations now ensure that the triangles that contain O and D do not contain vertices that are problematic for the start or the end of the path.

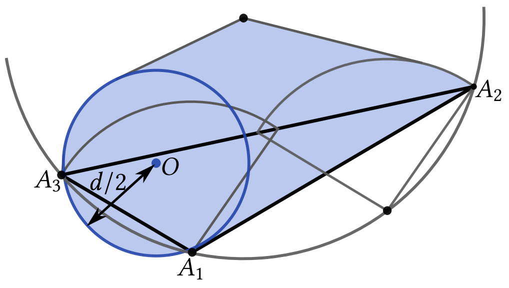


Figure 7.10 – Extended safe zone.

Indeed, after adding and removing successively O and D to the refined constrained triangulation, the triangles that now contain the feasible positions O and D for a robot of diameter d include an extended safe zone that cannot be crossed by constrained segments. This extended safe zone is illustrated in Figure 7.10. Every robot that is able to traverse respectively the first or the last interior edge of a channel, can perform this operation while remaining confined in the increased safe zone of the triangles that respectively contain O and D.

[5] Similar remarks as for the insertion algorithm apply.

7.5.4 String Pulling

In order for the generalized funnel algorithm to behave correctly, the exact same issue as the one described in Section 6.3.5.3 has to be taken into account.

After adding and removing both the origin and destination to a refined triangulation, every channel extracted for a robot of diameter d admits a least one feasible path that pass by this channel. But, since the two triangles at the extremities of a channel (i.e., those that contain O or D as vertex) are not triangles of the triangulation, they do not satisfy the Delaunay criterion. Therefore, there may still exist obstacles that could interfere with the shortest path at the beginning and the end of the channel (i.e., before the first and after the last interior edge).

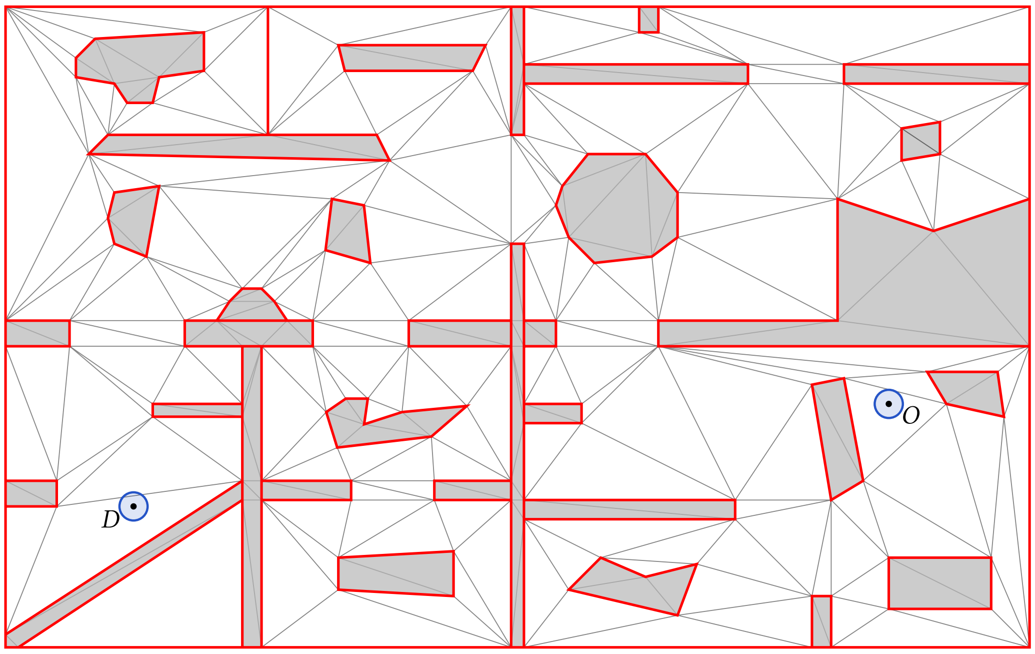
One could remark that, due to Delaunay's criterion, the vertices potentially located in the problematic area described in Section 6.3.5.3, are necessarily closer to O (or D) than to any other vertices (located outside this area). Thus, when we check whether O (or D) is feasible, if the problematic area contains vertices, then at least one of them must be connected to O (or D). During this check, one can then store the list of vertices to which O or D is connected, and use it afterwards in order to efficiently determine whether the problematic area is empty[6] or not.

7.5.5 Illustration

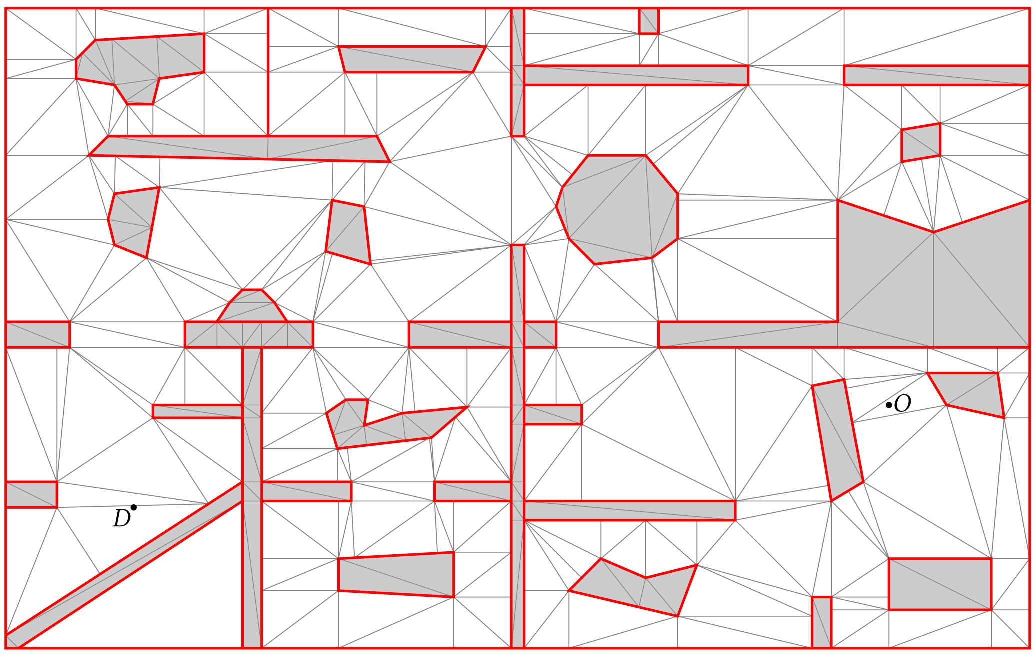
We illustrate the successive steps of the path planning method introduced in this chapter in Figure 7.11. Figure 7.11(a) shows an area with polygonal obstacles, origin O and destination D locations (along with the required safety distance). Figure 7.11(a) also shows the constrained Delaunay triangulation of these obstacles. The refined constrained Delaunay triangulation is illustrated in Figure 7.11(b). Figure 7.11(c) depicts the triangulation after insertion and deletion of the origin and destination locations. Figure 7.11(d) illustrates the roadmap graph that corresponds to the final triangulation.

This graph is then searched, for a robot of the given diameter, and the shortest path from origin to destination is highlighted in green. Figure 7.11(d) also shows the channel that correspond to the shortest path in the graph. Figure 7.11(e) shows the result of the generalized funnel algorithm performed on this channel. Finally, Figure 7.11(f) shows the same result for a larger robot.

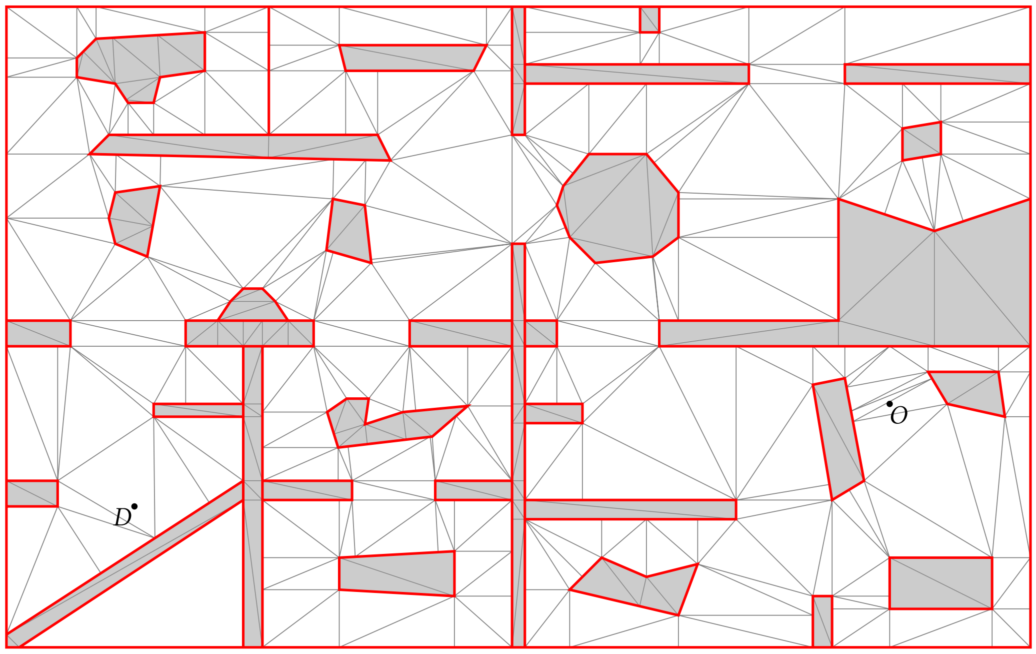
[6] For the same reason, the other vertices in the problematic area (if any), are necessarily connected together, and can therefore be discovered efficiently.



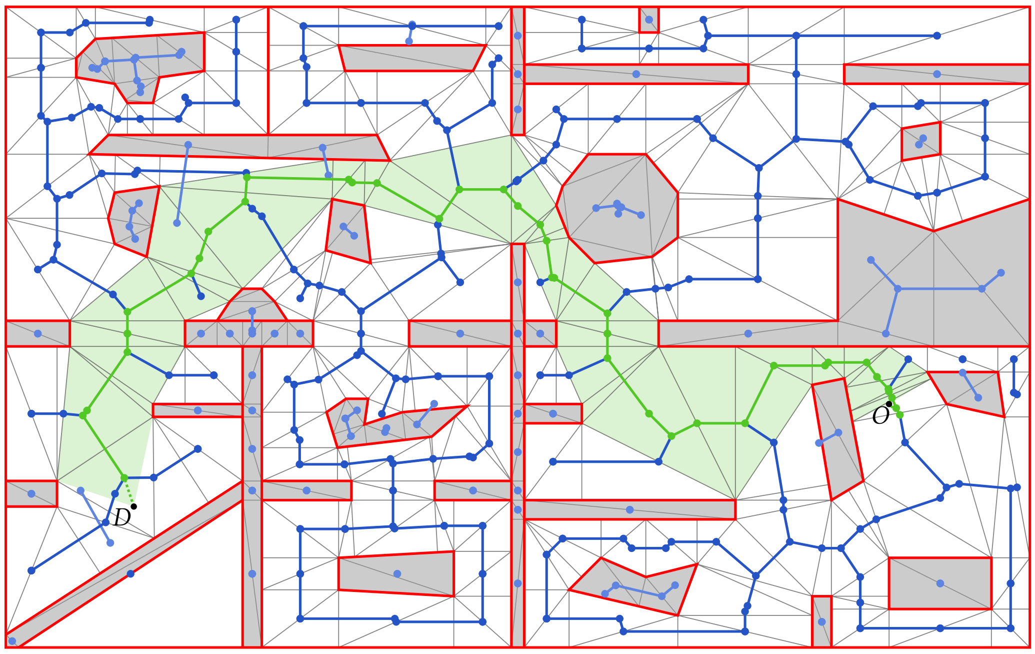
(a) Problem statement and constrained Delaunay triangulation.



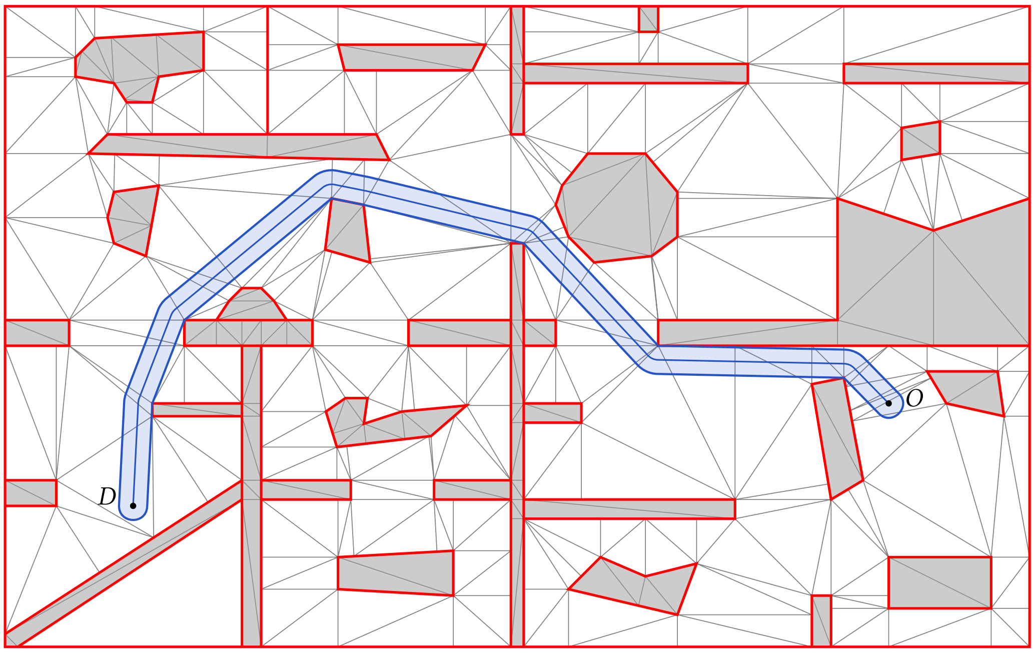
(b) Refined constrained Delaunay triangulation.



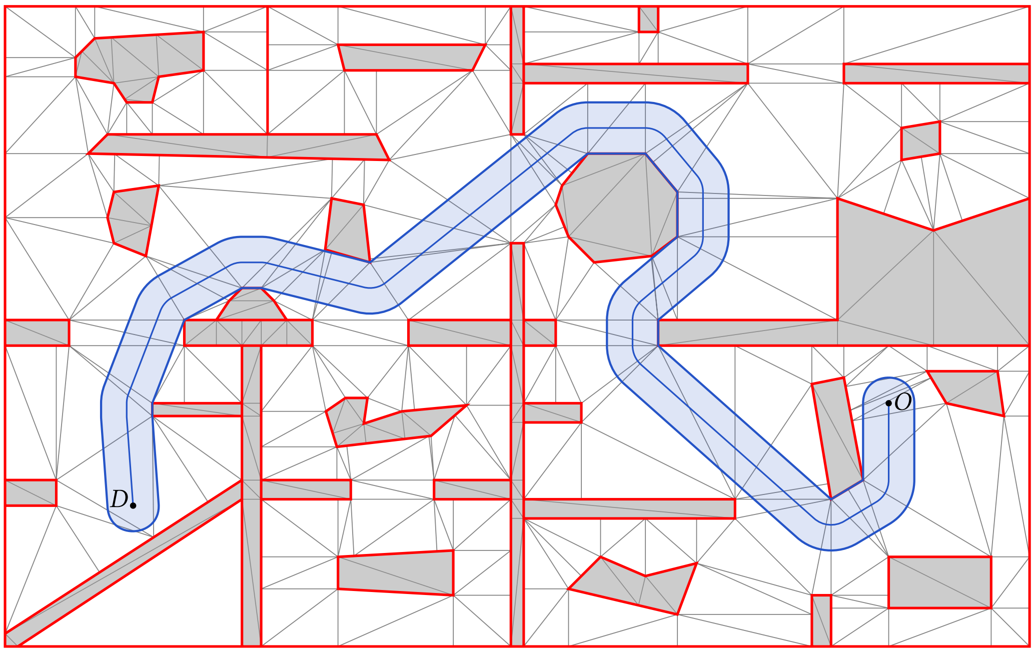
(c) Origin and destination inserted and removed from the refined triangulation.



(d) Feasible arcs of the corresponding graph (shortest path highlighted in green) and extracted channel.



(e) Shortest path in the channel.



(f) Shortest path for a larger robot.

Figure 7.11 – Illustration of path planning with a refined constrained triangulation.

7.6 Experimental Results

We have implemented the refinement algorithm introduced in this chapter, starting from a constrained Delaunay triangulation, in order to compare it to solutions such as [Ger10, Kal14]. In addition to being simpler, our technique also has the advantage of offering significantly reduced execution times, even though it produces slightly larger refined triangulations than [Kal14]. We provide in Figure 7.12 measurements performed on randomly generated sets of obstacles of increasing size, using an i5-460M CPU running at 2.53 GHz.

Points	Triangles	Refined points	Refined triangles	Refinement procedure
1227	2448	1420	2641	0.27 ms
5558	11110	6179	11731	1.23 ms
15042	30078	16644	31680	3.49 ms
30205	60404	33311	63510	9.08 ms
54990	109974	60563	115547	19.78 ms
136168	272330	150022	286184	52.66 ms
325058	650110	358931	683983	128.31 ms
649632	1299258	719149	1368775	260.05 ms
1298879	2597752	1443726	2742599	526.40 ms

Figure 7.12 – Experimental results.

It also worth mentioning that, unlike [Kal14] which requires to store additional information in triangles, this refinement algorithm produces refined triangulations that are compatible with the ones produced by a classical Delaunay triangulation. Therefore, we can directly use (without modifications) the graph search methods or the generalized funnel algorithm described in Chapter 6.

In comparison to the method described in Chapter 6, this refinement algorithm clearly adds a smaller number of (Steiner) points to the triangulations than the number of points needed for discretizing the obstacles. This results in much simpler triangulations, and therefore, in much more effective graphs searches.

The constrained Delaunay triangulation of a set of points and line segments can be computed in $O(n \log n)$ time, where n denotes the total number of vertices [Che89, Slo93]. In addition, we have observed experimentally that the refinement algorithm is averaging roughly $O(n)$ in time for the case studies that we have considered. This results practically in a path planning method that is more computationally efficient than the one described in Chapter 6. It worth noting that, determining the theoretical worst-case complexity of the refinement procedure has proved to be difficult and it remains an open problem[7].

[7] Note that, using the simple heuristic that consists of first refining the triangles that have one and only one constrained segment, we were unable to construct cases where the computational cost of the refinement procedure was worst than linear in the number of triangles in the constrained Delaunay triangulation.

Our solution thus allows near-optimal path planning in real-time for larger problems than previously feasible. In the framework of the Eurobot contest, this made it possible to evaluate several strategies (e.g., in order to check the time needed to perform different actions) and to choose the most promising one in real-time.

In order to compare the methods described in Chapter 6 and 7, we provide in Figure 7.13 the time needed to perform the various steps of both procedures. The problem[8] considered is the one illustrated in Figure 7.11. For this problem, the refinement algorithm allows us to find the near-shortest path more than 8 times faster than the discretization-based approach.

	Discretization-based	Refinement-based
Points	2131	159
Triangles	3740	291
Refined points	/	280
Refined triangles	/	412
Triangulation	1819.87 μs	168.95 μs
Refinement procedure	/	28.60 μs
Origin & Destination	7.56 μs	7.70 μs
Dijkstra's	250.64 μs	36.58 μs
Funnel	18.74 μs	5.17 μs
Total time to find the short path	2096.81 μs	247.00 μs

Figure 7.13 – Comparison between discretization-based and refinement-based path planning for the problem illustrated in Figure 7.11.

Note that, even if it is difficult to directly compare our method to other approaches, since shared datasets and implementations are usually not available, it nevertheless clearly appears — based on the execution times claimed by the authors for problems with similar complexity — to be much faster than [BG08, Ger10, JSTL14, Kal14].

Even if popular probabilistic approaches, such as PRM (Probabilistic Roadmaps) [GO04] or RRT (Rapidly-exploring Random Trees) [LaV98], are usually the most efficient methods for path planning in complex high-dimensional configuration spaces, they do not necessarily provide the best solutions for mobile robot moving in two-dimensional space. Indeed, in our case (path planning of a cylindrically shaped robot in an Euclidean plane), using a triangulation-based path planning usually results in lower running times and in better quality paths than using probabilistic methods.

[8] Note that for this problem, both methods produce the exact same shortest path. This does not necessarily happen in all cases. Indeed, due to the approximated metric used in the graph search, the two methods may produce two different short paths for a particular problem (but this seldom happens). In such a case, both paths are very similar or at least have similar lengths.

It is also worth mentioning that, in order to behave correctly, probabilistic methods rely on efficient collision detection algorithms, which requires to preprocess the obstacles to create data structures (e.g., quadtree, k-d tree or even a triangulation) that are necessary for their proper operation. Since these data preprocessing operations are more or less equivalent to computing a Delaunay triangulation of the obstacles, probabilistic methods cannot, in any way, be much faster than the method presented in this chapter.

Furthermore, even if we ignore this aspect, probabilistic methods also have other drawbacks. First, due to their probabilistic nature, PRM or RRT methods produce paths that usually contain lots of unneeded direction changes, clear obstacles at unnecessarily large distances, and do not exploit narrow passages. This results in much longer paths, with usually a lower quality[9] than the ones produced by methods that rely directly on a Delaunay triangulation (i.e., after applying the funnel algorithm).

Finally, probabilistic methods do not provide roadmaps suited for path planning of robots with an arbitrary size. Indeed, RRTs can only do single query planning (i.e., fixed origin and destination locations, and fixed size), and PRMs provide only roadmaps that are usually incomplete and only suited for robots of a fixed given size. Therefore, replanning or evaluating several strategies (i.e., testing different goals or different clearances) becomes computationally expensive.

[9] Methods have been proposed to optimize such paths, but they are quite ineffective in comparison to the generalized funnel algorithm.

CHAPTER 8

Experimental Results and Conclusions

"It doesn't matter how beautiful your theory is,
it doesn't matter how smart you are.
If it doesn't agree with experiment, it's wrong."

— Richard P. Feynman

8.1 Experiments

This section describes some of the experiments performed in order to evaluate the performances of our complete trajectory planning approach; i.e., the combination of the path planning algorithms of Chapters 6 and 7, the path interpolation method of Chapter 5 and the speed profile computation procedure of Chapter 4.

8.1.1 Experimental Setup

As already mentioned, this work has primarily been motivated by the Eurobot contest, in which small autonomous mobile robots have to compete against opponents in a $2 \times 3\,\mathrm{m}^2$ area in the presence of obstacles.

In this framework, it is essential to synthesize fast and accurate trajectories in real-time. Indeed, obtaining accurate advance information about the locations that will be visited by the robot and their associated timestamps is crucial to implementing coordinated actions between the locomotion system and other actuators, or between several robots. In addition, generating trajectories that can be followed without slowing down in the vicinity of obstacles and that manage to quickly reach their target provides a clear advantage.

In order to participate to this contest, several differential drive and tricycle robotic platforms have been developed since 2008 at the University of Liège. Each platform is composed of a low-cost embedded computer running real-time Linux, together with several custom-made electronic cards. Our algorithms have been implemented and evaluated in these various robots, which amounts to hundreds of thousands of trajectories successfully synthesized.

These robots are cylindrically shaped (or at least can be over-approximated by cylinders) with diameters that vary between 20 and 40 cm. They can achieve speeds of more than $1.5\,\mathrm{m\,s^{-1}}$ and accelerations of more than $3\,\mathrm{m\,s^{-2}}$.

With the exception of the constrained Delaunay triangulation computation for which we use the efficient and robust C library Triangle[1] developed by J. R. Shewchuk [She96], all the algorithms and methods described in this thesis have been implemented by us in C and are not based on external[2] libraries. This strategy allows better control of the memory consumption and of the overall efficiency, which are of primary interest for an embedded real-time application. Our C implementation of trajectory planning amounts to more than 17500 non-commented lines of code spread over thirty-two source files.

In addition to the Eurobot contest, which was a remarkable way to confront the theory with real world, in order to evaluate the robustness and the performance of our solution in greater depth, we have also tested our algorithms on much complex problems, like navigating in buildings, mazes, or between randomly generated sets of hundred of thousands of obstacles. Some of these problems are presented in the next subsection.

8.1.2 Experimental Results

The problem instances considered in these experiments were constructed as follows. Problem 1, illustrated in Figure 8.6, represents a particular problem instance encountered in the Eurobot framework. Problem 2, illustrated in Figure 8.7, depicts a building in which the robot has to move. Problem 3, illustrated in Figure 8.8, represents a maze that comes from the KrazyDad.com maze collection, in which the robot must find its way out. Problem 4, illustrated in Figure 8.9, represents the labyrinth that was installed, until 1779, on the floor of the nave of the Cathedral of Reims, for which the aim is to find the way to the center.

[1] In order to make fair comparisons, we also use Triangle to benchmark the computations of the classical Delaunay triangulation. It also worth noting that Triangle efficiently detects potential intersections between constrained segments, and splits them if necessary, which can be useful for complex problems.

[2] At the exception of the GNU C Library.

Problems 5 to 9 are randomly generated sets of obstacles of increasing size. Problems 5 to 7 are illustrated in Figures 8.10 to 8.12. Figures for Problems 8 and 9 are not provided (since they are too complex to fit on a A4 page), but are similar to Figure 8.12 with larger numbers of obstacles. Generated obstacles are convex polygons of random (bounded) sizes with a random number of vertices (from 3 to 10). During the construction of these sets, each new polygon is randomly added to the area while ensuring a roughly uniform density of obstacles.

All these problems have been evaluated with a tricycle robot, using the physical parameters summarized in Figure 8.1.

Axle width:	27 cm
Wheel base:	18 cm
Absolute maximal speed of the steering wheel:	$1.3\,\mathrm{m\,s^{-1}}$
Absolute maximal acceleration of the steering wheel:	$1.0\,\mathrm{m\,s^{-2}}$
Absolute maximal tangential acceleration:	$1.0\,\mathrm{m\,s^{-2}}$
Absolute maximal radial acceleration:	$1.0\,\mathrm{m\,s^{-2}}$
Absolute maximal steering speed:	$6.0\,\mathrm{rad\,s^{-1}}$

Figure 8.1 – Parameters of the tricycle robot.

Figure 8.3 reports various parameters and results related to these nine problems. For each problem, it provides the total number of obstacle vertices and the number of triangles in the constrained Delaunay triangulation. It also shows the final number of points and triangles in the refined triangulation and the number of triangles of the shortest channel, extracted from the roadmap, with respect to the given obstacle clearance (i.e., the minimal distance between the center of the robot and the obstacles). Next, it gives the number of line segments of the broken line that results from the generalized funnel algorithm applied on this channel.

This figure also shows the benefits on the travel time (for the considered tricycle robot) provided by the interpolation procedure. The travel time after the first step of this procedure (i.e., bounding curvature by using circular arcs) is also reported. One can remark that the benefits of the interpolation procedure are clearly significant, even if they, of course, depend on the ratio between the length of the path and the number of orientation changes.

Finally the length of the interpolated path and the number of discretized steps are reported. It worth noting that, as discussed in Chapter 3, the path discretization error can be arbitrarily bounded by choosing sufficiently small sampling steps. In the context of the Eurobot framework, since precision is of primary interest, we use a relatively large number of steps to discretize the paths. Each of these steps has roughly a length of 5 mm. For bigger robots, or in situations where such a precision is not needed, this step can of course be increased, which would result in smaller speed profile computation and discretization runtimes.

The timings reported in Figures 8.4 and 8.5 were obtained by running our implementation on an i5-460M processor at 2.53 GHz (on a single thread).

These results show that the computational cost of planning a smooth, accurate, short and fast feasible trajectory for a robot in an highly cluttered environment of hundred of thousands obstacles amounts to less than a second of CPU time. To the best of our knowledge, this makes our complete solution one of the fastest methods suited for trajectory planning of a mobile robot in the Euclidean plane in the presence of obstacles.

Thanks to these results, one observes on Figure 8.2 that, even though some computation steps such as the Delaunay triangulation theoretically require $O(n \log n)$ time, the overall experimentally measured time complexity of our path planning approach remains $O(n)$, where n denotes the number of obstacles vertices. Part of this good result comes from the fact that the Triangle library that we used to compute the constrained Delaunay triangulation has a linear cost in most cases.

In particular, one also clearly notices that the discretization and the speed profile procedures have a linear cost in term of the number of discretized steps and that the cost of the interpolation procedure is linear in the number of segments in the broken line, in our experimental setting.

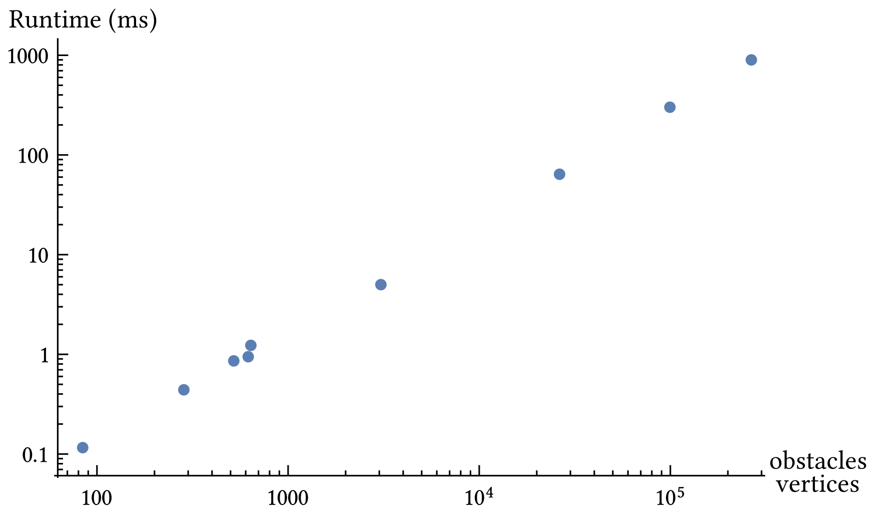


Figure 8.2 – Path planning runtimes for the nine problems considered.

Problem	Points	Triangles	Refined points	Refined triangles	Obstacles clearance	Channel triangles	Broken line segments
1. Eurobot	84	154	127	197	32 cm	32	10
2. Building	284	562	499	777	100 cm	68	11
3. Maze	524	1042	1254	1772	24 cm	397	81
4. Reims	642	1278	1803	2439	16 cm	1144	317
5. Random 1	622	1238	974	1590	20 cm	94	16
6. Random 2	3047	6088	4620	7661	40 cm	199	29
7. Random 3	26249	52492	39349	65592	40 cm	591	50
8. Random 4	99356	198706	149128	248478	40 cm	1104	103
9. Random 5	263464	526922	385161	648619	40 cm	1611	112

Problem	Broken line travel time	Circular arcs path travel time	Interpolated path travel time	Interpolated path length	Discretized steps
1. Eurobot	21.31 s	11.60 s	7.62 s	3.08 m	632
2. Building	72.23 s	60.33 s	47.72 s	57.51 m	11521
3. Maze	220.78 s	148.85 s	96.86 s	38.50 m	8193
4. Reims	985.34 s	748.42 s	675.77 s	246.29 m	53258
5. Random 1	37.86 s	26.11 s	14.72 s	6.88 m	1404
6. Random 2	97.16 s	65.43 s	44.11 s	49.12 m	9860
7. Random 3	306.69 s	251.10 s	205.32 s	259.37 m	51936
8. Random 4	642.14 s	513.61 s	427.90 s	542.14 m	108548
9. Random 5	1175.15 s	1035.44 s	930.61 s	1201.17 m	240357

Figure 8.3 – Experimental results.

Problem	Triangulation	Refinement Procedure	Origin & Destination	Dijkstra's	Funnel	Path planning (Total)
1. Eurobot	77.24 μs	11.96 μs	8.34 μs	14.01 μs	4.81 μs	116.36 μs
2. Building	275.69 μs	80.78 μs	12.37 μs	68.45 μs	8.80 μs	446.09 μs
3. Maze	470.06 μs	123.57 μs	10.34 μs	200.35 μs	50.47 μs	854.79 μs
4. Reims	809.34 μs	175.33 μs	11.28 μs	159.37 μs	147.80 μs	1303.12 μs
5. Random 1	623.43 μs	168.41 μs	11.34 μs	106.52 μs	14.36 μs	924.06 μs
6. Random 2	3257.86 μs	859.30 μs	17.91 μs	818.57 μs	27.64 μs	4981.28 μs
7. Random 3	38.45 ms	10.59 ms	0.06 ms	14.77 ms	0.08 ms	63.95 ms
8. Random 4	190.03 ms	53.57 ms	0.08 ms	60.54 ms	0.14 ms	304.36 ms
9. Random 5	574.54 ms	140.75 ms	0.14 ms	175.56 ms	0.20 ms	891.19 ms

Figure 8.4 – Path planning runtimes.

Problem	Path planning	Interpolation	Discretization	Speed Profile	Total time
1. Eurobot	0.12 ms	0.02 ms	0.06 ms	0.21 ms	0.41 ms
2. Building	0.45 ms	0.02 ms	0.69 ms	2.41 ms	3.57 ms
3. Maze	0.85 ms	0.22 ms	0.72 ms	3.04 ms	4.83 ms
4. Reims	1.30 ms	0.76 ms	5.97 ms	22.06 ms	30.09 ms
5. Random 1	0.90 ms	0.03 ms	0.13 ms	0.45 ms	1.51 ms
6. Random 2	4.98 ms	0.07 ms	0.62 ms	2.06 ms	7.73 ms
7. Random 3	63.95 ms	0.12 ms	5.33 ms	13.21 ms	82.61 ms
8. Random 4	304.36 ms	0.25 ms	11.19 ms	28.73 ms	344.53 ms
9. Random 5	891.19 ms	0.29 ms	23.83 ms	61.70 ms	977.01 ms

Figure 8.5 – Trajectory planning runtimes.

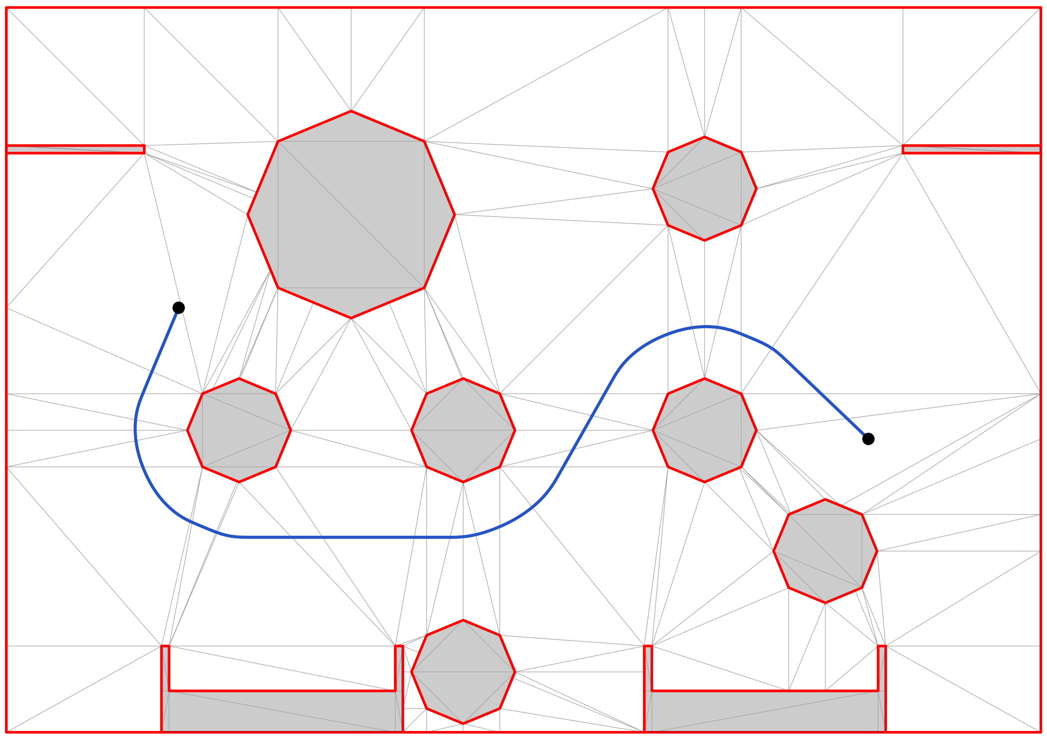


Figure 8.6 – Problem 1: Eurobot.
(area: $2.1 \times 3.0\,\mathrm{m}^2$; obstacles clearance: 32 cm; path length: 3.08 m)

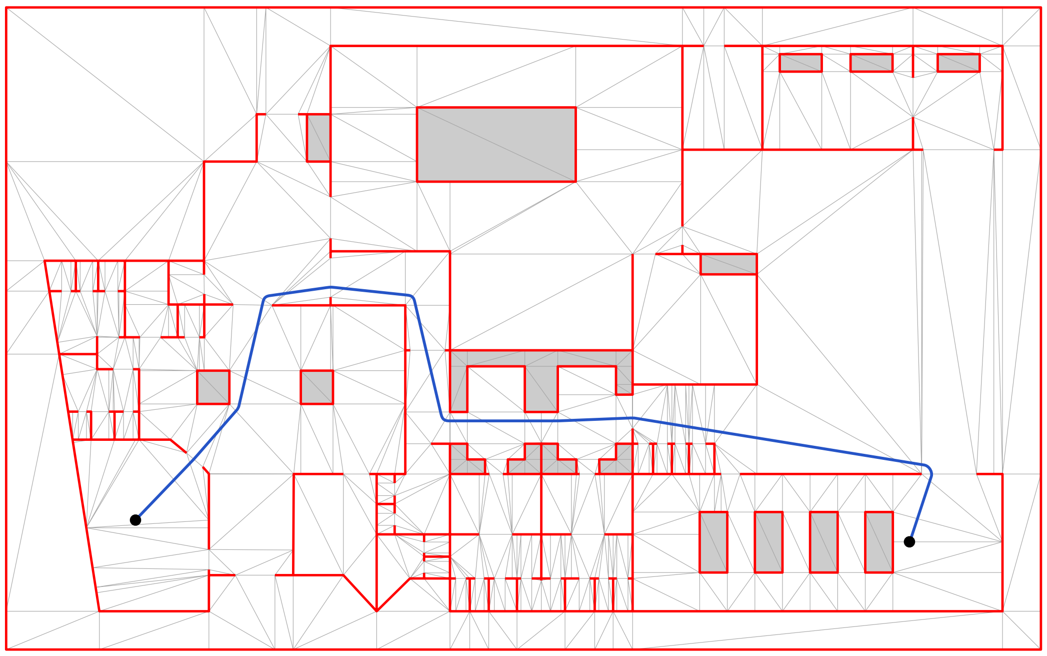


Figure 8.7 – Problem 2: Building.
(area: $33.4 \times 53.8\,\mathrm{m}^2$; obstacles clearance: 100 cm; path length: 57.51 m)

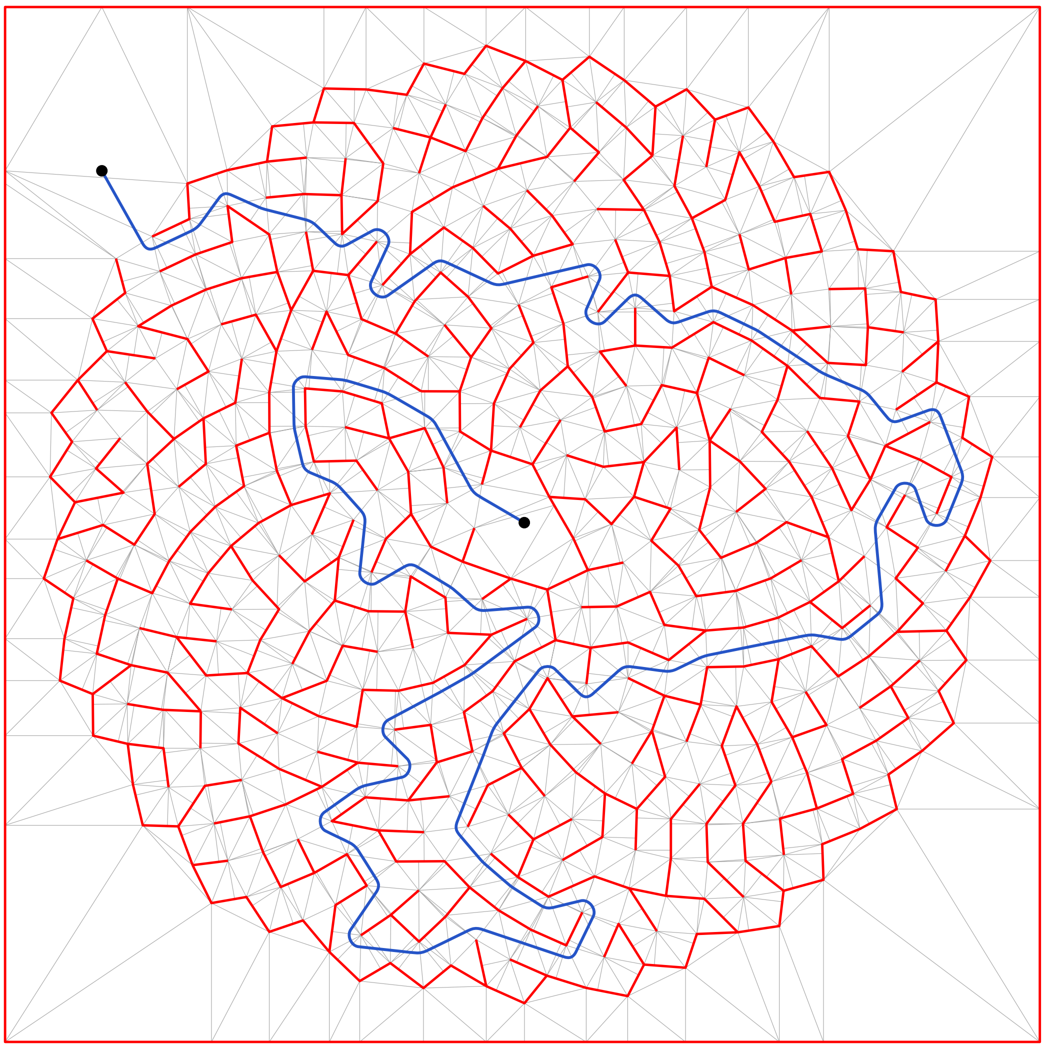


Figure 8.8 – Problem 3: Maze.
(area: 53.5 × 53.5 m^2; obstacles clearance: 24 cm; path length: 38.50 m)
N.B.: This maze is part from the KrazyDad.com collection — http://www.krazydad.com

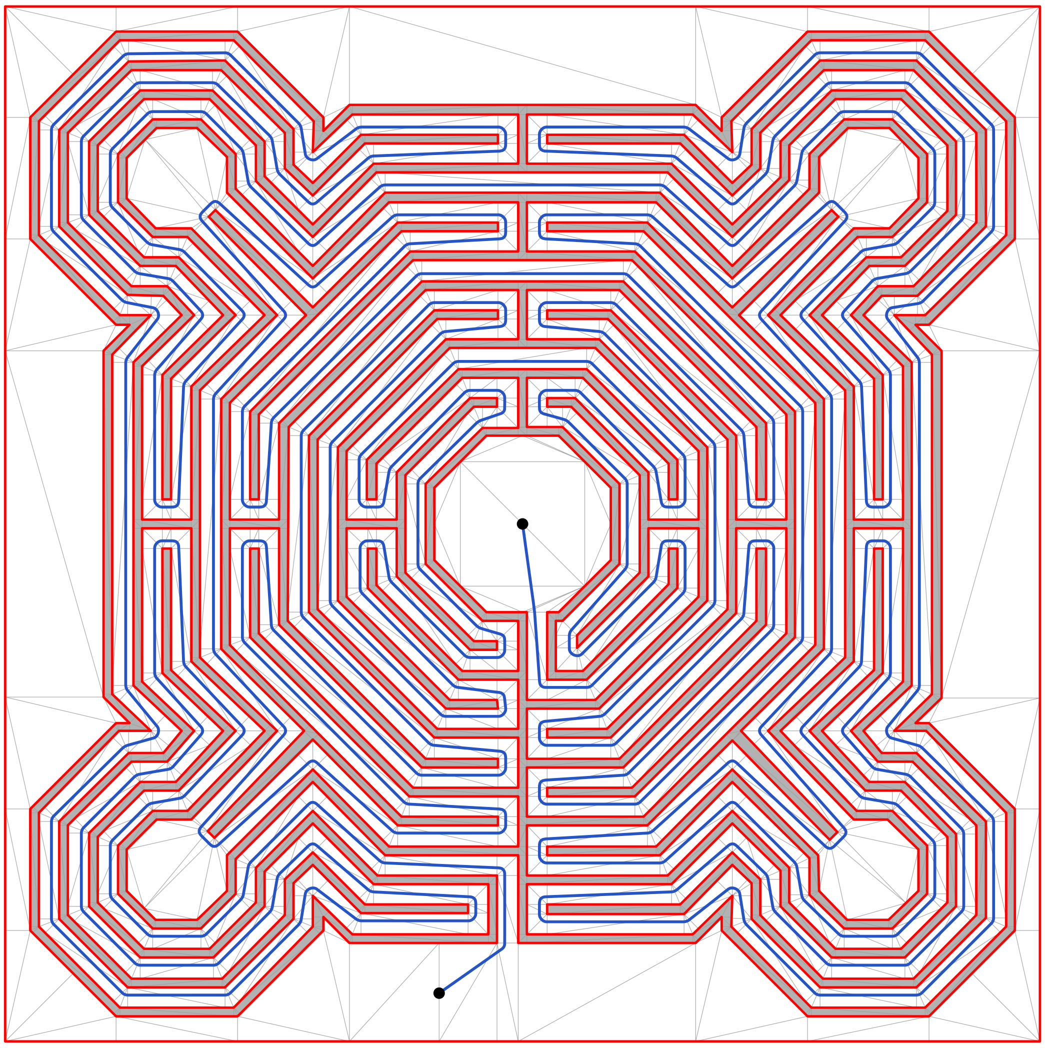


Figure 8.9 – Problem 4: Reims Cathedral.
(area: 10.7 × 10.7 m^2; obstacles clearance: 16 cm; path length: 246.29 m)

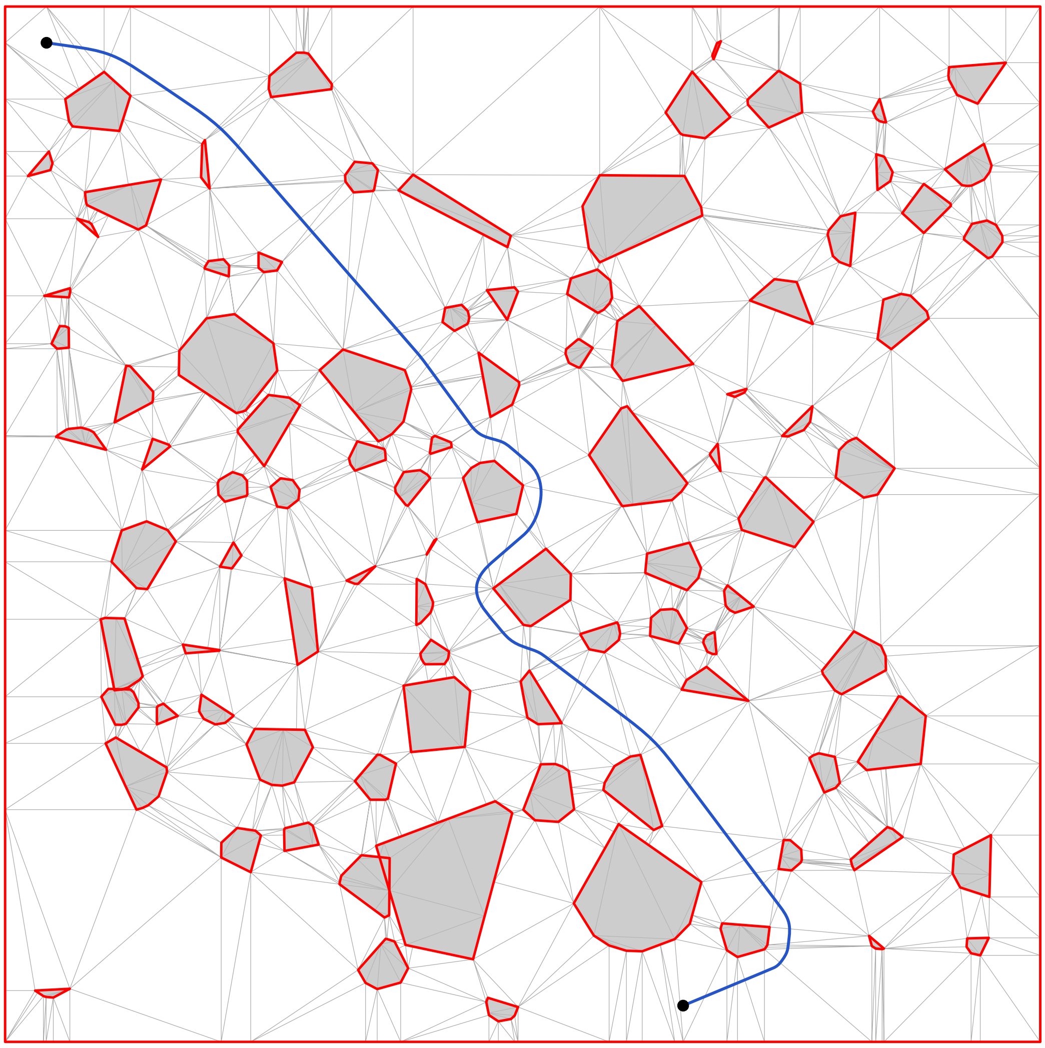


Figure 8.10 – Problem 5: Random 1.
(area: $5 \times 5\,\mathrm{m}^2$; obstacles clearance: 20 cm; path length: 6.88 m)

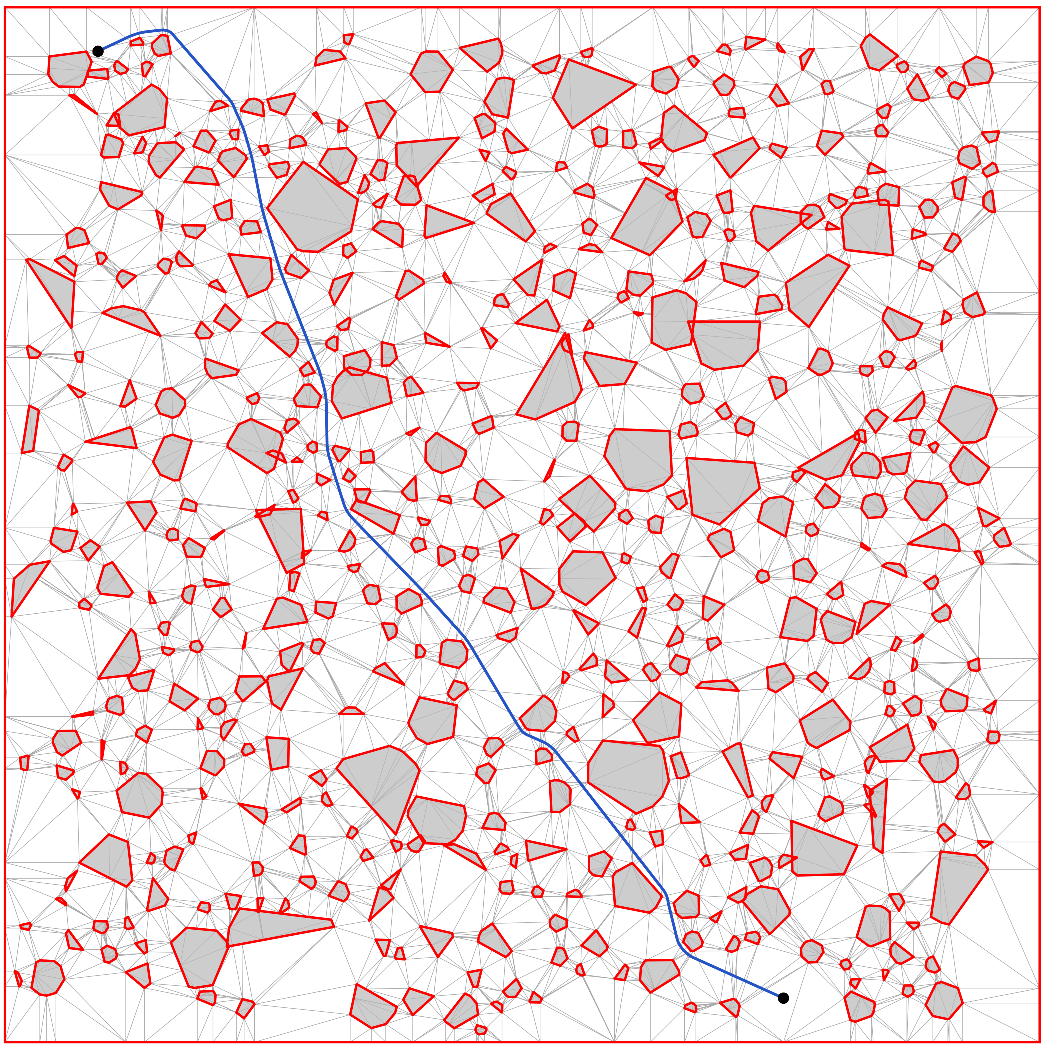


Figure 8.11 – Problem 6: Random 2.
(area: 40 × 40 m^2; obstacles clearance: 40 cm; path length: 49.12 m)

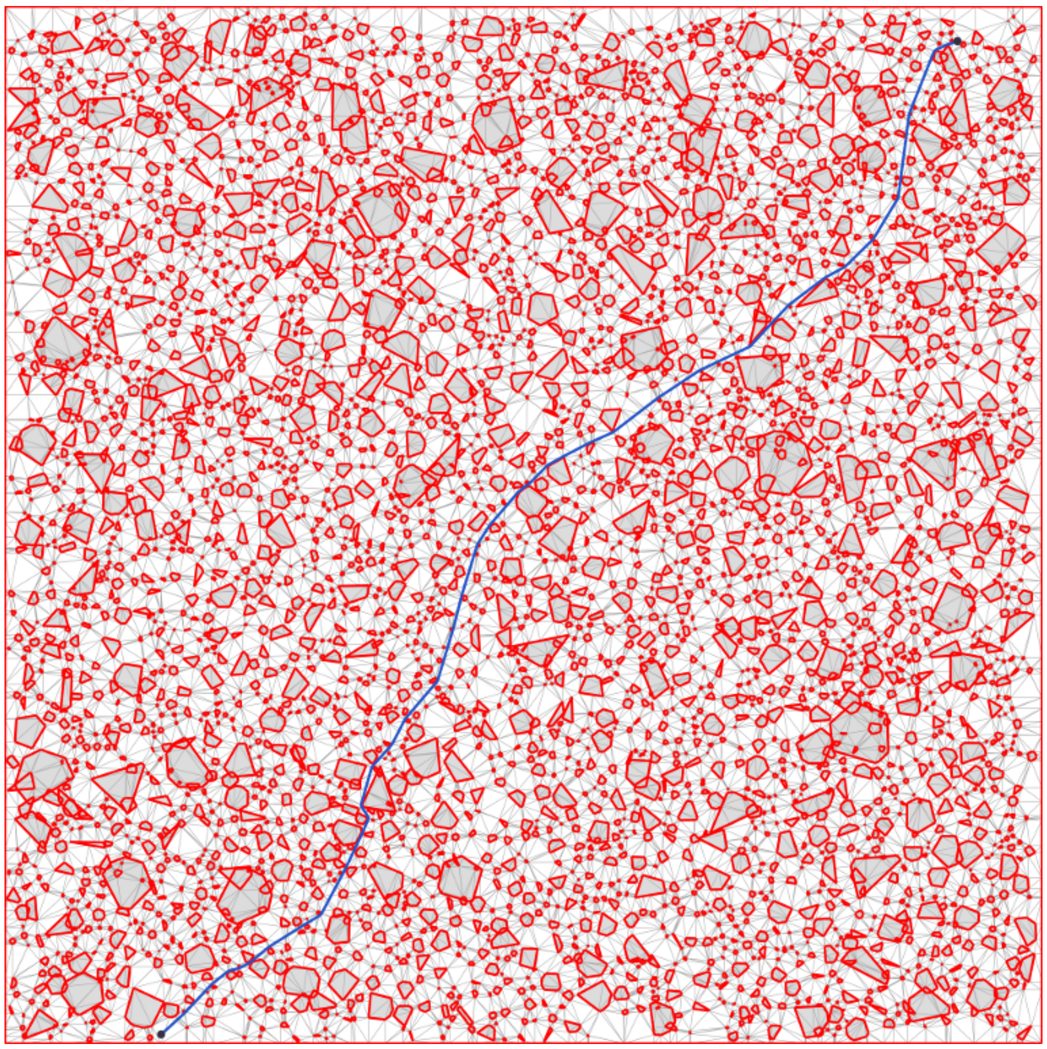


Figure 8.12 – Problem 7: Random 3.
(area: $200 \times 200\,\text{m}^2$; obstacles clearance: 40 cm; path length: 259.37 m)

8.2 Comparison with Other Work

Since this work has been primarily motivated by a contest, some of its aspects are atypical. Unlike some other approaches, our strategy was always to keep a pragmatic viewpoint in order to develop a complete solution that can be used on a real mobile robot with a low computational cost. The idea was not to generate optimal trajectories, but still do our best to generate trajectories that manage to reach their target in acceptable time.

This pragmatic viewpoint led us to a solution in which the motion planning problem has been broken into several (simpler) subproblems. For each of them, we used existing efficient solutions when possible, and concentrated our efforts in improving or developing new ones when necessary. The result takes the form of a complete workflow that has been successfully implemented and tested in a real application. This workflow produces, in real-time, smooth, accurate, and fast trajectories that are consistent with the various physical constraints of the robot, which provides, in our case studies, a significant advantage over other approaches.

The key to efficiency is that each subproblem is, as much as possible, decoupled from the other ones, the tradeoff being that the generated trajectories are not guaranteed to be optimal (neither the shortest or the fastest). Thus, in comparison to global approaches such as [RW98, BK08], our method generates slower trajectories but with a smaller computational cost. Note that, since we have environmental uncertainties (drift, sensor errors, modeling inaccuracies, ...) anyway, obtaining global optimality at all cost is not practically relevant. Moreover, we have seen in Chapter 6 that the solutions generated by Delaunay-based methods, like ours, are usually close to the shortest ones (using a suitable metric during the search).

We have divided the motion planning problem into three main subproblems. The first one is aimed at finding a path that avoids obstacles and manages to reach the destination, without taking into account nonholonomic constraints. Such a path takes the form of a sequence of straight line segments that clears the obstacles at a specified safety distance. The second step is to smooth this path, in order to reduce the minimal time needed by a real robot to follow it. The third step is to convert the path into a fast trajectory by computing a time-optimal speed profile that can be followed by the robot, taking into account its physical constraints.

We have already compared our solutions to these subproblems to other relevant work in their corresponding chapter; a brief summary of these comparisons follows.

First, the path planning method that we have developed in Chapters 6 and 7 is based on the refinement of a constrained Delaunay triangulation of the obstacles. From this refined triangulation we obtain a roadmap suited for path planning of robots with an arbitrary size. This roadmap is then searched and the extracted path is optimized by the generalized funnel algorithm. The main advantage of this approach is its efficiency, with the drawback that, unlike techniques that rely on visibility graphs or visibility-Voronoi complexes [WBH05], it does not generate the shortest feasible path in the plane. Nevertheless using a suitable metric during the graph search, we know that generated paths are usually short in comparison to the global shortest ones [Kal10].

Our technique is also faster than other existing roadmap approaches such as [BG08, Ger10, JSTL14]. Compared with the method developed in [Kal14] that relies on a similar refinement of a constrained Delaunay triangulation, our technique has the advantage of being simpler, easier to implement, and offers significantly reduced execution times, even though it produces slightly larger refined triangulations.

Finally, our path planning technique is also faster and produces shorter paths than probabilistic path planning methods such as probabilistic roadmaps (PRM) [GO04] or rapidly exploring random trees (RRT) [LaV98], the drawback being that it can only handle cylindrically shaped robots (for which the orientation is irrelevant). This is usually not problematic, since lots of actual mobile robots satisfy this property, or can be approximated in this way.

Next, since we do not consider nonholonomic constraints during the path planning phase, the path is interpolated in order to obtain a smooth path that can be precisely followed by the robot without slowing down excessively. This interpolation, developed in Chapter 5, is performed in two steps, the first one being aimed at producing a path in which the absolute curvature is bounded at all points, and the second one modifying this path, using pairs of clothoids, in order to now bound the rate of variation of curvature. In both steps, the interpolation is computed in such a way that the resulting path still clears obstacles.

Compared with other interpolation methods such as [SS90, KH97, FS04, BP11a, ESJ15], our approach of joining only two arcs of clothoids for moving from one curvature to another has the advantage of being simpler and computationally cheaper, the tradeoff being that the generated curves are not guaranteed to be optimal (i.e., with respect to the travel time). But, none of the other approaches guarantee optimality either. Compared with other simple approaches that only use circular arcs to interpolate paths, such as [KS12], our method clearly generates smoother paths (i.e., without curvature discontinuities) that can be travelled faster.

Note also that, unlike some path planning methods that directly take into account kinematic constraints of robots, such as [SF96, FS04] (that also rely on clothoids), [LKJ00, KKT+09] (that rely on RRT), or [TMD+06, CE07, UAB+08] (that are used in autonomous cars), our method is not able to deal with complex maneuvers that require to change the direction of motion (i.e., such as parallel parking), but, since cylindrically shaped robots usually do not impose bounds on the curvature of the paths (i.e., they are able to make rotations around their center), such maneuvers are, in any case, unnecessary to reach the destination locations.

After the interpolation procedure, the last subproblem consists in computing a time-optimal speed profile that corresponds to the interpolated path, i.e., the speed profile which minimizes the total time needed by a robot for following the path while being consistent with the various physical constraints of the robot.

Our solution to this problem first consists of an original discretization procedure (introduced in Chapter 3), which can efficiently and precisely represent arbitrary paths. In comparison with other techniques that use analytical closed-form formulation such as [LJTM94, CC00, FS04, BP11b, ZT13], this discretized representation eases the different manipulations that can be done on trajectories and, in particular, allows efficient and accurate trajectory interpolation and resampling.

In addition, this representation also allows to develop a time-optimal speed profile computation procedure (in Chapter 4), which has not only proved to be several orders of magnitude faster than techniques such as [CC00, BP11b, KS12], but is also easier to implement, and supports a much broader range of physical constraints, such as individual constraints on the various wheels, constraints on the steering mechanism or even context-sensitive constraints (like imposing tighter speed bounds in the vicinity of some obstacles, or expressing the maximal acceleration of a wheel as a function of its speed).

Finally, the synthesized trajectory (i.e., the combination of the discretized representation of the path and of its associated speed-profile) provides accurate advance information about the locations that will be visited by the robot and their associated timestamps, which is necessary to implementing coordinated actions between the locomotion system and other actuators, or between several robots. Compared with efficient online trajectory planning methods such as [Kha86, FBT97, TMD+06, KKT+09], which, due to their reactive nature, cannot provide such information, this clearly constitutes an important advantage. Note also that, for the same limited horizon reasons, online methods usually produce longer and slower trajectories than our method.

8.3 Conclusions

Autonomous mobile robots are used increasingly in various fields, such as transporting materials (like the thousand of robots in the Amazon warehouses), making domestic tasks (such as mowers or vacuum cleaners) or transporting passengers (like driverless taxis). In order to execute their actions correctly, one of the key problem for those robots is to automatically decide which move to make in order to reach a goal while taking into account environmental constraints, which amounts to trajectory planning.

The research described in thesis addresses this problem, and although it was first motivated by the participation to a robotic contest, it provides algorithms that may certainly be useful in other contexts, in particular, when the available amount of embedded computing power is limited.

Indeed, the combination of our path planning, interpolation and speed-profile techniques provides a complete trajectory planning approach that is, to the best of our knowledge, one of the most computationally effective method suited for the motion of cylindrically shaped mobile robots in two-dimensional space.

In addition to this complete trajectory planning solution, we can highlight some other important contributions of this work.

First, in Chapter 3 we have developed an original data structure that is able to efficiently represent the spatial properties of arbitrary paths that can be followed by nonholonomic mobile robots. This structure eases the different manipulations that need to be performed on such paths, and allows efficient and accurate interpolation and resampling.

Second, built on top of this data structure, we have presented, in Chapter 4, an algorithm for computing time-optimal speed profiles for arbitrary paths. Such a speed profile associates each point of the path with a timestamp that provides the instant at which it will be visited by the robot, in such a way that it minimizes the total time needed by the robot to travel the whole path, while satisfying at all times the kinematic constraints of the robot. This speed profile is important not only to allow the robot to quickly reach its destination location, but also to provide the advance information necessary to implementing coordinated actions (e.g., between the locomotion system and other actuators).

This algorithm has not only proved to be faster, but is also simpler and easier to implement than other existing methods. Moreover, it supports a much broader range of physical constraints, such as context-sensitive constraints or individual constraints on the various wheels. Note also that, this speed profile computation procedure can be used in combination with other path planning techniques, since it does not make any assumptions on the path geometry.

Third, in Chapter 5, we have developed a new algorithm for efficiently interpolating a broken line, i.e., a sequence of connected straight line segments, into a smooth curve, while remaining at a safe distance from obstacles, which produces, for physically realistic robots, a faster speed profile than for broken lines. Similarly to other work, the interpolated curve is not guaranteed to be optimal (i.e., with respect to the travel time), but our method is simpler and faster than other existing techniques.

Finally, after the description in Chapter 6 of an efficient path planning method for cylindrically shaped robots based on a Delaunay triangulation of points obstacles, we have presented in Chapter 7 a generalization of the method in the case where obstacles can be represented by polygons. This generalization relies on an original refinement procedure of a constrained Delaunay triangulation of the obstacles. It has proved to be one of the fastest approach for path planning of cylindrical robots among polygonal obstacles.

It also worth mentioning that, in order to use our complete trajectory method in actual applications, additional problems need to be solved, that are not addressed in this thesis. For instance, robots have to follow the computed trajectories. For Eurobot, we have developed control algorithms that are derived from [MS93]. In a few words, we use multiple PID control loops to not only accurately following spatially the trajectory (i.e., the path really followed by the robot should stay close to the computed one), but also accurately following it temporally (i.e., the time at which locations are reached should correspond to the computed ones).

In some applications, robots also have to deal with dynamic obstacles, such as, in the context of Eurobot, opponent robots. Since movements of these robots are somehow unpredictable, we solve this problem by using a strategy module that is capable of estimating the probability that a trajectory will conflict with one of these dynamical obstacles in the near future, and we compute braking trajectories and perform replanning when potential short term collisions are detected. The discussion of these mechanisms is out of scope of this thesis.

To conclude, as already mentioned several times, the results presented in this thesis have been successfully implemented in the robots built for Eurobot at the University of Liège. In this framework, the very low computational cost of our complete approach and the accuracy of the synthesized trajectories made it possible to perform near-optimal trajectory planning in real-time and plan coordinated actions, which provided a clear competitive advantage over the designs of other teams.

8.4 Future Work and Perspectives

To conclude this thesis, we discuss a non-exhaustive list of improvements or perspectives that it has opened.

First, with the advent of unmanned aircraft such as quadcopters, trajectory planning for free-flying robots clearly becomes an important topic. Thus, an interesting problem would be to extend this work to three-dimensional space. In particular, it would be challenging to translate to 3D our refinement procedure for constrained Delaunay triangulations.

Next, it has been shown that 3D Delaunay triangulations can take advantage of parallelization [CMPS93]. A natural question would be to study whether our refinement procedure can also take advantage of such parallelization.

Moreover, in the case where only small incremental changes are made to the environment, it is clearly ineffective to recompute the complete refined triangulation. One could detect such situations and develop algorithms to efficiently update the refined triangulation.

It should also be possible to efficiently avoid the insertion of some Steiner points during the refinement procedure, for instance, in the case where some bounds on the size of the robots are known. For example, one could avoid refining triangles that are clearly too narrow to be traversed by the robot.

In some applications, it is less crucial to quickly reach destination locations, therefore, the speed-profile computation procedure could be extended to take into account other optimization criteria, such as maximizing the energy efficiency, or ensuring that the robot reaches a given position at a certain fixed time, in order to make, for instance, rendezvous points.

As already discussed, trajectories that are synthesized by our method are not optimal (in the sense that they are not the fastest feasible ones). By using richer information than the clearance parameters, it could be possible to allow slightly larger path deformations in the interpolation procedure. This could result in paths that can be followed even faster.

Finally, one of our future project is to make available the implementation of the algorithms presented in this dissertation for anyone who would want to use or evaluate them.

“Roads ?
Where we’re going, we don’t need roads.”

— Back to the Future (1985)

APPENDIX A

Constraints Developments Related to the Speed Profile Computation

This appendix presents the mathematical developments related to the 16 distinct physical constraints that we consider for our main application (the Eurobot contest). Those developments are used with the method presented in Chapter 4.2 in order to compute a time-optimal speed profile (i.e., the speed profile that reaches as quickly as possible the destination) for a given path segment representation (m, σ).

A.1 Forward Steps

Let us consider the case of a path segment (m, σ) entirely composed of forward steps, i.e., such that $\nu_i = F$ for all $i \in [0, m-1]$.

From Section 4.2, we have defined the velocity z_i for all $i \in [0, m]$

$$z_i = \sqrt{\frac{v_{L_i}^2 + v_{R_i}^2}{2}}. \tag{A.1}$$

A.1.1 Local State Constraints

The first group of constraints contains those that translate into an upper bound z_{max_i} on the velocity at a path index i (i.e., local constraints that are not influenced by the velocity at previous or future states). In this section, we address the computation of z_{max_i} (for all $i \in [0, m]$) from the parameters v_0, v_{fm}, v_{LRmin}, v_{LRmax}, v_{Smax}, a_{Cmin}, a_{Cmax}, $\dot{\theta}_{Smax}$, and the functions v_{max} and $\dot{\theta}_{max}$ (all these parameters and functions are defined in Section 4.1.5).

Recall that forward path segments are such that

$$\tilde{\kappa}_0 = \tilde{\kappa}_m = 0.$$

We therefore have

$$\begin{aligned} z_{max_0} &= z_{init} = v_0 \\ z_{max_m} &= \min(v_{fm}, v_{LRmax}, v_{Smax}, v_{max}(m)). \end{aligned}$$

Let us express the values of v_{L_i} and v_{R_i} in terms of z_i. From Equations (3.16, 3.17), for each $i \in [0, m]$, we have

$$v_{L_i}\left(1 + \frac{e\tilde{\kappa}_i}{2}\right) = v_{R_i}\left(1 - \frac{e\tilde{\kappa}_i}{2}\right).$$

where e denotes the axle width of the robot (i.e., the distance between the two wheel contact patches).

By defining

$$\omega_i = \arctan\left(1 + \frac{e\tilde{\kappa}_i}{2}, 1 - \frac{e\tilde{\kappa}_i}{2}\right),$$

we get

$$v_{L_i} = z_i\sqrt{2}\cos\omega_i, \tag{A.2}$$

$$v_{R_i} = z_i\sqrt{2}\sin\omega_i. \tag{A.3}$$

Note that, from Equation 2.5 we have

$$v_i = \frac{v_{L_i} + v_{R_i}}{2} \geq 0$$

and thus

$$\sin\omega_i + \cos\omega_i > 0, \tag{A.4}$$

since the path segment is composed of forward steps.

As a consequence, we have

$$-\frac{\pi}{4} < \omega_i < \frac{3\pi}{4}$$

for every $i \in [0, m]$.

■ v_{LRmax} constraint

The constraints

$$v_{L_i} \leq v_{LRmax},$$
$$v_{R_i} \leq v_{LRmax},$$

become

$$z_{max_i}\sqrt{2}\cos\omega_i \leq v_{LRmax},$$
$$z_{max_i}\sqrt{2}\sin\omega_i \leq v_{LRmax}.$$

This gives the upper bound

$$z_{max_i} \leq \frac{v_{LRmax}}{\sqrt{2}\cos\omega_i} \quad \text{if } -\frac{\pi}{4} < \omega_i \leq \frac{\pi}{4},$$
$$z_{max_i} \leq \frac{v_{LRmax}}{\sqrt{2}\sin\omega_i} \quad \text{if } \frac{\pi}{4} < \omega_i < \frac{3\pi}{4}.$$

■ v_{LRmin} constraint

The constraints

$$v_{L_i} \geq v_{LRmin},$$
$$v_{R_i} \geq v_{LRmin},$$

must also be considered if one of the two wheels has a negative speed (i.e., if $\sin\omega_i < 0$ or $\cos\omega_i < 0$, or in other words if $-\frac{\pi}{4} < \omega_i < 0$ or $\frac{\pi}{2} < \omega_i < \frac{3\pi}{4}$).

We get the upper bound

$$z_{max_i} \leq \frac{v_{LRmin}}{\sqrt{2}\sin\omega_i} \quad \text{if } -\frac{\pi}{4} < \omega_i < 0,$$
$$z_{max_i} \leq \frac{v_{LRmin}}{\sqrt{2}\cos\omega_i} \quad \text{if } \frac{\pi}{2} < \omega_i < \frac{3\pi}{4}.$$

■ v_{Smax} constraint

From Equation 3.24, we have for the steering wheel

$$v_{S_i} = v_i\sqrt{1 + (e'\tilde{\kappa}_i)^2}$$

$$\begin{aligned} v_{S_i} &= \frac{v_{L_i} + v_{R_i}}{2} \sqrt{1 + (e'\tilde{\kappa}_i)^2} \\ &= z_i \frac{\sin\omega_i + \cos\omega_i}{\sqrt{2}} \sqrt{1 + (e'\tilde{\kappa}_i)^2}. \end{aligned}$$

where e' denotes the wheelbase of the robot (i.e., the distance between the contact patch of the steering wheel and the rear axle).

The constraint

$$v_{S_i} \leq v_{S\,max}$$

then yields

$$z_{max_i} \leq \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\sqrt{1 + (e'\tilde{\kappa}_i)^2}} v_{S\,max},$$

since $\sin\omega_i + \cos\omega_i > 0$ (cf. Equation A.4).

Note that, the constraint $v_{S_i} \geq v_{S\,min}$ naturally always holds, since the path segment is composed of forward steps ($v_{S_i} \geq 0$).

■ **$v_{max}(i)$ constraint**

Next, we have for the speed v_i of the reference point of the robot

$$\begin{aligned} v_i &= \frac{v_{L_i} + v_{R_i}}{2} \\ &= z_i \frac{\sin\omega_i + \cos\omega_i}{\sqrt{2}}. \end{aligned}$$

The constraint

$$v_i \leq v_{max}(i)$$

then gives

$$z_{max_i} \leq \frac{\sqrt{2}}{\sin\omega_i + \cos\omega_i} v_{max}(i).$$

Note that, the constraint $v_i \geq v_{min}(i)$ naturally always holds, since the path segment is composed of forward steps ($v_i \geq 0$).

■ $\dot{\theta}_{max}(i)$ constraint

The angular speed $\dot{\theta}_i$ of the robot is given, from Equation 3.7, by

$$\begin{aligned}\dot{\theta}_i &= \tilde{\kappa}_i\, v_i\\ &= z_i\, \frac{(\sin\omega_i + \cos\omega_i)\, \tilde{\kappa}_i}{\sqrt{2}}.\end{aligned}$$

If $\tilde{\kappa}_i \neq 0$, the constraint

$$|\dot{\theta}_i| \leq \dot{\theta}_{max}$$

then becomes

$$z_{max_i} \leq \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\, |\tilde{\kappa}_i|}\, \dot{\theta}_{max}(i).$$

If $\tilde{\kappa}_i = 0$, the constraint becomes

$$0 \leq \dot{\theta}_{max},$$

which naturally always holds.

■ $a_{C\,min}$ and $a_{C\,max}$ constraints

Let us now address the constraints on the radial acceleration of the reference point of the robot.

We have, from Equation 3.13, for every $i \in [1, m-1]$

$$\begin{aligned}a_{C_i} &= -\tilde{\kappa}_i\, v_i^2\\ &= -\frac{\tilde{\kappa}_i\, (v_{L_i} + v_{R_i})^2}{4}\\ &= -\frac{\tilde{\kappa}_i\, (\sin\omega_i + \cos\omega_i)^2}{2}\, z_i^2.\end{aligned}$$

From the constraints

$$a_{C\,min} \leq a_{C_i} \leq a_{C\,max},$$

we get

$$z_{max_i} \leq \frac{\sqrt{\dfrac{-2a_{Cmax}}{\tilde{\kappa}_i}}}{\sin\omega_i + \cos\omega_i} \quad \text{if } \tilde{\kappa}_i < 0,$$

$$z_{max_i} \leq \frac{\sqrt{\dfrac{-2a_{Cmin}}{\tilde{\kappa}_i}}}{\sin\omega_i + \cos\omega_i} \quad \text{if } \tilde{\kappa}_i > 0.$$

Note that, if $\tilde{\kappa}_i = 0$, the constraints become

$$a_{Cmin} \leq 0 \leq a_{Cmax},$$

which naturally always holds.

■ $\dot{\theta}_{S max}$ **constraint**

Finally, the constraint on the rate of variation of the steering angle also leads to an upper bound on the value of z_{max_i}. Indeed, for every $i \in [1, m-1]$, we can estimate the instantaneous rate of variation of the steering angle in function of the travelled distance.

First the derivative of the instantaneous curvature can be estimated in the same way that we have estimated the instantaneous curvature (see Equation 4.1.1), assuming that the curvature change linearly with the travelled distance between two steps.

Thus, we can estimate the derivative of the instantaneous curvature $\dfrac{d\tilde{\kappa}_i}{ds}$ by

$$\frac{d\tilde{\kappa}_i}{ds} = \begin{cases} 0 & \text{if } \kappa_{i-1} = 0 \text{ and } \kappa_i = 0 \\ \dfrac{2\kappa_i}{|s_i|} & \text{if } \kappa_{i-1} = 0 \text{ and } \kappa_i \neq 0 \\ \dfrac{-2\kappa_{i-1}}{|s_{i-1}|} & \text{if } \kappa_{i-1} \neq 0 \text{ and } \kappa_i = 0 \\ \dfrac{2(\kappa_i - \kappa_{i-1})}{|s_i| + |s_{i-1}|} & \text{otherwise} \end{cases}$$

for every $i \in [1, m-1]$.

Note that, since $\kappa_0 = \kappa_{m-1} = \kappa_m = 0$, by hypothesis, we also have

$$\frac{d\tilde{\kappa}_0}{ds} = \frac{d\tilde{\kappa}_m}{ds} = 0.$$

Recall that, in the case of a tricycle platform, the orientation θ_{S_i} of the steering wheel is given (from Equation 3.5) by

$$\theta_{S_i} = \arctan(e'\tilde{\kappa}_i)$$

where e' denotes the wheelbase of the robot.

The derivative of the instantaneous rate of variation of the steering angle $\frac{d\theta_{S_i}}{ds}$ is then

$$\frac{d\theta_{S_i}}{ds} = \frac{d(\arctan(e'\tilde{\kappa}_i))}{ds} = \frac{e'\frac{d\tilde{\kappa}_i}{ds}}{e'^2\,\tilde{\kappa}_i^2 + 1}. \tag{A.5}$$

The constraint

$$\left|\dot{\theta}_{S_i}\right| = \left|\frac{d\theta_{S_i}}{ds}\right| \cdot v_i \leq \dot{\theta}_{Smax}$$

then yields

$$z_{max_i} \leq \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\left|\frac{d\theta_{S_i}}{ds}\right|}\,\dot{\theta}_{Smax}$$

if $\frac{d\theta_{S_i}}{ds} \neq 0$ (i.e., if $\kappa_{i-1} \neq \kappa_i$), for every $i \in [1, m-1]$.

Note that, if $\frac{d\theta_{S_i}}{ds} = 0$, the constraint becomes

$$0 \leq \dot{\theta}_{Smax},$$

which naturally always holds.

Summary

In summary, we obtain

$$\begin{aligned} z_{max_0} &= z_{init} = v_0 \\ z_{max_m} &= \min(v_{fm}, v_{LRmax}, v_{Smax}, v_{max}(m)), \end{aligned}$$

and for each $i \in [1, m-1]$

$$z_{max_i} = \min\left(\frac{v_{LRmax}}{\sqrt{2}\max(\sin\omega_i, \cos\omega_i)}, \frac{v_{LRmin}}{\sqrt{2}\min(\sin\omega_i, \cos\omega_i)}, \right.$$

$$\frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\sqrt{1 + (e'\tilde{\kappa}_i)^2}} v_{Smax},$$

$$\frac{\sqrt{2}}{\sin\omega_i + \cos\omega_i} v_{max}(i), \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\,|\tilde{\kappa}_i|} \dot{\theta}_{max}(i),$$

$$\left. \frac{\sqrt{\frac{-2a_{Cm}}{\tilde{\kappa}_i}}}{\sin\omega_i + \cos\omega_i}, \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\left|\frac{d\theta_{S_i}}{ds}\right|} \dot{\theta}_{Smax} \right),$$

where

- the second term is only considered if $\sin\omega_i < 0$ or $\cos\omega_i < 0$;
- the fifth and sixth terms (involving $\dot{\theta}_{max}(i)$ and a_{Cm}) are only considered if $\tilde{\kappa}_i \neq 0$;
- the seventh term (involving $\dot{\theta}_{Smax}$) is only considered if $\kappa_{i-1} \neq \kappa_i$;
- the constant a_{Cm} is defined as follows:

$$a_{Cm} = \begin{cases} a_{Cmax} & \text{if } \tilde{\kappa}_i < 0, \\ a_{Cmin} & \text{if } \tilde{\kappa}_i > 0. \end{cases}$$

A.1.2 Transition Constraints

This second group of constraints contains those that jointly involve the velocity at successive path indices (e.g., constraints that involve the velocities z_i and z_{i+1} at two successive indices i and $i+1$ of the path). This group mainly includes constraints expressed in terms of acceleration.

A.1.2.1 Forward Stage

Due to the transition constraints, the value of z_i at the beginning of the $(i+1)$-th step determines an upper bound on the value of z_{i+1} after the step:

$$z_{i+1} \leq z_{maxF_i}(z_i),$$

where $z_{maxF_i} : [0,\ z_{max_i}] \to \mathbb{R}_{\geq 0}$, for all $i \in [0, m-1]$.

In this section, we address the computation, for each $i \in [0, m-1]$, of the function $z_{maxF_i}(z)$ in terms of the parameters $a_{T\,min}$, $a_{T\,max}$, and the functions a_{LRmin}, a_{LRmax} $a_{S\,min}$ and $a_{S\,max}$ (all these parameters and functions are defined in Section 4.1.5).

During the $(i+1)$-th step, the reference point of the robot travels the distance $s_i > 0$. The speed of this reference point at $\sigma(i)$ and $\sigma(i+1)$ is respectively given by

$$v_i = \frac{v_{L_i} + v_{R_i}}{2} = \frac{\sin\omega_i + \cos\omega_i}{\sqrt{2}}\, z_i$$

and

$$v_{i+1} = \frac{\sin\omega_{i+1} + \cos\omega_{i+1}}{\sqrt{2}}\, z_{i+1}.$$

The amount of time Δt_i needed for performing this step is given by

$$\Delta t_i = \frac{2s_i}{v_i + v_{i+1}} = \frac{2\sqrt{2}\, s_i}{(\sin\omega_i + \cos\omega_i)\, z_i + (\sin\omega_{i+1} + \cos\omega_{i+1})\, z_{i+1}}. \tag{A.6}$$

■ $a_{T\,min}$ and $a_{T\,max}$ constraints

The tangential acceleration a_{T_i} of the reference point during the $(i+1)$-th step can thus be estimated, from Equation 3.12, as

$$\begin{aligned} a_{T_i} &= \frac{v_{i+1} - v_i}{\Delta t_i} = \frac{v_{i+1}^2 - v_i^2}{2s_i} \\ &= \frac{(\sin\omega_{i+1} + \cos\omega_{i+1})^2\, z_{i+1}^2 - (\sin\omega_i + \cos\omega_i)^2\, z_i^2}{4s_i}. \end{aligned}$$

assuming constant-acceleration motion during the step.

The constraints

$$a_{T\,min} \le a_{T_i} \le a_{T\,max}$$

then become

$$4a_{T\,min}\, s_i \le (\sin\omega_{i+1} + \cos\omega_{i+1})^2\, z_{i+1}^2 - (\sin\omega_i + \cos\omega_i)^2\, z_i^2 \le 4a_{T\,max}\, s_i.$$

Since we have $\sin\omega_i + \cos\omega_i > 0$ and $\sin\omega_{i+1} + \cos\omega_{i+1} > 0$, by hypothesis (since the path segment is composed of forward steps), we can write

$$\frac{(\sin\omega_i + \cos\omega_i)^2 z_i^2 + 4a_{T\,min}\, s_i}{(\sin\omega_{i+1} + \cos\omega_{i+1})^2} \leq z_{i+1}^2 \leq \frac{(\sin\omega_i + \cos\omega_i)^2 z_i^2 + 4a_{T\,max}\, s_i}{(\sin\omega_{i+1} + \cos\omega_{i+1})^2},$$

which yields

$$\begin{cases} z_{maxF_i}(z) \geq \begin{cases} \dfrac{\sqrt{(\sin\omega_i + \cos\omega_i)^2 z^2 + 4a_{T\,min}\, s_i}}{\sin\omega_{i+1} + \cos\omega_{i+1}} & \text{if } (\sin\omega_i + \cos\omega_i)^2 z^2 + 4a_{T\,min}\, s_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ z_{maxF_i}(z) \leq \dfrac{\sqrt{(\sin\omega_i + \cos\omega_i)^2 z^2 + 4a_{T\,max}\, s_i}}{\sin\omega_{i+1} + \cos\omega_{i+1}} \end{cases}$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

■ $a_{LRmin}(v_{L_i})$, $a_{LRmax}(v_{L_i})$, $a_{LRmin}(v_{R_i})$ and $a_{LRmax}(v_{R_i})$ constraints

We proceed in a similar way for the acceleration constraints on the left and right wheels.

The speed of those wheels at σ_i and σ_{i+1} are respectively given by

$$\begin{aligned} v_{L_i} &= z_i\sqrt{2}\cos\omega_i, \\ v_{R_i} &= z_i\sqrt{2}\sin\omega_i, \end{aligned}$$

and

$$\begin{aligned} v_{L_{i+1}} &= z_{i+1}\sqrt{2}\cos\omega_{i+1}, \\ v_{R_{i+1}} &= z_{i+1}\sqrt{2}\sin\omega_{i+1}. \end{aligned}$$

The tangential accelerations a_{L_i} and a_{R_i} during the $(i+1)$-th step can be estimated as

$$\begin{aligned} a_{L_i} &= \frac{v_{L_{i+1}} - v_{L_i}}{\Delta t_i} \\ &= \frac{(\cos\omega_{i+1}\, z_{i+1} - \cos\omega_i\, z_i)\Big((\sin\omega_i + \cos\omega_i)\, z_i + (\sin\omega_{i+1} + \cos\omega_{i+1})\, z_{i+1}\Big)}{2s_i} \end{aligned}$$

$$\begin{aligned} a_{R_i} &= \frac{v_{R_{i+1}} - v_{R_i}}{\Delta t_i} \\ &= \frac{(\sin\omega_{i+1}\, z_{i+1} - \sin\omega_i\, z_i)\Big((\sin\omega_i + \cos\omega_i)\, z_i + (\sin\omega_{i+1} + \cos\omega_{i+1})\, z_{i+1}\Big)}{2s_i} \end{aligned}$$

assuming constant-acceleration motion during the step.

For the left wheel, the constraints

$$a_{LRmin}(v_{L_i}) \leq a_{L_i} \leq a_{LRmax}(v_{L_i})$$

become

$$\begin{aligned} 2a_{LRmin}(z_i\sqrt{2}\cos\omega_i)\, s_i \leq \cos\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\, z_{i+1}^2 + \\ (\sin\omega_i \cos\omega_{i+1} - \sin\omega_{i+1}\cos\omega_i)\, z_i\, z_{i+1} - \\ \cos\omega_i\,(\sin\omega_i + \cos\omega_i)\, z_i^2 \leq 2a_{LRmax}(z_i\sqrt{2}\cos\omega_i)\, s_i, \end{aligned}$$

and simplify into

$$\begin{aligned} 2a_{LRmin}(z_i\sqrt{2}\cos\omega_i)\, s_i \leq \cos\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\, z_{i+1}^2 + \\ \sin(\omega_i - \omega_{i+1})\, z_i\, z_{i+1} - \cos\omega_i\,(\sin\omega_i + \cos\omega_i)\, z_i^2 \leq 2a_{LRmax}(z_i\sqrt{2}\cos\omega_i)\, s_i, \end{aligned}$$

which is equivalent to

$$\begin{cases} -\cos\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\, z_{i+1}^2 - \sin(\omega_i - \omega_{i+1})\, z_i\, z_{i+1} + \\ \qquad \cos\omega_i\,(\sin\omega_i + \cos\omega_i)\, z_i^2 + 2a_{LRmin}(z_i\sqrt{2}\cos\omega_i)\, s_i \leq 0 \\ \\ \cos\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\, z_{i+1}^2 + \sin(\omega_i - \omega_{i+1})\, z_i\, z_{i+1} - \\ \qquad \cos\omega_i\,(\sin\omega_i + \cos\omega_i)\, z_i^2 - 2a_{LRmax}(z_i\sqrt{2}\cos\omega_i)\, s_i \leq 0. \end{cases}$$

Each of these two inequations determines a set of possible values for z_{i+1}. Let $S_{Lmin}(z_i)$ and $S_{Lmax}(z_i)$ denote these two sets (respectively resulting from the first and the second inequation), we then have

$$z_{i+1} \in S_{Lmin}(z_i) \cap S_{Lmax}(z_i),$$

which yields

$$z_{maxF_i}(z) \in S_{Lmin}(z) \cap S_{Lmax}(z)$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Each of these two inequations is of the form $a\,z_{i+1}^2 + b\,z_{i+1} + c \leq 0$, the two sets $S_{Lmin}(z_i)$ and $S_{Lmax}(z_i)$ can then be determined in the following way:

- If $a = 0$:

  We thus have $\cos\omega_{i+1} = 0$, i.e., $\omega_{i+1} = \frac{\pi}{2}$ and $v_{L_{i+1}} = 0$.

  - If $b = 0$:

    We thus have $\omega_i = \omega_{i+1} = \frac{\pi}{2}$ or $z_i = 0$. In both cases this implies

    $$c = 2a_{LRmin}(z_i\sqrt{2}\cos\omega_i)\,s_i \leq 0$$

    for the first inequation, and

    $$c = -2a_{LRmax}(z_i\sqrt{2}\cos\omega_i)\,s_i \leq 0$$

    for the second. We obtain

    $$z_{i+1} \in [0,\ +\infty]\,.$$

  - If $b > 0$:

    Then the constraint becomes

    $$z_{i+1} \leq -\frac{c}{b}.$$

    We obtain

    $$z_{i+1} \in \begin{cases} \left[0,\ -\dfrac{c}{b}\right] & \text{if } -\dfrac{c}{b} \geq 0 \\ \emptyset & \text{otherwise.} \end{cases}$$

  - If $b < 0$:

    Then the constraint becomes

    $$z_{i+1} \geq -\frac{c}{b}.$$

    We obtain

    $$z_{i+1} \in \begin{cases} \left[-\dfrac{c}{b},\ +\infty\right] & \text{if } -\dfrac{c}{b} \geq 0 \\ [0,\ +\infty] & \text{otherwise.} \end{cases}$$

- If $a > 0$: Let $\Delta = b^2 - 4ac$.
  - If $\Delta < 0$: Then
  $$z_{i+1} \in \emptyset,$$
  since $a\, z_{i+1}^2 + b\, z_{i+1} + c$ is always positive.
  - If $\Delta \geq 0$: Then
  $$z_{i+1} \in \begin{cases} \emptyset & \text{if } \dfrac{-b + \sqrt{\Delta}}{2a} < 0 \\ \left[\dfrac{-b - \sqrt{\Delta}}{2a}, \dfrac{-b + \sqrt{\Delta}}{2a}\right] & \text{if } \dfrac{-b - \sqrt{\Delta}}{2a} \geq 0 \\ \left[0, \dfrac{-b + \sqrt{\Delta}}{2a}\right] & \text{otherwise.} \end{cases}$$
- If $a < 0$: Let $\Delta = b^2 - 4ac$.
  - If $\Delta < 0$: Then
  $$z_{i+1} \in [0,\ +\infty]\,,$$
  since $a\, z_{i+1}^2 + b\, z_{i+1} + c$ is always negative.
  - If $\Delta \geq 0$: Then
  $$z_{i+1} \in \begin{cases} [0,\ +\infty] & \text{if } \dfrac{-b - \sqrt{\Delta}}{2a} < 0 \\ \left[0, \dfrac{-b + \sqrt{\Delta}}{2a}\right] \cup \left[\dfrac{-b - \sqrt{\Delta}}{2a}, +\infty\right] & \text{if } \dfrac{-b + \sqrt{\Delta}}{2a} \geq 0 \\ \left[\dfrac{-b - \sqrt{\Delta}}{2a}, +\infty\right] & \text{otherwise.} \end{cases}$$

Similarly, for the right wheel, the constraints

$$a_{LRmin}(v_{R_i}) \leq a_{R_i} \leq a_{LRmax}(v_{R_i})$$

become

$$2a_{LRmin}(z_i \sqrt{2} \sin \omega_i)\, s_i \leq \sin \omega_{i+1} \,(\sin \omega_{i+1} + \cos \omega_{i+1})\, z_{i+1}^2 + \\ \sin(\omega_{i+1} - \omega_i)\, z_i\, z_{i+1} - \sin \omega_i \,(\sin \omega_i + \cos \omega_i)\, z_i^2 \leq 2a_{LRmax}(z_i \sqrt{2} \sin \omega_i)\, s_i,$$

which is equivalent to

$$\begin{cases} -\sin\omega_{i+1}\,(\sin\omega_{i+1}+\cos\omega_{i+1})\,z_{i+1}^2 - \sin(\omega_{i+1}-\omega_i)\,z_i\,z_{i+1} + \\ \qquad \sin\omega_i\,(\sin\omega_i+\cos\omega_i)\,z_i^2 + 2a_{LRmin}(z_i\sqrt{2}\sin\omega_i)\,s_i \leq 0 \\ \\ \sin\omega_{i+1}\,(\sin\omega_{i+1}+\cos\omega_{i+1})\,z_{i+1}^2 + \sin(\omega_{i+1}-\omega_i)\,z_i\,z_{i+1} - \\ \qquad \sin\omega_i\,(\sin\omega_i+\cos\omega_i)\,z_i^2 - 2a_{LRmax}(z_i\sqrt{2}\sin\omega_i)\,s_i \leq 0. \end{cases}$$

Each of these two inequations determines a set of possible values for z_{i+1}. Let $S_{Rmin}(z_i)$ and $S_{Rmax}(z_i)$ denote these two sets (respectively resulting from the first and the second inequation), we then have

$$z_{i+1} \in S_{Rmin}(z_i) \cap S_{Rmax}(z_i),$$

which yields

$$z_{maxF_i}(z) \in S_{Rmin}(z) \cap S_{Rmax}(z)$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Each of these two inequations is also of the form $a\,z_{i+1}^2 + b\,z_{i+1} + c \leq 0$, and thus can be solved as for the left wheel case.

Note that, if $\omega_i = \omega_{i+1} = \frac{\pi}{4}$ (i.e., if $\tilde{\kappa}_i = \tilde{\kappa}_{i+1} = 0$), the robot follows a straight line during the $(i+1)$-th step, and, in order to speed up the computation time, we can only address the constraints for the left wheel, as the set of solutions for the right wheel will be identical to the one provided by the left wheel.

■ $a_{S\,min}(v_{S_i})$ and $a_{S\,max}(v_{S_i})$ constraints

Finally, we address the constraints on the tangential acceleration of the steering wheel.

The speed of this wheel at σ_i and σ_{i+1} is respectively given, from Equation 3.24, by

$$\begin{aligned} v_{S_i} &= \frac{v_i}{\cos\theta_{S_i}} \\ &= \frac{\sin\omega_i + \cos\omega_i}{\sqrt{2}\cos\theta_{S_i}}\,z_i \\ v_{S_{i+1}} &= \frac{\sin\omega_{i+1} + \cos\omega_{i+1}}{\sqrt{2}\cos\theta_{S_{i+1}}}\,z_{i+1}. \end{aligned}$$

The tangential acceleration a_{S_i} during the $(i+1)$-th step can be estimated as

$$\begin{aligned}
a_{S_i} &= \frac{v_{S_{i+1}} - v_{S_i}}{\Delta t_i} \\
&= \frac{\left(\dfrac{sc_{i+1}}{\sqrt{2}\cos\theta_{S_{i+1}}}\, z_{i+1} - \dfrac{sc_i}{\sqrt{2}\cos\theta_{S_i}}\, z_i\right)\left(sc_i\, z_i + sc_{i+1}\, z_{i+1}\right)}{2\sqrt{2}\, s_i} \\
&= \frac{\left(\dfrac{sc_{i+1}\, z_{i+1}}{\cos\theta_{S_{i+1}}} - \dfrac{sc_i\, z_i}{\cos\theta_{S_i}}\right)\left(sc_i\, z_i + sc_{i+1}\, z_{i+1}\right)}{4 s_i},
\end{aligned}$$

where

$$sc_i = \sin\omega_i + \cos\omega_i,$$

$$sc_{i+1} = \sin\omega_{i+1} + \cos\omega_{i+1},$$

assuming constant-acceleration motion during the step.

The constraints

$$a_{S\,min}(v_{S_i}) \le a_{S_i} \le a_{S\,max}(v_{S_i})$$

become

$$\begin{aligned}
4a_{S\,min}(v_{S_i})\, s_i \le \frac{sc_{i+1}^2}{\cos\theta_{S_{i+1}}}\, z_{i+1}^2 + \\
\left(\frac{1}{\cos\theta_{S_{i+1}}} - \frac{1}{\cos\theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} - \frac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 \le 4a_{S\,max}(v_{S_i})\, s_i
\end{aligned}$$

which is equivalent to

$$\begin{cases}
-\dfrac{sc_{i+1}^2}{\cos\theta_{S_{i+1}}}\, z_{i+1}^2 - \left(\dfrac{1}{\cos\theta_{S_{i+1}}} - \dfrac{1}{\cos\theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} + \dfrac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 + 4a_{S\,min}(v_{S_i})\, s_i \le 0 \\[2ex]
\dfrac{sc_{i+1}^2}{\cos\theta_{S_{i+1}}}\, z_{i+1}^2 + \left(\dfrac{1}{\cos\theta_{S_{i+1}}} - \dfrac{1}{\cos\theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} - \dfrac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 - 4a_{S\,max}(v_{S_i})\, s_i \le 0.
\end{cases}$$

Each of these two inequations determines a set of possible values for z_{i+1}. Let $S_{S\,min}(z_i)$ and $S_{S\,max}(z_i)$ denote these two sets (respectively from the first and the second inequation), we then have

$$z_{i+1} \in S_{S\,min}(z_i) \cap S_{S\,max}(z_i),$$

which yields

$$z_{maxF_i}(z) \in S_{S\,min}(z) \cap S_{S\,max}(z)$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Each of these two inequations is also of the form $a\, z_{i+1}^2 + b\, z_{i+1} + c \leq 0$, and thus can be solved as for the left wheel case.

Summary

In summary, we obtain for the function z_{maxF_i}:

$$\begin{aligned} z_{maxF_i}(z) = \max\Big(& [z_{T\,min}(z),\ z_{T\,max}(z)] \cap \\ & S_{L\,min}(z) \ \cap\ S_{L\,max}(z) \cap \\ & S_{R\,min}(z) \ \cap\ S_{R\,max}(z) \cap \\ & S_{S\,min}(z) \ \cap\ S_{S\,max}(z)\Big), \end{aligned}$$

for all $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$, where

- sc_i and sc_{i+1} are defined as follows:

$$sc_i = \sin\omega_i + \cos\omega_i,$$

$$sc_{i+1} = \sin\omega_{i+1} + \cos\omega_{i+1};$$

- $z_{T\,min}(z)$ and $z_{T\,max}(z)$ are defined as follows:

$$z_{T\,min}(z) = \begin{cases} \dfrac{\sqrt{sc_i^2\, z^2 + 4a_{T\,min}\, s_i}}{sc_{i+1}} & \text{if } sc_i^2\, z^2 + 4a_{T\,min}\, s_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$z_{T\,max}(z) = \frac{\sqrt{sc_i^2\, z^2 + 4a_{T\,max}\, s_i}}{sc_{i+1}};$$

- $S_{Lmin}(z)$, $S_{Lmax}(z)$, $S_{Rmin}(z)$, $S_{Rmax}(z)$, $S_{Smin}(z)$ and $S_{Smax}(z)$ are respectively the sets of solutions for z_{i+1} of the following six quadratic inequations:

$$-\cos\omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 - \sin(\omega_i - \omega_{i+1})\, z\, z_{i+1} + \cos\omega_i\, sc_i\, z^2 + 2a_{LRmin}(v_{L_i})\, s_i \leq 0$$

$$\cos\omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 + \sin(\omega_i - \omega_{i+1})\, z\, z_{i+1} - \cos\omega_i\, sc_i\, z^2 - 2a_{LRmax}(v_{L_i})\, s_i \leq 0$$

$$-\sin\omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 - \sin(\omega_{i+1} - \omega_i)\, z\, z_{i+1} + \sin\omega_i\, sc_i\, z^2 + 2a_{LRmin}(v_{R_i})\, s_i \leq 0$$

$$\sin\omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 + \sin(\omega_{i+1} - \omega_i)\, z\, z_{i+1} - \sin\omega_i\, sc_i\, z^2 - 2a_{LRmax}(v_{R_i})\, s_i \leq 0$$

$$-\frac{sc_{i+1}^2}{\cos\theta_{S_{i+1}}}\, z_{i+1}^2 - \left(\frac{1}{\cos\theta_{S_{i+1}}} - \frac{1}{\cos\theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} + \frac{sc_i^2}{\cos\theta_{S_i}}\, z^2 + 4a_{Smin}(v_{S_i})\, s_i \leq 0$$

$$\frac{sc_{i+1}^2}{\cos\theta_{S_{i+1}}}\, z_{i+1}^2 + \left(\frac{1}{\cos\theta_{S_{i+1}}} - \frac{1}{\cos\theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} - \frac{sc_i^2}{\cos\theta_{S_i}}\, z^2 - 4a_{Smax}(v_{S_i})\, s_i \leq 0,$$

  where

$$v_{L_i} = z_i \sqrt{2}\cos\omega_i,$$

$$v_{R_i} = z_i \sqrt{2}\sin\omega_i,$$

$$v_{S_i} = \frac{\sin\omega_i + \cos\omega_i}{\sqrt{2}\cos\theta_{S_i}}\, z_i,$$

  and where the third and fourth inequations are only considered if $\tilde{\kappa}_i \neq 0$ or $\tilde{\kappa}_{i+1} \neq 0$.

Note that, if the transition constraints are not satisfiable (i.e., if the intersection of all the sets is empty), z has to be lowered into the largest value that makes the constraints satisfiable (cf. Algorithm 4.2 in Chapter 4.2).

A.1.2.2 Backward Stage

Similarly, due to the same transition constraints as in the forward stage, the value of z_i at the beginning of the $(i+1)$-th step determines an upper bound on the value of z_{i-1} at the beginning of the previous step:

$$z_{i-1} \leq z_{maxP_i}(z_i),$$

where $z_{maxP_i} : [0,\, z_{max_i}] \to \mathbb{R}_{\geq 0}$, for all $i \in [1, m]$.

In this section, we address the computation, for each $i \in [1, m]$, of the function $z_{maxP_i}(z)$ in terms of the parameters $a_{T\,min}$, $a_{T\,max}$, and the functions a_{LRmin}, a_{LRmax} $a_{S\,min}$ and $a_{S\,max}$ (all these parameters and functions are defined in Section 4.1.5).

As explained in Section 4.2.5, constraints expressed as functions may depend on z_{i-1}, and in order to ensure the satisfiability of the constraints during the backward stage, we will use the following approximation of the velocity:

$$\tilde{z}_{i-1} = \min(\check{z}_{i-1}, z_i),$$

where $\check{z}_{i-1}$ is the tentative value for the velocity at the beginning of the i-th step obtained after the forward stage.

During the i-th step, the reference point of the robot travels the distance $s_{i-1} > 0$. The amount of time Δt_{i-1} needed for performing this step is given by

$$\Delta t_{i-1} = \frac{2s_{i-1}}{v_{i-1} + v_i} = \frac{2\sqrt{2}\, s_{i-1}}{(\sin\omega_{i-1} + \cos\omega_{i-1})\, z_{i-1} + (\sin\omega_i + \cos\omega_i)\, z_i}.$$

■ $a_{T\,min}$ and $a_{T\,max}$ constraints

Similarly to the forward stage, the tangential acceleration $a_{T_{i-1}}$ of the reference point during the i-th step can be estimated, as

$$\begin{aligned} a_{T_{i-1}} &= \frac{v_i - v_{i-1}}{\Delta t_{i-1}} = \frac{v_i^2 - v_{i-1}^2}{2s_{i-1}} \\ &= \frac{(\sin\omega_i + \cos\omega_i)^2 z_i^2 - (\sin\omega_{i-1} + \cos\omega_{i-1})^2 z_{i-1}^2}{4s_{i-1}}. \end{aligned}$$

assuming constant-acceleration motion during the step.

The constraints

$$a_{T\,min} \le a_{T_{i-1}} \le a_{T\,max}$$

then become

$$4a_{T\,min}\, s_{i-1} \le (\sin\omega_i + \cos\omega_i)^2 z_i^2 - (\sin\omega_{i-1} + \cos\omega_{i-1})^2 z_{i-1}^2 \le 4a_{T\,max}\, s_{i-1}.$$

Since we also have $\sin\omega_i + \cos\omega_i > 0$ and $\sin\omega_{i-1} + \cos\omega_{i-1} > 0$, we can write

$$\frac{(\sin\omega_i + \cos\omega_i)^2 z_i^2 - 4a_{T\,max}\, s_{i-1}}{(\sin\omega_{i-1} + \cos\omega_{i-1})^2} \le z_{i-1}^2 \le \frac{(\sin\omega_i + \cos\omega_i)^2 z_i^2 - 4a_{T\,min}\, s_{i-1}}{(\sin\omega_{i-1} + \cos\omega_{i-1})^2},$$

which yields

$$\begin{cases} z_{maxP_i}(z) \geq \begin{cases} \dfrac{\sqrt{(\sin\omega_i + \cos\omega_i)^2 z^2 - 4a_{T\,max}\, s_{i-1}}}{\sin\omega_{i-1} + \cos\omega_{i-1}} & \text{if } (\sin\omega_i + \cos\omega_i)^2 z^2 - 4a_{T\,max}\, s_{i-1} \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ z_{maxP_i}(z) \leq \dfrac{\sqrt{(\sin\omega_i + \cos\omega_i)^2 z^2 - 4a_{T\,min}\, s_{i-1}}}{\sin\omega_{i-1} + \cos\omega_{i-1}} \end{cases}$$

for every $i \in [1, m]$ and $z \in [0, z_{max_i}]$.

■ $a_{LRmin}(\tilde{v}_{L_{i-1}})$, $a_{LRmax}(\tilde{v}_{L_{i-1}})$, $a_{LRmin}(\tilde{v}_{R_{i-1}})$ and $a_{LRmax}(\tilde{v}_{R_{i-1}})$ constraints

We proceed in a similar way for the acceleration constraints on the left and right wheels.

Recall that the speeds that will be used to evaluate the constraints are given by

$$\begin{aligned} \tilde{v}_{L_{i-1}} &= \tilde{z}_{i-1}\sqrt{2}\cos\omega_{i-1}, \\ \tilde{v}_{R_{i-1}} &= \tilde{z}_{i-1}\sqrt{2}\sin\omega_{i-1}. \end{aligned}$$

The tangential accelerations $a_{L_{i-1}}$ and $a_{R_{i-1}}$ during the i-th step can be estimated as

$$\begin{aligned} a_{L_{i-1}} &= \frac{v_{L_i} - v_{L_{i-1}}}{\Delta t_{i-1}} \\ &= \frac{(\cos\omega_i\, z_i - \cos\omega_{i-1}\, z_{i-1})\Big((\sin\omega_{i-1} + \cos\omega_{i-1})\, z_{i-1} + (\sin\omega_i + \cos\omega_i)\, z_i\Big)}{2s_{i-1}} \\ a_{R_{i-1}} &= \frac{v_{R_i} - v_{R_{i-1}}}{\Delta t_{i-1}} \\ &= \frac{(\sin\omega_i\, z_i - \sin\omega_{i-1}\, z_{i-1})\Big((\sin\omega_{i-1} + \cos\omega_{i-1})\, z_{i-1} + (\sin\omega_i + \cos\omega_i)\, z_i\Big)}{2s_{i-1}} \end{aligned}$$

assuming constant-acceleration motion during the step.

For the left wheel, the constraints

$$a_{LRmin}(\tilde{v}_{L_{i-1}}) \leq a_{L_{i-1}} \leq a_{LRmax}(\tilde{v}_{L_{i-1}})$$

become

$$\begin{aligned}2a_{LRmin}(\tilde{v}_{L_{i-1}})\, s_{i-1} \le -\cos\omega_{i-1}\,(\sin\omega_{i-1}+\cos\omega_{i-1})\, z_{i-1}^2 + \\ \sin(\omega_{i-1}-\omega_i)\, z_i\, z_{i-1} + \cos\omega_i\,(\sin\omega_i+\cos\omega_i)\, z_i^2 \le 2a_{LRmax}(\tilde{v}_{L_{i-1}})\, s_{i-1},\end{aligned}$$

which is equivalent to

$$\begin{cases} \cos\omega_{i-1}\,(\sin\omega_{i-1}+\cos\omega_{i-1})\, z_{i-1}^2 - \sin(\omega_{i-1}-\omega_i)\, z_i\, z_{i-1} - \\ \qquad \cos\omega_i\,(\sin\omega_i+\cos\omega_i)\, z_i^2 + 2a_{LRmin}(\tilde{v}_{L_{i-1}})\, s_{i-1} \le 0 \\ \\ -\cos\omega_{i-1}\,(\sin\omega_{i-1}+\cos\omega_{i-1})\, z_{i-1}^2 + \sin(\omega_{i-1}-\omega_i)\, z_i\, z_{i-1} + \\ \qquad \cos\omega_i\,(\sin\omega_i+\cos\omega_i)\, z_i^2 - 2a_{LRmax}(\tilde{v}_{L_{i-1}})\, s_{i-1} \le 0. \end{cases}$$

Each of these two inequations determines a set of possible values for z_{i-1}. Let $SP_{Lmin}(z_i)$ and $SP_{Lmax}(z_i)$ denote these two sets (respectively from the first and the second inequation), we then have

$$z_{i-1} \in SP_{Lmin}(z_i) \cap SP_{Lmax}(z_i),$$

which yields

$$z_{maxP_i}(z) \in SP_{Lmin}(z) \cap SP_{Lmax}(z)$$

for every $i \in [1, m]$ and $z \in [0, z_{max_i}]$.

Each of these two inequations is of the form $a\, z_{i+1}^2 + b\, z_{i+1} + c \le 0$, and thus can be solved as for the inequations in the forward stage.

Similarly, for the right wheel, the constraints

$$a_{LRmin}(\tilde{v}_{R_{i-1}}) \le a_{R_{i-1}} \le a_{LRmax}(\tilde{v}_{R_{i-1}})$$

become

$$\begin{aligned}2a_{LRmin}(\tilde{v}_{R_{i-1}})\, s_{i-1} \le -\sin\omega_{i-1}\,(\sin\omega_{i-1}+\cos\omega_{i-1})\, z_{i-1}^2 + \\ \sin(\omega_i-\omega_{i-1})\, z_i\, z_{i-1} + \sin\omega_i\,(\sin\omega_i+\cos\omega_i)\, z_i^2 \le 2a_{LRmax}(\tilde{v}_{R_{i-1}})\, s_{i-1},\end{aligned}$$

which is equivalent to

$$\begin{cases} \sin\omega_{i-1}\,(\sin\omega_{i-1}+\cos\omega_{i-1})\, z_{i-1}^2 - \sin(\omega_i-\omega_{i-1})\, z_i\, z_{i-1} - \\ \qquad \sin\omega_i\,(\sin\omega_i+\cos\omega_i)\, z_i^2 + 2a_{LRmin}(\tilde{v}_{R_{i-1}})\, s_{i-1} \le 0 \\ \\ -\sin\omega_{i-1}\,(\sin\omega_{i-1}+\cos\omega_{i-1})\, z_{i-1}^2 + \sin(\omega_i-\omega_{i-1})\, z_i\, z_{i-1} + \\ \qquad \sin\omega_i\,(\sin\omega_i+\cos\omega_i)\, z_i^2 - 2a_{LRmax}(\tilde{v}_{R_{i-1}})\, s_{i-1} \le 0. \end{cases}$$

Each of these two inequations determines a set of possible values for z_{i-1}. Let $SP_{Rmin}(z_i)$ and $SP_{Rmax}(z_i)$ denote these two sets (respectively from the first and the second inequation), we then have

$$z_{i-1} \in SP_{Rmin}(z_i) \cap SP_{Rmax}(z_i),$$

which yields

$$z_{maxP_i}(z) \in SP_{Rmin}(z) \cap SP_{Rmax}(z)$$

for every $i \in [1, m]$ and $z \in [0, z_{max_i}]$.

Each of these two inequations is also of the form $a\,z_{i+1}^2 + b\,z_{i+1} + c \leq 0$, and thus can be solved as for the left wheel case.

Note that, as in the forward stage, if $\omega_i = \omega_{i-1} = \frac{\pi}{4}$ (i.e., if $\tilde{\kappa}_i = \tilde{\kappa}_{i-1} = 0$), the robot follows a straight line during the i-th step, and, in order to speed up the computation time, we can only address the constraints for the left wheel, as the set of solutions for the right wheel will be identical to the one provided by the left wheel.

■ $a_{S\,min}(\tilde{v}_{S_{i-1}})$ and $a_{S\,max}(\tilde{v}_{S_{i-1}})$ constraints

Finally, we address the constraints on the tangential acceleration of the steering wheel.

Recall that the speed that will be used to evaluate the constraints is given by

$$\tilde{v}_{S_{i-1}} = \frac{\sin\omega_{i-1} + \cos\omega_{i-1}}{\sqrt{2}\cos\theta_{S_{i-1}}}\,\tilde{z}_{i-1}.$$

The tangential accelerations $a_{S_{i-1}}$ during the i-th step can be estimated as

$$\begin{aligned} a_{S_{i-1}} &= \frac{v_{S_i} - v_{S_{i-1}}}{\Delta t_{i-1}} \\ &= \frac{\left(\dfrac{sc_i\,z_i}{\cos\theta_{S_i}} - \dfrac{sc_{i-1}\,z_{i-1}}{\cos\theta_{S_{i-1}}}\right)(sc_{i-1}\,z_{i-1} + sc_i\,z_i)}{4s_{i-1}}, \end{aligned}$$

where

$$sc_{i-1} = \sin\omega_{i-1} + \cos\omega_{i-1},$$

$$sc_i = \sin\omega_i + \cos\omega_i,$$

assuming constant-acceleration motion during the step.

The constraints

$$a_{S\,min}(\tilde{v}_{S_{i-1}}) \leq a_{S_{i-1}} \leq a_{S\,max}(\tilde{v}_{S_{i-1}})$$

become

$$\begin{aligned} 4a_{S\,min}(\tilde{v}_{S_{i-1}})\, s_{i-1} \leq -\frac{sc_{i-1}^2}{\cos\theta_{S_{i-1}}}\, z_{i-1}^2 + &\\ \left(\frac{1}{\cos\theta_{S_i}} - \frac{1}{\cos\theta_{S_{i-1}}}\right) sc_{i-1}\, sc_i\, z_i\, z_{i-1} + \frac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 &\leq 4a_{S\,max}(\tilde{v}_{S_{i-1}})\, s_{i-1} \end{aligned}$$

which is equivalent to

$$\begin{cases} \dfrac{sc_{i-1}^2}{\cos\theta_{S_{i-1}}}\, z_{i-1}^2 - \left(\dfrac{1}{\cos\theta_{S_i}} - \dfrac{1}{\cos\theta_{S_{i-1}}}\right) sc_{i-1}\, sc_i\, z_i\, z_{i-1} - \\ \qquad\qquad \dfrac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 + 4a_{S\,min}(\tilde{v}_{S_{i-1}})\, s_{i-1} \leq 0 \\ -\dfrac{sc_{i-1}^2}{\cos\theta_{S_{i-1}}}\, z_{i-1}^2 + \left(\dfrac{1}{\cos\theta_{S_i}} - \dfrac{1}{\cos\theta_{S_{i-1}}}\right) sc_{i-1}\, sc_i\, z_i\, z_{i-1} + \\ \qquad\qquad \dfrac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 - 4a_{S\,max}(\tilde{v}_{S_{i-1}})\, s_{i-1} \leq 0. \end{cases}$$

Each of these two inequations determines a set of possible values for z_{i-1}. Let $SP_{S\,min}(z_i)$ and $SP_{S\,max}(z_i)$ denote these two sets (respectively from the first and the second inequation), we then have

$$z_{i-1} \in SP_{S\,min}(z_i) \cap SP_{S\,max}(z_i),$$

which yields

$$z_{maxP_i}(z) \in SP_{S\,min}(z) \cap SP_{S\,max}(z)$$

for every $i \in [1, m]$ and $z \in [0, z_{max_i}]$.

Each of these two inequations is also of the form $a\, z_{i+1}^2 + b\, z_{i+1} + c \leq 0$, and thus can be solved as for the left wheel case.

Summary

In summary, we obtain for the function z_{maxP_i}:

$$\begin{aligned} z_{maxP_i}(z) = \max\Big(& [z_{T\,min}(z),\ z_{T\,max}(z)] \cap SP_{L\,min}(z) \cap SP_{L\,max}(z) \cap \\ & SP_{R\,min}(z) \cap SP_{R\,max}(z) \cap SP_{S\,min}(z) \cap SP_{S\,max}(z)\Big), \end{aligned}$$

for all $i \in [1, m]$ and $z \in [0, z_{max_i}]$,

where

- sc_{i-1} and sc_i are defined as follows:

$$sc_{i-1} = \sin\omega_{i-1} + \cos\omega_{i-1},$$
$$sc_i = \sin\omega_i + \cos\omega_i;$$

- $z_{T\,min}(z)$ and $z_{T\,max}(z)$ are defined as follows:

$$z_{T\,min}(z) = \begin{cases} \dfrac{\sqrt{sc_i^2\, z^2 - 4a_{T\,max}\, s_{i-1}}}{sc_{i-1}} & \text{if } sc_i^2\, z^2 - 4a_{T\,max}\, s_{i-1} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$z_{T\,max}(z) = \frac{\sqrt{sc_i^2\, z^2 - 4a_{T\,min}\, s_{i-1}}}{sc_{i-1}};$$

- $SP_{L\,min}(z)$, $SP_{L\,max}(z)$, $SP_{R\,min}(z)$, $SP_{R\,max}(z)$, $SP_{S\,min}(z)$ and $SP_{S\,max}(z)$ are respectively the sets of solutions for z_{i-1} of the following six quadratic inequations:

$$\cos\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 - \sin(\omega_{i-1} - \omega_i)\, z\, z_{i-1} - \cos\omega_i\, sc_i\, z^2 + 2a_{LRmin}(\tilde{v}_{L_{i-1}})\, s_{i-1} \leq 0$$

$$-\cos\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 + \sin(\omega_{i-1} - \omega_i)\, z\, z_{i-1} + \cos\omega_i\, sc_i\, z^2 - 2a_{LRmax}(\tilde{v}_{L_{i-1}})\, s_{i-1} \leq 0$$

$$\sin\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 - \sin(\omega_i - \omega_{i-1})\, z\, z_{i-1} - \sin\omega_i\, sc_i\, z^2 + 2a_{LRmin}(\tilde{v}_{R_{i-1}})\, s_{i-1} \leq 0$$

$$-\sin\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 + \sin(\omega_i - \omega_{i-1})\, z\, z_{i-1} + \sin\omega_i\, sc_i\, z^2 - 2a_{LRmax}(\tilde{v}_{R_{i-1}})\, s_{i-1} \leq 0$$

$$\frac{sc_{i-1}^2}{\cos\theta_{S_{i-1}}}\, z_{i-1}^2 - \left(\frac{1}{\cos\theta_{S_i}} - \frac{1}{\cos\theta_{S_{i-1}}}\right) sc_{i-1}\, sc_i\, z_i\, z_{i-1} - \frac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 + 4a_{S\,min}(\tilde{v}_{S_{i-1}})\, s_{i-1} \leq 0$$

$$-\frac{sc_{i-1}^2}{\cos\theta_{S_{i-1}}}\, z_{i-1}^2 + \left(\frac{1}{\cos\theta_{S_i}} - \frac{1}{\cos\theta_{S_{i-1}}}\right) sc_{i-1}\, sc_i\, z_i\, z_{i-1} + \frac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 - 4a_{S\,max}(\tilde{v}_{S_{i-1}})\, s_{i-1} \leq 0,$$

where

$$\tilde{v}_{L_{i-1}} = \tilde{z}_{i-1}\sqrt{2}\cos\omega_{i-1},$$

$$\tilde{v}_{R_{i-1}} = \tilde{z}_{i-1}\sqrt{2}\sin\omega_{i-1},$$

$$\tilde{v}_{S_{i-1}} = \frac{\sin\omega_{i-1} + \cos\omega_{i-1}}{\sqrt{2}\cos\theta_{S_{i-1}}}\tilde{z}_{i-1},$$

with

$$\tilde{z}_{i-1} = \min(\check{z}_{i-1}, z_i),$$

where $\check{z}_{i-1}$ is the tentative value for the velocity at the beginning of the i-th step obtained after the forward stage.

As in the forward stage, the third and fourth inequations can only be considered if $\tilde{\kappa}_i \neq 0$ or $\tilde{\kappa}_{i+1} \neq 0$.

Note that, as shown in Section 4.2.5, the transition constraints during the backward stage are always satisfiable (i.e., the intersection of all the sets is not empty).

A.1.3 Wheel Speeds

We finally recall the formulas that express the speeds of the left, right and steering wheels in terms of the values of the variables z_i.

We have for all $i \in [0, m]$:

$$v_{L_i} = z_i\sqrt{2}\cos\omega_i,$$

$$v_{R_i} = z_i\sqrt{2}\sin\omega_i,$$

$$v_{S_i} = z_i\frac{\sin\omega_i + \cos\omega_i}{\sqrt{2}\cos\theta_{S_i}}$$

where

$$\omega_i = \arctan\left(1 + \frac{e\tilde{\kappa}_i}{2}, 1 - \frac{e\tilde{\kappa}_i}{2}\right).$$

A.2 Backward Steps

The case of a path segment (m, σ) entirely composed of backward steps, i.e., such that $v_i = B$ for all $i \in [0, m-1]$, is handled in a similar way as in Section A.1.

Like in the case of forward path segments, the velocity z_i is defined for all $i \in [0, m]$ as

$$z_i = \sqrt{\frac{v_{L_i}^2 + v_{R_i}^2}{2}}.$$

A.2.1 Local State Constraints

In this section, we address the computation of z_{max_i} (for all $i \in [0, m]$) from the parameters v_0, v_{fm}, v_{LRmin}, v_{LRmax}, v_{Smin}, a_{Cmin}, a_{Cmax}, $\dot{\theta}_{Smax}$, and the functions v_{min} and $\dot{\theta}_{max}$ (all these parameters and functions are defined in Section 4.1.5).

Recall that backward path segments are also such that

$$\tilde{\kappa}_0 = \tilde{\kappa}_m = 0.$$

For the boundary conditions, we obtain

$$\begin{aligned} z_{max_0} &= z_{init} \;=\; -v_0 \\ z_{max_m} &= \min(-v_{fm}, -v_{LRmin}, -v_{Smin}, -v_{min}(m)). \end{aligned}$$

The speeds of the left and right wheels can be expressed as

$$\begin{aligned} v_{L_i} &= -z_i\sqrt{2}\cos\omega_i, \\ v_{R_i} &= -z_i\sqrt{2}\sin\omega_i, \end{aligned}$$

where ω_i is still defined as

$$\omega_i = \arctan\left(1 + \frac{e\tilde{\kappa}_i}{2}, 1 - \frac{e\tilde{\kappa}_i}{2}\right).$$

Note that we still have

$$\sin\omega_i + \cos\omega_i > 0$$

and

$$-\frac{\pi}{4} < \omega_i < \frac{3\pi}{4}$$

for every $i \in [0, m]$,

since backward path segments are such that

$$v_i = \frac{v_{L_i} + v_{R_i}}{2} \leq 0$$

for every $i \in [0, m]$.

■ v_{LRmin} constraint

The constraints

$$v_{L_i} \geq v_{LRmin},$$
$$v_{R_i} \geq v_{LRmin},$$

give the upper bound

$$z_{max_i} \leq -\frac{v_{LRmin}}{\sqrt{2}\cos\omega_i} \quad \text{if } -\frac{\pi}{4} < \omega_i \leq \frac{\pi}{4},$$
$$z_{max_i} \leq -\frac{v_{LRmin}}{\sqrt{2}\sin\omega_i} \quad \text{if } \frac{\pi}{4} < \omega_i < \frac{3\pi}{4}.$$

■ v_{LRmax} constraint

The constraints

$$v_{L_i} \leq v_{LRmax},$$
$$v_{R_i} \leq v_{LRmax},$$

must also be considered if one of the two wheels has a positive speed (i.e., if $-\frac{\pi}{4} < \omega_i < 0$ or $\frac{\pi}{2} < \omega_i < \frac{3\pi}{4}$).

We get the upper bound

$$z_{max_i} \leq -\frac{v_{LRmax}}{\sqrt{2}\sin\omega_i} \quad \text{if } -\frac{\pi}{4} < \omega_i < 0,$$
$$z_{max_i} \leq -\frac{v_{LRmax}}{\sqrt{2}\cos\omega_i} \quad \text{if } \frac{\pi}{2} < \omega_i < \frac{3\pi}{4}.$$

■ v_{Smin} constraint

The constraint

$$v_{S_i} \geq v_{Smin}$$

yields

$$z_{max_i} \leq -\frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\sqrt{1 + (e'\tilde{\kappa}_i)^2}} v_{Smin}.$$

Note that, the constraint $v_{S_i} \leq v_{S\,max}$ naturally always holds, since the path segment is composed of backward steps ($v_{S_i} \leq 0$).

■ $v_{min}(i)$ constraint

The constraint

$$v_i \geq v_{min}(i)$$

gives

$$z_{max_i} \leq -\frac{\sqrt{2}}{\sin \omega_i + \cos \omega_i}\, v_{min}(i).$$

Note that, the constraint $v_i \leq v_{max}(i)$ naturally always holds, since the path segment is composed of forward steps ($v_i \geq 0$).

■ $\dot{\theta}_{max}(i)$ constraint

If $\tilde{\kappa}_i \neq 0$, the constraint

$$|\dot{\theta}_i| \leq \dot{\theta}_{max}$$

becomes, just like in the forward case,

$$z_{max_i} \leq \frac{\sqrt{2}}{(\sin \omega_i + \cos \omega_i)\,|\tilde{\kappa}_i|}\, \dot{\theta}_{max}(i).$$

■ $a_{C\,min}$ and $a_{C\,max}$ constraints

For the constraints

$$a_{C\,min} \leq a_{C_i} \leq a_{C\,max},$$

on the radial acceleration of the reference point of the robot, the situation is identical to the case of forward segments and yields

$$z_{max_i} \leq \frac{\sqrt{\dfrac{-2a_{C\,max}}{\tilde{\kappa}_i}}}{\sin \omega_i + \cos \omega_i} \quad \text{if } \tilde{\kappa}_i < 0,$$

$$z_{max_i} \leq \frac{\sqrt{\dfrac{-2a_{C\,min}}{\tilde{\kappa}_i}}}{\sin \omega_i + \cos \omega_i} \quad \text{if } \tilde{\kappa}_i > 0.$$

■ $\dot{\theta}_{S max}$ constraint

Finally, by using the same Equation A.5 for $\dfrac{d\theta_{S_i}}{ds}$, the constraint

$$\left|\dot{\theta}_{S_i}\right| = \left|\frac{d\theta_{S_i}}{ds}\right| \cdot v_i \le \dot{\theta}_{S max}$$

on the rate of variation of the steering angle, like in the forward case, leads to

$$z_{max_i} \le \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\left|\dfrac{d\theta_{S_i}}{ds}\right|}\,\dot{\theta}_{S max}$$

if $\dfrac{d\theta_{S_i}}{ds} \ne 0$ (i.e., if $\kappa_{i-1} \ne \kappa_i$).

Summary

In summary, we obtain

$$\begin{aligned} z_{max_0} &= z_{init} \;=\; -v_0 \\ z_{max_m} &= \min(-v_{fm}, -v_{LRmin}, -v_{S min}, -v_{min}(m)), \end{aligned}$$

and for each $i \in [1, m-1]$

$$\begin{aligned} z_{max_i} = \min\Bigg(& -\frac{v_{LRmin}}{\sqrt{2}\max(\sin\omega_i, \cos\omega_i)}, -\frac{v_{LRmax}}{\sqrt{2}\min(\sin\omega_i, \cos\omega_i)}, \\ & -\frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\sqrt{1 + (e'\tilde{\kappa}_i)^2}}\,v_{S min}, \\ & -\frac{\sqrt{2}}{\sin\omega_i + \cos\omega_i}\,v_{min}(i), \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\,|\tilde{\kappa}_i|}\,\dot{\theta}_{max}(i), \\ & \frac{\sqrt{\dfrac{-2a_{Cm}}{\tilde{\kappa}_i}}}{\sin\omega_i + \cos\omega_i}, \frac{\sqrt{2}}{(\sin\omega_i + \cos\omega_i)\left|\dfrac{d\theta_{S_i}}{ds}\right|}\,\dot{\theta}_{S max}\Bigg), \end{aligned}$$

where

- the second term is only considered if $\sin\omega_i < 0$ or $\cos\omega_i < 0$;
- the fifth and sixth terms (involving $\dot{\theta}_{max}(i)$ and a_{Cm}) are only considered if $\tilde{\kappa}_i \ne 0$;

- the seventh term (involving $\dot{\theta}_{S\,max}$) is only considered if $\kappa_{i-1} \neq \kappa_i$;
- the constant a_{Cm} is defined as follows:

$$a_{Cm} = \begin{cases} a_{C\,max} & \text{if } \tilde{\kappa}_i < 0, \\ a_{C\,min} & \text{if } \tilde{\kappa}_i > 0. \end{cases}$$

A.2.2 Transition Constraints

A.2.2.1 Forward Stage

In this section, we address the computation, for each $i \in [0, m-1]$, of the function $z_{maxF_i}(z)$ in terms of the parameters $a_{T\,min}$, $a_{T\,max}$, and the functions a_{LRmin}, a_{LRmax} $a_{S\,min}$ and $a_{S\,max}$ (all these parameters and functions are defined in Section 4.1.5).

Recall that, by our convention, the distance s_i travelled by the reference point of the robot is negative for all $i \in [0, m-1]$ in the case of a backward path segment.

The amount of time Δt_i needed for performing this step is given by

$$\Delta t_i = \frac{2 s_i}{v_i + v_{i+1}} = \frac{-2\sqrt{2}\, s_i}{(\sin \omega_i + \cos \omega_i)\, z_i + (\sin \omega_{i+1} + \cos \omega_{i+1})\, z_{i+1}}.$$

■ **$a_{T\,min}$ and $a_{T\,max}$ constraints**

The constraints

$$a_{T\,min} \leq a_{T_i} \leq a_{T\,max}$$

yields

$$\begin{cases} z_{maxF_i}(z) \geq \begin{cases} \dfrac{\sqrt{(\sin \omega_i + \cos \omega_i)^2 z^2 + 4 a_{T\,max}\, s_i}}{\sin \omega_{i+1} + \cos \omega_{i+1}} & \text{if } (\sin \omega_i + \cos \omega_i)^2 z^2 + 4 a_{T\,max}\, s_i \geq 0 \\ 0 & \text{otherwise} \end{cases} \\ z_{maxF_i}(z) \leq \dfrac{\sqrt{(\sin \omega_i + \cos \omega_i)^2 z^2 + 4 a_{T\,min}\, s_i}}{\sin \omega_{i+1} + \cos \omega_{i+1}} \end{cases}$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

■ **$a_{LRmin}(v_{L_i})$, $a_{LRmax}(v_{L_i})$, $a_{LRmin}(v_{R_i})$ and $a_{LRmax}(v_{R_i})$ constraints**

For the left wheel, the constraints

$$a_{LRmin}(v_{L_i}) \leq a_{L_i} \leq a_{LRmax}(v_{L_i})$$

become

$$\begin{cases} \cos\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\,z_{i+1}^2 + \sin(\omega_i - \omega_{i+1})\,z_i\,z_{i+1} - \\ \qquad\qquad \cos\omega_i\,(\sin\omega_i + \cos\omega_i)\,z_i^2 - 2a_{LRmin}(v_{L_i})\,s_i \leq 0 \\ \\ -\cos\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\,z_{i+1}^2 - \sin(\omega_i - \omega_{i+1})\,z_i\,z_{i+1} + \\ \qquad\qquad \cos\omega_i\,(\sin\omega_i + \cos\omega_i)\,z_i^2 + 2a_{LRmax}(v_{L_i})\,s_i \leq 0. \end{cases}$$

Each of these two inequations is of the form $a\,z_{i+1}^2 + b\,z_{i+1} + c \leq 0$, and thus can be solved as for the forward segment case. Those inequations determine a set of possible values for z_{i+1}. Let $S_{Lmin}(z_i)$ and $S_{Lmax}(z_i)$ denote these two sets (respectively resulting from the first and the second inequation), we then have

$$z_{maxF_i}(z) \in S_{Lmin}(z) \cap S_{Lmax}(z)$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Similarly, for the right wheel, the constraints

$$a_{LRmin}(v_{R_i}) \leq a_{R_i} \leq a_{LRmax}(v_{R_i})$$

become

$$\begin{cases} \sin\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\,z_{i+1}^2 + \sin(\omega_{i+1} - \omega_i)\,z_i\,z_{i+1} - \\ \qquad\qquad \sin\omega_i\,(\sin\omega_i + \cos\omega_i)\,z_i^2 - 2a_{LRmin}(v_{R_i})\,s_i \leq 0 \\ \\ -\sin\omega_{i+1}\,(\sin\omega_{i+1} + \cos\omega_{i+1})\,z_{i+1}^2 - \sin(\omega_{i+1} - \omega_i)\,z_i\,z_{i+1} + \\ \qquad\qquad \sin\omega_i\,(\sin\omega_i + \cos\omega_i)\,z_i^2 + 2a_{LRmax}(v_{R_i})\,s_i \leq 0. \end{cases}$$

Each of these two inequations is also of the form $a\,z_{i+1}^2 + b\,z_{i+1} + c \leq 0$, and can be solved as for the forward segment case. Those inequations determine a set of possible values for z_{i+1}. Let $S_{Rmin}(z_i)$ and $S_{Rmax}(z_i)$ denote these two sets (respectively resulting from the first and the second inequation), we then have

$$z_{maxF_i}(z) \in S_{Rmin}(z) \cap S_{Rmax}(z)$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Note that, if $\omega_i = \omega_{i+1} = \frac{\pi}{4}$ (i.e., if $\tilde{\kappa}_i = \tilde{\kappa}_{i+1} = 0$), the robot follows a straight line during the $(i+1)$-th step, and, in order to speed up the computation time, we can also only address the constraints for the left wheel, as the set of solutions for the right wheel will be identical to the one provided by the left wheel.

■ $a_{S\,min}(v_{S_i})$ and $a_{S\,max}(v_{S_i})$ constraints

Finally, the constraints

$$a_{S\,min}(v_{S_i}) \le a_{S_i} \le a_{S\,max}(v_{S_i})$$

become

$$\begin{cases} \dfrac{sc_{i+1}^2}{\cos\theta_{S_{i+1}}}\, z_{i+1}^2 + \left(\dfrac{1}{\cos\theta_{S_{i+1}}} - \dfrac{1}{\cos\theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} - \dfrac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 - 4a_{S\,min}(v_{S_i})\, s_i & \le 0 \\ -\dfrac{sc_{i+1}^2}{\cos\theta_{S_{i+1}}}\, z_{i+1}^2 - \left(\dfrac{1}{\cos\theta_{S_{i+1}}} - \dfrac{1}{\cos\theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} + \dfrac{sc_i^2}{\cos\theta_{S_i}}\, z_i^2 + 4a_{S\,max}(v_{S_i})\, s_i & \le 0. \end{cases}$$

where

$$sc_i = \sin\omega_i + \cos\omega_i,$$

$$sc_{i+1} = \sin\omega_{i+1} + \cos\omega_{i+1},$$

Once again, each of these two inequations is also of the form $a\, z_{i+1}^2 + b\, z_{i+1} + c \le 0$, and can be solved as for the forward segment case. Those inequations determine a set of possible values for z_{i+1}. Let $S_{S\,min}(z_i)$ and $S_{S\,max}(z_i)$ denote these two sets (respectively resulting from the first and the second inequation), we then have

$$z_{maxF_i}(z) \in S_{S\,min}(z) \cap S_{S\,max}(z)$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Summary

In summary, we obtain for the function z_{maxF_i}:

$$\begin{aligned} z_{maxF_i}(z) = \max\Big(& [z_{T\,min}(z),\ z_{T\,max}(z)] \cap \\ & S_{L\,min}(z) \cap S_{L\,max}(z) \cap \\ & S_{R\,min}(z) \cap S_{R\,max}(z) \cap \\ & S_{S\,min}(z) \cap S_{S\,max}(z)\Big), \end{aligned}$$

for all $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$, where

- sc_i and sc_{i+1} are defined as follows:

$$sc_i = \sin \omega_i + \cos \omega_i,$$
$$sc_{i+1} = \sin \omega_{i+1} + \cos \omega_{i+1};$$

- $z_{T\,min}(z)$ and $z_{T\,max}(z)$ are defined as follows:

$$z_{T\,min}(z) = \begin{cases} \dfrac{\sqrt{sc_i^2\, z^2 + 4a_{T\,max}\, s_i}}{sc_{i+1}} & \text{if } sc_i^2\, z^2 + 4a_{T\,max}\, s_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$z_{T\,max}(z) = \frac{\sqrt{sc_i^2\, z^2 + 4a_{T\,min}\, s_i}}{sc_{i+1}};$$

- $S_{L\,min}(z)$, $S_{L\,max}(z)$, $S_{R\,min}(z)$, $S_{R\,max}(z)$, $S_{S\,min}(z)$ and $S_{S\,max}(z)$ are respectively the sets of solutions for z_{i+1} of the following six quadratic inequations:

$$\cos \omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 + \sin(\omega_i - \omega_{i+1})\, z\, z_{i+1} - \cos \omega_i\, sc_i\, z^2 - 2a_{LRmin}(v_{L_i})\, s_i \leq 0$$

$$-\cos \omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 - \sin(\omega_i - \omega_{i+1})\, z\, z_{i+1} + \cos \omega_i\, sc_i\, z^2 + 2a_{LRmax}(v_{L_i})\, s_i \leq 0$$

$$\sin \omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 + \sin(\omega_{i+1} - \omega_i)\, z\, z_{i+1} - \sin \omega_i\, sc_i\, z^2 - 2a_{LRmin}(v_{R_i})\, s_i \leq 0$$

$$-\sin \omega_{i+1}\, sc_{i+1}\, z_{i+1}^2 - \sin(\omega_{i+1} - \omega_i)\, z\, z_{i+1} + \sin \omega_i\, sc_i\, z^2 + 2a_{LRmax}(v_{R_i})\, s_i \leq 0$$

$$\frac{sc_{i+1}^2}{\cos \theta_{S_{i+1}}}\, z_{i+1}^2 + \left(\frac{1}{\cos \theta_{S_{i+1}}} - \frac{1}{\cos \theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} - \frac{sc_i^2}{\cos \theta_{S_i}}\, z^2 - 4a_{S\,min}(v_{S_i})\, s_i \leq 0$$

$$-\frac{sc_{i+1}^2}{\cos \theta_{S_{i+1}}}\, z_{i+1}^2 - \left(\frac{1}{\cos \theta_{S_{i+1}}} - \frac{1}{\cos \theta_{S_i}}\right) sc_i\, sc_{i+1}\, z_i\, z_{i+1} + \frac{sc_i^2}{\cos \theta_{S_i}}\, z^2 + 4a_{S\,max}(v_{S_i})\, s_i \leq 0,$$

where

$$v_{L_i} = -z_i \sqrt{2} \cos \omega_i,$$
$$v_{R_i} = -z_i \sqrt{2} \sin \omega_i,$$
$$v_{S_i} = -\frac{\sin \omega_i + \cos \omega_i}{\sqrt{2} \cos \theta_{S_i}}\, z_i,$$

and where the third and fourth inequations can only be considered if $\tilde{\kappa}_i \neq 0$ or $\tilde{\kappa}_{i+1} \neq 0$.

A.2.2.2 Backward Stage

In this section, we address the computation, for each $i \in [1,m]$, of the function $z_{maxP_i}(z)$ in terms of the parameters $a_{T\,min}$, $a_{T\,max}$, and the functions a_{LRmin}, a_{LRmax} $a_{S\,min}$ and $a_{S\,max}$ (all these parameters and functions are defined in Section 4.1.5).

The reasoning is once again similar to the case of forward segments, and we obtain for the function z_{maxP_i}:

$$z_{maxP_i}(z) = \max\Big([z_{T\,min}(z),\ z_{T\,max}(z)] \cap SP_{L\,min}(z) \cap SP_{L\,max}(z) \cap \\ SP_{R\,min}(z) \cap SP_{R\,max}(z) \cap SP_{S\,min}(z) \cap SP_{S\,max}(z)\Big),$$

for all $i \in [1,m]$ and $z \in [0, z_{max_i}]$,
where

- sc_{i-1} and sc_i are defined as follows:

$$sc_{i-1} = \sin\omega_{i-1} + \cos\omega_{i-1},$$
$$sc_i = \sin\omega_i + \cos\omega_i;$$

- $z_{T\,min}(z)$ and $z_{T\,max}(z)$ are defined as follows:

$$z_{T\,min}(z) = \begin{cases} \dfrac{\sqrt{sc_i^2\, z^2 - 4a_{T\,min}\, s_{i-1}}}{sc_{i-1}} & \text{if } sc_i^2\, z^2 - 4a_{T\,min}\, s_{i-1} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$z_{T\,max}(z) = \frac{\sqrt{sc_i^2\, z^2 - 4a_{T\,max}\, s_{i-1}}}{sc_{i-1}};$$

- $SP_{L\,min}(z)$, $SP_{L\,max}(z)$, $SP_{R\,min}(z)$, $SP_{R\,max}(z)$, $SP_{S\,min}(z)$ and $SP_{S\,max}(z)$ are respectively the sets of solutions for z_{i-1} of the following six quadratic inequations:

$$-\cos\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 + \sin(\omega_{i-1} - \omega_i)\, z\, z_{i-1} + \cos\omega_i\, sc_i\, z^2 - 2a_{LRmin}(\tilde{v}_{L_{i-1}})\, s_{i-1} \leq 0$$

$$\cos\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 - \sin(\omega_{i-1} - \omega_i)\, z\, z_{i-1} - \cos\omega_i\, sc_i\, z^2 + 2a_{LRmax}(\tilde{v}_{L_{i-1}})\, s_{i-1} \leq 0$$

$$-\sin\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 + \sin(\omega_i - \omega_{i-1})\, z\, z_{i-1} + \sin\omega_i\, sc_i\, z^2 - 2a_{LRmin}(\tilde{v}_{R_{i-1}})\, s_{i-1} \leq 0$$

$$\sin\omega_{i-1}\, sc_{i-1}\, z_{i-1}^2 - \sin(\omega_i - \omega_{i-1})\, z\, z_{i-1} - \sin\omega_i\, sc_i\, z^2 + 2a_{LRmax}(\tilde{v}_{R_{i-1}})\, s_{i-1} \leq 0$$

$$-\frac{sc_{i-1}^2}{\cos\theta_{S_{i-1}}} z_{i-1}^2 + \left(\frac{1}{\cos\theta_{S_i}} - \frac{1}{\cos\theta_{S_{i-1}}}\right) sc_{i-1}\, sc_i\, z_i\, z_{i-1} + \frac{sc_i^2}{\cos\theta_{S_i}} z_i^2 - 4a_{S\,min}(\tilde{v}_{S_{i-1}})\, s_{i-1} \leq 0$$

$$\frac{sc_{i-1}^2}{\cos\theta_{S_{i-1}}} z_{i-1}^2 - \left(\frac{1}{\cos\theta_{S_i}} - \frac{1}{\cos\theta_{S_{i-1}}}\right) sc_{i-1}\, sc_i\, z_i\, z_{i-1} - \frac{sc_i^2}{\cos\theta_{S_i}} z_i^2 + 4a_{S\,max}(\tilde{v}_{S_{i-1}})\, s_{i-1} \leq 0,$$

where

$$\tilde{v}_{L_{i-1}} = -\tilde{z}_{i-1}\sqrt{2}\cos\omega_{i-1},$$

$$\tilde{v}_{R_{i-1}} = -\tilde{z}_{i-1}\sqrt{2}\sin\omega_{i-1},$$

$$\tilde{v}_{S_{i-1}} = -\frac{\sin\omega_{i-1} + \cos\omega_{i-1}}{\sqrt{2}\cos\theta_{S_{i-1}}}\tilde{z}_{i-1},$$

with

$$\tilde{z}_{i-1} = \min(\check{z}_{i-1}, z_i),$$

where $\check{z}_{i-1}$ is the tentative value for the velocity at the beginning of the i-th step obtained after the forward stage.

As in the forward stage, the third and fourth inequations can only be considered if $\tilde{\kappa}_i \neq 0$ or $\tilde{\kappa}_{i+1} \neq 0$.

A.2.3 Wheel Speeds

We finally recall the formulas that express the speeds of the left, right and steering wheels in terms of the values of the variables z_i.

We have for all $i \in [0, m]$:

$$v_{L_i} = -z_i\sqrt{2}\cos\omega_i,$$

$$v_{R_i} = -z_i\sqrt{2}\sin\omega_i,$$

$$v_{S_i} = -z_i\,\frac{\sin\omega_i + \cos\omega_i}{\sqrt{2}\cos\theta_{S_i}}$$

where

$$\omega_i = \arctan\left(1 + \frac{e\tilde{\kappa}_i}{2}, 1 - \frac{e\tilde{\kappa}_i}{2}\right).$$

A.3 Local Turns

The case of local turn segments, is simpler than forward or backward segments. Recall, from Section 3.1.2, that local turns are such that

$$v_i = \frac{v_{L_i} + v_{R_i}}{2} = 0$$

for every $i \in [0, m]$.

We therefore have

$$a_{T_i} = a_{C_i} = 0$$

for every $i \in [0, m]$, and the constraints

$$v_{min}(i) \leq v_i \leq v_{max}(i),$$
$$a_{T\,min} \leq a_{T_i} \leq a_{T\,max},$$
$$a_{C\,min} \leq a_{C_i} \leq a_{C\,max},$$

naturally always holds.

A.3.1 Right Local Turns

Let us now develop the constraints for the case of a path segment (m, σ) performing a right local turn, i.e., such that $\nu_i = R$ for all $i \in [0, m-1]$. From Equation 4.1 and Definition 4.1.4, we have for all $i \in [0, m]$

$$z_i = \sqrt{\frac{v_{L_i}^2 + v_{R_i}^2}{2}} = v_{L_i},$$

in other words, the variables z_i undergoing the optimization process correspond to the speed of the left wheel.

A.3.1.1 Local State Constraints

We first address the computation of the bounds z_{max_i} from the parameters v_{LRmin}, v_{LRmax}, $v_{S\,max}$ and the function $\dot{\theta}_{max}$.

At the extremities of the segment, we have

$$z_{max_0} = z_{init} = 0$$
$$z_{max_m} = 0.$$

For each $i \in [1, m-1]$, the speeds of the left and right wheels are given by

$$\begin{aligned} v_{L_i} &= z_i, \\ v_{R_i} &= -z_i. \end{aligned}$$

The speed of the steering wheel and the angular speed of the robot are given, respectively from Equation 3.22 and Equation 2.5, by

$$\begin{aligned} v_{S_i} &= \frac{2e' z_i}{e}, \\ \left|\dot{\theta}_i\right| &= \left|\frac{v_{R_i} + v_{L_i}}{2}\right| = \frac{2z_i}{e}, \end{aligned}$$

where e denotes the axle width and e' denotes the wheelbase of the robot.

For each $i \in [1, m-1]$, the bounds on the speeds of the three wheels and on the angular speed of the robot

$$\begin{aligned} v_{L_i} &\leq v_{LRmax} \\ v_{R_i} &\geq -v_{LRmin} \\ v_{S_i} &\leq v_{Smax} \\ \left|\dot{\theta}_i\right| &\leq dot\theta_{max}(i), \end{aligned}$$

yield

$$\begin{aligned} z_{max_i} &\leq v_{LRmax} \\ z_{max_i} &\leq -v_{LRmin} \\ z_{max_i} &\leq \frac{e}{2e'}\, v_{Smax} \\ z_{max_i} &\leq \frac{e}{2}\, \dot{\theta}_{max}(i), \end{aligned}$$

from which we obtain

$$z_{max_i} = \min\left(v_{LRmax}, -v_{LRmin}, \frac{e}{2e'}\, v_{Smax}, \frac{e}{2}\, \dot{\theta}_{max}(i)\right).$$

Note that, the constraint $v_{S_i} \geq v_{Smin}$ naturally always holds, since local turns are defined such that $v_{S_i} \geq 0$.

A.3.1.2 Transition Constraints

Forward Stage

Next, for each $i \in [0, m-1]$, we define the function z_{maxF_i} in terms of the functions a_{LRmin}, a_{LRmax} and a_{Smax}.

We know that the distance driven by the left wheel (i.e., the length of the circular arc) during the $(i+1)$-th step is given by

$$s_{L_i} = \begin{cases} \frac{e}{2}\left[\theta_i - \theta_{i+1}\right] & \text{if } \left[\theta_i - \theta_{i+1}\right] > 0, \\ \frac{e}{2}\left(\left[\theta_i - \theta_{i+1}\right] + 2\pi\right) & \text{if } \left[\theta_i - \theta_{i+1}\right] < 0. \end{cases}$$

If we assume constant-acceleration motion during the step, let a_{L_i} and Δt_i denote respectively the linear acceleration of the left wheel during that step, and the duration of the step, then we have

$$v_{L_{i+1}} = v_{L_i} + a_{L_i}\Delta t_i$$

and

$$s_{L_i} = \frac{v_{L_{i+1}} + v_{L_i}}{2}\Delta t_i,$$

from which we extract

$$a_{L_i} = \frac{v_{L_{i+1}}^2 - v_{L_i}^2}{2 s_{L_i}}.$$

The constraint

$$a_{L_i} \leq a_{LRmax}(v_{L_i})$$

can be rewritten as

$$v_{L_{i+1}} \leq \sqrt{v_{L_i}^2 + 2 s_{L_i}\, a_{LRmax}(v_{L_i})},$$

which yields

$$z_{maxF_i}(z) \leq \sqrt{z^2 + 2 s_{L_i}\, a_{LRmax}(z)}$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Note that the speed of the right wheel is negative, and that we have to take into account the deceleration constraint for this wheel.

Recall that we have

$$v_{R_i} = -v_{L_i},$$
$$s_{r_i} = -s_{L_i},$$

therefore, the constraint

$$a_{R_i} \geq a_{LRmin}(v_{R_i})$$

can be rewritten as

$$|v_{R_{i+1}}| \leq \sqrt{v_{R_i}^2 + 2s_{R_i}\, a_{LRmin}(v_{R_i})},$$

which is equivalent to

$$v_{L_{i+1}} \leq \sqrt{v_{L_i}^2 - 2s_{L_i}\, a_{LRmin}(-v_{L_i})},$$

and yields

$$z_{maxF_i}(z) \leq \sqrt{z^2 - 2s_{L_i}\, a_{LRmin}(-z)}.$$

The upper bound on the tangential acceleration a_{S_i} of the steering wheel during the $(i+1)$-th step is handled in a similar way. Recall that we have in the case of a right local turn

$$v_{S_i} = \frac{2e'}{e}\, v_{L_i}$$

and

$$a_{S_i} = \frac{2e'}{e}\, a_{L_i},$$

where v_{S_i} denotes the linear speed of the steering wheel.

We thus obtain for every $i \in [0, m-1]$

$$a_{S_i} = \frac{e'}{e} \cdot \frac{v_{L_{i+1}}^2 - v_{L_i}^2}{s_{L_i}}.$$

The constraint

$$a_{S_i} \leq a_{Smax}(v_{S_i})$$

then becomes

$$v_{L_{i+1}} \leq \sqrt{v_{L_i}^2 + \frac{e}{e'}\, s_{L_i} a_{Smax}\left(\frac{2e'}{e} v_{L_i}\right)},$$

which leads to

$$z_{maxF_i}(z) \leq \sqrt{z^2 + \frac{e}{e'}\, s_{L_i}\, a_{Smax}\left(\frac{2e'z}{e}\right)}.$$

In summary, we obtain for the function z_{maxF_i}:

$$z_{maxF_i}(z) = \sqrt{z^2 + s_{L_i} \min\left(2a_{LRmax}(z),\ -2a_{LRmin}(-z),\ \frac{e}{e'}\, a_{Smax}\left(\frac{2e'z}{e}\right)\right)},$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$.

Backward stage

We now express the function z_{maxP_i} in terms of the functions a_{LRmin}, a_{LRmax} and a_{Smin}.

Once again, as explained in Section 4.2.5, constraints expressed as functions may depend on z_{i-1}, and in order to ensure the satisfiability of the constraints during the backward stage, we will use the following approximation of the velocity:

$$\tilde{z}_{i-1} = \min(\check{z}_{i-1}, z_i),$$

where $\check{z}_{i-1}$ is the tentative value for the velocity at the beginning of the i-th step obtained after the forward stage.

We have, for every $i \in [1, m]$,

$$a_{L_{i-1}} = \frac{v_{L_i}^2 - v_{L_{i-1}}^2}{2s_{L_{i-1}}}.$$

The constraint

$$a_{L_{i-1}} \geq a_{LRmin}(\tilde{v}_{L_{i-1}})$$

thus becomes

$$v_{L_{i-1}} \leq \sqrt{v_{L_i}^2 - 2s_{L_{i-1}}\, a_{LRmin}(\tilde{v}_{L_{i-1}})},$$

which gives

$$z_{maxP_i}(z) \leq \sqrt{z^2 - 2s_{L_{i-1}}\, a_{LRmin}(\tilde{z}_{i-1})}.$$

Similarly to the forward stage, the constraint on the right wheel

$$a_{R_{i-1}} \leq a_{LRmax}(\tilde{v}_{R_{i-1}})$$

yields

$$z_{maxP_i}(z) \leq \sqrt{z^2 + 2s_{L_{i-1}}\, a_{LRmax}(-\tilde{z}_{i-1})}.$$

By a similar reasoning applied to the constraint

$$a_{S_i} \geq a_{S\,min}(v_{S_i}),$$

we obtain

$$z_{maxP_i}(z) \leq \sqrt{z^2 - \frac{e}{e'}\, s_{L_{i-1}}\, a_{S\,min}\left(\frac{2e'\tilde{z}_{i-1}}{e}\right)}.$$

In summary, we obtain for the function z_{maxP_i}:

$$z_{maxP_i}(z) = \sqrt{z^2 - s_{L_{i-1}} \max\left(2a_{LRmin}(\tilde{z}_{i-1}),\ -2a_{LRmax}(-\tilde{z}_{i-1}),\ \frac{e}{e'}\, a_{S\,min}\left(\frac{2e'\tilde{z}_{i-1}}{e}\right)\right)},$$

for every $i \in [1, m]$ and $z \in [0, z_{max_i}]$.

A.3.1.3 Wheel Speeds

Finally, we recall the formulas that express the speeds of the left, right, and steering wheels of the robot in terms of the variables z_i.

We have for all $i \in [0, m]$:

$$v_{L_i} = z_i,$$

$$v_{R_i} = -z_i,$$

$$v_{S_i} = \frac{2e'}{e}\, z_i.$$

A.3.2 Left Local Turns

The case of a path segment (m, σ) performing a left local turn, i.e., such that $v_i = L$ for all $i \in [0, m-1]$, is handled in essentially the same way as in Section A.3.1.

We now have

$$z_i = v_{R_i}$$

for all $i \in [0, m]$, and

$$s_{R_i} = \begin{cases} \frac{e}{2}\,[\theta_{i+1} - \theta_i] & \text{if } [\theta_{i+1} - \theta_i] > 0, \\ \frac{e}{2}\,([\theta_{i+1} - \theta_i] + 2\pi) & \text{if } [\theta_{i+1} - \theta_i] < 0. \end{cases}$$

for all $i \in [0, m-1]$.

We finally obtain

$$z_{max_0} = z_{init} = 0$$
$$z_{max_m} = 0,$$

$$z_{max_i} = \min\Big(v_{LRmax},\ -v_{LRmin},\ \frac{e}{2e'}\, v_{S\,max},\ \frac{e}{2}\,\dot{\theta}_{max}(i)\Big)$$

for all $i \in [1, m-1]$,

$$z_{maxF_i}(z) = \sqrt{z^2 + s_{R_i} \min\Big(2a_{LRmax}(z),\ -2a_{LRmin}(-z),\ \frac{e}{e'}\, a_{S\,max}\Big(\frac{2e'z}{e}\Big)\Big)},$$

for every $i \in [0, m-1]$ and $z \in [0, z_{max_i}]$, and

$$z_{maxP_i}(z) = \sqrt{z^2 - s_{R_{i-1}} \max\Big(2a_{LRmin}(\tilde{z}_{i-1}),\ -2a_{LRmax}(-\tilde{z}_{i-1}),\ \frac{e}{e'}\, a_{S\,min}\Big(\frac{2e'\tilde{z}_{i-1}}{e}\Big)\Big)},$$

for every $i \in [1, m]$ and $z \in [0, z_{max_i}]$.

After the optimization procedure has been carried out, the speeds of the wheels are given by

$$v_{L_i} = -z_i,$$
$$v_{R_i} = z_i,$$
$$v_{S_i} = \frac{2e'}{e}\, z_i.$$

with

$$\tilde{z}_{i-1} = \min(\check{z}_{i-1}, z_i),$$

where $\check{z}_{i-1}$ is the tentative value for the velocity at the beginning of the i-th step obtained after the forward stage.

A.4 Static Steps

The case of static steps is much easier to handle than forward, backward or local turn path segments. First, since a sequence of static steps is semantically equivalent to a single one, we can assume w.l.o.g. that every static segment is composed of a single static step (i.e., that the path segment (m, σ) under analysis is such that $m = 1$ and $v_0 = S$). The problem then reduces to computing the delay $\mu(0)$ associated to this step. There are two situations to consider:

- *With a differential drive locomotion platform.* No specific constraint has to be satisfied, hence one has

$$\mu(0) = 0.$$

- *With a tricycle locomotion platform.* In this case, the maximum rate of variation of the steering angle must not be exceeded. The angle steered during the step is given by

$$[\theta_{S_1} - \theta_{S_0}],$$

where for each $i \in \{0, 1\}$, we have

$$\theta_{S_i} = \begin{cases} \arctan(e'\tilde{\kappa}_i) & \text{if } \tilde{\kappa}_i \neq \{-\infty, +\infty\} \\ \dfrac{\pi}{2} & \text{if } \tilde{\kappa}_i = +\infty \\ -\dfrac{\pi}{2} & \text{if } \tilde{\kappa}_i = -\infty. \end{cases}$$

The constraint

$$\frac{|[\theta_{S_1} - \theta_{S_0}]|}{\Delta t_0} \leq \dot{\theta}_{S\,max}$$

then yields

$$\mu(0) = \Delta t_0 = \frac{|[\theta_{S_1} - \theta_{S_0}]|}{\dot{\theta}_{S\,max}}.$$

A.5 Reciprocal of the $\hat{s}_i(v_w)$ Function

We now present the developments related to the calculation of the reciprocal of the $\hat{s}_i(v_w)$ function introduced in Section 4.2.4.

Recall that, if the maximal acceleration of a wheel is given as a function of the wheel speed $v_w \geq 0$ by

$$a(v_w) = a_0 - b\, v_w,$$

then $\hat{s}_i(v_w)$ is defined as the function that gives out the distance travelled by that wheel when its speed reaches $v_w \geq v_{w_i}$ with maximal acceleration, starting from an initial speed of $v_{w_i} \geq 0$.

We have

$$\hat{s}_i(v_w) = \frac{b\,(v_{w_i} - v_w) + a_0 \log \frac{a_0 - b\,v_{w_i}}{a_0 - b\,v_w}}{b^2}.$$

with $a_0 > 0$, $b > 0$ and $v_{w_i} \geq 0$.

Calculating the reciprocal function, we have

$$\begin{aligned}
& s = \frac{b\,(v_{w_i} - v_w) + a_0 \log \frac{a_0 - b\,v_{w_i}}{a_0 - b\,v_w}}{b^2} \\
\Leftrightarrow\ & b\,s = v_{w_i} - v_w + \frac{a_0}{b}\log(a_0 - b\,v_{w_i}) - \frac{a_0}{b}\log(a_0 - b\,v_w) \\
\Leftrightarrow\ & v_w + \frac{a_0}{b}\log(a_0 - b\,v_w) = v_{w_i} - b\,s + \frac{a_0}{b}\log(a_0 - b\,v_{w_i}) \\
\Leftrightarrow\ & \frac{b}{a_0}v_w + \log(a_0 - b\,v_w) = \frac{b}{a_0}\,(v_{w_i} - b\,s) + \log(a_0 - b\,v_{w_i}) \\
\Leftrightarrow\ & \mathrm{e}^{\frac{b}{a_0}v_w}\,(a_0 - b\,v_w) = \mathrm{e}^{\frac{b}{a_0}(v_{w_i} - b\,s)}\,(a_0 - b\,v_{w_i}) \\
\Leftrightarrow\ & \mathrm{e}^{\frac{b}{a_0}v_w}\left(1 - \frac{b}{a_0}v_w\right) = \mathrm{e}^{\frac{b}{a_0}(v_{w_i} - b\,s)}\left(1 - \frac{b}{a_0}v_{w_i}\right).
\end{aligned}$$

One can remark that this equation is of the form

$$\mathrm{e}^x\,(1 - x) = C, \tag{A.7}$$

and, in terms of the Lambert W-function $W(x)$ defined as the reciprocal function of $f(W) = W\mathrm{e}^W$, we obtain

$$W(x\,\mathrm{e}^x) = x,$$

and thus Equation A.7 can be solved for x in the following way:

$$\begin{aligned}
& \mathrm{e}^x\,(1 - x) = C \\
\Leftrightarrow\ & \mathrm{e}^x\,(x - 1) = -C \\
\Leftrightarrow\ & (x - 1)\,\mathrm{e}^{x-1} = -\frac{C}{\mathrm{e}} \\
\Leftrightarrow\ & x - 1 = W\!\left(-\frac{C}{\mathrm{e}}\right) \\
\Leftrightarrow\ & x = 1 + W\!\left(-\frac{C}{\mathrm{e}}\right).
\end{aligned}$$

Using this result, we have

$$\begin{aligned}
&\mathrm{e}^{\frac{b}{a_0} v_w}\left(1-\frac{b}{a_0} v_w\right)=\mathrm{e}^{\frac{b}{a_0}\left(v_{w_i}-b\, s\right)}\left(1-\frac{b}{a_0} v_{w_i}\right) \\
\Leftrightarrow\ & \frac{b}{a_0} v_w=1+W\left(-\frac{\mathrm{e}^{\frac{b}{a_0}\left(v_{w_i}-b\, s\right)}\left(1-\frac{b}{a_0} v_{w_i}\right)}{\mathrm{e}}\right) \\
\Leftrightarrow\ & v_w=\frac{a_0}{b}\left(1+W\left(\left(\frac{b}{a_0} v_{w_i}-1\right) \mathrm{e}^{\left(\frac{b}{a_0}\left(v_{w_i}-b\, s\right)-1\right)}\right)\right).
\end{aligned}$$

Note that, for this equation to hold, we must have

$$\begin{aligned}
&\left(\frac{b}{a_0} v_{w_i}-1\right) \mathrm{e}^{\left(\frac{b}{a_0}\left(v_{w_i}-b\, s\right)-1\right)} \geq-\frac{1}{\mathrm{e}} \\
\Leftrightarrow\ & s \geq \frac{b\, v_{w_i}+a_0 \log \frac{a_0-b\, v_{w_i}}{a_0}}{b^2} \\
\Leftrightarrow\ & s \geq \hat{s}_i(0),
\end{aligned}$$

since $v_{w_i} \geq 0$ and since we are accelerating (i.e., the speed can only increase or stay unchanged), we have $s \geq 0$, which proves that this inequation is always verified.

For the same reason, as the speed stays positive, we have $v_w \geq 0$ and thus

$$W\left(\left(\frac{b}{a_0} v_{w_i}-1\right) \mathrm{e}^{\left(\frac{b}{a_0}\left(v_{w_i}-b\, s\right)-1\right)}\right) \geq -1.$$

Note that, $W(x)$ is double-valued for $x \in [-\frac{1}{\mathrm{e}}, 0[$, the constraint $W(x) \geq -1$ defines the function $W_0(x)$ (i.e., the main branch of the Lambert W-function), the constraint $W(x) \leq -1$ defines the lower branch $W_{-1}(x)$.

We finally have

$$\hat{v}_i(s_w)=\frac{a_0}{b}\left(1+W_0\left(\left(\frac{b}{a_0} v_{w_i}-1\right) \mathrm{e}^{\left(\frac{b}{a_0}\left(v_{w_i}-b\, s_w\right)-1\right)}\right)\right),$$

where $\hat{v}_i(s_w)$ denotes the function that compute the speed reached by the wheel with maximal acceleration after travelling the distance s_w, starting from an initial wheel speed of $v_{w_i} \geq 0$.

APPENDIX B

Chaining Pairs of Clothoids

This appendix is related to developments of Chapter 5, and presents the proof of Theorem 5.4.1 that states that there is always an unique pair of clothoids that interpolates the path within the safe zone.

B.1 Existence and Uniqueness of a Pair of Clothoids

Theorem 5.4.1. *For every rotation angle $\beta_i \in]0, \frac{\pi}{2}]$ and for every pair of curvatures κ_1, κ_2 such that $0 \leq \kappa_1 < \kappa_C$ and $0 \leq \kappa_2 < \kappa_C$, there exist two clothoids arcs moving respectively from the curvatures κ_1 to κ_M and from κ_M to κ_2, with $\kappa_M > \kappa_C$, the concatenation of which interpolates the path from t_{i_1} to t_{i_2} within the safe zone, with initial and final tangential angles of respectively θ_1 and $\theta_2 = \theta_1 + \beta_i$ and with continuity of the tangent vector at the junction point between the two curves. The parameters of these two clothoids arcs are uniquely determined by κ_1, κ_2, κ_C, and the rotation angle β_i.*

In this section, we assume w.l.o.g. that the tangential angle[1] θ_1 at the origin (i.e., at t_{i_1}) is equal to $-\frac{\pi}{2}$. The problem is illustrated in Figure B.1 (exaggerating the curvatures in order to make the interpolated path stand out from the circle arc).

Let s_M denote the distance travelled along the first arc, s_F the total distance travelled over both arcs, and $s_{FM} = s_F - s_M$ the distance travelled along the second arc. The linear rate of variation of curvature for these arcs are respectively denoted by c_1 and $-c_2$.

[1] Note that, as in all chapters, the orientation is chosen such that $\theta = 0$ corresponds to a direction that follows the y-axis.

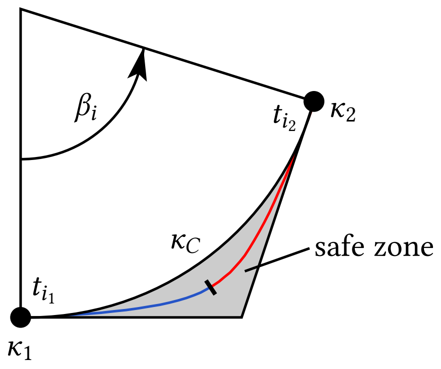


Figure B.1 – Path geometry.

From Equation 5.4, one has

$$\kappa_M = \kappa_1 + c_1 s_M = \kappa_2 + c_2 s_{FM}, \tag{B.1}$$

and from Equation 5.5, one has

$$\theta_M = \kappa_1 s_M + \frac{1}{2} c_1 s_M^2 = \beta_i - \kappa_2 s_{FM} - \frac{1}{2} c_2 s_{FM}^2, \tag{B.2}$$

where θ_M is the angular deviation at the intersection between the two clothoids.

Since, as described in Chapter 5, κ_C denotes the curvature of an equivalent circular arc, the displacements along the axes for this arc are given by

$$dx = \frac{\sin \beta_i}{\kappa_C},$$

$$dy = \frac{1 - \cos \beta_i}{\kappa_C}.$$

From Equations 5.6, one obtains two other equations that need to be satisfied in order to determine the two clothoids:

$$\int_0^{s_M} \cos\left(\kappa_1 u + \frac{1}{2} c_1 u^2\right) du + \int_0^{s_{FM}} \cos\left(\beta_i - \kappa_2 u - \frac{1}{2} c_2 u^2\right) du = \frac{\sin \beta_i}{\kappa_C} \tag{B.3}$$

$$\int_0^{s_M} \sin\left(\kappa_1 u + \frac{1}{2} c_1 u^2\right) du + \int_0^{s_{FM}} \sin\left(\beta_i - \kappa_2 u - \frac{1}{2} c_2 u^2\right) du = \frac{1 - \cos \beta_i}{\kappa_C}. \tag{B.4}$$

Considering that $0 \leq \kappa_1 < \kappa_C$ and $0 \leq \kappa_2 < \kappa_C$, in order to efficiently interpolate the path within the safe zone, the curvature of the two clothoids obviously need to stay positive (i.e., to always turn in the same direction) and exceed (at some point) the curvature κ_C of the equivalent circular arc (i.e., in order to compensate the lower curvature at the beginning and the end of the interpolated path). Since κ_M is the maximum curvature of the two clothoids, we then have $\kappa_M > \kappa_C$.

This also directly implies the following constraints:

$$c_1 > 0, \;\; c_2 > 0,$$

and

$$s_M > 0, \;\; s_{FM} > 0.$$

B.1.1 Existence and Uniqueness of the Solution

Since $s_M > 0$ and $s_{FM} > 0$, from Equations B.1 and B.2, one can express c_1 and c_2 in terms of the other variables:

$$c_1 = \frac{2\beta_i - (\kappa_1 + \kappa_2)\, s_{FM} - 2\kappa 1 s_M}{s_M\,(s_M + s_{FM})} \tag{B.5}$$

$$c_2 = \frac{2\beta_i - (\kappa_1 + \kappa_2)\, s_M - 2\kappa 2 s_{FM}}{s_{FM}\,(s_M + s_{FM})} \tag{B.6}$$

It thus remains to compute s_M and s_{FM} given κ_1, κ_2, κ_C and β_i. This can be done by solving the nonlinear system of Equations B.3 and B.4.

Let us first observe this problem graphically. If $E_x(s_M, s_{FM})$ and $E_y(s_M, s_{FM})$ are defined as the functions:

$$E_x(s_M, s_{FM}) = \int_0^{s_M} \cos\Big(\kappa_1 u + \frac{1}{2}c_1 u^2\Big)\, du + \int_0^{s_{FM}} \cos\Big(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\Big)\, du - \frac{\sin\beta_i}{\kappa_C},$$

$$E_y(s_M, s_{FM}) = \int_0^{s_M} \sin\Big(\kappa_1 u + \frac{1}{2}c_1 u^2\Big)\, du + \int_0^{s_{FM}} \sin\Big(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\Big)\, du - \frac{1 - \cos\beta_i}{\kappa_C},$$

and if we plot on the same graph (as depicted in Figure B.2) the surfaces that represent these two functions, as well as the plane that represents the function $Z(s_M, s_{FM}) = 0$, the solutions of the nonlinear system will then be located at the concurrent intersections of the 3 surfaces.

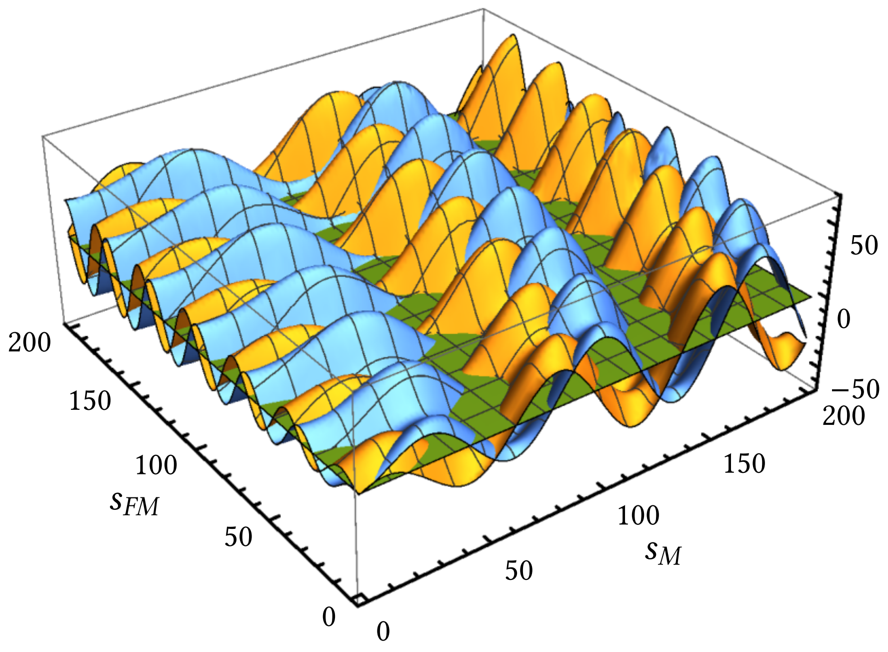


Figure B.2 – ($\beta_i = 1.3$, $\kappa_1 = 0.34$, $\kappa_2 = 0.76$ and $\kappa_C = 1.4$.)
The surface in blue represents the function $E_x(s_M, s_{FM})$, the surface in orange represents the function $E_y(s_M, s_{FM})$ and the plane in green represents the function $Z(s_M, s_{FM}) = 0$.

One can see on Figure B.2 that, due to Fresnel integrals, this system is not easy to solve and yields, in general (i.e., when both κ_1 and κ_2 are not equal to 0), an infinity of solutions. Nevertheless, we can establish that this system has an unique[2] solution which satisfies the constraints $c_1 > 0$ and $c_2 > 0$.

Indeed, recall that we have $\beta_i \in]0, \frac{\pi}{2}]$, $0 \le \kappa_1 < \kappa_C$, $0 \le \kappa_2 < \kappa_C$ and, in order to ensure that we always turn in the same direction, also $c_1 > 0$ and $c_2 > 0$. This gives constraints on s_M and s_{FM}:

- If $\kappa_1 = \kappa_2 = 0$: Then,

$$\begin{aligned} s_M &> 0, \\ s_{FM} &> 0. \end{aligned}$$

[2] All the other solutions (i.e., with $c_1 < 0$ and/or $c_2 < 0$) are composed of a non-zero number of coils turning in one direction and then in the other direction (which is undesirable, inefficient and is not guaranteed to stay in the safe zone).

- If $0 \leq \kappa_1 \leq \kappa_2$ and $\kappa_2 \neq 0$: Then,

$$0 < s_M < \frac{2\beta_i}{\kappa_1 + \kappa_2},$$

$$0 < s_{FM} < \frac{2\beta_i - (\kappa_1 + \kappa_2)\, s_M}{2\kappa_2}.$$

- If $0 \leq \kappa_2 < \kappa_1$: Then,

$$0 < s_M < \frac{\beta_i}{\kappa_1},$$

$$0 < s_{FM} < \frac{2\beta_i - 2\kappa_1 s_M}{\kappa_1 + \kappa_2}.$$

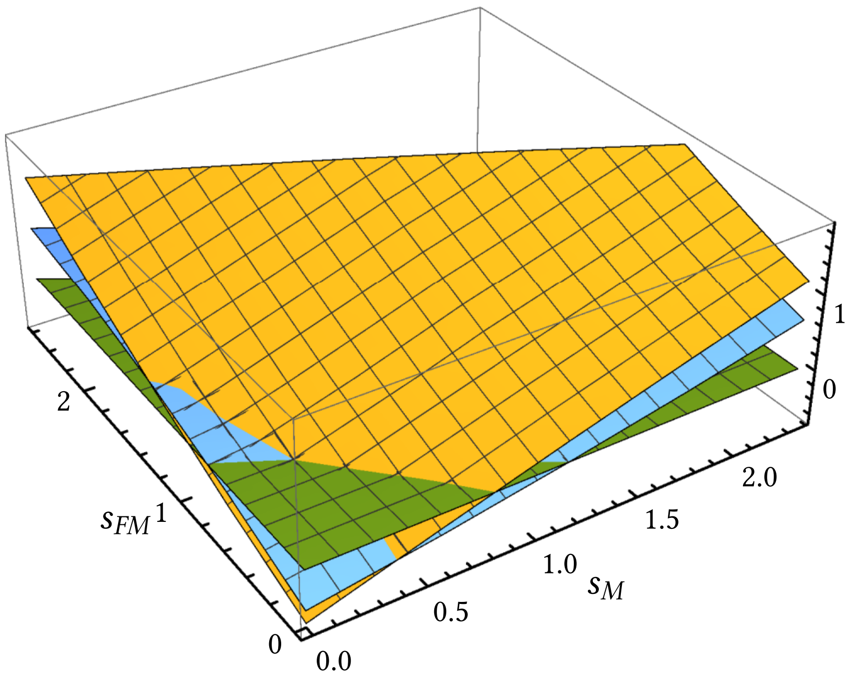


Figure B.3 – ($\beta_i = 1.3$, $\kappa_1 = 0.34$, $\kappa_2 = 0.76$ and $\kappa_C = 1.4$.)
Same surfaces as Figure B.2, but focused on the region of interest where the constraints on s_M and s_{FM} are satisfied.

One can see on Figure B.3 that, when we only plot the region in which constraints on s_M and s_{FM} are satisfied, it admits exactly one point at the intersection between the 3 surfaces and therefore an unique solution to the nonlinear system. We can also see that, in this region, the two surfaces that represents the functions $E_x(s_M, s_{FM})$ and $E_y(s_M, s_{FM})$ are manifolds with small local curvatures. On this basis, we can expect rapid convergence with numerical methods (e.g. Newton-Raphson).

In fact, providing that the constraints on s_M and s_{FM} are satisfied, we can prove that

$$\frac{dc_1}{ds_M} < 0, \ \frac{dc_1}{ds_{FM}} < 0,$$

$$\frac{dc_2}{ds_M} < 0, \ \frac{dc_2}{ds_{FM}} < 0,$$

$$\frac{d\kappa_M}{ds_M} < 0, \ \frac{d\kappa_M}{ds_{FM}} < 0,$$

$$\frac{d\theta_M}{ds_M} > 0, \ \frac{d\theta_M}{ds_{FM}} < 0,$$

which implies

$$\frac{d}{s_M}\left(\int_0^{s_M} \cos\left(\kappa_1 u + \frac{1}{2}c_1 u^2\right) du + \int_0^{s_{FM}} \cos\left(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\right) du\right) > 0, \tag{B.7}$$

$$\frac{d}{s_M}\left(\int_0^{s_M} \sin\left(\kappa_1 u + \frac{1}{2}c_1 u^2\right) du + \int_0^{s_{FM}} \sin\left(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\right) du\right) > 0, \tag{B.8}$$

$$\frac{d}{s_{FM}}\left(\int_0^{s_M} \cos\left(\kappa_1 u + \frac{1}{2}c_1 u^2\right) du + \int_0^{s_{FM}} \cos\left(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\right) du\right) > 0, \tag{B.9}$$

$$\frac{d}{s_{FM}}\left(\int_0^{s_M} \sin\left(\kappa_1 u + \frac{1}{2}c_1 u^2\right) du + \int_0^{s_{FM}} \sin\left(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\right) du\right) > 0. \tag{B.10}$$

These results show that for one value of s_M (resp. s_{FM}) that satisfies the constraints, there is at most one value of s_{FM} (resp. s_M) that also satisfies them and that is a solution of Equation B.3. Similarly, for one value of s_M (resp. s_{FM}) that satisfies the constraints, there is at most one value of s_{FM} (resp. s_M) that also satisfies them and that is a solution of Equation B.4.

Furthermore, since

$$\lim_{s_M \to 0,\ s_{FM} \to 0}\left(\int_0^{s_M} \cos\left(\kappa_1 u + \frac{1}{2}c_1 u^2\right) du + \int_0^{s_{FM}} \cos\left(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\right) du\right) = 0,$$

$$\lim_{s_M \to 0,\ s_{FM} \to 0}\left(\int_0^{s_M} \sin\left(\kappa_1 u + \frac{1}{2}c_1 u^2\right) du + \int_0^{s_{FM}} \sin\left(\beta_i - \kappa_2 u - \frac{1}{2}c_2 u^2\right) du\right) = 0,$$

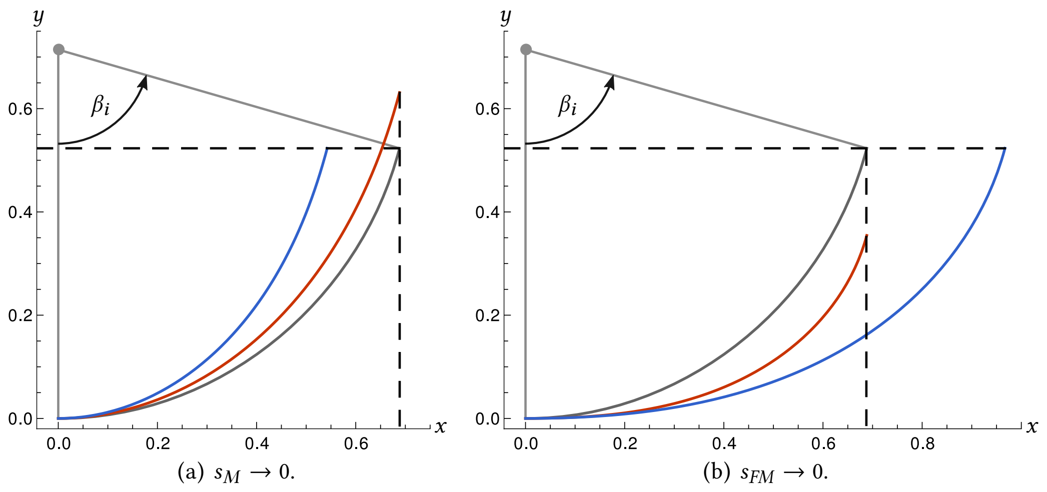


Figure B.4 – ($\beta_i = 1.3$, $\kappa_1 = 0.34$, $\kappa_2 = 0.76$ and $\kappa_C = 1.4$.) In red the clothoid arc that is solution of Equation B.3, and in blue the clothoid arc that is solution of Equation B.4.

and since, whenever s_M (resp. s_{FM}) tends to 0 while s_{FM} (resp. s_M) satisfies the constraints, the x and y projections for the first clothoid arc (resp. the second) are necessarily greater or equal to the ones of the equivalent circular arc, we necessarily have one solution for Equation B.3 and one solution for Equation B.4 when s_M (resp. s_{FM}) tends to 0.

Let s_{FM_x} and s_{FM_y} denote respectively the solution of Equation B.3 and the solution of Equation B.4 when s_M tends to 0. We have

$$s_{FM_x} > s_{FM_y}.$$

Let us show graphically on Figure B.4(a) that this property holds. Indeed, the constraints on the derivatives of κ_M, c_1 and c_2 imply that the two clothoids arcs that solve these two equations stay necessarily inside (i.e., on the left) of the circular arc. These constraints also imply that the red clothoid arc that solves Equation B.3 stays necessarily between the blue clothoid arc that solves Equation B.4 and the circular arc. Therefore, since $\beta_i \leq \frac{\pi}{2}$, the red clothoid arc can only be longer than the one in blue, and we have $s_{FM_x} > s_{FM_y}$.

Symmetrically, let s_{M_x} and s_{M_y} denote respectively the solution of Equation B.3 and the solution of Equation B.4 when s_{FM} tends to 0. As we can check on Figure B.4(b), we have

$$s_{M_x} < s_{M_y}.$$

In addition, from Derivatives B.7, B.8, B.9 and B.10, we show that, for $s_M \in]0, s_{M_x}]$, the solution s_{FM} for Equation B.3 (i.e., in order to reach the x-coordinate) strictly decreases when s_M increases, and we have exactly one value of s_{FM} that is a solution of Equation B.3. Similarly, for $s_{FM} \in]0, s_{FM_x}]$, the solution s_M for Equation B.4 (i.e., in order to reach the y-coordinate) strictly decreases when s_M increases, and we have exactly one value of s_M that is a solution of Equation B.4.

Finally, since (as it can be observed[3] in Figure B.5) the difference between the two curves of solutions strictly decreases when s_M increases from 0 to s_{M_x}, there is an unique solution for the tuple $\{s_M, s_{FM}\}$ that both solves Equations B.3 and B.4 and satisfies the constraints $c_1 > 0$, $c_2 > 0$, $s_M > 0$ and $s_{FM} > 0$, which proves the theorem.

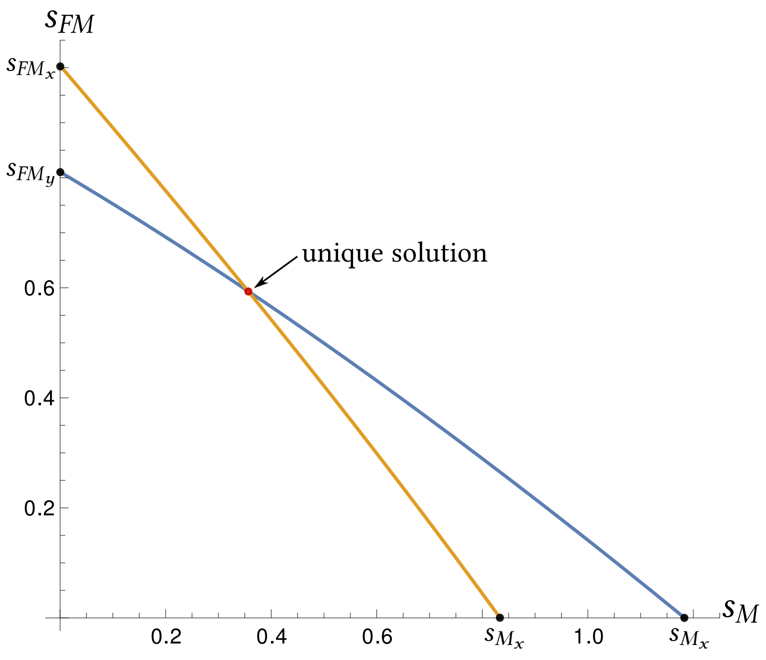


Figure B.5 – ($\beta_i = 1.3$, $\kappa_1 = 0.34$, $\kappa_2 = 0.76$ and $\kappa_C = 1.4$.) Solutions curves for Equations B.3 and B.4.

It is worth noting that we can observe that, the closer κ_1 is to κ_C, the closer s_{FM_x} is to s_{FM_y}, and symmetrically, the closer κ_2 is to κ_C, the closer s_{M_x} is to s_{M_y}. Therefore when both κ_1 and κ_2 are close to κ_C, the two solutions curves are mostly parallel and it is why, as seen in Chapter 5, it is more difficult to converge rapidly to the solution in this situation.

[3] We can also prove graphically that the solution s_{FM} for Equation B.3 (i.e., in order to reach the x-coordinate) necessarily decrease faster when s_M increase than for Equation B.4 (i.e., in order to reach the y-coordinate).

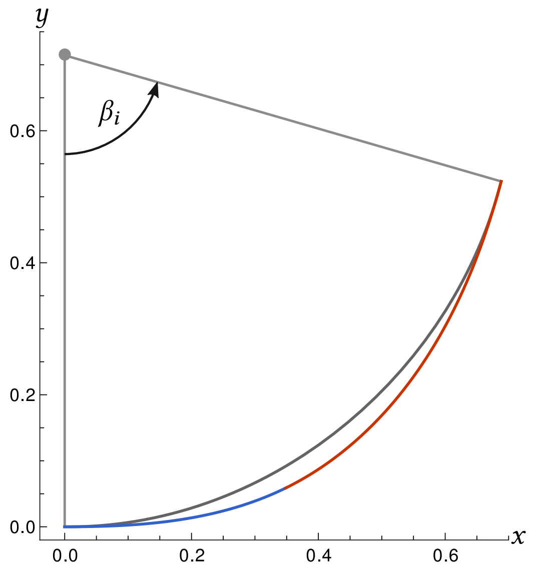


Figure B.6 – ($\beta_i = 1.3$, $\kappa_1 = 0.34$, $\kappa_2 = 0.76$ and $\kappa_C = 1.4$.)
The unique solution for the pair of clothoids that satisfies the constraints.
($s_M = 0.35633376$ and $s_{FM} = 0.59389425$.)
The first clothoid arc is in blue and the second in red.

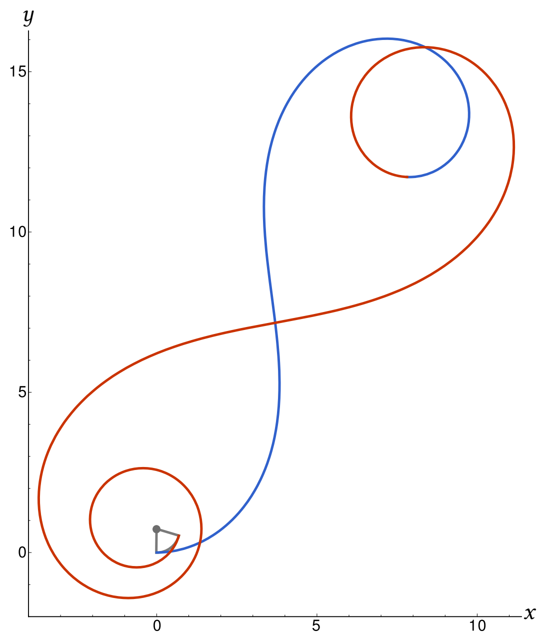


Figure B.7 – ($\beta_i = 1.3$, $\kappa_1 = 0.34$, $\kappa_2 = 0.76$ and $\kappa_C = 1.4$.)
One of the undesirable solutions that do not satisfy the constraints (i.e., $c_1 < 0$ and $c_2 < 0$).
($s_M = 42.699676$ and $s_{FM} = 74.196288$.)
The first clothoid arc is in blue and the second in red.

B.1.2 The Solution Stays in the Safe Zone

Since, the unique solution that we compute is such that $c_1 > 0$ and $c_2 > 0$, the two clothoids arcs stay within the safe zone:

- Since $c_1 > 0$ and $c_2 > 0$, the curvatures of the two clothoids arcs stay positive. Therefore, they do not intersect the two segments. Otherwise, the sign of the curvature would need to change (at some point) in order to reach the destination.

- Since $\beta_i \leq \frac{\pi}{2}$, the clothoid arcs do not intersect the circular arc. Otherwise, the sign of the angular variation, and therefore the sign of the curvature, would need to change (at some point) in order to reach the destination.

Bibliography

[AC05] Aleotti, J., and Caselli, S. 2005. Trajectory clustering and stochastic approximation for robot programming by demonstration. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, IEEE, 1029 – 1034.

[AK00] Aurenhammer, F., and Klein, R. 2000. Voronoi diagrams. *Handbook of Computational Geometry 5*, 201 – 290.

[BK08] Backer, J., and Kirkpatrick, D. 2008. A complete approximation algorithm for shortest bounded-curvature paths. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, Springer-Verlag, 628 – 643.

[BCKO08] de Berg, M., Cheong, O., van Kreveld, M., and Overmars, M. H. 2008. *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag TELOS.

[Ber04] Bern, M. 2004. Triangulations and mesh generation. In *Handbook of Discrete and Computational Geometry*, J. E. Goodman and J. O'Rourke, Eds., 2nd ed. Chapman & Hall/CRC, 563 – 582.

[BG08] Bhattacharya, P., and Gavrilova, M. L. 2008. Roadmap-based path planning-using the Voronoi diagram for a clearance-based shortest path. *Robotics & Automation Magazine, IEEE 15*, 2, 58 – 66.

[BP11a] Brezak, M., and Petrović, I. 2011. Path smoothing using clothoids for differential drive mobile robots. In *18th IFAC World Congress 2011*, 1133 – 1138.

[BP11b] Brezak, M., and Petrović, I. 2011. Time-optimal trajectory planning along predefined path for mobile robots with velocity and acceleration constraints. In *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, IEEE, 942 – 947.

[BLP85] Brooks, R., and Lozano-Perez, T. 1985. A subdivision algorithm in configuration space for findpath with rotation. *Systems, Man and Cybernetics, IEEE Transactions on 15*, 2, 224 – 233.

[Can88] Canny, J. F. 1988. *The Complexity of Robot Motion Planning.* ACM doctoral dissertation awards. MIT Press, Cambridge (Mass.).

[Cha82] Chazelle, B. 1982. A theorem on polygon cutting with applications. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, IEEE, 339 – 349.

[Cha87] Chazelle, B. 1987. Approximation and decomposition of shapes. *Advances in Robotics 1*, 145 – 185.

[Che89] Chew, L. P. 1989. Constrained Delaunay triangulations. *Algorithmica 4*, 97 – 108.

[CCE08] Choi, J.-w., Curry, R., and Elkaim, G. 2008. Path planning based on Bézier curve for autonomous ground vehicles. In *World Congress on Engineering and Computer Science 2008, WCECS'08. Advances in Electrical and Electronics Engineering-IAENG Special Edition of the*, IEEE, 158 – 166.

[CLH$^+$05] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. 2005. *Principles of Robot Motion: Theory, Algorithms, and Implementations.* The MIT Press.

[CMPS93] Cignoni, P., Montani, C., Perego, R., and Scopigno, R. 1993. Parallel 3D Delaunay triangulation. *Computer Graphics Forum 12*, 3, 129 – 142.

[CE07] Connors, J., and Elkaim, G. 2007. Analysis of a spline based, obstacle avoiding path planning algorithm. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, IEEE, 2565 – 2569.

[CC00] Costantinescu, D., and Croft, E. 2000. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of Robotic Systems 17*, 5, 233 – 249.

[Dem07] Demyen, D. 2007. *Efficient triangulation-based pathfinding.* Master's thesis, University of Alberta, Canada.

[DB06] Demyen, D., and Buro, M. 2006. Efficient triangulation-based pathfinding. In *Proceedings of the 21st national conference on Artificial intelligence*, AAAI Press, 942 – 947.

[DPT02] Devillers, O., Pion, S., and Teillaud, M. 2002. Walking in a triangulation. *International Journal of Foundations of Computer Science 13*, 02, 181 – 199.

[Dub57] Dubins, L. E. 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics 79*, 3, 497 – 516.

[ESJ15] Elbanhawi, M., Simic, M., and Jazar, R. N. 2015. Continuous path smoothing for car-like robots using B-spline curves. *Journal of Intelligent & Robotic Systems*, 1 – 34.

[For87] Fortune, S. 1987. A sweepline algorithm for Voronoi diagrams. *Algorithmica 2*, 1, 153 – 174.

[FBT97] Fox, D., Burgard, W., and Thrun, S. 1997. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine 4*, 1, 23 – 33.

[FS04] Fraichard, T., and Scheuer, A. 2004. From Reeds and Shepp's to continuous-curvature paths. *Robotics, IEEE Transactions on 20*, 6, 1025 – 1035.

[Fuk13] Fukushima, T. 2013. Precise and fast computation of Lambert W-functions without transcendental function evaluations. *Journal of Computational and Applied Mathematics 244*, 77 – 89.

[GC02] Ge, S. S., and Cui, Y. J. 2002. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots 13*, 3, 207 – 222.

[Ger10] Geraerts, R. 2010. Planning short paths with clearance using explicit corridors. In *ICRA'10: Proceedings of the IEEE International Conference on Robotics and Automation*, IEEE, 1997 – 2004.

[GO04] Geraerts, R., and Overmars, M. H. 2004. A comparative study of probabilistic roadmap planners. In *Algorithmic Foundations of Robotics V*. Springer, 43 – 58.

[GKS92] Guibas, L. J., Knuth, D. E., and Sharir, M. 1992. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica 7*, 1-6, 381 – 413.

[HS94] Hershberger, J., and Snoeyink, J. 1994. Computing minimum length paths of a given homotopy class. *Computational Geometry 4*, 2, 63 – 97.

[HS99] Hershberger, J., and Suri, S. 1999. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing 28*, 6, 2215 – 2256.

[HIKL+99] Hoff III, K. E., Keyser, J., Lin, M., Manocha, D., and Culver, T. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 277 – 286.

[IKM10] Inkulu, R., Kapoor, S., and Maheshwari, S. 2010. A near optimal algorithm for finding Euclidean shortest path in polygonal domain. *arXiv preprint arXiv:1011.6481.*

[JC93] Jacobs, P., and Canny, J. 1993. Planning smooth paths for mobile robots. In *Nonholonomic Motion Planning*. Springer, 271 – 342.

[JSTL14] Jan, G. E., Sun, C.-C., Tsai, W. C., and Lin, T.-H. 2014. An $O(n \log n)$ shortest path algorithm based on Delaunay triangulation. *Mechatronics, IEEE/ASME Transactions on 19*, 2, 660 – 666.

[Jaz14] Jazar, R. N. 2014. *Vehicle Dynamics: Theory and Application*, 2nd ed. Springer.

[Kal10] Kallmann, M. 2010. Shortest paths with arbitrary clearance from navigation meshes. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, Eurographics Association, 159 – 168.

[Kal14] Kallmann, M. 2014. Dynamic and robust local clearance triangulations. *ACM Transactions on Graphics 33*, 5, 161:1 – 161:17.

[KD85] Kambhampati, S., and Davis, L. S. 1985. Multiresolution path planning for mobile robots. Tech. rep., DTIC Document.

[KM85] Kanayama, Y., and Miyake, N. 1985. Trajectory generation for mobile robots. In *Proc. of the Int. Symp. on Robotics Research*, 16 – 23.

[KH97] Kanayama, Y. J., and Hartman, B. I. 1997. Smooth local-path planning for autonomous vehicles. *The International Journal of Robotics Research 16*, 3, 263 – 284.

[KMM97] Kapoor, S., Maheshwari, S., and Mitchell, J. S. 1997. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete & Computational Geometry 18*, 4, 377 – 383.

[KWP+11] Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., and Teller, S. 2011. Anytime motion planning using the RRT*. In *ICRA'11: Proceedings of the IEEE International Conference on Robotics and Automation*, IEEE, 1478 – 1483.

[KSLO96] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on 12*, 4, 566 – 580.

[Kha86] Khatib, O. 1986. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research 5*, 1, 90 – 98.

[KPA03] Kim, J., Pearce, R. A., and Amato, N. M. 2003. Extracting optimal paths from roadmaps for motion planning. In *ICRA'03: Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, IEEE, 2424 – 2429.

[KB91] Koren, Y., and Borenstein, J. 1991. Potential field methods and their inherent limitations for mobile robot navigation. In *ICRA'91: Proceedings of the IEEE International Conference on Robotics and Automation*, IEEE, 1398 – 1404.

[KS12] Kunz, T., and Stilman, M. 2012. Time-optimal trajectory generation for path following with bounded acceleration and velocity. In *Robotics: Science and Systems*, vol. 8, 09 – 13.

[KKT+09] Kuwata, Y., Karaman, S., Teo, J., Frazzoli, E., How, J. P., and Fiore, G. 2009. Real-time motion planning with applications to autonomous urban driving. *Control Systems Technology, IEEE Transactions on 17*, 5, 1105 – 1118.

[LD04] Lamarche, F., and Donikian, S. 2004. Crowd of virtual humans: A new approach for real time navigation in complex and structured environments. *Computer Graphics Forum 23*, 3, 509 – 518.

[LL98] Lamiraux, F., and Laumond, J.-P. 1998. From paths to trajectories for multi-body mobile robots. In *Experimental Robotics V.* Springer, 301 – 309.

[Lat91] Latombe, J.-C. 1991. *Robot Motion Planning.* Kluwer Academic Publishers.

[LJTM94] Laumond, J.-P., Jacobs, P. E., Taix, M., and Murray, R. M. 1994. A motion planner for nonholonomic mobile robots. *Robotics and Automation, IEEE Transactions on 10*, 5, 577 – 593.

[Lau98] Laumond, J.-P. 1998. *Robot Motion Planning and Control.* Springer.

[LaV98] LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning. Tech. Rep. 98-11, Computer Science Department, Iowa State University.

[LaV06] LaValle, S. M. 2006. *Planning Algorithms.* Cambridge University Press.

[LKJ00] LAVALLE, S. M., AND KUFFNER JR, J. J. 2000. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions.*

[LP84] LEE, D. T., AND PREPARATA, F. P. 1984. Euclidean shortest paths in the presence of rectilinear barriers. *Networks 14*, 3, 393 – 410.

[Len08] LENS, S. 2008. *Locomotion d'un robot mobile.* Master's thesis, Université de Liège.

[LB15a] LENS, S., AND BOIGELOT, B. 2015. Efficient path interpolation and speed profile computation for nonholonomic mobile robots. submitted for publication.

[LB15b] LENS, S., AND BOIGELOT, B. 2015. From constrained Delaunay triangulations to roadmap graphs with arbitrary clearance. submitted for publication.

[MS93] MICAELLI, A., AND SAMSON, C. 1993. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. Tech. rep., INRIA SophiaAntipolis.

[Mie00] MIELENZ, K. D. 2000. Computation of Fresnel integrals II. *Journal of Research of the National Institute of Standards and Technology (NIST) 105*, 4, 589 – 590.

[Mit93] MITCHELL, J. S. 1993. Shortest paths among obstacles in the plane. In *Proceedings of the ninth annual symposium on Computational geometry*, ACM, 308 – 317.

[MGD03] MOSTAFAVI, M. A., GOLD, C., AND DAKOWICZ, M. 2003. Delete and insert operations in Voronoi/Delaunay methods and applications. *Computers & Geosciences 29*, 4, 523 – 530.

[ÓY85] Ó'DÚNLAING, C., AND YAP, C. K. 1985. A "retraction" method for planning the motion of a disc. *Journal of Algorithms 6*, 1, 104 – 111.

[O$^+$92] OVERMARS, M. H., ET AL. 1992. A random approach to motion planning. Tech. rep., Utrecht University, Dept. of Computer Science.

[Pie13] PIERLOT, V. 2013. *Design, performance analysis, and implementation of a positioning system for autonomous mobile robots.* PhD thesis, Université de Liège.

[PV95] POCCHIOLA, M., AND VEGTER, G. 1995. Computing the visibility graph via pseudo-triangulations. In *Proceedings of the eleventh annual symposium on Computational geometry*, ACM, 248 – 257.

[RW98] Reif, J., and Wang, H. 1998. The complexity of the two dimensional curvature-constrained shortest-path problem. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics: the Algorithmic Perspective*, AK Peters, Ltd., 49 – 57.

[Rei79] Reif, J. H. 1979. Complexity of the mover's problem and generalizations. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, IEEE, 421 – 427.

[SF96] Scheuer, A., and Fraichard, T. 1996. Planning continuous-curvature paths for car-like robots. In *IROS 96: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, IEEE, 1304 – 1311.

[SS88] Schwartz, J. T., and Sharir, M. 1988. A survey of motion planning and related geometric algorithms. *Artificial Intelligence 37*, 1, 157 – 169.

[She96] Shewchuk, J. R. 1996. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, M. C. Lin and D. Manocha, Eds., vol. 1148 of *Lecture Notes in Computer Science*. Springer-Verlag, May, 203 – 222. From the First ACM Workshop on Applied Computational Geometry.

[SS90] Shin, D. H., and Singh, S. 1990. Path generation for robot vehicles using composite clothoid segments. Tech. Rep. CMU-RI-TR-90-31, Robotics Institute, Carnegie Mellon University.

[SM85] Shin, K. G., and McKay, N. D. 1985. Minimum-time control of robotic manipulators with geometric path constraints. *Automatic Control, IEEE Transactions on 30*, 6, 531 – 541.

[SK08] Siciliano, B., and Khatib, O. 2008. *Springer Handbook of Robotics.* Springer.

[SSVO09] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. 2009. *Robotics: Modelling, Planning and Control.* Springer.

[Slo93] Sloan, S. 1993. A fast algorithm for generating constrained Delaunay triangulations. *Computers & Structures 47*, 3, 441 – 450.

[SLYD14] Sun, L., Luo, Y., Yu, Y., and Ding, X. 2014. Voronoi diagram generation algorithm based on Delaunay triangulation. *Journal of Software 9*, 3, 777 – 784.

[TMD+06] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., et al. 2006. Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics 23*, 9, 661 – 692.

[UAB+08] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C., et al. 2008. Autonomous driving in urban environments: Boss and the Urban Challenge. *Journal of Field Robotics 25*, 8, 425 – 466.

[VT08] Velenis, E., and Tsiotras, P. 2008. Minimum-time travel for a vehicle with acceleration limits: Theoretical analysis and receding-horizon implementation. *Journal of Optimization Theory and Applications 138*, 2, 275 – 296.

[WW07] Wagner, D., and Willhalm, T. 2007. Speed-up techniques for shortest-path computations. In *STACS 2007*. Springer, 23 – 36.

[WBH05] Wein, R., van den Berg, J. P., and Halperin, D. 2005. The visibility – Voronoi complex and its applications. In *Proceedings of the twenty-first annual symposium on Computational geometry*, ACM, 63 – 72.

[ZT13] Zhao, Y., and Tsiotras, P. 2013. Speed profile optimization for optimal path tracking. In *American Control Conference (ACC), 2013*, IEEE, 1171 – 1176.