DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Path planning for autonomous vehicles using clothoid based smoothing of A* generated paths and optimal control

MARCUS LUNDBERG

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Path planning for autonomous vehicles using clothoid based smoothing of A* generated paths and optimal control

MARCUS LUNDBERG

Degree Projects in Scientific Computing (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2017
Supervisor at at Volvo Construction Equipment: Johan Sjöberg
Supervisor at KTH: Elias Jarlebring
Examiner at KTH: Michael Hanke

TRITA-MAT-E 2017:73
ISRN-KTH/MAT/E--17/73--SE

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Abstract

Autonomous vehicles is a rapidly expanding field and the need for robust and efficient path planners is high. We approach the global path-planning problem for an autonomous load carrier for quarry environments, developed at Volvo Construction Equipment, using two methods: a two-step path planning and path smoothing approach, and a method based on an optimal control formulation of the path planning problem.

The two-step method is based on smoothing an initial path found by A*, an efficient grid search algorithm, by fitting a curve consisting of as few clothoid segments as possible to the A* path. The smoothing is done by rewriting the non-linear optimization problem to a convex form by a linearization of the deviation constraints around the curvature of the A* path. An iterative method is then used to relax the ℓ_0-norm, which measures the number of non-zero elements in a vector, with a weighted ℓ_1-norm which, in turn, is then solved efficiently using CVX in Matlab. The optimal control based path planning method solves the nonlinear optimization problem using IPOPT.

It was found that the completeness of the A* algorithm, coupled with the guaranteed solution of a convex problem, resulted in a very robust method that was able to find paths through mazes and difficult situations. The optimal control approach produced better paths, but had a tendency to sometimes show inconsistent behavior.

Sammanfattning

Vägplanering för autonoma fordon med hjälp av en klotoid-baserad utjämning av A*-genererade vägar och optimal styrteori.

Autonoma fordon är ett snabbt växande forskningsområde och behovet av robusta och effektiva lösningar för vägplanering är stort. Vi undersöker ett globalt vägplaneringsproblem för en autonom dumper som utvecklas på Volvo Construction Equipment och som är byggd för att användas i bergtäkter. Två metoder undersöks: en tvåstegsmetod med grafbaserad vägplanering följt av en utslätning av vägen, och en metod som bygger på optimal styrteori.

Tvåstegsmetoden går ut på att först hitta en väg fri från hinder med hjälp av A*, en effektiv grafbaserad sökalgoritm, och sedan släta ut denna väg genom att anpassa ett så lågt antal klotoid-kurvor som möjligt till vägen. Denna utslätning genomförs genom att skriva om det icke-linjära problemet på konvex form med hjälp av en linjärisering kring referensvägens kurvatur. En iterativ metod beskrivs för att relaxera ℓ_0-normen, som mäter antalet nollskilda element i en vektor, med en viktad ℓ_1-norm. Sedan löses problemet med CVX i Matlab. I metoden baserad på optimal styrteori löser vi det icke-linjära problemet direkt med IPOPT.

Det visade sig att robustheten hos A* tillsammans med ett lättlöst konvext problem gjorde att tvåstegsmetoden blev mycket robust och pålitlig och kunde hitta tillåtna vägar genom labyrinter och svåra situationer. Vägar planerade med hjälp av optimal styrteori var kortare och hade bättre egenskaper, men algoritmen hade en tendens att få svårigheter med konvergens.

Contents

1 Introduction

Research and development on autonomy in vehicles and robots is an area that has gained much popularity in industrial applications over the last three decades [1]. The field has recently seen an increase in usage also in consumer products, such as autonomous driving features in personal cars and drones. The subject is broad and spans many different disciplines. Although there are applications of motion planning within robotics and autonomous vehicles, the subject is also of interest in other areas, such as animation, computer games and computational chemistry [2].

The underlying motivation behind research on autonomous vehicles has been, and still is, to have vehicles independently navigate themselves from some location to a goal, often requiring them to perform one or several tasks along the way. Several general and more problem-specific difficulties have to be overcome. Problems that need to be addressed in virtually all applications of autonomous vehicles are path planning and motion control of the vehicle along the path. The vehicles should be able to calculate driveable paths through their specific type of environment subject to given constraints. Common criteria are to avoid obstacles, static or in some applications moving, while satisfying physical vehicle limitations. Constraints can be imposed on the paths, such as curvature limits, path smoothness, and physical actuator limitations. Often position and orientation constraints for the start and goal positions need to be met. This thesis is concerned with the path planning problem for load carriers operating in a quarry environment.

1.1 Objective

Two methods for non-holonomic path planning will be examined and compared in this thesis; Firstly, a 2-step lattice based planning and smoothing approach. Secondly, path planning based on an optimal control formulation. The output of both methods are driveable paths with specified boundary values, i.e. coordinates and headings for the start and goal positions of the vehicle. A driveable path is one which respects vehicle constraints such as maximum steering angle and physical width limitations. In addition, the path also avoids obstacles and is minimal with respect to path length, i.e. as short as possible.

The lattice based method is split into two steps. First, finding an obstacle free path using a grid based search algorithm. This path is the shortest path in the grid but is not driveable. Due to the grid, each segment of the trajectory is limited to one of 8 directions, i.e. vertical, horizontal, and diagonal. This discrete set of possible headings leads to sharp corners in the path that a physical vehicle cannot follow closely, as shown in figure 1.1.

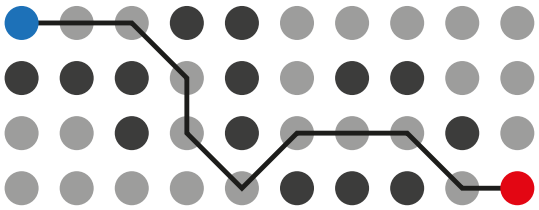

Figure 1.1: Example of an A* solution on a grid. The path starts at the blue node and ends at the red. The black nodes represent obstacles. This path contains sharp corners and needs to be smoothed before a vehicle can follow it.

The obstacle free path through the grid is then smoothed and made driveable. This is done by fitting curves with continuous piecewise linear curvature, called clothoids, to the original path, while respecting deviation constraints. The appealing properties of clothoids make them a popular choice in path planning applications [1, 3, 4].

The second method is to formulate the path planning as a nonlinear optimal control problem. It is solved using a tool developed at Volvo Construction Equipment that is based on the solver IPOPT, a software package for large scale nonlinear optimization written in C++ [5].

The problem can be formulated in terms of time or distance travelled along the path. An advantage of a distance based formulation is that the system can be controlled with only one control parameter, in our case the angular steering velocity. In a formulation based on time, a control parameter for acceleration or velocity of the vehicle must be included. This second parameter comes with the advantage of allowing reversing maneuvers, which could be necessary in some environments. As opposed to a formulation in time, which yields a solution completely describing the vehicles motion along the path, a formulation in distance also requires a velocity planner to calculate a velocity profile for the vehicle to be able to drive along the path. In this thesis, we limit ourselves to the spatial problem of finding obstacle free, driveable paths.

1.2 Introduction to path planning methods

One of the goals pushing the development of path planning algorithms forward is the ambition to have robots and vehicles able to follow high level instructions without detailing all intermediate steps necessary for a completion of a task. The intermediate steps, such as path planning and path following, should be handled autonomously by the robot or vehicle [6]. Current methods used to approach the problem of determining a path that will take a vehicle or robot through some environment, while avoiding obstacles and satisfying imposed constraints, can be grouped according to a set of general properties. The most common characteristics used to classify algorithms are if they are local or global, static or dynamic, whether they are complete and if they are heuristic [7].

In local planning, the environment is not considered known a priori and the vehicle uses sensors to discover new information and plans its path around obstacles it encounters. These planners are often run locally on the vehicle's computer, which sets tough limits on the execution time and complexity of the algorithms since the path has to be updated quickly when an obstacle is found.

Conversely, global planners have access to a complete map of the environment and are not expected to handle unforeseen obstacles or situations. Global planners will of course find better paths through an environment than a local planner can, but a complete map is not always available. Static and dynamic path planning refers to algorithms capable of handling only static obstacles, or a combination of static and moving obstacles, respectively. A path planner is said to be complete if it in finite time either finds a solution or correctly determines that no feasible path exists between the start and goal points. An algorithm is said to be probabilistically complete if the probability of finding a path tends to one as time tends to infinity. Probabilistic methods see more use in path planning for vehicles with many degrees of freedom, as the larger search spaces quickly increases the computational complexity of the problem. The methods examined in this thesis are global and static. Moreover, the lattice-based method is complete. Note that a path consists only of spatial coordinates, time dependence is not involved at all.

Another important property of a path planner is whether it operates in the vehicle's configuration space or in its operational space (i.e. Cartesian space). A major difference between the two alternatives is how the obstacles are represented and checked for collisions. Paths generated within the configuration space need to be checked for intersections with obstacles by integration of the differential equations governing the vehicle's movement. A path in the operational space however can be checked directly if it intersects any obstacles. An advantage of working in the configuration space is that problems with motion singularities can be easier to handle [8].

The first non-holonomic path planning algorithms were purely geometric and did not take obstacles into account [9]. The work was pioneered by Dubins [10], who considered particles moving with constant velocity with a fixed highest curvature constraint. He proved that optimal paths, i.e. the shortest paths that respect non-holonomic constraints consist only out of straight line segments or circular arcs with the maximum allowed curvature, shown in figure 1.2. For a vehicle to follow such a path it would need to stop at the point between two segments and realign its wheels to the new curvature before moving on. Such motions are undesirable in our application and we will describe other methods without this disadvantage.

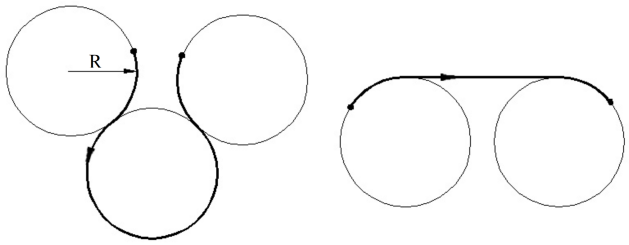

Figure 1.2: Example of Dubin's shortest path between start and goal points for a particle with maximum curvature constraint $\kappa_{max} = 1/R$, [9].

The major families of methods that take obstacles into consideration that have been explored and studied in recent years [1, 3] are:

- Probabilistic Roadmaps (PRM) [2, 8, 11]
- Rapidly-exploring Random Trees (RRT) [12, 13]
- Lattice based motion planners [14, 15]
- Optimal control theory [3]
- Hybrid approaches [16]

In the Probabilistic RoadMap (PRM) approach, the idea is to generate a large, but finite, set of collision free (i.e. allowed) vehicle placements in the operational space (i.e. randomly placing the vehicle on the map) and then connecting these placements with feasible motions [2]. These connections then constitute the roadmap used to solve the path planning problem, figure 1.3 shows an example. The roadmap is constructed iteratively by repeatedly sampling random vehicle placements a predetermined number of times. Provided that the newly sampled placement passes a collision check with known obstacles, it is added to the roadmap. Then the algorithm attempts to connect the placement to previously generated placements using a local planner if they are closer than some threshold distance, often measured in Euclidean distance. These connections are usually made with straight lines through the operational space, but only if they also pass a collision test with known obstacles. This test is often done by uniformly sampling points along the line at a sufficiently high resolution and then checking that all of these points are allowed. If no collisions are detected, the new point and its connection to one or several of the previously generated nearby points are added to the roadmap. If the line between the points is not collision free, the connection is discarded and the placement is added to the roadmap without any connection to other placements. After a roadmap has been created, which only has to be done once, the path planning problem has been reduced to finding a shortest path through a graph with local collision free paths, which is solved efficiently. If no path is found, more iterations can be run to expand the roadmap [8].

One alternative to creating a full roadmap before the path planning step is when the interest is in fast single-query path planning, i.e. when a path is needed between only a given start and goal state. The rest of the map will not be used, and is not needed for future planning. The concept of Rapidly-exploring Random Trees, which by means of some heuristic, samples and expands the graph between the start and goal points. A bidirectional RRT method is investigated in [17], in which two separate random trees are expanded at the endpoints, directing the generation of new nodes toward the other end of the path. With this design, the sampling of points in the free configuration space that are not likely to be traversed, is much reduced.

The sampling based methods, i.e. PRM, RRT and lattice based motion planners, have all been shown to be effective in large configuration spaces and at coping with obstacles. The use of efficient graph search algorithms such as A*, ARA* and D*Lite [18] make lattice based motion planners a good choice. However, sampling based methods suffer in accuracy because of their discrete

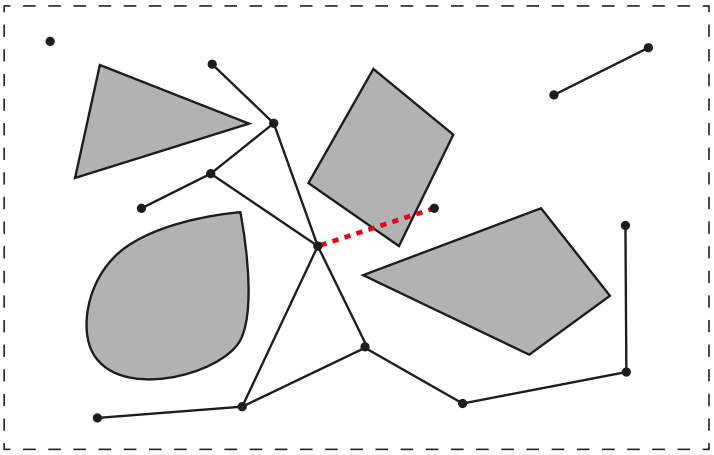

Figure 1.3: Example of a probabilistic roadmap where the black dots are sampled placements around the grey obstacles. The black lines between the dots are feasible motions. The dashed red line shows a connection that does not pass a collision test.

nature. This includes inaccuracies and discontinuities along the path or at the start and end positions and headings, compared to the goal values. This introduces uncertainties which prevents the necessary accuracy for use in many industrial applications [3]. Another trade-off is that sampling based methods are not complete. Instead they will find solutions with any probability, given a sufficient amount of time to run.

A lattice based approach with motion primitives is based on solving a path finding problem through a tree with edges representing possible maneuvers. A set of predefined motion primitives, e.g. based on clothoid curves [19] are used to discretely define the motions the vehicle can perform. Including backward motions in the set of allowed motions means that complex situations that require reversing can be handled. The generated path will be driveable but often suffers from unnecessary swaying since the motion primitives were selected from a discrete set and are unable to align perfectly to the optimal path.

Combinations of the methods listed above are also common. For instance, a hybrid approach was used in [16] for the path planning in Junior, shown in figure 1.4, an autonomous vehicle developed at Stanford that participated in the *DARPA urban challenge*, 2007. The path planner was based on Field D*, an interpolation based grid search algorithm [20] that instead of using only the midpoint in each grid cell as in the A* algorithm, allows arbitrary positions within the cells using linear interpolation. Furthermore, the approach implemented in Junior includes an extra step in each iteration where it checks whether or not there exists an allowed control signal that would make the vehicle drive to the next grid-point. This makes sure that the final path is driveable.

Figure 1.4: Junior, Stanford's contribution to the DARPA Urban Challenge 2007, a 96 kilometers long course through an urban environment [21]. Image courtesy: Stanford News.

Other hybrid planners have been developed that work through a combination of online and offline planning. The global planner generally works with low-resolution (or sparse) maps without information about small obstacles, such as rocks and trees etc. A combination of the online and offline components is then used to find the optimal paths.

1.3 Autonomous vehicles

The use of autonomous vehicles in industrial applications is extensive and their role in production and in other areas is predicted to increase in the future [1, 22]. Today, common users of autonomous vehicles and autonomous robots are storage and production facilities, where replacing manual transportation and sorting of items increases pipeline speeds and production throughput [1].

The industry standard today is for mobile robots and vehicles to follow predefined paths between the start and goal locations [3]. The environment in such facilities is very controlled and the paths are often programmed manually or recorded from a human operated vehicle. For vehicles to operate in less controlled environments, such as outdoors, in cities, or on construction sites, this approach has several drawbacks. It is often time consuming to manually define paths and the lack of flexibility can be costly. If the environment is altered, such as obstacles moved to other locations, new paths for the vehicle will have to be created manually. In the case of an obstacle blocking the vehicle's path, only very simple strategies can be used. Often the strategy is to just stop and wait until the obstacle has been moved [3]. Extra care must be taken to plan feasible paths with respect to vehicle constraints and obstacle avoidance. Algorithms for planning paths must take into consideration several factors which vary with the vehicle type. A short path length, short travel time and low energy consumption are common objectives. A smoothness requirement is also common for vehicles meant for transporting goods or passengers [6]. The smoothness of the path is enforced by penalizing derivatives of the path since they, in order of successive differentiation, correspond to velocity, acceleration and jerk.

The use of self-driving vehicles brings with it a number of advantages compared to human operated vehicles [1, 23] and are already being developed and

produced for production, construction, and urban environments [23]. Notably, Tesla car models and other cars with autonomous driving features are becoming more widespread [1]. A big selling point for autonomous cars for civilians is the increase in safety with computer aided driving. It is well established that the human factor plays a major role in accidents [24]. A computer processes inputs from several sensors that detect obstacles and dangers much faster than a human, which gives the autonomous vehicle more time to e.g. apply breaks or perform an avoiding maneuver if it detects an impending accident.

The short reaction time allows for reduced safety margins and vehicles can drive faster and closer to each other, also operating in conditions unsuitable for human operators [1]. These factors are advantageous in many industrial applications as well, and not only for civilian cars. Moreover, as construction sites move further out from cities and populated areas, autonomous vehicles enable companies to operate on bigger scales since the number of personnel needed to run the operations is lower [1].

Pre-planned paths chosen as short as possible, or with as low or as smooth control inputs as possible etc., result in reduced fuel consumption and driving time. This in turn enables the vehicles to meet more restrictive environmental standards and release less greenhouse gases [1]. In fully autonomous vehicles, the lack of a drivers compartment and tools necessary for a human operator, further reduces the weight of the vehicles, letting them operate more efficiently.

1.4 Levels of autonomy

As more autonomous and automated vehicles and robots are introduced, the laws around this area is developing quickly. There are different levels of autonomy, where more or less of the driving and control of the vehicle is handled by the machine and not by a human operator. To distinguish between these levels SAE International, a global association with technical experts in the aerospace, automotive and commercial-vehicle industries, has defined six levels ranging from ”no automation” to ”full automation” [25].

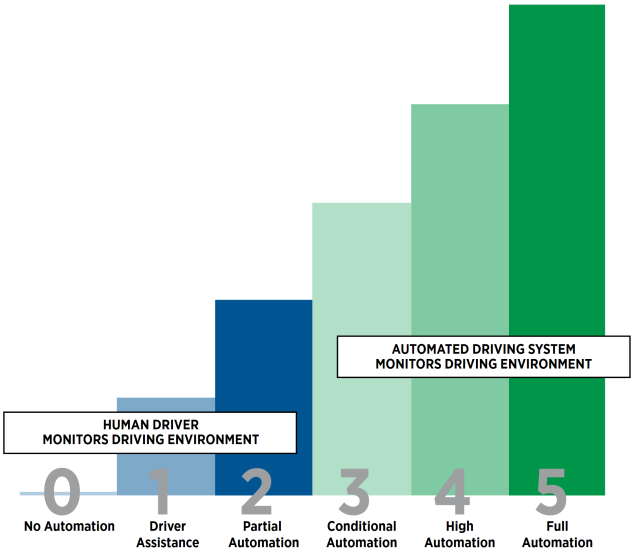

Figure 1.5: Levels of automation defined by SAE International [25]

The levels are defined as follows:

Level 0 The vehicle may issue warnings but is completely controlled by the human driver.

Level 1 The vehicle has limited control over one aspect of the driving at a time, for instance steering, cruise control acceleration/braking or parking assistance. It is expected that a human driver is present and ready to take control over the vehicle at any time.

Level 2 The vehicle can handle several aspects of the control at the same time using input from the environment, for instance having control over steering, acceleration and braking. The vehicle must be monitored by the driver and he or she should be able to step in and take over at any time.

Level 3 The vehicle controls all aspects of the dynamic driving and the human driver is not obliged to keep full attention on what the vehicle is currently doing, with the exception that the human operator will respond appropriately to requests to take over within some specified time limit.

Level 4 The vehicle operates on its own and should be able to handle situations even if a human driver does not respond to requests. In such cases the vehicle can abort its current action and for instance park and wait for assistance. The self-driving of Level 4 vehicles is only allowed in certain situations, such as traffic jams or within limited areas.

Level 5 The vehicle is fully self-driving and performs self-controlled driving under all circumstances and conditions. No human intervention at all is expected.

1.5 Electric Site

Volvo Construction Equipment is running a research project aiming at introducing electric and hybrid vehicles to reduce carbon emissions in the quarry and aggregates industry. The prototype machines consist of a hybrid wheel loader, a grid connected excavator and the fully autonomous electric load carrier HX1, shown in figure 1.7. By powering the machines with electricity generated from renewable and environmentally friendly sources, such as hydropower and wind power, it is estimated that the total carbon emission can be reduced by 95 % and total cost of ownership can be reduced by 25 % compared to traditional fossil fuel based machines operating in quarries today [26].

Figure 1.6: An overview of vehicle operation at a quarry [26]. The HX1 vehicles are loading material from the crusher (left) to be transported up the hill and dumped. Image courtesy of Volvo Construction Equipment.

During the investigation of possible improvements of quarry operation, it was noted that the transportation of aggregate done by a few large haulers comes with potential draw-backs. In case of a problem with any of the haul trucks the entire operation suffers. To reduce the dependence on a few vehicles, thereby reducing risks, Volvo Construction Equipment introduced the HX1, which is smaller than the haulers used today. By controlling a fleet of HX1 vehicles, the same amount of aggregate can be transported with a reduced impact on the quarry production resulting from a potential failure in one vehicle [26]. The HX1 is the vehicle that is meant to use the paths calculated by the two methods we examine in this thesis and is shown in figure 1.7.

Figure 1.7: A fully loaded autonomous load carrier HX1 developed at Volvo Construction Equipment. It has a four-wheel drive with steering on both the front and back wheel pairs. A design feature of the HX1 is that it can position itself close behind another HX1 and overlap it with the truck bed due to the lack of a driver's compartment. They are then loaded continuously by the crusher by driving slowly forward under the outlet which removes the need for intermediary loading by another vehicle, further reducing fuel costs. Image courtesy of Volvo Construction Equipment.

2 Related work

The clothoid smoothing algorithm in this thesis is based on work done in [1]. The authors propose a clothoid-based path sparsification algorithm and a clothoid-based model predictive controller for use in fully autonomous trucks in mining applications. The trucks are supposed to be able to drive long distances and, with high-resolution path data, there exists a need to reduce the number of data points without sacrificing accuracy. A parametric description of the path is therefore appealing since if one is available, a path of any resolution can be obtained with a comparably small number of path parameters. In addition to providing smooth traveling, clothoids are well suited to on-road path sparsification since many roads are designed using clothoid segments.

The ℓ_0-norm relaxation used in this thesis is based on results from [27, 28], where a method for iteratively updating weights in a relaxed ℓ_1-formulation is derived. The problem would require a computationally infeasible combinatorial search if it was not relaxed in some manner.

Path planning based on optimal control in an environment containing obstacles is performed in [3]. First, a driveable path is found using a lattice based search with motion primitives. The resulting path is then used to constrain the optimization problem with linear constraints chosen such that they do not overlap any obstacle. The obstacle free path is then used as initial path and is smoothed by the solver, based on a time-formulation of the problem.

3 Theory

3.1 Kinematic vehicle model

The construction of the robot or autonomous vehicle is important in the calculation of its optimal path. Robots operating over a two-dimensional environment (the xy-plane) often have three degrees of freedom. Translation along the x and y axes, and rotation about the z axis, orthogonal to the plane. This hypothetical robot can rotate on the spot and has no limits on how sharp turns it can make. For car-like vehicles however, there are restrictions on how they can maneuver in the environment, a consequence of their steering constraints [9]. Such a vehicle has two controls, the linear and angular velocity, and the fact that the number of controls is less than the number of configuration variables makes the equations non-integrable and the car-like vehicle is called non-holonomic. This adds an extra layer of complexity to the path-planning, since even if the vehicles are controllable, they are not free to perform any arbitrary motion at any given time.

To model the vehicle, we will use a kinematic bicycle model [1,29]. The maximum speed of the HX1 is approximately 40 km/h and the driving speed when fully loaded is lower than that. This lets us use this comparably simple model by assuming that lateral dynamics have little influence. A dynamical model is more appropriate for higher speeds where lateral dynamics have a bigger impact. We describe the vehicle by only considering its kinematic equations. In this model, the rear wheel is locked in the direction of the vehicle, while the front wheel is rotated by a steering angle ϕ. Since the HX1 is four-wheel steered, we cannot directly apply the model. To do that, we consider driving with symmetric and opposite steering angles for the front and rear wheels. It can be shown that under this condition, the vehicle can be modelled with a bicycle model where the rear wheel in the model is at the center of the front and rear wheel axles of the original vehicle. This means that we can use the bicycle model for the HX1 provided we use half the length of the vehicle, L, as the length of the vehicle in the model.

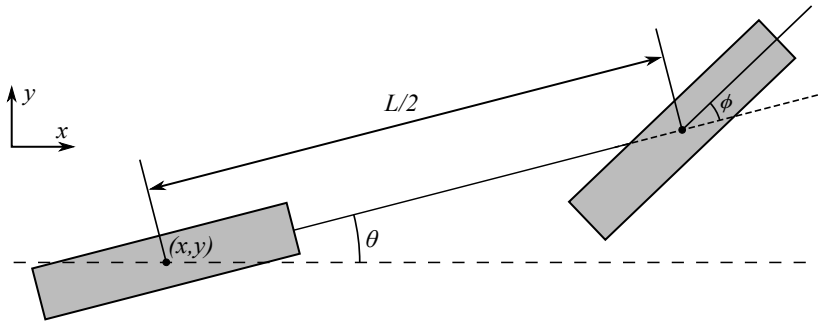

Figure 3.1: Bicycle model.

The kinematic equations derived in [30] for a symmetrically steered vehicle are

$$\begin{aligned}\frac{dx(t)}{dt} &= v(t)\cos(\theta(t)) \\ \frac{dy(t)}{dt} &= v(t)\sin(\theta(t)) \\ \frac{d\theta(t)}{dt} &= \frac{2}{L}v(t)\tan(\phi(t)),\end{aligned} \tag{3.1}$$

where x and y are the x and y-coordinates of the rear wheel in the model, i.e. the center of the HX1, in the coordinate system of the ground. Furthermore, θ is the yaw angle and L is the distance between the front and rear axles of the HX1. These equations can be rephrased in terms of the distance parameter s along the path. We have $v(t) \cdot dt = ds$, which if we assume $v(t)$ is a continuous function and $v(t) \neq 0$, lets us rewrite the kinematic equations as

$$\begin{aligned}\frac{dx(s)}{ds} &= \cos(\theta(s)), \\ \frac{dy(t)}{ds} &= \sin(\theta(s)), \\ \frac{d\theta(s)}{ds} &= \frac{2}{L}\tan(\phi(s)).\end{aligned} \tag{3.2}$$

From (3.2) we can calculate the maximum curvature κ_{max} that the path is allowed to have, by inserting the maximum steering angle ϕ_{max} of the HX1.

3.2 Clothoids

A clothoid curve, also known as Euler spiral or Cornu spiral, has a curvature κ that varies linearly over the arc length parameter s, i.e.

$$\kappa(s) = cs \ + \ \kappa_0. \tag{3.3}$$

The linear curvature is the reason why clothoids are well suited to describe vehicle paths. It corresponds to turning the steering wheel with a constant angular velocity. The curvature is the derivative of the tangent angle and the set of equations defining a clothoid curve is [31]

$$\begin{aligned}x(s) &= x_0 + \int_0^s \cos\big(\theta(t)\big)dt, \\ y(s) &= y_0 + \int_0^s \sin\big(\theta(t)\big)dt, \\ \theta(s) &= \theta_0 + \int_0^s \kappa(t)dt,\end{aligned} \tag{3.4}$$

which, after insertion of a linear $\kappa(s)$, are known as Fresnel integrals. An example of a clothoid is shown in figure 3.2.

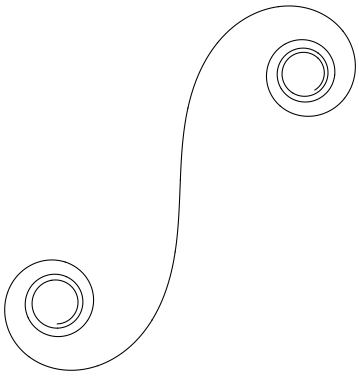

Figure 3.2: One clothoid segment. If we consider curvature as a signed quantity, the clothoid forms a double spiral with odd symmetry [32].

The fact that this curve has several names (i.e. clothoid curve, Euler spiral and Cornu spirals) is a result of the different areas in which it was discovered and used independently over the years, in completely different fields [32]. The first to discover the clothoid curve was Jacob Bernoulli around 1694 in his work related to elasticity. The curve was a solution to the problem:

> *Which function defines the shape of a pre-curved spring such that when the free end is pulled down by a weight, the spring straightens into a line?*

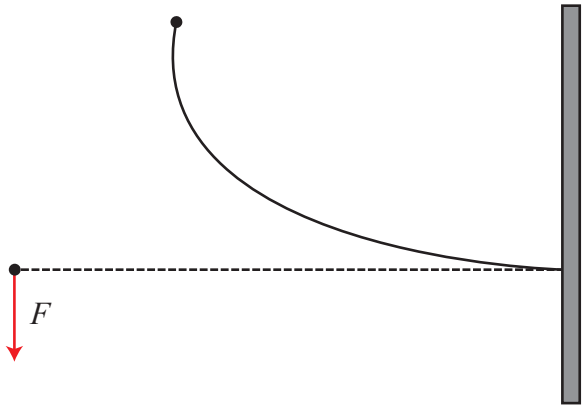

Figure 3.3: A pre-curved spring mounted on a wall. After application of a force F to the end, the spring straightens into a line. This problem was later investigated and solved by Leonhard Euler in the middle of the eighteenth century [32].

The next person to discover the curve was Augustin Fresnel around 1818 when he investigated a problem of light diffraction and derived the well-known Fresnel integrals. The spirals defined by the Fresnel integrals were accurately

plotted by Alfred Marie Cornu in 1874, unaware of Euler's earlier investigation of the curve. The Fresnel integrals were derived once again by Arthur Talbot in 1890, this time in the context of transitional curves for railway tracks designed for a smooth riding experience. A more thorough history of the applications of the clothoid curve can be found in [32].

Today, clothoids are widely used as transitional curves between straight line segments and circular arcs, and between circular arcs of radii $R_1 \neq R_2$, in road and railway path planning [4]. Transitions based on clothoids are appealing since the smooth change in curvature makes for a gradual change in the centrifugal force experienced by the vehicle and the passengers. In addition to reducing the discomfort of the passengers, the gradual change in centrifugal force significantly reduces the risk of accidents [33]. Curves consisting of segments of straight lines, circular arcs and clothoid curves are called clothoid splines. This is due to the fact that straight lines and circular arcs can be seen as limiting cases of clothoid curves. For the straight line, $c = 0$ and $\kappa_0 = 0$ in (3.3), and for the circular arcs, $c = 0$ and $\kappa_0 \neq 0$ is a constant. Clothoids have historically been a popular choice in curves and has also been investigated as transition curves for vertical highway alignments.

3.3 Continuous clothoid optimization problem

We want to fit a curve $(x(s), y(s))$, consisting of as few clothoid segments as possible, to a reference path $(x_{ref}(s), y_{ref}(s))$. The curvature function $\kappa(s)$ that solves this problem will be a continuous piece-wise linear function over the interval $[p_0, p_n]$, with $n-1$ discontinuities in $\kappa'(s)$ at $p_1, \ldots, p_{n-1}$. Let

$$\kappa(s) = \begin{cases} \kappa_1(s), & s \in [p_0, p_1) \\ \quad \vdots & \\ \kappa_i(s), & s \in [p_{i-1}, p_i) \\ \quad \vdots & \\ \kappa_n(s), & s \in [p_{n-1}, p_n]. \end{cases} \tag{3.5}$$

We use the fact that the second derivatives of linear functions are 0 to construct the optimization problem

$$\begin{aligned} \underset{\kappa(s),\ n\in\mathbb{N}}{\text{minimize}} \quad & n \\ \text{subject to} \quad & x'(s) = \cos(\theta(s)) \\ & y'(s) = \sin(\theta(s)) \\ & \theta'(s) = \kappa(s) \\ & \kappa_i''(s) = 0,\ s \in (p_{i-1}, p_i), \quad i = 1, \ldots, n \\ & |x(s) - x_{ref}(s)| < \epsilon \\ & |y(s) - y_{ref}(s)| < \epsilon \\ & x(0) = x_{ref}(0),\ x(p_n) = x_{ref}(p_n) \\ & y(0) = y_{ref}(0),\ y(p_n) = y_{ref}(p_n) \\ & \theta(0) = \theta_{ref}(0),\ \theta(p_n) = \theta_{ref}(p_n), \end{aligned} \tag{3.6}$$

where $\epsilon \in \mathbb{R}$ is the maximum deviation allowed from the reference path. The boundary constraints make sure the smoothed path starts and ends at the same coordinates, with the same heading angles, as the reference path.

The formulation (3.6) is a difficult problem since it is non-linear and contains the discrete optimization variable n. It can be seen as a combination of two problems, a partitioning problem and a curve fitting problem. In the method section, we will discretize this formulation and recast it as an optimization problem without any integer valued variables. The technique is used in [1] and relies on a relaxation of the ℓ_0 norm (which counts the number of non-zero elements in a vector) using the ℓ_1 norm, resulting in a convex relaxation of the original problem [27]. We will then linearize the nonlinear differential constraints and obtain a convex optimization problem as a result. The problem is then solved using CVX, a modeling system for convex optimization problems implemented in Matlab.

3.4 A* grid search

An initial path for the smoothing algorithm is calculated using A*, which is an informed graph-search algorithm. A* is an extension of Dijkstra's algorithm and uses a heuristic to guide the path expansion toward the target node by looking at the total length of a path from the start node to the end node, passing through the currently investigated node. As is described in [34], which first introduced the A* algorithm in 1968, there are usually two approaches to graph traversing problems which they call the *mathematical approach* and the *heuristic approach*. The mathematical approach is less concerned with the computational feasibility of the algorithms and instead focuses more on the ultimate goal, the finding of solutions to the problem. It typically deals with algorithms that perform an ordered examination of nodes in a graph to find minimum cost paths. These algorithms are guaranteed to find the shortest path, provided one exists.

Opposed to this orderly examination of nodes until the minimum cost path is found, the heuristic approach incorporates problem specific knowledge to guide the search expansion toward the goal to reduce computational costs. The A* algorithm combines the two approaches, a heuristic guiding the search, while still keeping the property of guaranteeing the path found to be a minimum cost solution. During traversal of the graph, the A* algorithm follows the path with the currently known lowest cost while keeping a sorted priority queue of alternate path segments that have been passed. If a path traversed by the algorithm has a higher cost than another previously explored path segment, it continues the search from the lower cost segment instead until the goal node is reached.

We discretize the map of the environment into a rectangular grid and restrict the search to only horizontal, vertical and diagonal directions, resulting in all angles between the points being multiples of $\pi/4$. We select a start and goal node and mark obstacles as black. Obstacle nodes are excluded from the search and, if a path is found, it is guaranteed to be obstacle free. However, the path found by the A* algorithm is not driveable due to the sharp angles and corners, and has to be smoothed to be used by a vehicle. One important result is that the A* algorithm is guaranteed to find *a* shortest path between two nodes, if a path exists. Due to the regular structure of the grid there may be several paths with the same total length, as shown in figure 3.4. The algorithm stops as soon

as the goal node is reached.

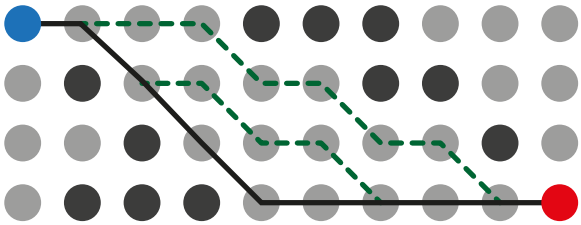

Figure 3.4: The dashed green lines show alternative paths with the same length as the solid black path between the blue start node and red goal node. Black nodes represent obstacles.

Algorithm 3.1: Pseudocode for the A* algorithm [35]

```
1  Function Main()
2      open = ∅
3      closed = ∅
4      g[s_start] = 0
5      parent[s_start] = s_start
6      open.Insert(s_start, g[s_start] + h(s_start))
7      while open ≠ ∅ do
8          s = open.Pop()
9          if s = s_goal then
10             return "path found"
11         closed = closed ∪ s
12         foreach s' ∈ neighbours(s) do
13             if s' ∉ closed then
14                 if s' ∉ open then
15                     g[s'] = ∞
16                     parent[s'] = NULL
17                 updateVertex(s, s')
18     return "No path found"
19 Function updateVertex(s,s')
20     if g[s] + c(s, s') < g[s'] then
21         g[s'] = g[s] + c(s, s')
22         parent[s'] = s
23         if s' ∈ open then
24             open.Remove(s')
25         open.Insert(s', g[s'] + h(s'))
```

Three values are stored for each node s:

- g is a list with values for all the nodes where the lowest cost so far of reaching that node is saved.

- *parent* is a list in which information of the parent nodes upstream along paths will be saved.
- $h(s)$ is the user-provided and problem specific heuristic function that is an estimate of the shortest possible distance between the node s and s_{goal}. The f-value $f(s) = g(s) + h(s)$ is the estimate of the length of the shortest path from the start node to the goal node that passes through the node s.

Two global data structures are also needed:

- *open* is a priority queue in which the nodes that the A* algorithm can consider for expansion are stored. $open.Insert(s, v)$ inserts node s with the key v into the correct position in the queue determined by ascending sorting of the keys v. $open.Remove(s)$ removes node s from *open* and $open.Pop()$ removes and returns from *open* the element with the smallest key v.
- *closed* is a set of nodes that has already been expanded. This list is checked to make sure that a node is not expanded more than once.

The A* algorithm is run by calling the $Main()$ function in 3.1 which begins by defining the *open* and *closed* sets. The program is initialized by adding the start node s_{start} to *open* with the initial f-value as its key. Inside the while loop, the node with the lowest key in the *open* queue is extracted and checked whether or not it is the goal node, in which case the algorithm is done and terminates. Otherwise, the current node s is added to the closed set and, for each neighbor of the current node, which is not already in the closed set, we check if it has been encountered before. If it has not, the node is initialized with $g[s'] = \infty$ and parent node set to $NULL$.

For each of the neighbors of s, the function $updateVertex(s, s')$ examines whether the current g-value of the neighbor s' is greater than the g-value of the current node s plus the distance $c(s, s')$ between the nodes s and s'. $c(s, s')$ is the Euclidean distance between the two nodes. If the sum $g[s] + c(s, s')$ is smaller, it means that the path through the current node s is shorter than the previous value saved for s', and $g[s']$ and $parent[s']$ are updated according to this shorter path. This procedure is then repeated until either *open* is empty, which means that no solution was found, or $s = s_{goal}$. The total path can then be traced back through the *parents* list.

Since the A* algorithm was first introduced, several improved algorithms have been suggested, for instance D*, Theta*, Field D* etc. D* is an implementation that is able to continually improve on the search and incorporate new data, for instance new obstacles, into the solutions without having to completely restart. This improvement has a big impact for a local online path planning algorithm in which new obstacles may be discovered. In our global path planning algorithm, D* would not improve our results. Field D* is an extension of D* in which the path is no longer bound to the grid-points, instead it uses linear interpolation within each grid cell to produce smoother and shorter paths [20]. Theta* is an extension of A* in which the parent of a node is allowed to be any node, while in A*, the parent is restricted to be a neighboring node, see figure 3.5. By skipping parent nodes further up in the chain that lie between nodes with a clear line of sight, a shorter and less jagged path is found [36].

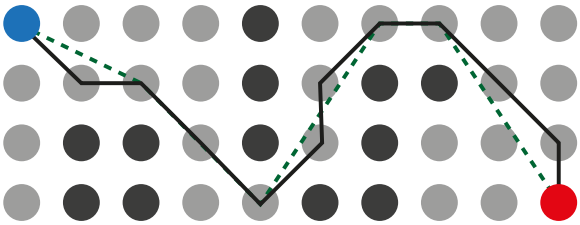

Figure 3.5: The black line is the path found by the A* algorithm. The dashed green line shows the smoother, and shorter, solution found by the Theta* algorithm.

3.5 Optimal control theory approach

In an optimal control problem, the goal is to find a control law for a system described by differential equations such that it results in the system taking an optimal path between a given start and final state. The state of the system can be affected by control parameters that are allowed to vary within some bounds. An objective function is used to measure how good the control law is, for example by measuring the total cost of the used control variable. The goal is to choose the control variables such that the objective function is minimized.

In our case, the state of the vehicle is described by the non-linear kinematic equations (3.2), which we will solve using IPOPT. To reduce the non-linearity of the problem we introduce the change of variables

$$\psi(s) = \tan(\phi(s)), \tag{3.7a}$$
$$|\psi(s)| \leq \psi_{max} = |\tan(\phi_{max})|. \tag{3.7b}$$

The control variable is chosen as the derivative of $\psi(s)$, denoted by $u(s) = \psi'(s)$, to allow us to introduce maximum limits on the angular velocity of the steering wheels.

There are several choices of how to represent obstacles. As opposed to the clothoid smoothing problem, where we have constrained areas for the (x, y) coordinates to stay within, we now intend to constrain the path to remain outside obstacles. This area is not convex and can therefore not be described by normal linear constraints. One way to describe the non-convex obstacle is with a conjunction of linear constraints, by introducing binary constraints, as was done in [37]. Instead, we approximate P obstacles with ellipses which the vehicle has to navigate around. These ellipses are defined by their center coordinates, (c_j^x, c_j^y), and the parameters a_j and b_j which describe their major and minor axes. An advantage of this formulation is that the constraints will be twice continuously differentiable, which is important for IPOPT.

We therefore seek to solve the following minimization problem, with state variable $\mathbf{x}(s) = (x(s), y(s), \theta(s), \psi(s))^T$, and control variable $u(s)$,

$$
\begin{aligned}
\underset{u(s),\ s_f}{\text{minimize}} \quad & Qs_f + \int_0^{s_f} u(s)^2 ds \\
\text{subject to} \quad & \frac{d}{ds}\begin{bmatrix} x(s) \\ y(s) \\ \theta(s) \\ \psi(s) \end{bmatrix} = \begin{bmatrix} \cos(\theta(s)) \\ \sin(\theta(s)) \\ (2/L)\psi(s) \\ u(s) \end{bmatrix} = \mathbf{f}(\mathbf{x}, s) \\
& \frac{(x(s) - c_j^x)^2}{a_j^2} + \frac{(y(s) - c_j^y)^2}{b_j^2} > 1, \quad j = 1, \ldots, P \\
& \mathbf{x}(0) = (x_0, y_0, \theta_0, \psi_0)^T,\ \mathbf{x}(s_f) = (x_f, y_f, \theta_f, \psi_f)^T \\
& x_{min} \leq x(s) \leq x_{max} \\
& y_{min} \leq y(s) \leq y_{max} \\
& |\psi(s)| \leq \psi_{max} \\
& |u(s)| \leq u_{max} \\
& s_f > 0,
\end{aligned}
\tag{3.8}
$$

where s_f is the length of the path, $Q \geq 0$ is a parameter for tuning the importance of the terms in the objective function, $\mathbf{x}(0)$ and $\mathbf{x}(s_f)$ are the start and goal states (i.e. coordinates, headings, and steering angles). The parameter s_f is the total path length, which we want to minimize. The values of ψ_{max} and u_{max}, the maximum steering angle and maximum angular velocity, respectively, are physical limits of the HX1 and are necessary to make the resulting path driveable.

Q in 3.8 controls the influence of the penalty on the control variable. Solving the optimization problem with a large value on Q reduces the impact of the integral in the objective function. This can lead to solutions where the angular acceleration of the steering wheel is very high, i.e. the angular speed changes instantaneously, which is difficult to achieve in a physical vehicle. To prevent this, we take into account the smoothness of $u(s)$ by penalizing it in the objective function.

3.6 IPOPT

IPOPT is an open source software package for large scale non-linear optimization, written in C++. It is based on an interior point line search filter method aiming to find a local solution to a general non-linear programming problem with linear or nonlinear constraints that are at least twice continuously differentiable [38]. To implement IPOPT we use software developed at Volvo Construction Equipment. The continuous problem formulation with constraints and cost function is specified in a Maple program, which discretizes the problem and outputs C++ code implementing the IPOPT solver. After the source code is compiled it is run via Matlab, where parameter values and variable bounds are specified.

4 Method

4.1 A* implementation

Algorithm 3.1 is implemented in Matlab with an approach to select obstacles based on obstacle center nodes. The vehicle's physical width has to be considered when selecting the obstacle nodes in the grid. Since the shortest path chosen by the A* algorithm will pass close to obstacles, extra margins need to be added to the obstacle boundaries to let the vehicle pass while driving centered on the path, as shown in figure 4.1.

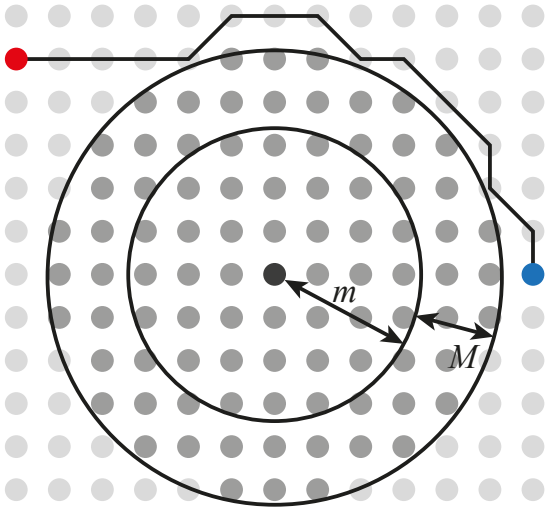

Figure 4.1: Selection of obstacles nodes around a grid center node (black node in the middle). The physical obstacle is contained inside the inner circle with radius m. A margin M is added to push the vehicle center further out.

Let $\mathcal{N}$ denote the set of obstacle center nodes and let $\mathcal{M}$ denote the set of total obstacle nodes that will be present in the A* algorithm. We initially choose center nodes and give them a radius value. We use circles of a specified radius around obstacle center nodes since it will assist in limiting the maximum curvature of the path. If we, for instance, were to select obstacles in a rectangle with sharp corners, the resulting path would be more difficult to smooth and make driveable due to the sharp corners.

Obstacles are selected by marking a number of nodes as obstacle center nodes and adding them to $\mathcal{N}$. We then add a margin M to all the specified radii to the obstacle center nodes. Then, iterating over all obstacle center nodes over and all grid-points in the grid, we check whether or not each grid-point lies within the circle around a node in $\mathcal{N}$. If it does, the grid-point is added to $\mathcal{M}$.

To approximately choose start and goal headings of the path we impose virtual obstacles around the start and end nodes which restrict the possible headings the path can have there. These virtual obstacles are shown in 4.2, which allows the heading at the end nodes to be specified in a range of around $\pm 45^o$ from the desired heading.

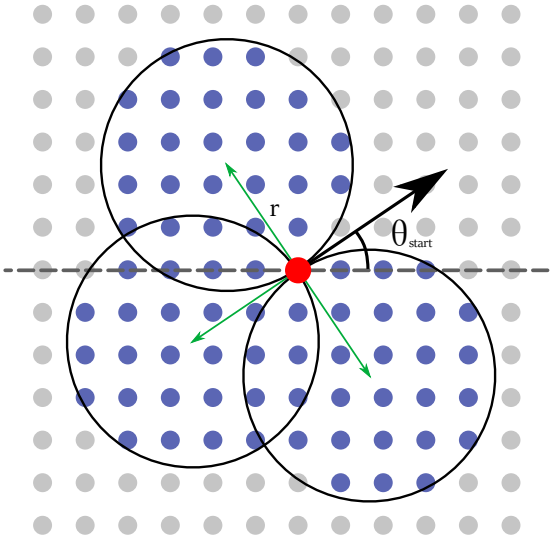

Figure 4.2: Selection of heading obstacles is done by placing circles of radius r with relative center points orthogonal to the start heading vector and one circle behind the red start point. Grid-points within the circles are marked as obstacles and are added to $\mathcal{M}$. The parameter r controls the distance of the circles and limits the start heading and curvature of the path.

Before applying the smoothing algorithm to the path, we adjust the coordinates of the second and the second last nodes to make the heading at the start and goal nodes exactly as wanted.

4.2 Clothoid smoothing implementation

To rewrite the optimization problem (3.6) in a discretized form we start by applying an Euler forward discretization to the differential constraints, as shown in (4.1). The path, of total length L is discretized into $N+1$ points s_i. $\mathbf{x}_i = (x(s_i), y(s_i))$, $i = 0, \ldots, N$. There are N sub-intervals of length $h_i = s_{i+1} - s_i$.

$$\begin{aligned} x_{i+1} &= x_i + h_i \cos\left(\theta_i\right) \\ y_{i+1} &= y_i + h_i \sin\left(\theta_i\right) \\ \theta_{i+1} &= \theta_i + h_i \kappa_i. \end{aligned} \tag{4.1}$$

To write the discretization in matrix-vector notation we define the following quantities, which are all vectors in $\mathbb{R}^{N+1}$,

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}, \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_{N-1} \\ \theta_N \end{bmatrix}, \\ \boldsymbol{\kappa} = \begin{bmatrix} \kappa_0 \\ \kappa_1 \\ \vdots \\ \kappa_{N-1} \\ \kappa_N \end{bmatrix}, \quad \mathbf{h} = \begin{bmatrix} h_0 \\ h_1 \\ \vdots \\ h_{N-1} \\ 0 \end{bmatrix}. \tag{4.2}$$

We also need the following $(N+1) \times (N+1)$ matrix

$$I_L = \begin{bmatrix} 0 & & & \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix}. \tag{4.3}$$

Using (4.1) we can write

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_i \\ \vdots \\ x_N \end{bmatrix} = \underbrace{\begin{bmatrix} x_0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{x}_0} + \begin{bmatrix} 0 \\ x_0 + h_0 \cos(\theta_0) \\ \vdots \\ x_{i-1} + h_{i-1} \cos(\theta_{i-1}) \\ \vdots \\ x_{N-1} + h_{N-1} \cos(\theta_{N-1}) \end{bmatrix} = \mathbf{x}_0 + I_L \mathbf{x} + I_L \big(\mathrm{diag}(\mathbf{h}) \cos(\boldsymbol{\theta})\big), \tag{4.4}$$

where $\mathrm{diag}(\mathbf{h})$ is a square zero matrix with $\mathbf{h}$ as diagonal, and the cosine function is applied element-wise. Solving this equation for $\mathbf{x}$, and doing a similar derivation for $\mathbf{y}$, we get

$$\begin{aligned} \mathbf{x} &= (I - I_L)^{-1} \big[\mathbf{x}_0 + I_L \mathrm{diag}(\mathbf{h}) \cos(\boldsymbol{\theta})\big] \\ \mathbf{y} &= (I - I_L)^{-1} \big[\mathbf{y}_0 + I_L \mathrm{diag}(\mathbf{h}) \sin(\boldsymbol{\theta})\big], \end{aligned} \tag{4.5}$$

where I is the $(N+1) \times (N+1)$ identity matrix. The next step is a similar derivation for $\boldsymbol{\theta}$.

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_i \\ \vdots \\ \theta_N \end{bmatrix} = \underbrace{\begin{bmatrix} \theta_0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\boldsymbol{\theta}_0} + \begin{bmatrix} 0 \\ \theta_0 + h_0 \kappa_0 \\ \vdots \\ \theta_{i-1} + h_{i-1} \kappa_{i-1} \\ \vdots \\ \theta_{N-1} + \kappa_{N-1} \end{bmatrix} = \boldsymbol{\theta}_0 + I_L \boldsymbol{\theta} + I_L \big(\mathrm{diag}(\mathbf{h}) \boldsymbol{\kappa}\big). \tag{4.6}$$

Solving for $\boldsymbol{\theta}$ gives

$$\boldsymbol{\theta} = (I - I_L)^{-1}\big[\boldsymbol{\theta}_0 + \text{diag}(\mathbf{h})\boldsymbol{\kappa}\big]. \tag{4.7}$$

Combining equations (4.5) and (4.7) we get the coordinates expressed in curvature as

$$\begin{aligned}
\mathbf{x} &= (I - I_L)^{-1}\left[\mathbf{x}_0 + I_L \text{diag}(\mathbf{h}) \cos\left((I - I_L)^{-1}\big[\boldsymbol{\theta}_0 + \text{diag}(\mathbf{h})\boldsymbol{\kappa}\big]\right)\right], \\
\mathbf{y} &= (I - I_L)^{-1}\left[\mathbf{y}_0 + I_L \text{diag}(\mathbf{h}) \sin\left((I - I_L)^{-1}\big[\boldsymbol{\theta}_0 + \text{diag}(\mathbf{h})\boldsymbol{\kappa}\big]\right)\right].
\end{aligned} \tag{4.8}$$

We can simplify this expression somewhat by setting

$$\begin{aligned}
A &\stackrel{\text{def}}{=} (I - I_L)^{-1}, \\
H &\stackrel{\text{def}}{=} \text{diag}(\mathbf{h}),
\end{aligned} \tag{4.9}$$

which yields

$$\begin{aligned}
\mathbf{x}(\boldsymbol{\kappa}) &= A\left[\mathbf{x}_0 + I_L H \cos\left(A\big[\boldsymbol{\theta}_0 + H\boldsymbol{\kappa}\big]\right)\right], \\
\mathbf{y}(\boldsymbol{\kappa}) &= A\left[\mathbf{y}_0 + I_L H \sin\left(A\big[\boldsymbol{\theta}_0 + H\boldsymbol{\kappa}\big]\right)\right].
\end{aligned} \tag{4.10}$$

The reference path $\mathbf{x}_{ref}$, given by the A* algorithm is used to constrain the smoothed path. The coordinates of the smoothed path must lie within a distance $\varepsilon_k,\ k = 0, \ldots, N$ from the reference path. We also want the first and last point to be the same as for the reference path and we want them to have the same heading. We can lock in the same heading as the reference path by setting ε_k very small for $k = 0, 1, N-1, N$.

$$\begin{aligned}
|\mathbf{x}^i(\kappa) - \mathbf{x}^i_{ref}| < \varepsilon_i, \quad i = 0, \ldots, N \\
|\mathbf{y}^i(\kappa) - \mathbf{y}^i_{ref}| < \varepsilon_i, \quad i = 0, \ldots, N
\end{aligned} \tag{4.11}$$

4.2.1 Linearization of path around reference curvature

The equations (4.10) that will be used as constraints are clearly non-linear which makes the problem non-convex. We therefore linearize them around a reference curvature $\hat{\boldsymbol{\kappa}}$ that will be calculated from the reference path. A general linearization of a multivariable function $\mathbf{f}(\mathbf{x})$ around a point $\mathbf{x}_0$ can be written

$$\mathbf{f}(\mathbf{x}) \approx \mathbf{f}(\mathbf{x}_0) + J(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0), \tag{4.12}$$

where $J(\mathbf{x}_0)$ is the Jacobian matrix evaluated in the point $\mathbf{x}_0$. The Jacobian matrix is defined component-wise as

$$J_{ij} = \frac{\partial f_i}{\partial x_j}. \tag{4.13}$$

Let

$$\begin{aligned}
\mathbf{f}(\mathbf{x}) &= \sin(B\mathbf{x} + \mathbf{c}), \\
\mathbf{g}(\mathbf{x}) &= \cos(B\mathbf{x} + \mathbf{c}),
\end{aligned} \tag{4.14}$$

where $B \in \mathbb{R}^{M\times M}$ and $\mathbf{c}, \mathbf{x} \in \mathbb{R}^M$ and the sine and cosine functions are taken element-wise. We will find the Jacobian for $\mathbf{f}(\mathbf{x})$ and then apply the result to (4.10). Write B as a column vector of row vectors,

$$B = \begin{bmatrix} — \ \mathbf{b}_1 \ — \\ \vdots \\ — \ \mathbf{b}_i \ — \\ \vdots \\ — \ \mathbf{b}_M \ — \end{bmatrix}, \tag{4.15}$$

which lets us write

$$\mathbf{f}(\mathbf{x}) = \sin(B\mathbf{x} + \mathbf{c}) = \begin{bmatrix} \sin(\mathbf{b}_1 \cdot \mathbf{x} + c_1) \\ \vdots \\ \sin(\mathbf{b}_i \cdot \mathbf{x} + c_i) \\ \vdots \\ \sin(\mathbf{b}_M \cdot \mathbf{x} + c_M) \end{bmatrix}. \tag{4.16}$$

By the same method, let

$$\begin{aligned} \big[\mathbf{f}(\mathbf{x})\big]_i &= f_i = \sin(\mathbf{b}_i \cdot \mathbf{x} + c_i), \\ \big[\mathbf{g}(\mathbf{x})\big]_i &= g_i = \cos(\mathbf{b}_i \cdot \mathbf{x} + c_i). \end{aligned} \tag{4.17}$$

Taking the derivative of f_i with respect to x_j gives

$$\frac{\partial f_i}{\partial x_j} = \cos(\mathbf{b}_i \cdot \mathbf{x} + c_i) b_{ij} = g_i b_{ij}, \tag{4.18}$$

where b_{ij} is the jth component of $\mathbf{b}_i$. Now we can write, with $\Delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$ and $\odot$ denoting element-wise multiplication,

$$\begin{aligned} J_f(\mathbf{x}_0)\Delta\mathbf{x} &= \begin{bmatrix} g_1(\mathbf{x}_0)b_{11} & \cdots & g_1(\mathbf{x}_0)b_{1M} \\ \vdots & \ddots & \vdots \\ g_M(\mathbf{x}_0)b_{M1} & \cdots & g_M(\mathbf{x}_0)b_{MM} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_M \end{bmatrix} \\ &= \begin{bmatrix} g_1(\mathbf{x}_0)\big(b_{11}\Delta x_1 + \ldots + b_{1M}\Delta x_M\big) \\ \vdots \\ g_M(\mathbf{x}_0)\big(b_{M1}\Delta x_1 + \ldots + b_{MM}\Delta x_M\big) \end{bmatrix} \\ &= \begin{bmatrix} g_1(\mathbf{x}_0) \\ \vdots \\ g_M(\mathbf{x}_0) \end{bmatrix} \odot \begin{bmatrix} \mathbf{b}_1 \cdot \Delta\mathbf{x} \\ \vdots \\ \mathbf{b}_M \cdot \Delta\mathbf{x} \end{bmatrix} = \mathbf{g}(\mathbf{x}_0) \odot \big(B\Delta\mathbf{x}\big) \\ &= \cos(B\mathbf{x}_0 + \mathbf{c}) \odot \big(B(\mathbf{x} - \mathbf{x}_0)\big). \end{aligned} \tag{4.19}$$

Similarly,

$$J_g(\mathbf{x}_0)\Delta\mathbf{x} = -\sin(B\mathbf{x}_0 + \mathbf{c}) \odot \big(B(\mathbf{x} - \mathbf{x}_0)\big). \tag{4.20}$$

Now we can use (4.20) and (4.19) to linearize (4.10) by identifying that $B = AH$, $\mathbf{c} = A\boldsymbol{\theta}_0$, and $N + 1 = M$,

$$\begin{aligned}\mathbf{x}_L(\boldsymbol{\kappa}) &= A\left[\mathbf{x}_0 + I_L H\left[\cos(B\hat{\boldsymbol{\kappa}}+\mathbf{c}) - \sin(B\hat{\boldsymbol{\kappa}}+\mathbf{c}) \odot \big(B(\boldsymbol{\kappa}-\hat{\boldsymbol{\kappa}})\big)\right]\right],\\ \mathbf{y}_L(\boldsymbol{\kappa}) &= A\left[\mathbf{y}_0 + I_L H\left[\sin(B\hat{\boldsymbol{\kappa}}+\mathbf{c}) + \cos(B\hat{\boldsymbol{\kappa}}+\mathbf{c}) \odot \big(B(\boldsymbol{\kappa}-\hat{\boldsymbol{\kappa}})\big)\right]\right].\end{aligned} \tag{4.21}$$

4.2.2 Calculation of reference curvature

Given a reference path, we can calculate its curvature using the discretization in (4.1). Rearranging the equations, we get

$$\begin{aligned} x_{i+1} - x_i &= h_i \cos\big(\theta_i\big),\\ y_{i+1} - y_i &= h_i \sin\big(\theta_i\big),\\ \theta_{i+1} &= \theta_i + h_i\kappa_i. \end{aligned} \tag{4.22}$$

Dividing the second equation by the first in (4.22) we can solve for θ_i

$$\begin{aligned} \theta_i &= \arctan\left(\frac{y_{i+1}-y_i}{x_{i+1}-x_i}\right),\\ \theta_{i+1} &= \theta_i + h_i\kappa_i. \end{aligned} \tag{4.23}$$

We can then combine the equations in (4.23) by adjusting the indices in the second equation, and get, for $i = 1, \ldots, N$,

$$\hat{\kappa}_{i-1} = \frac{1}{h_{i-1}}\left[\arctan\left(\frac{y_{i+1}-y_i}{x_{i+1}-x_i}\right) - \theta_{i-1}\right]. \tag{4.24}$$

Using (4.23) and (4.24) we can solve for $\hat{\boldsymbol{\kappa}} = [\hat{\kappa}_0, \hat{\kappa}_1, \ldots, \hat{\kappa}_{N-1}, \hat{\kappa}_N]^{\mathrm{T}}$ iteratively, since θ_0 is known. We set $\hat{\kappa}_N = \hat{\kappa}_{N-1}$, but this value has no impact on the path.

4.2.3 ℓ_0-optimization formulation

The coordinates where one clothoid segment ends and the next begins are called kink-points. We want a continuous piece-wise linear curvature, and in each kink-point the derivative of the curvature changes. We want to smooth our path by representing it by as few clothoid segments as possible and that is equivalent to minimizing the number of kink-points. We can therefore rewrite the objective function in problem (3.6) as follows. Assume that we have a continuous, piece-wise linear curvature $\boldsymbol{\kappa}$. By multiplying with a second order difference matrix

$$D = \begin{bmatrix} 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{bmatrix} \in \mathbb{R}^{(N-1)\times(N+1)}, \tag{4.25}$$

we get non-zero elements only at the kink-points. The optimization problem that we need to solve is therefore

$$\begin{aligned} \underset{\boldsymbol{\kappa}}{\text{minimize}} \quad & \|D\boldsymbol{\kappa}\|_0 \\ \text{subject to} \quad & |\mathbf{x}_L^i(\boldsymbol{\kappa}) - \mathbf{x}_{ref}^i| < \varepsilon_i, \quad i = 0, \ldots, N \\ & |\mathbf{y}_L^i(\boldsymbol{\kappa}) - \mathbf{y}_{ref}^i| < \varepsilon_i, \quad i = 0, \ldots, N \\ & |\boldsymbol{\kappa}^i| < \kappa_{\max}, \quad i = 0, \ldots, N, \end{aligned} \tag{4.26}$$

where $\|\cdot\|_0$ denotes the ℓ_0-norm defined as

$$\|\boldsymbol{\kappa}\|_0 = \sum_{i=0}^{N} \mathbb{I}(\boldsymbol{\kappa}^i \neq 0), \tag{4.27}$$

where $\mathbb{I}$ is an indicator function. This is a sparse problem and most of the elements of the solution $\boldsymbol{\kappa}^\star$ are zero. But this formulation is of little practical use since the optimization problem (4.26) is NP-hard. An intractable combinatorial search has to be performed because of the binary variable complexities introduced by the ℓ_0-norm [1, 27]. In the following sections, we will explore a method for relaxing the ℓ_0-norm and recast 4.26 into a convex linear program that can be solved efficiently.

4.2.4 MM-algorithms

A majorize-minimization (MM) algorithm will be used to substitute the difficult nonconvex ℓ_0-norm for the continuous and convex ℓ_1-norm. The idea of the MM algorithm is to use a simpler majorizing function and run the optimization on it instead. As is discussed in [39], a scalar function $g(\mathbf{x}|\mathbf{x}_m)$, where $\mathbf{x}, \mathbf{x}_m \in \mathbb{R}^n$, is a majorizer of the function $f(\mathbf{x})$ at the point $\mathbf{x}_m$ if and only if

$$f(\mathbf{x}_m) = g(\mathbf{x}_m|\mathbf{x}_m), \tag{4.28a}$$

$$f(\mathbf{x}) \leq g(\mathbf{x}|\mathbf{x}_m), \quad \mathbf{x} \neq \mathbf{x}_m. \tag{4.28b}$$

$\mathbf{x}_m$ denotes the current iterate in the search after the surface $f(\mathbf{x})$. Intuitively, the surface $g(\mathbf{x}|\mathbf{x}_m)$ is always above the surface $f(\mathbf{x})$ except at the point $\mathbf{x} = \mathbf{x}_m$ where $f(\mathbf{x})$ and $g(\mathbf{x}|\mathbf{x}_m)$ are tangent to each other. Figure 4.3 shows a one-dimensional example.

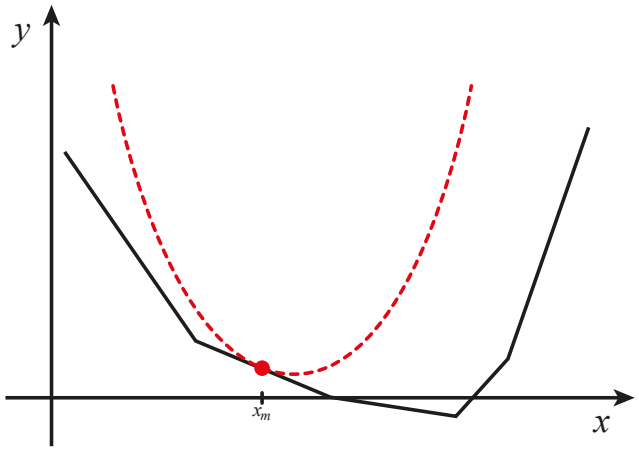

Figure 4.3: The function plotted with a dashed red line $g(\mathbf{x}|\mathbf{x}_m)$ is a majorizer to a continuous piecewise linear black curve $f(\mathbf{x})$.

By minimizing the majorizer, the underlying function is also forced down. Let $\mathbf{x}_{m+1}$ denote the minimum of the function $g(\mathbf{x}|\mathbf{x}_m)$. Then,

$$f(\mathbf{x}_{m+1}) \leq g(\mathbf{x}_{m+1}|\mathbf{x}_m) \leq g(\mathbf{x}_m|\mathbf{x}_m) = f(\mathbf{x}_m), \tag{4.29}$$

where the first inequality follows from (4.28b), the second inequality follows from the fact that $\mathbf{x}_{m+1}$ is a minimum of $g(\mathbf{x}, \mathbf{x}_m)$, and the equality follows from (4.28a). This iterative method guarantees that $f(\mathbf{x}_m)$ converges to a local optimum or a saddle point as $m \to \infty$.

4.2.5 Relaxation of ℓ_0-norm for sparse problems

Consider a minimization problem on the general form

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \|\mathbf{x}\|_0 \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \end{aligned} \tag{4.30}$$

with $A \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^{\mathbf{m}}$ defining linear equality constraints. As it is stated now, this is a difficult problem since it has to be solved by a combinatorial search. A commonly used method to solve problems similar to (4.30) is to relax them by using the ℓ_1-norm [40], defined as

$$\|\mathbf{x}\|_1 = \sum_{i=1}^{n} |x_i|, \tag{4.31}$$

resulting in the problem

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} |x_i| \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}. \end{aligned} \tag{4.32}$$

With this relaxation, the problem (4.30) can be recast as a linear program for which efficient methods exist to solve it. This relaxation has been used successfully in many applications and the method dates back many decades. An early application where it saw use was reflection seismology where the goal was to separate different subsurface layers of the earth using band-limited data [41]. However, as the ℓ_1-norm takes magnitude into account as opposed to the ℓ_0-norm, they cannot be expected to yield exactly the same solutions. Theoretical investigations have been made to analyze and find conditions for when the optimal solution to (4.32) is also the optimal solution to (4.30). For instance, in [42], the problem of reconstructing an input vector from corrupted measurements using ℓ_1-minimization was investigated. It was found that, under suitable conditions of the coding matrix used to encode the input vector, and provided that the percentage of corrupted elements in this matrix was less than a specified constant, the input vector could be recovered as the solution of a ℓ_1-minimization problem.

However, as is discussed in [40], the theoretical results are often of limited value since they are difficult to use to guarantee or verify optimality in practical implementations. Therefore, the solution to the ℓ_1-relaxed problem is in general suboptimal to (4.30).

To remedy this problem an iterative algorithm for a reweighted ℓ_1 minimization was proposed in [27],

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} w_i|x_i| \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \end{aligned} \tag{4.33}$$

where $w_i \geq 0$, $i = 1, \ldots, n$. This weighted optimization problem can still be recast as a linear program. If we collect all the weights in the vector $\mathbf{w} \in \mathbb{R}^{\mathbf{n}}$ the problem (4.33) can then be thought of as the relaxation of the weighted problem

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \|\mathbf{w} \odot \mathbf{x}\|_{\mathbf{0}} \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \end{aligned} \tag{4.34}$$

where $\odot$ denotes element-wise multiplication. It is noted in [27] that if there exists is a unique solution to (4.30), and, as long as the weights do not vanish, that solution is also the unique solution to (4.34). However, that property is not preserved between the original and weighted ℓ_1 relaxed problems, (4.32) and (4.33), as they generally have different solutions. This enables us to consider the weights w_i as free parameters, and the question now is how to choose the weights.

Suppose we had the true solution $\mathbf{x}_0$ to the problem (4.30) which is k-sparse, i.e., the number of non-zero elements are less than or equal to k, and $k \leq m$. If we then choose the weights to 'normalize' the magnitude of the elements in $\mathbf{x}_0$, i.e.,

$$w_i = \begin{cases} \frac{1}{|x_{0,i}|}, & x_{0,i} \neq 0, \\ \infty, & x_{0,i} = 0, \end{cases} \tag{4.35}$$

then (4.33) is guaranteed to have the correct solution, $\mathbf{x}_0$ [27]. This is due to the very large weights put on all elements that are supposed to have the value 0

and the comparably much smaller weights put on the elements that should have non-zero values. By the construction of the weights, these elements correspond exactly to the true solution. However, the exact solution is generally not known in beforehand and some appropriate algorithm for choosing the weights has to be found. What this example suggests though, is that larger weights could be used to push down non-zero elements while smaller weights could be used to 'allow' non-zero elements.

The algorithm proposed in [27] is

Algorithm 4.1: Proposed algorithm for choosing the weights w_i.

$w_i^{(0)} = 1,\ i = 1, \ldots, n;$
$m = 0;$
while *Not converged* **do**
  $\mathbf{x}^{(m)} = \arg\min \|\mathbf{w}^{(\mathbf{m})} \odot \mathbf{x}\|_{\mathbf{1}} \quad \text{s.t.} \quad A\mathbf{x} = \mathbf{b};$
  $w_i^{(m+1)} = \frac{1}{|x_i^{(m)}| + \xi},\ ,\ i = 1, \ldots, n;$
  $m = m + 1;$

The parameter ξ has been introduced to provide stability and also, more importantly, to not restrict the possibility of a non-zero element in the next iteration by giving it an infinite weight. This is important since it is not known from the beginning which elements should or should not be zero-valued.

We will now try to justify Algorithm 4.1 analytically. The method is based on a choice of a continuous function that will more closely resemble the ℓ_0 norm than the ℓ_1-norm does. There are several choices of this function, but we use the function used in [27].

$$f(t) = \alpha \ln(1 + |t|/\xi), \tag{4.36}$$

where $\xi > 0$ is a parameter and $\alpha = 1/\ln(1 + 1/\xi)$. The choice of α is a result from the condition $f(1) = 1$. Figure 4.4 shows the connection between these functions.

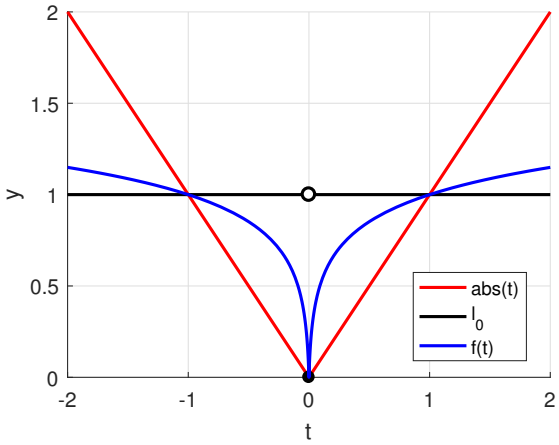

Figure 4.4: Plot of the functions that approximate the ℓ_0-norm. $f(t)$, here plotted with $\xi = 0.01$, is clearly a better approximation to the ℓ_0-norm near the origin than the ℓ_1-norm is. As $\xi \to 0$, $f(t)$ more and more closely resembles the ℓ_0-norm.

To find the iterative method we first begin by relaxing the problem (4.30) using the function $f(t)$ in (4.36) instead of with the ℓ_1-norm

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} \ln(1 + |x_i|/\xi) \\ \text{s.t.} \quad & A\mathbf{x} = \mathbf{b}, \end{aligned} \tag{4.37}$$

where we have omitted the constant α since it does not affect the minimizer $\mathbf{x}^\star$ of the problem.

Lemma 4.1. *Let $A \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$, and let $h : \mathbb{R} \to \mathbb{R}$ be a strictly increasing function over $(0, \infty)$. Let $\mathbf{x}_1$ be the minimizer of*

$$\begin{aligned} V_1 = \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} h(|x_i|) \\ s.t. \quad & A\mathbf{x} = \mathbf{b}, \end{aligned} \tag{4.38}$$

and let $(\mathbf{x}_2, \mathbf{u}_2)$ be the minimizer of

$$\begin{aligned} V_2 = \min_{\mathbf{x},\mathbf{u}\in\mathbb{R}^n} \quad & \sum_{i=1}^{n} h(u_i) \\ s.t. \quad & A\mathbf{x} = \mathbf{b} \\ & |x_i| \le u_i, \quad i = 1, \dots, n. \end{aligned} \tag{4.39}$$

Then, $\mathbf{x}_1 = \mathbf{x}_2$ and the optimization problems (4.38) *and* (4.39) *are equivalent.*

Proof. From (4.39) we have

$$V_2 = \begin{array}{l} \min\limits_{\mathbf{x}\in\mathbb{R}^n} \\ \text{s.t. } A\mathbf{x} = \mathbf{b} \end{array} \left[\begin{array}{ll} \min\limits_{\mathbf{u}\in\mathbb{R}^n} & \sum\limits_{i=1}^{n} h(u_i) \\ \text{s.t.} & |x_i| \le u_i, \quad i = 1, \dots, n \end{array} \right]. \tag{4.40}$$

Note that

$$\left[\begin{array}{ll} \min\limits_{\mathbf{u}\in\mathbb{R}^n} & \sum\limits_{i=1}^{n} h(u_i) \\ \text{s.t.} & |x_i| \leq u_i, \quad i = 1, \ldots, n \end{array}\right] = \sum_{i=1}^{n} h(|x_i|), \tag{4.41}$$

since $h(\cdot)$ is a strictly increasing function. Therefore,

$$\begin{array}{rll} V_2 = & \min\limits_{\mathbf{x}\in\mathbb{R}^n} & \sum\limits_{i=1}^{n} h(|x_i|) \\ & \text{s.t.} & A\mathbf{x} = \mathbf{b} \end{array} \quad = \quad V_1 \tag{4.42}$$

by the definition of V_1 in (4.38).

□

By lemma 4.1, since $\ln(1+t)$ is concave on its domain $(-1, \infty)$ due to the derivative, $1/(1+t)$ being a strictly decreasing, positive valued function, an equivalent formulation of (4.37) is

$$\begin{array}{ll} \min\limits_{\mathbf{x},\mathbf{u}\in\mathbb{R}^n} & \sum\limits_{i=1}^{n} \ln(1 + u_i/\xi) \\ \text{s.t.} & A\mathbf{x} = \mathbf{b} \\ & |x_i| \leq u_i, \quad i = 1, \ldots, n. \end{array} \tag{4.43}$$

We have now formulated problem (4.43) in a form that can be written generally as

$$\begin{array}{ll} \min\limits_{z} & g(z) \\ \text{s.t.} & z \in \mathcal{C}, \end{array} \tag{4.44}$$

where $\mathcal{C}$ is a convex set. Since a sum of concave functions is concave, so is our choice of function g in (4.43). That means that g is below its tangent and a guess z at the solution can be improved by minimizing a linearization of g around z. From this observation, we can define the following MM algorithm inductively by starting with $z^{(0)} \in \mathcal{C}$ and

$$\begin{array}{ll} z^{(m+1)} = \arg\min\limits_{z} & g(z^{(m)}) + \nabla g(z^{(m)}) \cdot (z - z^{(m)}) \\ \text{s.t.} & z \in \mathcal{C}. \end{array} \tag{4.45}$$

The above optimization problem can be simplified by noting that only one term is not constant in each iteration,

$$\begin{array}{ll} z^{(m+1)} = \arg\min\limits_{z} & \nabla g(z^{(m)}) \cdot z \\ \text{s.t.} & z \in \mathcal{C}. \end{array} \tag{4.46}$$

Differentiation of the objective function in (4.43) and insertion into (4.46) yields

$$
\begin{aligned}
(\mathbf{x}^{(m+1)}, \mathbf{u}^{(m+1)}) = \underset{\mathbf{x},\mathbf{u}\in\mathbb{R}^n}{\arg\min} \quad & \sum_{i=1}^{n} \frac{u_i}{u_i^{(m)} + \xi} \\
\text{s.t.} \quad & A\mathbf{x} = \mathbf{b} \\
& |x_i| \leq u_i, \quad i = 1, \ldots, n,
\end{aligned}
\tag{4.47}
$$

which is equivalent to

$$
\begin{aligned}
\mathbf{x}^{(m+1)} = \underset{\mathbf{x}\in\mathbb{R}^n}{\arg\min} \quad & \sum_{i=1}^{n} \frac{|x|_i}{|x_i^{(m)}| + \xi} \\
\text{s.t.} \quad & A\mathbf{x} = \mathbf{b}.
\end{aligned}
\tag{4.48}
$$

The weights in algorithm 4.1 can now be seen in (4.48).

4.2.6 MM-formulation of the optimization problem

The problem (4.26) is now relaxed using a weighted ℓ_1 norm with weights defined by algorithm 4.1. It is noted in [1] that several properties of this algorithm need more research and investigation and that the algorithm is analyzed from a practical point of view, where no claim of optimality and convergence is made. We rewrite the problem as

$$
\begin{aligned}
\underset{\boldsymbol{\kappa}_m}{\text{minimize}} \quad & \|\mathbf{w}_m \odot (D\boldsymbol{\kappa}_m)\|_1 \\
\text{subject to} \quad & |\mathbf{x}_L^i(\boldsymbol{\kappa}_m) - \mathbf{x}_{ref}^i| < \varepsilon_i, \quad i = 0, \ldots, N \\
& |\mathbf{y}_L^i(\boldsymbol{\kappa}_m) - \mathbf{y}_{ref}^i| < \varepsilon_i, \quad i = 0, \ldots, N \\
& |\boldsymbol{\kappa}_m^i| < \kappa_{\max}, \quad i = 0, \ldots, N.
\end{aligned}
\tag{4.49}
$$

Here, $\mathbf{w} \in \mathbb{R}^{N+1}$ is a vector with weights where all elements are initially set to 1. The weights are updated in each iteration of the optimization as

$$
\mathbf{w}_{m+1}^i = \frac{1}{|(D\boldsymbol{\kappa}_m)^i| + \xi}, \quad i = 0, \ldots, N,
\tag{4.50}
$$

It was shown in [27] that a good rule of thumb for the value of ξ is to be "*slightly smaller than the expected non-zero magnitudes of* $D\boldsymbol{\kappa}$". The effect on the re-weighted elements is that elements in $D\boldsymbol{\kappa}$ that are close to zero will get a higher weight. The elements with non-zero values, i.e. the kink-points, get a lower weight. Intuitively, a higher weight in the optimization essentially pushes down the value of that element and makes the objective function behave more like the ℓ_0-norm.

4.2.7 Curve length issues

The fact that the smoothing algorithm leaves the original path length unchanged can cause issues when dealing with paths generated by the A* algorithm. These paths often contain many sharp corners and jagged edges which, when smoothed, leaves the path "too long". This is often observed as unnecessary ripples on the path, or as the solver being unable to find a feasible solution. A simple method to reduce unnecessary path-length is to apply a moving average filter on the path coordinates. By tuning the filter strength parameter, which controls how

many points to average over, we can smooth out short-term fluctuations while retaining the general shape of the path. We perform the linearization around the averaged path but still limit the deviation around the original A* output to keep the obstacle avoiding property of the path. This has proven to be a useful method that resulted in a reduced path length and reduced ripple effects.

4.3 Optimal control implementation

To discretize the original problem (3.8) we divide the path parameter s into $N+1$ points s_i, with uniform step size s_f/N. $\mathbf{x}_i = (x_i, y_i, \theta_i, \psi_i),\ i = 0, \ldots, N$. For historical reasons, the discretization used for the differential constraints is the backward Euler method. This is because the software used in this thesis was previously developed for a stiff problem. This discretization will be compared to a forward Euler discretization in the results section. The integral in the objective function in (3.8) is approximated by a sum, which yields

$$
\begin{aligned}
\underset{\mathbf{u}, s_f}{\text{minimize}} \quad & Qs_f + \frac{s_f}{N}\sum_{i=0}^{N} u_i^2 \\
\text{subject to} \quad & \mathbf{x}_{i+1} = \mathbf{x}_i + \frac{s_f}{N}\mathbf{f}(\mathbf{x}_{i+1}, s_i) \\
& \frac{(x_i - c_j^x)^2}{a_j^2} + \frac{(y_i - c_j^y)^2}{b_j^2} > 1, \quad j = 1, \ldots, P, \quad i = 0, \ldots, N \\
& \mathbf{x}_0 = (x_0, y_0, \theta_0, \psi_0)^T,\ \mathbf{x}_N = (x_f, y_f, \theta_f, \psi_f)^T \\
& x_{min} \leq x_i \leq x_{max}, \quad i = 0, \ldots, N \\
& y_{min} \leq y_i \leq y_{max}, \quad i = 0, \ldots, N \\
& |\psi_i| \leq \psi_{max}, \quad i = 0, \ldots, N \\
& |u_i| \leq u_{max}, \quad i = 0, \ldots, N \\
& s_f > 0.
\end{aligned}
\tag{4.51}
$$

5 Numerical simulations

5.1 2-phase approach

The artificial, maze-like map, shown in figure 5.1, is used to test the stability and performance of the algorithm. The obstacle center nodes are given a radius of $m = 5$ meters. We further introduce a margin of 2 meters and compensate for the approximate 3 meter width of the vehicle by setting the obstacle margin $M = 4$ meters to the obstacle radii. The extra margin is shown in figure 5.2, where the A* algorithm has found a collision free path. The next step is to apply the clothoid algorithm to smooth it.

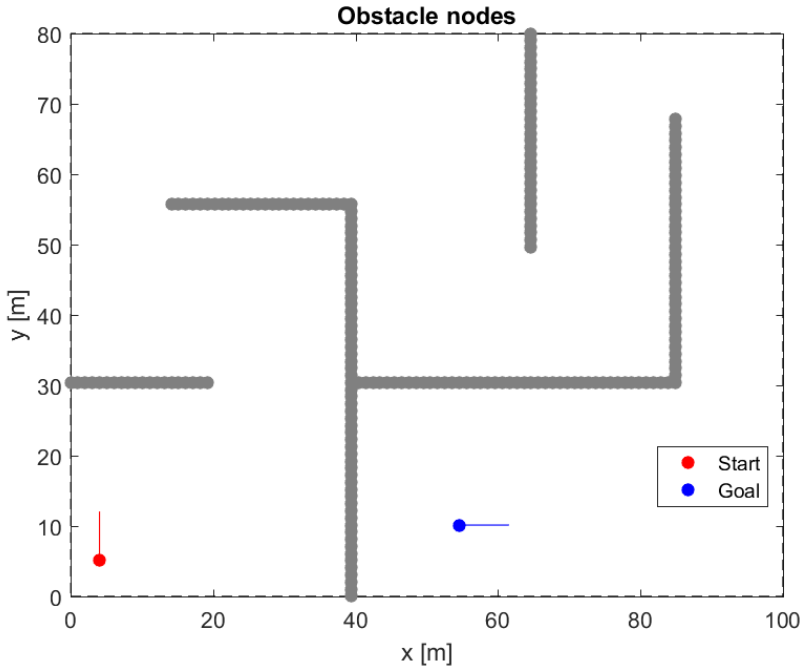

Figure 5.1: The grey obstacle center nodes define the center of the obstacles using a radius parameter. The red and blue lines connected to the start and goal nodes show the predefined heading at each point.

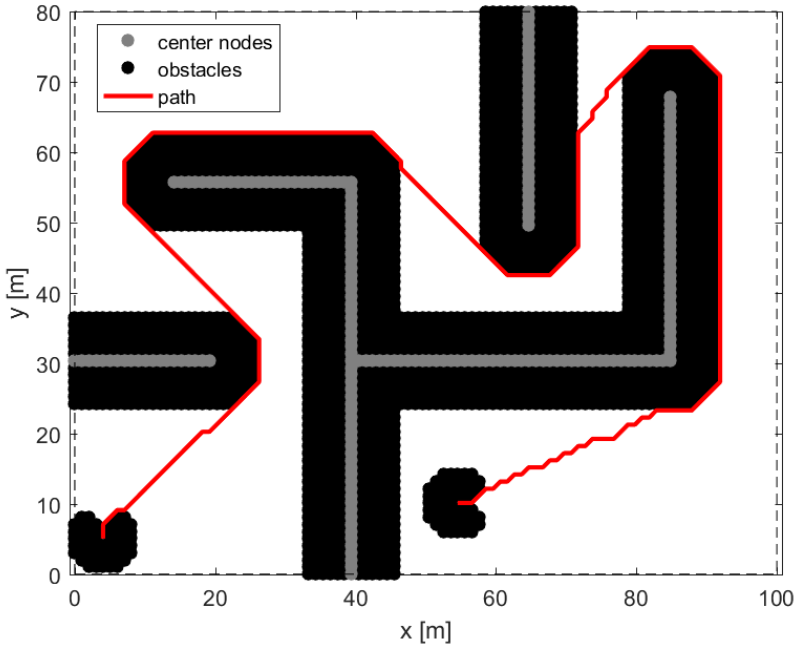

Figure 5.2: This figure shows the reference map used by the A* algorithm to calculate an initial path where the grid consists of 100×80 nodes. The path is not yet driveable and contains some unwanted turns and edges, especially around the coordinates $(x, y) = (80, 60)$.

The smoothed path output from the clothoid algorithm is shown in figure 5.3. The deviation parameters used are $\epsilon_i = 2/\sqrt{2},\ i = 0, \ldots, N-2$, i.e. all points except for the last two. $\epsilon_{N-1} = \epsilon_N = 0.001 \cdot 2/\sqrt{2}$. By restricting the coordinates of the last two points we ensure that we get the correct goal heading. Note that the deviation around the start point will always be 0 as a consequence of the discretization, no matter what the deviation constraint value is. The initial heading will, by the same reason, also be exactly the heading specified by the first two points on the reference path.

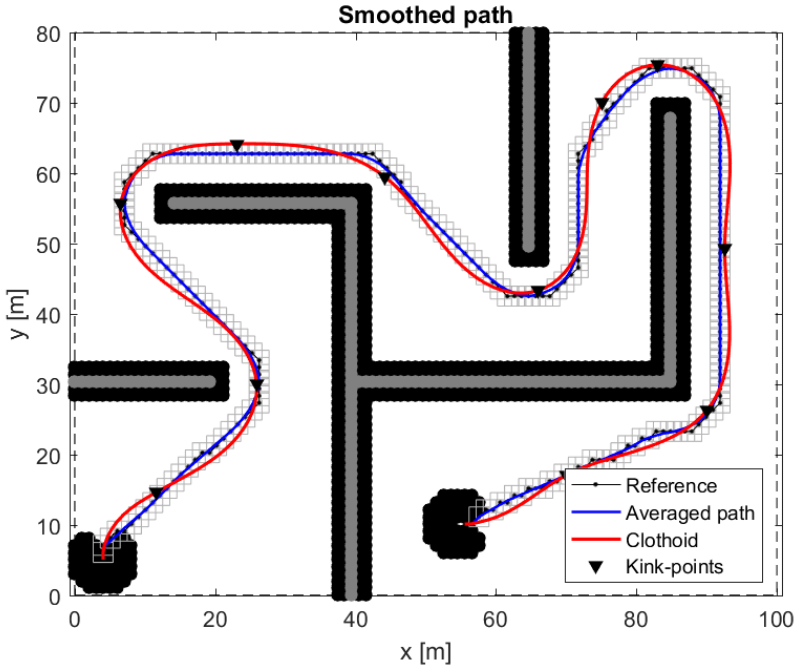

Figure 5.3: Smoothed path where the kink-points have been marked on the path. The grey squares represent the position constraints along the path. We see that the heading and positions for the endpoints are satisfied. The problematic area around $(x, y) = (80, 60)$ has also been smoothed out.

The smoothed path in 5.3 is driveable but not optimal. A swaying can be seen in the straight portions of the path around $(x, y) = (90, 50)$, where one might expect the vehicle to drive straight. An approach to get straighter paths is to add a term $|\boldsymbol{\kappa}|$ to the objective function to penalize curvature. Figure 5.4 shows the curvature of this path. As can be seen in the plot, the curvature profile has been smoothed considerably compared to the initial path.

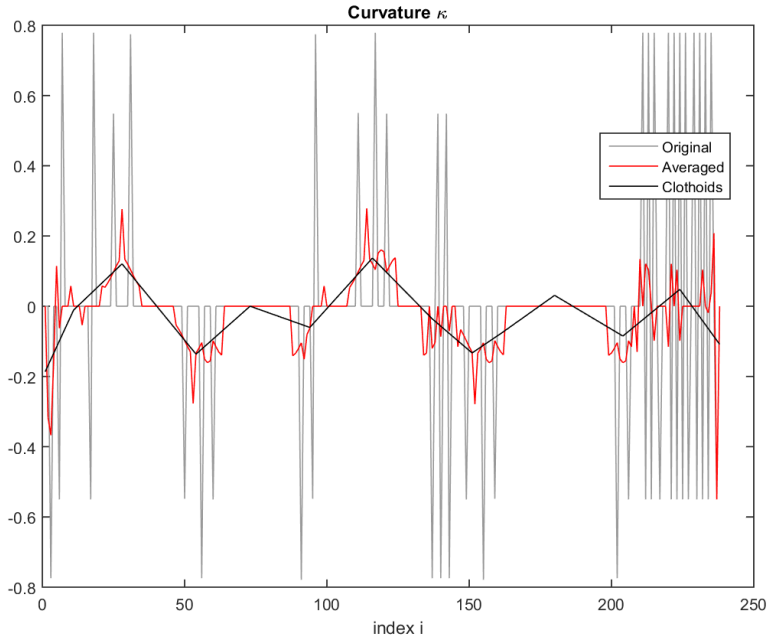

Figure 5.4: The curvature along the path before and after smoothing. The grey line is the curvature of the original path from the A* algorithm. The red curve shows the curvature of the path after a moving average filter has been applied. It has reduced values in the curvature but still contains much noise. The result from the clothoid smoothing is shown in black.

5.1.1 Convergence of ℓ_0-norm relaxation

The convergence of the solution can be checked by plotting the convergence of solution iterates $\boldsymbol{\kappa}_m$ and weights $\mathbf{w}_m$ for different parameters ξ in equation (4.50). By running the outer iterations until the error plateaus we can study the convergence of the solutions. By subtracting each iterate $\boldsymbol{\kappa}_m$ from the last solution, $\boldsymbol{\kappa}_{ref}$, we get the plot in figure 5.5. The error stops decreasing at iteration $m = 3$ around 10^{-8}. The default tolerance used in CVX is set as $\epsilon^{1/2} \approx 10^{-8}$.

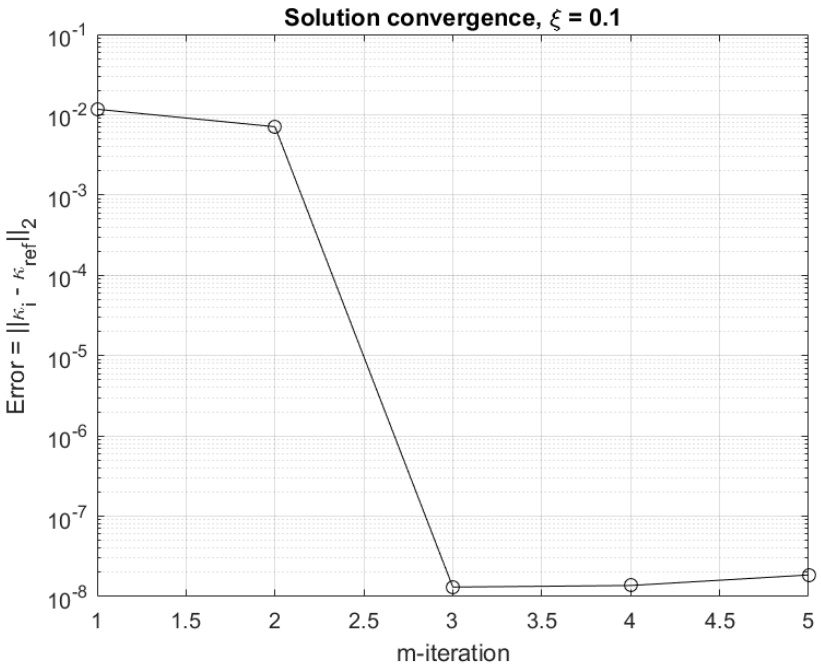

Figure 5.5: Error measured using the Euclidean norm of the iterates $\boldsymbol{\kappa}_m$ with the parameter $\xi = 0.01$, defined in (4.50).

The convergence weight vector $\mathbf{w}$ is shown in figure 5.6. The error for the weights $\mathbf{w}_m$ is higher than for the iterates $\boldsymbol{\kappa}_m$ by a factor of approximately 10^2.

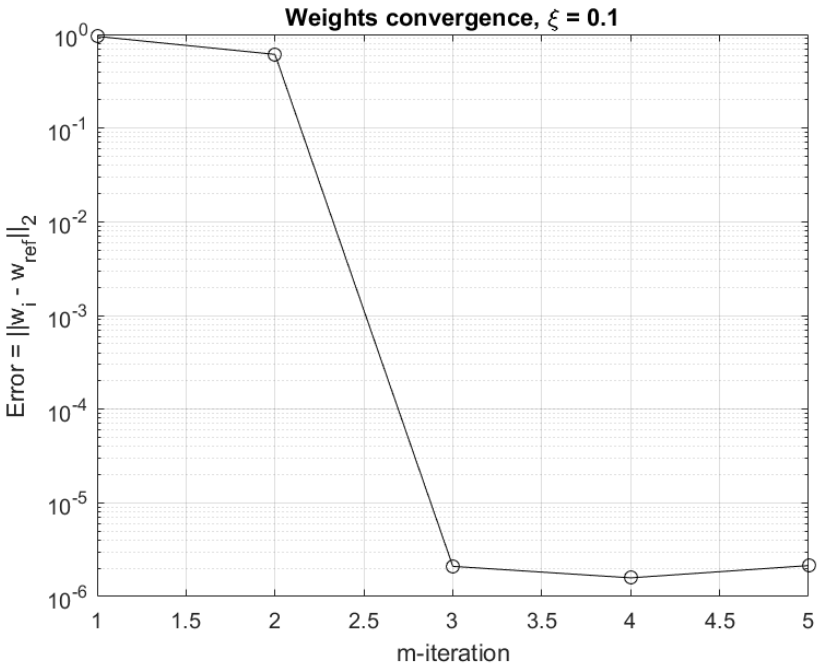

Figure 5.6: Error measured using the Euclidean norm of the iterates $\boldsymbol{\kappa}_m$ with the parameter $\xi = 0.01$.

A plot of the convergence for the weights and solution iterates for several different values of ξ is shown in figure 5.7 and 5.8.

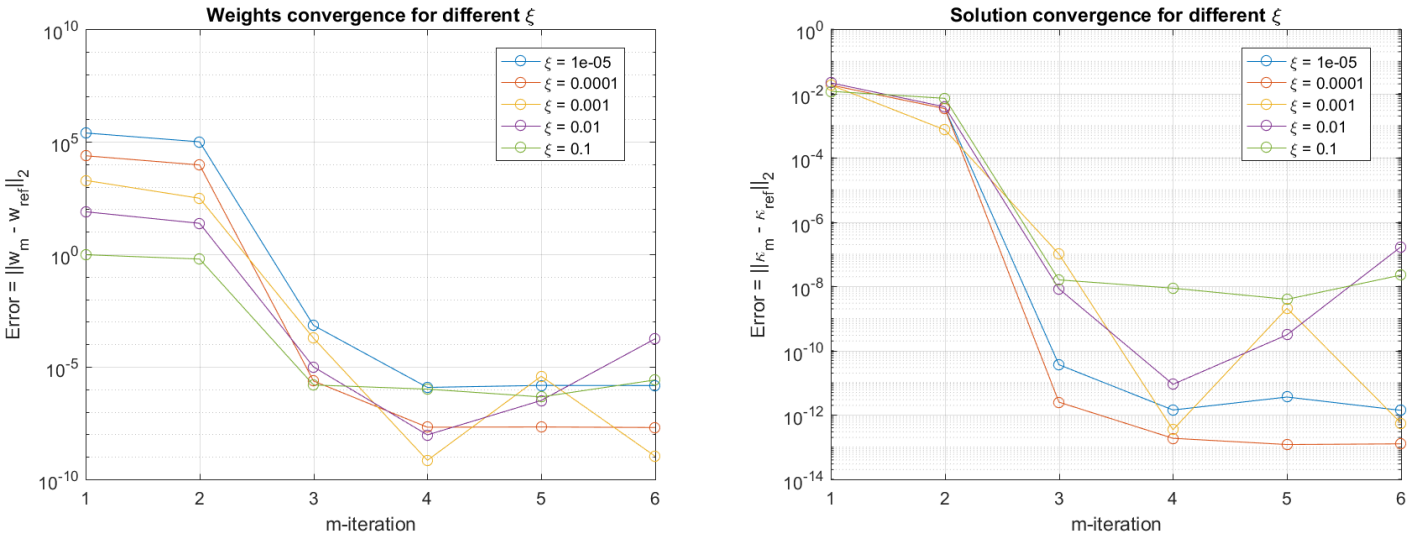

Figure 5.7: Convergence for $\mathbf{w}_m$ (left) and $\boldsymbol{\kappa}_m$ (right). The errors in each figure are calculated as $\|\mathbf{w}_m(\xi) - \mathbf{w}_M(\xi)\|_2$ and $\|\boldsymbol{\kappa}_m(\xi) - \boldsymbol{\kappa}_M(\xi)\|_2$ respectively, which shows the convergence for each value of ξ separately. M corresponds to the index of the last iteration, and in the figures above $M = 7$. It can be seen that the parameter value $\xi = 10^{-4}$ is giving the lowest error after convergence.

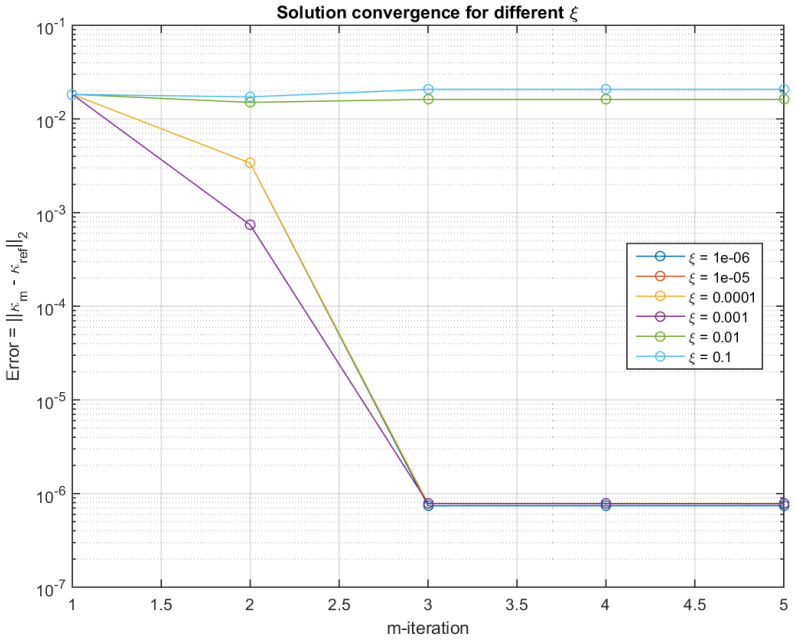

Figure 5.8: This plot shows how the solutions $\boldsymbol{\kappa}_m(\xi)$ converge as $\xi \to 0$. The errors are calculated as $\|\boldsymbol{\kappa}_m(\xi) - \boldsymbol{\kappa}_M(\xi_{min})\|_2$, where $\xi_{min} = 10^{-7}$. M corresponds to the index of the last iteration, and in the figure above $M = 6$. The solutions appear to converge after 3 iterations for $\xi \leq 10^{-3}$.

5.2 Optimal control approach

To study the stability of the solutions obtained from solving the optimal control problem (4.51) we will use the test map shown in figure 5.9 and calculate the error for discretizations with an increasing number of points, N.

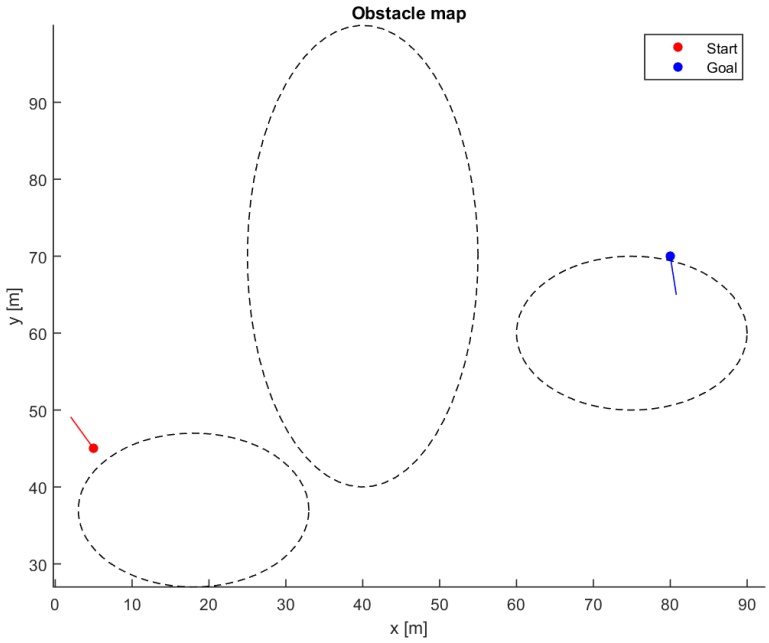

Figure 5.9: The obstacle map that will be used to test the numerical stability of the discretization. Three ellipses define the obstacles, and the start and goal headings are displayed by the lines from the start and goal points.

We solve the optimization problem with different numbers of points N and the solution paths can be seen in 5.11. Since the problem (3.8) is highly nonlinear, a somewhat accurate initial guess is important for the solution to converge. We set the initial values without any regard to the obstacles. We then initialize the variables x and y with the uniformly spaced line between the start and goal coordinates, as shown in 5.10. This does of course not give the correct headings at the start or the end. For θ we set as initial value the linear interpolation between the three angles; start heading, heading of the vector between the start and goal coordinates, and goal heading, also shown in figure 5.10.

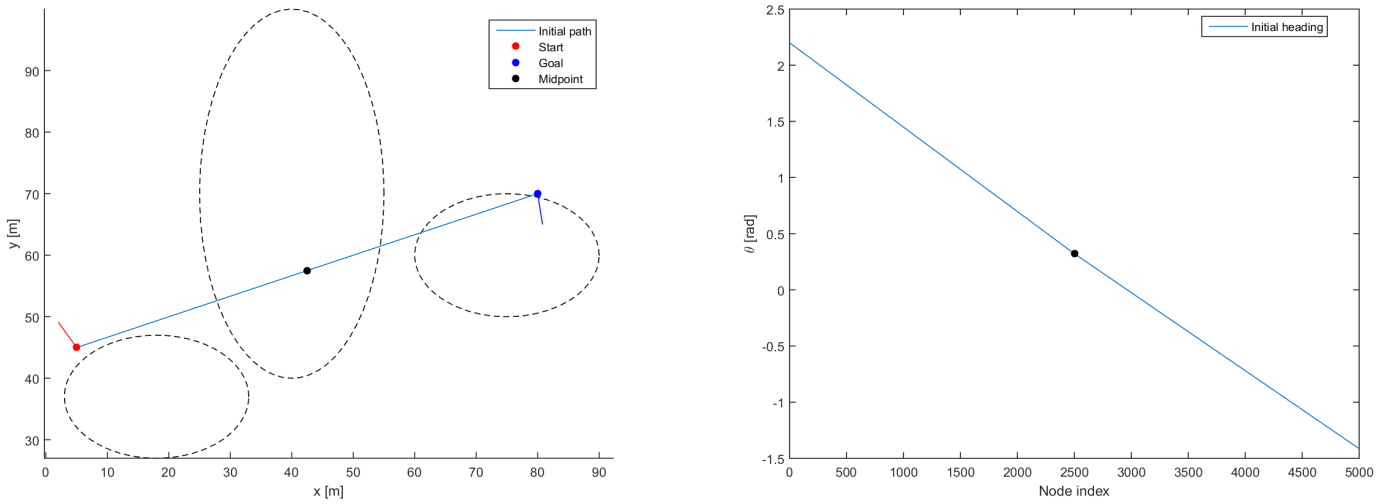

Figure 5.10: A plot of the initial values used for initialization in IPOPT for $N = 5000$. The (x, y) coordinates are initialized using the straight line between the start and goal points, shown to the left. The right figure shows the initial values used for the heading θ. The black point is the midpoint of the path. The initial heading is interpolated linearly between angles at the start and midpoint, and the midpoint and the goal, respectively.

As initial value for the parameter s_f we set the Euclidean distance between the start and goal coordinates. The vectors $\boldsymbol{\psi}$ and $\mathbf{u}$ are initialized with every element set as 1.

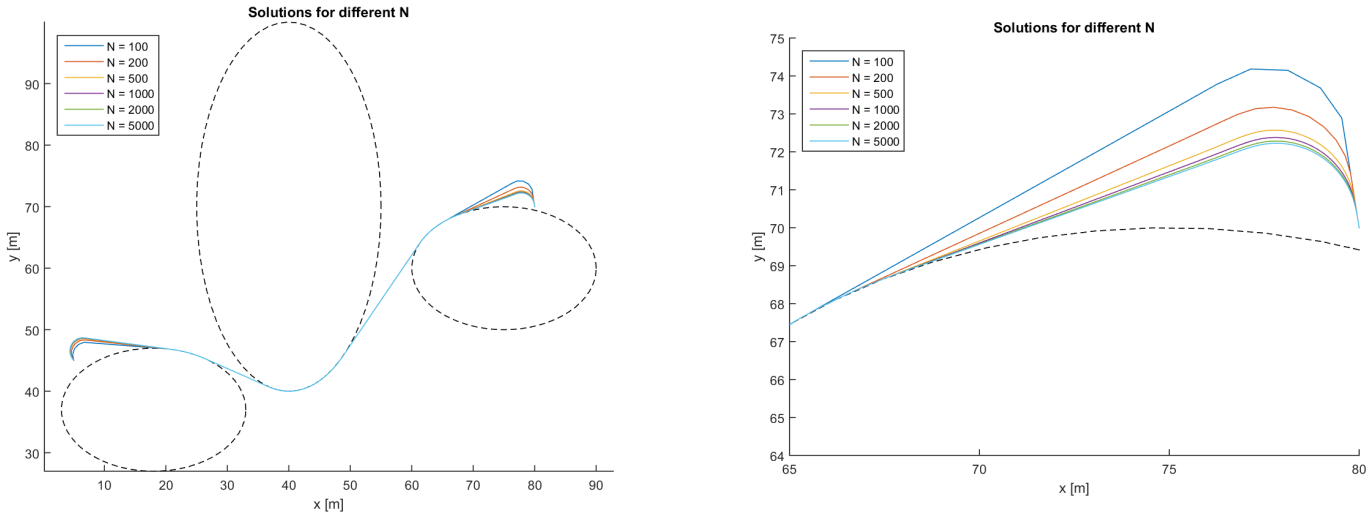

Figure 5.11: A plot of the solutions for different number of substeps N is shown to the left. To the right is the same figure zoomed in on the region around the goal point. The solutions differ only near the endpoints. The middle part is very similar since it is constrained by the obstacles. As N is increased the solution paths make sharper turns and appear to converge.

5.2.1 Error for different step sizes

We will analyze the numerical stability by looking at the iterates $\mathbf{u}_N$ for a discretization for Backward Euler and Forward Euler. We calculate solutions $\mathbf{u}_N$ for $N = 100, 200, 500, 1000, 2000, 5000$ and set $\mathbf{u}_{ref} = \mathbf{u}_{5000}$ as the reference solution. Since there are different numbers of elements in the vectors $\mathbf{u}_N$ we perform a linear interpolation of them using $N_{min} = 100$ uniformly spaced sample points, giving us the set of solutions $\mathbf{z}_N$. The error is then calculated as

$$e_N = \|\mathbf{z}_N - \mathbf{z}_{ref}\|_2 \ , \tag{5.1}$$

and is shown in figure 5.12.

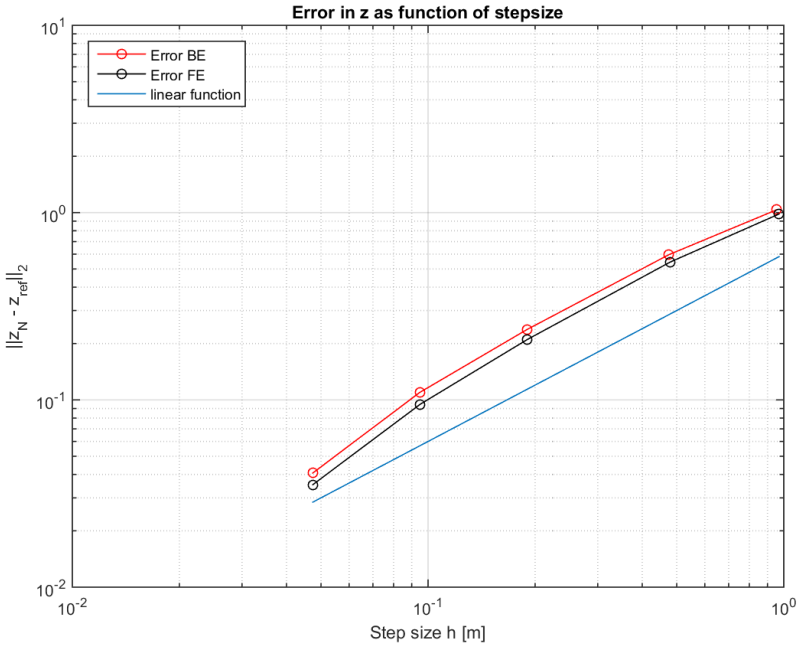

Figure 5.12: The error in the linearly interpolated solutions for different number of sub-steps. The blue line is a linear function $y = kh$ and is plotted to compare the slope of the error plot. The step size is calculated by dividing the total resulting length of the solution path by N.

In figure 5.12 we see that we approximately get a first order convergence in the solution. Normally when doing stability analyses on solutions of differential equations, the order of accuracy of the discretization method can be seen in the slope of the error vs step-size plot. Since Forward and Backward Euler are first order methods, linear convergence is expected.

5.2.2 Influence of the parameter Q

The parameter Q in (4.51) controls the weighting between the length and the smoothness of the curve. Figure 5.13 shows the influence on the path different Q have and figure 5.14 shows the corresponding plot of the control variable $u(s)$.

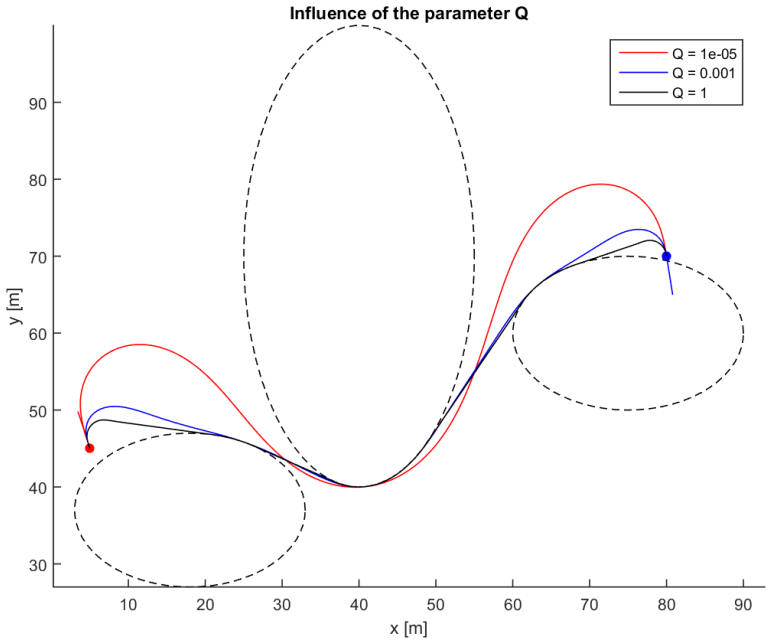

Figure 5.13: Solutions to the test problem for different values of Q, with $N = 400$. As Q is increased, the penalty of the path length is increased in the objective function and results in the paths taking sharper turns.

There is a trade-off between path smoothness and path length to take into consideration. The vehicle is able to drive faster on a smoother path than on a path with sharper turns. A lower value on Q was seen to increase the chances of convergence and finding of a solution to the optimization problem. However, this yields longer paths and could increase the fuel consumption of the vehicle. The control signal $u(s)$ for the different solutions are shown in figure 5.14.

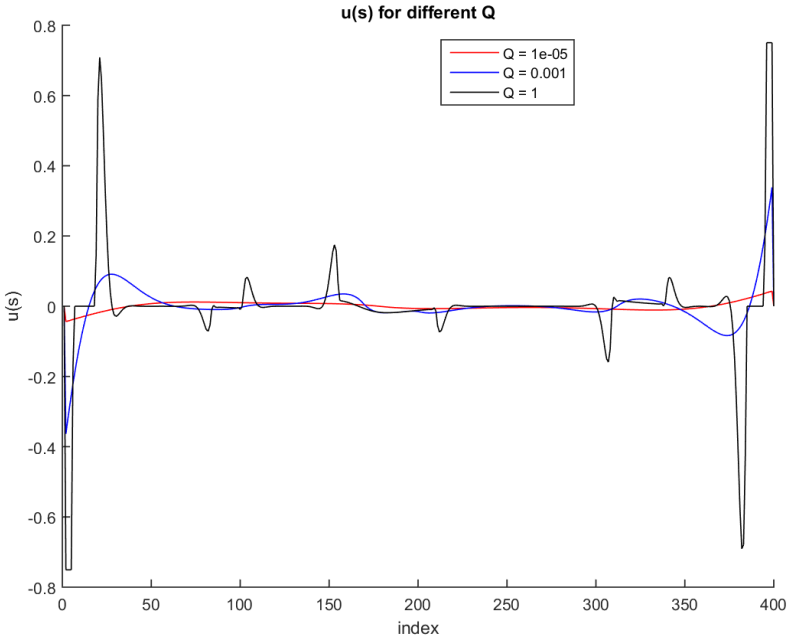

Figure 5.14: Control signal $u(s)$ for solutions to the optimization problem with different parameters Q. As the weights are increased, the variation in the control signal increase too. For $Q = 1$ it can be seen in the beginning and the end of the interval that the control signal strength reaches the allowed limit, as expected.

5.2.3 Empirical stability analysis

A test case setup was created to solve the problem (4.51) multiple times with slight perturbations to obstacle parameters, path coordinates and headings. It was used to empirically evaluate the stability and convergence of the IPOPT algorithm. A relatively simple obstacle map, shown in figure 5.15 is used.

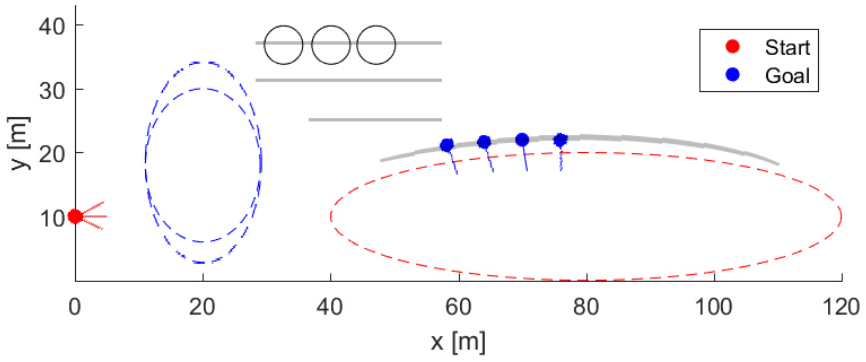

Figure 5.15: A schematic map used to test convergence properties for variations of the problem parameters. As is indicated in the figure, several different parameter variations were used. The blue obstacle to the left had two variations in y-radius, the single small black circle was moved uniformly along the grey lines, and the end position and heading was moved along the boundary of the big red obstacle. Lastly, the start heading was given three possible values. All possible permutations of these variations were tested.

A number of parameters were given a set of values and the algorithm was then run for each possible permutation of these parameters. It was found that some iterations that we would expect to converge quickly, for instance where the obstacle was moved to a position not intersecting with the optimal path, the algorithm performed erratically and the solution diverged. Several methods have been tested to improve the rate of success of convergence. By solving the optimization problem several times with different parameters, using the previous solution as initial values for the variables, the rate of successful runs could be improved.

It was noted that, for instance, low values of the parameter Q in (4.51) improved the chances of convergence in some cases. Then that solution was used to initialize the variables for a new pass of the algorithm in which the correct (i.e. higher) value of Q was used. Chaining several iterations of this type where the parameter Q was changed was performed. This method improved the rate of success for solving the problems with convergence. However, there were several instances where the changed Q parameter still did not result in a converged solution. A list of the procedures that were used is shown in table 5.1. An important note is that since we solve the complete problem in each of these steps, we double the execution time by just adding one extra step. Furthermore, a divergent result from the first step will often cause divergence in the steps after as well, due to the poor initial values they would be given.

Procedure	Steps
P1	1. Low Q 2. High Q
P2	1. Cost function only s_f 2. Normal cost function with high Q
P3	1. No obstacles 2. Adding one obstacle at a time
P4	1. No obstacles 2. Adding all obstacles at once

Table 5.1: A summary of the different procedures used to improve convergence.

The idea behind procedure P3, to solve the problem with divergent solutions, was to disable obstacles in the solution. This was done by first running the optimization without any obstacles, then adding them back in one by one. In each iteration, the solution from the previous step was used to initialize all variables. An observation from this method was that the solution path tended to stay on the same side of obstacles as the initial guess from the previous iteration. In some cases that resulted in unnecessarily long paths and convergence only to local optima. This method, even in combination with the method to use bigger Q values, did not ensure convergence in all cases either. Finally, in case of many obstacles present in the problem, this procedure is computationally heavy due to the number of iterations needed.

Procedure P4 gave similar results as P3 with a shorter computational time. The result after the first step is an improved version of the straight line interpolation used as initial value. It also provides a better initial guess for θ and ψ. However, similar to the other procedures, P4 did not yield 100 % convergent solutions either.

The use of these procedures improved the rate of successful runs of the algorithm in many cases, but not in all. It was not uncommon that a problem setup that worked in P1 and P2 did not work in P3 or P4 and vice versa. There were also iterations that did not converge for any procedure. The conclusion from this was that we were unable to formulate a procedure able to converge for all the cases in our test setup.

A test where obstacle avoiding paths generated by the A* algorithm were used to initialize parameter values was also performed. However, this did not yield any improvement at all to the chances of convergence compared to using the linear interpolations shown in figure 5.10.

6 Discussion

6.1 2-step approach discussion

A grid based path planner will always find a shortest path if such a path exists. Moreover, it is easy to define obstacles of any shape without affecting the performance of the algorithm. However, a significant disadvantage with a path planning algorithm based on A* is that the path produced is not directly driveable since physical vehicle constraints, such as turning radius etc., are not respected. In the case of thin obstacles, or obstacles with sharp corners, the A* path will contain very sharp turns that are impossible for the vehicle to follow. This problem was handled by introducing obstacle nodes with given radii that can be set, limiting the maximum curvature around sharp corners. A more refined solution would be to implement a grid-based planner based on driveable motion primitives.

In a grid, there are often several paths that have the same length, where some, by inspection by a human, can be judged to be worse than others. There is an element of randomness involved in which of these paths of identical length that the algorithm will return. This often introduces excessive wobbliness to the path. The clothoid smoothing algorithm is very dependent on the reference path which is why paths planned by this 2-step approach may exhibit this wobbliness. However, the clothoid smoothing algorithm is appealing since it is a convex problem with a guaranteed solution. In a system where optimal paths are important but stability is critical, the 2-step approach with guaranteed convergence is a good option. The path generated by the two-step approach will be close but never optimal.

6.2 Optimal control discussion

In choosing to formulate the optimal control problem in distance s rather than in time t, we simplify the problem by reducing the number of necessary control signals, from velocity and steering to only steering. However, there are several ways to quantify how good a path is besides the total length or smoothness of the control signals. In a time formulation, one could instead be interested in the path with the shortest traversal time. This is not necessarily the shortest path. For instance, given a map with many small obstacles located in the middle, the shortest path might be the one snaking its way between the obstacles. A path that circles around all obstacles in one sweep could be longer but allow the vehicle to drive faster, therefore being the faster route. Furthermore, a time formulation would allow reversing maneuvers and hence be able to solve more complex maps.

Another alternative is to omit the dependence on path length and traversal time completely and instead minimize the total energy needed to traverse the path. These approaches would be interesting to investigate further.

6.2.1 Issues with convergence

As opposed to the convex formulation of the clothoid smoothing problem, the optimal control formulation is not guaranteed to converge to a global optimum. A good initial guess is critical for the algorithm to converge to the correct

optimum. The grid based planner can handle any number and configuration of obstacles and solves mazes easily, while the optimal control algorithm is very sensitive to the obstacle configuration.

From practical experience, it has been clear that a problem setup, seemingly similar to a setup that was successfully solved regarding obstacle placement and boundary conditions for the path, may result in erratic behavior by the algorithm. One goal of this project was to evaluate the stability and convergence of these methods to calculate paths for the HX1. Convergence is critical and any risk that the HX1 could freeze up because of a solution that was unable to converge is unacceptable. It was found that the methods we used were not able to guarantee convergence and more investigation is needed to understand why and to find better solutions.

6.3 Future work

Several topics mentioned in this thesis would be interesting to investigate further. These topics include time domain models, different cost functions (e.g. penalizing path length, traversal time, control input smoothness and used energy) and using improved grid search algorithms (Theta*, Field D*, etc.). In locations with large variation in altitudes, three-dimensional smoothing that takes the z-coordinate into account as well would be of interest.

7 Conclusion

Two methods that solve a path finding problem for an autonomous vehicle in the presence of obstacles have been shown. The first approach was to separate the task into two steps, obstacle free path finding and path smoothing, to obtain a drivable path. This was done by representing the vehicle's operating environment by a map with obstacles on a grid and using the efficient grid-based search algorithm A* to find a path from start to goal. The A* algorithm does not respect heading constraints, which was taken into account by adding virtual obstacles that restrict the headings at the start and end of the path, approximately fitting the desired headings. The path was then smoothed by fitting the smallest number of continuous clothoid segments to the path, satisfying maximum deviation constraints. The continuous mixed integer non-linear optimization problem was discretized and rewritten on convex form using a relaxation algorithm for the ℓ_0 norm, and linearization of the deviation constraints around the curvature of the reference path. The now convex problem was solved in Matlab using the solver CVX, which resulted in a smoothed path that keeps the initial and goal headings.

The second approach was to formulate the path finding problem as an optimal control optimization problem. With this method, the path finding and path smoothing are performed in the same step. This approach allowed for finding optimal paths passing through a set of obstacles defined by ellipses. The problem was discretized with Forward and Backward Euler and then solved using IPOPT. It was found that a first order convergence in the error could be seen. For use in an autonomous system, where convergence is critical, it can be argued that the 2-step method is more robust, even though the resulting path

from that algorithm is not as good as the path calculated by the optimal control formulation.

8 References

[1] P. F. Lima, *Predictive control for autonomous driving*. PhD thesis, KTH, 2016. Unpublished thesis.

[2] R. Geraerts and M. H. Overmars, "Sampling techniques for probabilistic roadmap planners," Tech. Rep. UU-CS-2003-041, Institute of Information and Computing Sciences, 2003.

[3] H. Andreasson, J. Saarinen, M. Cirillo, T. Stoyanov, and A. J. Lilienthal, "Fast, continuous state path smoothing to improve navigation accuracy," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, may 2015.

[4] D. S. Meek and D. J. Walton, "Clothoid spline transition spirals," *Mathematics of Computation*, vol. 59, pp. 117–117, sep 1992.

[5] "Ipopt." `https://projects.coin-or.org/Ipopt`. (Accessed on 10/18/2017).

[6] C. Sprunk, "Planning motion trajectories for mobile robots using splines." Student project, University of Freiburg, 2008.

[7] N. Buniyamin, W. Wan Ngah, N. Sariff, and Z. Mohamad, "A simple local path planning algorithm for autonomous mobile robots," *International Journal of Systems Applications, Engineering and Development*, vol. 5, no. 2, pp. 151–159, 2011.

[8] S. Behere, *A Generic Framework for Robot Motion Planning and Control*. PhD thesis, KTH Royal Institute of Technology, 2010.

[9] A. Pasha, *Path planning for nonholonomic vehicles and its application to radiation environments*. PhD thesis, University of Florida, 2003.

[10] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, p. 497, jul 1957.

[11] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[12] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Department of Computer Science, Iowa State University, 1998.

[13] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846–894, jun 2011.

[14] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *Journal of Field Robotics*, vol. 26, pp. 308–333, mar 2009.

[15] M. Cirillo, T. Uras, and S. Koenig, "A lattice-based approach to multi-robot motion planning for non-holonomic vehicles," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, sep 2014.

[16] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun, "Junior: The stanford entry in the urban challenge," *Journal of Field Robotics*, 2008.

[17] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, IEEE, 2000.

[18] S. Koenig and M. Likhachev, "D*lite," in *Eighteenth National Conference on Artificial Intelligence*, (Menlo Park, CA, USA), pp. 476–483, American Association for Artificial Intelligence, 2002.

[19] D. Knowles and M. R. Murray, "Real time continuous curvature path planner for an autonomous vehicle in an urban environment." California Institute of Technology, September 2006.

[20] D. Ferguson and A. T. Stentz, "The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments," Tech. Rep. CMU-RI-TR-05-19, Carnegie Mellon University, Pittsburgh, PA, June 2005.

[21] Stanford Report, "Stanford, volkswagen team up to create automotive research lab." `http://news.stanford.edu/news/2007/november28/volks-112807.html`, November 2007. (Accessed on 06/09/2017).

[22] P. F. Lima, M. Trincavelli, J. Mårtensson, and B. Wahlberg, "Clothoid-based model predictive control for autonomous driving," in *:*, 2015. QC 20150826.

[23] R. A. Ferlis, "Public roads - the dream of an automated highway." `https://www.fhwa.dot.gov/publications/publicroads/07july/07.cfm`, Jul/Aug 2007. (Accessed on 05/08/2017).

[24] U. D. of Transportation, "Critical reasons for crashes investigated in the national motor vehicle crash causation survey," February 2015. [Online; posted February-2015].

[25] S. international, "Automated driving: Levels of driving automation are defined new sae international standard j3016." `https://www.sae.org/misc/pdfs/automated_driving.pdf`, January 2014. (Accessed on 06/04/2017).

[26] "Innovation — volvo construction equipment global." `https://www.volvoce.com/global/en/this-is-volvo-ce/what-we-believe-in/innovation/`. (Accessed on 05/19/2017).

[27] E. J. Candès, M. B. Wakin, and S. P. Boyd, "Enhancing sparsity by reweighted ℓ 1 minimization," *Journal of Fourier Analysis and Applications*, vol. 14, pp. 877–905, oct 2007.

[28] M. S. Lobo, M. Fazel, and S. Boyd, "Portfolio optimization with linear and fixed transaction costs," *Annals of Operations Research*, vol. 152, pp. 341–365, dec 2006.

[29] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," 2016.

[30] A. Micaelli and C. Samson, "Trajectory tracking for unicycle-type and two-steering-wheels mobile robots," Research Report RR-2097, INRIA, 1993.

[31] E. Bertolazzi and M. Frego, "G1fitting with clothoids," *Mathematical Methods in the Applied Sciences*, vol. 38, pp. 881–897, mar 2014.

[32] R. Levien, "The euler spiral: a mathematical history," Tech. Rep. UCB/EECS-2008-111, EECS Department, University of California, Berkeley, Sep 2008.

[33] M. E. Vázquez-Méndez and G. Casal, "The clothoid computation: A simple and efficient numerical algorithm," *Journal of Surveying Engineering*, vol. 142, p. 04016005, aug 2016.

[34] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[35] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-angle path planning on grids," *Journal Of Artificial Intelligence Research, Volume 39, pages 533-579, 2010*, 2014.

[36] A. Nash, S. Koenig, and C. Tovey, "Lazy theta*: Any-angle path planning and path length analysis in 3d," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pp. 147–154, AAAI Press, 2010.

[37] L. Blackmore, "Robust path planning and feedback design under stochastic uncertainty," in *in Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2008.

[38] "Introduction to ipopt: A tutorial for downloading, installing, and using ipopt." `http://web.mit.edu/ipopt_v3.11.6/doc/documentation.pdf`, November 2013. (Accessed on 05/22/2017).

[39] K. Lange, *Optimization*. Springer New York, 2013.

[40] M. Feng, J. E. Mitchell, J.-S. Pang, X. Shen, and A. Wächter, "Complementarity formulations of l0-norm optimization problems," *Industrial Engineering and Management Sciences. Technical Report. Northwestern University, Evanston, IL, USA*, 2013.

[41] J. F. Claerbout and F. Muir, "ROBUST MODELING WITH ERRATIC DATA," *GEOPHYSICS*, vol. 38, pp. 826–844, oct 1973.

[42] E. Candes, M. Rudelson, T. Tao, and R. Vershynin, "Error correction via linear programming," in *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*, IEEE, 2005.

TRITA -MAT-E 2017:73
ISRN -KTH/MAT/E--17/73--SE