NTNU – Trondheim
Norwegian University of
Science and Technology

Motion Planning Algorithms for Marine Vehicles

Knut Hvamb

Master of Science in Cybernetics and Robotics
Submission date: July 2015
Supervisor: Morten Breivik, ITK
Co-supervisor: Anastasios Lekkas, IMT

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem description

The candidate will consider the problem of implementing and adapting motion-planning algorithms for unmanned vehicle-related applications. The following elements must be considered:

1. Implement a number of sampling-based and combinatorial algorithms (such as RRTs, Probabilistic Roadmaps and Voronoi diagrams) in two dimensions. Use environments with obstacles.
2. Continue with the RRT and:
   - Apply Dubins car dynamics and clothoids for curvature continuity.
   - Apply Dubins car dynamics and Fermat's spirals for curvature continuity.
   - Compare the different methods.
3. Investigate the possibility of using the generated paths (Fermat's and clothoid) as input for guidance techniques, such as path tracking in the case where ocean current is present as a disturbance.

Assignment given: January 12th 2015
Supervisors: Morten Breivik, ITK and Anastasios Lekkas, IMT

Abstract

In this Master's thesis, some of the most popular techniques in the motion-planning field are implemented in two-dimensional areas with obstacles. The Voronoi diagram method, the probabilistic roadmap method and the rapidly-exploring random trees method (RRT). Because the RRT has proved to be very efficient in finding paths connecting two points on a map while avoiding obstacles in earlier work, the method is in this thesis combined with differential constraints such as curvature continuity, which can be achieved using Dubin's vehicle kinematics with Fermat's spiral and clothoid transitions. A comparison is made showing that the Fermat's spirals computational time is much lower than for the clothoid. To determine whether the smoothing methods are suitable for collision-avoidance applications for marine vehicles, the main part of the thesis is dedicated to test the generated paths as input for an extended version of the trajectory tracking guidance system from Lekkas and Fossen (2014). Even when introducing ocean current to the vehicle while tracking, the method shows promising results, and future work could revolve around further tuning of the system and the generated paths, or expanding the method with another dimension to use for 3-D applications.

Sammendrag

(Norwegian translation of the abstract)

I denne masteroppgaven blir noen av de mest populære baneplanleggnings metodene innen feltet implementert for to-dimensjonale områder med hindringer. Det er Voronoi diagram metoden, probabilistisk veikart metoden og den raskt utforskende tilfeldige trær metoden (RRT). Fordi RRT i tidligere arbeid har vist seg å være svært effektiv til å finne veier som forbinder to punkter på et kart samtidig som man unngår hindringer, blir metoden i denne avhandlingen kombinert med differensielle begrensninger som kontinuerlige krumninger ved hjelp Dubins kjøretøy kinematikk kombinert med Fermats spiral og klotoide overganger. En sammenlikning av metodene viser at Fermats spiralens utregningstid er veldig mye lavere enn for klotoiden. For å finne ut om metodene er egnet for kollisjonsunngående applikasjoner som marine kjøretøy er hoveddelen av oppgaven dedikert til å teste de genererte banene som grunnlag for en utvidet versjon av systemet i Lekkas and Fossen (2014). Det viser seg at selv når man tilfører havstrømmer på kjøretøyet gir metoden lovende resultater, og videre arbeid kan dreie seg om ytterligere testing eller å utvide metoden for 3-D-applikasjoner.

Preface

This report is a Master's thesis written to conclude my five years of studying engineering cybernetics at NTNU Gløshaugen in Trondheim, Norway. During these years I have learned a lot about myself, made many new friends and experienced both ups and downs. The freedom in the life of a student is something I would recommend to everyone.

I really want to thank my family for always believing in me when I faced some struggles during my studies. I am really happy that in the end I am able to complete my degree in five years!

I am also very thankful towards my supervisors Morten Breivik and Anastasios Lekkas who have helped and supported me this last year at NTNU. The knowledge you two have in the field of cybernetics is incredible, thanks for everything you have taught me.

Trondheim, July 13, 2015

Knut Hvamb

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The motion planning problem originates from 1950s when the famous American mathematician and electronic engineer Claude Shannon (Figure 1.1) made an electronic mice which explored a maze with a simple maze-searching algorithm. This method saw the environment as a check board and every square would include two bits, which after being visited indicated the direction the mice left. This simple algorithm would later be optimised to find the shortest path to the exit, by changing the check board into a graph (set of objects where some pairs of objects are connected by links). Then solve by using shortest-path methods for graphs which are described in most algorithm introduction courses.

Figure 1.1: Claude Shannon and his maze-solving mouse in 1952, from CybernéticZoo (2009)

Also during the 40s the idea of manipulator arms (robotic arms) was introduced, first for the purpose of handling radioactive material. The more intelligent control systems for such arms wasn't researched until the late 60s where the dynamics, kinematics and trajectories were taken into account. This eventually led to the problem of motion planning being clearly defined during the 70s:

> *Motion planning involves getting a robot to automatically determine how to move while avoiding collisions with obstacles.* - LaValle (2011a)

During the 80s combinatorial solutions were found for the problem. One of them was the Voronoi diagram method. In the 90s the sampling-based methods such as the RRT and the PRM were introduced and was shown to offer more practical solutions for modern industry. Because these methods have proved to be very efficient in finding paths by essentially connecting two points on a map, they are very suitable for motion-control scenarios where the goal is to navigate in narrow environments filled with obstacles. It is because of these properties the methods have been extended to other fields than robotics, such as unmanned vehicle-related applications. An important question today is if these sampling-based methods from the robotics literature can compete with the complete motion planning techniques, such as the Voronoi diagrams, where you can determine if there exists a solution.

Figure 1.2: Kongsberg Maritimes AUV Remus 100, from `http://www.ntnu.edu/amos/aur-lab`

One of the sampling-based methods which are very relevant today is Steven M. LaValle's Rapidly-exploring Random Trees, LaValle (1998). By taking differential constraints into account and creating motion primitives, such as curvature continuity, can this method provide satisfying results when generating paths to be used as input for today's guidance systems? A lot of today's research revolves around creating motion planning systems for generating paths for autonomous vehicles. Can the Rapidly-exploring Random Trees contribute to this field?

1.2 Contribution

The main contribution of this thesis is the combination of taking a collision free path generated by the Rapidly-exploring Random Trees method, and using it as a basis to generate a Dubin's path with Fermat's spiral transitions for curvature continuity. The resulting path will then be used as input for a guidance system tracking that path while under the influence of ocean currents. The results are also compared with results from tracking a Dubin's path with clothoid transitions and concludes which method is the fastest. The thesis also contributes by using an alternative parametrization of the Fermat's spiral (which was suggested in Lekkas et al. (2013)) in order to achieve the tracking.

1.3 Outline

This thesis is divided into chapters which together aims to build a system using motion planning to generate paths as input for a guidance system tracking that path.

Chapter 2 provides theoretical background on path planning by formulating the path planning problem and explains the many approaches available. It also introduces shortest path algorithms and explains how obstacles can be added and avoided in a path planner algorithm.

In Chapter 3, three of the most common path planning methods in the field are presented and explained in detail. Chapter 3 also compares the methods for 2D scenarios.

Chapter 4 increases the complexity by introducing kinematics and differential constraints to be added to the Rapidly-exploring Random Trees method in Chapter 5. Chapter 5 also explains how to generate paths with different levels of smoothness.

Chapter 6 goes over the basic elements of a guidance system which will use the paths generated in Chapter 5 as input.

In Chapter 7, the results of using the paths from chapter 5 as input for the guidance system from Chapter 6 are presented and discussed.

Chapter 8 presents the final conclusion together with suggestions for future work.

1.4 Notations

α	$\triangleq$	Angle
P_s	$\triangleq$	Starting pose
P_f	$\triangleq$	Finishing pose
r	$\triangleq$	Path
q	$\triangleq$	Path parameter
$\mathbb{R}$	$\triangleq$	Euclidean space
X	$\triangleq$	Metric space
$\mathcal{W}$	$\triangleq$	Workspace
$\mathcal{O}$	$\triangleq$	Obstacle space
$\mathcal{C}$	$\triangleq$	Configuration space
R	$\triangleq$	Radius
$\subset$	$\triangleq$	Subset
$\in$	$\triangleq$	Included in
$\mathcal{A}$	$\triangleq$	Set of points occupied by a vehicle
κ	$\triangleq$	Curvature
c	$\triangleq$	Sharpness
ϕ	$\triangleq$	Steering angle
ϖ	$\triangleq$	Path parameter
$\forall$	$\triangleq$	For all
ψ	$\triangleq$	Vessel heading
ω	$\triangleq$	Angular velocity

1.5 Abbreviations

RRT	-	Rapidly-exploring random trees
PRM	-	Probabilistic roadmap method
DOF	-	Degrees of freedom
FS	-	Fermat's spiral
NED	-	North-East-Down
LOS	-	Line-of-sight
RSL	-	Right-Straight-Left

Chapter 2

Theoretical Background

This chapter provides a short introduction to path planning and different concepts in the field of path planning. Path planning uses concepts from many different fields of study. The most commonly used terminology comes from the field of robotics, where path planning has been an important part since the beginning. Other disciplines which have contributed to the field of path planning are computer science, mathematics, control theory and in more recent years autonomous offshore and aerial applications. For more in depth explanations of the concepts presented in this chapter, the reader is advised to look at the book from Tsourdos et al. (2010) as well as LaValle (2011a), LaValle (2011b) and Kavraki and LaValle (2008). Other recommended papers are Dahl (2013), Haugen (2010) and Loe (2008).

2.1 Path planning theory

2.1.1 Formulating a path planning problem

In Tsourdos et al. (2010), the path planning problem is formulated as follows: For any kind of vehicle or robot we have a starting pose P_s and a finishing pose P_f. Path planning then involves producing a path $r(q)$ connecting P_s and P_f,

$$P_s \xrightarrow{r(q)} P_f \tag{2.1}$$

where q in $r(q)$ is a path parameter which depends on the path formulation (straight-line or curved).
In most problems where path planning is applied there are constraints which has to be taken into account. Constraints can include obstacles in the region, vehicle limitations, communication limitations and different safety measures. In the simplest case constraints are represented as the symbol $\amalg$. (2.1) can then be written as:

$$P_s \xrightarrow{\amalg r(q)} P_f \tag{2.2}$$

From (2.2) we see that a path planner acts as a black box system, in the sense that it produces a path from given inputs. This analogy is pictured as a block diagram approach in Figure 2.1.

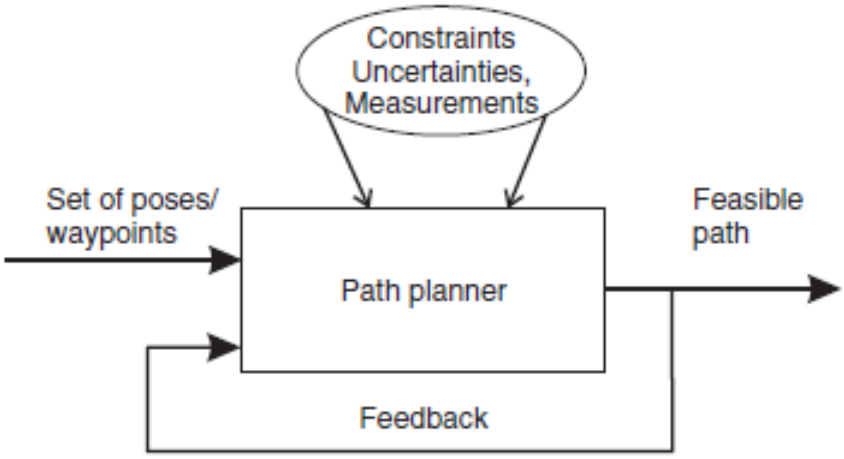


Figure 2.1: Block diagram approach to path planning, from Tsourdos et al. (2010)

The inputs are the poses P_s and P_f, and the output is the path $r(q)$ which is also fed back into the path planner to determine if the produced path meets the constraints.

2.1.2 Solving a path planning problem

In both Tsourdos et al. (2010) and LaValle (2011a) the basic concepts of a solver for the path planning problem are explained in depth. A path planners task is to produce a series of nodes,

connected by straight lines, which shows the path from start to finish. This is called a global path planner. In turn the series of nodes can be optimised and used as input for a vehicles controller. For complex problems, some systems also include a local path planner, which provides a more detailed solution.

There are a wide variety of approaches for a path planner to create a solution. The approach can be split into two parts, first transforming the given environment into a searchable database, and second finding a suitable algorithm to compute the path into a list of nodes.

Translating the environment is dominated by three methods documented by Latombe (1991) as the potential field method, cell decomposition method and roadmap method. These methods are also explained in Tsourdos et al. (2010).

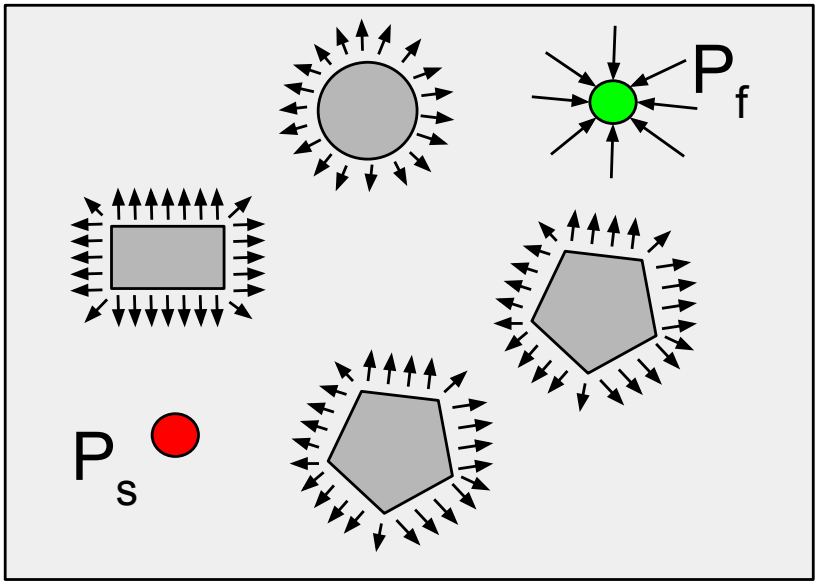


Figure 2.2: Potential field example

The **potential field method** sees the environment as an artificial potential field, where destinations are seen as attractive, and obstacles are seen as repulsive (see Figure 2.2). This continuous method will in the end produce a path which follows a line of maximum potential, and is therefore vulnerable to getting trapped in a local maximum.

The **cell decomposition method** divides the environment into a cell structure with free cells and occupied cells, as can be seen in Figure 2.3. This discrete representation can then be used as input for a solver finding a path consisting of free cells.

The **roadmap method** will be covered in more detail as we get further into the theory, but to

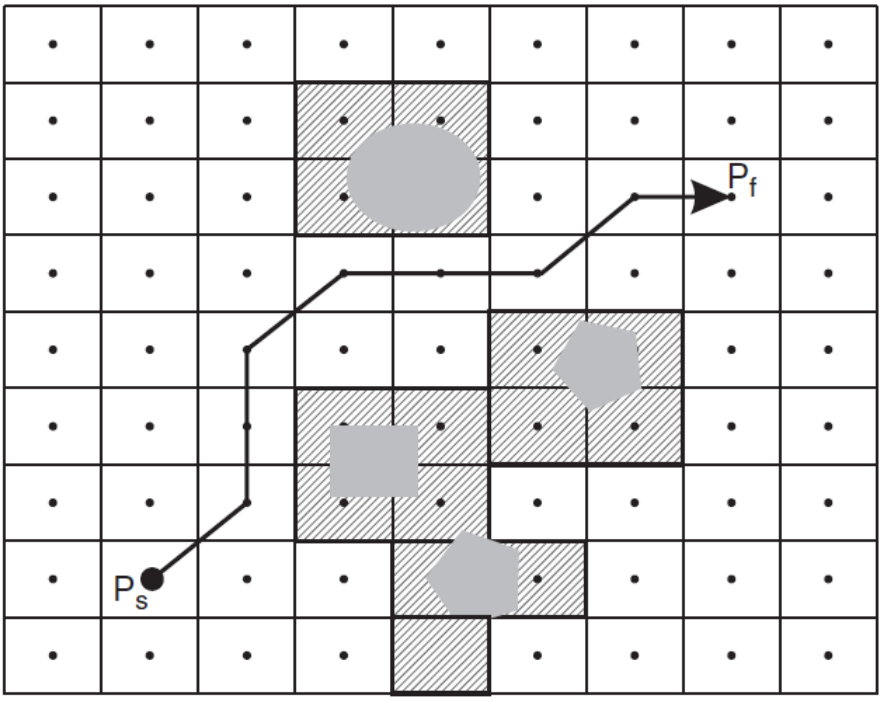


Figure 2.3: Cell decomposition example, from Tsourdos et al. (2010)

understand how road maps work, the space in which it works in must be explained.

2.1.3 The configuration space and workspace

The configuration space or C-space is a complete specification of the position of every point in a system. In a simple case, the position of a particle in a 3D space can be defined by the vector $r(x, y, z)$. The configuration space of the particle is therefore $\mathbb{R}^3$.

A workspace however, is modified to accommodate for the physical size of the vehicle.

To better illustrate the configuration space, an example from LaValle (2011a) and Kavraki and LaValle (2008) is considered. A 2D workspace denoted $\mathcal{W} = \mathbb{R}^2$. The workspace has obstacles $\mathcal{O} \subset \mathcal{W}$. The goal is create a path for a vehicle which occupies a closed set of points $\mathcal{A}(q) \subset \mathcal{W}$ from an inital configuration q_i to a goal configuration q_G. Assuming the vehicle is unable to rotate the configuration space will be equal to the workspace $\mathcal{C} = \mathcal{W} = \mathbb{R}^2$. Every time the vehicle changes configuration it must be without colliding with an obstacle, this noncolliding space can be defined as

$$\mathcal{C}_{free} = \{q \in \mathcal{C} | \mathcal{A}(q) \cap \mathcal{O} = \emptyset\} \tag{2.3}$$

and is called the free space. Hence the colliding space is $\mathcal{C}_{obs} = \mathcal{C}/\mathcal{C}_{free}$. With this description the path-planning problem can be visualised as seen in Figure 2.4.

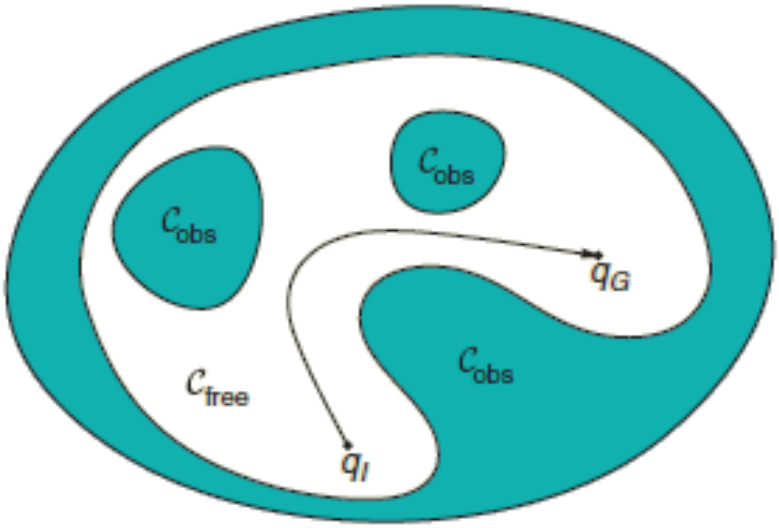


Figure 2.4: Connecting q_i to q_G while remaining in $\mathcal{C}_{free}$, from LaValle (2011a)

2.1.4 Roadmaps

A roadmap is a network of straight lines (2D or 3D) connecting the starting point of a vehicle and the goal without colliding with any defined obstacles in the environment. Since the roadmap works in the configuration space, the vehicle is considered (as mentioned in the Section 3.2) to be a point, and the space takes the vehicles dimensions into account when defining $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$.

To better understand a roadmap, a simple example is considered in Tsourdos et al. (2010). After the environment has been transformed into $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ a network of straight lines are generated, creating a graph which results in a set of polygons surrounding the obstacles on the map. The result can be seen in Figure 2.5.

2.1.5 Planning algorithms

After a path planning problem has been translated by a path planner into a suitable configuration space, a planning algorithm can be applied. The algorithm requires input in the form of the vehicle's initial placement, the desired goal placements, and a description of the vehicle itself and the workspace. The goal is then to deliver a precise description of how the vehicle can move from start to finish. As all algorithms, a planning algorithm works step-by-step when calculating a path. When receiving a workspace with known points or paths all ready defined, simple search algorithms can be used to calculate the resulting path. But in other cases the algorithms has to place out the points and paths by itself, incrementally getting closer to the goal.

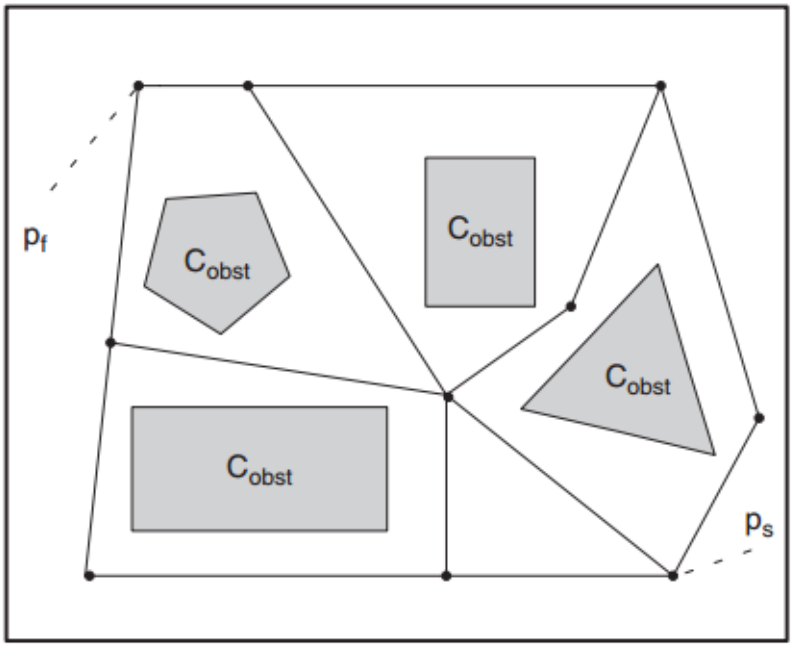


Figure 2.5: Starting point p_s connected to a goal point p_f using the roadmap method taken from Tsourdos et al. (2010)

2.1.6 Global and local methods

In this report the maps are without dynamic constraints. Thus the only methods needed are considered global methods. A global method uses global information to save the environment in a map, which is used during the entire calculation of the path planner. This means the global map does not change, and exists in the memory available to the planner. The computational time of a global method is often long, as static environments does not require quick reactions in common cases.

However, if the environment is to include dynamic constraints, such as other vehicles acting as moving obstacles, a more precise approach is needed. This is where a local method can be introduced. A local method only operates within a small area of the environment, and only considers one situation at a time, making it much faster than a global method. Local methods can be simplified version of the global methods, as long as they fulfil the demands required. Local methods work off sensors and are therefore memoryless.

In large areas filled with both dynamic and static obstacles, a combination of both a global and a local path planner is the most beneficial way to design a path planner.

2.2 Dijkstra's algorithm

Known as one of the most famous shortest path graph search methods in computer science, Dijkstra's algorithm from Dijkstra (1959) assigns every node in a graph a tentative distance

value (initially all nodes are set to infinity and the starting node as zero). It then marks all nodes except x_{init} as unvisited, and sets x_{init} as the current node. The algorithm then considers the current nodes unvisited neighbours by calculating their tentative distance and comparing it to their previous value, and updates it if a lower value is calculated. When all neighbours are visited, they are no longer marked unvisited. After each loop of visiting neighbours is complete the unvisited node marked with the smallest tentative distance is set as the new current node. When the destination has been marked as visited, or there exists no connection between the visited and unvisited nodes, the algorithm is finished.

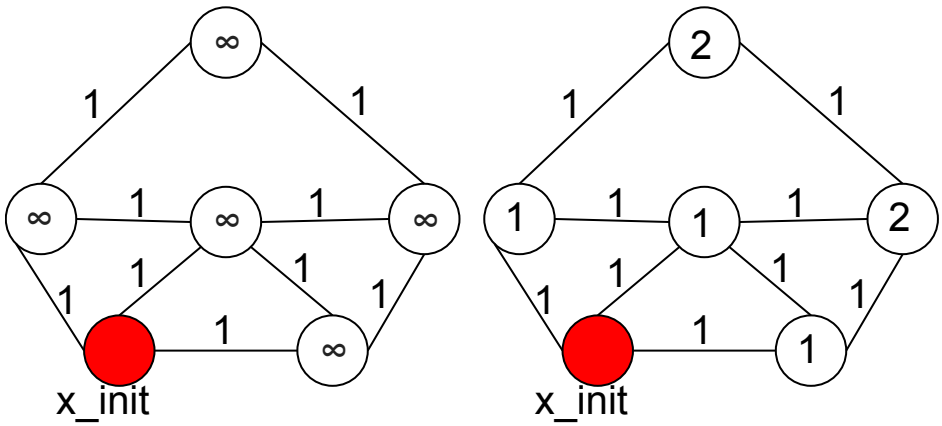


Figure 2.6: Dijkstra's algorithm initially and finished

2.3 Obstacles

Since this report mostly explores path planning in two dimensions, 2D obstacles is naturally what we want to avoid when finding a path from start to finish. In 2D cases obstacles can be pictured as polygons which makes avoiding them a simple thing to implement. Another advantage of using polygons is that they are easy to increase in size if a clearance constraint is also a wanted feature. The clearance constraint concept will be explained in Chapter 4.

2.3.1 Placing points

Having the obstacles as polygons is an advantage when a path planner is placing out points to explore, as the simple Point-in-polygon method (inpolygon, see MathWorks (2014b)) can be applied. Point-in-polygon determines whether a given point is inside or on the edge of a polygonal region, and returns either true or false. This is pictured in Figure 2.7 where the blue points can be used by planner, and the red points will be discarded.

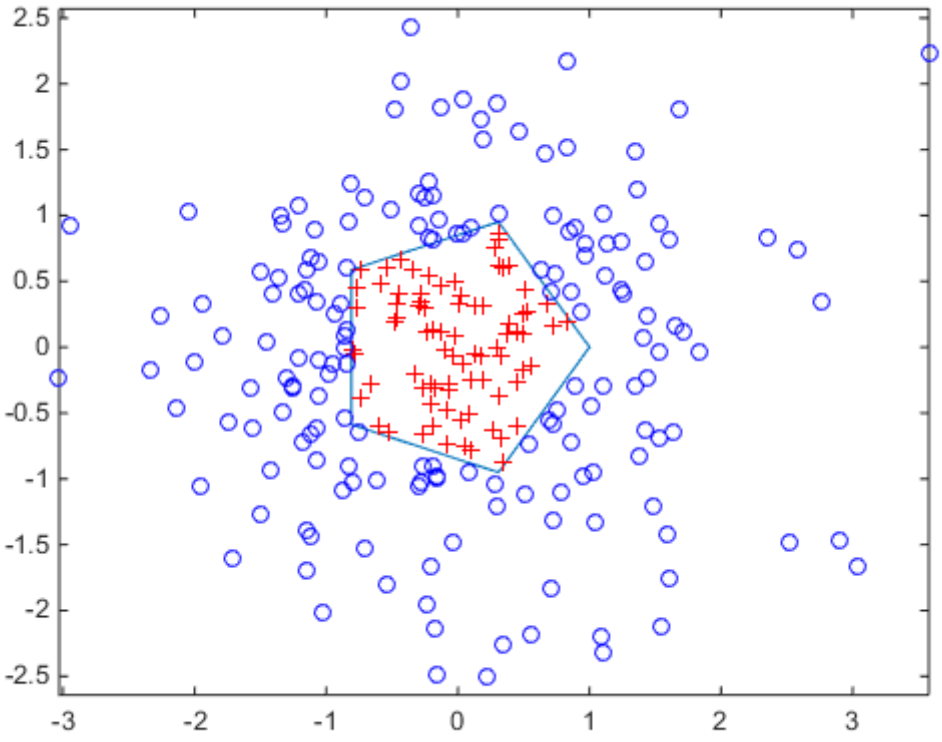


Figure 2.7: Placing points and deciding whether they are inside or outside a polygonal region, from MathWorks (2014b)

2.3.2 Checking edges

When finding out if a generated edge collides with an obstacle, it is once again favourable to have the obstacles defined as polygons. MATLAB's polyxpoly function (see MathWorks (2014c)) creates a set containing all intersecting points between the given edge and the obstacle. Checking if this set is empty afterwards determines whether the edge collides with the obstacle or not.

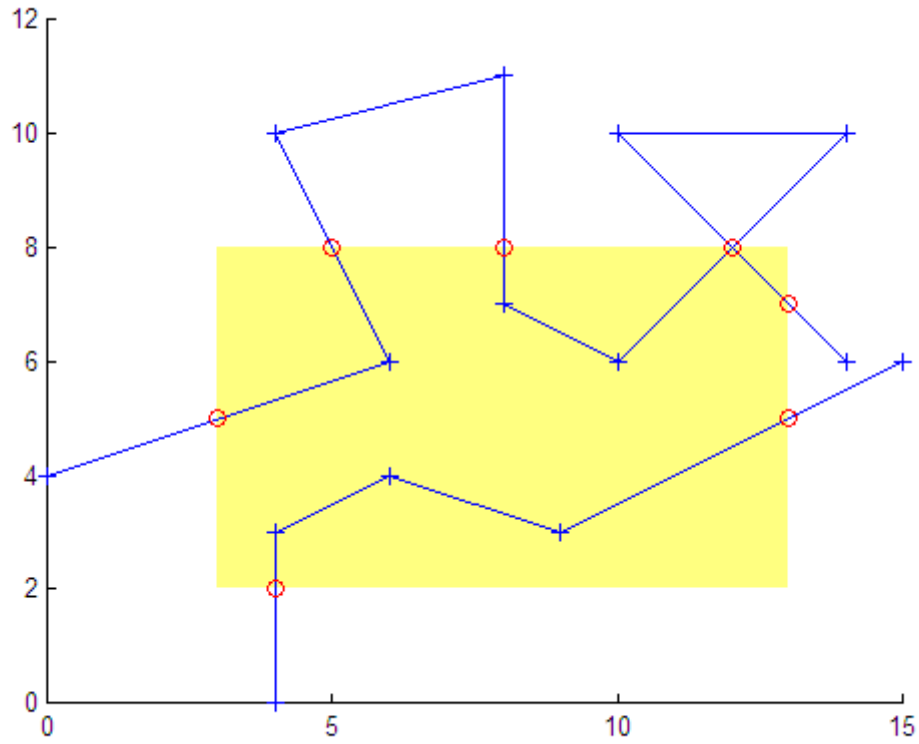


Figure 2.8: Finding the intersecting points between a line and a rectangle, from MathWorks (2014c)

Chapter 3

Path Planning Methods

This chapter introduces three common path planning methods and then compares them in a two dimensional environment. The methods are the Voronoi diagram method, first introduced in Voronoi (1908), the Probabilistic road map method, first introduced in Kavraki et al. (1996), and the Rapidly-exploring random trees method introduced in LaValle (1998).

Path planning is often split into two types, combinatorial and sampling-based. In the book LaValle (2006) combinatorial planning is described as a methods which find paths through the continuous configuration space without resorting to approximations. While sampling-based methods probes the configuration space using a sampling-scheme. Thus very complex geometric principles can be simplified, and paths can be found quickly.

For the voronoi diagram method Lekkas (2014) includes a good literary review of the method. For the RRT LaValle (1998), LaValle and Kuffner Jr (2000a), LaValle and Kuffner Jr (2000b) and LaValle and Kuffner (2001) are recommended to get a good introduction of the method, and for some of the methods uses see Pepy et al. (2006), Xinggang et al. (2014), and Heo and Chung (2013). For path planning method design in general Sucan et al. (2012) is recommended.

3.1 The Voronoi diagram method

The Voronoi diagram method is named after the ukranian mathematician Voronoi who defined and studied the n-dimensional case of the method in Voronoi (1908). Originally, the method dates back as far as Descartes who used it in 1644. The method can bee seen as partitioning a plane into regions based on a set of points, and is visualised in its simplest form in Figure 3.1. When applied to a path planning problem, the set of points will represent obstacles.

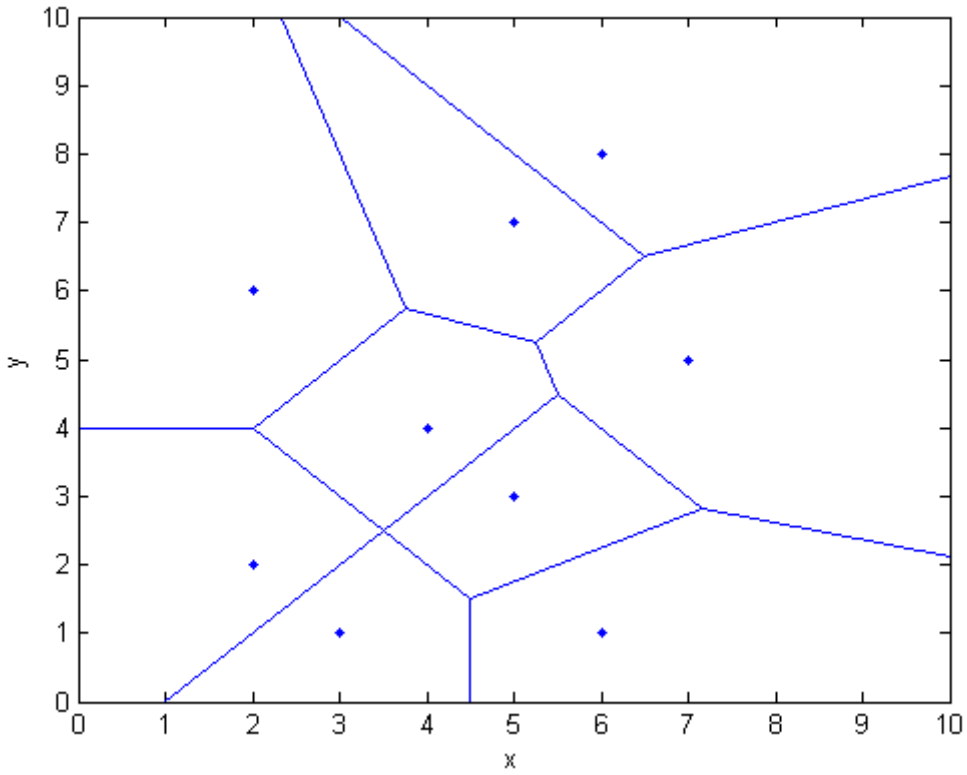


Figure 3.1: The Voronoi diagram of a set of 9 points

3.1.1 The basics of Voronoi diagrams

The Voronoi diagram is based on a finite set of points $P = \{p_1, ..., p_2\}$ contained in a workspace X. In this report we are working in $X \subset \mathbb{R}^2$. The space X has a metric function defined as $d(\cdot)$. When the space and points have been defined the method starts by associating a Voronoi region R_i for each point $p_i \in P \subset X$, where R_i is defined as the set of points $x_i \in X$ such that their distance to p_i is lower than the distance from x_i to any other point of P. Mathematically speaking the metric is defined as:

$$d(x, p) = \inf\{d(x, p)|p \in P\} \tag{3.1}$$

such that the Voronoi region is defined as:

$$R_k = \{x \in X | d(x, P_k) \leq d(x, P_j) \forall j \neq k\} \tag{3.2}$$

In the end the final Voronoi diagram is the tuple of cells $(R_k)_{k \in K}$, with K as a set of indices.

3.1.2 Voronoi diagrams for path planning

As mentioned earlier, the Voronoi diagram can be used to solve a path planning problem. When doing so, the points used as inputs are put on the edges of the obstacles, to create collision free edges for the starting and finish point to connect to, and then the resulting path will be along the edges. In this section a step-by-step explanation of how the Voronoi works is presented.

3.1.3 Placing points

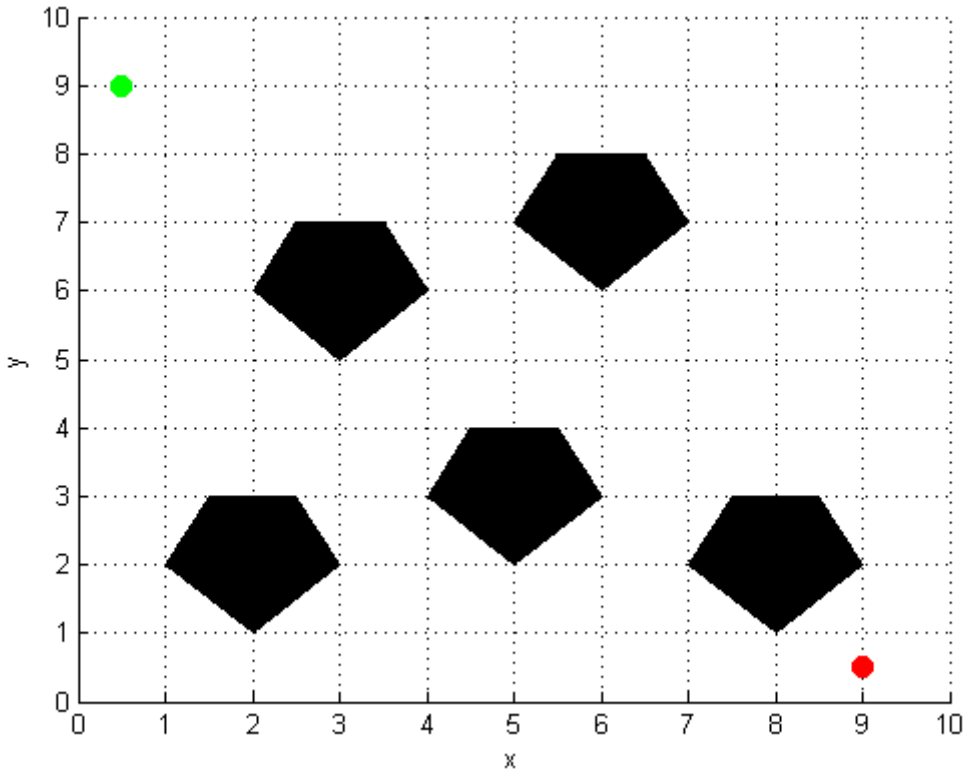


Figure 3.2: A path planning scenario for the Voronoi method

Figure 3.2 shows a map containing five obstacles, a start point and a finish point. To convert this to an input for the Voronoi diagram method the sides of the obstacles must be discretized into points, and the area must be contained. In this case using the corners of the obstacles are

enough, but with larger more complex obstacles a large amount of points must be provided to get a reliable result.

3.1.4 Placing the Voronoi diagram

When the input points have been set, they can be fed into the Voronoi diagram function. In this case the built-in MATLAB function voronoi was used MathWorks (2014d). This function requires two vectors as input. One with the x-coordinates and the other with the y-coordinates. When executed the result is as shown on Figure 3.3. The blue points are the input points, and the blue lines are the edges of the polygons created by the method.

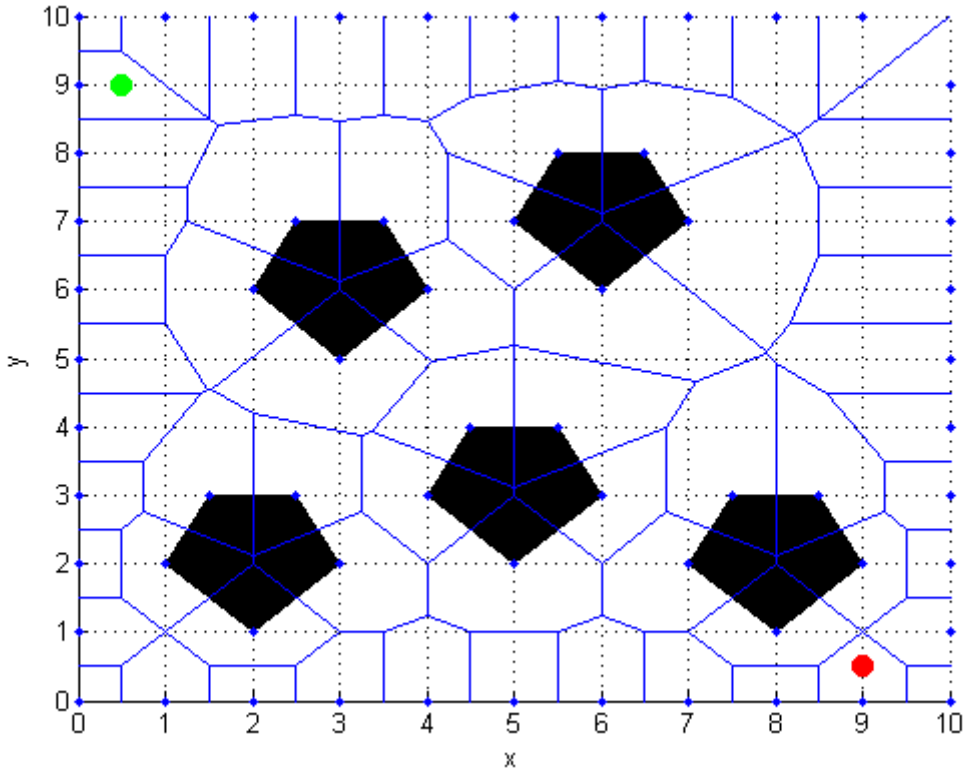


Figure 3.3: Resulting Voronoi diagram for Figure 3.2

3.1.5 Filtering edges within obstacles

When the diagram has been created, the next step is to remove the edges and points that lead out of the area or go through obstacles. This is achieved by going through each edge and removing them if some part of it is within an obstacle or at the border of the area. When finished, a collision free network has been generated, see Figure 3.4.

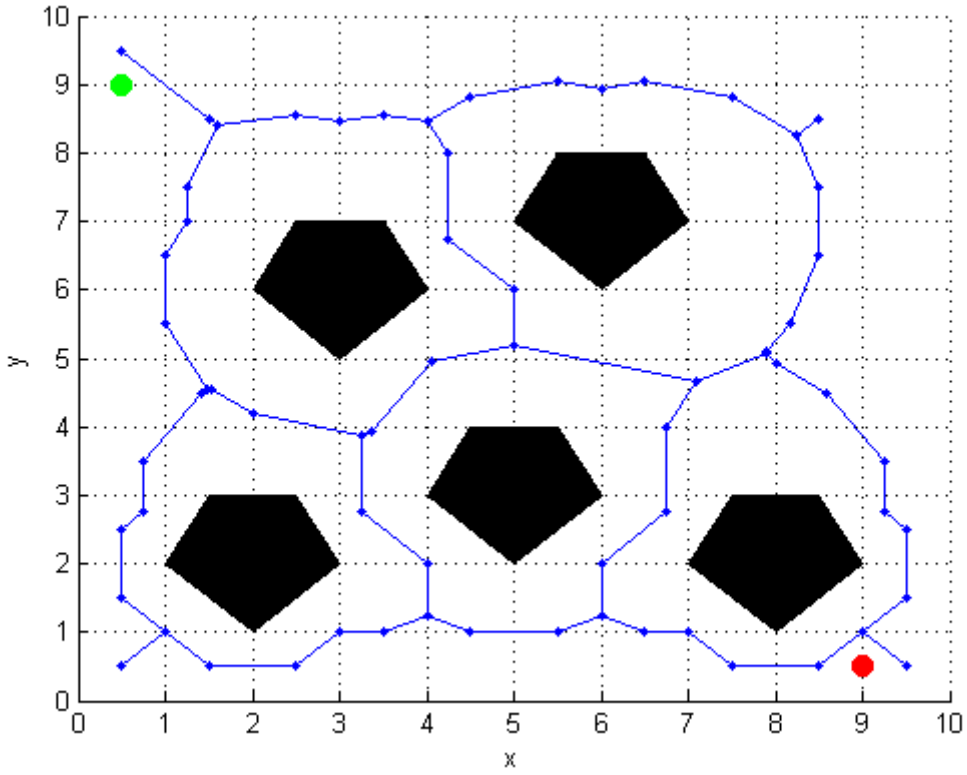


Figure 3.4: Figure 3.3 after removing edges and points within obstacles

3.1.6 Connecting to the network and finding a path

The final step of the method is to connect the starting and finishing point to the network and applying a shortest path algorithm such as Dijkstra's. The initial path will follow the Voronoi edges, but by applying a waypoint reduction a final path can consist of very few points. Figure 3.5 shows how the problem in Figure 3.2 has been solved.

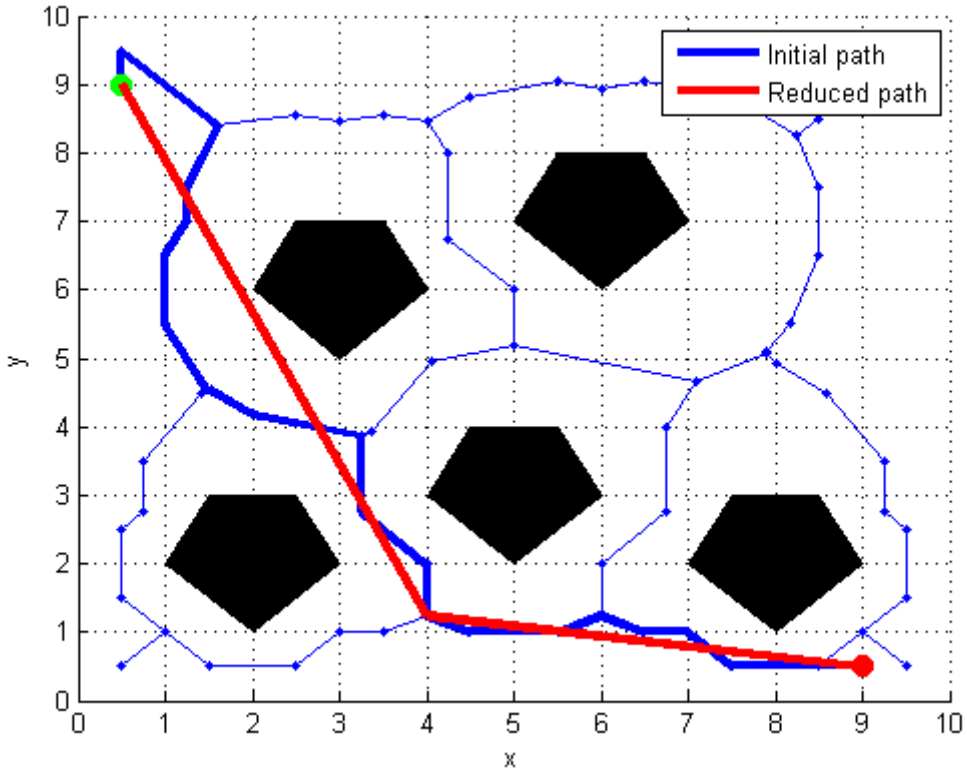


Figure 3.5: Figure 3.4 after applying Dijkstra's algorithm and waypoint reduction

3.2 The Probabilistic Roadmap method

The probabilistic road map method was first introduced in Kavraki et al. (1996). It is a sampling based method which computes collision-free paths by generating random states in the configuration space, checking if they are in $\mathcal{C}_{free}$, and applies a local planner to connect neighbouring states to each other. In the end there will be a graph including all generated states which the states x_{init} and x_{goal} can be connected to, and a shortest path algorithm can be applied to find the resulting path.

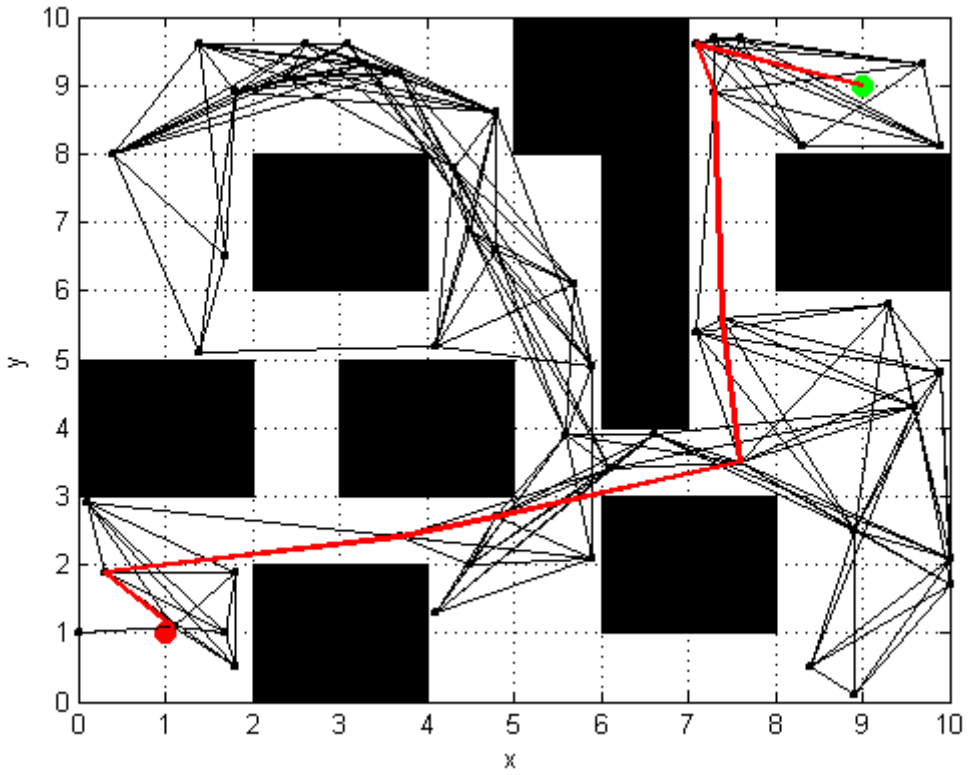


Figure 3.6: Probabilistic road map method

3.2.1 The basics of PRM

When applying the PRM method, the first task is to create the space in which the method will work. When using PRM three spaces are needed; the workspace $\mathcal{W}$, which in most cases is the same as the configuration space $\mathcal{C}$, the collision free part of $\mathcal{C}$, namely $\mathcal{C}_{free}$, and the obstacle space $\mathcal{C}_{obs}$. When these are set a roadmap is generated as an undirected graph $R = (N, E)$. The graph includes the nodes N which are a set of randomly chosen configurations, or points, in the free space $\mathcal{C}_{free}$. E are the possible connections between nodes, called edges. This is called the learning phase of the PRM. After the roadmap is complete, the query phase takes over and

connects x_{init} and x_{goal} to the roadmap and calculates a feasible path. The two phases can be intervowen, but are best explained when separated.

Learning phase

As mentioned, the goal of the learning phase is to create a roadmap over a given workspace. The phase starts by constructing the roadmap. The nodes N of the initially empty graph $R = (N, E)$ are repeatedly updated with additional random free configurations called c. For every new node generated, a neighbourhood set is generated, $N_c \subset N$, which contains nearby existing nodes within a chosen radius. Then the newly created node tries to connect to its neighbours, and if successful, the edge is added to E. For a more in-depth explanation of how the neighbourhood set is chosen see Kavraki et al. (1996). When the number of nodes added to the graph is satisfying the learning phase is over and a roadmap has been created. An example of a finished learning phase can be seen in Figure 3.7. Here the starting and ending points are represented as the green and red dot, respectively.

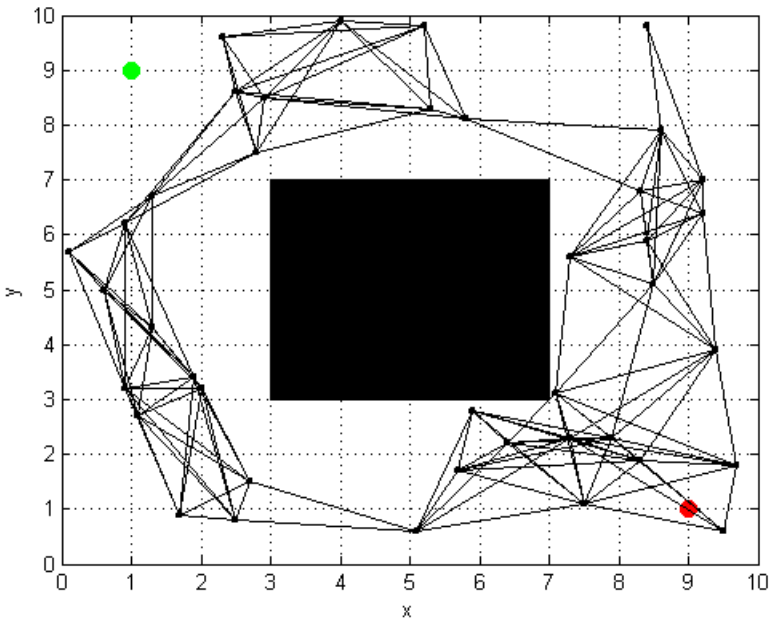


Figure 3.7: PRM after the learning phase

Query phase

After the learning phase is complete, the query phase takes over. Here the goal is to connect the initial point and the goal point to the roadmap. This is done in a similar way to how new nodes are added in the learning phase. However the neighbourhood nodes can be chosen with a different radius, and other constraints. In the general case, the starting point and goal point connect to the first point available, which may not be the best. This can be countered with

waypoint reduction of the finalized path. Figure 3.8 shows how the query phase has connected both points to the roadmap generated in Figure 3.7.

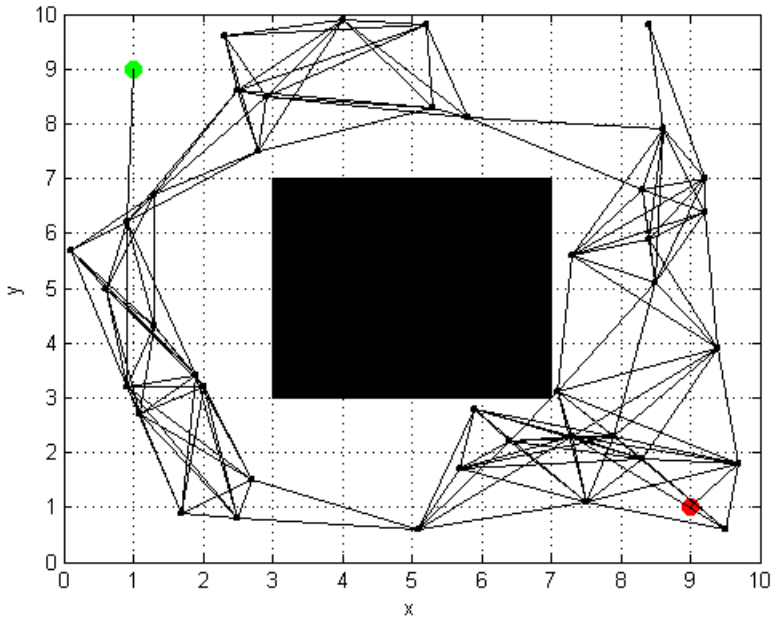


Figure 3.8: PRM after the query phase

PRM finding goal

When the graph $R = (N, E)$ has been updated with the start and finishing point including the edges from the query phase, a simple graph shortest path algorithm can be applied to find a path. This is illustrated in Figure 3.9 as the red line connecting start to finish.

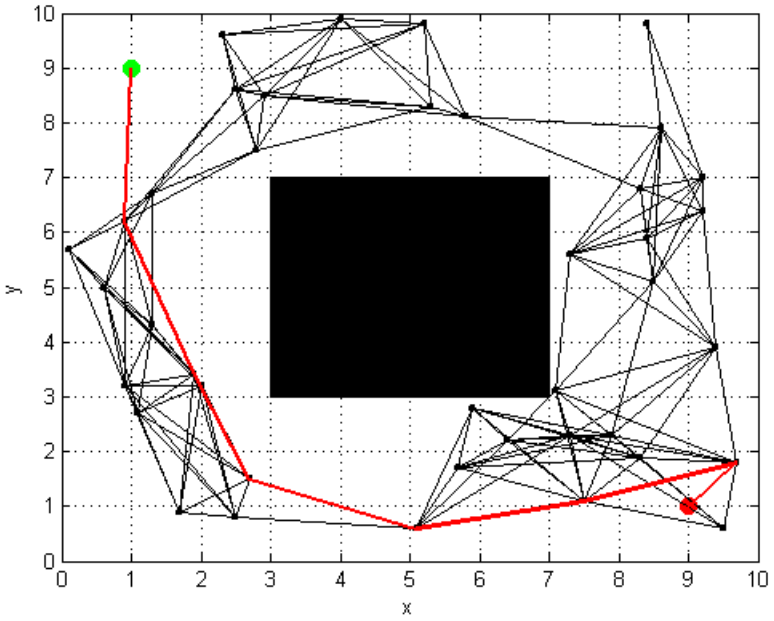


Figure 3.9: PRM after applying shortest path algorithm

3.2.2 The PRM algorithm

The probabilistic roadmap method is relatively easy to implement, and programs such as MATLAB have many functions which shortens the process. Below in Algorithm 1, a pseudo code based on a MATLAB implementation is presented. The code starts out by initialising the configuration space $\mathcal{C}$, and generates an empty graph called R. Then the learning phase starts at line 3 by iteratively creating up to K nodes in $\mathcal{C}_{free}$. For every iteration a node c is generated randomly in $\mathcal{C}_{free}$ and saved in R. Then a neighbourhood N_c is created by defining a radius, d, around c. For each neighbour n in N_c, c checks if a connection is possible and that the edge doesn't all ready exist. If that is the case, an edge between c and n is saved in R. The learning phase ends at line 13 when K nodes have been added to R and all possible edges have been found.

When the learning phase is done, the query phase tries to connect the start and finishing point to R. If this is possible for both points, a graph shortest path method is applied to the finished roadmap, and a path is generated. If no connection is possible, the method fails.

3.2.3 PRM extensions

Combining learning and query phase

One way to improve the PRM is to combine the learning and query phase so they are applied together. This extension makes the method more reliable, in the way that if the query phase fails

```
initialize(C, x_init, x_goal, N, E);
R ← ({N, E});
for k = 1 to K do
    c ← RANDOM_STATE(C_free);
    N_c ← GENERATE_NEIGHBOURHOOD(c, N, d);
    N ← N ∪ {c};
    while n ∈ N_c do
        if CAN_CONNECT(c,n) and NOT_CONNECTED(c,n) then
            N ← N ∪ {(c, n)};
        end
    end
    k = k + 1;
end
if CONNECT(x_init,x_goal,R,d) then
    graphshortestpath(R,x_init,x_goal);
else
    return failure;
end
```

Algorithm 1: PRM

in connecting the starting and finishing point to the roadmap, the learning phase can continue making an even larger roadmap until the query phase finds its solution.

Waypoint reduction

Waypoint reduction is a simple technique to apply to the initial path generated by the PRM. It both reduces the length and the complexity of the path. The method uses the points and edges of the initial path as input, and works through the path by continuously attempting to connect points that are not already connected, and deleting the middle points if a connection is possible. An example can be seen in Figure 3.10.

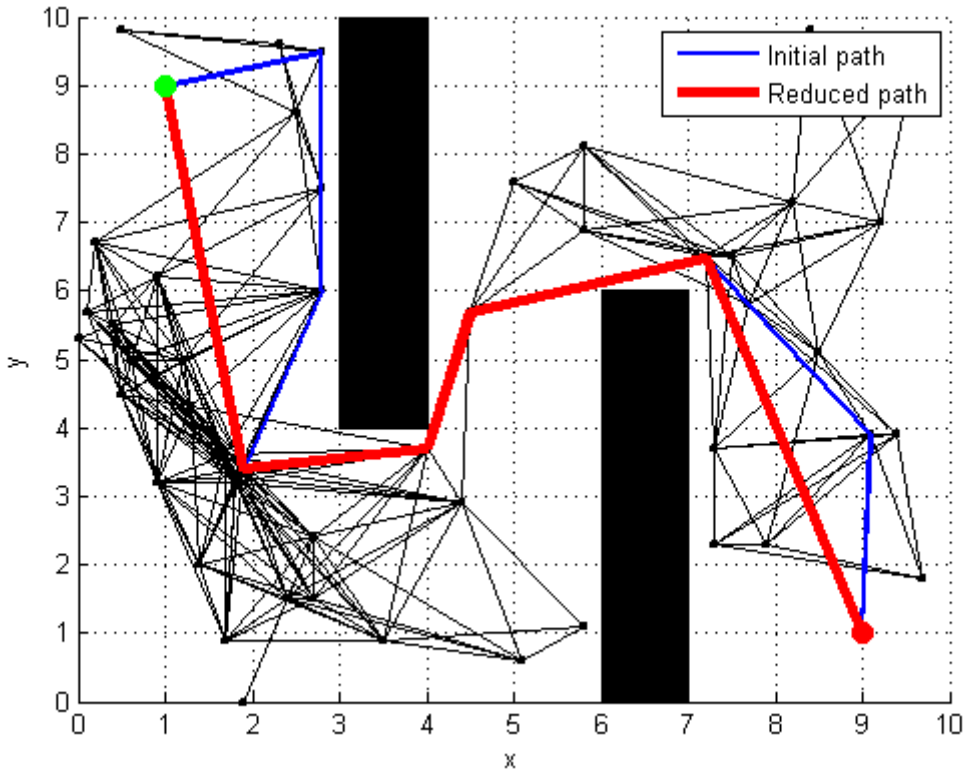


Figure 3.10: Waypoint reduction for the PRM

3.3 The Rapidly-exploring Random Trees method

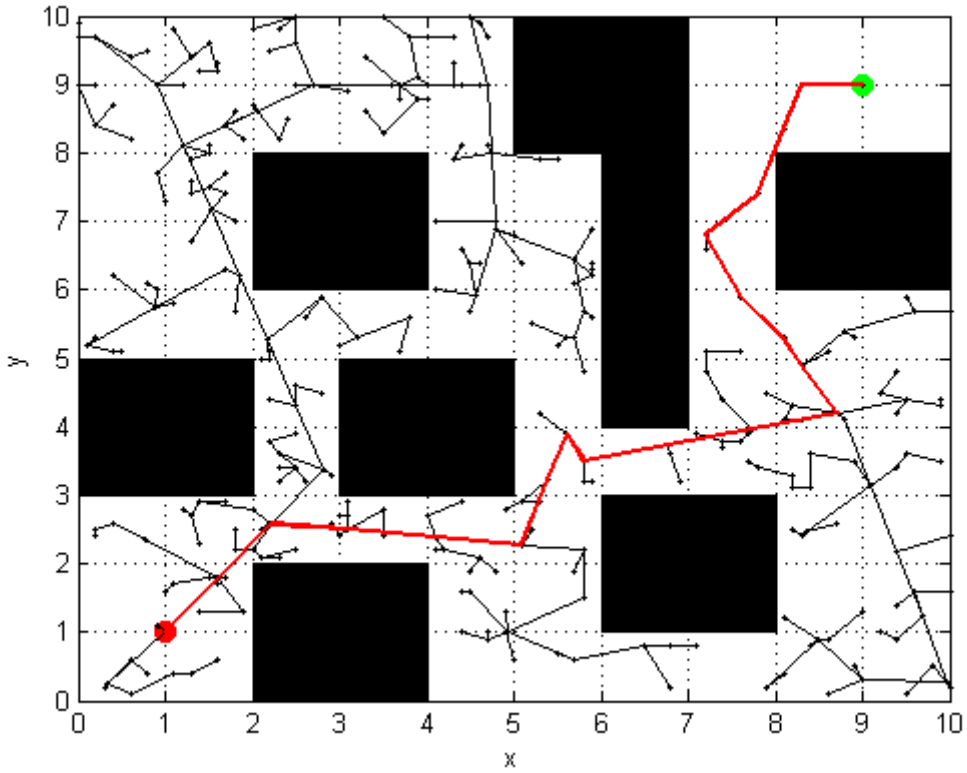
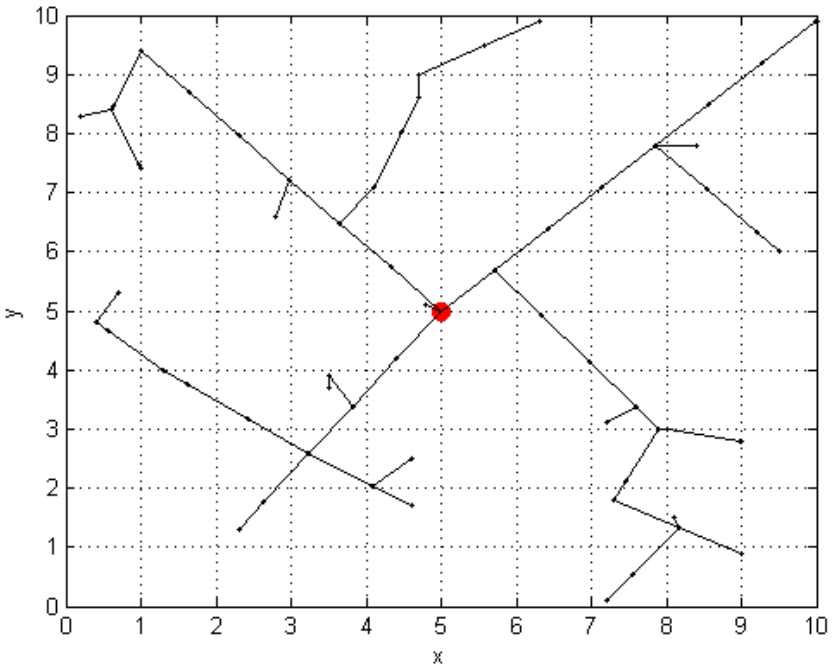
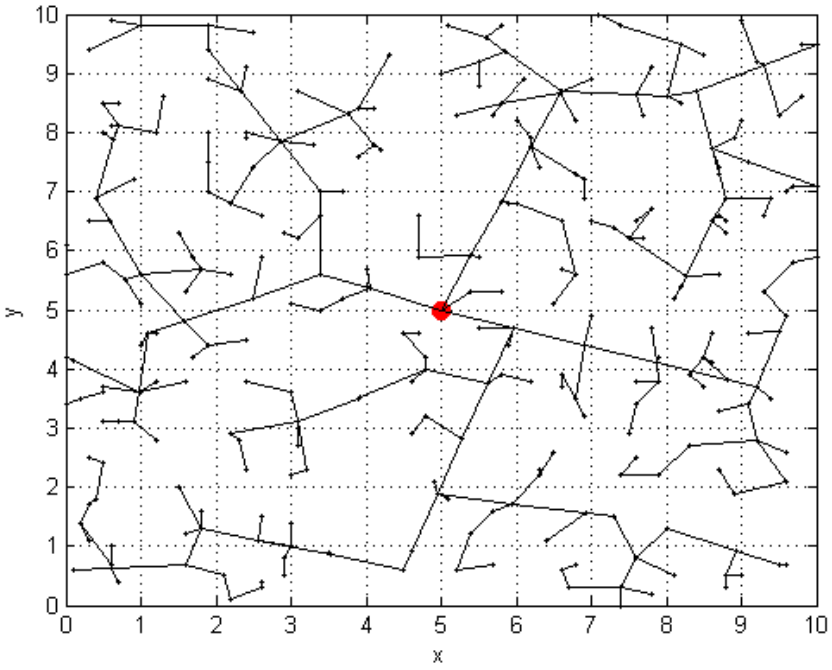


Figure 3.11: Rapidly exploring random trees method

The Rapidly-exploring random trees (RRT) algorithm was introduced in LaValle (1998). It is a randomised method designed for problems with nonholonomic constraints. In other words it can create a path while considering the dynamics of a vehicle at the same time. The randomness of the RRT gives it the property of being able to explore a space and find solutions much faster than a complete path planning method. The property of randomness makes the method favour exploring unexplored areas, as well as increasing the chance of reaching the goal for each iteration. A simple MATLAB implementation shows the exploring properties of the RRT in environments with and without obstacles in Figure 3.12 and 3.13.

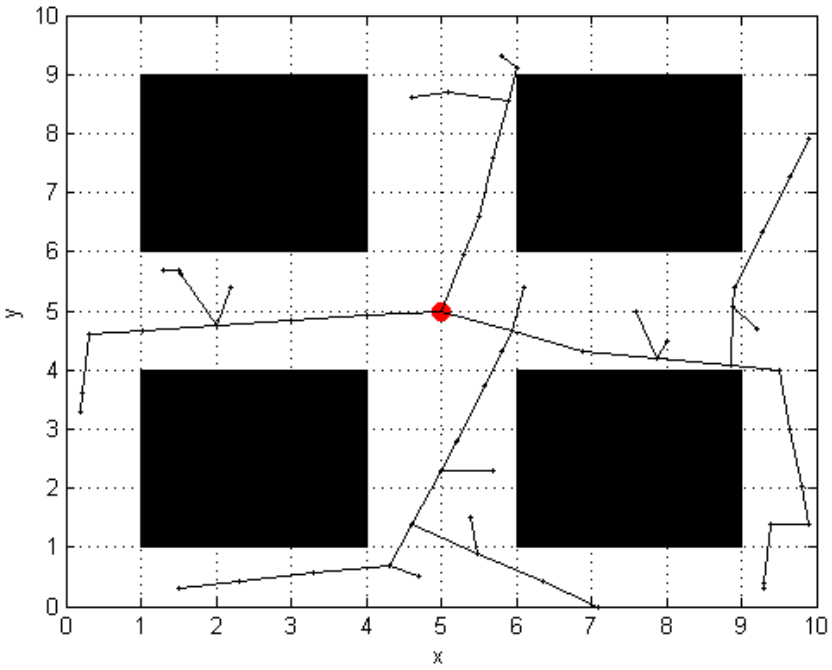


(a) 30 iterations

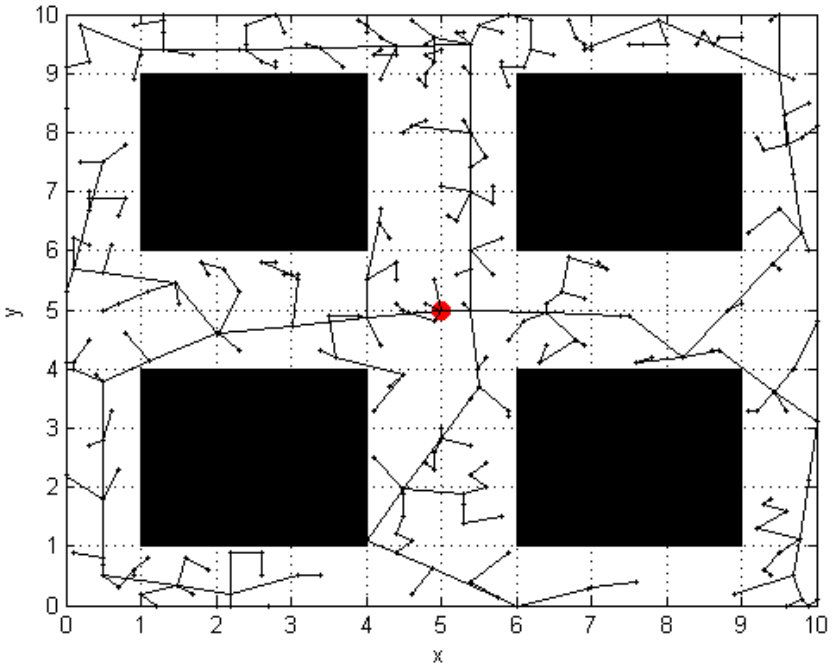


(b) 250 iterations

Figure 3.12: RRT exploring an area w/o obstacles



(a) 30 iterations



(b) 250 iterations

Figure 3.13: RRT exploring an area with obstacles

3.3.1 The basics of RRT

The basis of the RRT algorithm is to generate a tree with the vehicle'ss initial state as the starting node. The tree grows by placing nodes randomly in C_{free} and checking if the tree's existing nodes can connect to them without colliding with C_{obs}. As the tree grows it will eventually reach the goal point and a guaranteed collision free path has been made. The RRT is able to take the vehicle's constraints into consideration when checking whether a new node can be connected to the existing ones. Ultimately the result will be a collision free and runnable path.

To visualise the method an example can be seen in Figure 3.14. A random state has been found in C_{free} as q_{rand}, and q_{near} has been found as the closest node from the expanding tree, T. Now the RRT uses an extend function towards q_{rand} by moving in small steps (relative to the size of the space), creating a new node in the tree for each step as q_{new}. After each step the extension function either continues, reaches q_{rand} or gets trapped. Afterwards, if the goal hasn't been reached, a new q_{rand} is generated, and the RRT continues until a goal has been found.

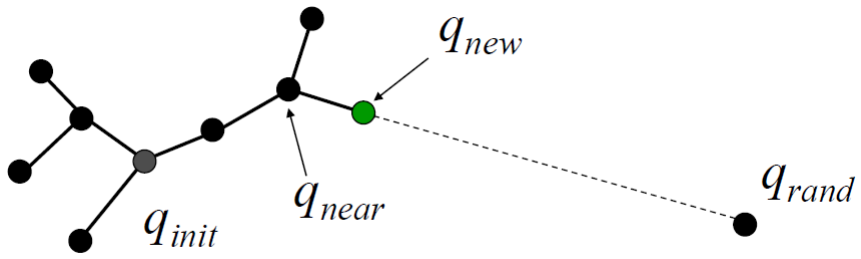


Figure 3.14: RRT extending, from LaValle and Kuffner Jr (2000a)

3.3.2 The RRT algorithm

One of the strengths of the RRT method is that it's relatively simple to implement. A pseudo code based of the implementation which Figure 3.12 and 3.13 was made from is presented in Algorithm 2, it is based on the theory from LaValle and Kuffner Jr (2000a). The algorithm runs in a loop of N iterations extending towards random states created by the RANDOM_STATE function, which randomly generates points in C_{free}. The algorithm then finds the newly generated point with NEAREST_NEIGHBOUR by using a minimum distance function to find the closest point to x_{rand}, called x_{near} from the existing tree, $\mathcal{T}$. An input u is generated by SELECT_INPUT and used to move safely towards x_{rand}. The length of u is determined by a desired step length and the limits of the area, and the resulting point is generated by NEW_STATE and saved as x_{new}. CAN_CONNECT then checks if the new point, x_{new}, can be added to $\mathcal{T}$, and if it can, the vertex continues to extend until x_{rand} is reached, or an obstacle is met. The method of extending this way is the same as in Figure 3.14 where q represents x.

```
initialize(x_init, C);
T ← TREE({x_init, 0});
for k = 1 to N do
    x_rand ← RANDOM_STATE(C_free);
    x_near ← NEAREST_NEIGHBOUR(x_rand, T);
    u ← SELECT_INPUT(x_rand, x_near);
    x_new ← NEW_STATE(x_near, u);
    while CAN_CONNECT(x_new) do
        T ← x_new;
        u ← SELECT_INPUT(x_rand, x_new);
        x_new ← NEW_STATE(x_new, u);
        if x_new == x_rand then
            break;
        end
    end
    k = k + 1;
end
```

Algorithm 2: RRT exploring for N iterations

3.3.3 RRT finding goal

Extending the existing pseudo code from Algorithm 2 to include a goal condition is simple. For each new node connected to the tree $\mathcal{T}$, a check can be added comparing the new node to the goal point, which when yielding a matching result, stops the algorithm. Now a simple shortest path search algorithm for a graph can be applied, in the case of Figure 3.15 and the other RRT results in this report, MATLAB R2014a's own shortest path function for graphs was used. The function is explained in the MATLAB documentation found on their web page MathWorks (2014a). The default search method of the function is the Dijkstra algorithm.

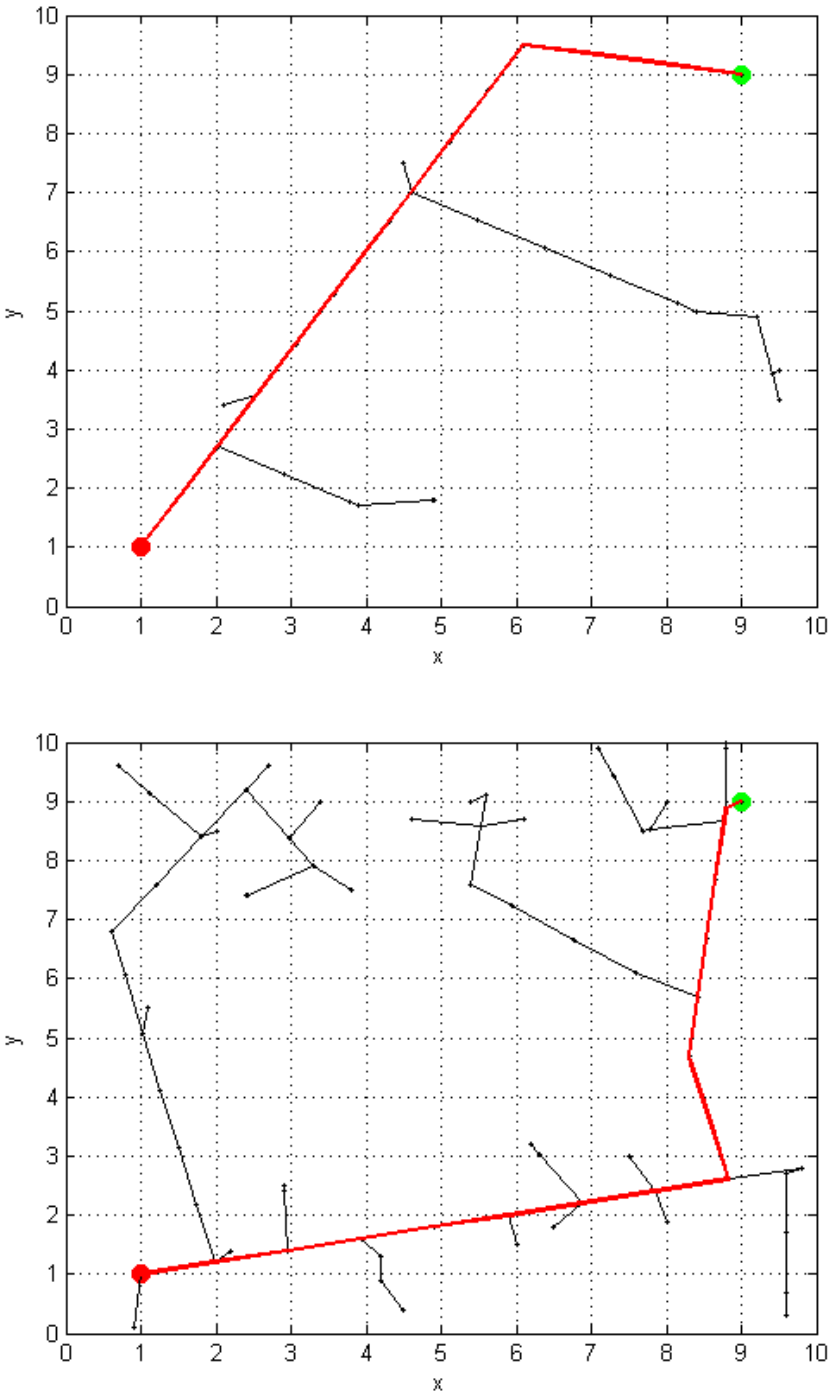


Figure 3.15: RRT finding goal in an obstacle-free area

3.3.4 RRT theory

Mathematical definition

The standard RRT presented in this report is working in a metric space X which is equal to the configuration space $\mathcal{C}$. This means the obstacle region can be noted as $X_{free} \subset X$. All the vertices the RRT creates are states in X_{free}, which means all the edges generated in the tree corresponds to a path which lies entirely in X_{free}. The method can then be formulated in six mathematical steps as seen in LaValle and Kuffner Jr (2000b):

1. **Creating a State Space:** Metric space, X_{free}.
2. **Defining Boundary Values:** $x_{init} \in X$, $x_{goal} \subset X$.
3. **Detecting Collisions:** $D : X \to \{true, false\}$, determines whether global constraints are satisfied.
4. **Generating Input:** A set, U, specifying the complete set of controls or actions that can affect the state.
5. **Incremental Simulator:** Given the current state, $x(t)$, and inputs from U applied over a time interval, $\{u(t')|t \leq t' \leq t + \Delta t$, compute $x(t + \Delta t)$.
6. **Metric:** A real valued function, $\rho : X \times X \to [0, \infty)$, specifying the distance between pairs of points in X.

Analysis

In the paper LaValle and Kuffner Jr (2000b), LaValle presents several lemmas and theorems on the properties of the RRT method. One of the results is that the distribution of the RRT vertices converge towards the sampling distribution, meaning that the RRT will come arbitrarily close to any point in a convex space. He also establishes probabilistic completeness, meaning that as more work is performed, the probability that a planner will fail in finding a path, asymptotically approaches zero. This is measured in rate of convergence, which obtaining theoretical bounds for, is described in LaValle and Kuffner Jr (2000b) as one of the key difficulties for RRT-based planners.

Proportionality with Voronoi regions

In LaValle and Kuffner Jr (2000b) and LaValle and Kuffner Jr (2000a) it is shown that the probability that a vertex is selected for extension in the RRT is proportional to the area of its Voronoi region. This can bee seen in Figure 3.16, where the top row shows the RRT expanding for a holonomic planning problem, and the bottom row shows the Voronoi diagram of the RRT vertices. This shows how the RRT is biased to rapidly explore, and also uniformly covers the space, which are both desired properties in a probabilistic planner.

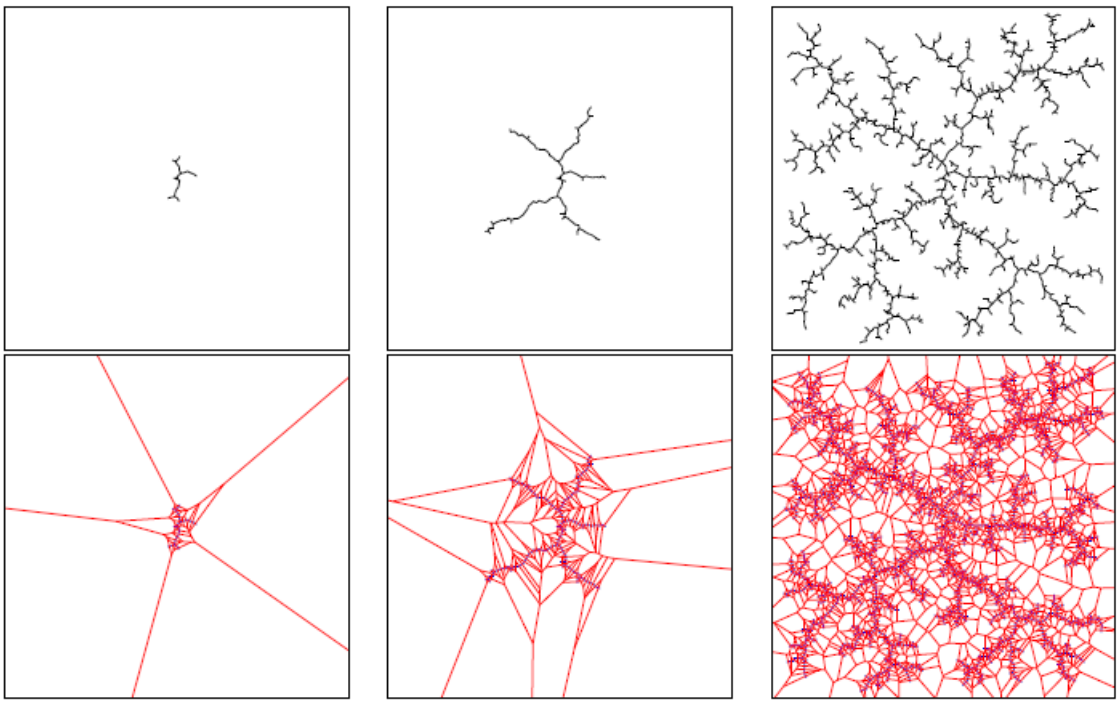


Figure 3.16: RRT biased by large Voronoi regions to rapdily explore, before uniformly covering the space, from LaValle and Kuffner Jr (2000b)

3.3.5 RRT extensions

In this section some of the RRT's properties are presented, together with MATLAB results for visual help.

Immediate goal connection

The results from Figure 3.15 may rise a question of why the path isn't found immediately as a straight line connecting the two points on the map. A feature may be added to the RRT, where for each new point added to the tree, a connection to the goal is checked. This feature is used in most RRT cases elsewhere, but in this report it is not added, as it tends to hide the RRT's properties, which is nice to observe to fully understand how the method works.

Waypoint reduction

Waypoint reduction for the RRT is exactly the same as for the PRM which was explained in Section 3.2.3. An example can be seen in Figure 3.17.

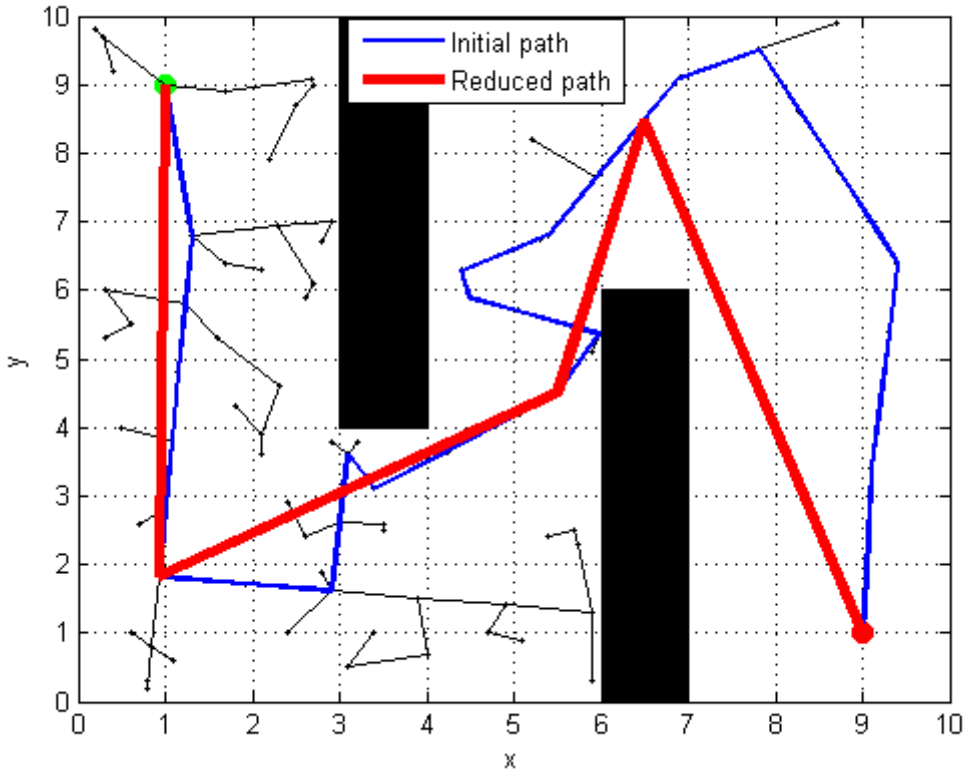


Figure 3.17: Waypoint reduction for the RRT

Weighted distribution (goal bias)

If there exists a solution for connecting a path between two points in a workspace $\mathcal{W}$, the RRT will always find it, see Section 4 in LaValle and Kuffner Jr (2000a). However, in large sets the convergence rate can be slow if there is no bias leading towards the goal. In LaValle and Kuffner Jr (2000b) LaValle debates on different ways to introduce a bias toward the goal.

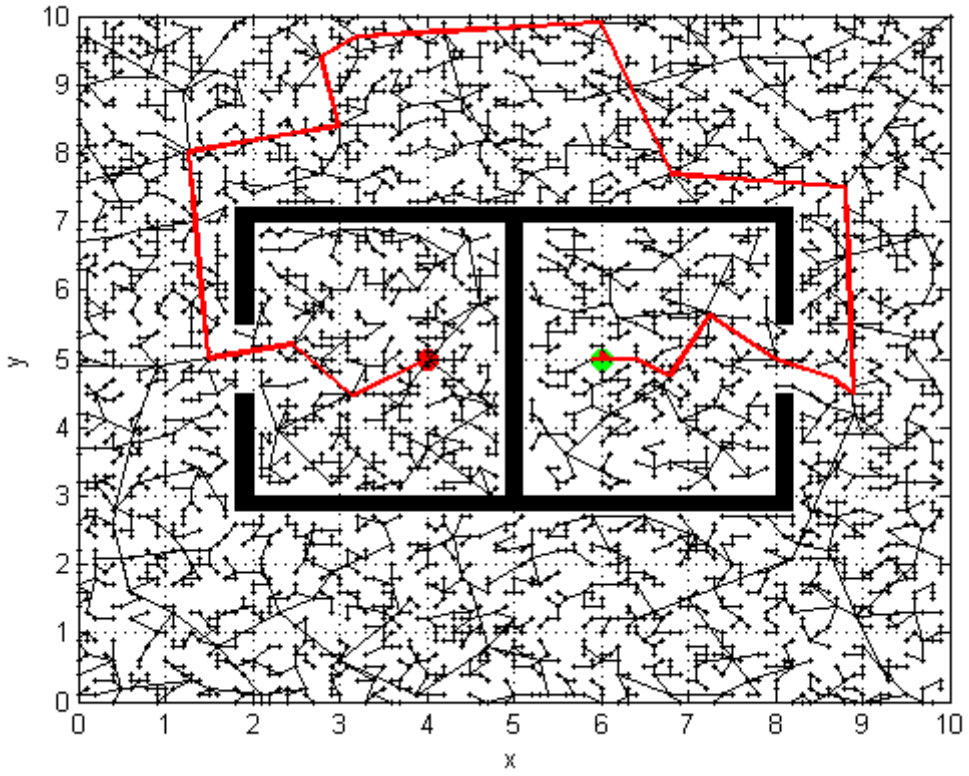


Figure 3.18: RRT cubicle with 0% goalbias

One way to make the convergence faster is to introduce a coin tossing function which decides if the RANDOM_STATE will find a random point in the entire workspace, or in a smaller area around the goal. A similar coin toss method can be used to choose either a completely random point from RANDOM_STATE or the goal point itself. There also exists methods where the weight of the coin increases as the RRT moves closer to the goal.

In the examples of RRT seen in this report, the sampling data which RANDOM_STATE generates a random point from, is distributed in a way that increases the chance of hitting the goal point. A small bias towards the goal, between 1-5% is enough to drastically increase the convergence rate of the RRT, see LaValle and Kuffner Jr (2000b). Figure 3.18 and 3.19 illustrate the difference between a bias of 0 and 1% in an example where the RRT moves from one cubicle

to another. In the 0% case the entire space is explored before the goal is found, using a lot more memory and time then the 1% case where the entire area is explored in a more simple way, before reaching the goal.

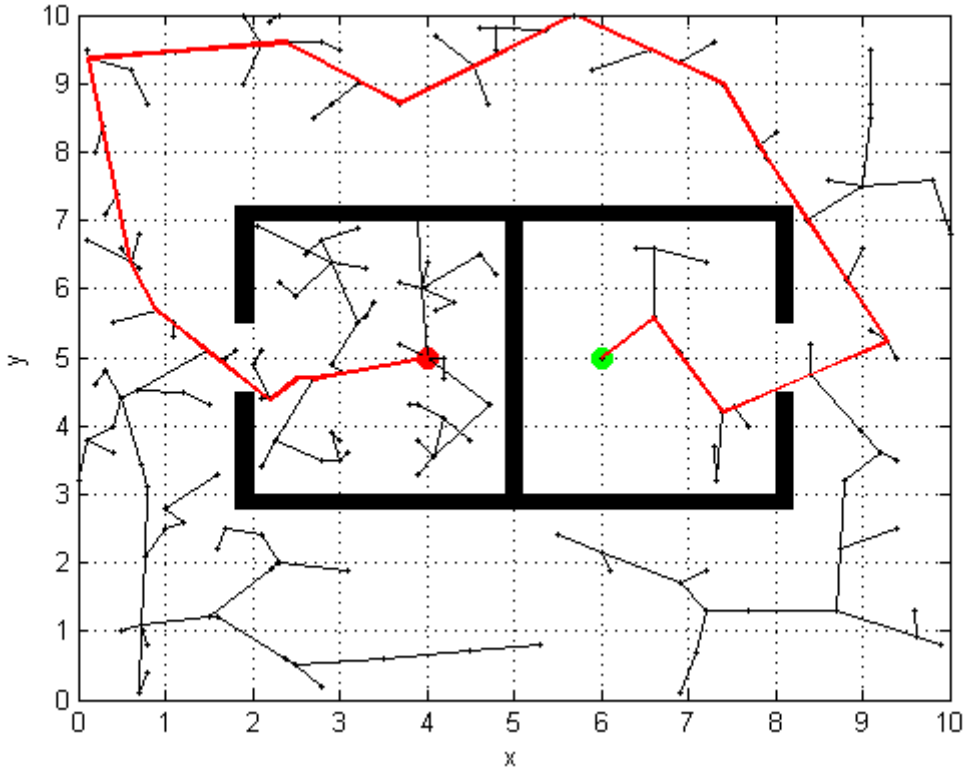


Figure 3.19: RRT cubicle with 1% goalbias

To have a constant bias, or a dynamic one increasing as the tree grows closer to the goal are both viable options, and to choose between them often depends on the shape of the workspace. For simple spaces a constant bias is sufficient, but in areas with elements of spirals or a "room-to-room" setup an increasing bias is a better choice.

Bidirectional RRT

To improve the performance of the RRT, the method can be extended into growing two RRTs towards each other, instead of just a single RRT. This method is called a bidirectional RRT and a detailed treatment can be found in LaValle and Kuffner Jr (2000b). With two trees, one growing from x_{init} and one from x_{goal} the solution is found when the two trees meet. Instead of each tree being biased towards the starting point of the other tree, the bidirectional biases towards the entire other tree instead. This is done to ensure that the trees meet well before covering the entire space.

```
initialize(x_init, x_goal, C);
T_a ← TREE({x_init, 0});
T_b ← TREE({x_goal, 0});
for k = 1 to N do
    x_rand ← RANDOM_STATE(C_free);
    if EXTEND(T_a, x_rand = reached) then
        if EXTEND(T_b, x_new = reached) then
            Return PATH(T_a, T_b)
        end
    end
    SWAP(T_a, T_b);
end
```

Algorithm 3: Bidirectional RRT

The bidirectional version of RRT can be seen as pseudo code in Algorithm 3. For simplicity all the extending operations seen in the standard RRT from Algorithm 2 have been compressed into a function called EXTEND. The bidirectional RRT maintains two trees, $\mathcal{T}_a$ and $\mathcal{T}_b$ simultaneously. For each iteration, one tree extends creating a new vertex. Then the other tree checks if it can connect its nearest vertex to the new vertex, and if successful a path has been found. If not, the roles are swapped and another iteration follows. Figure 3.20 and 3.21 illustrates the advantages the method provides over the standard RRT. The amount of waypoints are reduced without any significant increase in computational time.

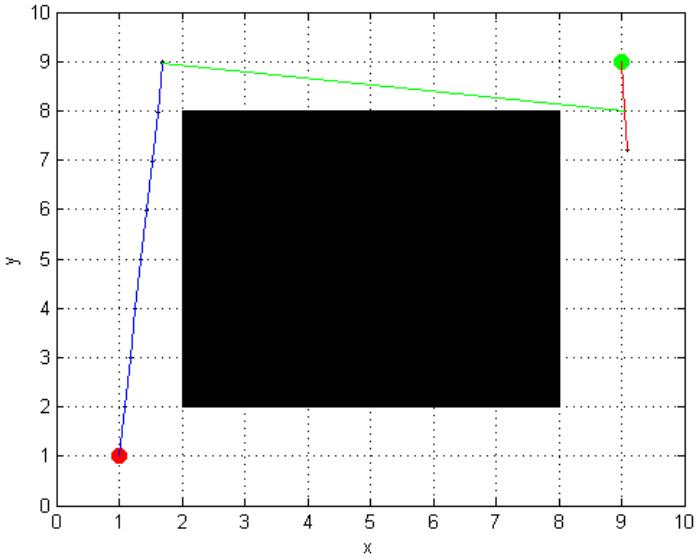


Figure 3.20: Basic Bidirectional RRT

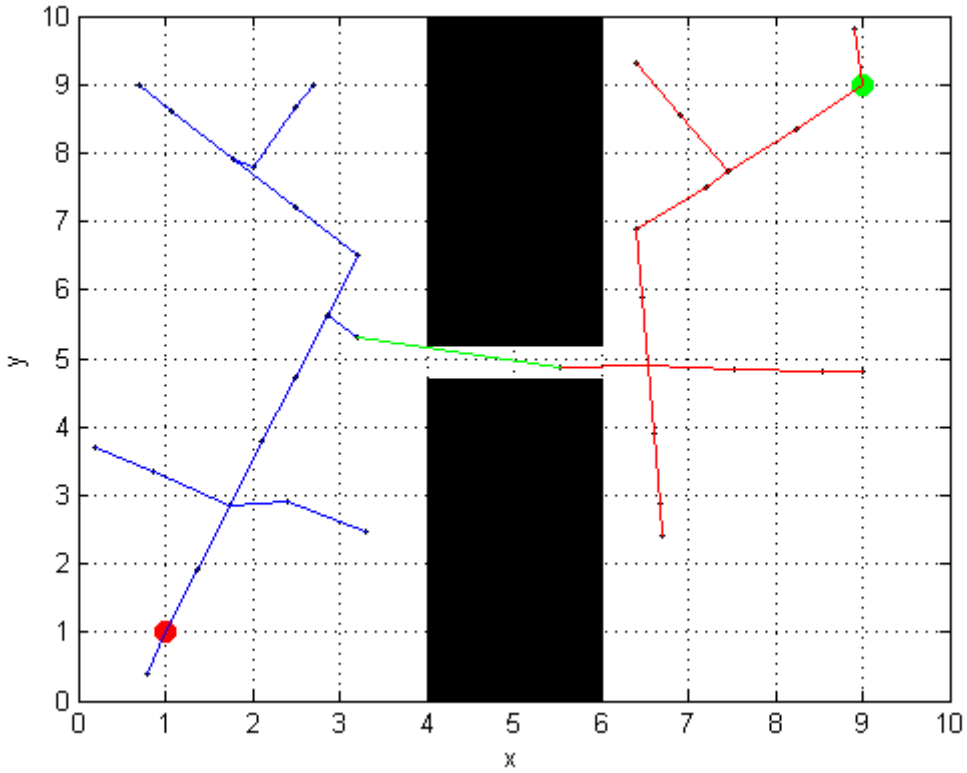


Figure 3.21: Bidirectional RRT in a narrow environment

3.3.6 3D RRT

Extending the RRT to include a third dimension is convenient when using the algorithm for underwater or flying autonomous vehicles. Introducing a third dimension increases the complexity of the method and the collision checking functions must be extended. In this thesis the MATLAB functions inpolygon() and polyxpoly() which were mentioned in Section 2.3 were extended to 3D to be able to check for collisions in all dimensions. Aside from that the graph can still be used as input for Dijkstra's algorithm and a collision free path between to points in the workspace can be found. Figure 3.22 and 3.23 shows a resulting RRT 3D plot after implementation in MATLAB.

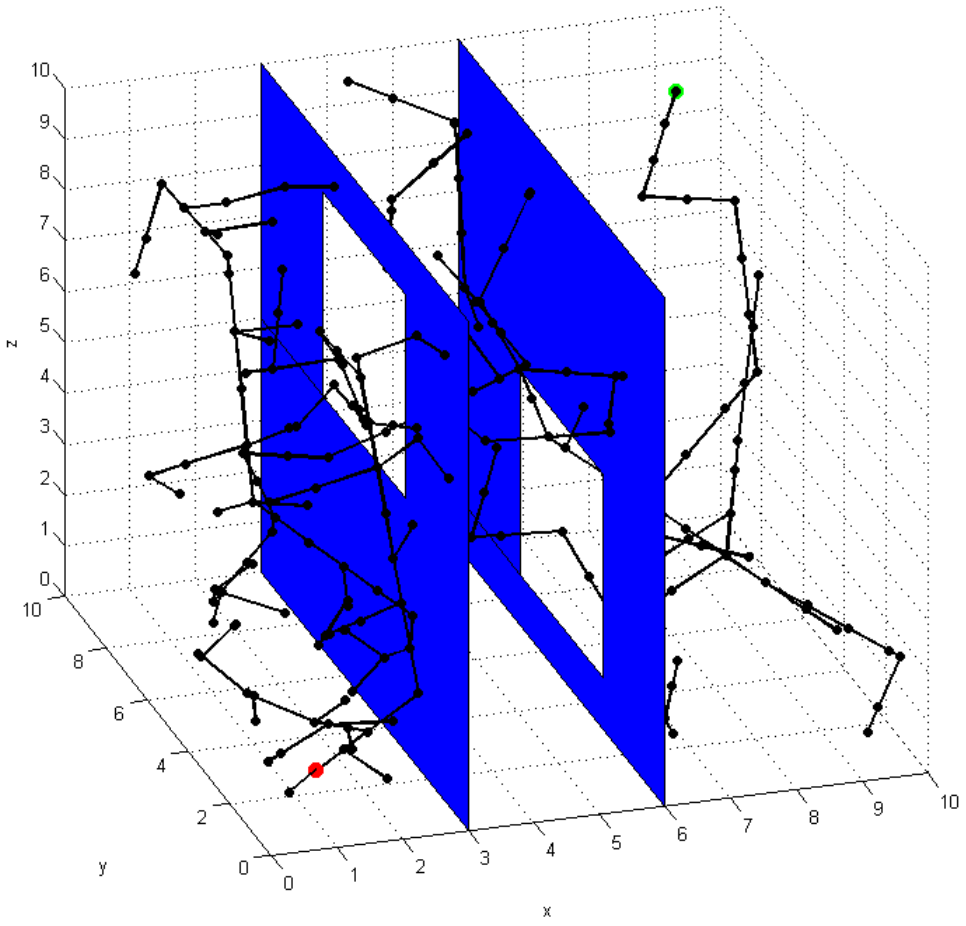


Figure 3.22: RRT generated tree for a 3D 10x10x10 map

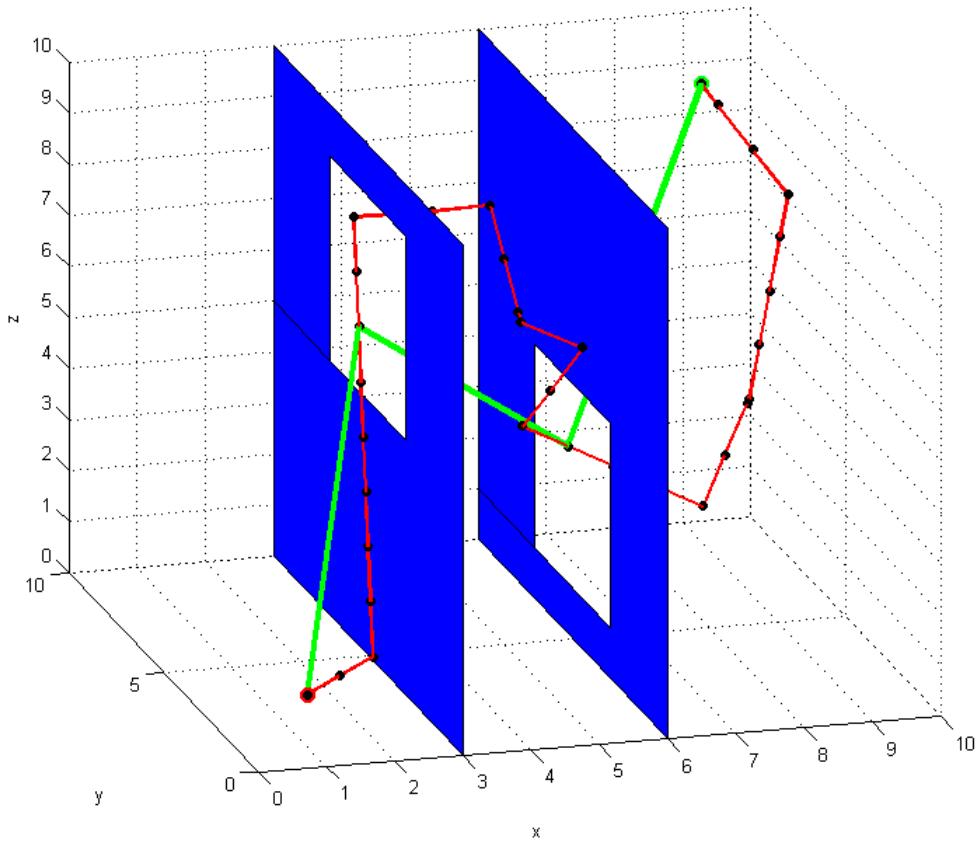


Figure 3.23: RRT original (red) and reduced (green) path for a 3D 10x10x10 map

3.3.7 Problems with the RRT

Even though the RRT has shown promising results in many fields, no method are without disadvantages. In this section some of the problems one might encounter with the RRT are described and illustrated.

Narrow passages

When calculating a path through very narrow environments, the RRT will struggle with finding a way past the obstacles. This is illustrated in Figure 3.24. The random exploration can in this case slow down the RRT, and will result in the method exploring almost the entire half of the space before finding a path to the goal. One way to solve this problem is by using the bidirectional approach from Section 3.3.5. This can be seen by comparing Figure 3.24 and Figure 3.21.

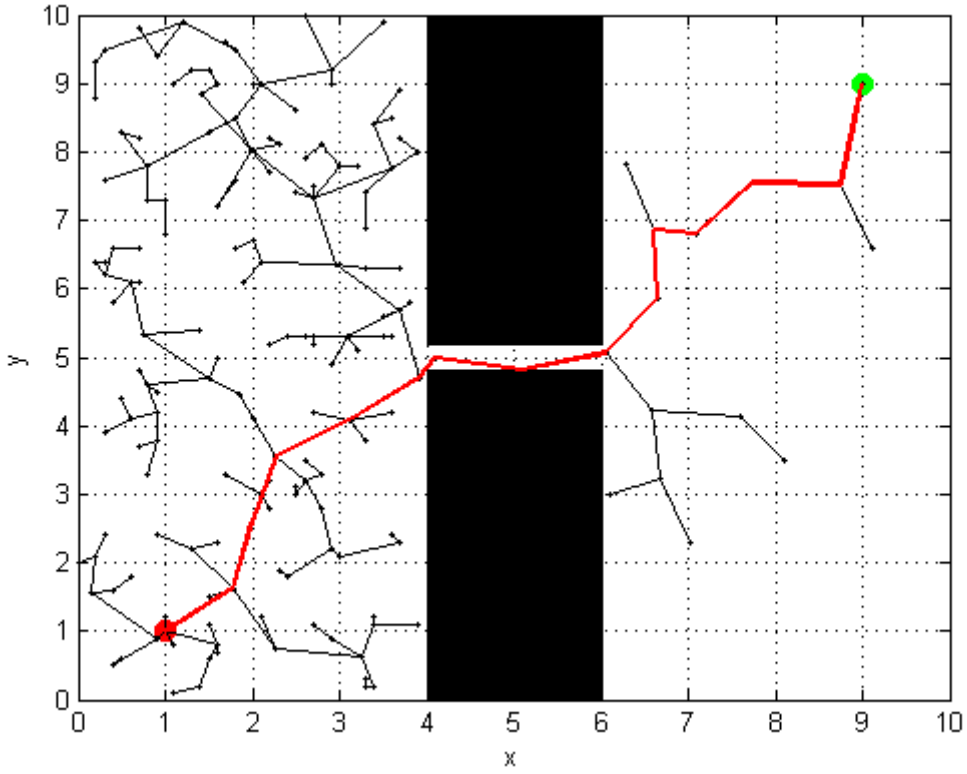


Figure 3.24: RRT with a narrow passage

Local maximas

When extending with a bias towards the goal the RRT might encounter what can be described as a local maximum problem. Illustrated in Figure 3.25, the issue is clearly visible. In this case, extending towards the goal in the early stages of the method is counterproductive, and only causes the computational time to increase substantially. Fortunately, there exists methods that can avoid the local maxima problem, such as a dynamic bias function described in LaValle and Kuffner Jr (2000b), or the bidirectional RRT where two growing trees will avoid the issue which is shown in Figure 3.26.

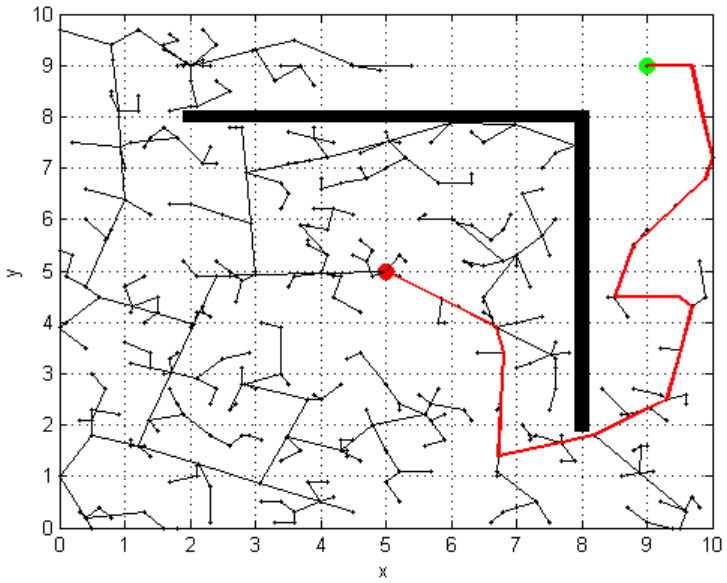


Figure 3.25: RRT local maxima

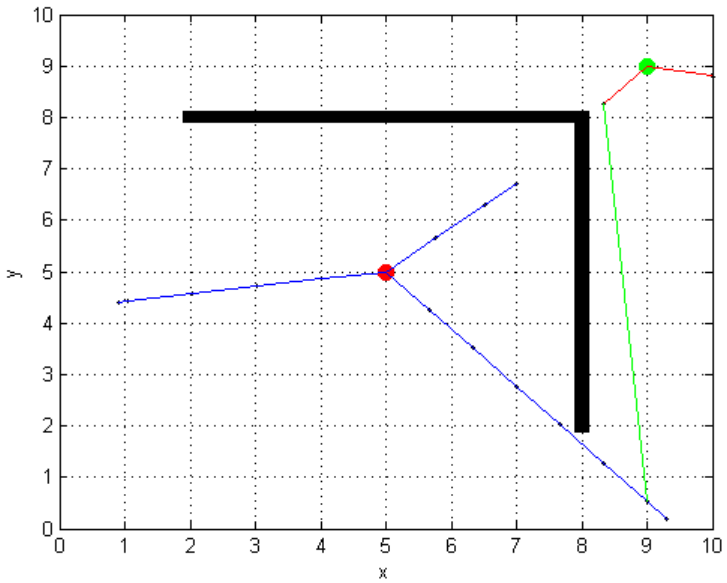


Figure 3.26: Bidirectional RRT local maxima

3.4 Comparing Voronoi, PRM and RRT

In two dimensions the three methods discussed in the previous sections show very similar results, as shown in two different cases in this section.

3.4.1 Small areas

This case is based on Figure 2.5 from the path planning theory section, which is a small area containing four different obstacles. All cases have been implemented in MATLAB using the same map and discretization.

Voronoi diagram

Figure 3.27 shows how the Voronoi diagram method has solved the path planning problem. By placing the Voronoi points on the borders of the map, as well as on the edges of the obstacles, and then remove the points and edges that fall inside the obstacles, a network appears. The network is then saved as a graph with nodes and edges, and can then be applied as input in Dijkstra's algorithm to calculate the shortest path from start to finish. Then the path is fed into a way point reduction function which results in the red path described by a total of four points, down from seventeen. It is important to note that as long as the same points are used as input for the Voronoi diagram method, the result will be exactly the same.

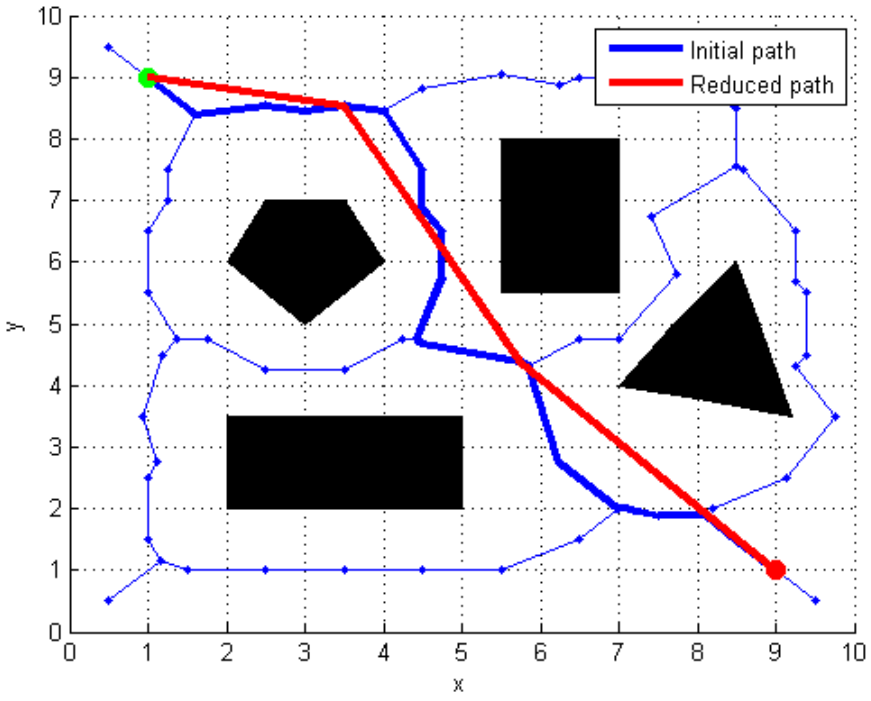


Figure 3.27: Voronoi applied to Figure 2.5

PRM

In Figure 3.28 the PRM's result is shown. The path is different from simulation to simulation because of the probabilistic nature, but in most cases the result is similar to the result from Figure 3.28. The final path, marked as the red path contains a total of three points, reduced from eight.

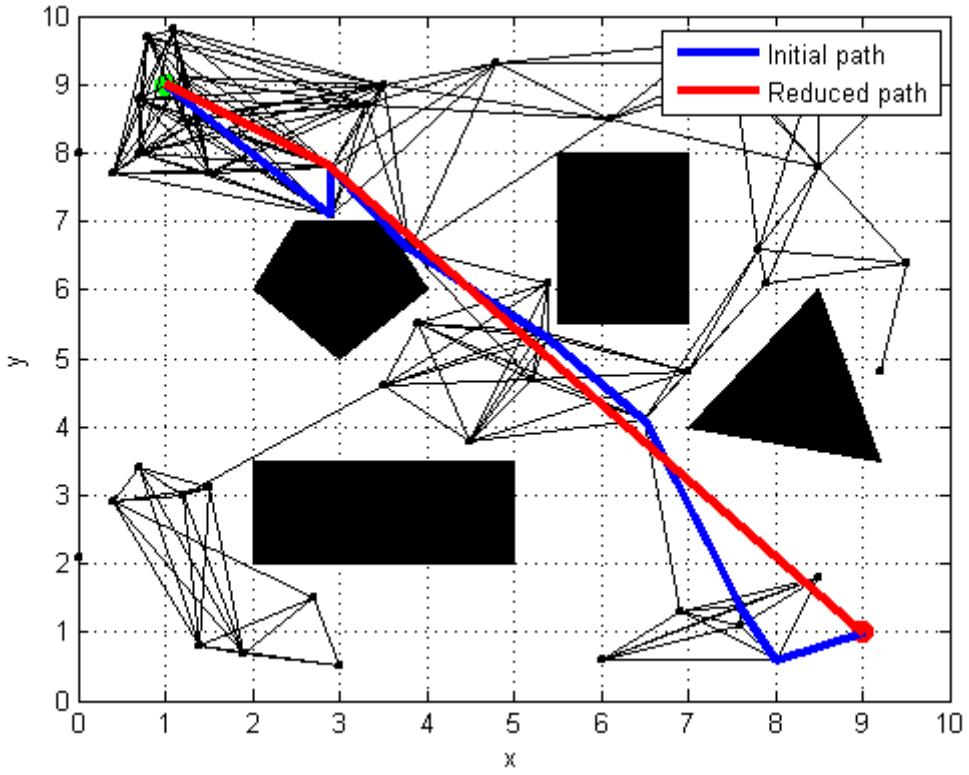


Figure 3.28: PRM applied to Figure 2.5

RRT

Figure 3.29 is the result of the RRT, which as the PRM produces a different result for every simulation, but due to the goal bias (5%) often finds path through the middle. Here the way point reduction function has the best result, as it reduces the path from a total of twenty eight points to only three.

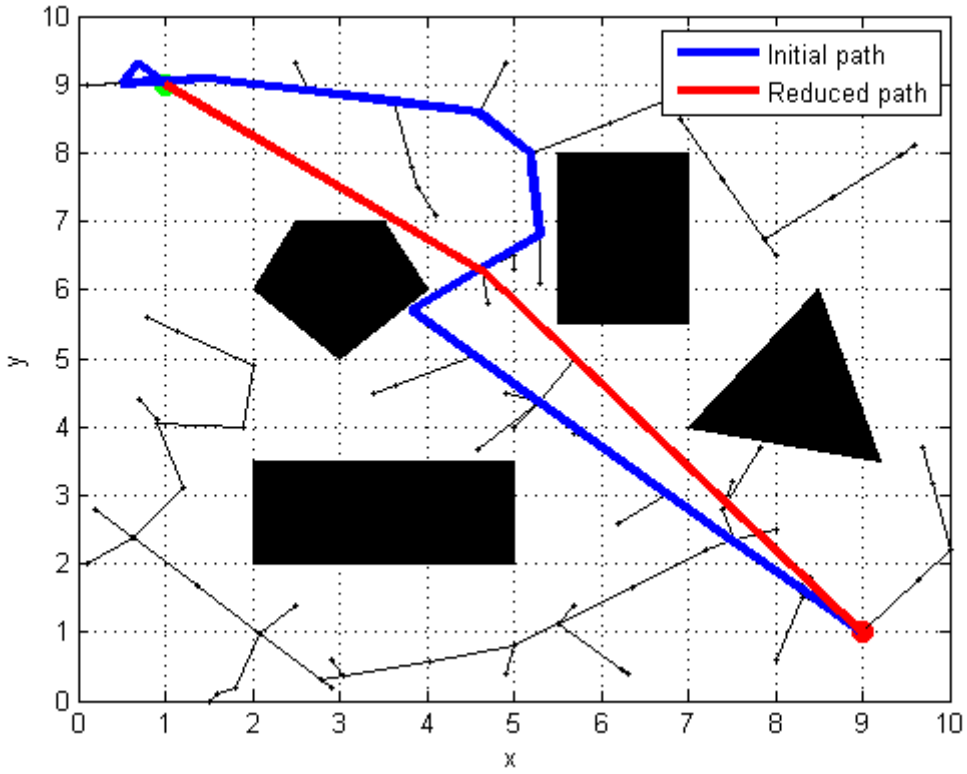


Figure 3.29: RRT applied to Figure 2.5, 5% goalbias

Comparison

Figure 3.30 show that all three paths are very similar, both in length and in the amount of way points the path consist of. The way point reduction is most beneficial for the Voronoi and RRT in terms of the amount of points that can be neglected. It is difficult to judge the methods by the simulation time as the implementation is far from optimal in that sense, but general observation using the inbuilt MATLAB timer show that no method stand out as significantly faster in the small 2D-case.

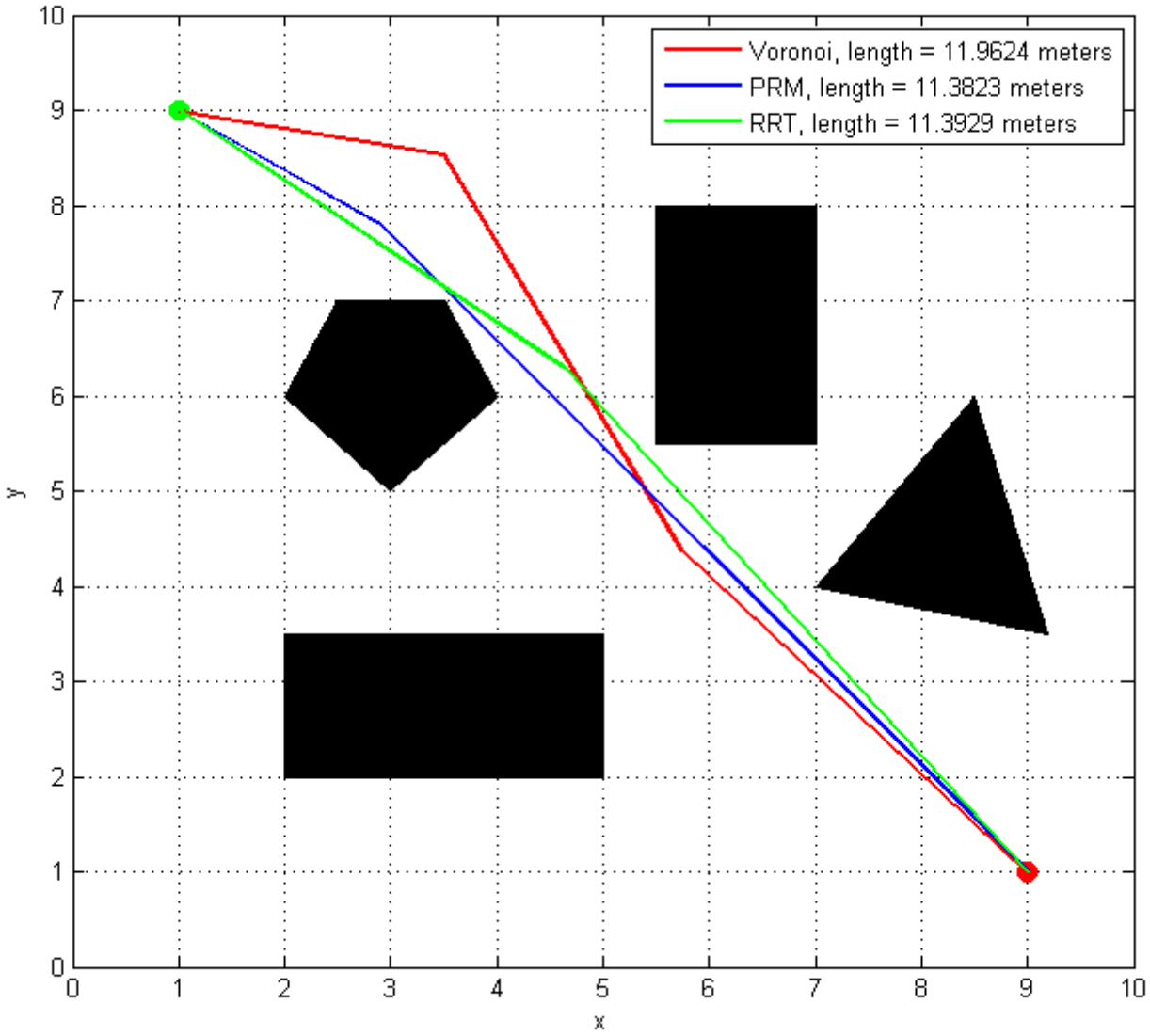


Figure 3.30: Comparing the paths from figures 3.27, 3.28 and 3.29

3.4.2 Large areas

Since the case in Section 3.4.1 using the classic roadmap from Figure 2.5 didn't show much difference in the three methods, a larger case can be considered. To consider the methods in a more realistic setting, a simulation where finding the path in a fjord can be considered. A map over the area can be seen in Figure 3.31, and the different landmasses in this area can be discretized as large polygons for an implementation. In the path planning sense, the entire map is seen as the configuration space $\mathcal{C}$, the water as $\mathcal{C}_{free}$ and the landmasses as $\mathcal{C}_{obs}$.

Figure 3.31: Map over a part of Oslofjorden, from Google (2014)

Voronoi diagram

When applying the Voronoi diagram to the method (Figure 3.32), points were placed along the edges of all the obstacles to create the network. This results in the initial path solution being in the middle relative to nearby obstacles. The reduced path abandons the middle road, and prioritises fewer way points, resulting in a shorter path. It's important to point out that just before reaching the goal in the northeast corner, the Voronoi chooses the extremely narrow path to the east of a small island. If this path was generated for a large vehicle, the absence of a clearance constraint could cause the vehicle to crash because of this choice.

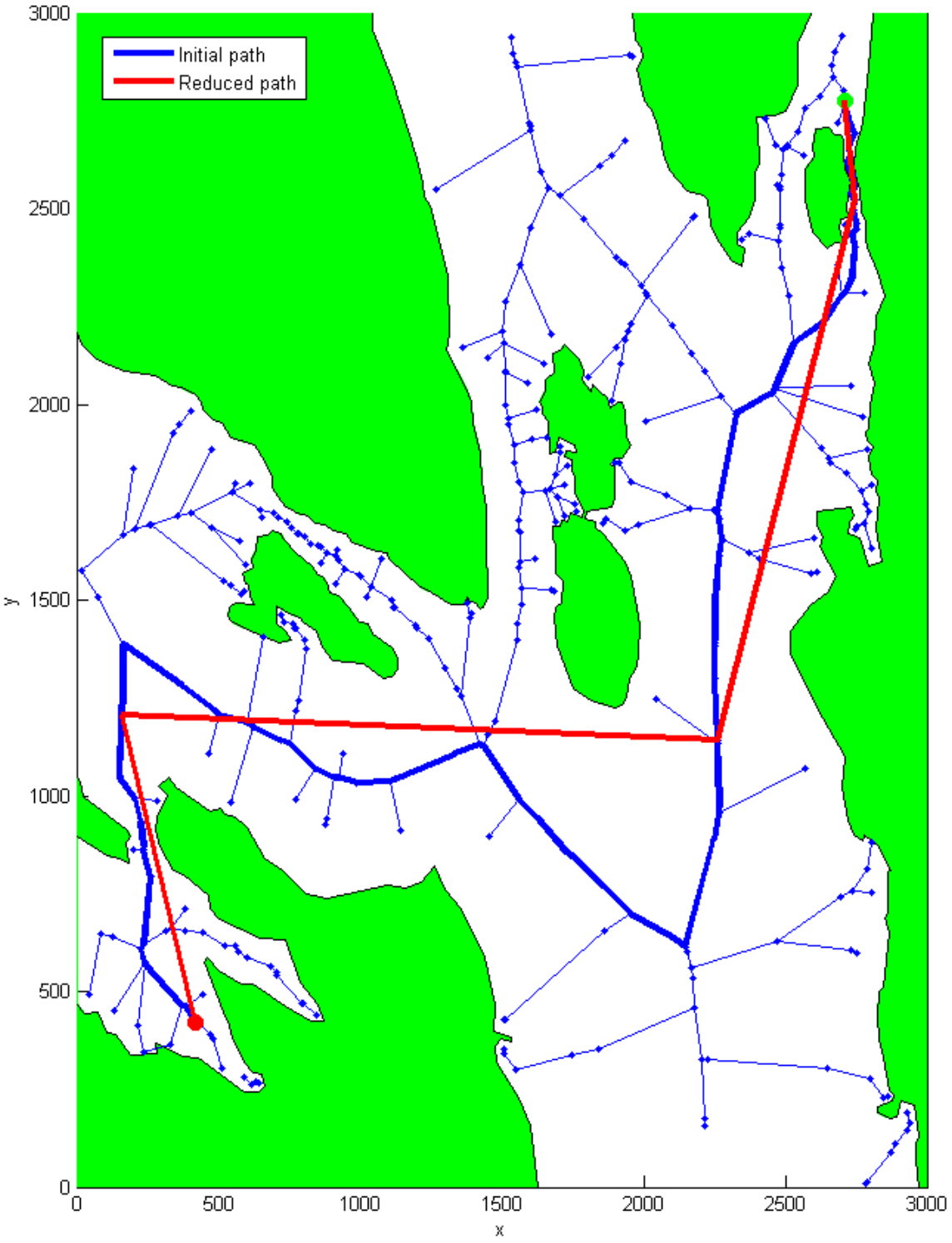


Figure 3.32: Voronoi result for the narrow fjord case

PRM

When finding a path using the PRM, some trial and error was necessary to find the right amount of nodes for the roadmap, as well as the correct maximum connection distance for new nodes. It is worth to mention that this trial and error could have been reduced if the learning and query phase of the PRM were interwoven, as mentioned in Section 3.2.3 In Figure 3.33 the max distance was set to 400 meters and the number of nodes was set to 170. With this setup the PRM covers the space sufficiently enough to find a path.

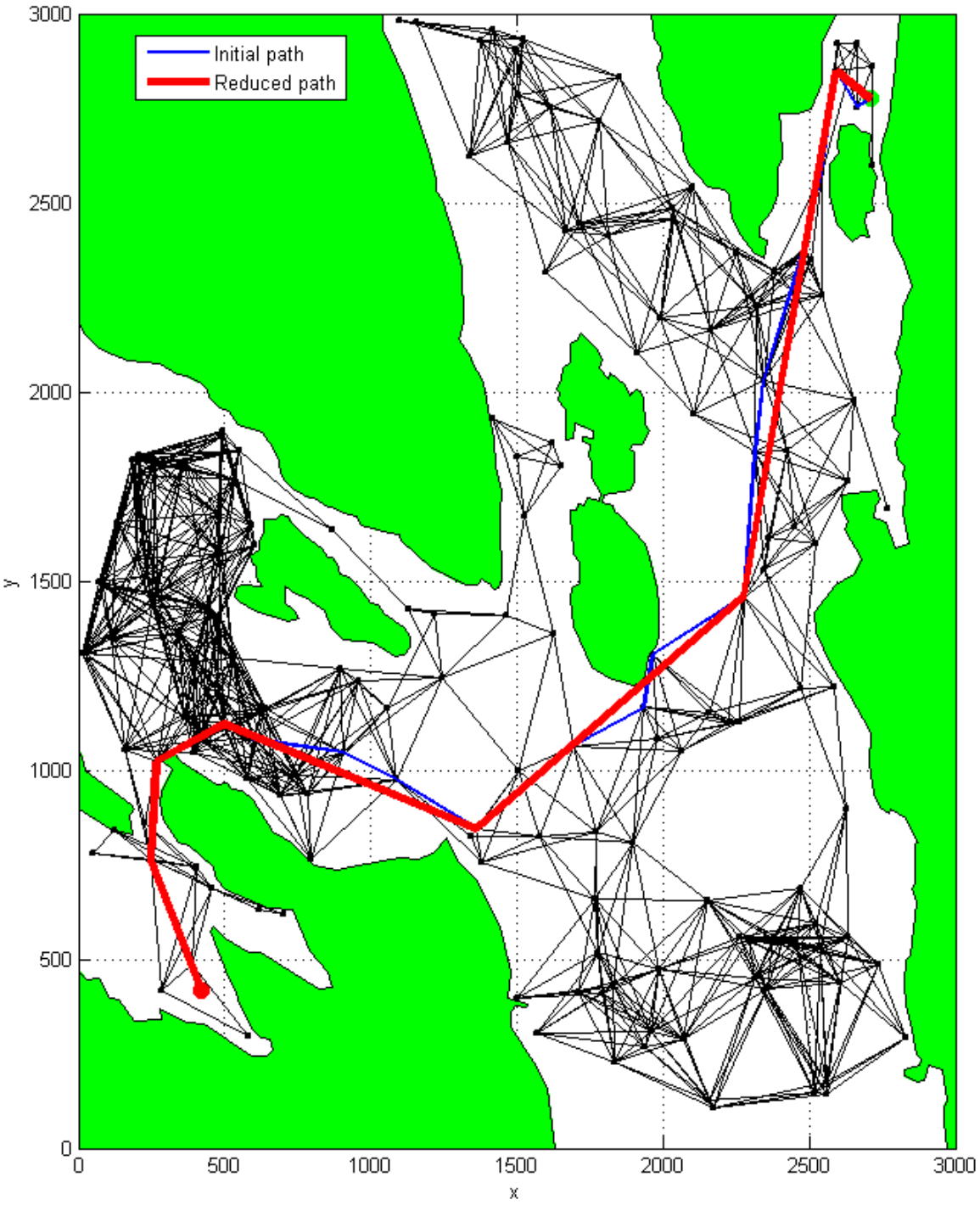


Figure 3.33: PRM result for the narrow fjord case

RRT

When applying the RRT to a large area, it is important to tune the step length of the extend function. In Figure 3.34 a step length of 100 meters has been used. The result shows both how the space is uniformly covered by the method, and how it finds the path relatively fast in terms of the amount of nodes generated, the goal bias was set to 5%.

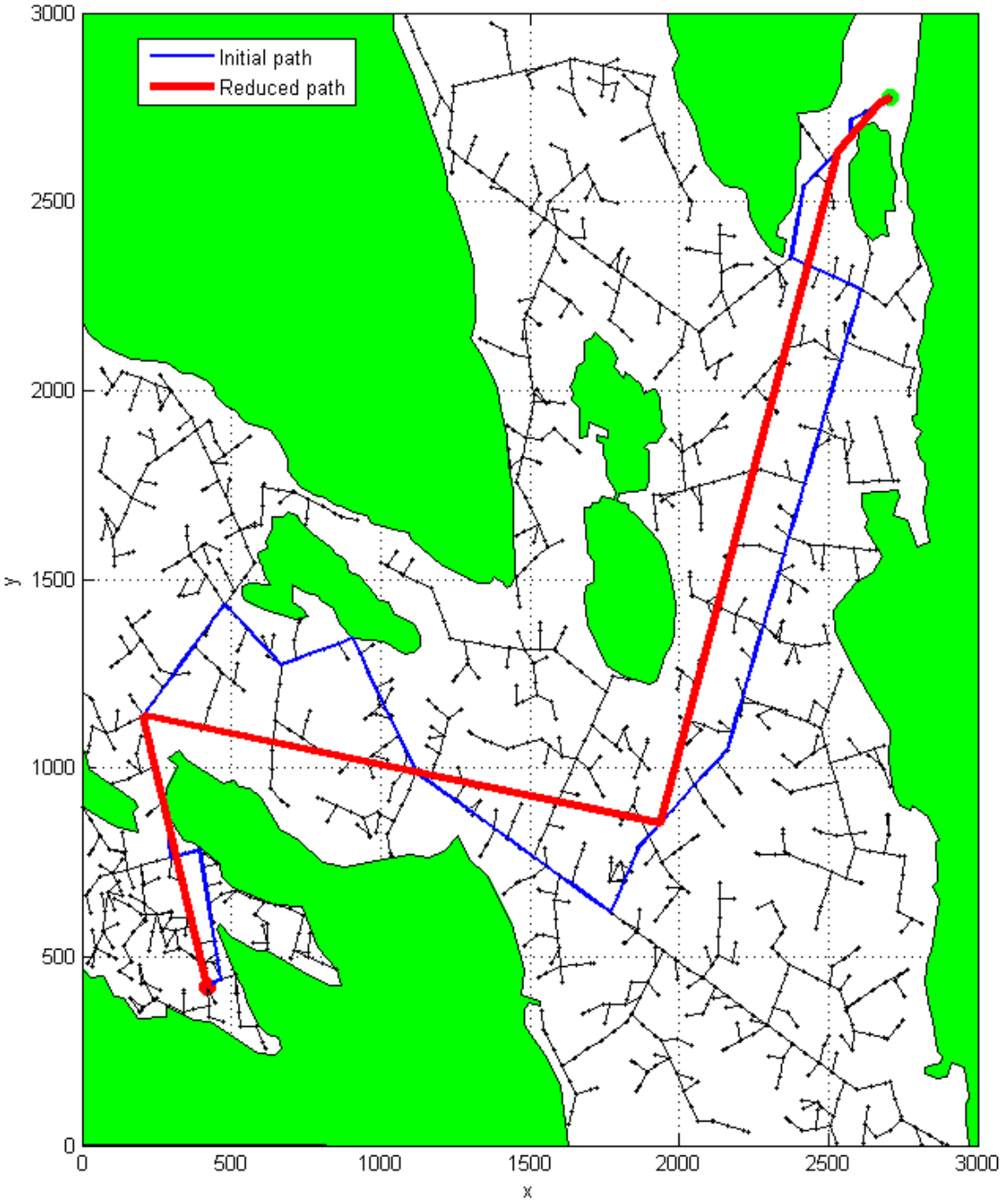


Figure 3.34: RRT result for the narrow fjord case, 5% goalbias

Comparison

Figure 3.35 shows the reduced paths from figures 3.32, 3.33 and 3.34 together, including the length of each path. Looking at allt the results in this section it is easy to conclude that in two dimensions, without any kinematics on the vehicle or dynamic constraints in the area, the Voronoi diagram method, the Probabilistic roadmap method (PRM) and the Rapidly exploring random trees (RRT) method end up with very similar results. This makes the choice of method when planning a path for a static two dimensional case come down to how easy the method is to implement, what the computational time is, and how easy it can be applied to different environments.

When it comes to implementation difficulties, all three methods took advantage of some of MATLAB's built-in functions, such as the graph shortest path method, which ends up doing much of the work. The RRT was implemented first, and was therefore the most challenging one. Where as the two others were trivial after learning a lot from the troubles encountered when implementing RRT. Looking back, it is recommended to implement PRM before RRT. Because the PRM only deals with connecting node to node in one operation, while the RRT uses a loop to extend towards nodes, which can be tricky for beginners.

The computational time of the three methods are all similar, with only small variations in the RRT and PRM due to their probabilistic nature. When it comes to the RRT, some improvement of the mean run time was seen when the bidirectional RRT was used.

When applying the methods to different environments, the Voronoi method has a major disadvantage in that all obstacles must be highly discretized to be sufficient enough as input. This is complicated and time demanding. However it has a built-in clearance constraint.

One way to improve the PRM is to make the learning phase and query phase work together simultaneously, as this will decrease the run time of the method as well as minimise the amount of nodes in the completed roadmap. For the RRT, more experiments on the goal bias, or looking further into the bidirectional RRT are some of the options that will improve the results.

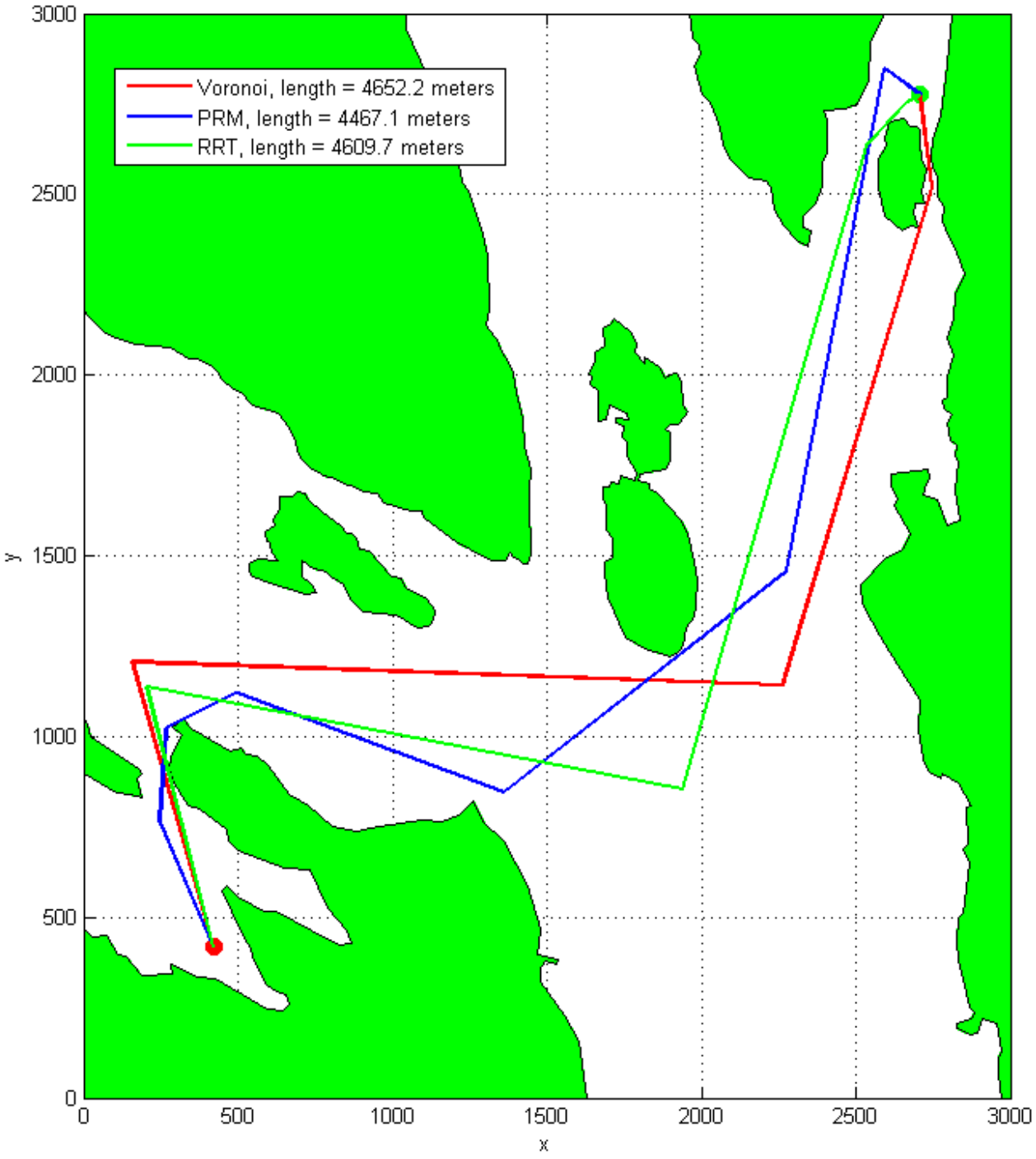


Figure 3.35: Comparing the narrow fjord paths from figures 3.32, 3.33 and 3.34

Chapter 4

Kinematics and Differential Constraints

Just creating collision free paths consisting of waypoints connected by straight lines is rarely useful in a motion planning problem. Fortunately, the planners from Chapter 3 can include a vehicle's kinematics and differential constraints when finding a path. In this chapter, some constraints and vehicle models are introduced, which will be used in simulations combined with the RRT algorithm.

Path parametrization is also considered as it is useful way to formulate paths when used as input for guidance techniques. The formulation comes from Breivik and Fossen (2009) and is also used in Dahl (2013), Lekkas et al. (2013) and Lekkas (2014).

4.1 Curvature

The term curvature is used in many different disciplines. In the motion planning world, curvature is used in a mathematical way to describe the ratio of the change in the angle of a tangent that moves over a given arc. In short, it describes how sharp a turn is.

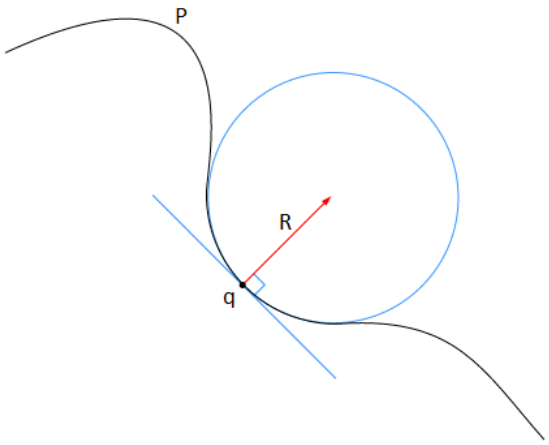


Figure 4.1: Defining curvature

In this report, we look at the geometrical definition of curvature, which is based on straight lines having zero curvature and curved lines being described by using circles. Figure 4.1 shows how the curvature of the point q on the path P can be found. Given any path P and a point q on it, there is a unique circle which approximates the curve close to q. The curvature κ of q can therefore be defined to be the curvature of that circle:

$$\kappa = \frac{1}{R} \tag{4.1}$$

When path planning wheeled systems the minimum turning radius is a central constraint. The maximum possible curvature κ_{max} is then described by R_{min}:

$$\kappa_{max} = \frac{1}{R_{min}} \tag{4.2}$$

To determine the maximum curvature of a vehicle, a common method is to run the vehicle with nominal speed and applying the vehicles maximal steering angle. This will result in the vehicle moving in a circle, and the radius of that circle is used as the minimum turning radius R_{min}.

4.2 Clearance constraint

In most path planning cases the resulting path for the vessel must not only avoid the obstacles but stay a certain distance away from them. In offshore scenarios this is often linked to how far below the waterline a surface vessel is, and using the clearance constraint to avoid shallow waters. For aerial applications where most obstacles are dynamic, a clearance constraint can be used to avoid coming to close to other applications and thus avoid turbulence and other unwanted disturbances.

With maximum curvature and minimum turning radius explained in the previous section it is possible to implement a simple but effective clearance constraint for the polygon obstacles introduced in Section 2.3. This method simply increases the obstacles by a ratio determined by the minimum turning radius. The resulting path will then avoid the obstacle as well as the area around them and guarantee that a vessel can always turn around before hitting the original sized obstacle.

4.3 Simple car model

To understand differential models a simple car model is considered. Since cars can only turn their front wheels, they are limited by a maximum curvature when moving.

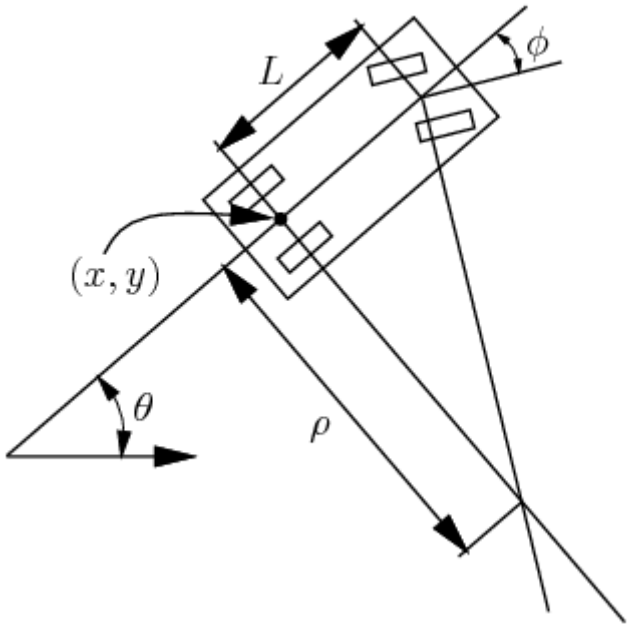


Figure 4.2: 3 DOF car model, with a 2D velocity space

The simple car model is illustrated in Figure 4.2. The car can move in the xy-plane while

the angle is represented as θ. The configuration space of the car is therefore $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}$. At any time the position of the car is denoted as the configuration $q = (x, y, \theta)$. Furthermore the car has a steering angle denoted ϕ, a constant length between the front and rear axle denoted L and a signed speed s. Note that when driving with a constant steering angle the car moves in a circle with radius ρ.

With these notations, a mathematical model of the car's motion can be written as a set of equations:

$$\begin{aligned} \dot{x} &= f_1(x, y, \theta, s, \phi) \\ \dot{y} &= f_2(x, y, \theta, s, \phi) \\ \dot{\theta} &= f_3(x, y, \theta, s, \phi) \end{aligned} \tag{4.3}$$

To find f_1, f_2 and f_3, we take a look at what happens to the position of the car in a small time interval Δt. For this small interval, the car moves approximately the same direction as the rear wheels are pointing, implying that $dy/dx = \tan\theta$. With $dy/dx = \dot{y}/\dot{x}$ and $\tan\theta = \sin\theta/\cos\theta$ we get the following condition:

$$-\dot{x}\sin\theta + \dot{y}\cos\theta = 0 \tag{4.4}$$

The constraint in equation (4.4) is satisfied if $\dot{x} = \cos\theta$ and $\dot{y} = \sin\theta$, and any scalar multiple of this solution, where the scaling factor is the speed s. To derive the equation for $\dot{\theta}$ we denote w as the distance travelled by the car. If the steering angle ϕ is at a fixed position, then the radius ρ represents the radius of the circle traversed, implying $dw = \rho d\theta$. Together with $\rho = L/\tan\theta$ from trigonometry, we get:

$$d\theta = \frac{\tan\phi}{L} dw \tag{4.5}$$

Which when divided by dt on each side and knowing that $\dot{w}$ is the speed s becomes:

$$\dot{\theta} = \frac{s}{L}\tan\phi \tag{4.6}$$

Although the car has been modelled, no input variables have been introduced. When controlling a car, both the speed s and steering angle ϕ can be controlled. These action variables controlling s and ϕ are denoted as u_s and u_ϕ respectively. The final model for the motion of the car is then:

$$\begin{aligned} \dot{x} &= u_s \cos\theta \\ \dot{y} &= u_s \sin\theta \\ \dot{\theta} &= \frac{u_s}{L}\tan u_\phi \end{aligned} \tag{4.7}$$

4.4 Dubins car

The Dubins car is a simplified model of the model in equation (4.8) from the previous section. For a Dubins car, the speed input u_s is restricted to $u_s \in \{0, 1\}$. This means a Dubins car can only move in three different ways: straight forward, left turn or right turn. It also has a maximum steering angle ϕ_{max} which results in a minimum turning radius of ρ_{min}. With constant speed the Dubins car can be modelled as:

$$\begin{aligned} \dot{x} &= \cos\theta \\ \dot{y} &= \sin\theta \\ \dot{\theta} &= u \end{aligned} \tag{4.8}$$

The input u is chosen as either of the three ways of available motion:

Symbol	Steering: u
S	0
L	1
R	-1

Table 4.1: Dubin's car motion primitives

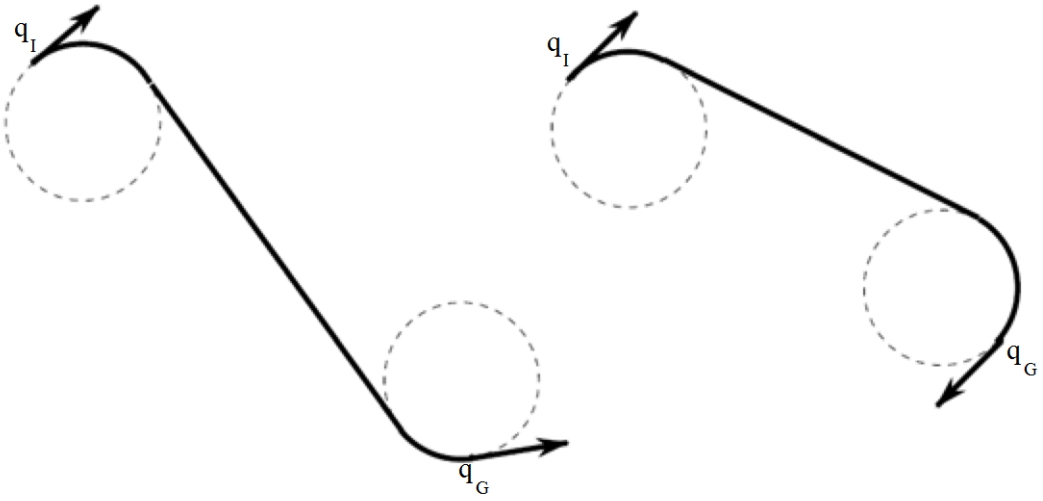


Figure 4.3: Dubins path examples. Left path RSL, right path RSR

4.4.1 Dubins curves

Dubins curves was introduced in Dubins (1957). The article showed that between any two configurations q_I and q_G, the shortest path for a Dubins car can always be expressed as a

combination of no more than three motion primitives, namely the primitives from Table 4.1. S translates to driving straight ahead, and L and R translates to turning left and right as sharply as possible (maximum curvature). With a path consisting of three primitives, there are ten possible paths, and in Dubins (1957) it is shown that out of these ten only six are possibly the optimal path. The six possible optimal paths are the following: {LRL,RLR,LSL,LSR,RSL,RSR}.

4.5 Path parametrization

In Section 2.1.1 a path was formulated as $r(q)$ as it is in Tsourdos et al. (2010). In this section, a different formulation of a path is given, which is more suited as input for guidance. The formulation has been used in Breivik and Fossen (2009), Dahl (2013) and Lekkas et al. (2013), and considers a planar path as a one-dimensional manifold that can be expressed as the set

$$\mathcal{P} := \{\mathbf{p} \in \mathbb{R}^2 \mid \mathbf{p} = \mathbf{p}_p(\varpi) \; \forall \; \varpi \in \mathbb{R}\} \tag{4.9}$$

where the symbol ϖ is called the path parameter and $\mathbf{p} = \mathbf{p}_p(\varpi)$ is then the position of a point belonging to the path. Choosing the path parameter ϖ is not always easy as it does not necessarily need to have any physical meaning. For example, in the case of a straight lines, the path length can be used, but in most cases choosing ϖ to be within the unit domain is more convenient:

$$\varpi \in [0, 1] \tag{4.10}$$

When implementing a path $\mathbf{p}(\cdot)$, it is common practice to implement the path as a piecewise-defined function, which will reduce the function complexity and in turn the computational time. Both Lekkas (2014) and Lekkas et al. (2013) mention this and state that even though this method has its advantages it demands consideration at the transition points between subpaths. In Haugen (2010), a definition for planar paths consisting of a number of curve segments was expressed as:

$$\begin{aligned} \mathcal{P}_i := \{&\mathbf{p}_i \in \mathbb{R}^2 | \\ &\mathbf{p}_i = \mathbf{p}_{i,p}(\varpi) \forall \varpi \in \mathcal{I}_i = [\varpi_{i,0}, \varpi_{i,1}] \subset \mathbb{R}\} \end{aligned} \tag{4.11}$$

The expression in equation (4.11) can be written as a superset of n curve segments:

$$\mathcal{P}_s = \bigcup_{i=1}^{n} \mathcal{P}_i \tag{4.12}$$

In this thesis, the type of path parametrization which will be used for two-dimensional curves is expressed as:

$$\mathbf{p}_p(\varpi) = \begin{bmatrix} x_p(\varpi) \\ y_p(\varpi) \end{bmatrix} \tag{4.13}$$

with the path tangential angle computed as:

$$\chi_p(\varpi) = \arctan 2(y'(\varpi), x'(\varpi)) \tag{4.14}$$

where $(\cdot)'$ denotes the first derivative w.r.t the path parameter ϖ.

4.5.1 Straight line

For a straight line connecting two points, the parametric form can be expressed similar to Breivik and Fossen (2009) as:

$$\mathbf{p}_{line}(\varpi) = \begin{bmatrix} x_0 + L\varpi\cos(\chi_l) \\ y_0 + L\varpi\sin(\chi_l) \end{bmatrix} \tag{4.15}$$

where $\mathbf{p}_0 = [x_0, y_0]^T$ is the starting point, L the path length, and χ_l the constant path-tangential angle.

4.5.2 Circular arc

In Section 4.4.1 a Dubins path was shown to be a path consisting of circular arcs and straight lines. A circular arc can be parameterized as:

$$\mathbf{p}_{cir}(\varpi) = \begin{bmatrix} c_x + R\cos(\alpha_0 + \varpi(\alpha_1 - \alpha_0)) \\ c_y + R\sin(\alpha_0 + \varpi(\alpha_1 - \alpha_0)) \end{bmatrix} \tag{4.16}$$

where $\mathbf{c} = [c_x, c_y]^T$ is the center of the circle, R is the radius and α_0 and α_1 are the angles at which the circular arc starts and ends, respectively.

4.5.3 Clothoid

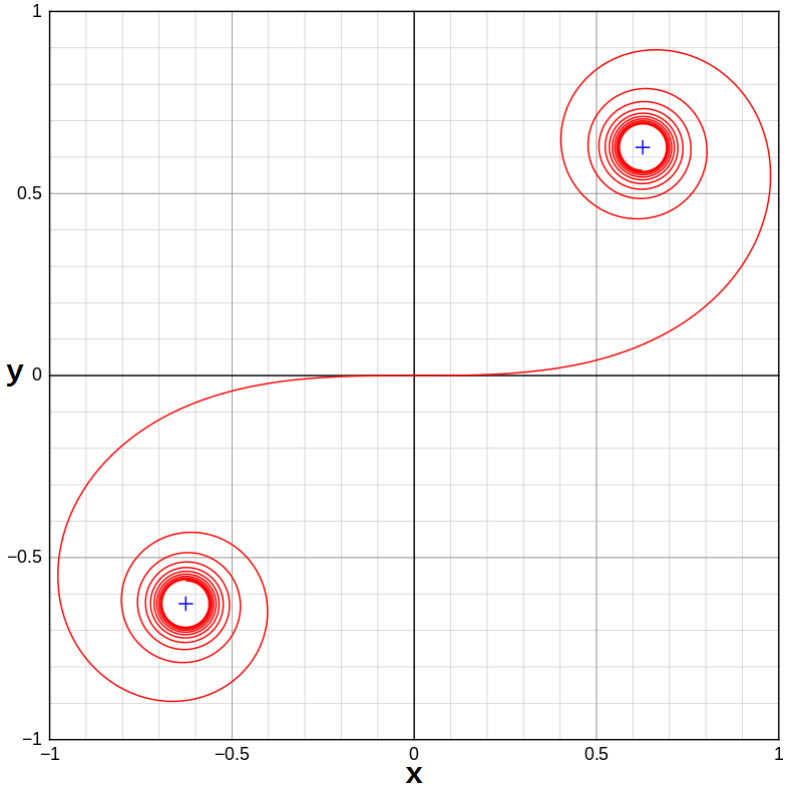


Figure 4.4: Double-end Euler spiral from Wikipedia (2015)

A clothoid is a spiral which has the property of a linear increase in curvature along its length. Its often used when designing railroads and roller coasters when creating a transition between a straight segment of a path and a circular segment. The clothoid is often called an Euler spiral as it was first presented in Euler (1744).

In this thesis, the clothoid will later be used together with Dubins curves for curvature continuous paths. But in this section the clothoid itself is presented. A clothoid can be parameterized as in Dahl (2013) and Lekkas et al. (2013):

$$S(x) = \int_0^x \sin(t^2)dt \tag{4.17}$$

$$C(x) = \int_0^x \cos(t^2)dt \tag{4.18}$$

Together the two equations (4.17) and (4.18) are called Fresnel integrals.

As mentioned, the main property of the clothoid is its linear increase in curvature proportional to the length of the arc. With the length of the arc denoted as s, the initial curvature as κ_0 and the sharpness coefficient as c the curvature κ is:

$$\kappa(s) = cs + \kappa_0 \tag{4.19}$$

With curvature being the rate of change in the tangential angle of a curve, integration gives the equation for χ:

$$\chi(s) = \int_0^s \kappa(\xi)d\xi = \frac{1}{2}cs^2 + \kappa_0 s + \chi_0 \tag{4.20}$$

where the constant χ_0 is the initial course. The tangent vector equation is then:

$$t(s) = \begin{bmatrix}\cos(\chi(s))\\ \sin(\chi(s))\end{bmatrix} = \begin{bmatrix}\cos(\frac{1}{2}cs^2 + \kappa_0 + \chi_0)\\ \sin(\frac{1}{2}cs^2 + \kappa_0 + \chi_0)\end{bmatrix} \tag{4.21}$$

The entire clothoid segment can be given as a path $\mathbf{p}_{clothoid}$, with initial position $\mathbf{p}_0 = [N_0, E_0]$, by integrating the tangent angle across the length of the arc s:

$$\mathbf{p}_{clothoid}(s) = \mathbf{p}_0 + \int_0^s t(\xi)d\xi = \begin{bmatrix}N_0 + \int_0^s \cos(\frac{1}{2}c\xi^2 + \kappa_0\xi + \chi_0)\ d\xi\\ E_0 + \int_0^s \sin(\frac{1}{2}c\xi^2 + \kappa_0\xi + \chi_0)\ d\xi\end{bmatrix} \tag{4.22}$$

By choosing a desired clothoid length s_{end} and sharpness c from equation (4.19) the segment can be manipulated to achieve any desired tangent angle χ_d.

In the case of a desired final tangent angle χ_d and maximal curvature κ_{max}, equation (4.19) and (4.20) can be changed to:

$$\kappa_{max} = \kappa(s_{end}) = cs_{end} \tag{4.23}$$

$$\chi_d = \chi(s_{end}) = \frac{1}{2}cs_{end} \tag{4.24}$$

which can be solved to yield the necessary length and sharpness to satisfy the conditions:

$$s_{end} = \frac{2\chi_d}{\kappa_{max}} \tag{4.25}$$

$$c = \frac{\kappa_{max}^2}{2\chi_d} \tag{4.26}$$

4.5.4 Fermat's spiral

The Fermat's spiral originates from the 1600s when the French mathematician Fermat first studied it. It was first published in Fermat (1697). Similar to the clothoid, the Fermat's spiral (FS) is a curve with initial curvature equal to zero and then increases over its length. One of the main reasons to replace the clothoid with the Fermat's spiral is to decrease the computational time, as the FS does not include any integral terms. FS is a part of the *Archimedean spirals* which is a family of curves described by the equation:

$$r = k\theta^{1/n} \tag{4.27}$$

taken from Weissten (2013a). Equation (4.27) is in polar coordinates where r is the radial distance, k is a scaling constant (sometimes denoted as a), θ is the polar angle and n is a constant which determines how tightly the spiral is wrapped. With $n = 2$ we get the FS equation:

$$r = k\sqrt{\theta} \tag{4.28}$$

with the curvature written as in Weissten (2013b):

$$\kappa = \frac{1}{k}\frac{2\sqrt{\theta}(3 + 4\theta^2)}{(1 + 4\theta^2)^{3/2}} \tag{4.29}$$

The chosen parametrization of the FS is in Cartesian coordinates where an FS with initial position $p_0 = \begin{bmatrix} x_0, y_0 \end{bmatrix}^T$, initial tangent angle χ_0 and rotational direction ρ can be written as in Lekkas et al. (2013):

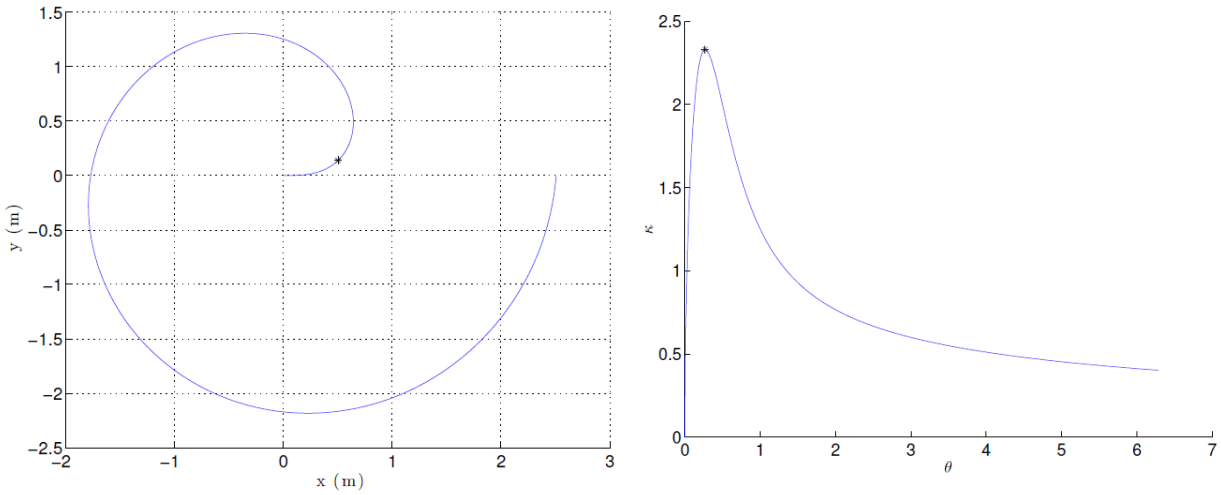
$$\mathbf{p}_{fermat}(\theta) = \begin{bmatrix} x_0 + k\sqrt{\theta}\cos(\rho\theta + \chi_0) \\ y_0 + k\sqrt{\theta}\sin(\rho\theta + \chi_0) \end{bmatrix} \tag{4.30}$$

This parametrization describes a curve which starts with zero curvature and then increases to the wanted curvature which is determined by the domain of θ, $\theta \in \begin{bmatrix} 0, \theta_{end} \end{bmatrix}$. This domain is neither unit domain nor unit speed, but can be unit domain by choosing the path parameter to be $\theta = \varpi\theta_{end}$.

When using the FS as entering and exiting segments for smoothing a turn, the exiting segment has its own parametrization which is a mirrored version of the entering segment. In other words the exiting segment must start at the chosen curvature and then decrease it to zero. In Lekkas et al. (2013) the exiting segment $\mathbf{p}_{\overline{fermat}}$ is written:

$$\mathbf{p}_{\overline{fermat}}(\theta) = \begin{bmatrix} x_{end} + k\sqrt{\theta_{end} - \theta}\cos(\rho(\theta - \theta_{end}) + \chi_{end}) \\ y_{end} + k\sqrt{\theta_{end} - \theta}\sin(\rho(\theta - \theta_{end}) + \chi_{end}) \end{bmatrix} \tag{4.31}$$

where $\mathbf{p}_{end} = [x_{end}, y_{end}]^T$ is the position at the end of the curve, i.e. $\mathbf{p}_{end} = \mathbf{p}_{\overline{fermat}}(\theta_{end})$, with χ_{end} as the course angle at $\mathbf{p}_{end}$.



(a) Fermat's spiral, one complete rotation. Asterisk marks the point where the curve has maximum curvature

(b) Fermat's spiral curvature. The asterisk marks the value of θ for which the curve has maximal curvature

Figure 4.5: Fermat spiral properties, from Lekkas et al. (2013)

4.5.5 Path table

In Dahl (2013), a path table is used to save each segment of a path as parameters and functions instead of saving the infinite amount of points $\mathbf{p} \in \mathbb{P}$ that constitute the path. This method is beneficial when using a path as input for guidance systems as each row of the table represents a subpath with the necessary parameters. The path parameter ϖ can then be used to traverse the table when the path is used as target for the guidance system.

Curve segment	1	2	3	4	5	6	7	8
Straight line	1	$p_{0,x}$	$p_{0,y}$	l_x	l_y	-	-	-
Circular arc	2	c_x	c_y	R	α_0	α_1	-	-
Clothoid	3	$p_{0,x}$	$p_{0,y}$	s_{end}	c	κ_0	χ_0	-
Fermat	4	$p_{0,x}$	$p_{0,y}$	a	θ_{end}	ρ	χ_0	ζ

Table 4.2: Path table, Dahl (2013)

Table 4.2 shows an example of the path table. The first value reveals what type of segment the curve has, followed by the necessary parameters for that type of segment as introduced in sections 4.5.1, 4.5.2, 4.5.3 and 4.5.4.

Chapter 5

2D Rapidly-exploring Random Trees with Dubins Car Dynamics

In this chapter the RRT algorithm is used as a basis for three interpolating algorithms which generate paths accommodated for a vehicles dynamic constraints, such as max curvature and Interpolating means that all the waypoints from the RRT-generated path are visited along the way.
The chapter includes a tutorial on how the paths can be computed, similar to Dahl (2013) and Lekkas et al. (2013), with figures of the resulting plots of the calculations. In the end some comparisons will be made, especially between the curvature continuous paths.

5.1 Generating the initial path with RRT

Generating the initial path with the RRT yields a path consisting of straight lines connected by waypoints. The input for the RRT includes the workspace, a chosen start and finishing pose, as well as a chosen minimum turning radius of the vehicle, which creates and extra layer around the obstacles. Compensating for minimum turning radius is done so that the vehicle is guaranteed to not collide with the original obstacle. Figure 5.2 shows the initial path consisting of straight lines connected by waypoints from the RRT, which will be used as the basis path when creating the Dubins path. The chosen start and finishing configurations as well as the minimum turning radius is listed in Table 5.2. Figure 5.1 is a list summarizing the different elements in the plots throughout this section.

Minimum turning radius R(m): 0.35	x(m)	y(m)	θ(rad)
Starting pose	1	1	0
Goal pose	9	9	0

Table 5.1: 2D RRT Dubins path input

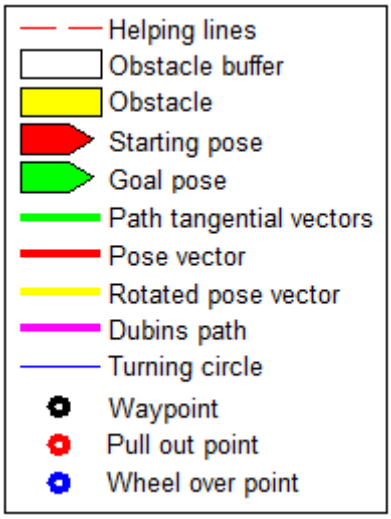


Figure 5.1: Legend for Figure 5.2-5.9

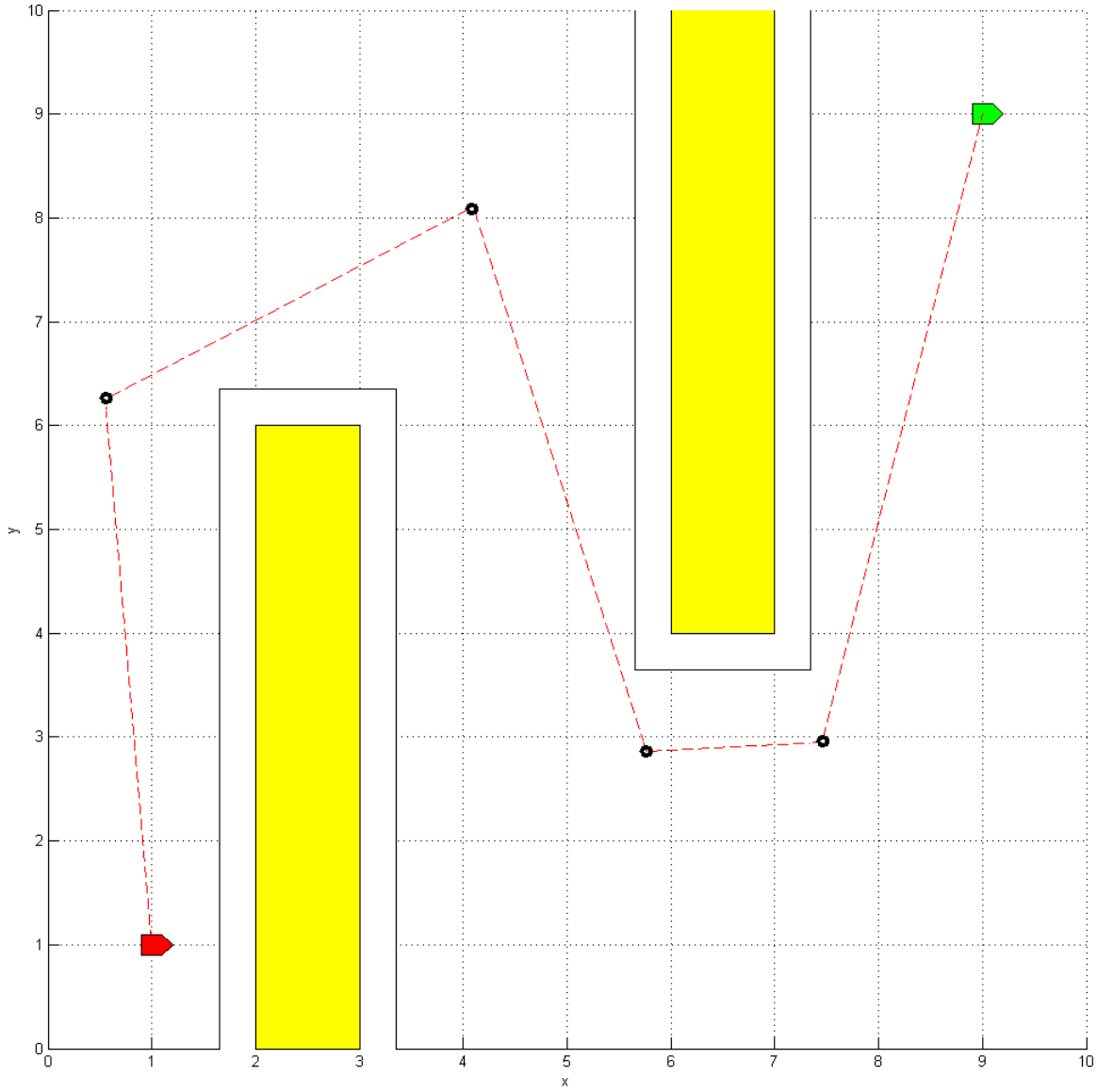


Figure 5.2: Initial path created by the RRT

5.2 Dubins path algorithm

The Dubins path algorithm for a sequence of waypoints is a method implemented by Andreas Dahl in Dahl (2013). It is based on the path generation method from Tsourdos et al. (2010) which generates a path between and initial and a final pose, but extended to a sequence of waypoints. In this thesis the method is applied to the sequence of waypoint created by the waypoint reduced RRT, with a chosen start and finish pose, not assuming the initial heading to be equal to the direction of the line connecting to the first interior waypoint.

5.2.1 Preparation

At the starting point, the finishing point and the interior waypoints a course tangent vector is calculated.

$$\vec{v}_{in,initial} = \begin{bmatrix} \cos(\theta_{init}) \\ \sin(\theta_{init}) \end{bmatrix} \tag{5.1}$$

$$\vec{v}_{in,i} = \frac{wpt_i - wpt_{i-1}}{\|wpt_i - wpt_{i-1}\|}, i \neq initial \tag{5.2}$$

$$\vec{v}_{out,i} = \frac{wpt_{i+1} - wpt_i}{\|wpt_{i+1} - wpt_i\|}, i \neq goal \tag{5.3}$$

$$\vec{v}_{out,goal} = \begin{bmatrix} \cos(\theta_{goal}) \\ \sin(\theta_{goal}) \end{bmatrix} \tag{5.4}$$

The tangent t for every point is then the normalized sum of v_{in} and v_{out}:

$$t = \frac{v_{in} + v_{out}}{\|v_{in} + v_{out}\|} \tag{5.5}$$

The next step is to find the center of the rotation circle for every point, and is done by rotating the tangent t 90 degrees in the direction of rotation ρ between v_{in} and v_{out}, and making it the length R, which is the minimum turning radius of the vehicle from equation (4.2).

$$\rho = -sign(v_{in,y}v_{out,x} - v_{in,x}v_{out,y}) \tag{5.6}$$

The rotated tangent is then:

$$t_{\llcorner} = \rho \begin{bmatrix} -t_N \\ t_E \end{bmatrix} \tag{5.7}$$

And with the rotated tangent calculated the circle center c is:

$$c = wpt_i + Rt_{\llcorner} \tag{5.8}$$

Figure 5.3 shows the initial calculations for every point on the path from Figure 5.2

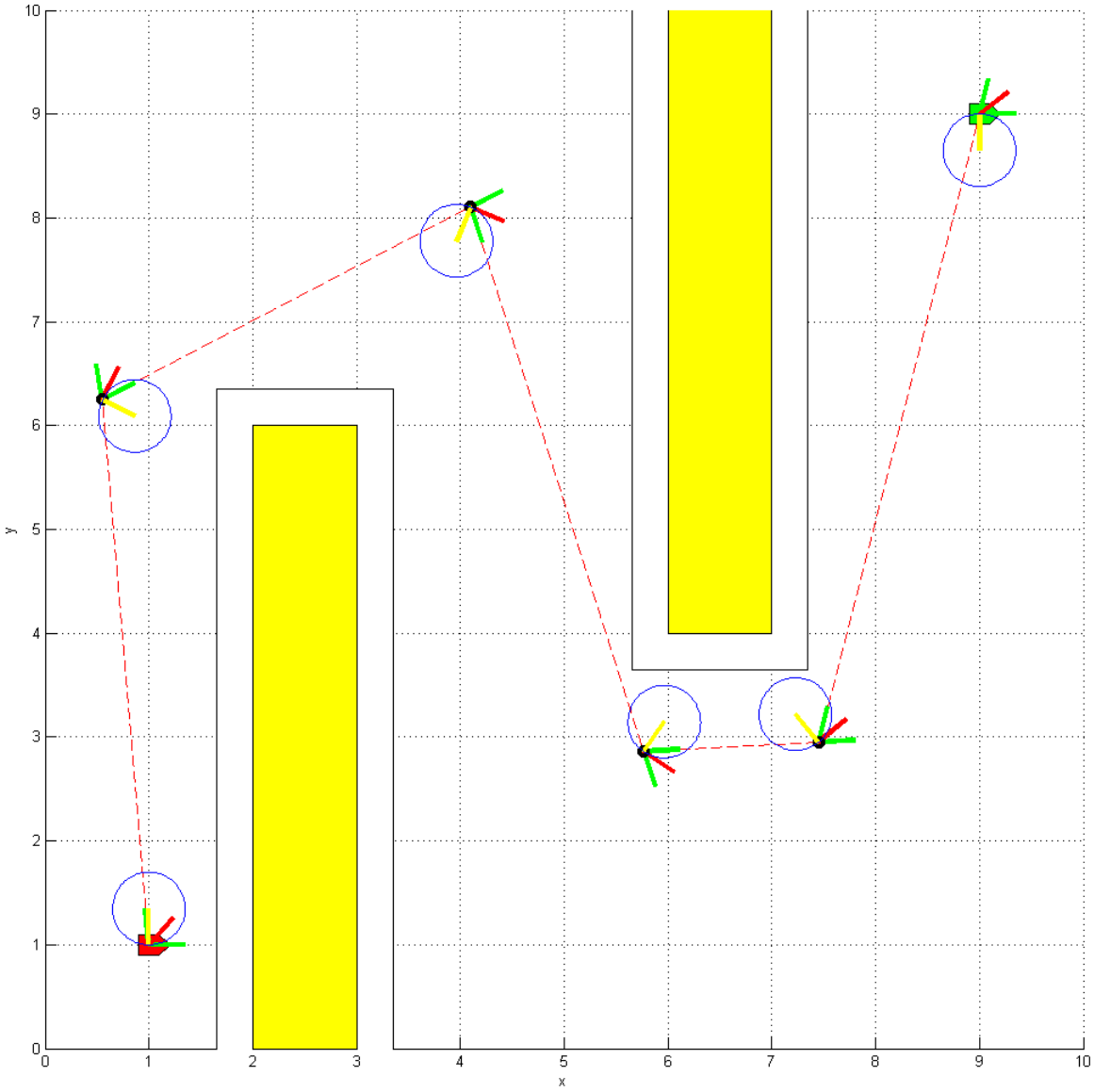


Figure 5.3: Showing the calculated tangents and turning circles for each point

5.2.2 Initial subpath

When creating the initial subpath a small change of the method had to be made to compensate for the freely chosen initial pose θ_{init}. The first step is to calculate the wheel over point p_{wop} for the first waypoint, which is where the linear path from the start point turning circle will connect to the turning circle of the first waypoint. This is done by finding the line H from the start point to the circle center of the waypoint and using trigonometry to extend a leg vector L from the circle center to the turning circle edge.

$$H = wpt_0 - c \tag{5.9}$$

$$\alpha = \arccos\left(\frac{R}{\|H\|}\right) \tag{5.10}$$

The leg vector L is at an angle α in the direction of rotation from the calculated line H. The orientation of H is with the two-argument tangent function:

$$\chi_H = \arctan 2(H_N, H_E) \tag{5.11}$$

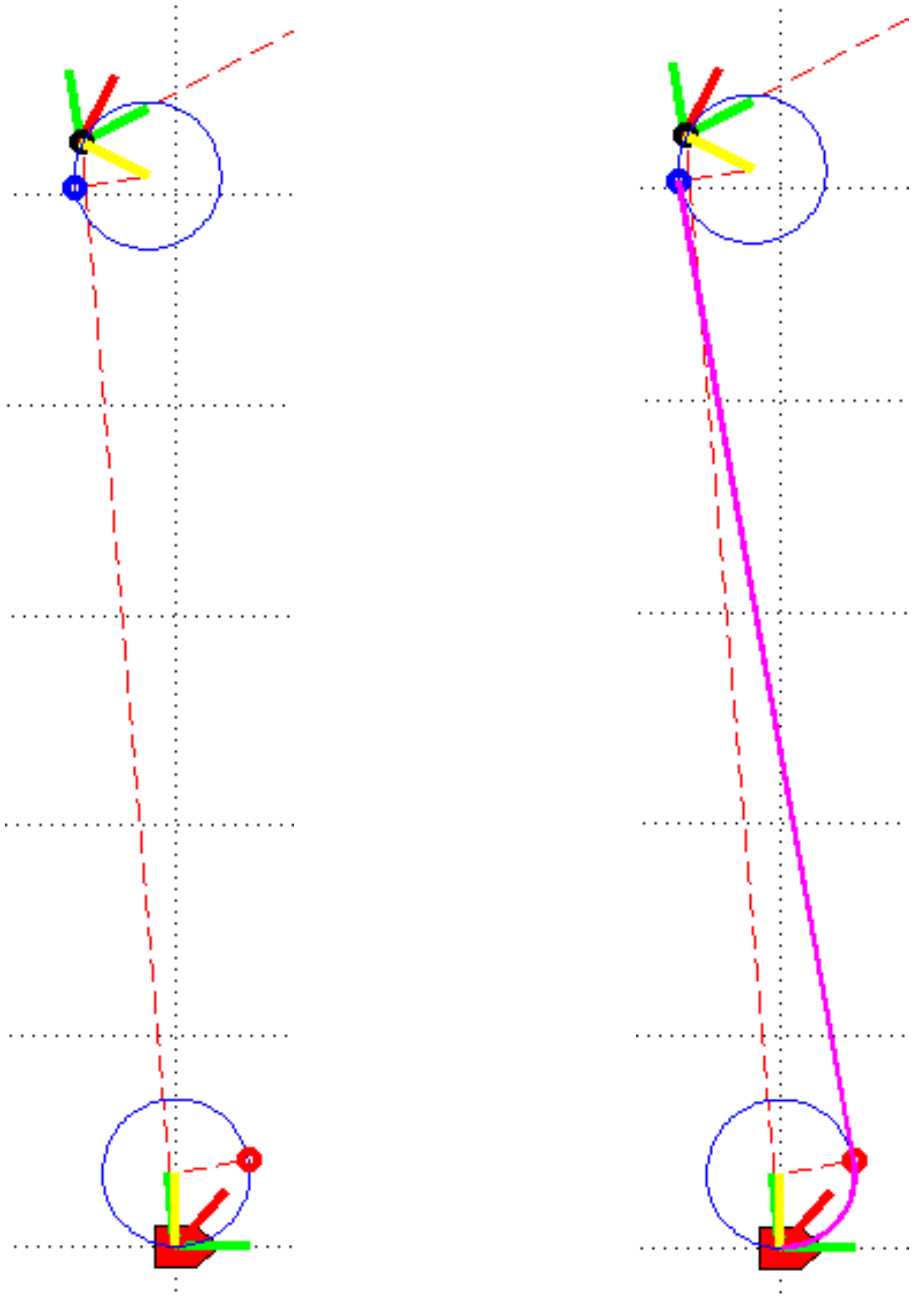
And L can be found by:

$$L = R\begin{bmatrix}\cos(\chi_H + \rho\alpha)\\ \sin(\chi_H + \rho\alpha)\end{bmatrix} \tag{5.12}$$

This makes the wheel over point equal to:

$$p_{wop} = c + L \tag{5.13}$$

In the original method a straight line is then connected from the start point to the wheel over point, but its not realistic to assume that the starting pose is always facing the wheel over point. Therefore the vehicle runs on its own initial turning circle until its directly facing the wheel over point of the first waypoint. Figure 5.4 shows the resulting initial subpath.



(a) Calculated p_{wop} and the pull out point of the first turning circle

(b) Initial subpath completion, a turn followed by a straight line

Figure 5.4: Initial subpath steps

5.2.3 Equivalent consecutive rotations

For equivalent consecutive rotations the first and second waypoint of Figure 5.3 are used as an example. Equivalent consecutive rotations means that the direction of rotation for both waypoints is the same. With v_{in}, v_{out}, the tangent t and the turning circle center for both waypoints already calculated, the next step is to find the pull out point p_{po} for the first waypoint and the wheel over point p_{wop} for the second waypoint.

To locate p_{wop} and p_{po} the vector J from the first waypoint's circle center c_{prev} to the second waypoint's circle center c is needed.

$$J = c - c_{prev} \tag{5.14}$$

Then a leg vector L perpendicular to J is extended with length R from both circle centers. L is found by normalizing J and rotating it 90 degrees opposite of the rotation ρ.

$$L = -\rho R \frac{1}{\|J\|} \begin{bmatrix} -J_N \\ J_E \end{bmatrix} \tag{5.15}$$

With the leg vector the pull out point and wheel over point are:

$$p_{po} = c_{prev} + L \tag{5.16}$$

$$p_{wop} = c + L \tag{5.17}$$

Figure 5.5 shows J and the leg vector L which places the pull out point and wheel over point on the turning circles. It also shows that the path between p_{po} and p_{wop} is parallel to J. Figure 5.6 is the complete subpath for the consecutive equivalent rotation. Moving on the first turning circle until the pull out point is reached, and then extending a straight line over to the wheel over point.

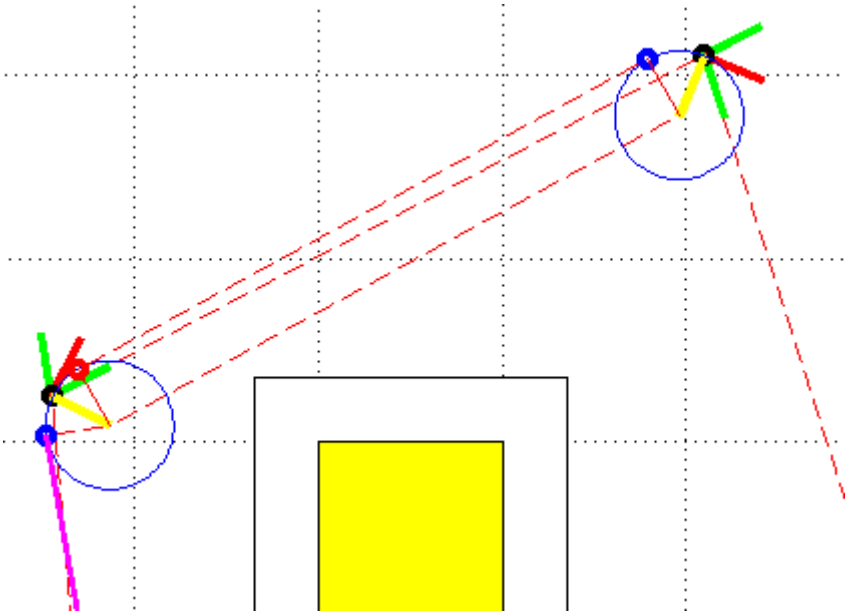


Figure 5.5: Finding p_{po} and p_{wop} in the case with equivalent consecutive rotations

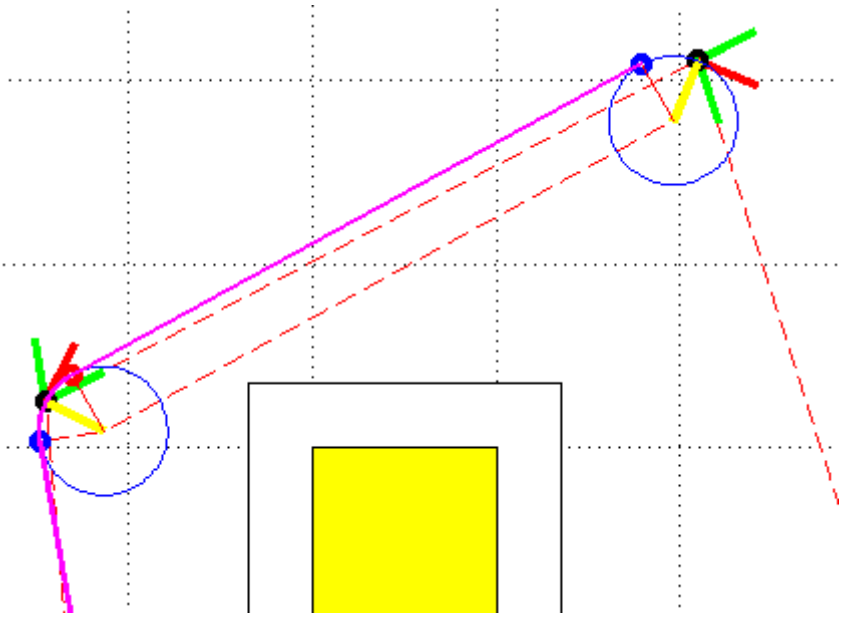


Figure 5.6: Equivalent consecutive rotations complete subpath

5.2.4 Different consecutive rotations

Different consecutive rotations starts out similar to equivalent rotations by using the previously computed circle centers from the two waypoints to find the vector from the first to the second center.

$$J = c - c_{prev} \tag{5.18}$$

Since the direction of rotation for the waypoints is opposite of each other, a symmetric trigonometric relation is used to find the leg vector L.

$$H = \frac{J}{2} \tag{5.19}$$

$$\alpha = \arccos\left(\frac{R}{\|H\|}\right) \tag{5.20}$$

Finding the orientation of H is done with the two-argument tangent function:

$$\chi_H = \arctan 2(H_N, H_E) \tag{5.21}$$

And the leg vector L is found by rotating the angle α in the direction of rotation from H:

$$L = R \begin{bmatrix} \cos(\chi_H + \rho\alpha) \\ \sin(\chi_H + \rho\alpha) \end{bmatrix} \tag{5.22}$$

The wheel over point and pull out points are then calculated:

$$p_{po} = c_{prev} + L \tag{5.23}$$

$$p_{wop} = c - L \tag{5.24}$$

Figure 5.7 shows J and the leg vector L which places the pull out point and wheel over point on the turning circles. Figure 5.8 is the complete subpath for the consecutive equivalent rotation. Moving on the first turning circle until the pull out point is reached, and then extending a straight line over to the wheel over point.

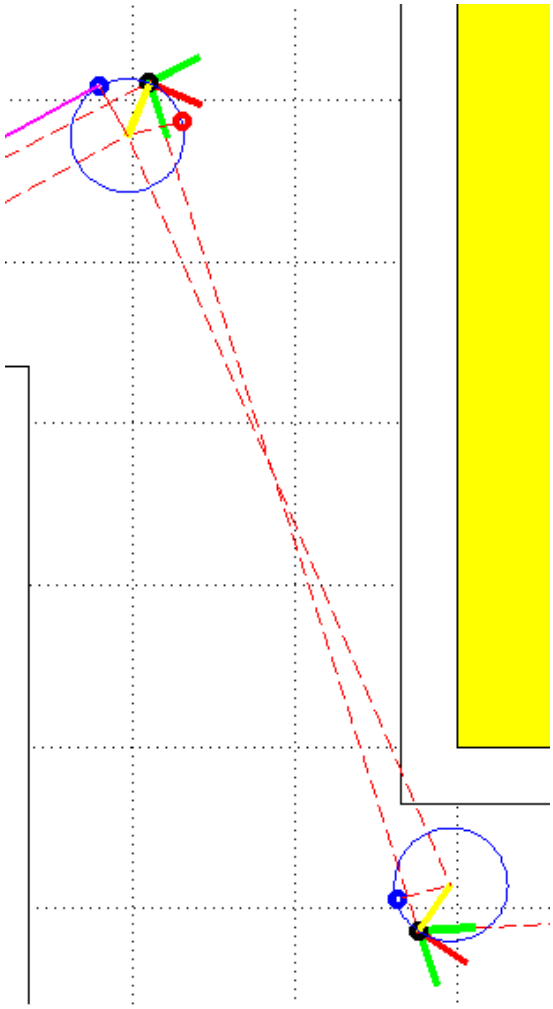


Figure 5.7: Finding p_{po} and p_{wop} with different consecutive rotations

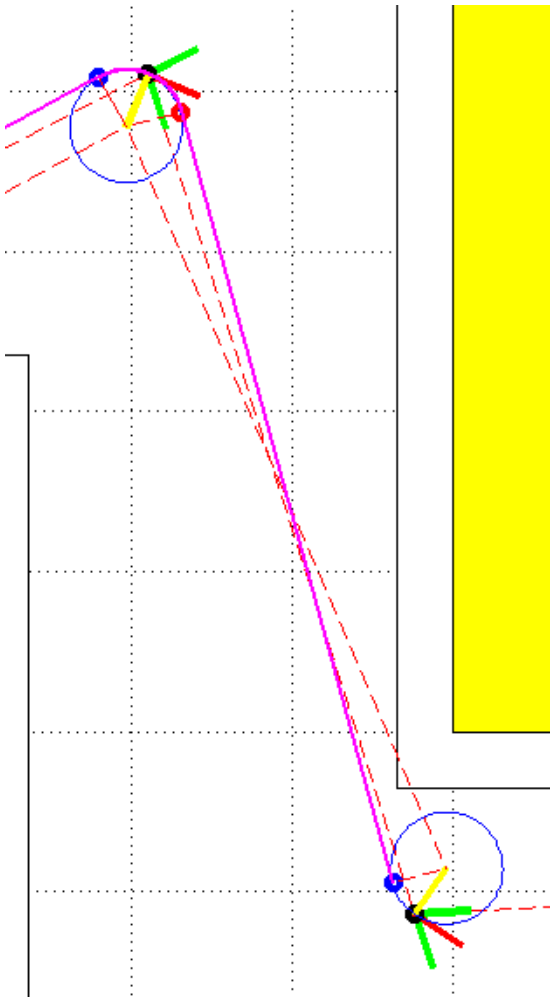


Figure 5.8: Different consecutive rotations complete subpath

5.2.5 Final subpath

When connecting the last interior waypoint to the goal point either the equivalent rotations step or the different rotations step is chosen based on the rotation of the last interior waypoint and the chosen final pose. The wheel over point for the goal point is chosen in a way that makes the vehicle end up at the exact chosen pose as well as position. This is done by rotation the unit vector for the final pose 90 degrees in the direction of rotation, instead of the calculated pose vector.

For the example path from Figure 5.2 the final subpath was a different consecutive rotation path. And the complete Dubins path for the example is plotted in Figure 5.9

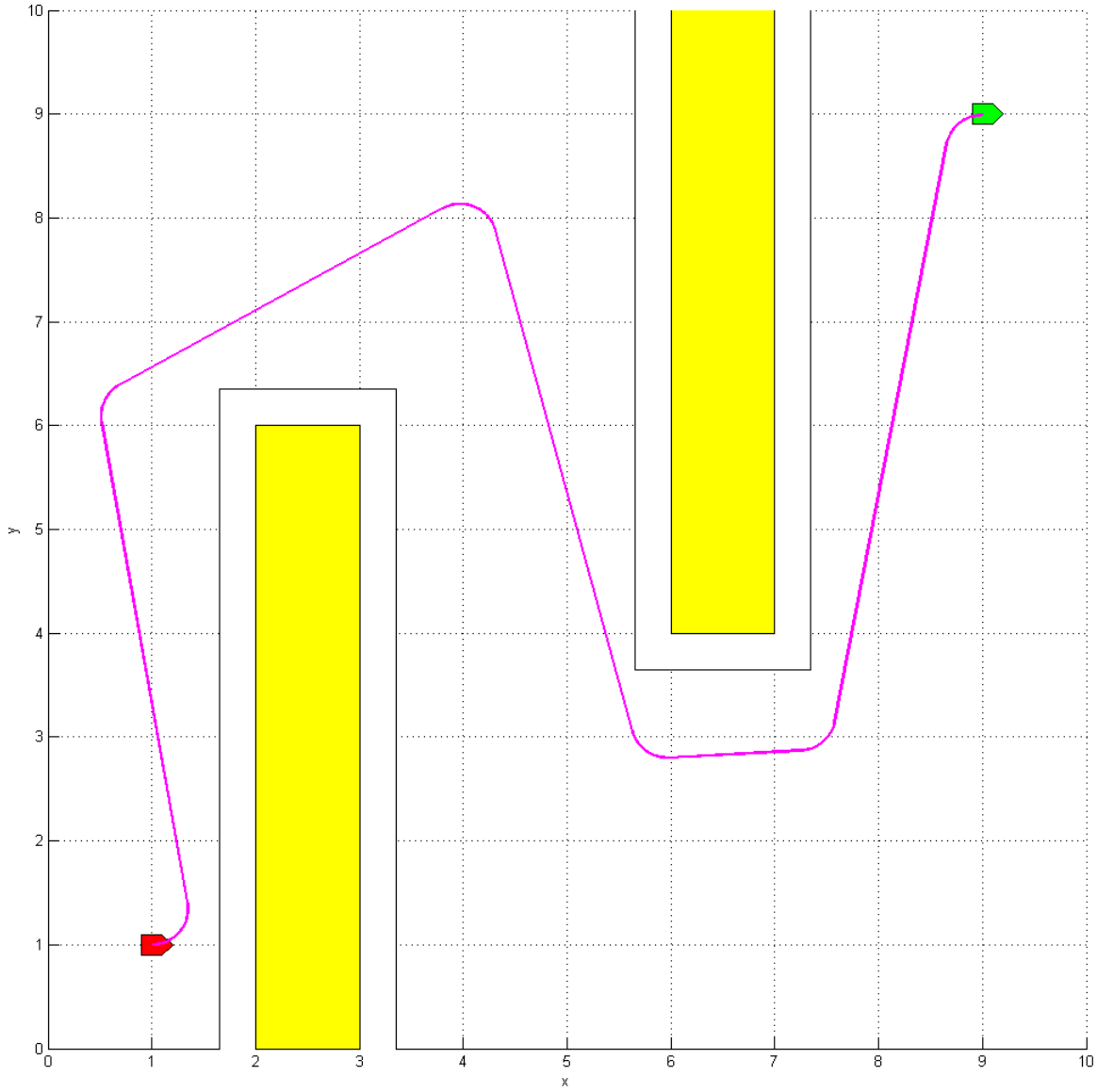


Figure 5.9: Dubins path

5.2.6 Path properties

Figure 5.10 shows some of the data concerning the path from Figure 5.9. The figure shows the north position, east position, heading angle and the curvature along the length of the path.

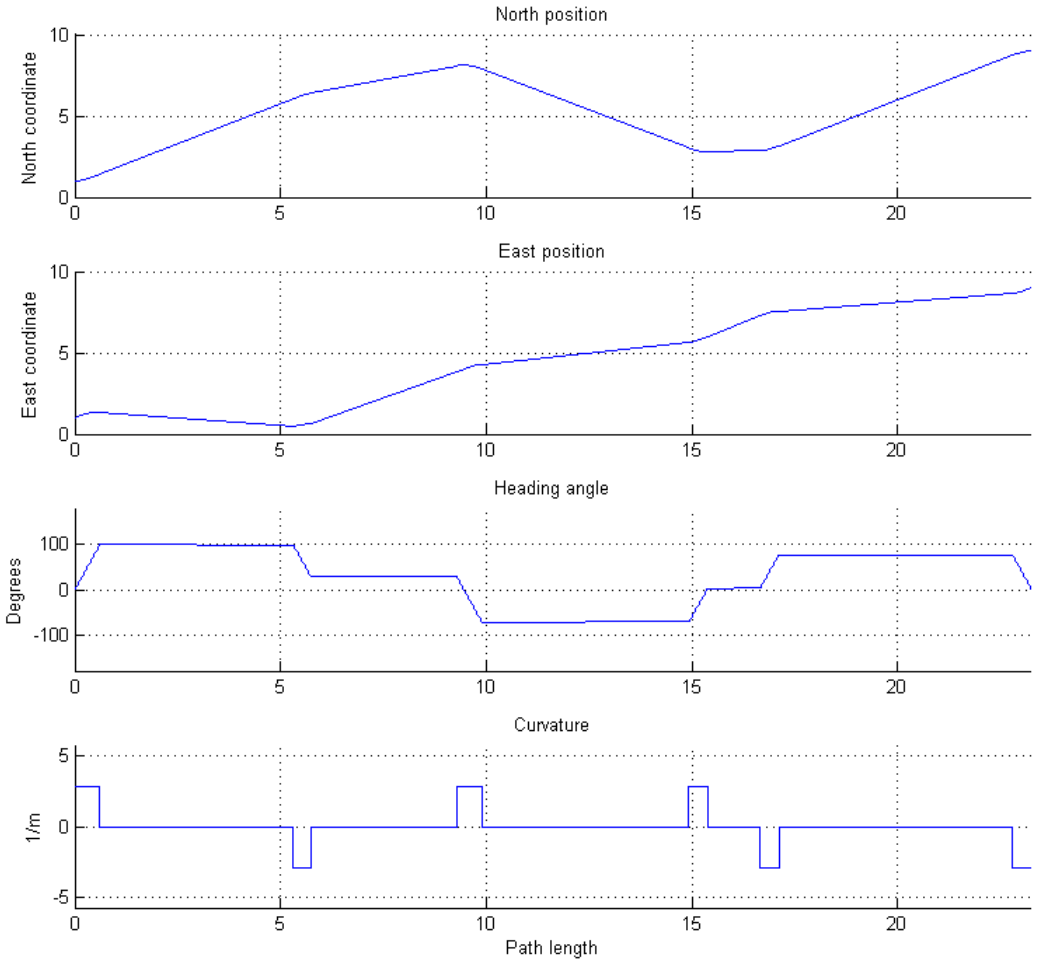


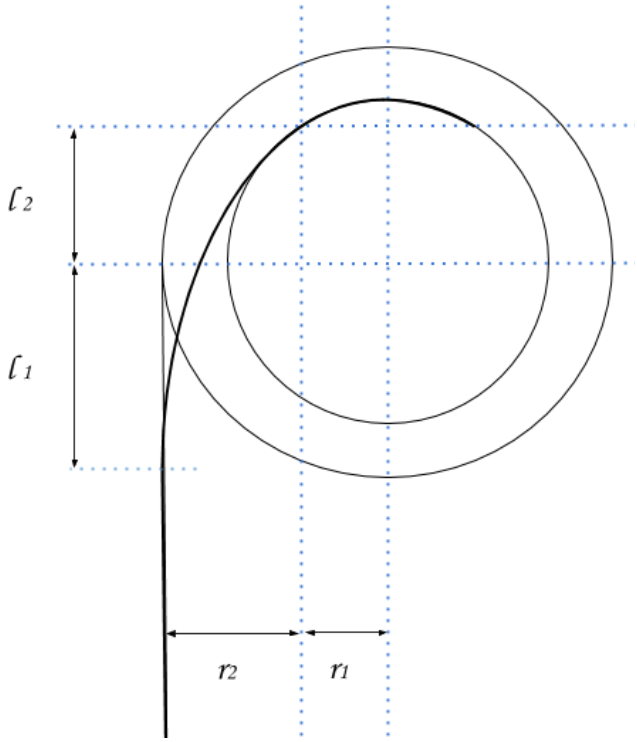
Figure 5.10: Dubins path properties

5.3 Curvature continuity using clothoids

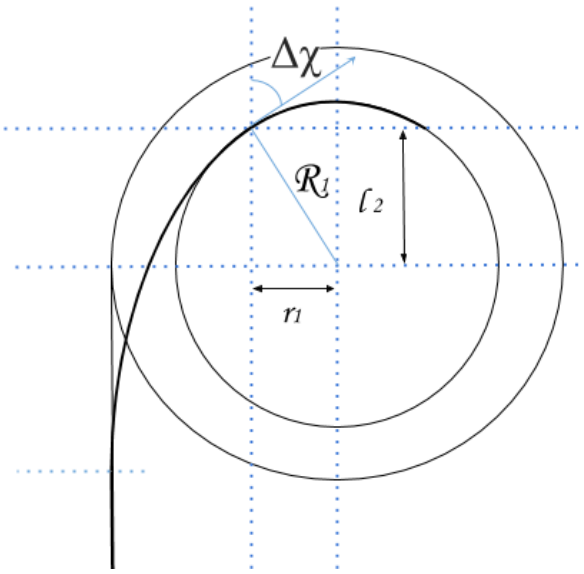
The path properties from Figure 5.10 show that the Dubins path in Figure 5.9 has an instant increase in curvature when entering a circular arc, going from zero curvature to the maximum value instantly. To achieve curvature continuity, a small segment must be added which linearly increases the curvature. This small segment is called a clothoid (see Section 4.5.3), which is a curve designed to gradually increase the curvature with respect to the vehicles properties.

5.3.1 Method

The method to achieve curvature continuity with clothoids is taken from Dahl (2013), and is an extension of the Dubins path algorithm form the previous section. It keeps the minimum turning radius circle as an inner turning circle, and generates an outer circle which is then used to generate a clothoid between the straight line segment and the inner turning circle linearly increasing the curvature.



(a) Line/circle transition



(b) Trigonometric observations

Figure 5.11: The clothoid transition, based on a figure from Dahl (2013)

By extending the method from Section 5.2 with the observations from Figure 5.11 the clothoid can be added as a step in the algorithm between the connection of a straight line segment with a circular arc or vice versa.

The inner circles radius R_1 is still the minimum turning radius, meaning the inverse of the maximum curvature κ_{max}

$$R_1 = \frac{1}{\kappa_{max}} \tag{5.25}$$

The outer circle radius R_2 is

$$R_2 = r_1 + r_2 \tag{5.26}$$

with r_1 as the offset from the circle center c to the clothoid-circle junction perpendicular to the linear path, and r_2 is the clothoid offset perpendicular to the linear subpath. Both r_1 and r_2 can be seen in Figure 5.11.

With a chosen maximum sharpness c_{max}, the length s_{end} of the transition clothoid can be found:

$$s_{end} = \frac{\kappa_{max}}{c_{max}} \tag{5.27}$$

This also leads to the course change $\Delta\chi$, which can also be seen in Figure 5.11b, being

$$\Delta\chi = \frac{1}{2} c_{max} s_{end}^2 \tag{5.28}$$

from the beginning to the end of the clothoid. With trigonometry r_1 can be computed:

$$r_1 = R_1 \cos(\Delta\chi) \tag{5.29}$$

And r_2 can be found via integration:

$$r_2 = \int_0^{s_{end}} \sin(\frac{1}{2} c_{max} \xi^2) d\xi \tag{5.30}$$

The wheel over point p_{wop} is now on the outer circle tangent, but at a distance l_1 before the line intersects with the outer circle. From Figure 5.11a the two distances l_1 and l_2, l_2 being the offset from the circle center to the clothoid-circle junction parallel to the linear path, gives the length of the clothoid projection on the tangent $l_{clothoid}$:

$$l_1 = l_{clothoid} - l_2 \tag{5.31}$$

And:

$$l_{clothoid} = \int_0^{s_{end}} \cos\left(\frac{1}{2} c_{max} \xi^2\right) d\xi \tag{5.32}$$

Finally, with the trigonometry from Figure 5.11b l_2 can be computed:

$$l_2 = R_1 \sin(\Delta\chi) \tag{5.33}$$

5.3.2 Final subpath and path properties

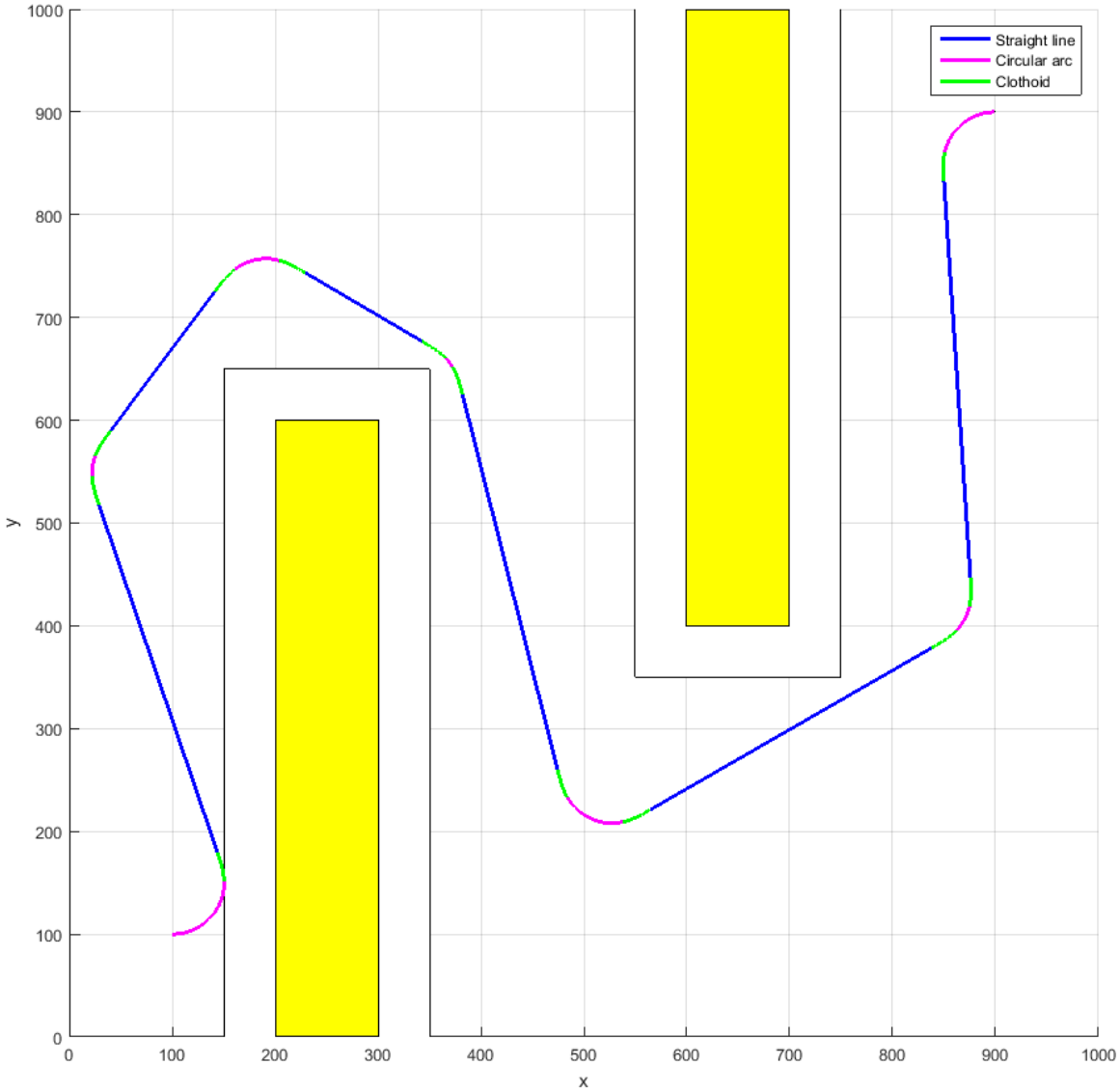


Figure 5.12: Dubins with clothoid transition path

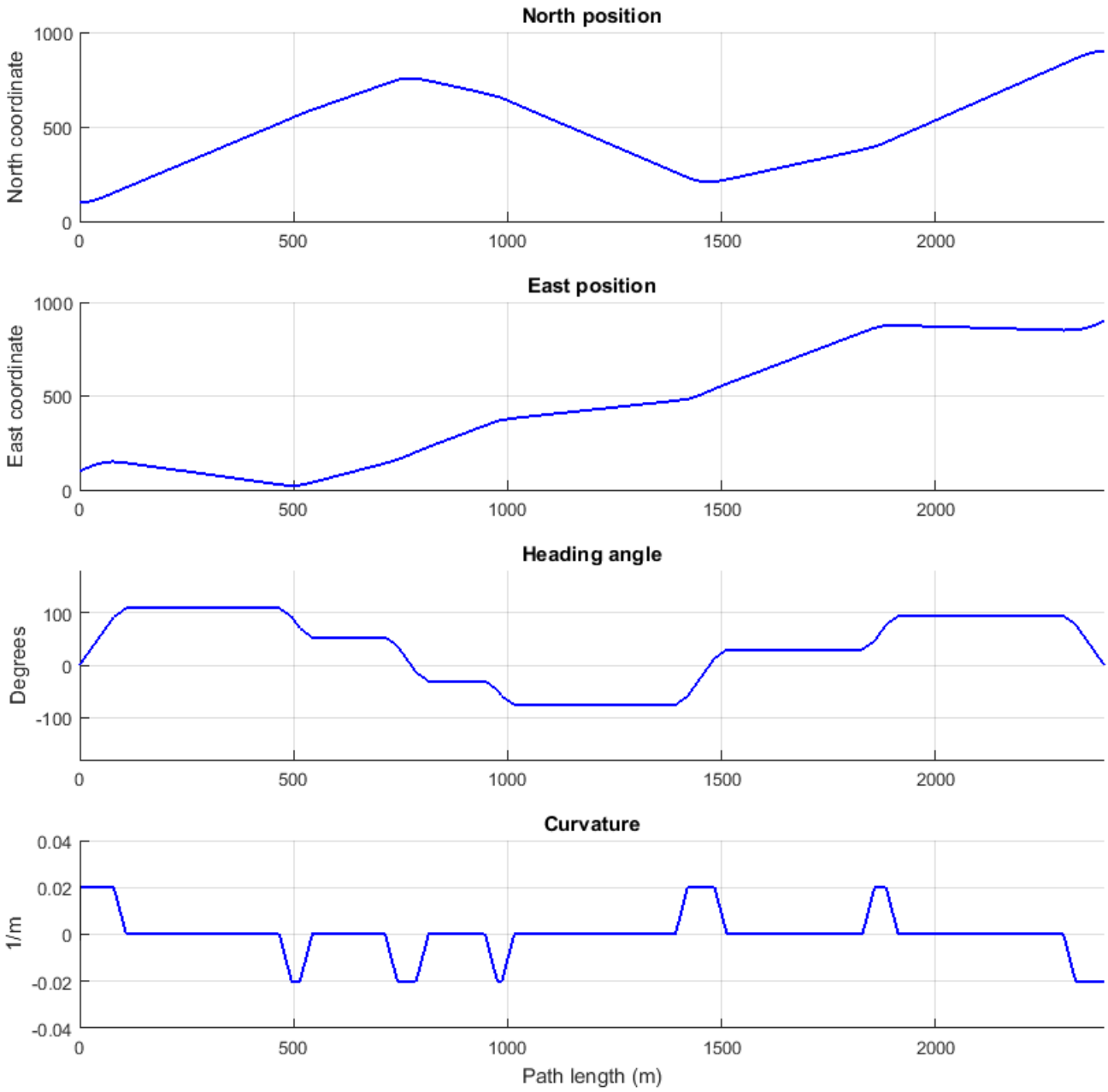


Figure 5.13: Dubins with clothoid transition path properties

5.4 Curvature continuity using Fermat's spiral

In Lekkas et al. (2013), the Fermat's spiral was suggested as a candidate for creating simple paths consisting of straight lines and arc segments. The Fermat's spiral holds has the same properties as the clothoid (curvature continuity) but has a much lower computational time. The method is similar to the clothoid transition method in Figure 5.11.

5.4.1 Method

With a given maximum curvature κ_{max}, meaning $R_1 = 1/\kappa_{max}$, and the point of maximal curvature being:

$$\theta_{\kappa_{max}} = min\left(\theta_{end}, \sqrt{\frac{\sqrt{7}}{2} - \frac{5}{4}}\right) \tag{5.34}$$

where θ_{end} is the chosen point of the curve where the maximum curvature appears, the scaling constant a (can also be written as k) can be determined:

$$a = \frac{1}{\kappa_{max}} \frac{2\sqrt{\theta_{\kappa_{max}}}(3 + 4\theta^2_{\kappa_{max}})}{(1 + \theta^2_{\kappa_{max}})^{\frac{3}{2}}} \tag{5.35}$$

With the scaling constant a calculated both the length of the FS and the value r_2 can be computed:

$$r_2 = a\sqrt{\theta_{end}}\sin(\theta_{end}) \tag{5.36}$$
$$l_{fermat} = a\sqrt{\theta_{end}}\cos(\theta_{end}) \tag{5.37}$$

And the rest of the required values are computed as:

$$\chi_{end} = \theta_{end} + \arctan(2\theta_{end}) \tag{5.38}$$
$$r_1 = R_1\cos(\chi_{end}) \tag{5.39}$$
$$l_2 = R_1\sin(\chi_{end}) \tag{5.40}$$
$$l_1 = l_{fermat} - l_2 \tag{5.41}$$
$$R_2 = r_1 + r_2 \tag{5.42}$$

5.4.2 Final subpath and path properties

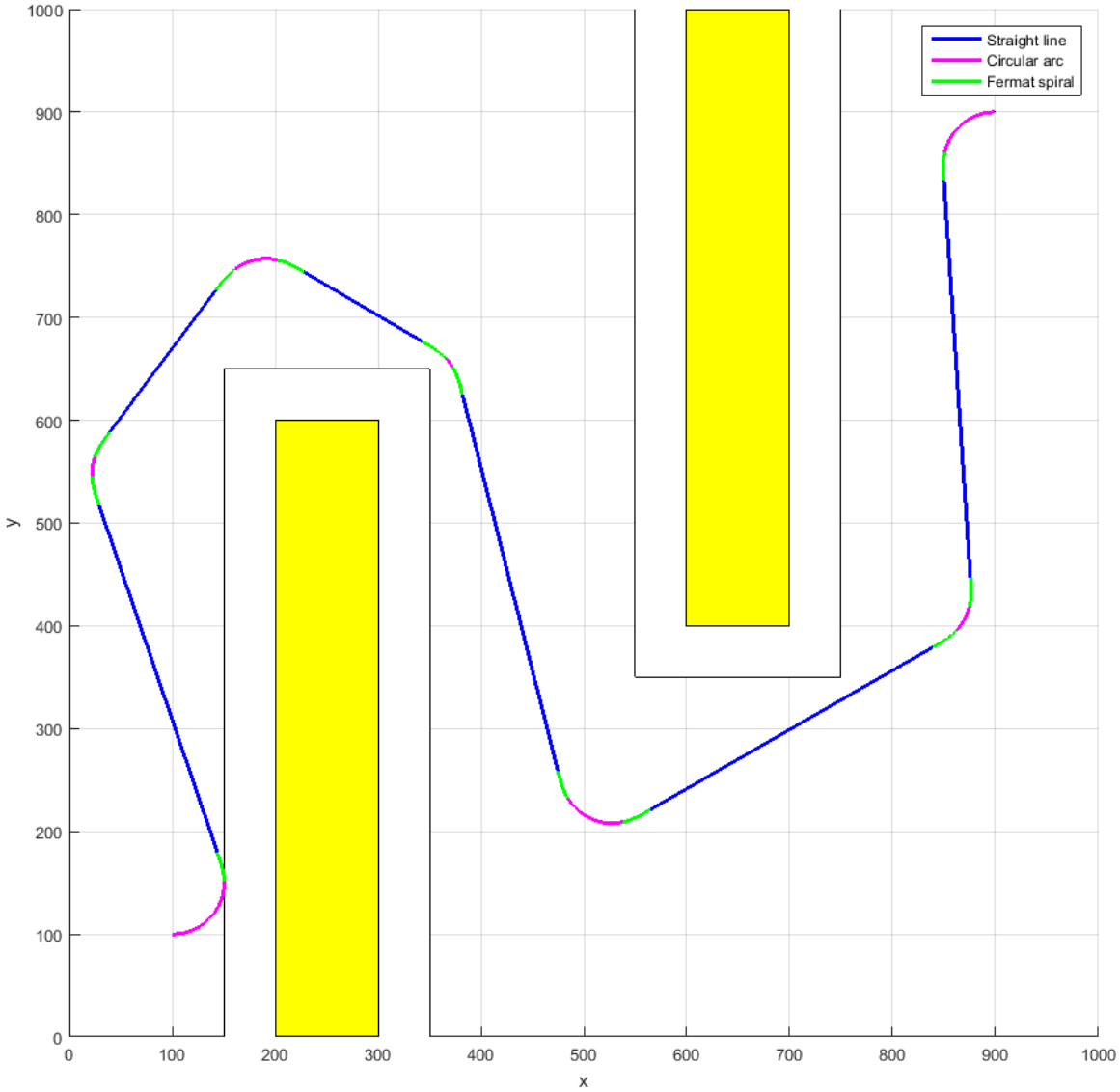


Figure 5.14: Dubins with Fermat transition path

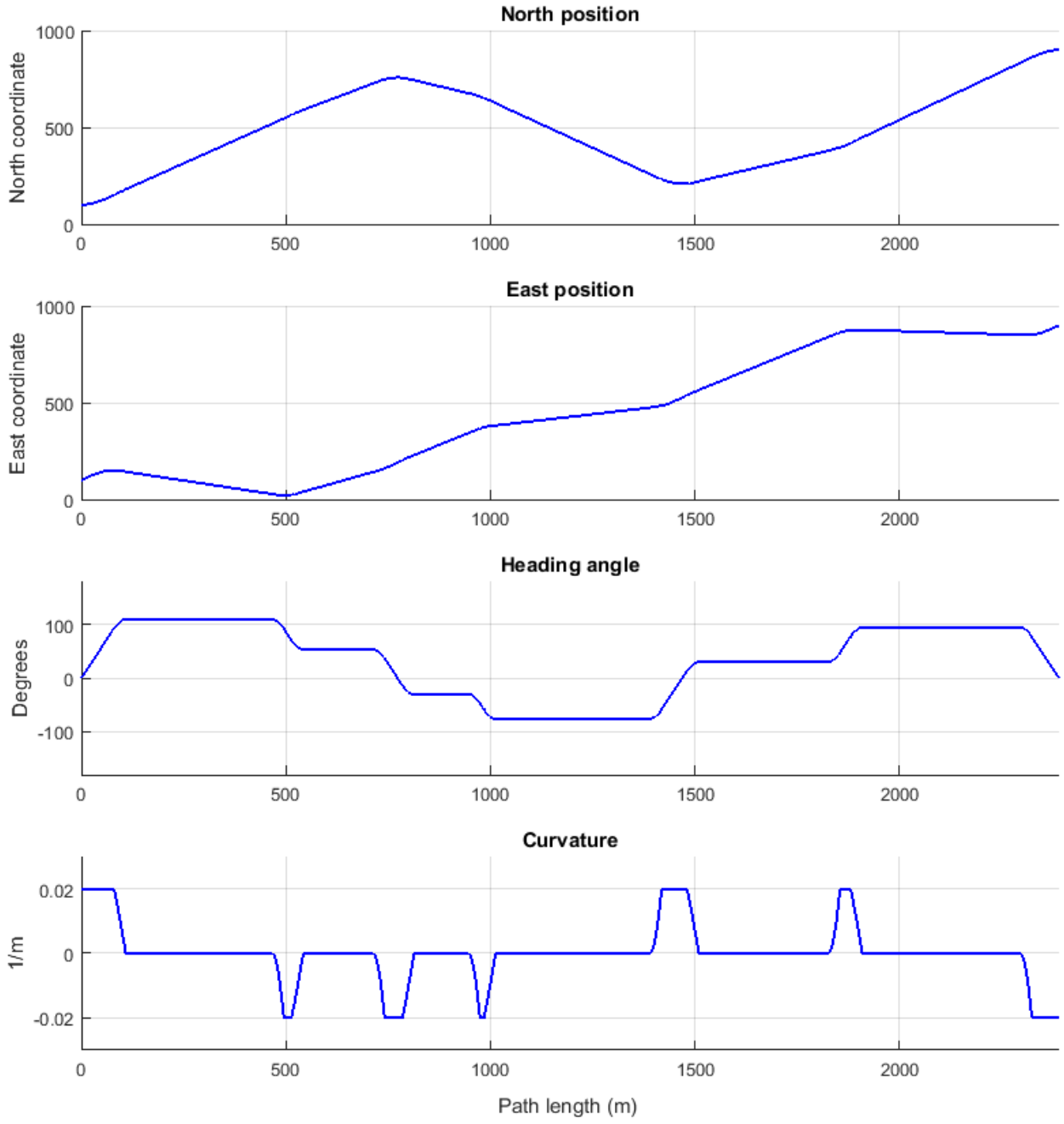


Figure 5.15: Dubins with Fermat transition path properties

5.5 Comparing the methods

5.5.1 Visual comparison

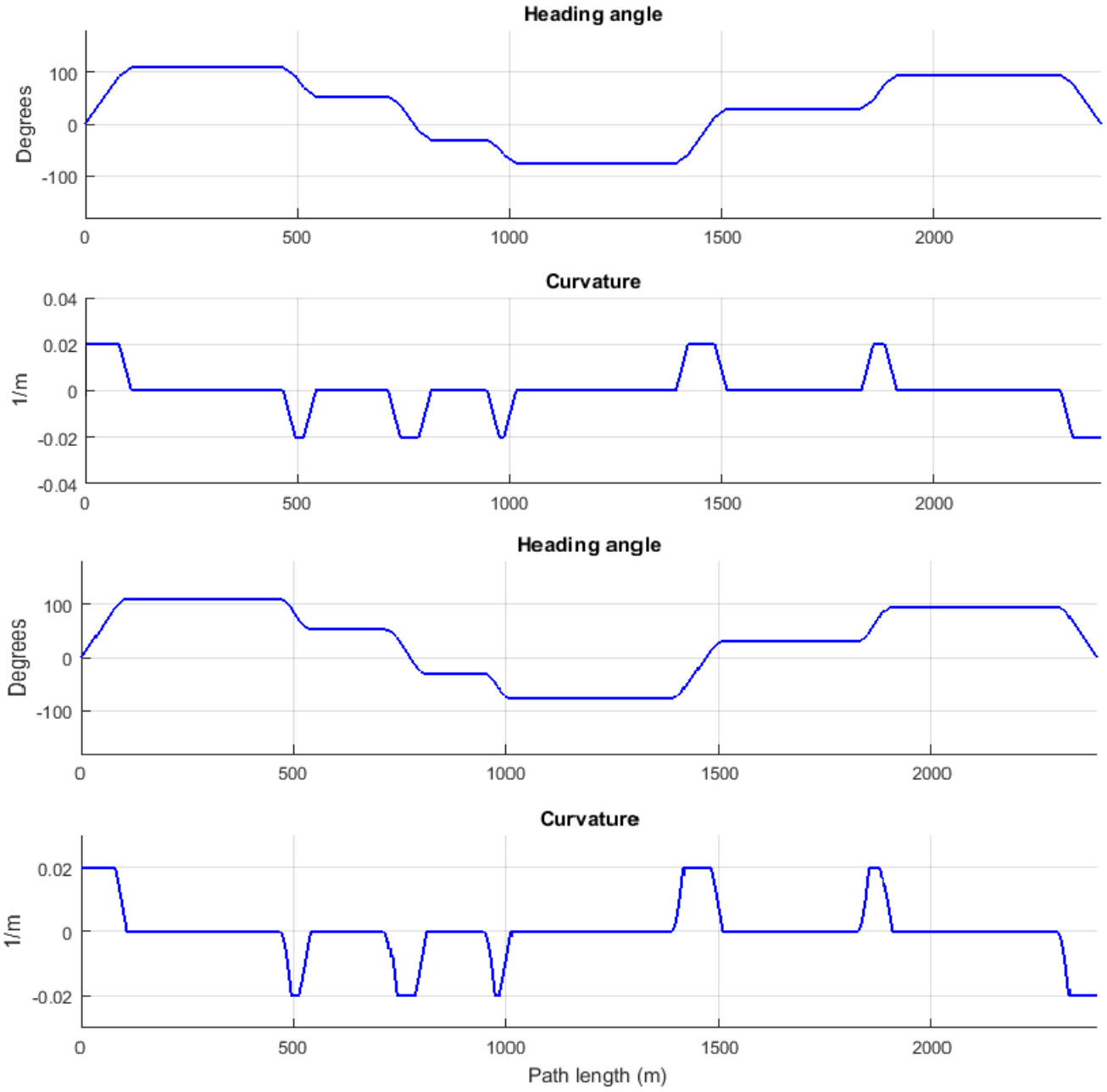


Figure 5.16: Clothoid (top two) and Fermat's spiral (bottom two) data compared

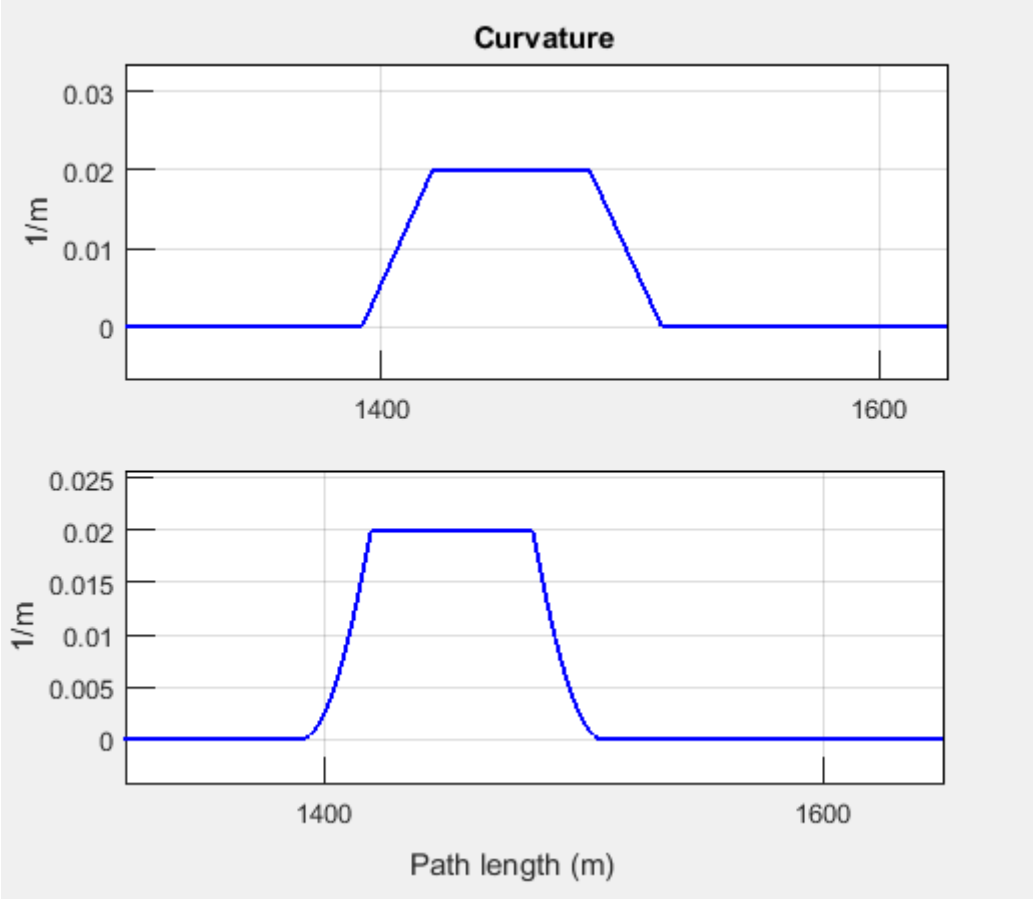


Figure 5.17: Closeup of the clothoid (top) and Fermat's spiral curvature at the same waypoint

Figure 5.16 shows how both the clothoid transition and FS transition have very similar results when looking at the heading angle. The change in angle is now smooth unlike the results without the transitions which were only linear (see Figure 5.10). This is a wanted feature for all vessels with a rudder as only linear changes in the heading angle would cause the rudder to skip (change in an instant) which is not possible for a rudder due to counteracting hydrodynamical forces.

Looking closer at the curvature for the clothoid transition and the FS transition the difference in curvature increase is visible. The clothoid transition shows a linear increase as expected from equation (4.19), and the FS shows the expected increase from equation (4.29). The main advantage of the FS transition is that its starts slower than the clothoid and may therefore contribute to a more stable system, however it catches up in the end with a much faster increase in curvature which can be unwanted.

The length of the path for three different methods is negligible, but naturally the original Dubins' path is somewhat shorter.

5.5.2 Computational time comparison

Implementing the methods in MATLAB showed huge differences in the time it took for the methods to generate a path using the same basis. Even though the two transitions are very similar in the way they are computed, the Fresnel integrals (equations (4.17) and (4.18)) used to generate the clothoid take a lot of computational time. For the specific runs plotted in figures 5.12 and 5.14 where the basis was the same the clothoid used 3.44 seconds while the FS used 0.00241 seconds. This difference in computational time is consistent for any basis, and the mean run time is around 2-3 seconds for the clothoid and around 0.002 seconds for the FS when planning in the workspace seen in their plots. The same difference in run time between the methods is expected for any workspace size. For the original Dubin's path the computational time is observed to be around the same as the FS.

5.5.3 Geometric continuity comparison

When classifying the smoothness of a path, geometric continuity is often used as the measure. Considering the intermediate waypoints of the generated paths, the basic measure of geometric continuity is G^n, where n describes the smoothness. In short terms G^0 only means the curves at each side of the waypoint connecting them. G^1 means the curves on each side also share a common tangent direction at the waypoint. And G^2 means they also share a common center of curvature at the joint point. The geometric continuity of the paths in this section are summarized in Table 5.2.

Method	G^0/C^0	G^1	G^2
Waypoint-reduced RRT (Piecewise linear)	✓	✗	✗
Dubins path (Circular arc smoothing)	✓	✓	✗
Dubins path w/ clothoid transition	✓	✓	✓
Dubins path w/ Fermat transition	✓	✓	✓

Table 5.2: 2D RRT Dubins path input

Chapter 6

Path Tracking and Ocean Current Estimation

To compare the different paths generated in the previous section, they can be used as input in a path tracking guidance system. In this thesis, an extension of the guidance system based of the work in Lekkas and Fossen (2014) is considered. In Lekkas and Fossen (2014) a guidance system for 2-D straight-line path tracking applications for underactuated marine vehicles exposed to ocean currents was developed. The system must therefore be extended to include tracking of circular arcs and clothoid/FS transitions.

6.1 Tracking segments

In a path tracking system the task of the vehicle is to track a virtual vehicles position (the target), in other words minimize the difference between its current position p and the virtual vehicles position p_t such that $p - p_t \to 0$. How to update the targets position differs between the type of line the target moves on.

6.1.1 Straight line

When the target moves on a straight line between to waypoints the path-tangential angle γ_p is constant between waypoints (meaning it only has to be computed once for the entire segment):

$$\gamma_p = \arctan 2(y_{k+1} - y_k, x_{k+1} - x_k) \tag{6.1}$$

with waypoints as (x_k, y_k) for $k = 1, 2, ..., N$. The position of the target $p_n^t = (x_t, y_t)$ can then be calculated by assuming its total speed to be $U_t > 0$ as:

$$\dot{x}_t = U_t \cos(\gamma_p) \tag{6.2}$$
$$\dot{y}_t = U_t \sin(\gamma_p) \tag{6.3}$$

Notice that in the case of a straight line there is no need to use the path parametrization from Section 4.5.1 as a simple check can determine when the segment is finished, instead of updating a path parameter ϖ and stopping when it reaches the upper value.
The position error for a vehicle at position $p = (x, y)$ is given by:

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = \mathbf{R}^\top(\gamma_p) \begin{bmatrix} x - x_t \\ y - y_t \end{bmatrix} \tag{6.4}$$

where $\mathbf{R}$ is the classic rotation matrix:

$$\mathbf{R}(\gamma_p) = \begin{bmatrix} \cos(\gamma_p) & -\sin(\gamma_p) \\ \sin(\gamma_p) & \cos(\gamma_p) \end{bmatrix} \tag{6.5}$$

Written out, equation (6.4) yields the along-track and cross-track error equations:

$$x_e = (x - x_t)\cos(\gamma_p) + (y - y_t)\sin(\gamma_p) \tag{6.6}$$
$$y_e = -(x - x_t)\sin(\gamma_p) + (y - y_t)\cos(\gamma_p) \tag{6.7}$$

With the along-track and cross-track error calculated an algorithm is used to control the actual vehicle to minimize the errors, this is explained in detail in Lekkas and Fossen (2014). The principle of tracking a straight line is illustrated in Figure 6.1.

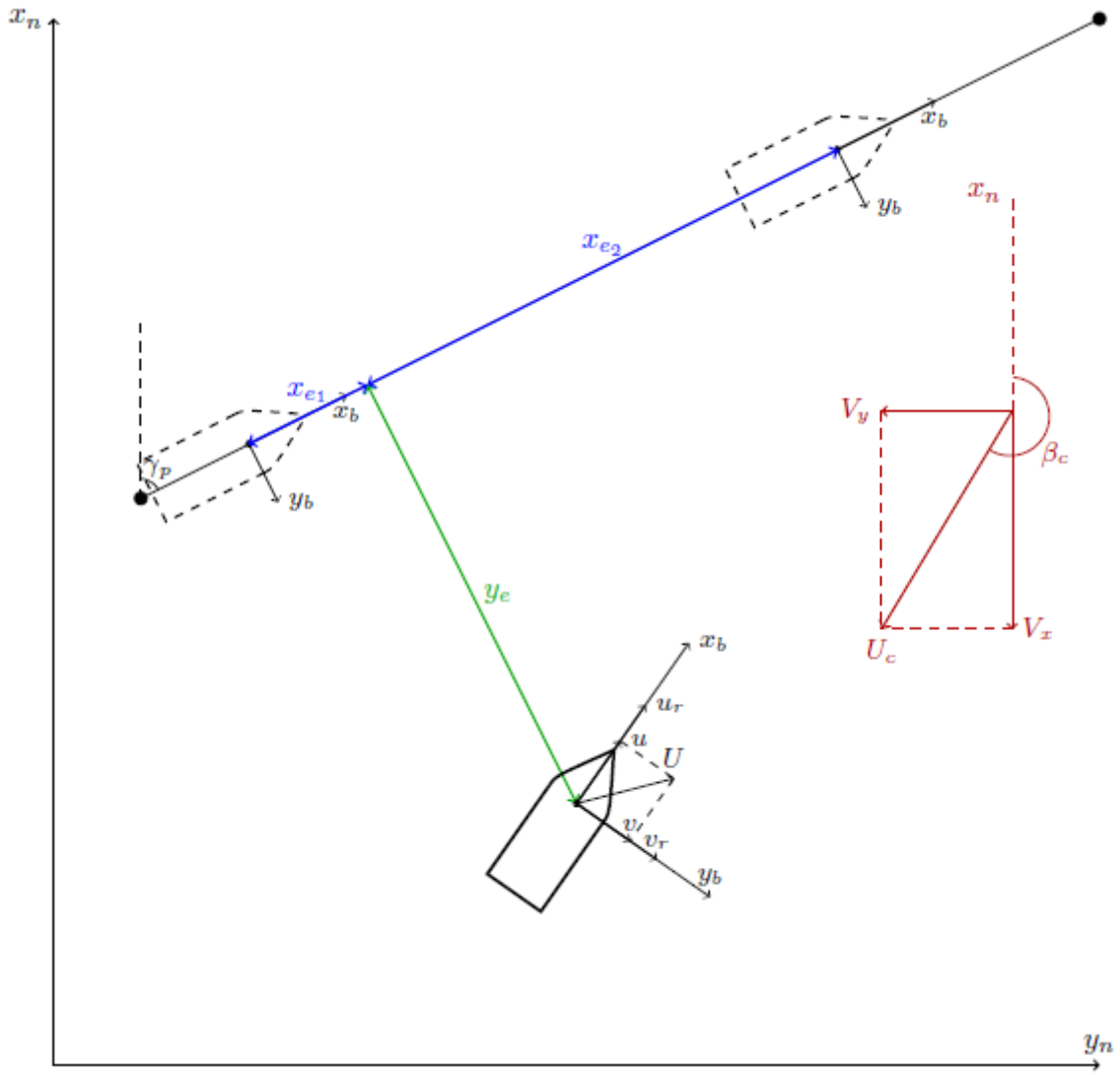


Figure 6.1: Tracking a straight-line path. The vehicle (solid-line) tracks the virtual vehicle (dotted line) which is moving on a straight line while under the influence of unknown ocean currents (red frame), taken from Lekkas and Fossen (2014)

6.1.2 Circular arc

Moving the target on a circular arc means that the path-tangential angle is not constant, and must be updated for each step. In this case the path parametrization for circular arcs (see Section 4.5.2) is used. When the target enters a circular arc segment the path parameter is set to zero ($\varpi = 0$) and the arcs parameters are found in the paths path table (as in Section 4.5.5). The targets next position is then found by inserting the path parameter and the arcs parameters into the path parametrization:

$$\mathbf{p}_{cir}(\varpi) = \begin{bmatrix} c_x + R\cos(\alpha_0 + \varpi(\alpha_1 - \alpha_0)) \\ c_y + R\sin(\alpha_0 + \varpi(\alpha_1 - \alpha_0)) \end{bmatrix} \tag{6.8}$$

where (c_x, c_y) is the circle centre, R is the radius and α_0 and α_1 are the angles which the circular arc starts and ends, respectively. The path tangential angle is:

$$\gamma_p(\varpi) = \begin{cases} (\alpha_0 + \varpi(\alpha_1 - \alpha_0)) + 90^\circ & \text{when } \alpha_0 < \alpha_1 \text{ and,} \\ (\alpha_0 + \varpi(\alpha_1 - \alpha_0)) - 90^\circ & \text{when } \alpha_0 > \alpha_1 \end{cases} \tag{6.9}$$

and the speed of the target $\mathbf{v}(\varpi) = \frac{d}{d\varpi}\mathbf{p}(\varpi)$ is computed:

$$\frac{d}{d\varpi}\mathbf{p}_{cir}(\varpi) = \begin{bmatrix} -(\alpha_1 - \alpha_0)R\sin(\alpha_0 + \varpi(\alpha_1 - \alpha_0)) \\ (\alpha_1 - \alpha_0)R\cos(\alpha_0 + \varpi(\alpha_1 - \alpha_0)) \end{bmatrix} \tag{6.10}$$

To use the circular arc segment for path-tracking the derivative of the parameter w.r.t must be defined. According to both Breivik and Fossen (2004) and Fossen (2011), for a desired vehicle speed $U_d(t)$ (chosen as a constant in this thesis) the parameter derivative w.r.t is determined by:

$$\dot{\varpi} = \frac{U_d(t)}{\sqrt{x_p'(\varpi)^2 + y_p'(\varpi)^2}} \tag{6.11}$$

where $x_p'(\varpi)$ and $y_p'(\varpi)$ is the result of $\frac{d}{d\varpi}\mathbf{p}(\varpi) = \left[x_p'(\varpi), y_p'(\varpi)\right]^T$.

The parameter is then updated as $\varpi_{next} = \varpi + \dot{\varpi}$ and the along-track and cross-track errors are calculated as in equation (6.6) and (6.7). The circular arc parametrization is a unit domain parametrization, meaning $\varpi \in \left[0, 1\right]$, which in turn means that target has reached the end of the segment when $\varpi = 1$.

6.1.3 Clothoid

In this thesis a unit domain parametrization is used when implementing the parametrization for the clothoid. To make the clothoid segment unit parametrized the path parameter ϖ must be scaled with the length of the segment s_{end}:

$$\mathbf{p}_{clothoid}(\varpi) = \begin{bmatrix} N_0 + \int_0^{\varpi s_{end}} \cos(\frac{1}{2}c\xi^2 + \kappa_0\xi + \chi_0)\ d\xi \\ E_0 + \int_0^{\varpi s_{end}} \sin(\frac{1}{2}c\xi^2 + \kappa_0\xi + \chi_0)\ d\xi \end{bmatrix} \tag{6.12}$$

with the values found in the path table entry for that segment as explained in sections 4.5.5 and 4.5.3. The path tangential angle for the segment is found by:

$$\gamma_p(\varpi) = \frac{1}{2}c(\varpi s_{end})^2 + \kappa_0 \varpi s_{end} + \chi_0 \tag{6.13}$$

and the speed parametrization $\mathbf{v}(\varpi) = \frac{d}{d\varpi}\mathbf{p}(\varpi)$ is:

$$\frac{d}{d\varpi}\mathbf{p}_{clothoid}(\varpi) = \begin{bmatrix} s_{end}\cos(\frac{1}{2}c\xi^2 + \kappa_0\xi + \chi_0) \\ s_{end}\sin(\frac{1}{2}c\xi^2 + \kappa_0\xi + \chi_0) \end{bmatrix} \tag{6.14}$$

For each step the path parameter ϖ is updated as in equation (6.11) and the along-track and cross-track errors are calculated, until the target reaches the end of the segment, at which in the unit parametrization case the path parameter is $\varpi = 1$ since $\varpi \in [0, 1]$.

6.1.4 Fermat's spiral

Finding the derivative of the FS parametrization in Section 4.5.4 is pretty straightforward:

$$\frac{d}{d\varpi}\mathbf{p}_{fermat}(\varpi) = \frac{k}{2\sqrt{\varpi}} \begin{bmatrix} \cos(\rho\varpi + \chi_0) - 2\rho\varpi\sin(\rho\varpi + \chi_0) \\ \sin(\rho\varpi + \chi_0) + 2\rho\varpi\cos(\rho\varpi + \chi_0) \end{bmatrix} \tag{6.15}$$

however the fraction $k/(2\sqrt{\varpi})$ creates a singularity at $\varpi = 0$. This is a drawback as both velocity and acceleration are undefined at the beginning of an entering FS segment and at the end of an exiting FS segment, which in turn makes the FS appear unsuitable for path-tracking. Luckily this is a property of the parametrization and in Lekkas et al. (2013) a change of variables is done to create a singularity free parametrization:

$$u := \sqrt{\varpi} \Rightarrow 0 \leq u \leq \sqrt{\varpi_{max}} \tag{6.16}$$

Where ϖ_{max} is the path table value θ_{end}. With the change of parameters, the parametrization for the FS entering segment becomes:

$$\mathbf{p}_{fermat}(u) = \begin{bmatrix} x_0 + ku\cos(\rho u^2 + \chi_0) \\ y_0 + ku\sin(\rho u^2 + \chi_0) \end{bmatrix} \tag{6.17}$$

which makes a singularity free expression for the velocity:

$$\frac{d}{du}\mathbf{p}_{fermat}(u) = k \begin{bmatrix} \cos(\rho u^2 + \chi_0) - 2\rho u^2\sin(\rho u^2 + \chi_0) \\ \sin(\rho u^2 + \chi_0) + 2\rho u^2\cos(\rho u^2 + \chi_0) \end{bmatrix} \tag{6.18}$$

The exiting segments position and velocity is parametrized as follows:

$$\mathbf{p}_{\overline{fermat}}(u) = \begin{bmatrix} x_0 + k(u_{end} - u)\cos(\rho(u - u_{end})^2 + \chi_0) \\ y_0 + k(u_{end} - u)\sin(\rho(u - u_{end})^2 + \chi_0) \end{bmatrix} \tag{6.19}$$

$$\frac{d}{du}\mathbf{p}_{\overline{fermat}}(u) = k \begin{bmatrix} \cos(\rho(u-u_{end})^2 + \chi_0) - 2\rho(u_{end}-u)^2 \sin(\rho(u-u_{end})^2 + \chi_0) \\ \sin(\rho(u-u_{end})^2 + \chi_0) + 2\rho(u_{end}-u)^2 \cos(\rho(u-u_{end})^2 + \chi_0) \end{bmatrix} \tag{6.20}$$

Updating the path parameter is done in the same way as for the circular arc and clothoid, namely equation (6.11). An important thing to note is that this is not a unit domain parametrized as $u \in [0, \sqrt{\varpi_{max}}]$. The segment is therefore over when the path parameter reaches $u = \sqrt{\varpi_{max}}$.

6.2 Vehicle kinematics

For the guidance system used in this thesis, the vehicle kinematic equations for horizontal plane motions are taken from Lekkas and Fossen (2014). In Lekkas and Fossen (2014) the equations can be expressed in terms of the relative surge and sway velocities $u_r = u - u_c$ and $v_r = v - v_c$ as:

$$\dot{x} = u_r \cos(\psi) - v_r \sin(\psi) + V_x \tag{6.21}$$
$$\dot{y} = u_r \sin(\psi) + v_r \cos(\psi) + V_y \tag{6.22}$$
$$\dot{\psi} = r \tag{6.23}$$

with ψ as the yaw angle and r as the yaw rate. V_x and V_y are the ocean current velocities in the North-East frame which are constant in NED, while the body-fixed velocities u_c and v_c depend on the heading ψ. Together they satisfy:

$$[u_c, v_c]^\top = \mathbf{R}^\top(\psi)[V_x, V_y]^\top \tag{6.24}$$

where $\mathbf{R}^\top(\psi)$ is the transforming matrix.

6.3 Ocean current

In this thesis, we use a constant current, however the way a constant current affects a moving vehicle depends on the position of the vehicle, and must therefore be estimated for each step. From Figure 6.1, we have the ocean current velocity magnitude U_c, the current orientation w.r.t. the inertial frame β_c and the ocean velocity magnitude in the North-East frame is:

$$V_x = U_c \cos(\beta_c) \tag{6.25}$$
$$V_y = U_c \sin(\beta_c) \tag{6.26}$$

In the body frame, the heading of the vessel ψ must be taken into account giving the equations for the ocean velocity magnitude in the body frame:

$$u_c = U_c \cos(\beta_c - \psi) \tag{6.27}$$
$$v_c = U_c \sin(\beta_c - \psi) \tag{6.28}$$

6.4 Guidance

When the target's position is known, as well as the vessel's kinematics and the ocean current, it is time to use that knowledge to control the vessel to minimize the cross-track and along-track errors. This part of the guidance system includes an adaptive observer, indirect adaptive integral LOS (line-of-sight) guidance and a surge controller to minimize the along track error. Only the main points will be considered in this section, a more detailed explanation is given in Lekkas and Fossen (2014).

6.4.1 Indirect adaptive integral LOS guidance

The LOS equation for calculating the desired heading ψ_d is:

$$\psi_d = \gamma_p - \beta_r + \arctan\left(-\frac{1}{\Delta}(y_e + \alpha_y) \right) \tag{6.29}$$

where $\beta_r = \arctan 2(v_r, u_r)$, $\Delta > 0$ is the lookahead distance and α_y is the control input. The desired yaw rate r_d can be computed using ψ_d together with the previous desired heading ψ_{dp}:

$$r_d = \frac{(\psi_d - \psi_{dp})}{h} \tag{6.30}$$

with h as the systems time step.

6.4.2 Adaptive observer

The adaptive observer is used to calculate the control input α_y. The observer estimates the rate of change in the cross-track and along-track errors as well as the rate of change of the ocean currents effects. The observer updates for each loop $x_{ob} = x_{ob} + h\dot{x}_{ob}$ using the equations given in Lekkas and Fossen (2014):

$$\dot{\hat{y}}_e = -\frac{U_r(\hat{y}_e + \alpha_y)}{\sqrt{\Delta^2 + (y_e + \alpha_y)^2}} + \hat{\theta}_y + k_1(y_e - \hat{y}_e) \tag{6.31}$$
$$\dot{\hat{\theta}}_y = k_2(y_e - \hat{y}_e) \tag{6.32}$$
$$\dot{\hat{x}}_e = -\hat{x}_e + \hat{\theta}_e + \alpha_x + k_3(x_e - \hat{x}_e) \tag{6.33}$$
$$\dot{\hat{\theta}}_x = k_4(x_e - \hat{x}_e) \tag{6.34}$$

where $\alpha_x = -\hat{\theta}_x$, and $k_1 > 0, k_2 > 0, k_3 > 0$ and $k_4 > 0$ are chosen gain values. The ocean current is then estimated as:

$$\hat{\beta}_c = \gamma_p + \arctan 2(\hat{\theta}_y, \hat{\theta}_x) \tag{6.35}$$

$$\hat{U}_c = \sqrt{\hat{\theta}_y^2 + \hat{\theta}_x^2} \tag{6.36}$$

6.4.3 Surge controller

The surge controller is used to minimize the along-track error x_e by finding a suitable relative velocity reference u_{rd}:

$$u_{rd} = \sqrt{1 + \xi_t^2}\left(- v_r \frac{\xi_t}{\sqrt{1 + \xi_t^2}} + U_t + \alpha_x - k_x x_e \right) \tag{6.37}$$

Equation (6.37) shows how u_{rd} is computed where $k_x > 0$ is a chosen gain value and ξ_t is:

$$\xi_t = \frac{v_r \Delta + u_r(y_e + \alpha_y)}{v_r(y_e + \alpha_y) - u_r \Delta} \tag{6.38}$$

Chapter 7

Simulations, Results and Discussion

Following the goals for this thesis, the final element was to investigate the possibility of using the generated paths in Chapter 5 as input to a guidance system such as the one presented in Chapter 6 from Lekkas and Fossen (2014).

This chapter show the results from using that guidance system when given the task of tracking three differently generated paths, namely the Dubins' path, and the Dubins' path with clothoid transitions and Fermat's spiral transitions. These three paths have all been generated as explained in Chapter 5 after receiving the same basic path consisting of straight lines connected by waypoints, which was made by the RRT algorithm from Chapter 3.

Simulations were made for three different scenarios, namely a run with zero ocean current, a run with small ocean current and a run with large ocean current. Then the resulting paths are plotted together with the generated tracking trajectory to see how far the vehicle deviates from the target. The simulations were done using MATLAB 2015a.

7.1 Simulation setup

As in any path planning scenario, the workspace had to be defined. In these simulations, a 1000x1000 meter map was made containing two large square obstacles (yellow colored). The obstacles are placed in a way that forces the path to have many intermediate waypoints and to check if the collision avoidance and the clearance constraints are satisfying. After the map is created, the vehicle is placed at an initial position on the map with an initial heading, and a goal position and heading is chosen. In the simulations, the start was set at $X_{start} = (x_{start}, y_{start}, \psi_{start}) = (100, 100, 0)$ and the goal was at $X_{goal} = (x_{goal}, y_{goal}, \psi_{goal}) = (900, 900, 0)$. Then the minimum turning radius is chosen, $R = 50$ which is also used as input for the clearance constraint increasing the obstacles size for the next step.

With the area, start position and goal position defined, the RRT algorithm tries to find a path connecting the two points without entering the extended obstacle area (plotted as the white area surrounding the obstacles in Figure 7.1). The initial path is plotted as the solid blue line in Figure 7.1. Using a simple waypoint reducing algorithm, the reduced path is shown as the dotted red line. The intermediate waypoints in this reduced path is then used as input for the path generator.

Once a path has been generated with the chosen smoothing, it is saved as a path table and used as input for the guidance. In the guidance system, the table is used to define the initial conditions for each segment the target will traverse, and also decide when to switch segment. The target speed is set at $U_t = 5m/s$. The current is also set and for the three scenarios it is set at 0, 1 and 3 m/s with $\beta_c = -40deg$. The initial position of the vehicle is set at X_{start} and the lookahead distance for the LOS algorithm is $\Delta = 50m$. The gain values are $k_x = 0.5$, $k_1 = 10$, $k_2 = 0.8$, $k_3 = 10$ and $k_4 = 1$. It is assumed that no prior estimation or measurement of the ocean current is available. This means the observer's initial conditions are $(\hat{x}_e, \hat{\theta}_x, \hat{y}_e, \hat{\theta}_y) = (0, 0, 0, 0)$. The simulation is run until the goal point has been reached by the actual vehicle.

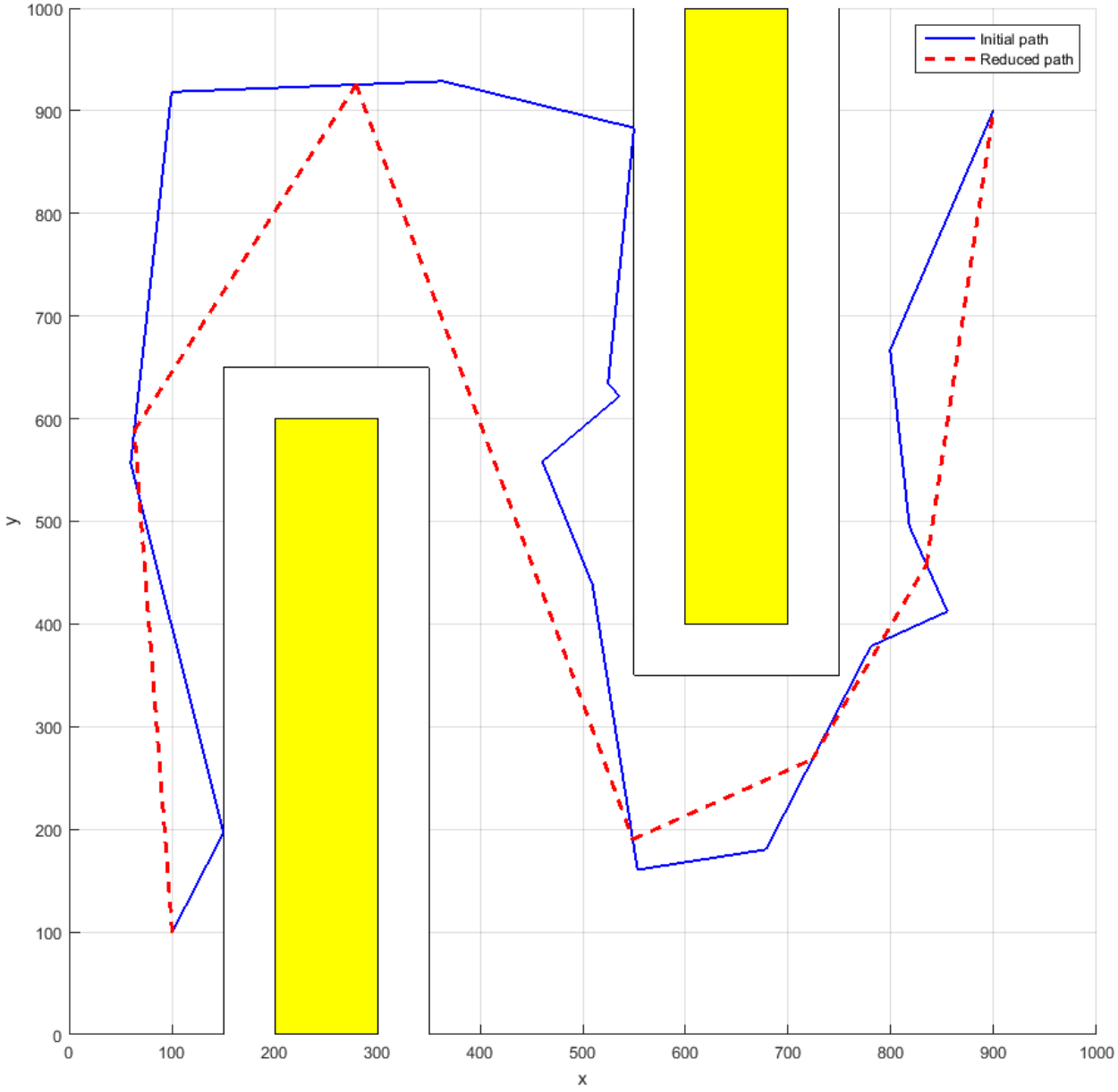


Figure 7.1: RRT-generated basis from the vessel's start position to the end position

7.2 Results with zero current

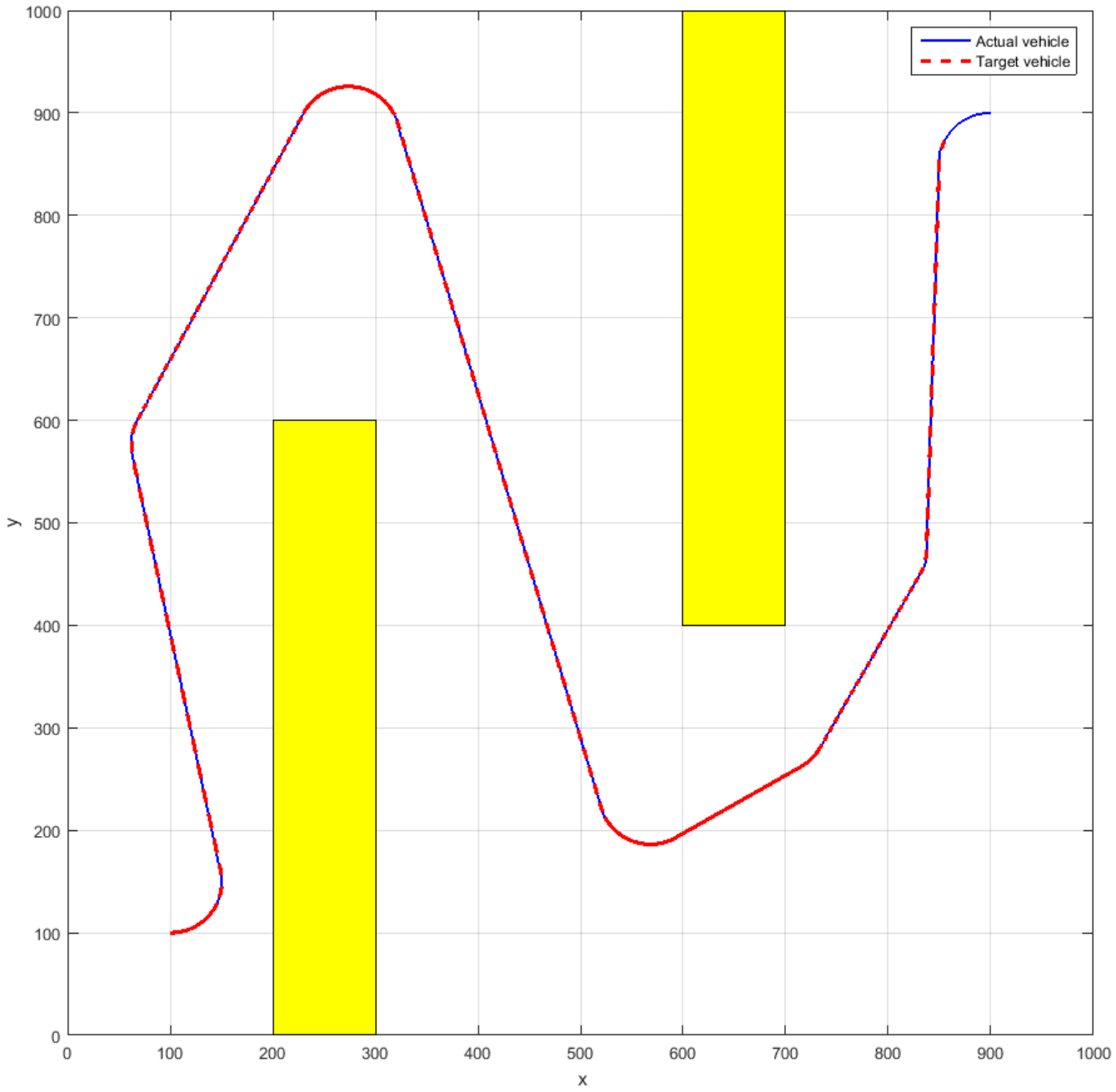


Figure 7.2: Tracking a Dubin's path with zero current

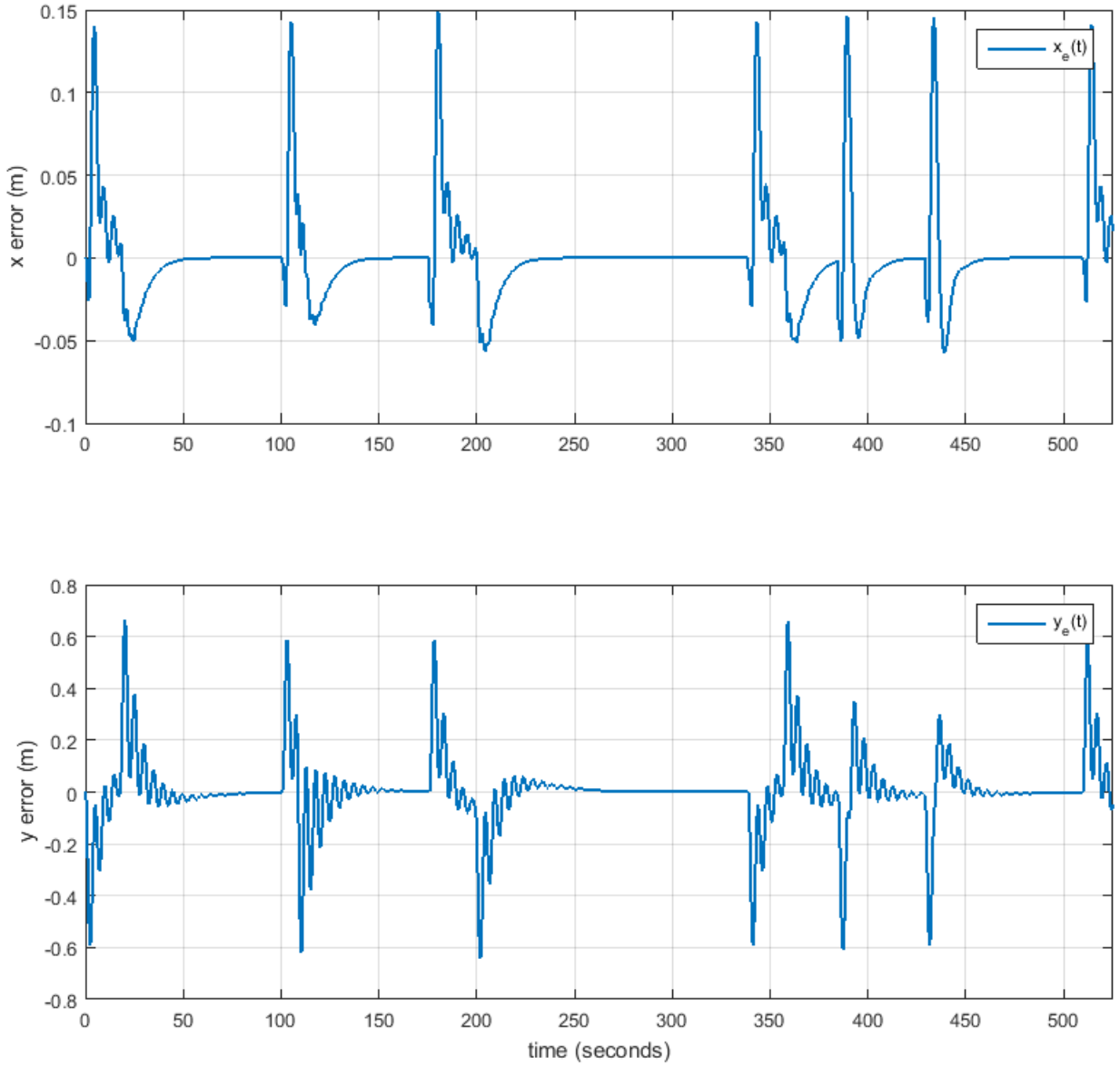


Figure 7.3: Plot of the along-track and cross-track errors when tracking a Dubin's path with zero current

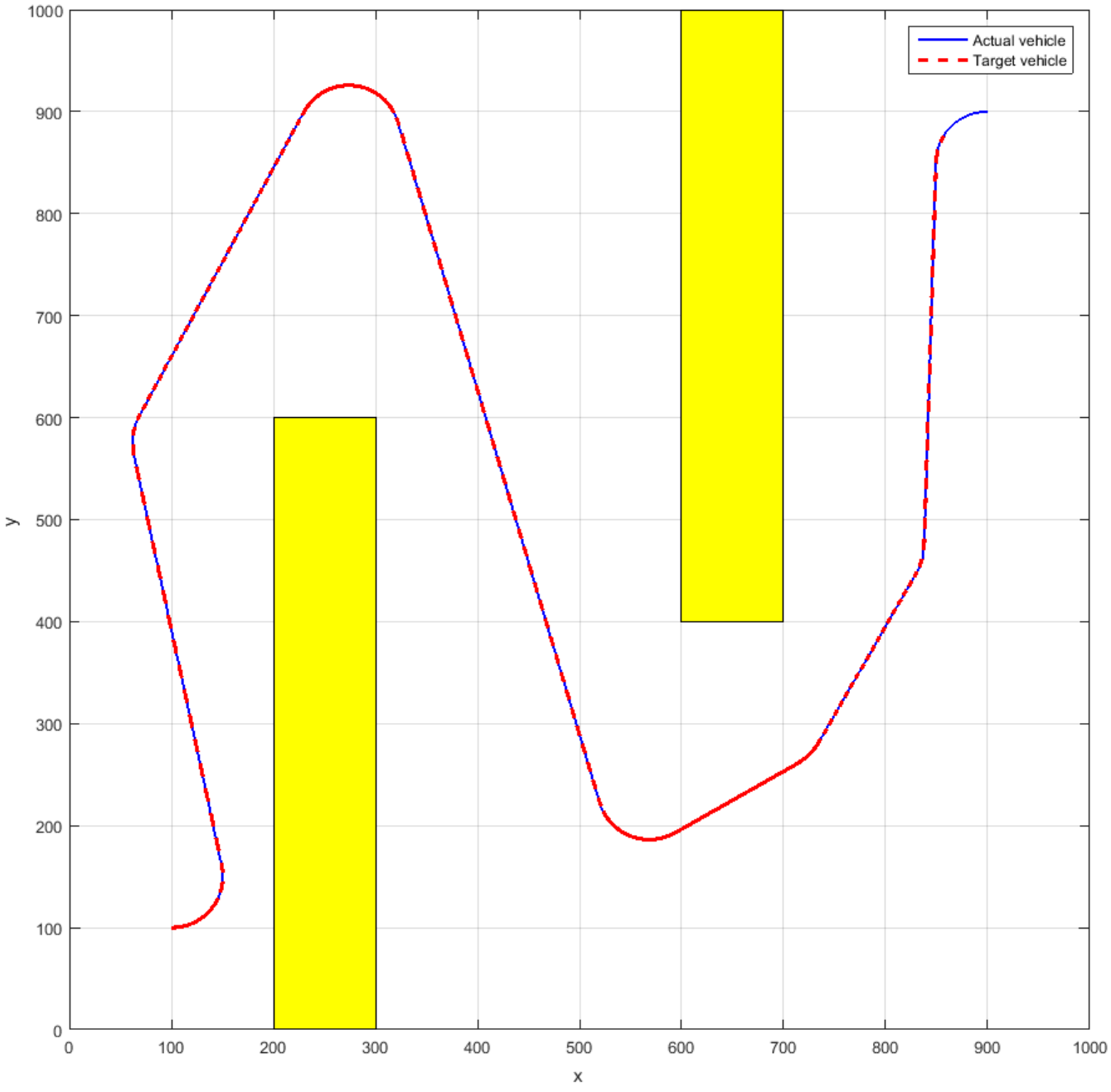


Figure 7.4: Tracking a Dubin's path with clothoid transition with zero current

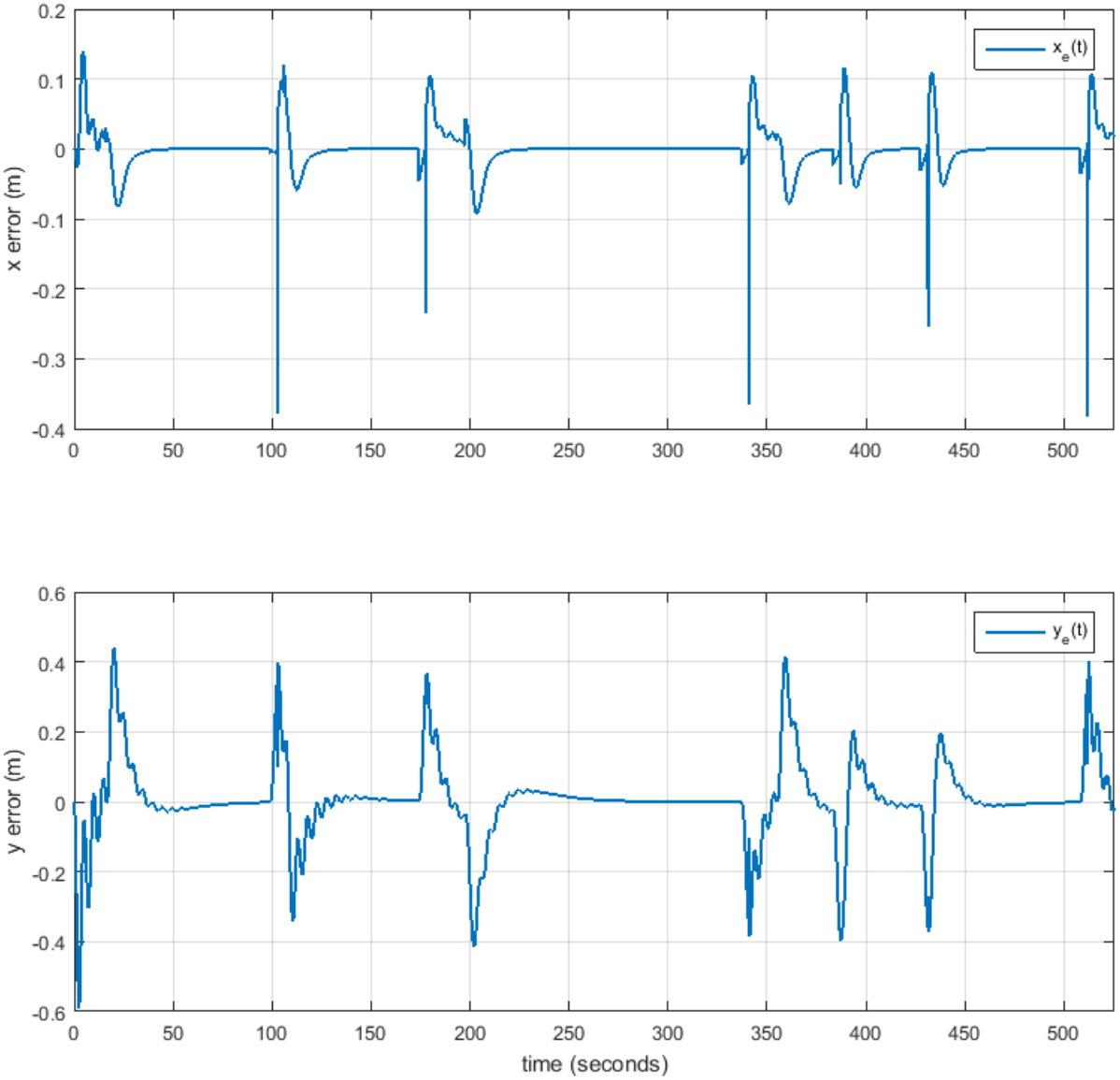


Figure 7.5: Plot of the along-track and cross-track errors when tracking a Dubin's path with clothoid transition with zero current

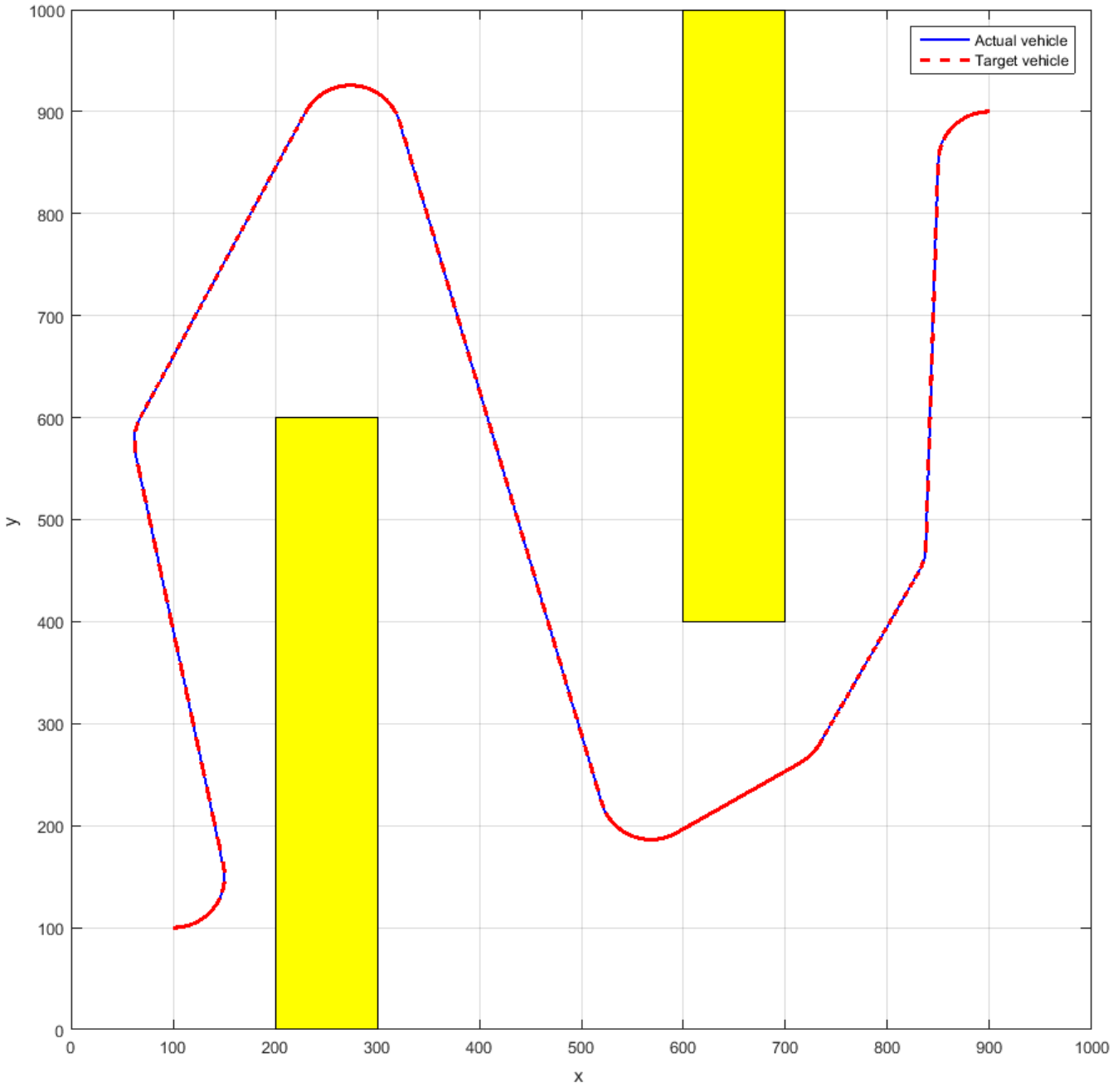


Figure 7.6: Tracking a Dubin's path with Fermat's spiral transition with zero current

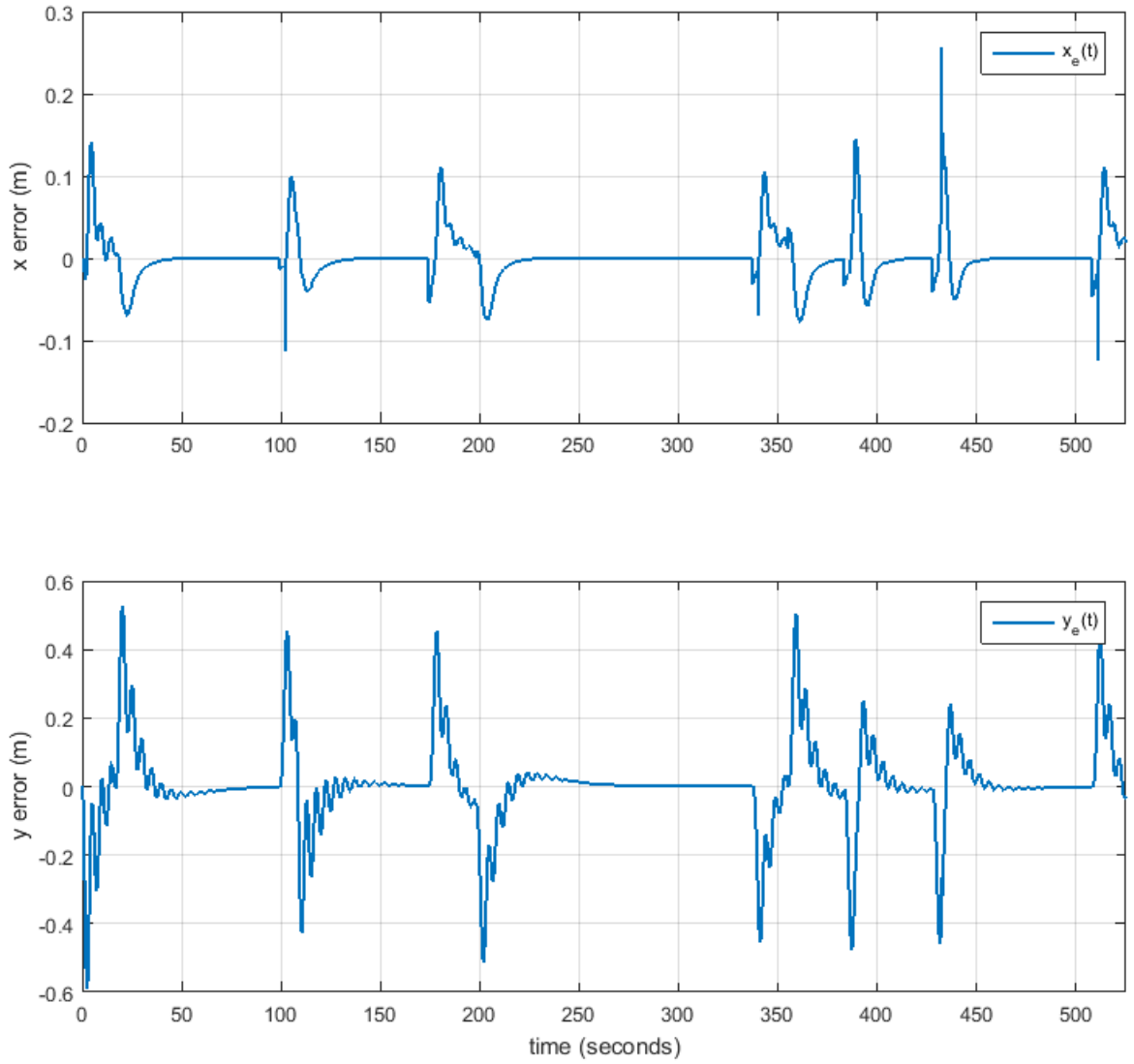


Figure 7.7: Plot of the along-track and cross-track errors when tracking a Dubin's path with Fermat's spiral transition with zero current

7.3 Results with small current

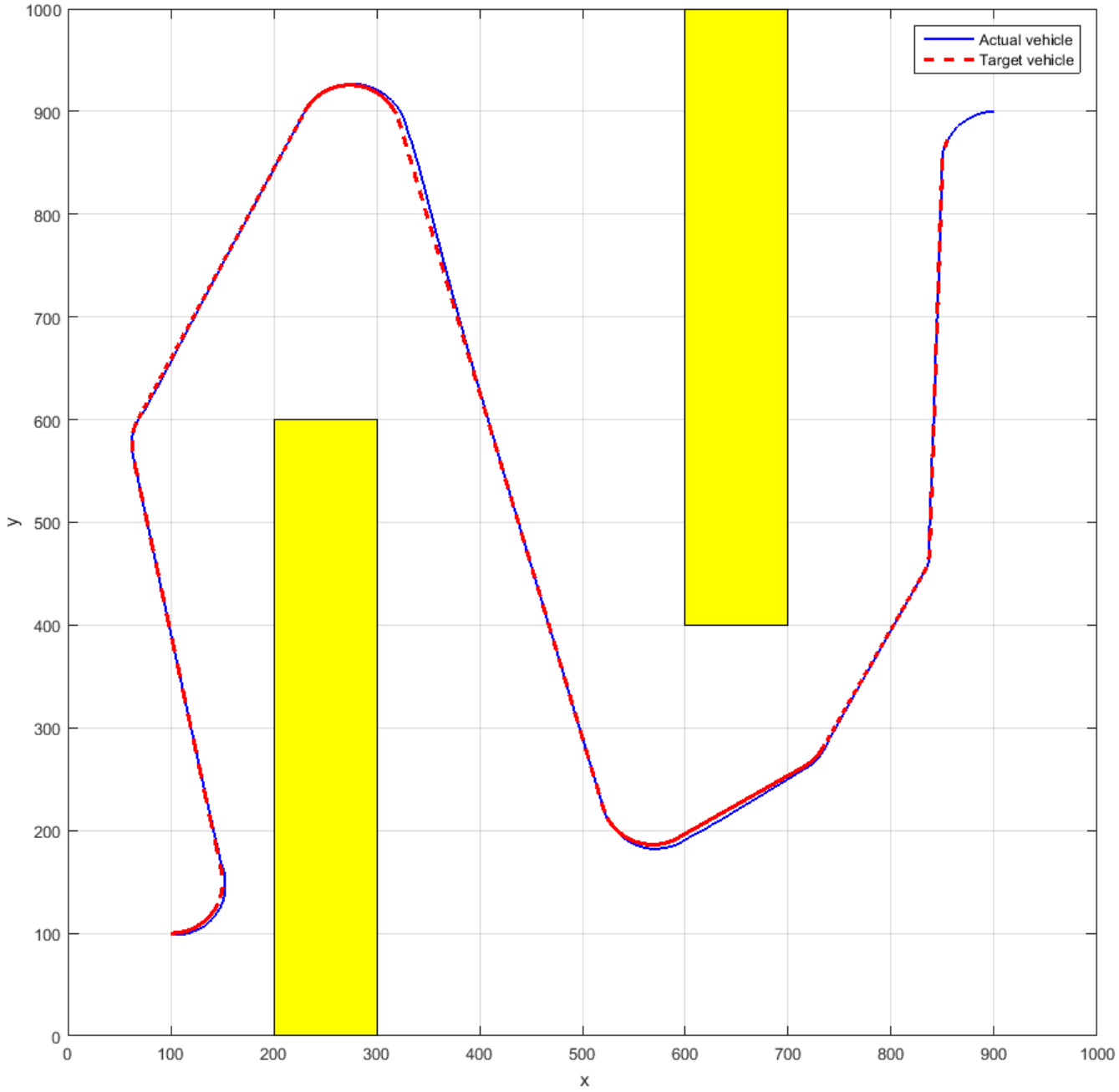


Figure 7.8: Tracking a Dubin's path with current $U_c = 1m/s$

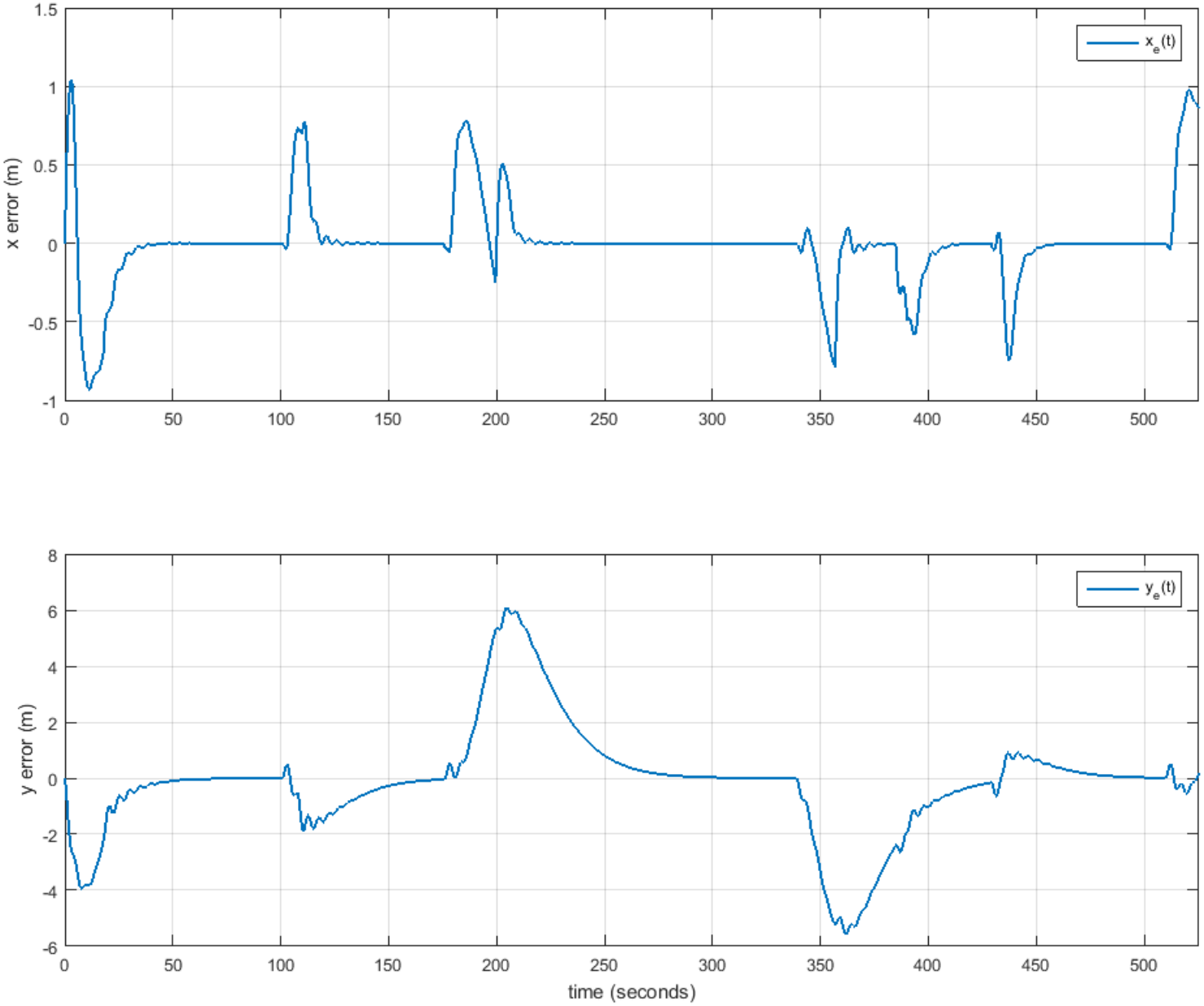


Figure 7.9: Plot of the along-track and cross-track errors when tracking a Dubin's path with current $U_c = 1m/s$

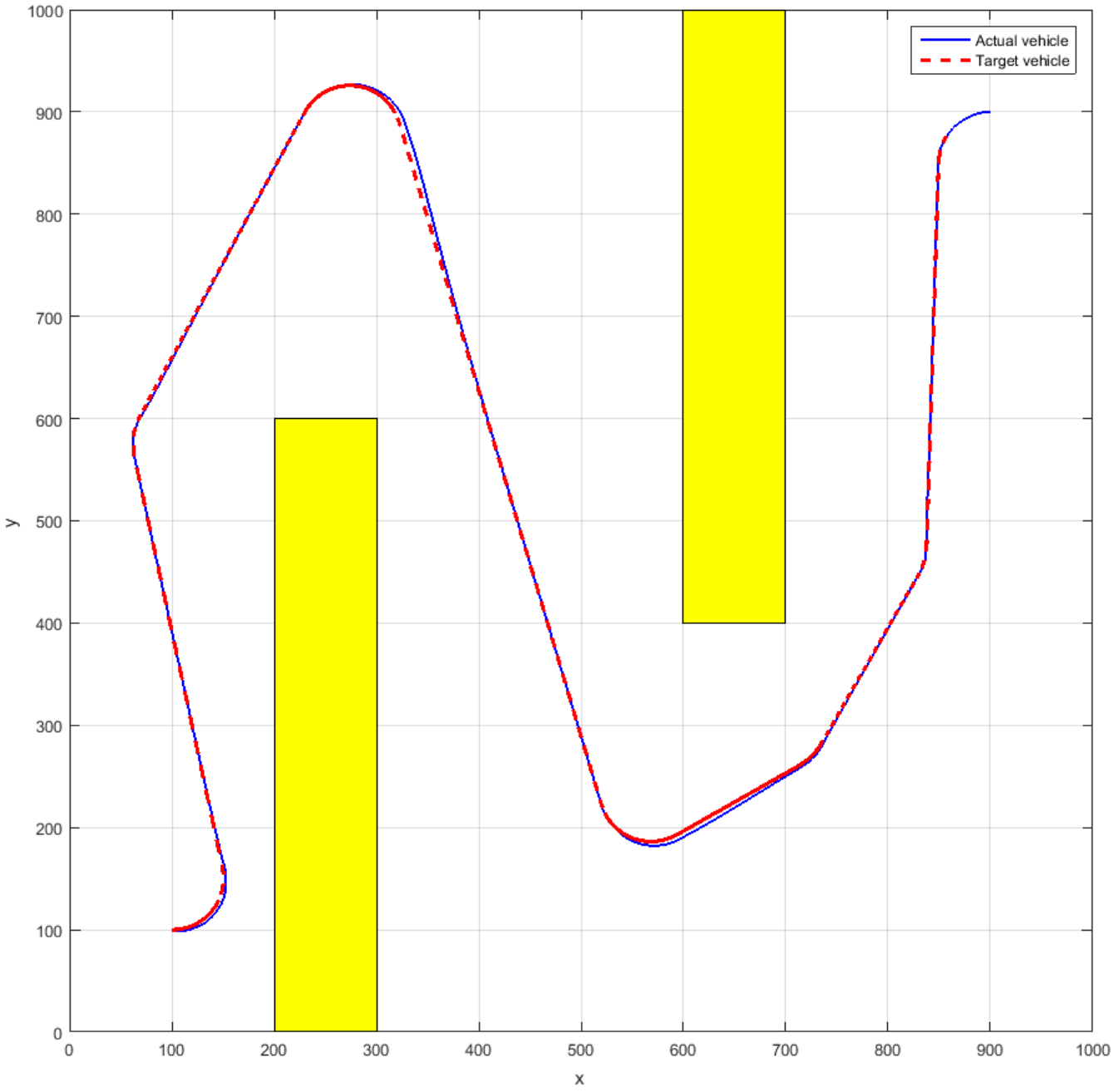


Figure 7.10: Tracking a Dubin's path with clothoid transition with current $U_c = 1m/s$

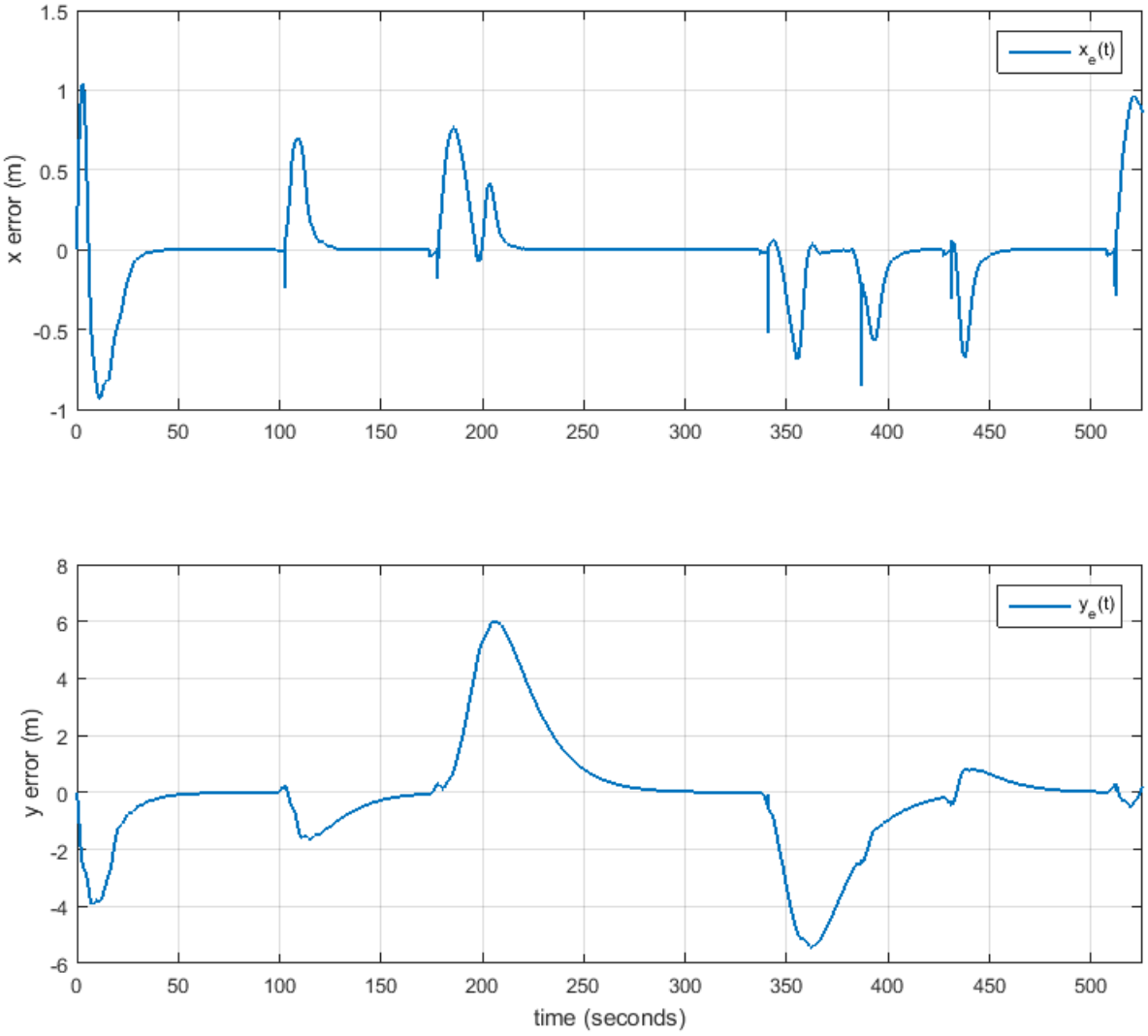


Figure 7.11: Plot of the along-track and cross-track errors when tracking a Dubin's path with clothoid transition with current $U_c = 1m/s$

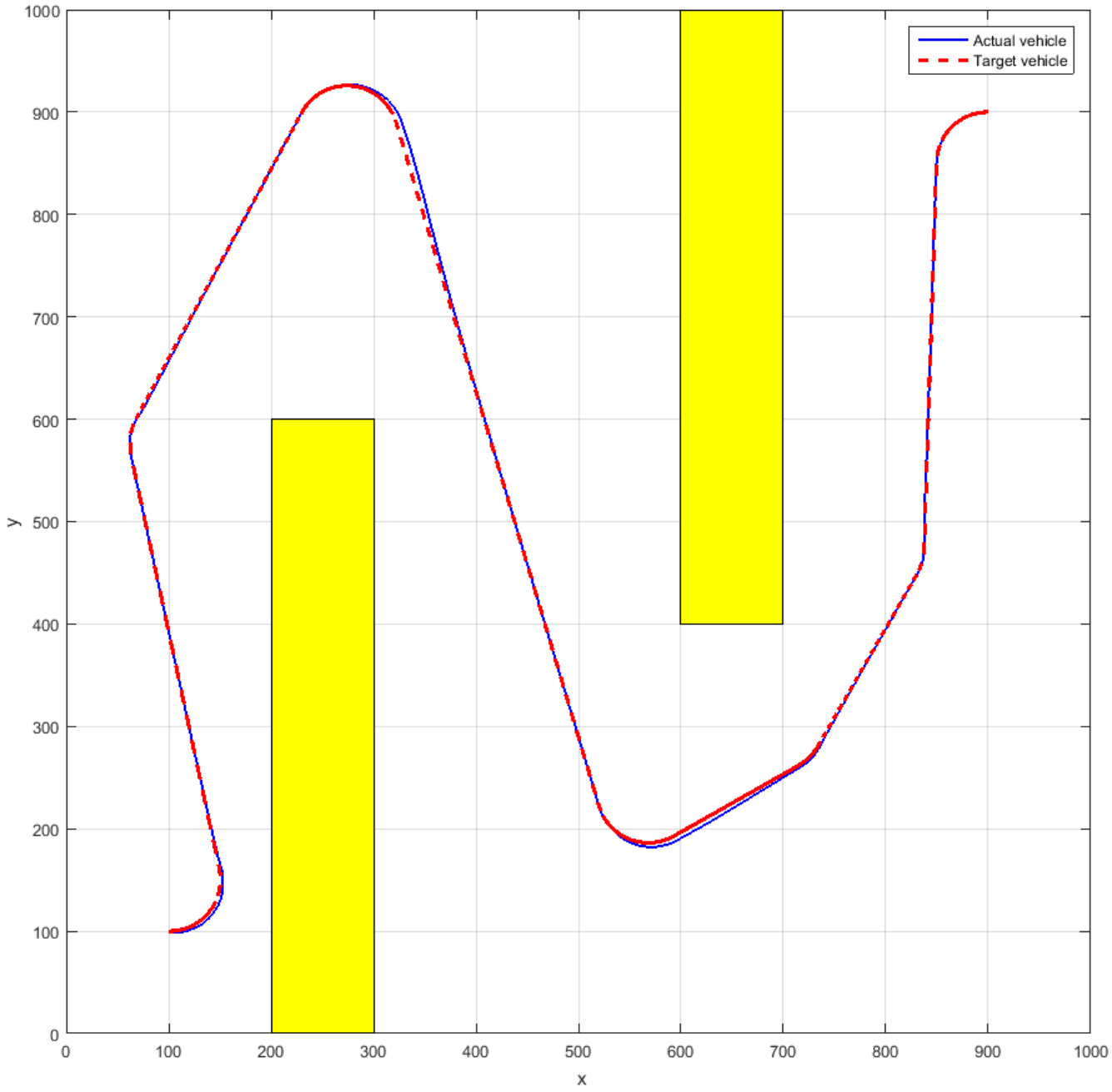


Figure 7.12: Tracking a Dubin's path with Fermat's spiral transition with current $U_c = 1m/s$

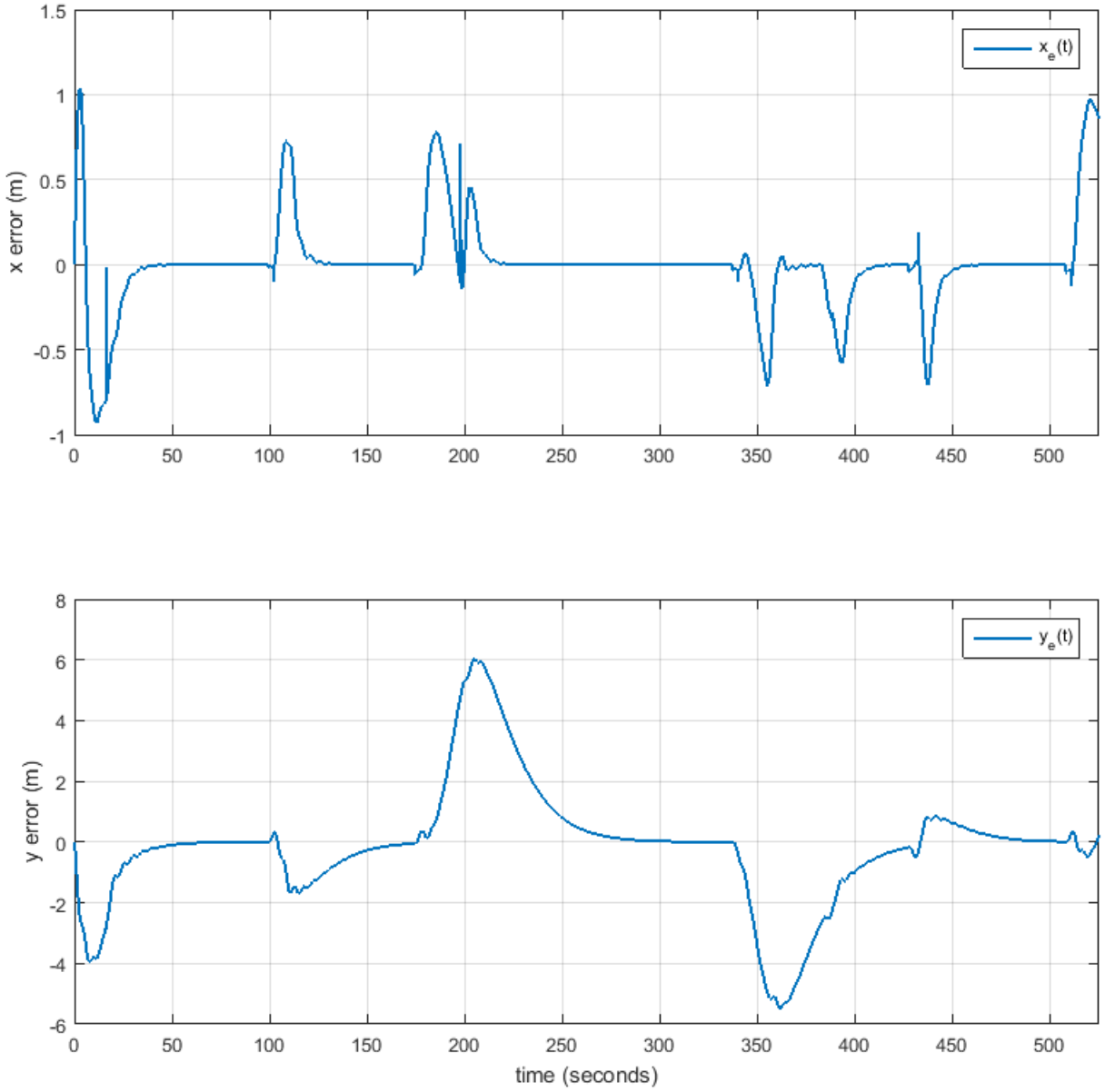


Figure 7.13: Plot of the along-track and cross-track errors when tracking a Dubin's path with Fermat's spiral transition with current $U_c = 1m/s$

7.4 Results with large current

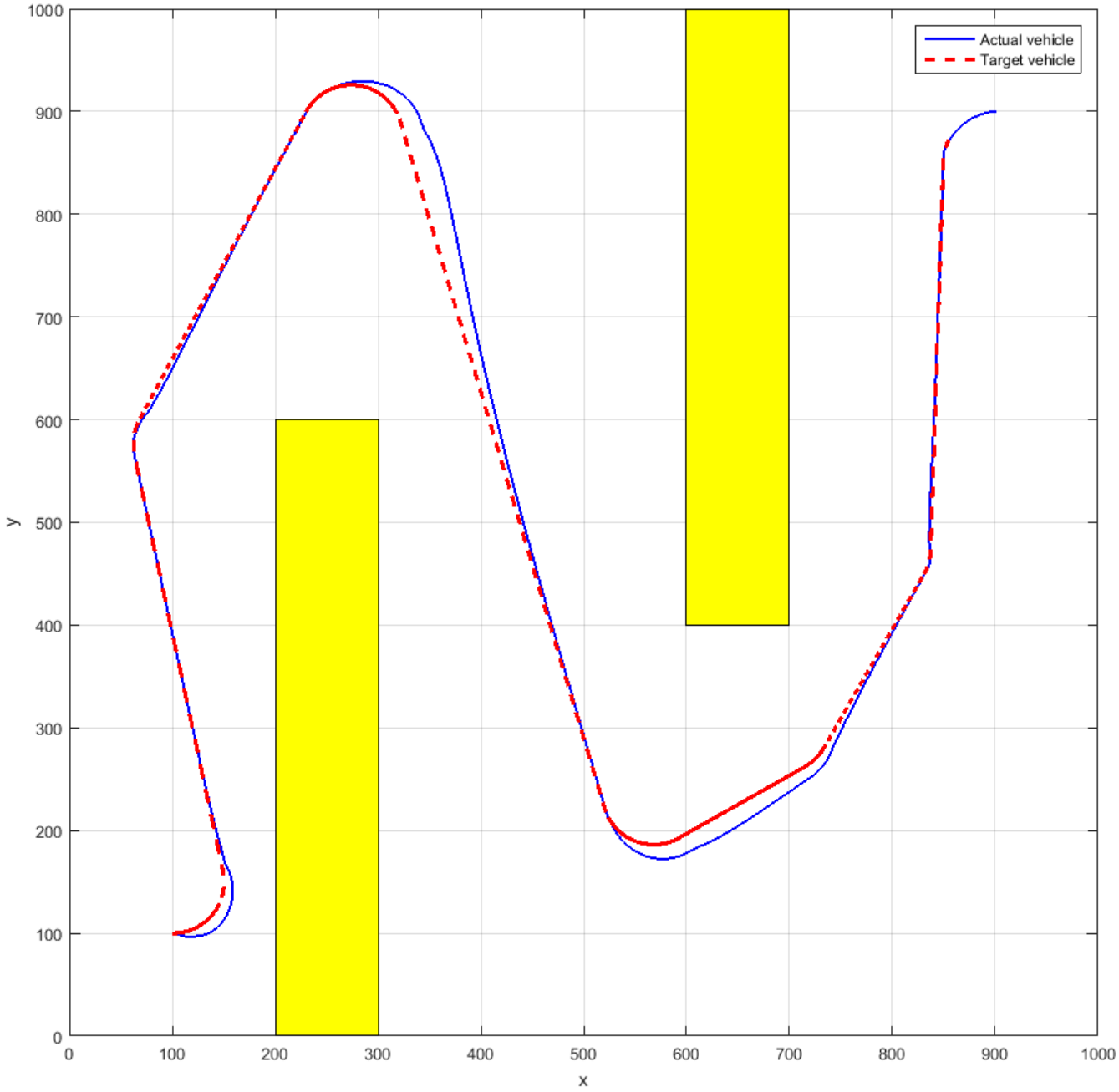


Figure 7.14: Tracking a Dubin's path with current $U_c = 3m/s$

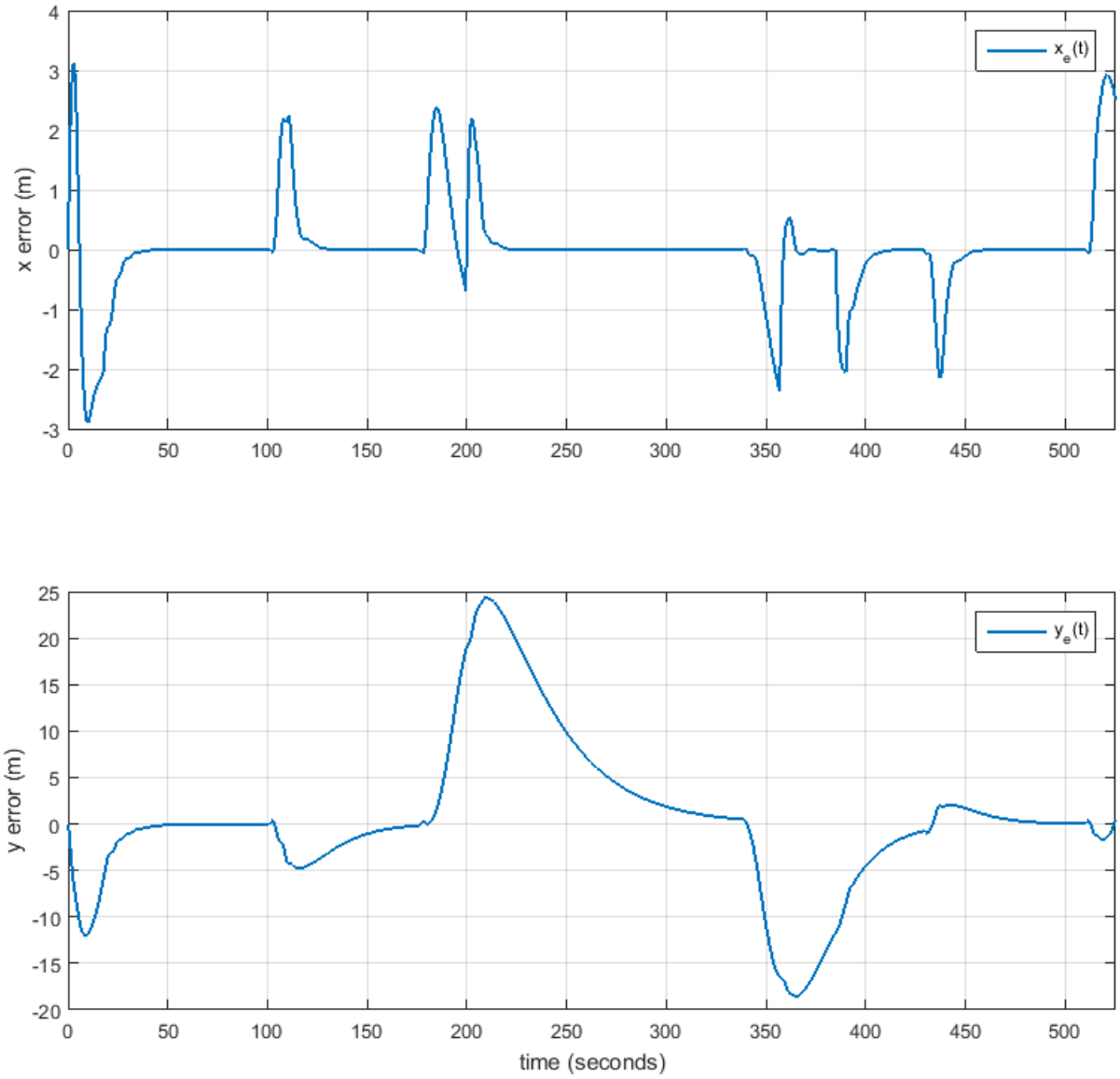


Figure 7.15: Plot of the along-track and cross-track errors when tracking a Dubin's path with current $U_c = 3m/s$

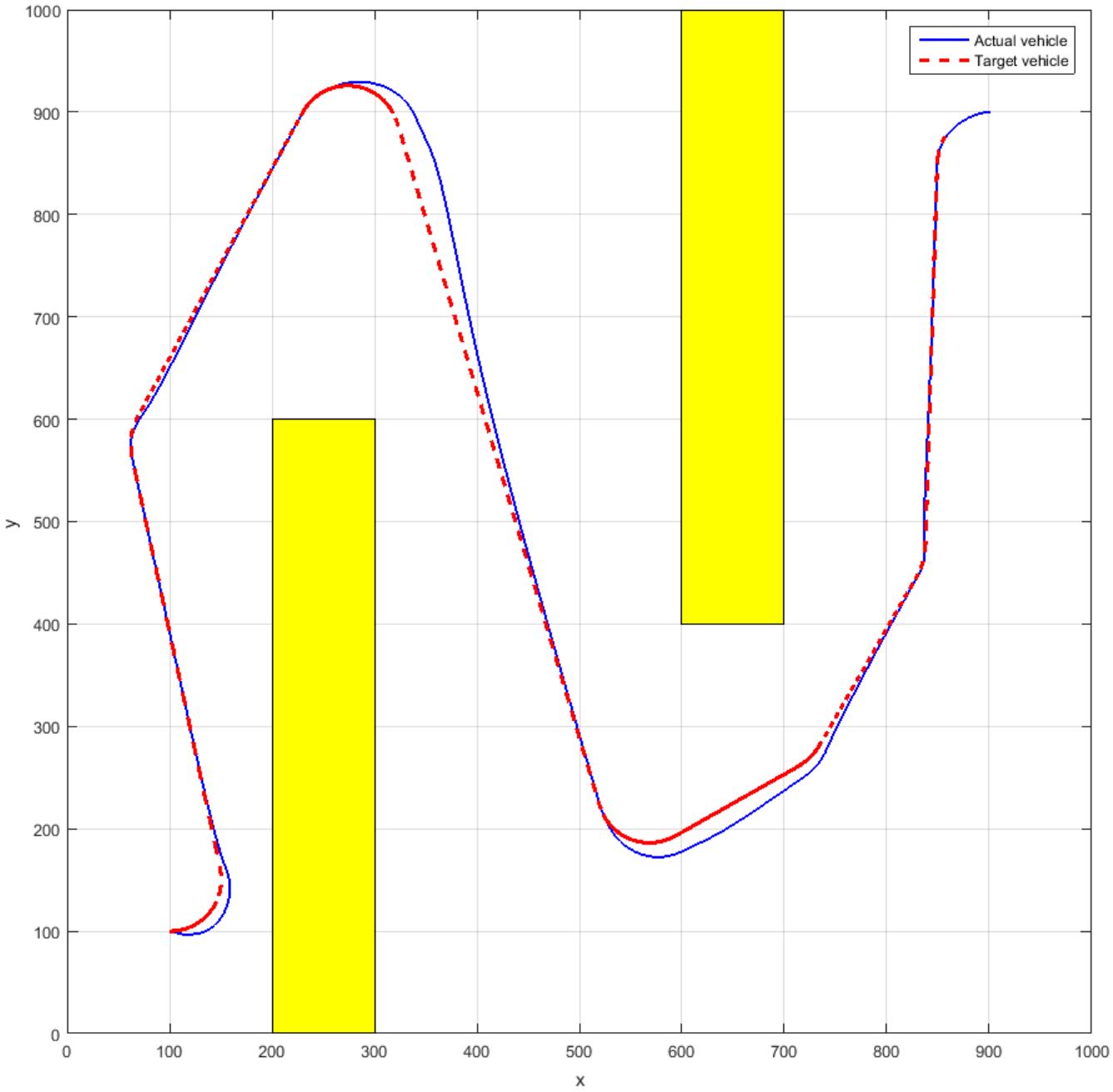


Figure 7.16: Tracking a Dubin's path with clothoid transition with current $U_c = 3m/s$

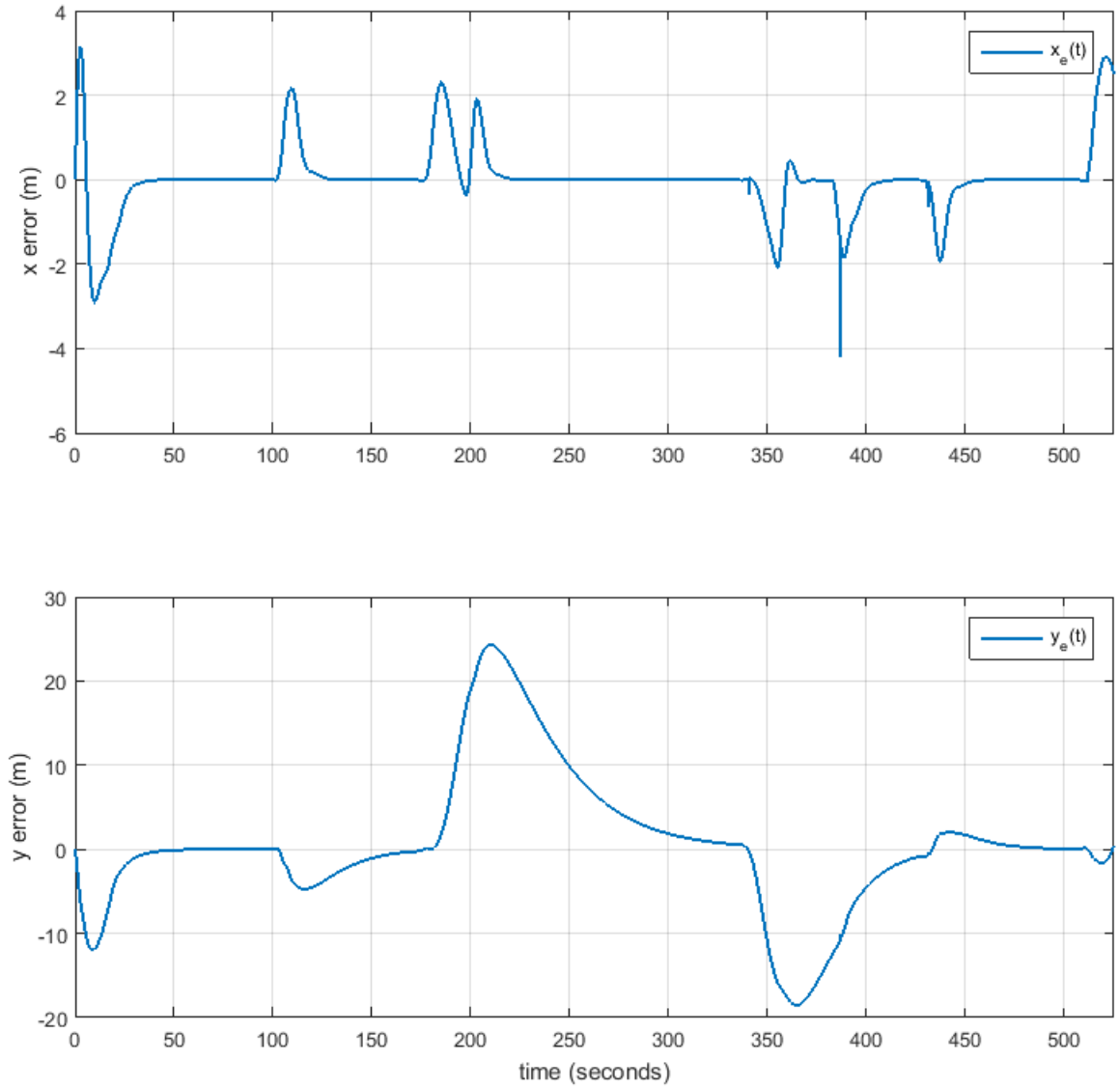


Figure 7.17: Plot of the along-track and cross-track errors when tracking a Dubin's path with clothoid transition with current $U_c = 3m/s$

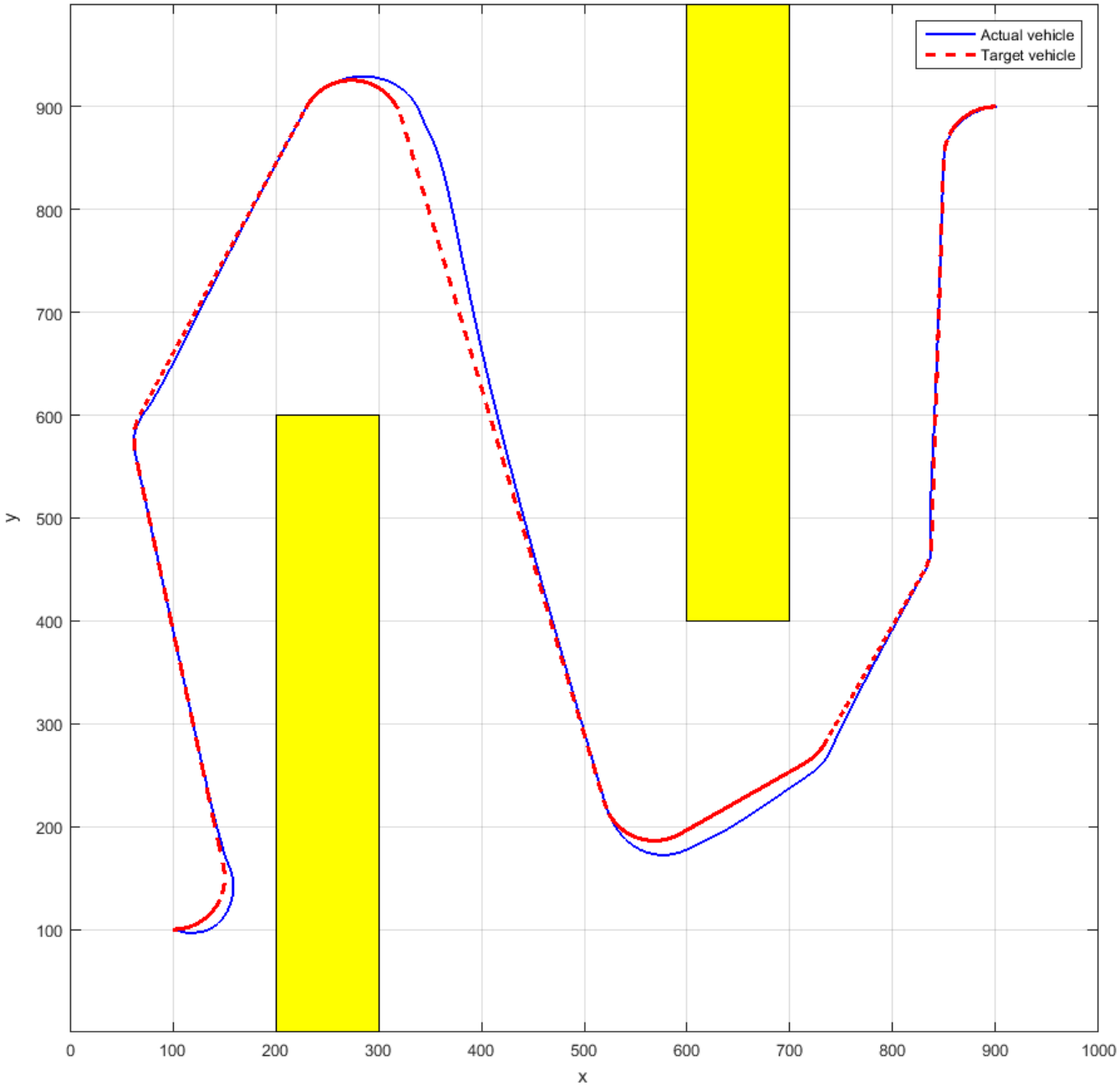


Figure 7.18: Tracking a Dubin's path with Fermat's spiral transition with current $U_c = 3m/s$

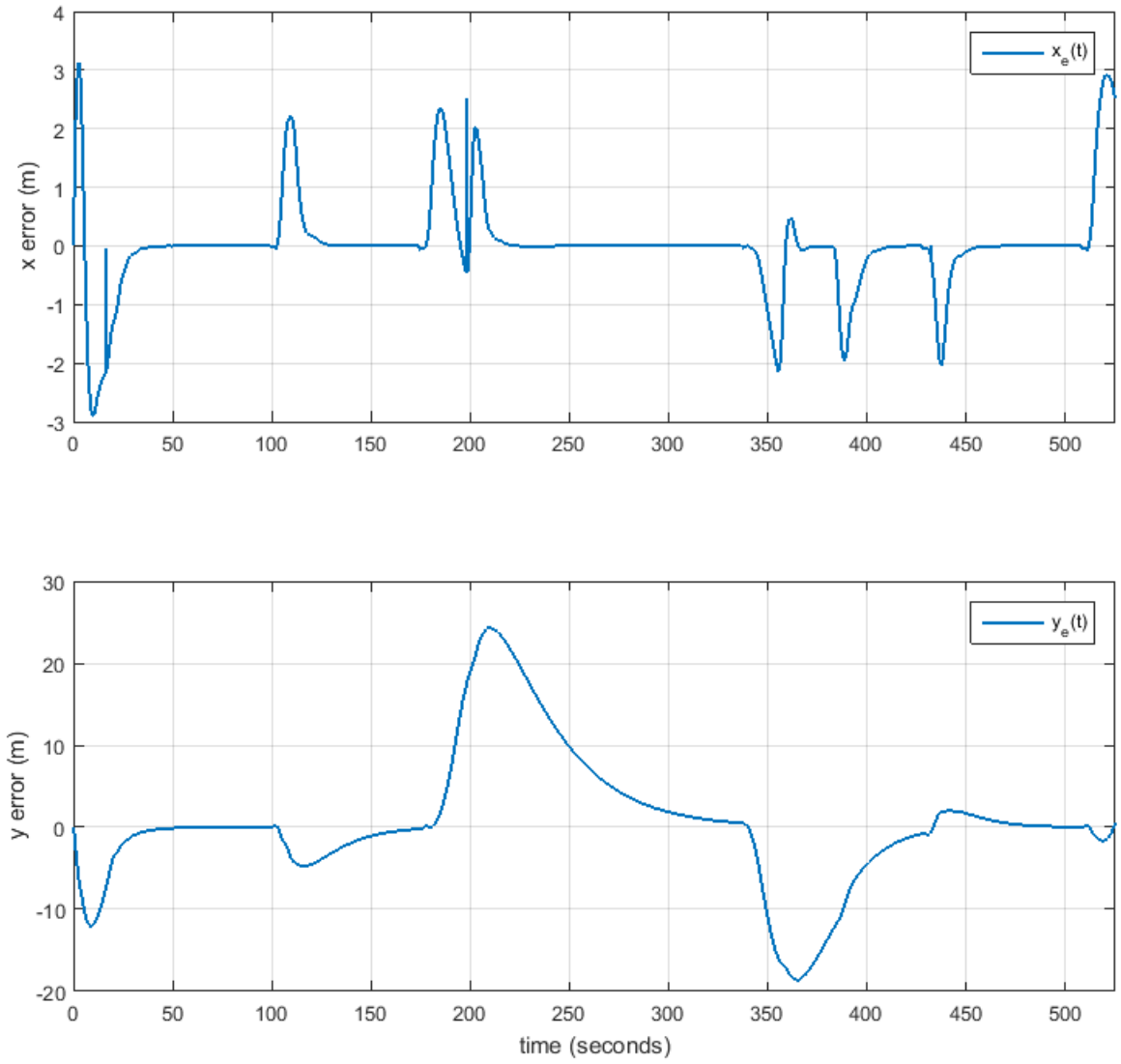


Figure 7.19: Plot of the along-track and cross-track errors when tracking a Dubin's path with Fermat's spiral transition with current $U_c = 3m/s$

7.5 Discussion

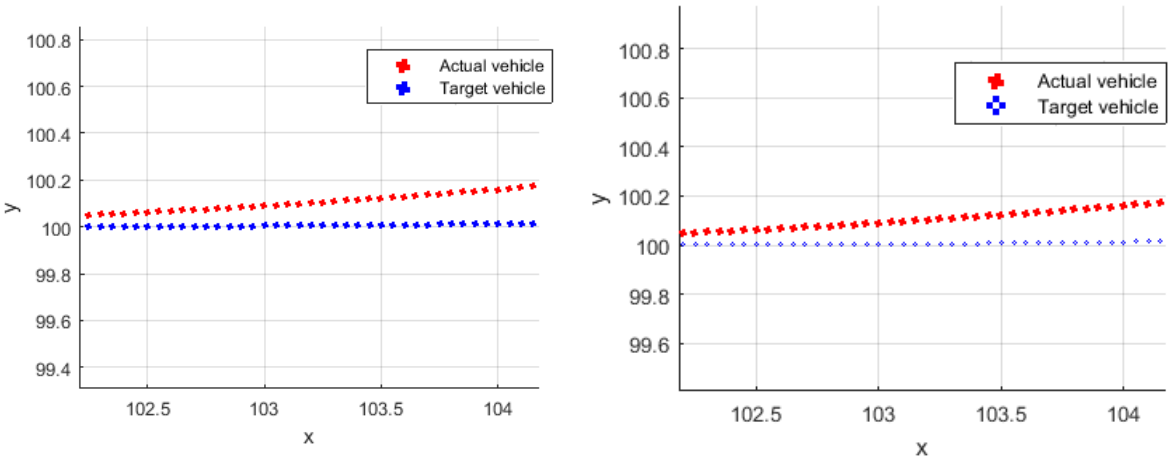
‘ When simulating with zero current, the tracking of the three different paths give very good results. In figures 7.2, 7.4 and 7.6, the cross-track and along-track errors are very small (less than 1 meter) as shown in the plots from figures 7.3, 7.5 and 7.7. Even though the errors are very small and always converge after the perturbation caused by turns, they still have some oscillatory behavior which is mostly due to insufficient tuning. It is not easy to see any difference between the regular Dubin's path and the curvature continuous Dubin's/clothoid path and the Dubin's/Fermat path.

In terms of run time, the Fermat's spiral is still superior over the clothoid as the same time difference mentioned in Section 5.5.2 was also visible in these simulations. To find out how this time difference would influence the system a test where the system requires a new path to be generated underway is proposed for future work. Such a test would also test how important the speed of RRT method is as well.

It is worth mentioning that the paths generated for these simulations were made to always start on a turning circle, which is a very frequent case due to the Dubin's path logic mentioned in Chapter 4 always starting with a turn. This is not recommended as the actual vehicle does not start with the rudder in a turning position, and therefore there will always be an error in the beginning when the vehicle tries to compensate. This phenomenon is more visible when under the influence of ocean currents, but zoomed in it can be seen with zero ocean current as well as shown in Figure 7.20. In the cases with Fermat's spiral and clothoid transition some of the deviation can be avoided if a transition is included at the start. However the initial deviation will always occur as long as the observer starts with no prior knowledge of the system, and there has to estimate the effect of the current.

Tracking the paths while under the influence of a low amount of current ($U_c = 1m/s$) is shown in figures 7.8-7.13 and still the plots show that tracking the three different target paths give very similar results. Looking at the error plots in figures 7.9, 7.11 and 7.13, the controller's along-track error x_e is much less than the cross-track error y_e which is expected as the relative velocity reference u_{rd} was designed to only minimize x_e. The initial deviation shown in Figure 7.20 is more visible in this low current case.

Under the influence of a high amount of current ($U_c = 3m/s$) the cross-track y_e is very large around the turning segments, especially along the longest turns as seen in figures 7.14-7.19. The actual vehicle uses a long time to converge towards the target after such an increase in the cross-track error. One of the main reasons for this large cross-track error is that the adaptive LOS computes the effect of the current in the body frame. When the vehicle changes direction, as it does in the turning segments, the parameters θ_y and θ_x are time varying and therefore it is not possible to track them 100%. This effect is amplified the longer the turn lasts as the results in Figure 7.14, 7.16 and 7.18 clearly show. This can be fixed by computing the current in the NED frame, which means that for each time instant the values θ_y and θ_x can be calculated as



(a) Dubin's/clothoid path

(b) Dubin's/Fermat path

Figure 7.20: The vehicle's initial deviation from the target caused by the first path segment being a circular arc

in the paper by Lekkas and Fossen (2014) instead of being estimated as in Section 6.4.2.

All the results show that the actual vessel stays well away from the obstacles when going from start to finish. This is due to the RRT creating the initial path of straight lines connected by waypoints while avoiding the extended obstacles, and shows the RRT is capable of being used as a basis in path tracking guidance systems. In every case the vessel also reaches the goal position with the desired pose.

Another reason for the vibrations on the along-track and cross-track errors x_e and y_e is that during a turn the vessel loses some of its surge speed which turns into sway. This causes the tracking algorithm to compensate by commanding additional thrust. This additional thrust cause the vessel to speed up during the turn and therefore escape the path because the rudder is not able to adjust fast enough.

The cross-track and along-track errors seem to be slightly larger when tracking the Dubin's path in every case, and with more tuning of the desired rate of change in curvature for the Fermat's spiral and clothoid their resulting cross and along-track errors are expected to be even smaller. For the clothoid this means looking deeper into the choice of sharpness c, and for the Fermat's spiral the chosen point of the curve where the maximum curvature should appear, namely the value θ_{end}.

Chapter 8

Conclusion and Future Work

The main goal for this thesis was to create a path with the Rapidly-exploring Random Trees algorithm. Smooth it with Dubins' car dynamics and clothoid or Fermat's spiral transitions for curvature continuity. And then find out if the generated path could be used as input for path tracking while being under the influence of ocean currents. By implementing such a system for a marine vessel in MATLAB the results from Chapter 7 are promising, but not without some issues. Under high ocean current value, long turns causes the vehicle to deviate from the target and in some cases causes the vehicle to be unable to converge with target until it reaches the next turn. Most of this deviation comes directly from the adaptive controller which does not promise to track a time-varying parameter, but still does it to a certain degree. An ad-hoc solution is suggested in Chapter 7. The results and comparison from Section 5.5.2 show that using the Fermat's spiral to achieve curvature continuity is far superior to the clothoid even though both paths are G^2 continuous, as the Fermat transitions are computed around a 100 times faster. This is mostly due to computing Fresnel integrals when creating a clothoid.

When it comes to collision avoidance, the results from Chapter 7 are also very promising. Adding clearance constraints provided results where the vehicle stayed clear of obstacles. The RRT algorithm is extremely efficient in doing this.

There are many paths one could take when continuing the work presented in this thesis. Tuning the path generating algorithms further as well as the guidance system should provide better results than the ones in Chapter 7. Time became an issue towards the end of the thesis work so not much tuning was done.

To investigate the computational speed of the RRT vs other methods, or the time difference between the Fermat's spiral and the clothoid, a test with dynamical obstacles requiring new paths to be generated on-the-way will provide interesting results.

Another approach could be to extend the controller from Chapter 6 which not only minimizes the along-track but also the cross-track error. It is also possible to replace the estimation of the current by calculating the actual current as mentioned in Section 7.5, hence being able to

compensate for the current more accurately, especially during turning segments.

A natural next step for the RRT-generated curvature continuous paths is to extend them with another dimension, testing them for 3-D applications such as AUVs or UAVs. This is a relatively fresh part of the field which can provide many opportunities to explore new parts of motion planning for autonomous vehicles.

For real-life applications, the paths should be combined with optimization methods. The requirements for creating efficient paths w.r.t. for example fuel consumption or path lengths are very relevant in the world today.

Bibliography

Morten Breivik and Thor I. Fossen. Path following for marine surface vessels. In *OCEANS '04. MTTS/IEEE TECHNO-OCEAN '04*, volume 4, pages 2282–2289, Nov 2004.

Morten Breivik and Thor I. Fossen. *Guidance laws for autonomous underwater vehicles.* Chapter 4, pages 51-76. INTECH Education and Publishing, 2009.

CyberneticZoo. Claude Shannon - Maze-Solving Mouse. `http://cyberneticzoo.com/wp-content/uploads/Shannon-Life-p60-x640.jpg`, 2009. [Online; accessed 16-December-2014].

Andreas R. Dahl. Path planning and guidance for marine surface vessels. Master's thesis, Norwegian University of Science and Technology, 2013.

Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Lester 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*, 79 (3):79:497–516, 1957.

Leonhard Euler. Methodus inveniendi lineas curvas maximi minimive proprietate gaudentes, sive solutio problematis isoperimetrici lattissimo sensu accepti. *The Euler Archive, http://eulerarchive.maa.org/*, 1744.

Pierre de Fermat. Ad locos planos et solidos isagoge. *Varia Opera Mathematica*, 1697.

Thor I. Fossen. *Handbook of marine craft hydrodynamics and motion control.* John Wiley & Sons Ltd., 2011.

Google. Map over Oslofjorden. `https://www.google.no/maps/@59.6746501,10.6045211,5415m/data=!3m1!1e3?hl=en`, 2014. [Online; accessed 17-December-2014].

Joakim Haugen. Guidance algorithms for planar path-based motion control scenarios. Master's thesis, Norwegian University of Science and Technology, 2010.

Young Jin Heo and Wan Kyun Chung. RRT-based path planning with kinematic constraints of AUV in underwater structured environment. In *2013 10th International Conference on*

Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, South-Korea, pages 523–525, Oct 2013.

Lydia E. Kavraki and Steven M. LaValle. Motion planning. *Springer Handbook of Robotics*, pages 109–131, 2008.

Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark J. Overmars. Probabilistic roadmaps for path planning in highdimensional configuration space. *International Transactions on Robotics and Automation*, 12:566–580, 1996.

Jean-Claude Latombe. *Robot Motion Planning: Edition en anglais*. The Springer International Series in Engineering and Computer Science, Boston, MA. Springer, 1991. URL `http://books.google.no/books?id=Mbo_p4-46-cC`.

Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. *Report No. TR 98-11. Computer Science Department, Iowa State University*, 1998.

Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

Steven M. LaValle. Motion planning: The essentials. *Robotics & Automation Magazine, IEEE*, 18(1):79–89, 2011a.

Steven M. LaValle. Motion planning: Wild frontiers. *Robotics & Automation Magazine, IEEE*, 18(2):108–118, 2011b.

Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

Steven M. LaValle and James J. Kuffner Jr. RRT-connect: An efficient approach to single-query path planning. *In proceedings of the 2000 IEEE International Conference on Robotics & Automation, San Fransisco, CA*, 2:995–1001, 2000a.

Steven M. LaValle and James J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects. *In Proceedings of the Workshop on the Algorithmic Foundations of Robotics, Dartmouth College, Hanover, NH, USA*, 2000b.

Anastasios M. Lekkas. *Guidance and Path-Planning Systems for Autonomous Vehicles*. PhD thesis, Norwegian University of Science and Technology, 2014.

Anastasios M. Lekkas and Thor I. Fossen. Trajectory tracking and ocean current estimation for marine underactuated vehicles. In *2014 IEEE Conference on Control Applications (CCA), Antibes, France*, pages 905–910, Oct 2014.

Anastasios M. Lekkas, Andreas R. Dahl, Morten Breivik, and Thor I. Fossen. Continuous-curvature path generation using Fermat's spiral. *Modeling, Identification and Control*, 34(4): 183–198, 2013.

Øivind A. G. Loe. Collision avoidance for unmanned surface vessels. Master's thesis, Norwegian University of Science and Technology, 2008.

MathWorks. MATLAB graphshortest. `http://se.mathworks.com/help/bioinfo/ref/graphshortestpath.html`, 2014a. [Online; accessed 18-December-2014].

MathWorks. MATLAB inpolygon. `http://se.mathworks.com/help/matlab/ref/inpolygon.html`, 2014b. [Online; accessed 16-December-2014].

MathWorks. MATLAB polyxpoly. `http://se.mathworks.com/help/map/ref/polyxpoly.html`, 2014c. [Online; accessed 18-December-2014].

MathWorks. MATLAB voronoi. `http://se.mathworks.com/help/matlab/ref/voronoi.html`, 2014d. [Online; accessed 18-December-2014].

Romain Pepy, Alain Lambert, and Hugues Mounier. Path planning using a dynamic vehicle model. In *Information and Communication Technologies, 2006. ICTTA '06. 2nd*, volume 1, pages 781–786, 2006.

Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. The open motion planning library. *Robotics Automation Magazine, IEEE*, 19(4):72–82, Dec 2012.

Antonios Tsourdos, Brian White, and Madhavan Shanmugavel. *Cooperative path planning of unmanned aerial vehicles*. John Wiley & Sons, 2010.

Georgy Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908. URL `http://eudml.org/doc/149291`.

Eric W. Weissten. "Archemdian sprial." from mathworld-a wolfram web resource. `http://mathworld.wolfram.com/ArchimedeanSpiral.html`, 2013a. [Online; accessed 3-July-2015].

Eric W. Weissten. "Fermat's spiral." from mathworld-a wolfram web resource. `http://mathworld.wolfram.com/ArchimedeanSpiral.html`, 2013b. [Online; accessed 3-July-2015].

Wikipedia. Euler spiral. `https://en.wikipedia.org/wiki/Euler_spiral`, 2015. [Online; accessed 6-July-2015].

Wu Xinggang, Guo Cong, and Li Yibo. Variable probability based bidirectional RRT algorithm for UAV path planning. In *The 26th Chinese Control and Decision Conference (2014 CCDC)*, pages 2217–2222, May 2014.