Motion Planning for a Reversing Full-Scale Truck and Trailer System

Olov Holmer
Master of Science Thesis in

**Motion Planning for a Reversing Full-Scale Truck and Trailer System**

Olov Holmer

LiTH-ISY-EX–16/4983–SE

Supervisor:  **Oskar Ljungqvist**  
ISY, Linköpings universitet  
**Henrik Pettersson**  
Scania CV AB

Examiner:  **Daniel Axehill**  
ISY, Linköpings universitet

*Automatic control*

Department of Electrical Engineering  
Linköping University  
SE-581 83 Linköping, Sweden

Copyright © 2016 Olov Holmer
Abstract

In this thesis improvements, implementation and evaluation have been done on a motion planning algorithm for a full-sized reversing truck and trailer system. The motion planner is based on a motion planning algorithm called Closed-Loop Rapidly-exploring Random Tree (CL-RRT).

An important property for a certain class of systems, stating that by selecting the input signals in a certain way the same result as reversing the time can be archived, is also presented. For motion planning this means that the problem of reversing from position A to position B can also be solved by driving forward from B to A and then reverse the solution. The use of this result in the motion planner has been evaluated and has shown to be very useful.

The main improvements made on the CL-RRT algorithm are a faster collision detection method, a more efficient way to draw samples and a more correct heuristic cost-to-go function. A post optimizing or smoothing method that brings the system to the exact desired configuration, based on numerical optimal control, has also been developed and implemented with successful results.

The motion planner has been implemented and evaluated on a full-scale truck with a dolly steered trailer prepared for autonomous operation with promising results.
Acknowledgments

First I would like to thank my examiner Daniel Axehill and my supervisors Oskar Ljungqvist and Henrik Pettersson for their help and advises.

I would also like to thank Niclas Evestedt and Oskar Ljungqvist for their work with the CL-RRT algorithm which has made this thesis work possible.

Finally, I would like to thank my family and friends for their support and encouragement.

Södertälje, June 2016
Olov Holmer
## Contents

**Notation** ix

1 **Introduction** 1
   1.1 Background 1
   1.2 Problem formulation 2
   1.3 Related work 2
      1.3.1 Modeling 2
      1.3.2 Motion planning 2
      1.3.3 Collision detection 3
   1.4 Vehicle 3
   1.5 System architecture 4
   1.6 Outline 5

2 **Truck and Trailer System** 7
   2.1 General n-Trailer 7
   2.2 General 2-Trailer 8
   2.3 Time reversibility 9
   2.4 Multiple axle trailer 10

3 **Motion Planner** 11
   3.1 The motion planning problem 11
      3.1.1 Constraints 12
      3.1.2 Constraints on the final configuration 12
      3.1.3 Objective function 12
   3.2 Closed-loop system 13
   3.3 Closed-loop rapidly-exploring random tree algorithm 14
   3.4 Uniform sampling of complex areas 14
      3.4.1 Outward offset polygon 15
      3.4.2 Approximating a polygon with a grid of squares 15
   3.5 Heuristics 16
      3.5.1 Jerk optimal velocity profile 17
   3.6 Collision detection 20
## Contents

4 Path Optimization 23
  4.1 Reaching the desired goal configuration 23
    4.1.1 The optimal control problem 25
  4.2 Direct Multiple Shooting 26
  4.3 Local optimization 27

5 Results 29
  5.1 Collision detection 29
  5.2 Heuristics 31
  5.3 Optimization 32
  5.4 Reversibility 33
  5.5 Motion planning with a full-scale truck and trailer system 36

6 Conclusions 41
  6.1 Conclusions 41
  6.2 Future work 42

Bibliography 43
Notation

Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_1$</td>
<td>Length of the truck</td>
</tr>
<tr>
<td>$M_1$</td>
<td>Length of the off-hitch</td>
</tr>
<tr>
<td>$(x_1, y_1)$</td>
<td>Center of the rear axle of the truck in global coordinates</td>
</tr>
<tr>
<td>$L_2$</td>
<td>Length of the dolly</td>
</tr>
<tr>
<td>$\beta_2$</td>
<td>Relative angle between the truck and the dolly</td>
</tr>
<tr>
<td>$L_3$</td>
<td>Length of the trailer</td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>Relative angle between the dolly and the trailer</td>
</tr>
<tr>
<td>$\theta_3$</td>
<td>Global orientation of the trailer</td>
</tr>
<tr>
<td>$(x_3, y_3)$</td>
<td>Center of the rear axle of the trailer in global coordinates</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Steering angle of the truck</td>
</tr>
<tr>
<td>$v$</td>
<td>Longitudinal velocity of the rear axle of the truck</td>
</tr>
<tr>
<td>$f_2$</td>
<td>Factor in the dynamic equations for a general 2-trailer that is independent of the longitudinal velocity of the rear axle of the truck</td>
</tr>
</tbody>
</table>

Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RRT</td>
<td>Rapidly-Exploring Random Tree</td>
</tr>
<tr>
<td>RRT*</td>
<td>Rapidly-Exploring Random Tree Star</td>
</tr>
<tr>
<td>CL-RRT</td>
<td>Closed-Loop Rapidly-exploring Random Tree</td>
</tr>
</tbody>
</table>
1 Introduction

This thesis is part of a project where an autonomous full-scale truck will be developed. The work was started in [15] where promising results were shown in simulations and lab-tests. However, some work is still needed for a successful implementation on a full-scale truck and trailer system. This work mainly consists of improving the motion planning algorithm, estimating the states of the truck and trailer and implementation of a path following algorithm. This thesis work will mainly deal with getting rid of the bottle necks in the CL-RRT algorithm [15]. In parallel another thesis work is running where an observer based controller is designed to follow planned trajectories.

1.1 Background

During the last decade a lot of research has been done in the area of autonomous vehicles. While there probably still is some time left before technology and regulations allow fully autonomous vehicles, doing all the things humans do today, on public roads there are some areas where there already exist some level of autonomy in vehicles. Some examples are autonomous parking of cars and so called lane keep assist. These type of applications have shown to have many benefits, like increasing safety on our roads and assist the driver when performing difficult maneuvers. They are also a great way to show the benefits of autonomous vehicles which helps to fund research and development of these systems and ultimately making the introduction of fully autonomous systems easier. Identifying more of these applications and show that it is possible to solve them with today’s technology is therefore of great interest.
1.2 Problem formulation

The main purpose of this thesis is to develop and implement a motion planning algorithm for a full-scale truck and trailer system. The work is a continuation of the work in [15] and the goal is to take the results from simulations and lab-tests on a small-scale truck and trailer and show that it also works on a full-scale truck and trailer system. The ambition is to have a robust demonstrator that can illustrate and communicate that it is possible to develop useful and advanced autonomous systems already with today’s technology. For this to work some parts of the algorithm need a deeper conceptual study and improvement.

1.3 Related work

Here some work relevant to this thesis is presented.

1.3.1 Modeling

A kinematic model of a general truck and trailer system has been done in [1] where a recursive formula is derived that holds under a no slip assumption on the wheels, which is valid at low speeds. The surface under the vehicle is also assumed to be relatively flat. Based on this [2] and [15] derived models suitable for control and motion planning for a general 2-trailer system. The model was then used to design controllers to stabilize the system around paths in backward and forward motion.

1.3.2 Motion planning

One common way to perform motion planning is to construct a graph search problem, like in [18] where the configuration space of the system is discretized using motion primitives and then graph search algorithms (A* [5] and D* Lite [10]) is used to solve the problem. One problem with this approach is that if the dimension of the configuration space is high the graph search problem will become very large.

Another common way to perform motion planning is to use sampling based algorithms like the Rapidly exploring Random Tree (RRT) algorithm that was first presented in [11]. The RRT algorithm builds a tree of kinodynamically feasible trajectories and the building of the tree is biased towards exploring the unexplored areas of the configuration space. This makes the RRT algorithm efficient to use even when the dimension of the configuration space is high.

A disadvantage of the RRT algorithm is that it does not give any guarantees on the optimality of the solution, to deal with this the RRT* algorithm [9] was developed. The RRT* algorithm converges to the optimal solution as the number of samples goes to infinity.

Another disadvantage of the RRT algorithm is that a control law between two system configurations has to be found. Generally when no analytical solution exists
a two point boundary value problem has to be solved in each sampling instance. For systems with complex dynamics solving the two point boundary value problem can be hard and time consuming making the RRT algorithm inefficient. If so is the case, the Closed-loop Rapidly exploring Random Trees (CL-RRT) algorithm [13] might be a better choice. The CL-RRT algorithm is based on the RRT algorithm but the samples are instead drawn in the input space to a closed loop stable system. The CL-RRT algorithm have proven to work well as a motion planner for vehicles [13, 4, 15].

1.3.3 Collision detection

In [12] and [3] they conclude that in most planning applications collision detection is the most time consuming part of the application. In [17, 14] they present collision detection algorithms that can make use of the fact that objects only move small distances between calls to the algorithm. Another common way to speed up collision detection is to use approximations when performing the collision detection, like in [22] where the bodies are approximated with discs. It is also possible to perform collision detection in parallel, like in [3].

1.4 Vehicle

The vehicle used in this thesis is a Scania R580 mining truck called Socius, see Figure 1.1. The vehicle has been prepared for autonomous operation and is equipped with numerous sensors. In this project a RTK-GPS and a LIDAR has been used. The software in Socius is module based and spread over several computers in the vehicle, communication between different modules and computers are done using Data Distribution Service (DDS).

![Figure 1.1: The Scania R580 mining truck called Socius that has been used in this thesis work.](image)
1.5 System architecture

To give the reader an overview of the system we will here explain the different components of the system. The components and how they communicates are shown in Figure 1.2. In this project we focus on the three components that are highlighted with yellow and in this thesis we focus on the motion planning part. The components are explained below.

**Environment** The environment consists of the vehicle and all other objects in proximity of it.

**High level planning** The high level planner is responsible for giving objectives to the motion planner in the form of way points where the vehicle should go. The way points can be generated in many ways, for example they could be locations to where there is something to be transported or loaded. The high level planners job would then be to determine these locations and distribute them to the available vehicles.

**Motion planning** The motion planner is responsible of finding a kinematically feasible path from the current location to a new location given by the high level planner without colliding with any obstacles in the environment. When a collision free path is found it is sent to the controller.

**Vehicle state estimation** In this module the states of the vehicle is estimated. In our case this means that the global position and orientation and the angles between the truck and the trailer should be estimated with high precision. In our case the global position and orientation is estimated using a high performance GPS and the angles between the truck and trailer are estimated using a LIDAR.

**Vehicle controller** The vehicle controller makes the vehicle follow a path given by the motion planner, to do this it must know all the states of the vehicle. In this project the controller presented in [16] is used. The controller is a path following controller with local asymptotic stability guarantees for a set of paths.

**Sensing environment** In this module the environment around the vehicle is determined. This can be done using different type of sensors like cameras, LIDARs and RADARs. Static information from maps, like positions of roads and buildings, can also be used. In our case the user or the high level planner defines where the vehicle is allowed to drive.
1.6 Outline

This thesis is organized in the following way:

**Chapter 2 – Truck and Trailer System** Here a model for the truck and trailer system and some of its properties are presented.

**Chapter 3 – Motion Planner** Here the motion planner is presented.

**Chapter 4 – Path Optimization** An optimization algorithm that optimizes paths found by the CL-RRT algorithm is presented.

**Chapter 5 – Results** Here the results of this thesis is presented.

**Chapter 6 – Conclusions** Contains some conclusions and possible future work.
In this chapter a mathematical model of the specific truck and trailer system used in this thesis will be presented along with some properties of this system. The system belongs to a class of systems called general n-trailer systems, more specifically the system is a general 2-trailer system. We will here present some theory for general n-trailers and general 2-trailers.

2.1 General n-Trailer

A general n-trailer is a truck pulling n trailers, see Figure 2.1. If we assume that the wheels roll without slip the kinematic model for the truck is the well known bicycle model:

\[
\begin{align*}
\dot{x}_1 &= v \cos \theta_1 \\
\dot{y}_1 &= v \sin \theta_1 \\
\dot{\theta}_1 &= v \frac{\tan \alpha}{L_1}
\end{align*}
\]  

(2.1)

In [1] the following recursive formula was derived for \( \theta_i \) based on holonomic and nonholonomic constraints. By assuming \( \theta_i \) and \( v_i \) are known it states that

\[
\dot{\theta}_{i+1} = \frac{v_{i+1} \sin(\theta_i - \theta_{i+1})}{L_{i+1}} - \frac{M_i \cos(\theta_i - \theta_{i+1})\dot{\theta}_i}{L_{i+1}}
\]  

(2.2)

and

\[
v_{i+1} = v_i \cos(\theta_i - \theta_{i+1}) + M_i \sin(\theta_i - \theta_{i+1})\dot{\theta}_i
\]  

(2.3)

Because of the structure in (2.1)–(2.3) Theorem 2.1 must hold.
Theorem 2.1. The velocity $v$ enters linearly in $\dot{\theta}_i$ for all $i \in \{1, 2, \ldots, n\}$.

Proof: Assume $v$ enters linearly in $\dot{\theta}_i$ and $v_i$. From (2.2) and (2.3) we see that this also implies that $v$ will enter linear in $v_{i+1}$ and $\dot{\theta}_{i+1}$. Since $v$ enters linearly in $\dot{\theta}_1$ and $v_1 = v$ we have shown, by induction, that $v$ enters linear in $\dot{\theta}_i$ and $v_i$ for all $i \in \{1, 2, \ldots, n\}$.

2.2 General 2-Trailer

Using the recursive formula (2.2)–(2.3) for the general n-trailer a model for the general 2-trailer in Figure 2.2 was derived in [15] and we will here present the result.

The states of the system are $p^T = \begin{bmatrix} x_3 & y_3 & \theta_3 & \beta_3 & \beta_2 \end{bmatrix}$, where $(x_3, y_3)$ denotes the center of the rear axle of the trailer in global coordinates, $\theta_3$ is the global orientation of the trailer, $\beta_3$ is the relative angle between the dolly and the trailer,
\( \beta_2 \) is the relative angle between the dolly and the truck. The control signals for the system are the steering angle, \( \alpha \), and the longitudinal velocity of the rear axle of the truck \( v \). The kinematic model for the general 2-trailer system is

\[
\begin{align*}
\dot{x}_3 &= v \cos \beta_3 \cos \beta_2 \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right) \cos \theta_3 \\ 
\dot{y}_3 &= v \cos \beta_3 \cos \beta_2 \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right) \sin \theta_3 \\ 
\dot{\theta}_3 &= \frac{v \sin \beta_3 \cos \beta_2}{L_3} \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right) \\ 
\dot{\beta}_3 &= v \cos \beta_2 \left( \frac{1}{L_2} \left( \tan \beta_2 - \frac{M_1}{L_1} \tan \alpha \right) - \frac{\sin \beta_3}{L_3} \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right) \right) \\ 
\dot{\beta}_2 &= v \left( \frac{\tan \alpha}{L_1} - \frac{\sin \beta_2}{L_2} + \frac{M_1}{L_1 L_2} \cos \beta_2 \tan \alpha \right)
\end{align*}
\] (2.4a,b,c,d,e)

As Theorem 2.1 stated the velocity enters linearly in (2.4) and the model can be written in a compact form as

\[
\dot{p} = v \mathbf{f}_2(p, \alpha)
\] (2.5)

### 2.3 Time reversibility

For a system with external inputs \( v \) and \( u \) and a dynamical equation which can be written on the form

\[
\dot{p}(t) = v(t) \mathbf{f}(p(t), u(t))
\] (2.6)

the following holds

**Theorem 2.2.** If the inputs \( v(t), u(t) \) \( t \in [0, T] \), when applied to (2.6) and where \( p(0) = p_i \), produce the state trajectory \( p_1(t) \). Applying the inputs

\[
\bar{v}(t) = -v(T - t) \quad \text{and} \quad \bar{u}(t) = u(T - t)
\]

on the same system but with \( p(0) = p_1(T) \), will produce the state trajectory

\[
p_2(t) = p_1(T - t), \quad \forall t \in [0, T]
\]
Proof: If we assume $p_2(t) = p_1(T - t)$ then

\[
\frac{d}{dt}p_2(t) = \frac{d}{dt}p_1(T - t) = \{\tau = T - t\}
\]

\[
= \frac{d\tau}{dt} \frac{d}{d\tau} p_1(\tau)
\]

\[
= -v(\tau)f(p_1(\tau), u(\tau))
\]

\[
= -v(T - t)f(p(T - t), u(T - t))
\]

\[
= \bar{v}(t)f(p_2(t), \bar{u}(t))
\]

This together with $p_2(0) = p_1(T)$ proves Theorem 2.2.

Since the dynamic equation of the general n-trailer can be written on the form (2.6), theoretically Theorem 2.2 says that a state trajectory where the system has been driven in forward motion from $p_i$ to $p_f$ always can be driven in backward motion from $p_f$ to $p_i$. This gives us the freedom to choose in which direction we want to perform the motion planning. Since the system is unstable in backward motion but stable in forward motion \cite{1, 2} the dynamics of the system is numerically easier to handle in forward motion, and the freedom to choose the direction therefore becomes useful.

However, in practice this result is not strictly true. Since the model is a simplified description of reality there are other conditions that makes this unrealistic, for example the assumption of no slip and the ability to choose the steering angle $\alpha$ arbitrarily at any time is often not realistic.

### 2.4 Multiple axle trailer

In the previous sections one assumption was that the trailer only had a single axle, however in reality two or three axles are very common. The assumption of no slip now becomes unreasonable, since that would only allow the trailer to move in a straight line. To deal with this we will here approximate the trailer with one axle placed in the middle of the axles. For a trailer indexed $i$ with $m$ axles this means $L_i$ will be the mean value of $L_{i,j} j \in \{1, \ldots, m\}$ where $L_{i,j}$ is the distance to axle $j$, see Figure 2.3.

![Figure 2.3: Figure illustrating a trailer with m axes.](image_url)
In this chapter the details of the motion planner used in this thesis and the improvements that have been done compared to [15] will be explained.

### 3.1 The motion planning problem

The main objective in the motion planning problem is to bring a system, described by a kinematic model

\[
\dot{p}(t) = f(p(t), u(t)) \tag{3.1}
\]

where \( p \) denotes the states of the system, from an initial configuration \( p(0) = p_i \) to a goal region \( p(T) = X_{\text{goal}} \). Generally there also exist constraints on the control signal, \( u(t) \in U(t) \), and on the state configuration, \( p(t) \in X_{\text{free}}(t) \). This can now be formulated as the following optimal control problem

\[
\begin{align*}
\text{minimize} \quad & \phi(x(T)) + \int_0^T f_0(X_{\text{free}}(t), p(t), u(t)) \, dt \\
\text{subject to} \quad & \dot{p} = f(p(t), u(t)) \\
& p(0) = p_i \\
& p(T) \in X_{\text{goal}} \\
& u(t) \in U(t) \\
& p(t) \in X_{\text{free}}(t)
\end{align*} \tag{3.2}
\]
where $\phi$ is a penalty on the end configuration and $f_0$ is a penalty on the state configuration during motion. Generally this problem is hard to solve analytically and instead numerical methods have to be used to find a suboptimal solution to (3.2). In this thesis we have used the CL-RRT algorithm to solve (3.2). In the remainder of this section we will explain how (3.2) can be modified to fit inside the CL-RRT algorithm.

### 3.1.1 Constraints

The CL-RRT makes a forward simulation of the system and during this simulation the constraints on the system are tested. This means that the CL-RRT can handle most types of constraints. Limitations in the steering mechanism makes it necessary that $\alpha(t) \leq \alpha_{\text{max}}$ and also $\dot{\alpha} \leq \dot{\alpha}_{\text{max}}$. We have used $\alpha_{\text{max}} = 0.7$ rad and $\dot{\alpha}_{\text{max}} = 1$ rad/s, both are slightly smaller than what the steering mechanism is capable of to leave room for the controller to suppress disturbances. We have also used the constraints $\beta_3 < 0.6$ rad and $\beta_2 < 0.6$ rad, partly because too large angles would make the vehicle collide with itself but also because trajectories that do not meet these conditions are harder to follow for a controller. It is also required that the vehicle does not collide with any of the obstacles in its surroundings, for this a collision detection algorithm is used.

### 3.1.2 Constraints on the final configuration

The goal region $X_{\text{goal}}$ defines the constraints on the final configuration. Given a desired goal position $p_g^T = [x_{3,g}, y_{3,g}, \theta_{3,g}, \beta_{3,g}, \beta_{2,g}]$ the constraints are defined as

\[
\begin{aligned}
(x_3(T) - x_{3,g})^2 + (y_3(T) - y_{3,g})^2 &\leq \Delta r \\
|\theta_3(T) - \theta_{3,g}| &\leq \Delta \theta_3 \\
|\beta_3(T) - \beta_{3,g}| &\leq \Delta \beta_3 \\
|\beta_2(T) - \beta_{2,g}| &\leq \Delta \beta_2
\end{aligned}
\]

(3.3)

where we have chosen $\Delta r = 2$ m, $\Delta \theta_3 = 0.07$ rad and $\Delta \beta_3 = \Delta \beta_2 = 0.08$ rad.

### 3.1.3 Objective function

The objective function in (3.2) can be generalized as a cost of a trajectory. Like in [15] we have used a weighted sum between the length of the path of the rear axle and the deviation from the desired goal position $p_g^T = [x_{3,g}, y_{3,g}, \theta_{3,g}, \beta_{3,g}, \beta_{2,g}]$. The objective function then becomes

\[
v = \sum_{i=1}^{N} \left( \sqrt{(x_3[i] - x_3[i-1])^2 + (y_3[i] - y_3[i-1])^2} + 1000(\theta_3[N] - \theta_{3,g})^2 + 1000(\beta_3[N] - \beta_{3,g})^2 + 1000(\beta_2[N] - \beta_{2,g})^2 \right)
\]

(3.4)

where $N$ is the number of discrete points in the trajectory.
3.2 Closed-loop system

In the CL-RRT algorithm a stable closed-loop system is needed. In this thesis we have used the same closed loop system as in [15], we will therefore only present the main details here. In the closed-loop system a pure pursuit controller is used as a high level controller. The pure pursuit controller calculates a look-ahead point \( P \) and tries to steer the vehicle towards this point. The look-ahead point is calculated by finding the intersection of a look-ahead circle and a reference path. The look-ahead circle has its center in a reference position \( P^* \) in the vehicle and a radius \( L_r \), which is a design parameter. The reference path is in this case piece-wise constant and made up out of line segments between a sequence of points. An illustration of the pure pursuit controller is shown in Figure 3.1.

![Figure 3.1: A figure presenting the geometry and notation used in the pure pursuit controller.](image)

In forward motion the reference point of the vehicle is the center of the rear axle of the truck and the pure pursuit controller calculates a steering angle that steers the vehicle towards the look-ahead point. In backward motion the reference point of the vehicles is instead the center of the rear axle of the trailer and the pure pursuit controller calculates a reference for the angle between the trailer and the dolly, \( \beta_{3,\text{ref}} \). Since the system is unstable in backward motion a gain scheduled LQ controller is used to stabilize \( \beta_3 \) and \( \beta_2 \) around the circular equilibrium point \((\beta_{3,e}, \beta_{2,e}, \alpha_e)\) defined by \( \beta_{3,e} = \beta_{3,\text{ref}} \). This means we can see \( \beta_{3,\text{ref}} \) as a virtual steering wheel and the controller steers the vehicle to achieve \( \beta_3 = \beta_{3,\text{ref}} \).
3.3 Closed-loop rapidly-exploring random tree algorithm

Since the system is unstable in backward motion a control law that stabilizes the system must be used. This makes it hard to use the RRT algorithm and therefore we instead use the closed-loop rapidly-exploring random tree (CL-RRT) algorithm, where instead a closed-loop stable version of the system is used.

The CL-RRT algorithm is shown in Algorithm 1. First a tree structure is initialized (line 1). After this a loop where the tree structure is filled with nodes, creating a tree of dynamically feasible trajectories, is started (line 2-17). Each iteration in the loop a random sample, \( s_{\text{rand}} = [x_{\text{rand}}, y_{\text{rand}}, \theta_{\text{rand}}] \), is drawn (line 3). The sample contains an input for the stabilizing controller of the system \( (x_{\text{rand}}, y_{\text{rand}}) \) and an angle \( \theta_{\text{rand}} \).

The nodes in \( T \) are then sorted according to the cost of connecting them to the random sample \( s_{\text{rand}} \), creating a sorted list \( Q_{\text{sorted}} \) (Line 4). The cost of connecting two nodes is evaluated using a heuristic.

On line 5 a loop where an extension attempt from each element \( q \) in \( Q_{\text{sorted}} \), starting with the first (with lowest cost), are started. The extensions are done by generating a trajectory from \( q \) to \( s_{\text{rand}} \) by using \( s_{\text{rand}} \) as an input to the controller.

If the trajectory produced in an extension attempt obeys the constraints on the system a new node is created and added to the tree (line 8). When a new node is added to the tree the loop started on line 5 is eventually terminated (line 17). However, before the loop is terminated a goal sample \( s_{\text{goal}} \) is drawn (line 10) and then an extension attempt from the new node towards \( s_{\text{goal}} \) is made (line 11). If the extension obeys the constraints on the system the node is added to the tree (line 14) and if it satisfies the constraints on the final configuration it is marked as a solution candidate to the problem (line 16).

After this the main loop starts over or if time limit is reached the best solution, if there is one, is sent to the controller (line 18-19).

3.4 Uniform sampling of complex areas

The input space to the stabilizing controller is the whole \( \mathbb{R}^2 \), however only a small part of it is suitable for sampling. Therefore this area should be determined and a method to draw samples in this area is needed.

First we can approximate the area where the vehicle is allowed to drive with a polygon, which the vehicle should be completely inside. The area suitable for sampling is then, when using a pure pursuit controller with look-ahead distance \( R \), the outward offset polygon of this polygon, with offset \( R \). This polygon often becomes very complex which makes it hard to draw samples in it. Therefore we instead approximate the polygon with equally sized squares in a grid. This is
Algorithm 1: CL-RRT algorithm

1. Initialize the planning tree $T$
2. while Time limit not reached do
3.     $s_{\text{rand}} \leftarrow$ Random sample
4.     $Q_{\text{sorted}} \leftarrow$ Sort nodes in $T$ according to cost to extend to $s_{\text{rand}}$
5.     for each node $q$ in $Q_{\text{sorted}}$ do
6.         $\sigma \leftarrow$ Generate trajectory from $q$ towards $s_{\text{rand}}$
7.         if $\sigma$ is collision free then
8.             $N \leftarrow$ Create node at the end of $\sigma$ with $x_{\text{near}}$ as parent.
9.             Insert $N$ in $T$
10.            $s_{\text{goal}} \leftarrow$ Random goal sample
11.            $\sigma_{\text{goal}} \leftarrow$ Generate trajectory from $N$ to $s_{\text{goal}}$
12.            if $\sigma_{\text{goal}}$ is collision free then
13.                $\bar{N}_{\text{goal}} \leftarrow$ Create node at the end of $\sigma_{\text{goal}}$ with $N$ as parent.
14.                Insert $\bar{N}_{\text{goal}}$ in $T$
15.                if $\bar{N}_{\text{goal}}$ satisfies the constraints on final configuration then
16.                    Mark $\bar{N}_{\text{goal}}$ as possible solution
17.                    break
18. if If a solution exists in $T$ then
19.     Send the best solution in $T$ to the controller

illustrated in Figure 3.2. A sample of this space can then be created by first randomly choose one of the squares in the approximation and then draw an uniform sample in this square.

3.4.1 Outward offset polygon

Determining the outward offset polygon of a general polygon is a complex operation and since we are only interested in marking this polygon in a grid it is enough to only determine the outward offset polygon of each line segment in the polygon and mark these in the grid. The outward offset polygon of a line segment, if we see it as a rectangle with zero width, consists of two circles and a rectangle, see Figure 3.3.

3.4.2 Approximating a polygon with a grid of squares

When approximating a polygon with a grid of squares each square in the grid that has any point inside the polygon should be a part of the approximation. To determine if a square has any point inside the polygon it is enough to determine if the perimeter of the square intersect with the perimeter of the polygon or if any of the four corners of the square lies inside the polygon.

To determine if a point lies inside a polygon one can count the number of in-
Figure 3.2: Figures showing how the squares for the sampling strategy is created. In (a) the area where driving is allowed is marked with green, in (b) the area suitable for sampling is marked with blue and finally in (c) the area suitable for sampling is approximated with red squares.

Figure 3.3: Figure illustrating the outward offset polygon of a line segment (red line). The outward offset consists of the perimeter of two circles, one in each end point, and a rectangle.

3.5 Heuristics

The heuristic is an important part of the CL-RRT algorithm and the quality of the solution often depends on its the ability to correctly estimate the cost of connecting two nodes [12]. Since evaluation of the heuristics is time consuming a heuristic map was calculated offline in [15]. To produce the map the system was simulated from an initial configuration where all states were equal to zero and to different locations creating a grid with extension costs. One drawback with this approach is that when using the heuristic map, different values on \( \beta_3 \) and \( \beta_2 \) do not result in different costs. Therefore we here extend the heuristic map to also
take into account different values on $\beta_3$ and $\beta_2$. We have use the length of the extension as cost.

When extending the heuristic map to also include $\beta_2$ and $\beta_3$ the map quickly becomes very large if we want a high resolution. For example if we want a range on $\beta_3$ and $\beta_2$ from $-\frac{\pi}{2}$ rad to $\frac{\pi}{2}$ rad with an accuracy of 0.1 rad and a range on $x_3$ and $y_3$ from $-100$ m to $100$ m with an accuracy of 1 m the number of elements in the map would be close to 40 million. To be able to reduce the number of elements in the map without reducing the performance significantly we use a map with less accuracy but interpolate values in the map.

**Figure 3.5:** Heuristic maps for some different $\beta_{3,i}$ and $\beta_{2,i}$. White areas mark where the simulation didn’t succeed and the coloring going from blue to yellow mark rising cost.

### 3.5.1 Jerk optimal velocity profile

The CL-RRT algorithm produces a path without a velocity profile, however a velocity profile is needed to make the vehicle start and stop in a controlled manner.
Here we have used a velocity profile which consists of three parts: an acceleration part, a part with constant velocity and a deceleration part.

The following model is used to describe the longitudinal movement of the vehicle

\[
\begin{align*}
\dot{s} &= v \\
\dot{v} &= a \\
\dot{a} &= u
\end{align*}
\]  

(3.5)

where \( s \) is the distance moved along the path, \( v \) is the velocity, \( a \) is the acceleration and \( u \) is the jerk. We also introduce the notation \( x^T = [v, a] \) so that (3.5) can be written \( \dot{x} = Ax + Bu \) where

\[
A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
\]

For a comfortable ride it is desirable to minimize the jerk \([21, 6]\), and the following optimal control problem is therefore of interest

\[
\begin{align*}
\text{minimize} & \quad \int_0^T \frac{1}{2} u^2 \, dt \\
\text{subject to} & \quad \dot{x} = Ax + Bu \\
& \quad v(0) = v_i \\
& \quad v(T) = v_f \\
& \quad a(0) = 0 \\
& \quad a(T) = 0
\end{align*}
\]

To solve this we first define the Hamiltonian as

\[
H(u, x, \lambda) = \frac{1}{2} u^2 + \lambda_v a + \lambda_a u
\]  

(3.6)

where \( \lambda^T = [\lambda_a, \lambda_v] \) is the Lagrange multiplier vector. According to the Pontryagin’s minimum principle \([19]\) the necessary and sufficient conditions for optimality becomes

\[
\begin{align*}
\dot{u}^* &= \arg \min_u H(x(t), u(t), \lambda(t), t) \\
\dot{\lambda} &= -H_x(x^*(t), u^*(t), \lambda(t), t)
\end{align*}
\]  

(3.7a)

(3.7b)

Since \( H \) is strictly convex w.r.t. \( u \) pointwise minimization yields

\[
H_u = u + \lambda_a = 0
\]  

(3.8)

and thus

\[
u^* = -\lambda_a
\]  

(3.9)
and from (3.7b) we get

\[ \dot{\lambda} = -H_x = \begin{bmatrix} 0 \\ -\lambda_v \end{bmatrix} \Rightarrow \lambda(t) = \begin{bmatrix} C_1 \\ -C_1 t + C_2 \end{bmatrix} \] (3.10)

Now, using \( \dot{a} = u = -\lambda_a \) we get

\[ a(t) = \frac{1}{2} C_1 t^2 - C_2 t + C_3 \] (3.11)

and since \( \dot{v} = a \) we get

\[ v(t) = \frac{1}{6} C_1 t^3 - \frac{1}{2} C_2 t^2 + C_3 t + C_4 \] (3.12)

The boundary conditions can now be written

\[
\begin{bmatrix}
    v(0) \\
    v(T) \\
    a(0) \\
    a(T)
\end{bmatrix} =
\begin{bmatrix}
    \frac{1}{6} C_1 T^3 - \frac{1}{2} C_2 T^2 + C_3 T + C_4 \\
    \frac{1}{6} T^3 - \frac{1}{2} T^2 T + 1 \\
    0 \\
    \frac{1}{2} T^2 - T + 1
\end{bmatrix}
\begin{bmatrix}
    C_1 \\
    C_2 \\
    C_3 \\
    C_4
\end{bmatrix}
= \begin{bmatrix}
    v_1 \\
    a_1 \\
    v_2 \\
    v_3
\end{bmatrix}
\] (3.13)

When \( T > 0 \) we get the unique solution

\[
\begin{bmatrix}
    C_1 \\
    C_2 \\
    C_3 \\
    C_4
\end{bmatrix} = \begin{bmatrix}
    \frac{12}{T^3} (v_i - v_f) \\
    \frac{6}{T^2} (v_i - v_f) \\
    0 \\
    v_i
\end{bmatrix}
\] (3.14)

finally we get

\[
\begin{align*}
    u(t) &= \frac{12}{T^3} (v_i - v_f)t - \frac{6}{T^2} (v_i - v_f) \\
    a(t) &= \frac{6}{T^3} (v_i - v_f)t^2 - \frac{6}{T^2} (v_i - v_f)t \\
    v(t) &= \frac{2}{T^3} (v_i - v_f)t^3 - \frac{3}{T^2} (v_i - v_f)t^2 + v_i
\end{align*}
\] (3.15)

Using \( \dot{s} = v \) and \( s(0) = 0 \), we get

\[ s = \frac{1}{2T^3} (v_i - v_f)t^4 - \frac{1}{T^2} (v_i - v_f)t^3 + v_i t \] (3.16)

Different values on \( T \) will give different \( a(t) \) and \( u(t) \), for comfort reasons and because of limitations in the system it is desirable that \( |u(t)| \leq u_{\text{max}} \) and \( |a(t)| \leq a_{\text{max}} \) where \( u_{\text{max}} \) and \( a_{\text{max}} \) is the maximum value on jerk and acceleration respectively. For example, in [6] they conclude that for a comfortable ride the jerk should not exceed 3 m/s\(^3\) and the acceleration should not be larger than 1.5 m/s\(^2\). Since

\[ \ddot{u} = \frac{12}{T^3} (v_i - v_f) \] (3.17)
is constant, $|u(t)|$ must have its maximum when $t = 0$ or $t = T$. Since

$$u(0) = -\frac{6}{T^2} (v_i - v_f) = -u(T) \tag{3.18}$$

we can conclude that

$$\max |u(t)| = \left| \frac{6}{T^2} (v_i - v_f) \right| = \frac{6}{T^2} |v_i - v_f| \tag{3.19}$$

Thus for a longitudinal speed profile to fulfill the constraint $|u| < u_{\text{max}}$ the following inequality must hold

$$T \geq \sqrt{\frac{6 |v_i - v_f|}{u_{\text{max}}}} \tag{3.20}$$

Furthermore since

$$\dot{a} = u(t) = \frac{12}{T^3} (v_i - v_f)t - \frac{6}{T^2} (v_i - v_f) = 0 \tag{3.21}$$

has the unique solution $t = \frac{T}{2}$ and $a(0) = a(T) = 0$ we can conclude that

$$\max |a(t)| = \left| a \left( \frac{T}{2} \right) \right| = \left| -\frac{3}{2T} (v_i - v_f) \right| = \frac{3}{2T} |v_i - v_f| \tag{3.22}$$

Thus, to satisfy $|a(t)| \leq a_{\text{max}}$ we get that $T$ also has to fulfill the inequality

$$T \geq \frac{3 |v_i - v_f|}{2a_{\text{max}}} \tag{3.23}$$

This result can now be used to create the acceleration and decelerating parts of the velocity profile. In the acceleration part we use $v_i = 0$ and $v_f = v_d$, where $v_d$ is desired velocity on the path, and in the deceleration part we use $v_i = v_d$ and $v_f = 0$. In both cases using (3.16) we get

$$s(T) = \frac{1}{2}(v_i + v_f)T = \frac{1}{2} v_d T \tag{3.24}$$

and thus the acceleration part and deceleration part are both of length $\frac{1}{2} v_d T$. From this we can now construct a velocity reference and an example is shown in Figure 3.6.

### 3.6 Collision detection

In this section an efficient way to perform collision detection between a target and some obstacles is presented. The target is made up of a rectangle and the obstacles are made up of line segments. This fits well inside the CL-RRT framework (the truck is approximated with two rectangles).

Collision detection can easily be done by testing if any segment belonging to the target intersect with any of the obstacles, see Figure 3.7. However, as the number
3.6 Collision detection

![Figure 3.6: A velocity profile for a 70 meters long trajectory with the desired velocity \( v_d = 5 \text{ m/s} \). The limitations on \( a \) and \( u \) are marked with red dotted lines. In this case it is the constraint on \( u \) that is active, but \( a \) is also very close to its limits.](image)

of obstacles increases this becomes costly. Here a method that speeds up this approach is presented.

![Figure 3.7: Two polygons in two different scenarios, one where they are not in collision (a) and one where they are in collision (b).](image)

The main idea of this algorithm is that if two objects are at a distance \( d \) of each other collision detection between these objects will not be necessary until they have moved at least a distance of \( d \) relative to each other. In our case this means that if we know the distance \( d_i \) from the target to the line segment \( S_i \) then collision detection between these two will not be necessary until any point on the target have moved at least \( d_i \), see Figure 3.8.

Instead of calculating exactly how far the target has moved we can, each time the target moves, subtract the length moved from \( d_i \) and thus \( d_i \) will be a lower estimate of the distance between them.
Figure 3.8: Figure showing the target $T$ and three obstacles, $S_1$, $S_2$ and $S_3$.

Figure 3.9: Figure illustrating a rectangle moving from $P_1$ with orientation $\theta_1$ to $P_2$ with new orientation $\theta_2$.

When the target moves from the point $P_1$ and orientation $\theta_1$ to a new position $P_2$ with a new orientation $\theta_2$ we can easily calculate how long the reference point moved like

$$l = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$  \hfill (3.25)

However in the collision detection algorithm we want to know the maximum length any point on the target have moved. This is a harder task but a reasonably good upper estimate can be calculated as

$$l' = l + r|\theta_2 - \theta_1|$$  \hfill (3.26)

where $r$ is the longest distance from the reference point to any point in the rectangle, which is one of its vertices. Note that $r$ is constant and only need to be calculated once. An illustration of this is shown in Figure 3.9.
Path Optimization

Since the CL-RRT algorithm does not give any optimality guarantees on the solution [9] and tend to give clearly suboptimal solutions, it is likely that using the path found by the CL-RRT algorithm as an initial guess for further numerical optimization could improve the solution. In [8] this approach has been integrated in RRT* with good results. In principle, this can be divided into two problems. The first is that the probability that the CL-RRT algorithm finds a solution that takes the system to the desired goal configuration $p_f$ is very low and the other is that there are no guaranties of the quality of a path from $p_i$ to $p_f$ found by the CL-RRT algorithm, here we will mainly focus on the first.

4.1 Reaching the desired goal configuration

Since the probability that the CL-RRT algorithm finds a solution with the exact desired end configuration $p_f$ is very low, we here present a way to achieve this by optimizing the end of the trajectory found by the CL-RRT algorithm.

Assume $\sigma(s)$ is the trajectory found by the CL-RRT algorithm, where $0 \leq s \leq s_{\text{max}}$ denotes the length moved along the trajectory. The goal here is to take the last part of the trajectory $\sigma(s)$, $s_o \leq s \leq s_{\text{max}}$, where $s_o$ is some point close to the end of the trajectory, and optimize it so that the desired end configuration is reached. To achieve this we formulate the problem as an optimal control problem which can be solved numerically using the initial path $\sigma$ as an initial guess for the solution.

The point $s_o$ from which the optimization starts has great impact on the solution to the problem, choosing a large $s_o$ close to the end of the trajectory leaves little room for the optimization and might make the problem unsolvable and small $s_o$ results in a more complex problem. It is therefore desirable to choose $s_o$ large
enough to be able to use the optimization algorithm but not too large so that the problem becomes too complex. Figure 4.1 shows for which values on $x_f$ and $y_f$ the optimization algorithm was able to solve the problem of going from $p_i = [0 0 0 0]^T$ to $p_f = [x_f \ y_f \ 0 \ 0]^T$ (see Figure 4.2 for an illustration of the problem), green areas mark where the optimization succeeded and red where it failed. In Figure 4.3 the states when doing a parallel movement close to the border, where the optimization fails, is shown. As can be seen this requires the steering angle $\alpha$ to be saturated at most times. In Figure 4.1 we see that $x_f > 20$ is required to be able to make any larger parallel movements, and since this is approximately the length of the vehicle we can conclude that approximately one vehicle length to optimize on, i.e. it is appropriate that $s_{\text{max}} - s_0$ is around one vehicle length.

**Figure 4.1:** Figure illustrating where the a solution was found to the parallel movement (green) and where it did not (red).

**Figure 4.2:** Figure illustrating a parallel movement.
4.1 Reaching the desired goal configuration

4.1.1 The optimal control problem

Since the dynamics in the steering mechanism is not modelled the model is less accurate when the steering effort, $\dot{\alpha}$, is high. It is therefore desirable to minimize $\dot{\alpha}$, we therefore introduce the virtual control signal $u = \dot{\alpha}$.

Since we are only interested in a path between $p_i$ and $p_f$ we will let $v$ to be constant. Also, from Theorem 2.2 we know that we only need to be able to solve the problem in one direction since this solution will be valid when driving in both directions, so we can always use $v = 1$. This also means that the length of the optimal path will be $L = vT = T$.

We can now write this as the following optimal control problem

$$\min_{u(\cdot)} \int_0^T u(t)^2 \, dt$$

subject to

\begin{align*}
\dot{p}(t) &= f_2(p(t), \alpha) \\
\dot{\alpha} &= u \\
p(0) &= p_i \\
p(T) &= p_f \\
|u| &\leq u_{\text{max}} \\
|\alpha| &\leq \alpha_{\text{max}}
\end{align*}

where $u_{\text{max}}$ and $\alpha_{\text{max}}$ denote the maximum steering angle rate and maximum steering angle respectively. The problem is solved numerically using ACADO.
toolkit [7] which uses direct multiple shooting that will be described in the next section.

## 4.2 Direct Multiple Shooting

The direct multiple shooting method [20] numerically solves an optimal control problem on the following form

\[
\begin{align*}
\text{minimize} & \quad \Phi(x(T)) + \int_{T_i}^{T_f} f_0(x(t), u(t)) dt \\
\text{subject to} & \quad \dot{x}(t) = f(x(t), u(t)) \\
& \quad x(T_i) \in X_i \\
& \quad x(T_f) \in X_f \\
& \quad h(x(t), u(t)) \leq 0
\end{align*}
\]

by first approximating the control with a piecewise constant function. The control law then becomes

\[
u(t) = u_k, \ t \in [t_{k-1}, t_k)
\]

for \( k = 1, \ldots, N \), see Figure 4.4. The system dynamics can then be written as

\[
\dot{x}(t) = f(x(t), u_k), \ t \in [t_{k-1}, t_k).
\]

By introducing artificial initial values, \( s_k \), at each time interval we get the following ODE:s

\[
\begin{align*}
\dot{x}_k(t; s_k, u_k) &= f(x_k(t; s_k, u_k), u_k), \ t \in [t_{k-1}, t_k) \\
x_k(t_k; s_k, u_k) &= s_k
\end{align*}
\]

for each time interval \( k = 0, \ldots, N-1 \), which can be solved separately from each other, see Figure 4.5. However, for the state trajectory to be continuous it must hold that

\[
x_k(t_{k+1}; s_k, u_k) = s_{k+1}.
\]

The integral in the objective function is solved numerically on each time interval as

\[
F_{0,k}(s_k, u_k) = \int_{t_k}^{t_{k+1}} f_0(x_k(t_{k+1}; s_k, u_k), u_k) dt.
\]
4.3 Local optimization

by solving the following ODEs

\[
\dot{F}_{0,k}(t; s_k, u_k) = f_0(x_k(t_{k+1}; s_k, u_k), u_k), \quad t \in [t_k, t_{k+1})
\]

(4.7)

\[
F_{0,k}(t_k; s_k, u_k) = 0
\]

(4.8)

and then

\[
F_{0,k}(s_k, u_k) = F_{0,k}(t_{k+1}; s_k, u_k)
\]

(4.10)

The constraints on \( x \) and \( u \) can be tested on each time interval as

\[
h(s_k, u_k) \leq 0
\]

(4.11)

Finally, this can be written as the following NLP

\[
\begin{align*}
\text{minimize} & \quad \Phi(s_N) + \sum_{k=0}^{N-1} F_{0,k}(s_k, u_k) \\
\text{subject to} & \quad x_k(t_{k+1}; s_k, u_k) = s_{k+1}, \quad k = 0, ..., N - 1 \\
& \quad s_0 \in X_i \\
& \quad s_N \in X_f \\
& \quad h(s_k, u_k) \leq 0, \quad k = 0, ..., N
\end{align*}
\]

which can be solved using standard NLP solution methods. Since the problem is very structured, tailored methods that exploit this should be used. In this thesis ACADO Toolkit [7] has been used.

\[\begin{array}{cccc}
T_i & t_{k-1} & t_k & T_f \\
\hline
u(t) \\
u_1 & u_2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & u_{k+1} & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & u_N \\
T_i & t_{k-1} & t_k & T_f \\
\end{array}\]

Figure 4.4: Figure illustrating how the control signal is approximated.

4.3 Local optimization

The optimization presented above can also be used on other parts of the trajectory \( \sigma(s) \) found by the CL-RRT algorithm, not only the last part. One interesting use of the optimization algorithm is to find parts of the trajectory which are hard to follow for a controller and optimize these parts. In our case it is desirable to optimize parts of the trajectory where the steering rate \( \frac{d\alpha}{ds} \) is high. If a part of the trajectory \([s_i, s_f]\) where the steering effort is high we can use the optimization
Figure 4.5: Figure illustrating how the states are solved separately on each interval.

algorithm with $p_i = \sigma(s_i)$ and $p_f = \sigma(s_f)$ to find a new smoother trajectory.
5

Results

5.1 Collision detection

Here we will present some results from the collision detection algorithm presented in Section 3.6. Since the performance of the collision detection algorithm depends on how the vehicle moves the algorithm has been evaluated when running inside the CL-RRT algorithm as this is the way it is intended to be used.

To evaluate the use of distance calculation when doing the collision detection the problem in Figure 5.1 has been used. The problem is to go from A to B avoiding the obstacles 1 to 12. The problem has been solved multiple times both with and without calculating distances to speed up the collision detection, and with different number of obstacles of the 12 obstacles present at each time.

In Figure 5.2 (a) the average time spent on one collision detection iteration is shown. The time seem to increase linearly in both cases but when using distance calculation the time is always smaller and also increases less when the number on obstacles increases. The reason why the time spent in collision detection is not zero when the number of obstacles is zero is because collision detection with the outer perimeter is always done. Figure 5.2 (b) shows the relative speedup from the same data, clearly showing the benefits of using distance calculation. Already with 6 obstacles the speedup is over 100% and with 12 obstacles the speed up is of around 200%.

In Figure 5.3 the proportion of time spent on collision detection is shown, both with and without calculating distances to speed up the algorithm. As can be seen, the proportion is a lot smaller when distances are used to speed up the algorithm. It is interesting that the proportion of time spent on collision detection, when not using distance calculation, does not seem to increase with the number of
**Figure 5.1:** A motion planning problem where the truck should go from A to B avoiding obstacle 1 to 12.

**Figure 5.2:** Two figures showing some performance data from the collision detection algorithm compared to the old method.

Obstacles when the number of obstacles are larger than around six. This is probably because that the forward simulations become shorter when the number of obstacles increase, since more collisions occur, which leads to fewer collision detections per iteration of the CL-RRT algorithm. However, when using distance calculation the time spent on collision detection seems to tend to zero as the number of obstacles increases.
5.2 Heuristics

To evaluate how using $\beta_2$ and $\beta_3$ in the heuristics map influence the performance of the CL-RRT algorithm the problem shown in Figure 5.4 has been solved both using $\beta_2$ and $\beta_3$ in the heuristics and without. The algorithm was run until a solution was found or a time limit of 30 seconds was reached. The algorithm was restarted 200 times, using both heuristics maps, to produce reliable results. The results of the runs are shown in Figure 5.5. As can be seen, when using $\beta_2$ and $\beta_3$ in the heuristics around 20 % of the runs failed to find a solution and when $\beta_2$ and $\beta_3$ were not used in the heuristics map this number was around 60 %.

Figure 5.3: Figure showing the proportion of time spent on collision detection, both with and without calculating distances to speed up the algorithm.

Figure 5.4: The problem used to evaluate the heuristics maps.
5.3 Optimization

To evaluate how well the optimization of the end configuration is working the problem in Figure 5.6 has been used. In this case the CL-RRT algorithm was not able to find a solution close to the desired end configuration. However, the optimization algorithm was able to find a solution, see Figure 5.7 and Figure 5.8. In this case the optimized trajectory differs quite a lot from the original trajectory found by the CL-RRT algorithm, this is because the original end configuration was very far from the desired one.

Figure 5.6: A problem where the truck should make a parallel parking. One solution to the problem is marked with a blue and green line, the blue line is found by the CL-RRT algorithm and the green is the optimized part.
5.4 Reversibility

Here we will show some benefits of using the result in Theorem 2.2. For this end we have used the problem in Figure 5.9. The problem is to reverse from A to B, however we know from Theorem 2.2 that it can be solved by driving forward from B to A and then reverse the solution. The CL-RRT algorithm has been used to solve the problem 500 times in both directions and the result is shown in Figure 5.10. As can be seen, when solving it by driving in reverse the algorithm failed to find a solution around 30% of the times. However, when driving forward it always found a solution. In Figure 5.12 the states from two solutions are shown, one when driving forward and one driving in reverse, here we can also see that the state trajectories from the solutions when driving forward is a lot more smooth which makes them easier to follow for a controller.
Figure 5.9: A problem where the truck should reverse from A to B.

Figure 5.10: Distribution of how long time it took to find a solution to the problem in Figure 5.9, both solving by driving forward and driving in reverse.

Figure 5.11: Path of the trailer in two solutions to the problem in Figure 5.9, one driving forward (blue) and one driving in reverse (red).
Figure 5.12: The states for two solutions to the problem in Figure 5.9, one driving forward (blue) and one driving in reverse (red). Here we can see that the steering effort is a lot higher for the one created by driving in reverse.
5.5 Motion planning with a full-scale truck and trailer system

Here we will present some results from some experiments made with a full sized truck and trailer system on a test track at Scania in Södertälje.

The planner was used to produce three different types of trajectories. The first type are trajectories from the CL-RRT algorithm created by performing forward simulations in backward motion. These type of trajectories proved to be quite hard for the controller to follow and most often the controller was not able to execute them. However, for some problems, like that in Figure 5.13, successful attempts where made. The reference states of one of the successful attempts are shown in Figure 5.14.

Figure 5.13: A problem and a possible solution created by simulating the system in backward motion.

The second type are trajectories from the CL-RRT algorithm created by simulating the system in forward motion. These types of trajectories proved to be a lot easier for the controller to follow. As long as $|\beta_3| < 0.6$ rad and $|\beta_2| < 0.6$ rad successful execution of these trajectories was possible in almost all cases. One example is the problem in Figure 5.15. In Figure 5.16 the reference states of one solution to this problem are shown, as can be seen this problem required quite advanced movements but the controller was still able to follow the entire reference trajectory.

The third and last type of trajectories are trajectories produced by the optimization algorithm presented in Section 4.1.1. These type of trajectories proved to be very easy for the controller to follow and all execution attempts of these type
5.5 Motion planning with a full-scale truck and trailer system

of trajectories were successful. In Figure 5.17 an example of such trajectory is shown and in Figure 5.18 the reference states of this solution is shown.
Figure 5.16: Reference states for a solution to the problem in Figure 5.15 created by simulating the system in forward motion.

Figure 5.17: A problem solved by the optimization algorithm.
Figure 5.18: Reference states for the problem in Figure 5.17.
In this thesis improvements for a CL-RRT based motion planner for a reversing truck and trailer system have been done. The motion planning algorithm has also been implemented and tested on a full scale truck and trailer system.

The main contribution of this thesis work has been to evaluate the performance of a CL-RRT based motion planner on a full scaled truck and trailer system. In this chapter we will summarize the main results and discuss some relevant future work.

6.1 Conclusions

In this thesis we have shown that motion planning for reversing general n-trailer can be done by driving forward and then reverse the solution. Results that indicate great benefits with this have been achieved, especially in the CL-RRT framework.

The importance of correctly estimating the cost of connecting two nodes, with a heuristic, in the CL-RRT algorithm has been evaluated and we have found that a more correct heuristic can greatly improve the result. We have also shown that when performing collision detection during a forward simulation the time spent in the collision detection algorithm can be reduced by calculating distances between the bodies and then use this information to be able to perform less calculations.

The use of post optimizing of trajectories found by the CL-RRT algorithm has been evaluated and has shown promising results. We have also seen that trajectories produced by the optimization algorithm, where the steering rate is mini-
mized, is well suited to use as reference for a controller.

6.2 Future work

The CL-RRT algorithm now uses piecewise constant references (line segments) and when the reference changes this tends to give quite aggressive behaviour of the closed loop system making the resulting trajectory hard to follow for a controller. It would therefore be of interesting to evaluate the use of smoother references for example circles and lines or so-called clothoids. This should result in solutions that are easier to follow for a controller.

The pure pursuit controller used in the CL-RRT algorithm calculates a reference for the angle between the trailer and the dolly, $\beta_{3,\text{ref}}$, since this reference varies with time it would be interesting to determine how it varies, i.e. find $\dot{\beta}_3$, and use this information when calculating the reference for the angle between the dolly and the truck and the steering angle.

The optimization algorithm used in this thesis has proven to work well to produce trajectories that are easy to follow for a controller. It would, however, be interesting to see if different types of numerical solvers could be used to reduce calculation time, making it applicable in more places. It would also be interesting to use the trajectories produced by the optimization algorithm as motion primitives and formulate the problem as a graph search problem. However, because of the high dimension of the system dynamics, some way to reduce the dimension of the problem would probably be needed for this to work.


[22] Julius Ziegler and Christoph Stiller. Fast collision checking for intelligent...