Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Edvard Grødem

Closed-Loop-RRT* path planning for a vehicle-trailer system

Master's thesis in Cybernetics and Robotics
Supervisor: Sebastien Gros
January 2021

NTNU
Kunnskap for en bedre verden

Edvard Grødem

Closed-Loop-RRT* path planning for a vehicle-trailer system

Master's thesis in Cybernetics and Robotics
Supervisor: Sebastien Gros
January 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

The last decade has brought considerable improvements to the field of autonomous vehicles, due to their potential of saving cost and increasing safety. This master thesis studies autonomous vehicles with a trailer attached. The trailer adds an unstable degree of freedom to the system, making the control of the system more complicated. A path planner algorithm is required for an autonomous vehicle-trailer system to navigate an unstructured environment, such as a parking lot. The path planner must create a feasible path that the vehicle-trailer system can follow while also avoiding obstructions. This is a challenging task due to the nonholonomic constraint on the system that limits the possible paths. Care must also be taken in reverse motion such that the trailer does not fold onto the vehicle in a jackknife configuration.

This master thesis presents a new planner for creating a feasible path between two set of states of the vehicle-trailer system in an unstructured environment. Such a planner is called a local planner since it cannot navigate around obstructions. The feasible path is generated by first approximating the path with a Dubins path and then improving upon the path in two additional stages. Each of these stages simulates a closed-loop vehicle-trailer around the reference created by the previous stage. This results in a feasible path that a vehicle-trailer system can follow from one pose to the other.

A global planner based on Rapidly-exploring Random Tree Star (RRT*) and Closed-Loop RRT (CL-RRT) is also presented. RRT* has the advantage over other RRT-based planners in that it is probabilistic optimal, meaning the solution will converge to an optimal solution with time. The local planner described above is used to find shortcuts in the path, enabling the optimal behavior of RRT*. To the authors' knowledge RRT* has not been implemented for vehicle-trailer systems before, likely due to the problem of finding a suitable local planner.

The global planner was tested using Monte Carlo-simulations in two environments. The results shows that the global planner is able to create feasible and short paths in a realistic scenario simulating a parking lot. It is also shown that the local planner created feasible paths between a wide range of states. However, the convergence rate of the algorithm is too slow for most real-time applications that require an optimal solution.

Sammendrag

I de siste 10 årene har det blitt gjort betydelige fremskritt innenfor utvikling av autonome kjøretøy. Dette er motivert av autonome kjøretøys potensiale til å både være ressursbesparende og skape tryggere kjøretøy. Denne masteroppgaven skal studere autonome biler med en tilkoblet henger. Hengeren legger til en ekstra ustabil frihetsgrad som gjør at styring av systemet blir mer utfordrende. For at kjøret-henger systemet skal kunne navigere i et ustrukturert miljø som en parkeringsplass, må en planleggings-algoritme brukes. Planleggingsalgoritmen må planlegge en bane som kan gjennomføres av kjøretøyet samtidig som banen også må hindre at kjøretøyet kolliderer med obstruksjoner. Dette er en utfordrende oppgave, fordi kjøretøyet har ikke-holonomiske begrensninger som setter grenser for hvilke baner som kan brukes. Det må også tas hensyn til at kjøretøyet ikke må miste kontroll på hengeren slik at hengeren knekkes over mot kjøretøyet.

Denne masteroppgaven presenterer en ny planleggingsmetode for å konstruere en kjørbar bane mellom to sett av tilstander av kjøretøy-henger systemet i et åpent miljø. Denne type planlegger kalles en lokal planlegger siden den ikke kan navigere rundt obstruksjoner. Den kjørbare banen er generert ved å på første steg approksimere banen med en "Dubins path". Deretter blir banen forbedret i to steg. Hvert seg simulerer kjøretøy-hengeren i lukket sløyfe ("closed-loop") rundt banen konstruert av det forrige steget. Resultatet er en kjørbar bane som et kjøretøy-henger system kan følge fra en tilstand til en annen.

En global planlegger basert på "Rapidly-exploring Random Tree Star" (RRT*) og Closed-Loop RRT (CL-RRT) blir også presentert. RRT* har fordelen fremfor andre RRT baserte planleggere at den er probabilistisk optimal, som betyr at løsningen den returnerer vil konvergere mot den optimale løsningen med tid. Den lokale planleggeren som er beskrevet ovenfor blir bruk til å finne snarveier i treet, slik at planleggeren får den optimale oppførselen til RRT*. Etter det forfatteren vet, har det ikke tidligere vært implementert RRT* for kjøretøy-henger systemer, og dette kan skyldes at det ikke har vært en tilgjengelig en egnet lokal planlegger.

Den globale planleggeren ble testet med Monte Carlo simuleringer i to miljøer. Resultatene viser at den globale planleggeren kan lage kjørbare og korte baner i et realistisk miljø som skal simulere en parkeringsplass. Det er også vist at den lokale planleggeren kan lage kjørbare baner mellom et bredt sett av tilstander. Konvergensraten til algoritmen er likevel for langsom til bruk for de fleste sanntidssystemer som krever en optimal løsning.

Acknowledgement

I would first like to thank my supervisor Professor Sebastien Gros at the Norwegian University of Science and Technology (NTNU), for all his support and excellent guidance. I am very thankful to Semcon, the company that has provided the topic for this thesis. It has been a great experience to collaborate with Semcon and its employees. I am very greatful to my supervisor at Semcon, Øystein Henriksen. Thank you for many encouraging discussions, with fantastic suggestions, guidance, and a friendly tone. Thank you for spending many evenings reading through my thesis and giving me feedback. I would also like to thank Tonny Mastad and Thomas Eriksen for being excellent and warm team managers during my internship at Semcon and for supporting me while writing this thesis. I am very grateful to Professor Halvor Schøyen at the University of South-East Norway (ISN), who provided valuable guidance through the last writing stages.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The last decade has brought considerable improvements to the field of autonomous mobile robots. Many companies are investing a lot of resources into developing autonomous vehicles [Schwarting, Alonso-Mora, and Rus 2018]. The interest in developing autonomous vehicles is due to the many benefits of autonomous cars and trucks. It is estimated that road safety will increase by as much as 90% if the existing car park were to be replaced with autonomous vehicles [Duleba et al. 2021]. Today we already see operating autonomous mobile robots in warehouses, where the robots gather and move inventory, in a human-free environment [Bolu and Korçak 2021]. These robots work faster and at a lower cost compare to manually controlled vehicles. In the future, it is possible that long-distance transport will be revolutionized by autonomous trucks, both due to their cost savings and improved safety [Duleba et al. 2021]. In addition, personal transport will potentially be greatly impacted by autonomous cars. Autonomous vehicles can enable shared mobility, improve driving comfort and improve road efficiency by communicating with other nearby autonomous vehicles [Bolu and Korçak 2021].

Many applications require a trailer to be attached to the vehicle. Such a system is called a 1-trailer system. These applications range from personal cars where a trailer can be attached occasionally for increased capacity to long-distance trucks with semi-trailers. The trailer adds another degree of freedom to the vehicle. The trailer also makes the system harder to control due to the unusable dynamics of the trailer while reversing. This significantly alters how the vehicle system can be operated as many drivers have experienced for themself.

This master thesis is written in cooperation with Semcon Norway, a company located in Kongsberg, Norway. Semcon has a long history in the automotive industry. Seeing the potential of specialized autonomous vehicles Semcon is developing autonomous vehicles for industrial use. These vehicles will be made in small volumes and specialized for each specific application. One of the use-cases is a vehicle pulling a trailer with cargo from a factory to a local seaport. The cargo is too heavy for most vehicles, and the trailer is therefore essential for the operation. Another advantage of using a trailer is that the trailer can be left by the port, and an empty trailer can be brought back to the factory. The ports are not public, and therefore few dynamic obstructions are present. However, the environment around the port is changing due to cargo being moved around. The autonomous vehicle-trailer system must therefore be able to adapt to the changing environment. The sister company of Semcon, Yeti Move, is developing autonomous snow trucks with trailers attached. The snow trucks are designed for clearing snow off of airport runways. While studying these use-cases, Semcon has identified that a path planner for a vehicle-trailer system is needed. Designing such a planner for industrial use is the motivation of this thesis. Since Semcon develops multiple systems, the goal is to make the planner

general enough that it can be employed on multiple different systems.

1.2 Problem Statement

This thesis will present an algorithm that will solve the following problem: **Find a feasible and collision free path from any pose to another for a general 1-trailer system in an unstructured environment.** It is assumed that the environment is fully known and static. This is reasonable for an industrial case, where the locations can be mapped both before and during the autonomous operations. It is also assumed that the vehicle-trailer system will be moving at low speed such that the system can be described by kinematics considerations. The planner must be able to utilize both forward and reverse motion when creating a path. The goal is that the algorithm should be probabilistic optimal, with respect to path length. This means that the length of the path should decrease with the running time of the algorithm. It is assumed that a shorter path will decrease the wear on the vehicle, increase the efficiency of the operation and make the autonomous system act predictable for humans, such that accidents can be avoided.

1.3 Outline

Section 2 gives the theoretical background for path planning of vehicle-trailer systems. This includes the theoretical concepts that are useful for path planning, such as nonholonomic constraints and metrics, and a larger section on path planning algorithms that previously have been developed for vehicle-trailer systems. The kinematic model of the general 1-trailer system is also derived in Section 2.

Section 3 will cover the algorithms developed in this thesis. From the kinematic model of the vehicle-trailer system, a novel exact local planner will be derived and presented in Section 3.1. Then another local planner that is based on the works of Ljungqvist, Axehill, and Helmersson 2016 is presented in Section 3.2. To save computational time, heuristics that estimates the length generated by the local planner is heavily used in this thesis. The construction of these heuristics are presented in Section 3.3. The implementation of the collision detection is presented in Section 3.4. Together, the sections of Section 3 are all put together in used in Section 3.5 where the CL-RRT* algorithm is described in detail.

In Section 4 the algorithm is tested on simulation of a vehicle-trailer system. The simulation setup is described in Section 4.1. In Section 4.2 and Section 4.3 the local planners are tested for computational time, accuracy and optimally. In Section 4.4 the CL-RRT* algorithm is tested in multiple scenarios using Monte Carlo simulations. The performance of CL-RRT is used as the performance baseline. Section 4.5 analyzes the CL-RRT* algorithm to identify bottlenecks.

The results are discussed in Section 5. This section highlights the weaknesses and strengths of the CL-RRT* algorithm of this thesis, and suggests further

work.

Section 6 sums up the thesis and gives a conclusion based on the results.

2 Background

2.1 The Terminology and Problem Formulation of Path Planning

Path planning is a vast field that still is in development. This thesis focuses on path planning for nonholonomic systems, particularly vehicle-trailer systems. This section will present the terminology that is frequently used for path planning. La Valle has made considerable contributions to the field of path planning, and especially his book [LaValle 2006] is commonly used as a source for terminology and mathematical background for path planning.

The path planning problem can be summed up as being the problem of finding a set of either continuous or discrete actions that will take a system from one initial state to some goal state while satisfying a set of constraints. For our purpose, the system is a mobile robot, and the constraints are set by the environment.

Borrowing the terminology from Ljungqvist 2020 and LaValle 2006, the robot for which we are doing path planning can be modeled as a nonlinear time-invariant system described by the following equations

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.1a}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{2.1b}$$

Where $\boldsymbol{x}$ is the n-dimensional state vector of the robot and $\boldsymbol{u}$ is the m-dimensional control input vector that defines the actions of the robot. The robot should navigate from an initial state $\boldsymbol{x}_0$ to a goal state $\boldsymbol{x}_{goal}$ in the real world. It must avoid obstacles while heading towards a goal state. The robot is likely to have constraints on both $\boldsymbol{x}$ and $\boldsymbol{u}$. The topological space of the possible states and inputs are respectively denoted $\mathbf{X}$ and $\mathbf{U}$, such that the system must satisfy $\boldsymbol{x}(t) \in \mathbf{X}$ and $\boldsymbol{u}(t) \in \mathbf{U}$. $\mathbf{X}$ and $\mathbf{U}$ can be any topological space and may not necessarily lay on the manifold of $\mathbb{R}^n$. This is an important point, especially for states such as orientation. The orientation of an object is not uniquely defined on $\mathbb{R}$ since we have that $\theta = \theta + k2\pi \ \forall \ k \in \mathbb{Z}$. Therefore we say that 2d orientations operate on the manifold $\mathbb{S}$ which wraps around itself such that every orientation is uniquely defined.

The set of all possible configurations $\mathbf{X}$ defines all the states in which the robot can be in an open world. However, the real world is full of obstructions, and we want the robot to navigate around these. These obstructions are represented by the closed set $\mathbf{X}_{obs}$ which are all the states where the robot is colliding with the obstructions. The space that the robot then is free to navigate through is called $\mathbf{X}_{free}$ and is simply defined as $\mathbf{X}_{free} = \mathbf{X} \setminus \mathbf{X}_{obs}$.

Summed up the path planning problem is to find a set of inputs $\boldsymbol{u}(t)$ to construct a path $\boldsymbol{x}(t)$ such that

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{2.2a}$$

$$\boldsymbol{x}(t_G) = \boldsymbol{x}_{goal} \tag{2.2b}$$

$$\boldsymbol{x}(t) \in \mathbf{X}_{free} \tag{2.2c}$$

$$\boldsymbol{u}(t) \in \mathbf{U} \tag{2.2d}$$

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.2e}$$

This definition only applies to the path planning problem where we are satisfied with any feasible path that takes the system from $\boldsymbol{x}_0$ to $\boldsymbol{x}_{goal}$. However, it is often desirable to find some path that optimizes some objective function J. The optimization problem can be formulated as

$$\underset{\boldsymbol{u}(t)}{\operatorname{argmin}} J = \int_{t_0}^{t_G} L(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \text{ subject to } 2.2 \tag{2.3}$$

As we can see from Eq. (2.3) that path planning for dynamic systems is similar to the problem formulated when designing Model Predictive Controller (MPC). The main difference is that where MPC often can use gradient descent or some derivative of gradient descent to solve the problem, path planning must often rely on other methods to navigate the nonlinear environment. The following chapters will present a few popular strategies for path planners for these nonlinear problems.

Example: Differential drive robot To make the definition above more concrete, we will present a simple example. The Differential driven robot is a popular vehicles for autonomous robots due to it simple design both mechanically and control wise. Assuming no slip on the wheels, Chitsaz et al. 2009 showed that this system can be modeled as

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(u_1 \cos(\theta) + u_2 \cos(\theta)) \\ \frac{1}{2}(u_1 \sin(\theta) + u_2 \sin(\theta)) \\ \frac{1}{d}(u_2 - u_1) \end{bmatrix} \tag{2.4a}$$

In this case, as is common for robots navigating on a plane, the states of the system is

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.5}$$

where x and y are the position of the center of the differential driven robot, and θ is the orientation. Notice that the topological space of $\mathbf{X} = \mathbb{R}^2 \times \mathbb{S}$ as is common with rigid body robots operating on a 2D plane. The input is the velocity of the two wheels.

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \tag{2.6}$$

The wheels has an upper limit to their velocity and we therefore have that $\mathbf{U} \subset \mathbb{R}^2$.

2.2 Nonholonomic Constraints

Before we define a nonholonomic constraint, we will first define a holonomic and nonholonomic system. The following definition is often used [LaValle 2006; Borisov and Mamaev 2005]

Holonomic A system on the from of Eq. (2.1) is said to holonomic if there exists a function $\boldsymbol{F}(\boldsymbol{x}) = 0$ such that Eq. (2.1) can be obtained by differentiating $\boldsymbol{F}(\boldsymbol{x})$ with respect to time.

Nonholonomic A system that is not holonomic is said to be nonholonomic

In practice, this means that a nonholonomic system cannot be solved analytically. An example of a holonomic system is the linear time-invariant system

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} \tag{2.7a}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{2.7b}$$

Where $\boldsymbol{A}$ is a n by n matrix and $\boldsymbol{x}$ is a n-dimensional state vector. The solution for this system can of course be found analytically.

An example of a nonholonomic system is a disk rolling on a surface given by

$$\dot{x} = -r\dot{\psi}\cos\theta \tag{2.8a}$$

$$\dot{y} = -r\dot{\psi}\sin\theta \tag{2.8b}$$

where x and y are the position of the contact point of the disc and the ground plane and ψ is the angle of the disk around its symmetrical axis and θ is the angle of the disc relative to the coordinate system. As shown in Golwala 2014 there cannot exist a function $F(x, y, \theta, \psi)$ such that Eq. (2.8) can be derived from $F(x, y, \theta, \psi)$ by time differentiation.

Holonomic constraints are constraints that can be expressed as a function of the systems states. This limits the system to operate on a sub-surface of the state space that the system is given in. In other words, holonomic constraints reduce the degrees of freedom of a system. For example, take a system of the point mass particle system moving in a gravitational field in $\mathbb{R}^2$

$$m\ddot{x} = R_x \tag{2.9a}$$

$$m\ddot{y} = R_y + gm \tag{2.9b}$$

subject to the constraint

$$x\dot{x} + y\dot{y} = 0 \tag{2.10}$$

Where x and y are the positions of the particle, R_x and R_y are the forces from the constraint, and m and g is the mass of the particle and gravitational acceleration respectively. In this case, the constraint can be derived from

$$x^2 + y^2 = r^2 \tag{2.11}$$

essentially constraining a system to be moving on a circle. We all know this system as a pendulum, and the state of this system can be explicitly be expressed by a single state θ, the angle of the particle with respect to the origin.

A nonholonomic constraint restricts the dynamics of the system, but does not decrease the degrees of freedom. This means that the constraints can only be expressed as a function of the derivatives of the system's states. Nonholonomic constraints are therefore sometimes referred to as differential constraints.

The no-slip car model is an example of a system with nonholonomic constrained. The state of the car can be described fully by the car's pose which is given by the $[x, y, \theta] \in \mathbb{R}^2 \times \mathbb{S}$. The system is said to be nonholonomic because the wheels restrict the vehicle to only move in the direction of the wheel's rotation.The car can obtain any pose, but the path to the pose is restricted.

For the case of path planning, it is important to determine whether or not the system is nonholonomic or not. In the case of the no-slip car model, a pose can be infinitesimally close to the car pose, without the car being able to reach the pose without driving a significant distance. As will be explained in Section 2.7 the metric that usually applies to $\mathbb{R}^2 \times \mathbb{S}$ does not give a good indication of the real distance that the car must travel.

2.3 Differential Flat Systems

A flat system with output $\boldsymbol{y} = [y_1(\boldsymbol{x}), ..., y_m(\boldsymbol{x})]$ has the following properties [Fliess et al. 1997]

- every system variable may be expressed as a function of the components of $\boldsymbol{y}$ and of a finite number of their time-derivatives.
- every component of $\boldsymbol{y}$ may be expressed as a function of the system variables and of a finite number of their time-derivatives.
- the number m of components of $\boldsymbol{y}$ is equal to the number of independent inputs

These properties are very useful for path planning for such systems. For a flat system, one can find a path from one state to another and connect them with a smooth path. Then the exact input that allows the system to follow the path can be found by combining a finite number of the outputs time derivative along the path. Many systems are deferentially flat. An example of a flat system is a car pulling trailers with each hitch connection directly above each the rear axle. In this case, the position of the rear axle of the last trailer can be used as a flat output [Rouchon, Michel Fliess, Lévine, et al. 1992]. If the hitch is offset

from the rear axle the system is no longer flat. The exception is for a vehicle pulling one single trailer, which is the system being studied in this thesis. A local planner based on the flat output will be described in Section 2.9

2.4 Vehicle-Trailer System Kinematics

This thesis presents a planning algorithm for a general 1-trailer system. It is called general since the hitch connection is not nessesarly directly above the middle rear wheel axle. An n-trailer system with no hitch offset is called a simple n-trailer system. In this thesis, when the context is clearly the general 1-trailer system, it will be referred to as the vehicle-trailer system.

The path planning algorithm of this thesis is intended for low-speed operations such as parking. It is, therefore, reasonable to assume that the wheels do not slip on the ground. We also assume that the ground is flat. These assumptions are known as the no-slip assumptions. We can therefore use a kinematic model of the vehicle-trailer system.

There are multiple intuitive ways to define the states of the system. This thesis defines the input of the system to be the steering angle of the front wheel δ, and the velocity of the vehicle rear wheel v_1. β is the relative angle between the trailer and the vehicle. v_1 is also a more intuitive velocity as most vehicles are driven from the vehicle, and not the trailer. No rate limit is put on the steering angle. This is done to simplify the problem and is common when modeling trailer systems [Svestka and Vleugels 1995; Fliess et al. 1997], however many others include dynamics of the steering angle [B. Li et al. 2019; Sekhavat et al. 1998; Michel Fliess et al. 1995]. The assumption is that a vehicle following the path found in real life only will deviate with a small distance from the path at the points where the control input is not continuous.

An overview of the geometry of the vehicle-trailer system is shown in Fig. 2.1.

The kinematics can be derived from the assumption of no slip. Since the wheels are not slipping, the velocities of the all the wheels must be in the direction of each wheel. This can be expressed as

$$\mathbf{v}_f = \begin{bmatrix} \mathrm{v_f}\ \cos\left(\delta + \theta_1\right) \\ \mathrm{v_f}\ \sin\left(\delta + \theta_1\right) \end{bmatrix} \tag{2.12a}$$

$$\mathbf{v}_1 = \begin{bmatrix} \mathrm{v_1}\ \cos\left(\theta_1\right) \\ \mathrm{v_1}\ \sin\left(\theta_1\right) \end{bmatrix} \tag{2.12b}$$

$$\mathbf{v}_2 = \begin{bmatrix} \mathrm{v_2}\ \cos\left(\theta_2\right) \\ \mathrm{v_2}\ \sin\left(\theta_2\right) \end{bmatrix} \tag{2.12c}$$

Where $\mathbf{v}_f$, $\mathbf{v}_1$, and $\mathbf{v}_2$ are the velocity vectors of the front, rear, and trailer wheels in inertial frame. v_f, v_1 and v_2 are the velocities at the front wheel and trailer wheel. θ_1 is the heading of the vehicle and θ_2 is the heading of the trailer.

.

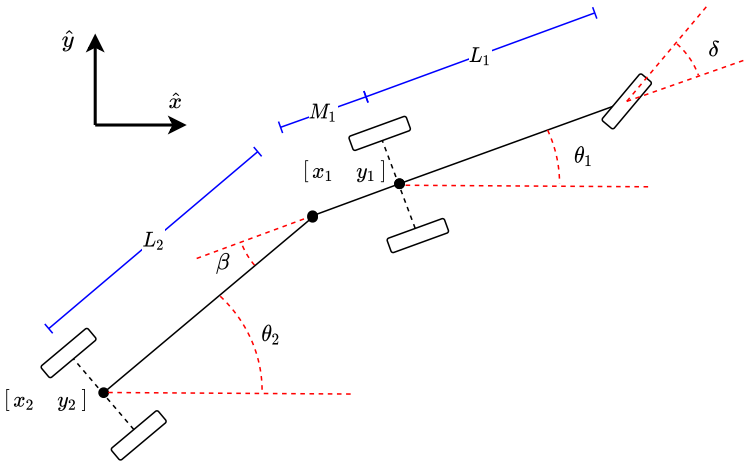


Figure 2.1: Diagram of the vehicle-trailer system. All angles are shown in positive direction.

The wheels are also constrained in their geometrical relation to each other and this can be expressed as

$$\boldsymbol{r}_f = \begin{bmatrix} x_2 + L_2\cos(\theta_2) + (M_1 + L_1)\cos(\theta_1) \\ y_2 + L_2\sin(\theta_2) + (M_1 + L_1)\sin(\theta_1) \end{bmatrix} \tag{2.13a}$$

$$\boldsymbol{r}_1 = \begin{bmatrix} x_2 + L_2\cos(\theta_2) + M_1\cos(\theta_1) \\ y_2 + L_2\sin(\theta_2) + M_1\sin(\theta_1) \end{bmatrix} \tag{2.13b}$$

$$\boldsymbol{r}_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \tag{2.13c}$$

Where $\boldsymbol{r}_f$, $\boldsymbol{r}_1$, and $\boldsymbol{r}_2$ are the position of the front, rear and trailer axles in inertial frame. $[x_2, y_2]$ are the position of the trailer axle, L_1 is the distance between the front wheel and the rear axle, and L_2 is the distance from the trailer hitch connection to the trailer axle. M_1 is the distance from the rear axle to the trailer hitch connection. On some vehicles, the hitch connection is in front of the axle of the rear wheel. In this case, M_1 would be negative, however, it will not change the calculations. The angle between the trailer and vehicle is denoted β and given by

$$\beta = \theta_2 - \theta_1 \tag{2.14}$$

Both constraints must be satisfied for the system, therefore we can take the time derivative of Eq. (2.13) and set it equal to the velocity constraints of Eq. (2.12).

$$\frac{d\boldsymbol{r}_f}{dt} = \mathbf{v}_f \tag{2.15a}$$

$$\frac{d\boldsymbol{v}_1}{dt} = \mathbf{v}_1 \tag{2.15b}$$

$$\frac{d\boldsymbol{v}_2}{dt} = \mathbf{v}_2 \tag{2.15c}$$

Finally we solve for the state vector $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u})$ arrive at the kinematics of the no-slip vehicle-trailer system

$$\dot{x}_2 = v_1 \cos(\theta_2) \frac{L_1 \cos\beta - M_1 \sin\beta \tan\delta}{L_1} \tag{2.16a}$$

$$\dot{y}_2 = v_1 \sin(\theta_2) \frac{L_1 \cos\beta - M_1 \sin\beta \tan\delta}{L_1} \tag{2.16b}$$

$$\dot{\theta}_2 = -v_1 \frac{L_1 \sin\beta + M_1 \cos\beta \tan\delta}{L_1 L_2} \tag{2.16c}$$

$$\dot{\beta} = -v_1 \frac{L_1 \sin\beta + (L_2 + M_1 \cos\beta) \tan\delta}{L_1 L_2} \tag{2.16d}$$

This also gives us the relation between v_1 and v_2

$$v_2 = v_1 \frac{L_1 \cos\beta - M_1 \sin\beta \tan\delta}{L_1} \tag{2.17}$$

Mark that Eq. (2.16) can be written as

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) = v_1(t) \bar{\boldsymbol{f}}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.18}$$

This gives rise to two important properties for the system which will be described underneath and are essential to the path planner developed in this thesis.

2.4.1 Time Independent Form

The kinematics of the vehicle-trailer system can be made independent of time. This is because we rewrite the system Eq. (2.16) on the form

$$\dot{\boldsymbol{x}}(t) = \frac{\partial \boldsymbol{x}(s_1)}{\partial s_1} \frac{\partial s_1(t)}{\partial t} = \frac{\partial \boldsymbol{x}(s_1)}{\partial s_1} v_1(t) \bar{v}_1 \tag{2.19}$$

Where s_1 is the arc length of the vehicle path and $\bar{v}_1 = sign(v_1(s_1))$ is the direction of travel. By inserting Eq. (2.19) for $\dot{\boldsymbol{x}}$ in Eq. (2.18), v_1 can be factored out and we are left with

$$\frac{\partial \boldsymbol{x}(s_1)}{\partial s_1} = \bar{v}_1 \bar{\boldsymbol{f}}(\boldsymbol{x}(s_1), \boldsymbol{u}(s_1)) \tag{2.20}$$

In practice, this means that any path traveled by the vehicle-trailer system can be followed by the same vehicle-trailer system at any velocity. This is useful since it allows path planning to be independent of velocity. The velocity of the real-world vehicle only needs to be low enough for the no-slip assumptions to hold.

2.4.2 Time Reversibility

Holmer 2016 showed that any system on the from Eq. (2.18) can be time-reversed by applying the inverse input. That is, we can achieve the states in reverse order by using the following input:

$$v_1(t) = -v_1(T - t) \tag{2.21}$$

$$\delta(t) = \delta(T - t) \tag{2.22}$$

This will give us the time-reversed states $\boldsymbol{x}^i$.

$$\boldsymbol{x}^i(t) = \boldsymbol{x}(T - t),\ \forall t \in [t_0, T] \tag{2.23}$$

Where T is the time at which the system is reversed from.

This is of course not only restricted to time as the variable, but applies to the system in the form of Eq. (2.20). The interpretation of this is that for any path that has been traversed in either forward or backward motion, one can follow the same path in the opposite direction by applying the same control input that was previously used on the same point on the path.

2.5 Exact Path Planning

Some path planning problems can be solved exactly and in finite time. These algorithms are said to be complete, that is they either will find a solution or return no solution if no solution exists. Completeness will be further discussed in the section on sampling-based algorithms. While these algorithms are not applicable to the motion planning problem for the vehicle-trailer system, an overview is presented here for the sake of giving a wide coverage of path planning algorithms and providing context for non-complete algorithms that will be presented next.

Many exact path planning algorithms do not take system dynamics into account. However, this is for many robotic systems, not an issue. The robotic system is of course constrained by the laws of nature, but the navigation system can be designed such that we do not have to take the dynamics into consideration while doing path planning. For example, a differential wheel-driven robot can follow any continuous path consisting of straight line segments by stopping at every sharp corner to change heading. Other examples are robots on omnidirectional wheels, and 6 DOF robotic arms [Kuffner and LaValle 2000]. Systems that are not in the real world at all, such as computer animations or computer characters

in video games are also free from dynamic constraints. We, therefore, see exact algorithms being used in popular game engines such as Unity 2020.

Exact algorithms require an analytic description of the obstacles. Many modern autonomous systems rely on SLAM [Bresson et al. 2017] to gain knowledge about the environment. There exist both real-time and post-processing techniques that will construct a high-level algebraic description of the environment from SLAM data such as in Yang et al. 2019. However, the field of analytic reconstruction is still in early development and most modern autonomous systems rely on point cloud or voxel obstruction maps as a representation of the environment [Cadena et al. 2016].

Most exact path planning algorithms find a path from a roadmap that is constructed from the environment. A roadmap is an undirected graph that can be searched through with a shortest path algorithm such as A* [Hart, Nilsson, and Raphael 1968]. For instance, the shortest path can be found for any environment defined by polygons. This is done by first constructing a road map as a visibility graph by connecting all vertices to all other vertices that can be reached with a straight line. The initial state $\boldsymbol{x}_0$ and the goal state $\boldsymbol{x}_{goal}$ are also included as vertices that are connected to the road map. Then a Dijkstra's algorithm is used to find the shortest path through the road map. An example of the shortest path from a visibility graph is shown in Fig. 2.2.

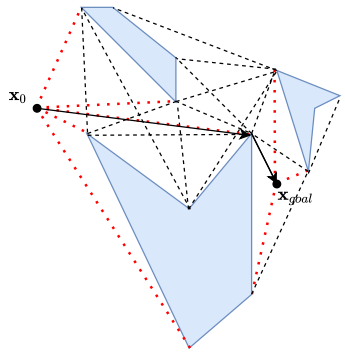


Figure 2.2: Visibility graph for a set of polygon obstacles and waypoints. The shortest path from $\boldsymbol{x}_0$ to $\boldsymbol{x}_{goal}$ can be found by doing a shortest path search on the resulting roadmap and is displayed with back arrows.

Another strategy is to divide $\mathbf{X}_{free}$ into convex regions and then search from the initial region containing the initial state outwards to the region containing the goal state. A version of this method called vertical cell decomposition was demonstrated in Abbadi et al. 2014.

There is much more to be said about exact algorithms, and chapter 6 in LaValle 2006 gives an excellent overview of these methods. The most important takeaway from this section, is however that these algorithms are not practical for real-world applications due to their poor computational performance, hard to

implement algorithms, and their need for analytical environments (LaValle 2006, Abbadi et al. 2014). These algorithms are also hard to apply to systems with nonholonomic constraints.

2.6 Sampling-Based Path Planning

As shown in the previous section there are multiple algorithms that are guaranteed to find a solution to a solvable path planning problem. However, the issues pointed out in the previous section also motivate us to look for other solutions. Sampling-based path planning algorithms have been very popular in robotic navigation literature [LaValle 2006; Ljungqvist 2020]. Sample-based algorithms simplify the planning problem by sampling down the continuous configuration space into discrete points. This has several advantages over the continuous approach seen in the last section. First, sampling-based algorithms are in general more efficient than complete algorithms [Kuffner and LaValle 2000]. Second, sampling-based algorithms often do not set any restrictions on the format of the obstructions in the environment. That is obstruction defined as analytic geometric shapes, point clouds, and occupancy grids can be used. This is because sampling-based algorithms only use virtual collision detection on the sampling point to navigate the obstructions. Complete algorithms rely on utilizing the geometric properties of the environment to guarantee a solution. This section will present three different families of sampling-based approaches, that are all relevant for the path planning problem for the vehicle-trailer systems. First, a lattice-graph search based algorithm will be described, then Rapidly Exploring Random Tree (RRT) and some relevant derivatives of RRT will be discussed and in the end hybrid A* will be presented.

First, it is useful to define the expected performance of planning algorithms from the notion of completeness. From LaValle 2006 and Jean-Paul Laumond, Sekhavat, and Lamiraux 1998 there are three levels of completeness. An algorithm is said to be

- **Complete** if for all input the algorithm in finite time resolves whether or not the problem is solvable.
- **Resolution Complete** if the algorithm finds a solution in finite time if one exists and if the sampling resolution of the grid is fine enough.
- **Probabilistic Complete** if the probability of finding a solution goes towards 1 as time goes to infinity, if a solution exists.

Completeness is important but we also often care how well the path performs on some metric. For instance, we may care about the length of the path, or the maximum curvature, and so on. This is referred to as the optimality of the algorithm for some cost function. The cost function is denoted $Cost(\boldsymbol{x}(t)_s)$ where $\boldsymbol{x}(t)_s$ is the solution path return by the algorithm. According to Ljungqvist 2020 and Karaman and Frazzoli 2011 an algorithm is said to be

- **Resolution Optimal** if the algorithm finds the optimal solution $\boldsymbol{x}(t)_s*$ in finite time if one exists in the sampling resolution of the algorithm.
- **Asymptotic Optimal** if the cost $Cost(\boldsymbol{x}(t)_s)$ approaches the optimal cost $Cost(\boldsymbol{x}(t)_s*)$ when time goes to infinity.

Optimality can be achieved using exact path planning algorithms as shown in the previous section, but only for a few selected problems such as maximum clearance and minimum distance LaValle 2006. For most problems with holonomic constraints, one can only hope to design an algorithm that is resolution optimal or probabilistic optimal. With these properties in mind, a few sampling-based methods that previously have been utilized for path planning for vehicle-trailer systems will be presented.

2.6.1 Lattice Graph Search

The lattice-graph based path planner was first introduced in Pancanti et al. 2004. The lattice graph is constructed by discretizing the state space of the path planning problem into a regular grid of nodes. Each node represents a distinct state of the system. The system can transition from one node to another with a set of precomputed feasible paths. Then A* is used to search through the lattice graph for the shortest path to the goal state that does not collide with the environment. Fig. 2.3 shows an illustration of a simple lattice graph for a car-like vehicle. In this illustration, the x and y directions are discretized into a regular grid and the heading can have four orientations. The motion primitives that define the node transitions are shown on the left in Fig. 2.3.

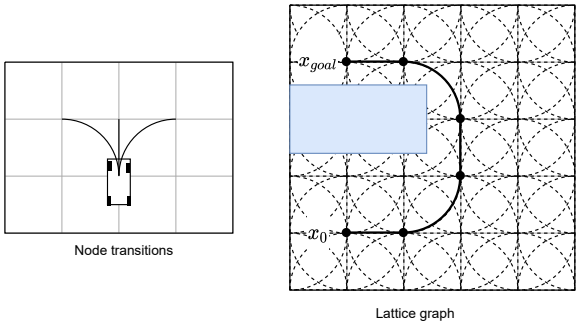


Figure 2.3: A simple lattice graph constructed from the motion primitives to the left. A path found from $\boldsymbol{x}_0$ to $\boldsymbol{x}_{goal}$ is highlighted on the lattice graph.

Ljungqvist demonstrates the use of a lattice graph search method for a general 2-trailer systems in Ljungqvist, Axehill, and Löfberg 2018; Ljungqvist, Evestedt, et al. 2019. The state space is discretized such that x, y has a resolution of $1m$. The orientation θ is discretized into 16 different angles and β is can take on

tree discrete angles. An Optimal Control Problem (OCP) was formalized and solved to connect nodes to a subset of the nearby nodes to create the transitions. Ljungqvist, Axehill, and Helmersson 2016 reports that the selection of which nodes to connect is a hard manual process, that greatly affects the performance of the algorithm. Selecting too many transitions will exponentially increase the running time of the algorithm, however too few limits the maneuverability of the system. The limited number of nodes also limits the maneuverability of the path found by the planner. An advantage to the lattice graph approach is that the algorithms are resolution optimal and resolution complete [Pivtoraiko, Knepper, and Kelly 2009].

2.6.2 Hybrid A*

A second sample-based category of path planning algorithms is Hybrid A*. Dolgov et al. 2008 demonstrates the use of Hybrid A* for a car navigating in complex environments at the 2007 DARPA Urban Challenge.

There are multiple implementations of Hybrid A*, but we will present the algorithm presented by Dolgov et al. 2008 as an introduction. Hybrid A* discretizes the configuration space into a regular grid in x and y direction as well for the heading for every position. Similar to RRT Hybrid A* builds a tree structure of poses. Each pose has a cost associated with it. The tree expands by selecting the node in the tree with the lowest cost. The node expands by a predefined set of motion primitives. The motion primitives are chosen such that they guarantee that they will bring the vehicle into a diffident grid cell than that of the selected node. Dolgov et al. 2008 uses a constant steering angle to construct the motion primitive. An illustration of the tree expansion is shown in Fig. 2.4. Whenever a grid cell is reached it is marked as visited. If a motion primitive causes a collision or brings the vehicle into an already visited grid cell, the tree is not expanded with this primitive. This alone will create a viable path planner, however, the search is in every direction. It is therefore common to add several biases in the form of potential fields.

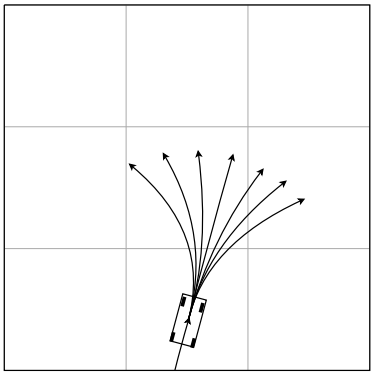


Figure 2.4: Illustration of how Hybrid A* expands the motion tree

Dolgov et al. 2008 thee potential fields are used to increase to the performance of Hybrid A*. The first is a nonholonomic distance heuristic, that estimates the distance to the goal pose for a car-like vehicle if no obstructions are present. This distance cab be found by using Reeds-Shepp paths [Reeds and Shepp 1990]. This distance will almost always be longer then the Euclidean distance. The second potential field uses the A* on the obstruction grid to calculate the distance from every cell to the goal pose as a walk between cell ignoring the nonholonomic constraints of the vehicle. This eliminates the exploration of dead-ends and guides the vehicle towards the goal pose. The third potential field is a Voronoi field. The Voronoi field is calculated by first creating a Voronoi diagram which gives the distance to the closest obstruction for every grid cell. The Voronoi field is then calculated from the Voronoi diagram such that the middle between two obstructions always has a cost of zero, and maximum cost at the edge of an obstruction. The Voronoi field ensures that the vehicle keeps a distance from obstructions, without rendering narrow corridors nontraversable.

Due to the discrete nature of the steering input, the paths generated by Hybrid A* are sub-optimal and often oscillate right and left. Dolgov et al. 2008 solves this by solving a non-linear optimization problem, minimizing the path length as well as avoiding obstructions. The result is a smooth path that Dolgov et al. 2008 reports often is close to the global optimal solution. The reported performance is good; the running time for the whole algorithm was 50–300ms.

B. Li et al. 2019 demonstrates that Hybrid A* algorithm can be used for a 3-trailer system. The paper uses an optimization step on the path found by Hybrid A* to decrease the length and curvature. The reported running time of the algorithm was on the order of 1-minute, making it not suitable for many real-time applications.

Nordeus 2015 published an open-source code along with a nonacademic blog post explaining Hybrid A*. The open-source code contains a video game-like environment where a Hybrid A* path planner is demonstrated. Interestingly, the planner runs within a second on the same test computer as this thesis's results were generated. Nordeus 2015 demonstrates that the method is also able to return a feasible path for a 1-trailer system. The video game demonstration is free to download and easy to set up on a windows computer.

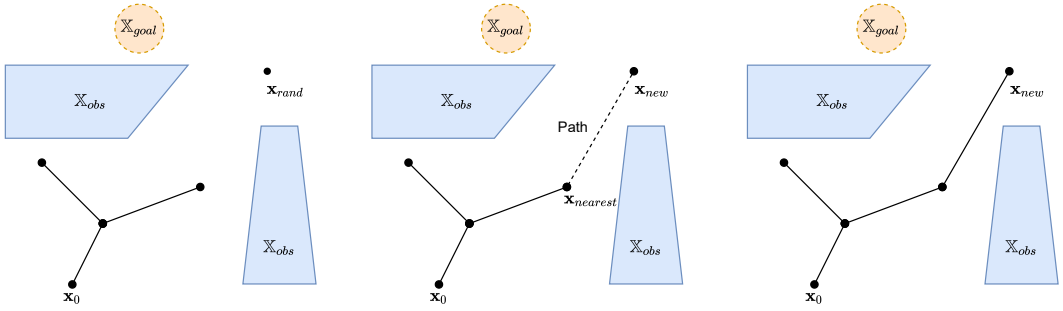
2.6.3 Rapidly Exploring Random Trees

Rapidly Exploring Random Trees (RRT) are a group of path planning algorithms that have become quite popular for mobile robotics.

RRT uses a tree structure of nodes. Each node in the tree represents a state. Each node has one parent node, and as many children nodes as necessary. If two nodes are connected together that means that there is a collision-free straight path between them. Since RRT places nodes randomly, it is not possible to reach a goal pose $\boldsymbol{x}_{goal}$, and one must therefore define a goal region $\boldsymbol{X}_{goal}$ in a ball around $\boldsymbol{x}_{goal}$.

Algorithm 1: Basic RRT

```
1 Tree.Init(x_0)
2 while Tree does not contain node in X_goal do
3     x_rand ← RandomState()
4     x_nearest ← Tree.Nearest(x_rand)
5     x_new, P ← Steer(x_nearest, x_rand)
6     if CollisionFree(P) then
7         Tree.AddNode(x_new, x_nearest)
```



(a) **(b)** **(c)**

Figure 2.5: Figure (a) to (c) shows the one iteration of a successful expansion of a random tree. (a) A state $\boldsymbol{x}_{rand}$ is selected at random from the configuration space. (b) A path from the closest node in the tree $\boldsymbol{x}_{nearest}$ to $\boldsymbol{x}_{rand}$ is constructed. The end of the path is located at $\boldsymbol{x}_{rand}$ and named $\boldsymbol{x}_{new}$. (c) Since the path does not collide, that is it does not enter $\mathbf{X}_{obs}$, $\boldsymbol{x}_{new}$ is added to the tree.

To construct the tree $\boldsymbol{x}_0$ is initialized as the root of the tree. While the tree structure does not contain a node within the goal region the algorithm will loop through the following steps. A random state $\boldsymbol{x}_{rand}$ in the input space is sampled. Then the closest state in the tree to $\boldsymbol{x}_{rand}$ is found and named $\boldsymbol{x}_{nearest}$. Then a steering function is used to construct a path $\mathbf{P}$ from $\boldsymbol{x}_{nearest}$ to $\boldsymbol{x}_{rand}$. The steering function is simply a function that constructs a path towards any state. For completeness with other algorithms in this thesis, the end of the path is named $\boldsymbol{x}_{new}$. If the path does not collide with the environment $\boldsymbol{x}_{new}$ is added to the tree as a node with $\boldsymbol{x}_{nearest}$ as the parent. The algorithm for a simple implementation of RRT is shown in Algorithm 1. Fig. 2.5 shows an example of a successfully expansion of a tree.

RRT is probabilistic complete for holonomic systems [LaValle 2006]. RRT is known to perform better than lattice graph algorithms for problems with high

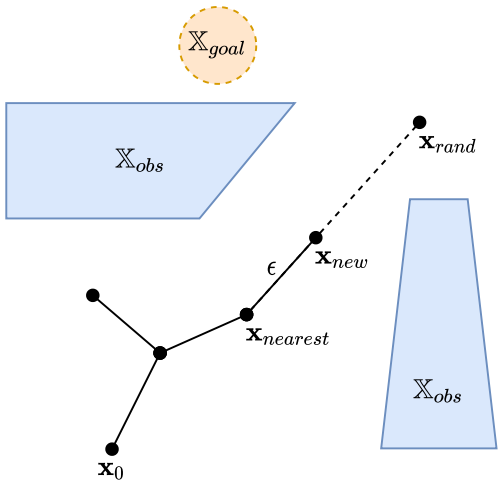
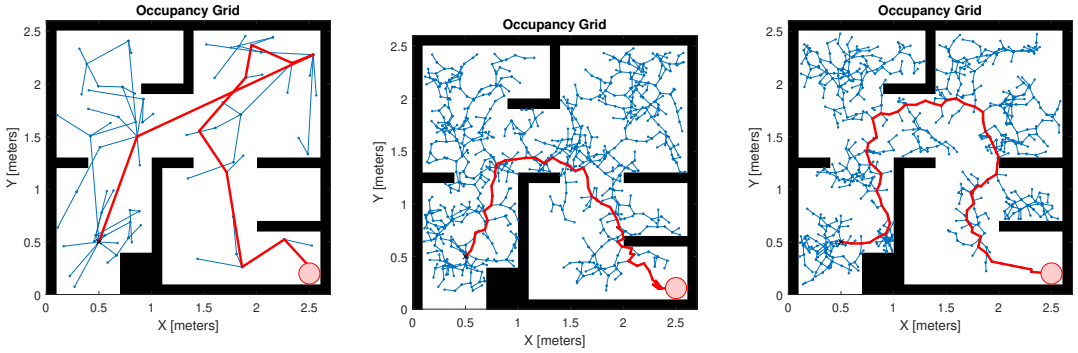


Figure 2.6: RRT with limitation of expansion length

dimensions [Noreen, Khan, Habib, et al. 2016] due to the exponential growth experienced by lattice graph-based methods for higher dimensions. A big drawback of RRT is that the solutions are almost always sub-optimal, and there is no mechanism in the algorithm for improving the path. Running RRT over a longer period of time will enable more nodes to reach the goal region and slightly decrease the length from the shortest path. However, the RRT will quickly reach a lower bound on the path length which usually is significantly higher than the shortest possible path.

There is a huge number of variations of RRT that change its behavior. One of the most common modifications is to limit the expansion length as described in Brandt 2006. If the the distance between $\boldsymbol{x}_{rand}$ and $\boldsymbol{x}_{nearest}$ is higher then some expansion threshold ϵ the steering function will place $\boldsymbol{x}_{new}$ at a distance ϵ in direction of $\boldsymbol{x}_{rand}$ from $\boldsymbol{x}_{nearest}$. The successfully limited expansion is shown in Fig. 2.6. Limiting the expansion distance does two things, first, it limits overshoot where the tree expands too far in a direction as seen in Fig. 2.7a. Second, since long paths have a higher probability of crossing an obstruction, more random samples would have to be made before a new collision-free path to a new node could be made. Another alternative is a greedy method where the steer function always will return a collision-free path up to the nearest obstructions. Kuffner and LaValle 2000 argues that this can be more efficient since every call to the nearest neighbor search will result in a new node being added to the tree.

Basic RRT expands in all directions and will therefore not only find a path to



(a) Basic RRT with no limit on expansion distance

(b) Basic RRT with $\epsilon = 0.1$

(c) Basic RRT with $\epsilon = 0.1$ and 0.1 chance of setting $\boldsymbol{x}_{rand} = \boldsymbol{x}_{goal}$

Figure 2.7: Three implementations of RRT. The red circle is the goal region $\mathbf{X}_{goal}$, $\mathbf{X}_{obs}$ is the black regions, the blue graph is the full graph of the RRT, the red path is the path found by the RRT from $\boldsymbol{x}_0$ to the goal region $\mathbf{X}_{goal}$.

the goal region but also to all other reachable regions. To increase performance it is common to add some bias to the expansion of the tree. For example can we can set $\boldsymbol{x}_{rand} = \boldsymbol{x}_{goal}$ for some fraction of the expansion iterations. LaValle 2006 warns that biasing can also be dangerous. Adding bias to the algorithm will sometimes result in worse performance if not done carefully.

Fig. 2.7 shows three implementation of RRT with the some of the modifications discussed above.

The non-optimal behavior of basic RRT makes it unsuitable for many applications. RRT* which was introduced by Karaman and Frazzoli 2011 has therefore become very popular [Noreen, Khan, Habib, et al. 2016] as it will return a path that converges to an optimal solution with time. An implementation of RRT* is shown in Algorithm 2.

RRT* works by first expanding the tree in the same manner as basic RRT. However, every node has a cost $Cost(\boldsymbol{x})$ associated with it that gives the cost of getting from the root node $\boldsymbol{x}_0$ to the node. Whenever a path $\mathbf{P}_{nearest}$ has successfully been connected to $\boldsymbol{x}_{new}$ from the closest node in the tree $\boldsymbol{x}_{nearest}$, $\boldsymbol{x}_{new}$ is not immediately added to the tree as in basic RRT. An effort is first made to connect the node to a set of nearby nodes $\mathbf{X}_{near}$ that are within a distance η such that the cost of getting from the root node to $\boldsymbol{x}_{new}$ is as small as possible. After a node is added to the tree, the tree is rewired. This is done by constructing paths to $\mathbf{X}_{near}$ and checking if a route from $\boldsymbol{x}_{new}$ will decrease the cost of getting from the root node to the nearby node $\boldsymbol{x}_{near}$. If that is the case $\boldsymbol{x}_{new}$ is set as the parent of the $\boldsymbol{x}_{near}$. The rewiring step is essentially to look for what new shortcuts $\boldsymbol{x}_{new}$ enabled when it was added to the tree.

Algorithm 2: RRT*

```
1  Tree.Init(x_0)
2  while time < TimeLimit do
3      x_rand ← RandomState()
4      x_nearest ← Tree.Nearest(x_rand)
5      x_new, P_nearest ← Steer(x_nearest, x_rand)
6      if CollisionFree(P) then
7          X_near ← Tree.Near(x_new, η )
8          x_min ← x_nearest
9          c_min ← Cost(x_nearest) + Cost(P_nearest)
           // Connect to the node with minimal cost
10         for x_near,c ∈ X_near do
11             P_near,c ← Steer(x_near,c, x_new)
12             if CollisionFree(P_near,c) and
                Cost(x_near,c) + Cost(P_near,c) < c_min then
13                 x_min ← x_near,c
14                 c_min ← Cost(x_near,c) + Cost(P_near,c)
15         Tree.AddNodeWithParent(x_new, x_min)
           // Rewire the tree
16         for x_near,r ∈ X_near do
17             P_near,r ← Steer(x_near,r, x_new)
18             if CollisionFree(P_near,r) and
                Cost(x_new,c) + Cost(P_near,r) < Cost(x_near,r) then
19                 Tree.SetParent(x_near,r, x_new)
```

The threshold η could be changed dynamically to increase performance, but should not be scaled faster than a factor of $\log(N)/N$ where N is the total number of nodes. An alternative to finding the nearby nodes by distance is to select $\mathbf{X}_{near}$ as the k-nearest neighbors. k can also be scaled dynamically, however it should not be decreased below a constant threshold $k_{RRT*} = 2^{(n+1)}e^{(1+1/n)}$ [Karaman and Frazzoli 2011].

The RRT algorithms presented above, all assume that the mobile robot can follow lines that are C^0 smooth. That is the path is continuous, but its derivative is not. For many systems, this may not be an issue, as discussed above. For non-holonomic systems such as a vehicle-trailer system, this may lead to paths that are impossible to follow. Closed-loop RRT (CL-RRT) was introduced in Kuwata et al. 2009. CL-RRT utilizes a controller to stabilize the dynamics of the robotics system. When a random sample is drawn to expand the tree, a reference from the closest nodes is formed. A Steer function then generates a path towards the random sample by simulating the closed-looped robotic system with the reference as an input. This ensures RRT can be used with

systems with unstable dynamics, such as a vehicle-trailer system in reverse motion. Ljungqvist implemented a CL-RRT path planner for a two-trailer system in Evestedt, Ljungqvist, and Axehill 2016.

Karaman and Frazzoli 2013 present a framework for RRT* for systems with nonholonomic constraints. They successfully demonstrated the RRT* on a no-slip car. The results showed however that the rate of convergence was very slow and only increased the shortest path by about 5% after 100000 iterations.

Several smart methods for biasing RRT have been investigated. Theta* RRT is an RRT algorithm for nonholonomic systems which was introduced in Palmieri, Koenig, and Arras 2016. The method searches through the environment using Theta* prior to building the tree. Theta* is an algorithm similar to A* in that it finds the shortest way from one point to another in a regular grid [Nash et al. 2007], however, Theta* can return a path of any angle. The expansion of RRT is than limited to a ball around the path found by the Theta*. In large spaces, this significantly reduces the space for which the tree has to explore. However, the method it not suited for applications where for instance a u-turn is required, since Theta* does not take the nonholonomic constraints into consideration. In Ichter, Harrison, and Pavone 2018 a method for generating bias for nonholonomic sampling-based planners is presented. Ichter, Harrison, and Pavone 2018 using a learning-based method that finds good sampling points for a nonholonomic system. A very fast deep-learning-based method is presented in Y. Li et al. 2018 called Neural-RRT. This method is more suited for holonomic systems since the training dataset is based on A*. However, the paper shows the potential of utilizing neural networks with RRT.

2.7 Metrics and Nearest Neighbor Search

For RRT it is essential to measure the distance between two states. [LaValle 2006, Ljungqvist, Axehill, and Helmersson 2016]. When the RRT expands the tree towards a random state, the state in the tree that is the closest is selected to be expanded from. For the RRT algorithm to be probabilistic complete a metric must be used to measure the distance [LaValle 2006]. A metric ρ has 4 properties that must hold

- **Nonnegativity** $\rho(\boldsymbol{x}_a, \boldsymbol{x}_b) \geq 0$.
- **Reflexivity** $\rho(\boldsymbol{x}_a, \boldsymbol{x}_b) = 0$ IFF $\boldsymbol{x}_a = \boldsymbol{x}_b$.
- **Symmetry** $\rho(\boldsymbol{x}_a, \boldsymbol{x}_b) = \rho(\boldsymbol{x}_b, \boldsymbol{x}_a)$.
- **Triangle inequality** $\rho(\boldsymbol{x}_a, \boldsymbol{x}_b) + \rho(\boldsymbol{x}_b, \boldsymbol{x}_3) \geq \rho(\boldsymbol{x}_a, \boldsymbol{x}_3)$

The most familiar metric for most people is the euclidean distance defined as

$$\rho_e(\boldsymbol{x}_a, \boldsymbol{x}_b) = \sum_{i=1}^{n} (x_{a,i} - x_{b,i})^2 \tag{2.24}$$

This is a good way of measuring the distance in many situations, but falls short for many applications. For instance take the case of a robot described by a pose $\boldsymbol{x} = [x, y, \theta]$. If robot has the pose $\boldsymbol{x}_a = [0, 0, 0]$ and we want to measure the distance to $\boldsymbol{x}_b = [0, 0, 2\pi - 0.1]$, clearly the euclidean distance does not provide a satisfying answer. Therefore, there are numerous other useful metrics that can be used for distance mobile robotics. For instance, LaValle 2006 gives a metric for the distance between two points on $\mathbb{R}^k \times \mathbb{S}^l$ that fulfill all four properties of a metric. A point on the manifold is defined by $\boldsymbol{x} = [q_1, q_2, ..., q_k, \theta_1, \theta_2, ..., \theta_l]$ where $q_i \in \mathbb{R}$ and $\theta_i \in \mathbb{S}$. The metric between the points $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ is then given by

$$\rho_s(\boldsymbol{x}_a, \boldsymbol{x}_b) = \sqrt{\sum_{i=1}^{k}(q_{a,i} - q_{b,i})^2 + \sum_{i=1}^{l}(cos(\theta_{a,i}) - cos(\theta_{b,i}))^2 + (sin(\theta_{a,i}) - sin(\theta_{b,i}))^2} \tag{2.25}$$

In many cases, a metric does not provide a good estimation of path length between two poses. This is the case for nonholonomic systems, where two states can be as close as we want in metric space, but the system must travel a considerably longer distance in reality. For instance, a car has to drive 1 meter forward and then reverse to move 10 cm to the left. For these situations, it is often more useful to use a psuedometric [LaValle 2006; Noreen, Khan, Habib, et al. 2016]. This is a function that behaves somewhat like a distance, but does not satisfy all the criteria for a metric. An examples of the use of psudometrics can be seen in Kvarnfors 2019 where a Dubins path is used as an heuristic of the distance. Other exampled of psychometric are cubic Bezier curve which was used by Lee, Song, and Shim 2014 and Reeds-Shepp paths.

Often most of the computational time of an planning algorithm is used on finding the nearest neighbors using some distance measurement [LaValle 2006; Atramentov and LaValle 2002]. For metric spaces, a Kd-tree can be used to efficiently search the space for close nodes as shown in Atramentov and LaValle 2002. This dramatically reduces the computational time. For algorithms relying on psudometrics a Kd-tree cannot be used as the space cannot be used directly. Often the psudometrics are also very computationally expensive to compute which significantly limits the performance of algorithms [Noreen, Khan, Habib, et al. 2016]. It is therefore beneficial to compute the psudometric in advance and store it in a lookup table. This is done for path planning for vehicle-trailer systems in Kvarnfors 2019 and Evestedt, Ljungqvist, and Axehill 2016.

2.8 Dubins Path

A popular tool for path planning are the famous Dubins paths. Dubins path was introduced in Dubins 1957 and solves the following problem.

Assume a vehicle that moves in $\mathbb{R}^2 \times \mathbb{S}$ that can move along a continuous path

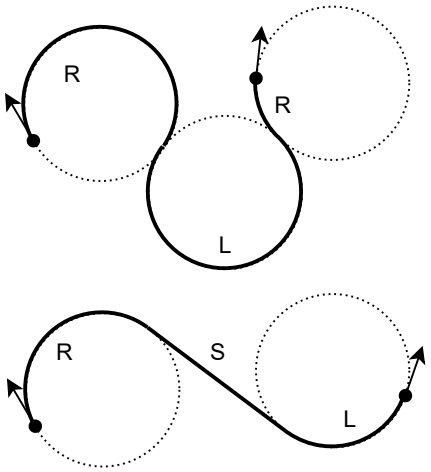


Figure 2.8: Two example of Dubins path between two poses. The top path is a RLR path and the one under is a RSL path

with a maximum curvature $\kappa_m = \frac{1}{R_m}$, where R_m is the minimum turning radius. Assume that vehicle can be controlled such that the curvature of the path it follows can change instantaneously. What is the shortest feasible path between any two poses?

Dubins proved that the shortest path always consists of two semi-circles of radius R_m joined by a line segment, or three semi-circles of radius R_m. There are 6 combinations of arches that can create the shortest path: (RSR, RSL, LSR, LSL, RLR, LRL), where R is a right turn, L is a left turn and S is a straight segment. Fig. 2.8 shows two examples of Dubins paths connecting two poses.

A known issue one often encounter when using Dubins path for path following is that a Dubins path does not have continuous curvature. For instance a car that is set to follow a Dubins path, would have to stop at the joint between two aches to change steering direction. This can be solved by using smoother paths such as Clothoids or Fermat's Spiral, which is widely used in the vehicle autonomy literature Lekkas 2014.

2.9 Flat Local Planner

As described previously the general 1-trailer system is differential flat for a selected output. This is easy to miss, as it is many times in the literature stated that the n-trailer system with no hitch offset is differently flat [LaValle 2006; Lamiraux and Laumond 2000; Ljungqvist, Evestedt, et al. 2019] without mentioning that 1-trailer system with hitch offset is also flat. Manav and Lazoglu 2021 which describes a path generation method for a general 1-trailer system fails to mention that an analytic method is available. Other such as B. Li et

al. 2019 wrongly states that no vehicle-trailer system with hitch offset is flat, however this is somewhat corrected further down in the paper.

Nevertheless, Fliess et al. 1997 and Rouchon, Michel Fliess, Lévine, et al. 1993 demonstrates how a local planner can be derived using the flat properties of the general 1-trailer system. The method is more complicated then for a N-trailer systems without hitch offset.

Fliess et al. 1997 proposes the following flat output $\boldsymbol{p}(\boldsymbol{x}) = [p_1(\boldsymbol{x}), p_2(\boldsymbol{x})]^T$ for a general vehicle-trailer system

$$p_1 = x_1 - L_2\cos(\theta_2) - L(\beta)\frac{L_2\sin\theta_2 + M_1\sin\theta_1}{\sqrt{M_1^2 + L_2^2 + 2M_1L_2\cos(\beta)}} \tag{2.26a}$$

$$p_2 = y_1 - L_2\sin(\theta_2) + L(\beta)\frac{L_2\cos\theta_2 + M_1\cos\theta_1}{\sqrt{M_1^2 + L_2^2 + 2M_1L_2\cos(\beta)}} \tag{2.26b}$$

where $L(\beta)$ is defined by the elliptical integral

$$L(\beta) = M_1L_2\int_0^\beta \frac{cos(\sigma)}{\sqrt{M_1^2 + L_2^2 + 2M_1L_2\cos(\sigma)}}d\sigma \tag{2.27}$$

Geometrical $\boldsymbol{p}$ is a point that are in vicinity of the vehicle-trailer system, but will change its relative location as a function of β. Note that for systems with no hitch offset, $M_1 = 0$ the path becomes $\boldsymbol{P} \equiv [x_2, y_2]$ which significantly simplifies the mathematics.

Fliess et al. 1997 demonstrated how this input can be used to generate a feasible path between any two poses $\boldsymbol{x}_{from}$ and $\boldsymbol{x}_{to}$. This is achieved by creating a sufficiently smooth curve $\boldsymbol{P}(s)$ between $\boldsymbol{p}(\boldsymbol{x}_{from})$ and $\boldsymbol{p}(\boldsymbol{x}_{to})$. $\boldsymbol{P}(s)$ must fulfill the constraints set by Eq. (2.26) and Eq. (2.16) at $\boldsymbol{P}(\boldsymbol{x}_{from})$ and $\boldsymbol{P}(\boldsymbol{x}_{to})$. These constraints can be found by taking the derivative of $\boldsymbol{P}$ with respect to arc length.

$$\frac{d\boldsymbol{P}}{ds} = \frac{\boldsymbol{r}_1 - \boldsymbol{r}_2}{\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|} \tag{2.28a}$$

$$\frac{d^2\boldsymbol{P}}{ds^2} = \kappa\boldsymbol{v} \tag{2.28b}$$

$$ds = J(\beta, \delta)ds_1 \tag{2.28c}$$

Where the position of rear wheel axle and the trailer axle is labeled $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ respectively. $\boldsymbol{v}$ is a unit vector orthogonal to $\frac{d\boldsymbol{P}}{ds}$ and κ is the curvature of $\boldsymbol{P}$ and is given by

$$\kappa = \frac{-\sin\beta}{\rho\cos\beta + L(\beta)\sin\beta} \tag{2.29}$$

$J(\beta, \delta)$ is defined by

$$J(\beta,\delta) = (L_2 + M_1\cos\beta - M_1^2/L_1\sin\beta\tan\delta)(\rho\cos\beta + \sin\beta L(\beta))/\rho^2 \quad (2.30a)$$

$$\rho = \sqrt{M_1^2 + L_2^2 + 2M_1L_2cos(\beta)} \quad (2.30b)$$

Fliess et al. 1997 states that a path $\boldsymbol{P}$ can easily be found with any regular parameterization such as a polynomial. Kvarnfors 2019 demonstrates the use of a polynomial with maximum curvature constraints as a path for a simple 1-trailer system.

κ is a one-to-one function from $[\gamma, 2\pi-\gamma]$ to $\mathbb{R}$. Where is defined by the function

$$\cos\gamma\sqrt{M_1^2 + L_2^2 - 2M_1L_2\cos\gamma} = 2M_1L_2\sin\gamma\int_\pi^\gamma \frac{cos(\sigma)}{\sqrt{M_1^2 + L_2^2 - 2M_1L_2\cos\sigma}}d \quad (2.31)$$

The state of the vehicle-trailer system can found directly by solving for $[x_2, y_2, \theta_2, \beta]$ in Eq. (2.26) and Eq. (2.28) for any point on $\boldsymbol{P}$ using numerical methods for inverting $\kappa(\beta)$.

The use of differential flatness in path planning is however limited. This might be due to reported issues when including constraints and adding performance measure om the path [Ljungqvist, Evestedt, et al. 2019]. The need to numerically invert $\kappa(\beta)$ and solve the integral $L(\beta)$ makes this methods relatively computationally expensive.

2.10 Optimal Local Planner

Another approach to exact local planning is to formulate the problem as a Optimal Control Problem(OCP). The problem can then be solved with nonlinear programming (NLP).

Holmer 2016 uses Direct Multiple Shooting [Bock and Plitt 1984] to minimize the path for a vehicle-trailer system. The optimal planner used a path found by a RRT to initialize the optimizer. B. Li et al. 2019 also used NLP to smooth the path found by a planner for a vehicle-trailer with an optimizer. In general it seems to us that the NLP is not a feasible method for generating feasible local paths for a RRT. We have not been able to find any examples of RRT that where implemented with Steer functions based on NLP. We therefore assume that using NLP for RRT is not efficient for real time applications due to the time consuming process of solving the OCP.

2.11 Collision Detection

Collision detection can be a complex matter, and a lot of methods exists that tries to optimize the collision detection processes. Collision detection is of course

essential for path planning as it is used to determine whether or not a path is feasible in the real wold. Many collision detection methods is based on objects constructed from polygons such as the methods described in Ericson 2004. Polygon defined collision tests are beneficial for applications where these polygons are easy to define, such as in computer games and in controlled environment such as a factory.

As mention previously, most modern mobile robots construct a point cloud of environment from a SLAM algorithm [Pan et al. 2013]. Constructing polygons from this point clouds is a very challenging task. It is therefore common in robotic application to use a regular grid to do collision detection on [Figueiredo et al. 2010]. Each rectangle in the grid has a value associated with it which represents weather the rectangle has an obstruction inside of it. Such a grid is called a occupancy grid. A point cloud generated by an SLAM algorithm can be converted into the occupancy gird by for instance using the techniques described in Meyer-Delius, Beinhofer, and Burgard 2012 and T. Collins, J. Collins, and Ryan 2007.

3 Method

This chapter presents the CL-RRT* path planner algorithm as implemented in this thesis. Two local path planners are presented in this section. One local planner is exact in that it creates a feasible path between two chosen states. This planner is implemented in the Connect function. The name Connect is chosen because the exact planner "Connects" two states exactly. The second local planner is not exact, but creates a path that approaches the desired state. The function using this planner is called Steer. This names is chosen because the second planner "steers" the vehicle-trailer system towards a state but not necessarily exactly to the state.

This chapter also describe how collision detection is implemented and how a the length of the path crated by the local planners are approximated with a heuristic.

Lastly this chapter presents the CL-RRT* algorithm that utilizes all the techniques described above.

3.1 Connect Function - Exact local path planning

By exact local planning, we mean an algorithm that solves the path planning problem of Eq. (2.2) by creating a path between two poses given that there are no obstructions. An exact local planner is necessary for implementing RRT* for nonholonomic systems. This is because RRT* rewires the tree, changing the parent of nodes. When a rewiring is performed one must be sure that the path from the new parent to the node goes exactly to the node. Else one cannot guarantee that the children of the node can also be reached. Finding such a method that is fast enough for path planning is often challenging. [Karaman and Frazzoli 2013]. We believe the method presented in this thesis is a novel method, and its use for path planning will be investigated. The method presented in this thesis generates paths between a start state $\boldsymbol{x}_{from}$ to a goal state $\boldsymbol{x}_{to}$. The path is not a perfect connection and can have a small deviation from the end states which will be investigated in Section 4.2. However, if the precision is within a few centimeters and we assume that it is good enough to enable the use of CL-RRT* for a vehicle-trailer system. Karaman and Frazzoli 2013 supports that an exact planner only needs to be exact to within a few centimeters such that the real world vehicle, can use the path as a reference. The alternative method for constructing exact local paths is to formulate the problem as an OCP and solve it with an optimizer as described in Section 2.10. This can be time-consuming and RRT* is therefore not been used for vehicle-trailer systems before Evestedt, Ljungqvist, and Axehill 2016. The other method is to use the deferentially flatness property of the vehicle-trailer system as described in Section 2.9. This technique still requires the selection of a smooth enough path that does not exceed the boundaries on . For every iteration of the flat planner, an integral must also be solved numerically. The method presented here, has very few computational processes for each iteration of the simulation, making

it an alternative to the flat method.

The algorithm for exact local path planning is divided into 3 stages and implemented in the Connect function. The three stages are illustrated in Fig. 3.1 where the paths for each stage are shown. An overview of the exact planner will be given here first, and then the details will be derived and explained in detail in the following pages.

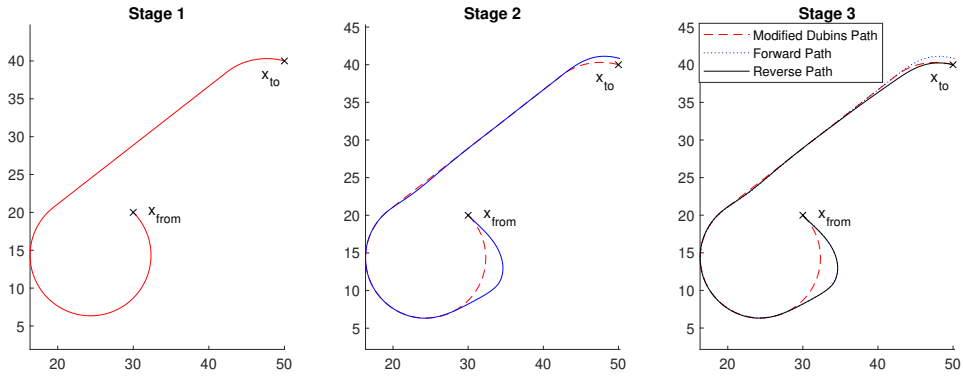


Figure 3.1: The three stages of the exact local planner. **Stage 1:** Modified Dubins Path Guidance. **Stage 2:** Forward Dubins Path following for feasible path generation. **Stage 3:** Reverse path following for exact path generation and connection. Notice how stage 2 misses $\boldsymbol{x}_{to}$ and how stage 3 is able to approach the path from stage 2

Stage 1 creates a Dubins path from the start state $\boldsymbol{x}_{from}$ to the end state $\boldsymbol{x}_{to}$. The Dubins path used in this thesis is slightly modified to achieve better performance of the local planner. A segment called the approach arch is added to the end of the Dubins path. The approach segment is curved such that a trailer in steady-state will have the desired β value at the endpoint.

At **Stage 2** a simulation of the vehicle trailer is performed. The Vehicle-trailer system uses an LQR controller to follow the modified Dubins path from stage 1. Since the vehicle-trailer system starts at $\boldsymbol{x}_{from}$ we are guaranteed that this $\boldsymbol{x}_{from}$ is connected with a feasible path to all other states of the simulated path. However, the end state $\boldsymbol{x}_{to}$ is not likely to lay exactly on the path. This is solved in stage 3.

Stage 3 takes advantage of the time reversibility property of Section 2.4.2 to achieve an exact local planner. The vehicle-trailer is simulated in reverse from $\boldsymbol{x}_{to}$ to $\boldsymbol{x}_{from}$ using the path generated by stage 2 as a reference and an LQR controller. Due to the time reversibility property, we can also use θ, β, and δ values from stage 2 as a reference and feed-forward to the controller. The vehicle-trailer which starts in $\boldsymbol{x}_{to}$ will quickly start to tightly track the path of

stage 2. Since the path of stage 2 is feasible the vehicle-trailer can easily follow the path to the $\boldsymbol{x}_{from}$.

Due to the time reversibility property this algorithm works for both forward and reverse motion. The only modification that is needed to adapt the algorithm to reverse motion is to start stage 2 from $\boldsymbol{x}_{to}$ and stage 3 from $\boldsymbol{x}_{to}$. A big advantage of this method is that the modified Dubins path of stage 1 can be used as an initial collision test for the path. This greatly reduces the number of simulations done by the global planner, which increases performance.

3.1.1 Modified Dubins Path

Due to the highly nonlinear nature of the exact local planner problem, a traditional control system is not sufficient to bring the vehicle to any pose. To solve this we use a modified version of the Dubins path. Dubins paths where covered in Section 2.8 and these paths can provide a guidance path for the vehicle as they create a continuous path from two general poses. The local planner must be able to connect all four states $[x_2, y_2, \theta_2, \beta]$ of the system. The Dubins path creates a continuous path for the tree first states, however, it does not take β into account. We observed during initial testing that after stage 2 of the local planner the vehicle-trailer system would not be sufficiently close to the desired $\boldsymbol{x}_{to}$. We solve this by adding a semicircle with a fixed length at the end of the Dubins path, with a radius R_a equal to the steady-state turning radius of β_{to}. An example of this path is shown in Fig. 3.2. The steady state curvature will be derived in the next section. This allows the vehicle-trailer system to have a few meters to settle down from the more chaotic maneuvering around the Dubins path before approaching $\boldsymbol{x}_{to}$ and should therefore be closer $\boldsymbol{x}_{to}$.

The Dubins paths are not feasible both due to the discontinuity in the curvature of the Dubins path. This is not an issue as stage 2 will generate a smooth and feasible path close to the modified Dubins path, and does not need to be perfect as stage 3 will do the final connection between $\boldsymbol{x}_{from}$ and $\boldsymbol{x}_{to}$

Other paths where considered for instance Continous Sharpness Turns as introduced in Oliveira et al. 2018 and demonstrated on a vehicle-trailer system in Kvarnfors 2019. Other paths of interest are Clothoids and Fermat's spirals. However, the modified Dubins paths has the advantage that there is only three different curvatures on the path, one for the straight segment, one for the Dubins path turning radius and one for the approach arch. This allows us to precompute the feed forward of the input which saves significant amounts of computing time.

3.1.2 LQR Path Follower

The path follower presented here is inspired by the works of Ljungqvist, Axehill, and Helmersson 2016. There are two main differences. First is that this thesis models and controls a 1-trailer vehicle while Ljungqvist, Axehill, and Helmersson 2016 models and controls a 2-trailer vehicle. The second is that

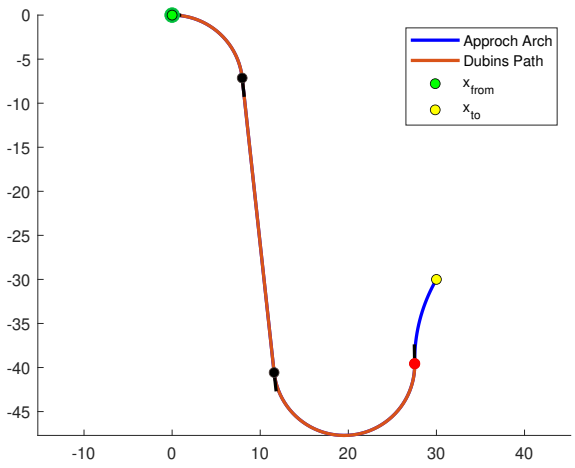


Figure 3.2: An example of the modified Dubins path with the extra segment shown in blue.

Ljungqvist, Axehill, and Helmersson 2016 uses the velocity of the last trailer as the input, while this thesis uses the velocity of the vehicle. This avoids some of the singularities described in Ljungqvist, Axehill, and Helmersson 2016.

The controller is designed around a model of the system in Frenet frame. The system of Eq. (2.16) can be described as a deviation from a nominal path given as a set of transverse coordinates. We assume that the nominal path is feasible. Since the nominal path $\boldsymbol{x}_n(s_{1,n})$ is feasible the kinematic constrains of Eq. (2.16) must also hold for the nominal path. The nominal path can be described by

$$\boldsymbol{x}_n(s_{1,n}) = \begin{bmatrix} x_{2,n}(s_{1,n}) \\ y_{2,n}(s_{1,n}) \\ \theta_{2,n}(s_{1,n}) \\ \beta_n(s_{1,n}) \end{bmatrix} \tag{3.1}$$

$$\frac{\partial \boldsymbol{x}_n(s_{1,n})}{\partial s_{1,n}} = \bar{v}_1 \bar{f}(\boldsymbol{x}_n(s_{1,n}), \boldsymbol{u}_n(s_{1,n})) \tag{3.2}$$

Where $s_{1,n}$ is the distance traveled by the vehicle on the nominal path. The vehicle-trailer system can then be modeled as a deviation from the path $\boldsymbol{x}_n$. For every state $\boldsymbol{x}(s_1)$ there are a state $\boldsymbol{x}_n(s_{1,n}(s_1))$ which is regarded as the closest. The Euclidean distance is used to determine the closest states, and the relationship between s_1 and $s_{1,n}$ is therefore defined as

$$s_{1,n}(s_1) := \arg\min_{s_{1,n}} \left\| \begin{bmatrix} x_1(s_1) \\ y_1(s_1) \end{bmatrix} - \begin{bmatrix} x_{1,n}(s_{1,n}) \\ y_{1,n}(s_{1,n}) \end{bmatrix} \right\|_2 \tag{3.3}$$

Given a s_1 the state of the vehicle-trailer system can be uniquely described in a ball around the nominal path by three transverse coordinates $[z_2(s_1), \tilde{\theta_2}(s_1), \tilde{\beta}(s_1)]$. $z_2(s_1)$ is the lateral deviation from the nominal path. The heading error of the trailer is defined as $\tilde{\theta_2}(s_1) := \theta_2(s_1) - \theta_{2,n}(s_1)$. The vehicle-trailer relative angle error is defined as $\tilde{\beta}(s_1) := \beta(s_1) - \beta_n(s_1)$.

Since δ always enters Eq. (2.16) as $\tan(\delta)$ we substitute $u = \tan(\delta)$. And around the nominal path we have $\tilde{u} = u - u_n$.

The relationship between the distance traveled by the vehicle relative to the distance traveled on the nominal path is given by

$$\frac{\partial s_{1,n}}{\partial s_1} = \frac{\bar{v}_1 \cos \tilde{\theta_1}(s_1)}{1 - \kappa_n(s_1) z_1(s_1)} \tag{3.4}$$

Where $\kappa_n(s_1)$ is the curvature of the nominal path and $z_1(s_1)$ is the lateral deviation of the vehicle. To simplify the equations we assume that $\kappa_n(s_1) z_1(s_1)$ is small so that we can write

$$\frac{\partial s_{1,n}}{\partial s_1} \approx \bar{v}_1 \cos(\tilde{\theta_1}) \tag{3.5}$$

The transverse dynamics of the trailer-vehicle system around the nominal path can then be derived by using the chain rule on Eq. (3.2) which gives

$$\frac{\partial z_2}{\partial s_1} = v_2 sin(\tilde{\theta}_2) \tag{3.6a}$$

$$\frac{\partial \tilde{\theta}_2}{\partial s_1} = -\bar{v_1} \frac{L_1 \sin(\beta_n + \tilde{\beta}) + M_1 \cos(\beta_n + \tilde{\beta})(u_n + u_t) - \cos \tilde{\theta}_2 (L_1 \sin \beta_n + M_1 u_n \cos \beta_n)}{L_1 L_2} \tag{3.6b}$$

$$\frac{\partial \tilde{\beta}}{\partial s_1} = -\bar{v_1} \frac{L_1 \sin(\beta_n + \tilde{\beta}) + (L_2 + M_1 \cos(\beta_n + \tilde{\beta}))(u_n + u_t) - \cos \tilde{\theta}_2 (L_1 \sin \beta_n + u_n (L_2 + M_1 \cos \beta_n))}{L_1 L_2} \tag{3.6c}$$

We are only interested in the transverse dynamics for the purpose of designing a stabilizing controller. The transverse states of the transverse dynamics are gathered into a vector $\tilde{\boldsymbol{p}}(s_1) = [z_2, \tilde{\theta}_2, \tilde{\beta}]^T$. And the transverse dynamics can be written as

$$\frac{\partial \tilde{\boldsymbol{p}}}{\partial s_1} = \tilde{f}(\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{u}}, \boldsymbol{p}_n, \boldsymbol{u}_n) \tag{3.7}$$

From Eq. (3.6) the system can be linearized around paths in steady-state. The linearized transverse dynamics has the form

$$\frac{\partial \hat{\tilde{\boldsymbol{p}}}}{\partial s_1} = \boldsymbol{A}(\boldsymbol{p}_n, \boldsymbol{u}_n) \hat{\tilde{\boldsymbol{p}}} + \boldsymbol{B}(\boldsymbol{p}_n, \boldsymbol{u}_n) \hat{\tilde{\boldsymbol{u}}} \tag{3.8}$$

Where

$$\boldsymbol{A}(\boldsymbol{p}_n, \boldsymbol{u}_n) = \left.\frac{\partial \tilde{f}(\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{u}}, \boldsymbol{p}_n, \boldsymbol{u}_n)}{\partial \tilde{\boldsymbol{p}}}\right|_{\tilde{p}=[0,0,0]^T} \tag{3.9a}$$

$$\boldsymbol{B}(\boldsymbol{p}_n, \boldsymbol{u}_n) = \left.\frac{\partial \tilde{f}(\tilde{\boldsymbol{p}}, \tilde{\boldsymbol{u}}, \boldsymbol{p}_n, \boldsymbol{u}_n)}{\partial \tilde{\boldsymbol{u}}}\right|_{\tilde{p}=[0,0,0]^T} \tag{3.9b}$$

We will consider two paths; a straight path and a circular path.

3.1.3 Linearization for Straight Paths

For a straight path that is in steady-state the nominal values $\boldsymbol{p}_n$ are trivial to find and given by

$$\beta_{n,s}(s_1) = n \tag{3.10a}$$

$$\delta_{n,s}(s_1) = 0 \tag{3.10b}$$

And $\theta_{2,n}$ does not enter into Eq. (3.6) and is therefore not needed in the calculations of the linearized transverse dynamics. Using Eq. (3.10) and Eq. (3.9) we can derive A_s and B_s matrices for straight paths.

$$\boldsymbol{A}_s = \begin{bmatrix} 0 & \bar{v}_1 & 0 \\ 0 & 0 & -\bar{v}_1/L_2 \\ 0 & 0 & -\bar{v}_1/L_2 \end{bmatrix} \tag{3.11a}$$

$$\boldsymbol{B}_s = \begin{bmatrix} 0 \\ -\bar{v}_1 \frac{M_1}{L_1 L_2} \\ -\bar{v}_1 \frac{L_2 + M_1}{L_1 L_2} \end{bmatrix} \tag{3.11b}$$

3.1.4 Linearization for Circular Path

For circular paths, the steady-state nominal state values can be derived from the geometric properties of the vehicle in steady-state. From Fig. 3.3 we can find a relationship between nominal vehicle-trailer angle $\beta_{n,c}$ and the turning radius of the trailer R_2

$$\tan\beta_{n,c} = \frac{L2 + sign(M_1)\sqrt{M_1^2 + L_y^2}}{R2} = \frac{L2 + M_1\sqrt{1 + \tan^2\beta_{n,c}}}{R2} \tag{3.12}$$

Solving for $\beta_{n,c}$ yields

$$\beta_{n,c}(R_2) = -sign(R_2)\tan^{-1}\left(\frac{L_2|R_2| + M_1\sqrt{L_2^2 - M_1^2 + R_2^2}}{M_1^2 - R_2^2}\right) \tag{3.13}$$

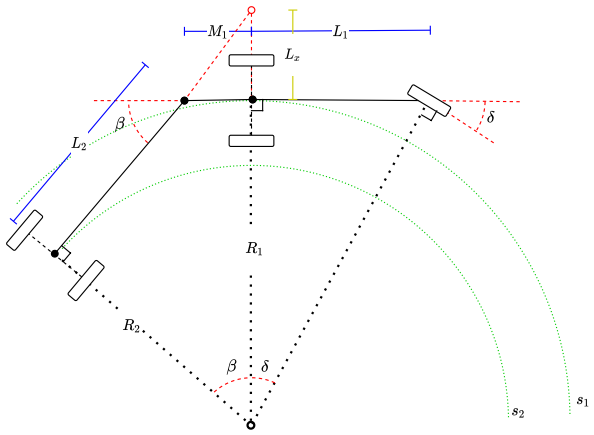


Figure 3.3: Illustration of the geometrical relationship of the vehicle-trailer system in steady state.

The vehicle-trailer is in steady state while following the path. This means that $\frac{\partial \beta_{n,c}}{\partial s_1} = 0$. $u_{n,c}$ can be found by solving for $u_{n,c} = \tan \delta_{n,c}$ in Eq. (2.16d) which yields

$$u_{n,c}(\beta_{n,c}) = -\frac{L_1 \sin \beta_{n,c}}{L_2 + M_1 \cos \beta_{n,c}} \tag{3.14}$$

The linearization around the nominal circular path can now be found by using Eq. (3.9), and inserting Eq. (3.13) and Eq. (3.14). This yields

$$\boldsymbol{A}_c = \begin{bmatrix} 0 & \bar{v}_1 \frac{L_1 \cos \beta_{n,c} - M_1 u_{n,c} \sin \beta_{n,c}}{L_1} & 0 \\ 0 & 0 & -\bar{v}_1 \frac{L_1 \cos \beta_{n,c} - M_1 u_{n,c} \sin \beta_{n,c}}{L_1 L_2} \\ 0 & 0 & -\bar{v}_1 \frac{L_1 \cos \beta_{n,c} - M_1 u_{n,c} \sin \beta_{n,c}}{L_1 L_2} \end{bmatrix} \tag{3.15a}$$

$$\boldsymbol{B}_c = \begin{bmatrix} 0 \\ -\bar{v}_1 \frac{M_1 \cos \beta_{n,c}}{L_1 L_2} \\ -\bar{v}_1 \frac{L_2 + M_1 \cos \beta_{n,c}}{L_1 L_2} \end{bmatrix} \tag{3.15b}$$

3.1.5 LQ gain

The LQR is a well-known and popular technique. Assuming that the system Eq. (3.8) is unconstrained the LQR technique can be used to find the optimal gain for the controller[Chen 1999]. In the case of path following we are minimizing the cost function

$$J = \int_0^\infty \hat{\tilde{\boldsymbol{p}}}^T \boldsymbol{Q} \hat{\tilde{\boldsymbol{p}}} + \hat{\tilde{\boldsymbol{u}}}^T \boldsymbol{R} \hat{\tilde{\boldsymbol{u}}} \, ds_1 \tag{3.16}$$

The optimal gain $\boldsymbol{K}$ can be calculated from

$$\boldsymbol{K} = \boldsymbol{R}^{-1} \boldsymbol{B}^T \boldsymbol{P} \tag{3.17}$$

Where $\boldsymbol{P}$ is found by solving the Riccati equation

$$\boldsymbol{A}^T \boldsymbol{P} + \boldsymbol{P} \boldsymbol{A} - \boldsymbol{P} \boldsymbol{B} \boldsymbol{R}^{-1} \boldsymbol{B}^T \boldsymbol{P} + \boldsymbol{Q} = 0 \tag{3.18}$$

Applying Eq. (3.11) and Eq. (3.15) to Eq. (3.17) and Eq. (3.18), we can find the optimal gain for straight paths $\boldsymbol{K}_s$ and circular paths $\boldsymbol{K}_c$.

3.1.6 Dubins Path Following Controller

With the results from the previous section we will derive two path following controllers. One controller for following Dubins paths, consisting of straight lines and semi-circles, and one controller for following a general feasible path. Note however that the Dubins path is not a feasible path, but can be seen as

a guidance path for the system. In the transition between different path types, the controller will change. It is at these transitions that it is impossible for the vehicle-trailer system to follow the path. A small deviation form the path is expected at these points, but if the deviation is not so large that vehicle-trailer system leaves the region where the linearization is valid, the system will asymptotically approach the path again.

If the curvature of the semi circles in known in advanced the optimal gain can be precalculated to save computational cost during tracking. While the rear end of the trailer is closest to a straight path the control input is calculated as

$$\tilde{\boldsymbol{p}}(s_1) = \begin{bmatrix} z_2(s_1) \\ \theta_2(s_1) - \pi_p \\ \beta(s_1) \end{bmatrix} \tag{3.19a}$$

$$u_s(s_1) = -\boldsymbol{K}_s \tilde{\boldsymbol{p}}(s_1) \tag{3.19b}$$

$$\delta_s(s_1) = \tan^{-1} u_s(s_1) \tag{3.19c}$$

Where π_p is the angle of the straight path relative to the inertial x-axis.

When the rear end of the trailer is closest to a circular path the steering input is calculated as

$$\tilde{\boldsymbol{p}}(s_1) = \begin{bmatrix} z_2(s_1) \\ \theta_2(s_1) - \theta_{2,n,c}(s_1) \\ \beta(s_1) - \beta_n \end{bmatrix} \tag{3.20a}$$

$$u_c = u_{n,c} - \boldsymbol{K}_c \tilde{\boldsymbol{p}}(s_1) \tag{3.20b}$$

$$\delta_c(s_1) = \tan^{-1} u_c(s_1) \tag{3.20c}$$

Where $\theta_{2,n,c}(s_1)$ is the angle of the tangent of the nominal path at s_1.

3.1.7 Feasible Path Following Controller

Given a nominal path that satisfy Eq. (3.2) another controller is used. Simulations show that the optimal gain only changes marginally as a function of R_2. We therefore use a constant value for the controller gain. This allows us to only compute K_s once while initializing the algorithm, saving significant amounts of computational time during simulation. The controller is realized as

$$\tilde{\boldsymbol{p}}(s_1) = \begin{bmatrix} z_2(s_1) \\ \theta_2(s_1) - \theta_{2,n,p}(s_1) \\ \beta(s_1) - \beta_{n,p} \end{bmatrix} \tag{3.21a}$$

$$u_p = u_{n,p} - \boldsymbol{K}_s \tilde{\boldsymbol{p}}(s_1) \tag{3.21b}$$

$$\delta_p(s_1) = \tan^{-1} u_p(s_1) \tag{3.21c}$$

Where $\theta_{2,n,p}$, $\beta_{n,p}$ and $u_{n,p}$ are the values used by the vehicle-trailer system which generated the feasible path at the closest point to the controlled vehicle-trailer system.

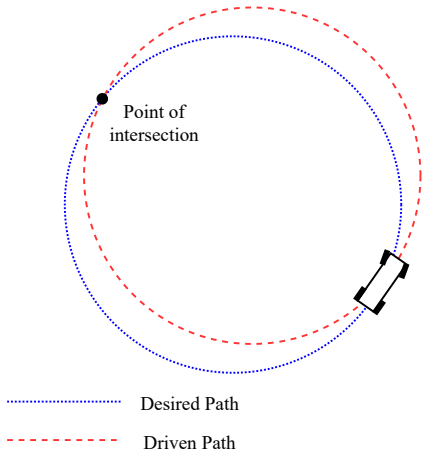


Figure 3.4: A vehicle following a path that was generated at δ_{lim} as input. Due to the disturbance the vehicle is not able to track the path before the intersection point

3.1.8 Constraints on β and δ

For most vehicles the steering angle δ is constrained within some range $\delta \in [-\delta_{lim}, \delta_{lim}]$. This limits the turning rate of the vehicle. If a reference path is constructed such that it saturates δ this can cause issues when following the path with a controller. Say a vehicle is following a path with the maximum curvature as shown in Fig. 3.4. Then a disturbance slightly pushes the vehicle such that the heading now point outwards from the circular arch. Since δ is fully saturated while following the path, the vehicle will drive a significant distance before the path circle and driving circle intersects.

Another consequence of the constraint on δ is that for most vehicle-trailer system there exists a minimum value of $|\beta| < \pi/2$ in reverse motion for which the vehicle-trailer system cannot avoid a jack-knifed configuration. This is the case for any vehicle where $L_2 > M_1$ [Hejase et al. 2018]. Almost all vehicles satisfy this condition.

Both of these issues can be solved by using a smaller range of δ such that $\delta \in [-\delta_{lim} + \epsilon, \delta_{lim} - \epsilon]$, where $\epsilon > 0$. This will allow the real world vehicle control system to have some margin around the nominal δ to stabilize the system around the generated path. The simulations that generates the path for the exact local planner can still result in a jack-knife configuration, however, these path will be discarded and not used by the RRT.

3.1.9 Connect Function

The exact local planned is implemented in the Connect Function. The function takes the start state $\boldsymbol{x}_{from}$, the end state $\boldsymbol{x}_{to}$ and an occupancy grid as inputs. A psudocode implementation of the function is shown in Algorithm 3. Starting at stage 1 where $\boldsymbol{x}_{from}$ to $\boldsymbol{x}_{to}$ is connected with a modified Dubins path. The path of stage 1 is then used to do a cheap collision check on the path to be followed. Stage 2 then simulates a vehicle-trailer system following the Dubins path in forward direction using the controller of Section 3.1.6. The control input and states from the simulations is saved and used as the nominal path $\boldsymbol{P}_{fwd}$ for stage 3. It is likely that the vehicle-trailer will miss the $\boldsymbol{x}_{to}$, however it is very likely that the system will be close to $\boldsymbol{x}_{to}$. During the stage 3 the feasible path controller of Section 3.1.7 is used to to follow the reverse path of stage 2, but this time the vehicle starts in the state $\boldsymbol{x}_{to}$. Since stage 2 ended the simulation close to $\boldsymbol{x}_{to}$, it is likely that the vehicle-trailer system starts within the region of attraction around the path created by the LQR controller of Section 3.1.7. The system will approach the feasible path created by stage 2 and follow it to $\boldsymbol{x}_{from}$. In this way we can cheaply construct an exact feasible path from $\boldsymbol{x}_{from}$ to $\boldsymbol{x}_{to}$. It is important that the vehicle at stage 2 does not use the full range of motion for δ as some buffer is needed for stage 3 to stabilize around the feasible path as described in Section 3.1.8. During stage 3 collision checks are done periodically on the full geometry of the vehicle on the occupancy grid. This will be descried in Section 3.4. If the connect function is able to simulate the vehicle-trailer along the path without collision, tree nodes for the RRT is distributed with a set interval on the path. These nodes are referred to as a branch since they together provide a non branching tree path from $\boldsymbol{x}_{from}$ to $\boldsymbol{x}_{to}$. Each node in the branch has a state associated with it as well as a path to the previous node.

Because of the time-reversible property of the system, a path for a reversing vehicle can easily be constructed by switching $\boldsymbol{x}_{from}$ and $\boldsymbol{x}_{to}$ as shown in Algorithm 3.

The construction of the modified Dubins path is somewhat expensive to calculate, but it only need to be done once per connection. Since a collision check can be done on this path, quite a few connections can be aborted early before the more expensive vehicle trailer simulation must performed. The functions used by control system is very simple and only requires simple addition and multiplications. The most computationally expensive part is the closest point calculations that finds the closes point from the trailer to the Dubins path as well as for the forward path.

3.2 Steer Function - Local Path Planning

Close Loop RRT requires a steering function to expand the tree. The steering function is a function that generates a path towards a general pose, but without guarantee for that it will actually hit the pose. The controller designed for the steering function in this thesis is based on the steering function described

Algorithm 3: Connect

Input: $\boldsymbol{x}_{from}$, $\boldsymbol{x}_{to}$, DriveDirection
Output: ConnectionMade, TreeBranch

if *DriveDirection = Forward* **then**
  $\boldsymbol{x}_{start} \leftarrow \boldsymbol{x}_{from}$, $\boldsymbol{x}_{end} \leftarrow \boldsymbol{x}_{to}$
else
  $\boldsymbol{x}_{start} \leftarrow \boldsymbol{x}_{to}$, $\boldsymbol{x}_{end} \leftarrow \boldsymbol{x}_{from}$
ModDubPath $\leftarrow$ ConstructModDubPath($\boldsymbol{x}_{start}$, $\boldsymbol{x}_{end}$)
if *Collision(ObstructionMap, ModDubPath)* **then**
  return *ConnectionMade* $\leftarrow$ *failed*
$\boldsymbol{x}_{fwd,0} \leftarrow \boldsymbol{x}_{start}$
$\boldsymbol{P}_{fwd} \leftarrow \boldsymbol{x}_{fwd,0}$
while *End of ModDubPath not reached* **do**
  $\boldsymbol{x}_{fwd,i+1} \leftarrow$ DubinPathControlStep($\boldsymbol{x}_{fwd,i}$, ModDubPath, $\bar{v}_1 = 1$)
  $\boldsymbol{P}_{fwd} \leftarrow [\boldsymbol{P}_{fwd}, \boldsymbol{x}_{fwd,i+1}]$
$\boldsymbol{x}_{rev,0} \leftarrow \boldsymbol{x}_{end}$
$\boldsymbol{P}_{rev} \leftarrow \boldsymbol{x}_{rev,0}$
while *Start of* $\boldsymbol{P}_{rev}$ *not reached* **do**
  $\boldsymbol{x}_{rev,i+1} \leftarrow$ FeasablePathControlStep($\boldsymbol{x}_{rev,i}$, $\boldsymbol{P}_{fwd}$, $\bar{v}_1 = -1$)
  if *Collision(ObstructionMap,* $\boldsymbol{x}_{rev,i}$*)* **then**
    return *ConnectionMade* $\leftarrow$ *failed*
  $\boldsymbol{P}_{rev} \leftarrow [\boldsymbol{P}_{fwd}, \boldsymbol{x}_{rev,i}]$
if *DriveDirection = Forward* **then**
  $\boldsymbol{P}_{rev} \leftarrow Flip(\boldsymbol{P}_{rev})$
TreeBranch $\leftarrow$ DistributeNodes($\boldsymbol{P}_{rev}$)
return *ConnectionMade* $\leftarrow$ *succsess, TreeBranch*

in Evestedt, Ljungqvist, and Axehill 2016. The controller of original paper is designed for a 2-trailer system, and therefore the some controller is not used in this thesis, but the same principles are used for the 1-trailer system in our thesis.

The steering control system has two stages, a pure-pursuit trailer heading reference generator, and a low level heading controller.

3.2.1 Heading Reference

Due to the nonlinear nature of moving from one pose to another for a nonholonomic system a guidance function is needed. The guidance law is similar to the enclousure based Line of Sight (LOS) guidance law as described in Chapter 12 of Fossen 2011. Given a target pose $\boldsymbol{q}_{to} = [x_p, y_p, \theta_p]$ a line segment can be constructed through the $[x_p, y_p]$ with a with an angle of θ_p relative to the x-axis. The pure-pursuit heading is defined as $\theta_e = \theta_{los} - \theta_2$. Where θ_{los} is defined

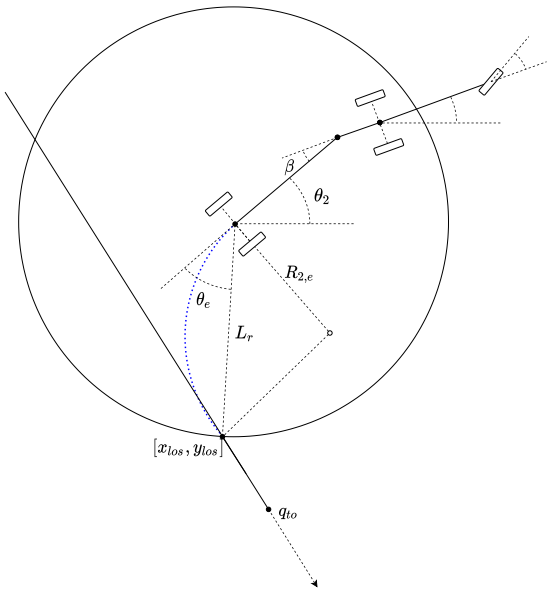


Figure 3.5: Diagram of the geometric relations for the pure-pursuit controller

as the angle of the a line through the trailer axis and the point of intersection $[x_{los}, y_{los}]$ between the target line and the line-of-sight circle of radius L_r around the trailer position. See Fig. 3.5 for a better visual explanation of the geometric relationships. If the line does not intersect, the closest point on the circle to the target line is used as the intersection point. This is shown in Fig. 3.6.

The point of intersection can be found by solving for $[x_{los}, y_{los}]$ in the equations

$$\tan(\theta_p) = \frac{y_p - y_{los}}{x_p - x_{los}} \tag{3.22a}$$

$$(x_{los} - x_2)^2 + (y_{los} - y_2)^2 = L_r^2 \tag{3.22b}$$

If Eq. (3.22) gives two real solutions, there will be two solutions for θ_{los}. In this case the θ_{los} that is closest to θ_p is choosen.

$$\theta_{los} = atan2(y_{los}, x_{los}) \tag{3.23}$$

If no real solutions to Eq. (3.22) can be found it means that the line is outside of the LOS circle. In this case θ_{los} can be found from

$$a = -sin(\theta_2)(x_2 - x_2) + cos(\theta_p)(y_2 - y_p); \tag{3.24a}$$

$$\theta_e = \theta_p - sign(a)\frac{\pi}{2} - \theta_2; \tag{3.24b}$$

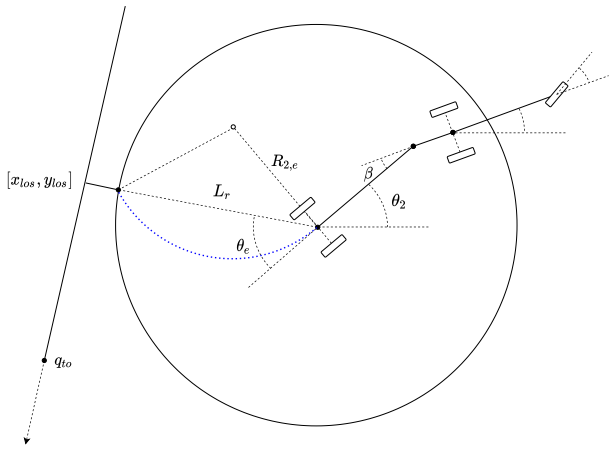


Figure 3.6: Diagram of the geometric relations for the pure-pursuit controller when the line though x_p is outside of the look-ahead circle.

3.2.2 Steering Control

A reference value for δ and β are created by using a circular arch that is tangential to the trailer at the trailer axis and intersects with the target line at $[x_{los}, y_{los}]$. This is shown as a blue arch in Fig. 3.5 and Fig. 3.6. The radius of this circular arc can be calculated from the geometrical relationship as a function of θ_e and is given by

$$R_{2,e}(\theta_e) = \frac{L_r}{2\sin\theta_e} \tag{3.25}$$

To approach this turning radius we use the steady state β-value for circular arcs as given in Eq. (3.13) as a reference. The steady state value for $\tan\delta$ as given in Eq. (3.14) is used as feed forward.

The steering angle is then set by

$$\delta = \delta_{n,c}(R_{2,e}) - K(\beta - \beta_{n,c}(R_{2,e})) \tag{3.26}$$

Where K is the controller gain. K can be fond experimentally. We found that a wide range of values worked for K.

3.2.3 Steer Function

When the RRT algorithm expands the tree, an exact local planner is not needed Kuwata et al. 2009. The steer function therefore constructs a path that will come close to $\boldsymbol{x}_{to}$ and run faster then the Connect Function. The psudocode for the Steer Function is presented in Algorithm 4.

Algorithm 4: Steer

Input: $\boldsymbol{x}_{from}$, $\boldsymbol{x}_{to}$, DriveDirection, ϵ
Output: TreeBranch

if *DriveDirection = Forward* **then**
  $\bar{v}_1 \leftarrow 1$
else
  $\bar{v}_1 \leftarrow -1$
$\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_{from}$
TreeBranch $\leftarrow \emptyset$
$\boldsymbol{P} \leftarrow \boldsymbol{x}_0$
PrevNodeIndex $\leftarrow 0$
while $\boldsymbol{x}_{i+1}$ *is not close to* $\boldsymbol{x}_{end}$ *and distance traveled* $< \epsilon$ **do**
  $\boldsymbol{x}_{i+1} \leftarrow$ LOSControlStep($\boldsymbol{x}_i$, $\bar{v}_1$)
  if *Collision(ObstructionMap,* $\boldsymbol{x}_{rev,i}$*)* **then**
    return *ConnectionMade* $\leftarrow$ *failed,* $\emptyset$
  $\boldsymbol{P} \leftarrow [\boldsymbol{P}, \boldsymbol{x}_{i+1}]$
  if *DistanceSinceLastNode > NodeInterval* **then**
    TreeBranch $\leftarrow$ [TreeBranch, Node($\boldsymbol{x}_{i+1}$, $\boldsymbol{P}$, PrevNodeIndex)]
    PrevNodeIndex $\leftarrow$ i
TreeBranch $\leftarrow$ [TreeBranch, Node($\boldsymbol{x}_{end}$, $\boldsymbol{P}$, PrevNodeIndex)]
return *ConnectionMade* $\leftarrow$ *succsess*

3.3 Distance Heuristic

As described in Section 2.7 it is important to have a good heuristic to be able to make good guesses on which nodes that will be able to connect to each other. In this context, we are interested in finding a heuristic that approximates how far away two states are from each other, given that they should be connected with a feasible path. The closed-loop vehicle trailer system is highly nonlinear, and euclidean distance would therefore not work as a heuristic of the distance to travel. The length of a modified Dubins path could be used as a heuristic. However, it is very costly to construct modified Dubins paths to every node to determine which one is the closest one. Motivated by Holmer 2016; Evestedt, Ljungqvist, and Axehill 2016 we, therefore, use two lookup tables with the distance from the origin to a grid of states around the origin as a heuristic. One lookup table is for the Steer Function, and one is for the Connect Function. The reason for having two lookup tables is that these functions produce quite different results, both in terms of distance to each grid node, but also in terms of which regions can be reached by the functions.

The each state of the grid is named $\boldsymbol{x}_{grid} = [x_{2,g}, y_{2,g}, \theta_{2,g}, \beta_g]^T$. $\boldsymbol{x}_{grid}$ is bounded such that $\boldsymbol{x}_{grid} \in [-R, R] \times [-R, R] \times [0, 2\pi] \times [-\pi/4, \pi/4]$. Where R is the maximum distance from the origin of the grid. $\boldsymbol{x}_{grid}$ are then spaced out evenly such that $\boldsymbol{x}_{grid}$ contains the origin as well as the boundaries. Since the

steering function cannot control the β-value, the lookup table does only hold values for $\beta = 0$. The Connect Function and Steering function are then used to create a path to every $\boldsymbol{x}_{grid}$ and the length of the path is saved in a lookup table. To get the heuristic from any $\boldsymbol{x}_{from}$ to any $\boldsymbol{x}_{to}$ one only needs to transform $\boldsymbol{x}_{to}$ such that the origin is set to $[x_{2,from}, y_{2,from}]^T$ and is align with $\theta_{2,from}$. If the $\boldsymbol{x}_{to}$ is to far away from $\boldsymbol{x}_{from}$, such that it is not covered by the lookup table, the closest point on the lookup table is used pluss the euclidian distance to that point. For point inside the grid, linear interpolation is used between the grid points to get an estimate for the travel distance.

Lookup heuristic for the connect function accepts paths that are within $0.5m$ of $\boldsymbol{x}_{grid}$ and the origin; all other are marked as having distance at infinity. This tells the Cl-RRT* algorithm not to bother with trying to connect two states that we have found unreachable with the connect function during the heuristic calculation phase. Precision is important for connections, and close connections will not be used. For the Steering function, precision is not as important, because the steering function should only explore in a general direction. The heuristic, therefore, has a quartic penalty on the distance from $\boldsymbol{x}_{grid}$ to the endpoint of the path generated with the steering function. This ensures that regions in the grid where the steering function generally performs well, are smooth, and have low cost. At the edges of the stable regions, the cost increases quickly, and the heuristic will therefore be able to determine that the vehicle should not try to steer in this direction. Visualizations of the heuristic lookup tables can be seen in the results in Section 4.2 and Section 4.3.

3.4 Implementation of Collision Detection

This thesis utilizes a regular occupancy grid for collision detection. This is done for several reasons. First, as mentioned in Section 2.11, a map produced by a SLAM algorithm can be converted to an occupancy grid without too much effort, and this is a common technique in the mobile robotics community. Second, it is easy to construct artificial environments for testing the system, using an image editor tool to draw a map. Third, the logic collision detection can be made very simple by also using an occupancy grid for the vehicle-trailer system. The occupancy grid for the vehicle-trailer system is constructed in the following way. For each rigid body of the vehicle-trailer system, a geometric collision boundary is created by combining rectangles such that the 2d cross-section of the rigid body is well represented. Each rigid body has d different occupancy grid, each representing a distinct angle for θ_1 and θ_2. For each angle, the rectangle representing the collision boundaries of the vehicle is rotated to the angle. Then the rectangles are digitized into an occupancy grid, for each cell marking it as occupied if one of the rectangles of the vehicle-trailer system is present inside.

Collision tests are performed by taking the occupancy grid of the vehicle and the trailer for the closest angles and translating them onto the occupancy map. If any of the grid cells with an obstruction from the environment and the vehicle-

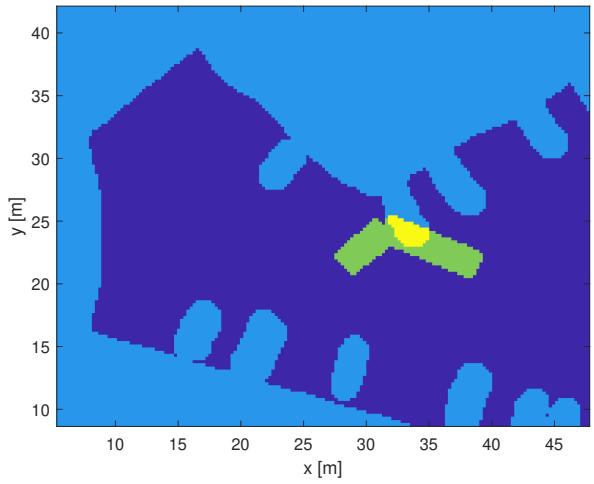


Figure 3.7: A visualization of the collision detection system. The open space is deep blue, the obstructed environment, here from a parking lot, is in light blue, the vehicle-trailer system is in green, and the area of intersection indicating a collision is in yellow. The resolution of the occupancy grids are $4px/m$.

trailer system overlap a collision is detected. Fig. 3.7 shows an example of the occupancy grid of the vehicle-trailer system overlayed on an occupancy grid of the environment.

This method has the advantage that any shape can easily be represented as long as its geometry is not more complicated than what can be represented by the occupancy grids' resolution. In this thesis the resolution of the occupancy grid are $res = 4\frac{px}{m}$ and the number of angles are $d = 64$.

This method is not precise due to the round error that will occur. The error of the occupancy grid representation of the vehicle-trailer system can be calculated. In this thesis, the trailer is the longest rigid body at $L_2 = 5.7m$. It rotates around the wheel axle, making the whole rigid body act like an arm. The maximum error due to rounding of the angles can be calculated as

$$E_{angle} = \sin\left(\frac{2\pi}{d}\right) L_2/2 = \sin\left(\frac{2\pi}{64}\right) 5.7m/2m = 0.28m \tag{3.27}$$

There will also be a rounding error from the position. This is because the environment occupancy grid and the vehicle-trailer system must be aligned. Maximum rounding error is calculated as

$$E_{pos} = \frac{1}{res}\frac{1}{2}px = \frac{1}{4}\frac{m}{px}\frac{1}{2}px = 0.125m \tag{3.28}$$

Padding around the vehicle-trailer occupancy grids is therefore required. In our

case, a padding of $0.5m$ is sufficient to guarantee no collision in real life.

3.5 Closed Loop Rapidly Exploring Random Tree Star

The algorithm developed in this thesis is inspired by the works of Evestedt, Ljungqvist, and Axehill 2016; Kvarnfors 2019. This thesis extends upon these techniques by merging them with RRT*. This is to decrease the length of the path generated by the path planning algorithm. The extension is only possible with the exact local planner that is fast enough for real-time applications as mentioned in Evestedt, Ljungqvist, and Axehill 2016. The exact local planner also enables more precise connections to the goal state than what is described in Evestedt, Ljungqvist, and Axehill 2016. This thesis also has added a smoothing step to the final path returned by the CL-RRT* this is to decrease non-intuitive and sub-optimal behavior as reported in Evestedt, Ljungqvist, and Axehill 2016.

3.5.1 Tree Structure

CL-RRT* build a tree of nodes and paths from the initial root pose $\boldsymbol{x}_0$ of the vehicle. Each node is referred to with a $\boldsymbol{x}$ which is intentional the same notation as for states. This is because each node is primarily is a state, but also has secondary properties associated with it. First, each node has a link to its parent and its children. Each node also has the length of the path from the root node. Reverse motion has double the weight of forward motion. This is to encourage the vehicle to drive forward. Lastly, the path from the parent node to itself is also stored with each node. Whenever a shortcut is found such that a parent of a node will be updated, the root distance of that node and all its children, will be updated recursively.

Whenever the Connect of Steer functions creates a path from $\boldsymbol{x}_{from}$ to $\boldsymbol{x}_to$, these states are connected with a series of nodes $\boldsymbol{P}_{new}$ with a maximum distance between them. This series of nodes is called a limb, to be consistent with the tree analogy. Since there cannot be any forks on the limb, we can define the length of the limb $length(\boldsymbol{P}_{new})$ to be the distance of the path that goes from $\boldsymbol{x}_{from}$ to $\boldsymbol{x}_to$ through all the nodes $\boldsymbol{P}_{new}$.

3.5.2 Algorithm Overview

The tree updates in 5 stages. A short description of the stages will be given first as an overview and then an in-dept description will be given underneath. Fig. 3.8 gives an illustrated overview of the flow of the algorithm.

Stage 1 finds a semi-random state $\boldsymbol{x}_{new}$ that is within the configuration space that the tree can easily expand to. Stage 2 looks at the nearby nodes of $\boldsymbol{x}_{new}$ to find a path to $\boldsymbol{x}_{new}$ such that $\boldsymbol{x}_{new}$ has as short path to the root node as possible. Stage 2 will then add a limb $\boldsymbol{P}_{new}$ from the tree to $\boldsymbol{x}_{new}$. Stage 3 looks at the new nodes $\boldsymbol{P}_{new}$ to see if they enable shortcuts in the tree to other nearby nodes. Stage 3 is the stage that is unique to RRT*. Stage 4 takes all

the nodes of $\boldsymbol{P}_{new}$ and checks if any of them can be connected to the goal state. After the maximum time has elapsed, the shortest path from the tree is made smoother by finding shortcuts within the path.

There are many implementation details that are needed for the algorithm to run efficiently. There are two processes that are consuming the most processing time, the nearest neighbor search and simulation of the closed-loop system. Therefore the implementation utilizes the heuristics for many applications in order to save computational time.

3.5.3 Stage 1: Random Exploration

First a random state $\boldsymbol{x}_{rand}$ is chosen from the open input space $\mathbb{X}_{free}$. To slightly improve performance, motivated by Informed RRT* Gammell, Srinivasa, and Barfoot 2014, $\boldsymbol{x}_{rand}$ is only sampled inside of an ellipsoid defined by $||\boldsymbol{x}_{rand} - \boldsymbol{x}_0||_2 + \boldsymbol{x}_{rand} - \boldsymbol{x}_{goal}||_2 < Cost(\boldsymbol{X}_{shortest})$, where $\boldsymbol{X}_{shortest}$ is the shortest path in the tree from $\boldsymbol{x}_0$ to $\boldsymbol{x}_{goal}$. This somewhat limits the space in which to select random states. Mark however that this limit is conservative because it do not take the nonholonomic constraints into consideration.

Then k_{steer} near nodes $\boldsymbol{X}_{near}$ that are estimated to have the shortest path length to $\boldsymbol{x}_{rand}$ is found in the tree using the steering distance heuristic from Section 3.3. The nodes in $\boldsymbol{X}_{near}$ are in sorted order. The Steer function from Section 3.2.3 is then used to steer from closest node $\boldsymbol{x}_{nearest}$ of $\boldsymbol{X}_{near}$ to $\boldsymbol{x}_{rand}$. If the steer function returns a collision free if will return a limb $\boldsymbol{x}_{new}$. If the Steer function does not return a collision free path, a new random pose is selected, and the process starts over again. Evestedt, Ljungqvist, and Axehill 2016 uses a greedy Steering function, while our implementation uses a ϵ limited Steering function. After some initial testing, we found the ϵ to be slightly better. Our intuition is that this is due to the greedy algorithm quickly increasing the size of the tree with nodes that are not that useful since they go directly into an obstruction. The ϵ approach will not expand the tree on every iteration, however, when the tree is expanded, it is likely to be in a more open area.

3.5.4 Stage 2: Semi-optimal Path from Root to New Node

At stage 2 the algorithm has a limb $\boldsymbol{P}_{new}$ extending from from tree towards $\boldsymbol{x}_{rand}$. At the end of this limb is the node $\boldsymbol{x}_{new,end}$. Now the algorithm will try to find a shorter path from $\boldsymbol{x}_0$ to $\boldsymbol{x}_{new,end}$. This is done by sorting $\boldsymbol{X}_{near}$ by each $\boldsymbol{x}_{near}$ length to the root and the length the steer heuristic estimates a path from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{new,end}$ will have. The steer function is used from each $\boldsymbol{x}_{near}$ until a limb $\boldsymbol{P}_{min}$ from $\boldsymbol{x}_{near}$ to $\boldsymbol{x}_{nearest}$ is created that makes for a shorter path from the root to $\boldsymbol{x}_{new,end}$. Then the limb $\boldsymbol{P}_{min}$ is added to the tree.

3.5.5 Stage 3: Finding Shortcuts to Other Nodes and Rewire the Tree

Next, the algorithm will try to find shortcuts in the tree that is enabled by $\boldsymbol{X}_{new,i}$. This is done by finding a set of $k_{connect}$ closest nodes $\boldsymbol{X}_{reachable}$ that the heuristic estimates has the shortest path from $\boldsymbol{x}_{new,end}$ to $\boldsymbol{x}_{reachable,j}$. Starting at the node that is closest to the root on the limb the algorithm checks with the heuristic for the estimated distance from the root via $\boldsymbol{X}_{new,i}$ to $\boldsymbol{x}_{reachable}$. If this distance is smaller then the path through the tree to $\boldsymbol{x}_{reachable,j}$ then the Connect function is used to make an exact connection between $\boldsymbol{X}_{new,i}$ and $\boldsymbol{x}_{reachable,j}$. If Connect returns a collision-free limb, then $\boldsymbol{x}_{reachable,j}$ removes its path to the old parent and uses the limb as the new parent. $\boldsymbol{x}_{reachable,j}$ is then removed from $\boldsymbol{X}_{reachable}$ since we now have found what is most likely the shortest path via $\boldsymbol{P}_{new}$ to it. This process is continued until there are no mode nodes in $\boldsymbol{X}_{reachable}$ or we have checked all the nodes up to $\boldsymbol{x}_{new,end}$.

Mark that we do not find a new set of $\boldsymbol{X}_{reachable}$ for every $\boldsymbol{x}_{new,i}$. This is to save processor time. The assumption here is that the nodes that are reachable from $\boldsymbol{x}_{new,end}$ would also be quite close to the rest of $\boldsymbol{P}_{new}$. This saves the number of times the closes nodes must be found as well as the number of calls to the Connect function.

Stage 3 is quite computationally expensive, and does not expand the tree into new regions. This stage is therefore not included until the tree has reached the goal state. At that point, we know that the tree is big enough to reach the goal, and we can therefore allow for slower exploration for the benefit of more optimal paths in the tree.

3.5.6 Stage 4: Goal Connections

At this stage, the algorithm loops through the nodes of the limb $\boldsymbol{P}_{new}$ and tries to make a collision-free path to the goal state. If it is successful the limb from $\boldsymbol{X}_{new,i}$ to $\boldsymbol{x}_{goal}$ is added to the tree. This step significantly reduces the time it takes the tree to reach $\boldsymbol{x}_{goal}$ or the region around $\boldsymbol{x}_{goal}$. Also due to the exact local planner, we can also expect the tree to create a path that reaches very close to $\boldsymbol{x}_{goal}$.

3.5.7 Path smoothing: Finding Shortcuts on the Shortest Path

The tree expansion process can sometimes lead to inefficient paths, that makes goes zig-zag or does unnecessary loops. Stage 2 and 3 are designed to minimize this behavior, but these stages are based on heuristics and only check a limited number of near nodes. The resulting path can therefore have non intuitive and sub-optimal behavior. To reduce this behavior, all the nodes in the shortest path $\boldsymbol{X}_{shortest}$ are tested for a reconnection with the other nodes in $\boldsymbol{X}_{shortest}$. Starting with the nodes that are furthest apart and working with nodes that are tighter and tighter together until all combinations are covered. This is of course an computationally expensive process, as it involve $\frac{(n-1)(n-2)}{2}$ calls to the

Algorithm 5: CL-RRT*

```
while time < TimeLimit do
    // Stage 1: Find a random state that the tree can connect to
    x_rand ← RandomSate()
    X_near ← Tree.kNearSorted(x_rand, k_steer)
    x_nearest ← X_near[1]
    X_near ← X_near \ x_nearest
    P_new ← Steer(x_nearest, x_rand)
    if CollisionFree(P_new) AND x_new ≈ x_rand then
        // Stage 2: Minimize the distance from the root to the new
           node
        X_min ← P_new
        c_min ← Cost(x_nearest) + Cost(X_nearest)
        X_near ←
         SortBy(X_near, Cost(x_near,i) + SteerHeuristic(x_near,i, x_new,end))
        foreach x_near,i ∈ X_near do
            [EstimatedLimbLength, ShortestDirection] ←
             SteerHeuristic(x_near,i, x_new,end)
            if Cost(x_near,i) + EstimatedLimbLength < c_min then
                X_new,m ← Steer(x_near, x_rand)
                if Cost(x_near,i) + Cost(X_min) < c_min then
                    X_min ← X_new,m
                    Break
        Tree.AddLimb(X_min)
        // Stage 3: Find shortcuts enabled by the new nodes.
        X_reachable ← Tree.kReachable(x_new, k_connect)
        foreach x_reachable,i ∈ X_reachable do
            [EstimatedLimbLength, ShortestDirection] ←
             ConnectHeuristic(x_new, x_reachable,i)
            if Cost(x_new) + EstimatedLimbLength < Cost(x_reachable,i) then
                P_shortcut ← Connect(x_new, x_reachable, ShortestDirection)
                if CollisionFree(P_shortcut) then
                    Tree.ReconnectThrought(x_reachable,i, x_new, P_shortcut)
        // Stage 4: Try to connect to the goal state x_goal.
        X_toGoal ← Connect(x_new, x_goal, Forward)
        if CollisionFree(X_toGoal) then
            Tree.AddNodePath(X_toGoal)
        X_toGoal ← Connect(x_new, x_goal, Reverse)
        if CollisionFree(X_toGoal) then
            Tree.AddNodePath(X_toGoal)
```

Connect function where n is the number of nodes in $\boldsymbol{X}_{shortest}$. An illustration of the smoothing step is shown in Fig. 3.9 and a pseudocode implementation is shown in Algorithm 6.

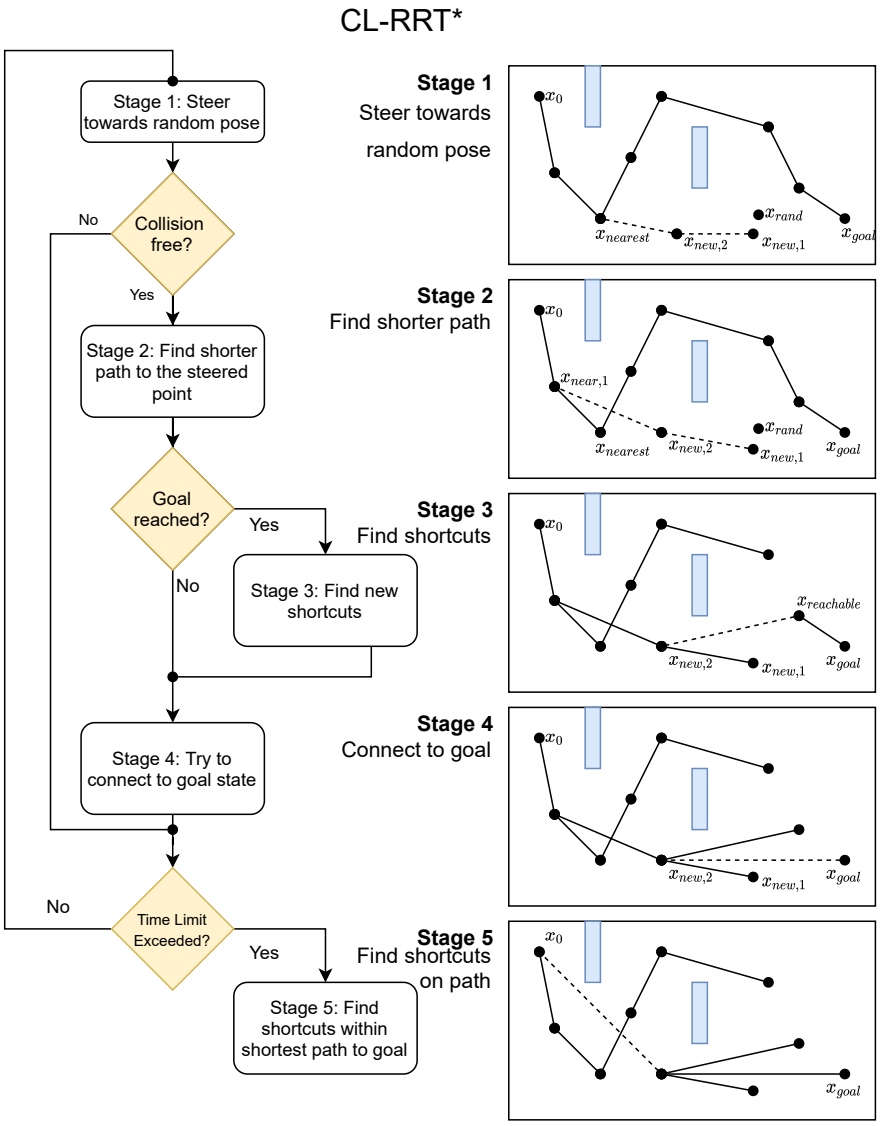


Figure 3.8: Flow diagram showing an overview of the CL-RRT* algorithm illustrated with an sample tree on the left. Mark that the tree in the illustration has straight paths between the nodes. In the real algorithm these paths would be smooth.

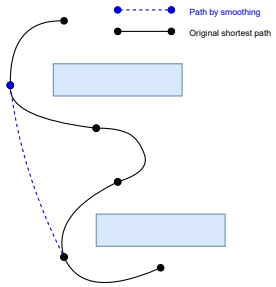


Figure 3.9: Illustration of the path being improved by the smoothing step

Algorithm 6: Path Smoothing

```
// Find shortcuts on shortest path
foreach Unordered combination of nodes x_from, x_to in X_shortest starting with
  the pair of nodes with the highest amount of node in between do
    [EstimatedLimbLength, ShortestDirection] ←
     ConnectHeuristic(x_from, x_to)
    if Cost(x_from) + EstimatedLimbLength < Cost(x_to) then
        P_shortcut ← Connect(x_from, x_to, ShortestDirection)
        if CollisionFree(P_shortcut) then
            Tree.ReconnectThrought(x_reachable,i, x_new, P_shortcut)
        Break
```

4 Results

4.1 Simulation Setup

The results generated in this chapter are all based on simulation on a vehicle-trailer system with the parameters are given in Table 4.1. The parameters are taken from a not public datasheet of the Mafi T 230e electric tractor. The tractor is used for container handling at seaports and therefore represents an application were backing up with a trailer is of great interest.

Parameter	**Value**
L_1	3 m
M_1	-0.68 m
L_2	5.7 m

Table 4.1: Vehicle-trailer parameters used for validating the path planning framework

The vehicle is simulated with explicit fixed-step Runge-Kutta method of 4th order with a step length $\Delta s_1 = 0.2m$ for the Connect function and $\Delta s_1 = 1.0m$. See Chapter 14 in Egeland and Gravdahl 2002 for details on this method. The reason for the different step lengths is that the Connect function's performance in terms of reaching the endpoints was significantly worse with a higher step length. The Steer function was more robust to higher step lengths than the Connect function. They were therefore given different step lengths to increase performance.

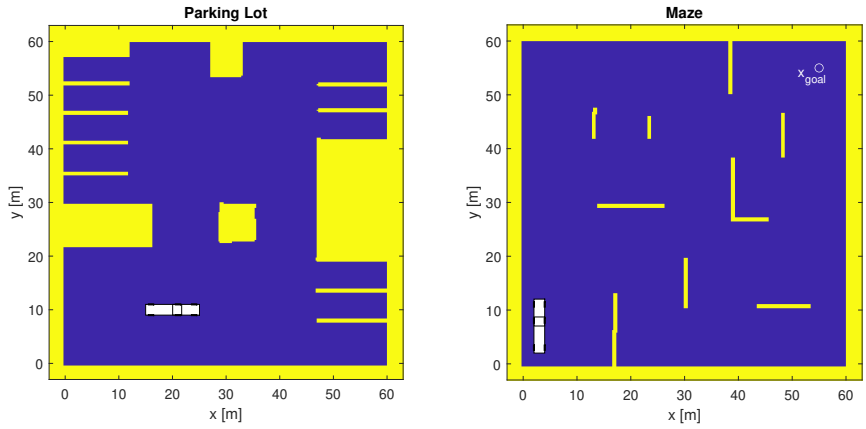
Multiple tuning parameters need to be set to run the simulations. The following parameters were used while testing CL-RRT*.

Parameter	**Value**
ϵ	23 m
Node Interval	5 m
k nearest Steer	5
k nearest Connect	15

Table 4.2: Tuning parameters that were used for CL-RRT*

The algorithm was implemented in MATLAB. The test reported in this section was performed on an Asus ZenBook with a Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz processor.

To test the performance of the CL-RRT* algorithm, two test environments where designed. The Parking Lot environment simulates a realistic use-case of path planning algorithm. The enviroments can be seen in Fig. 4.1. The Parking Lot is 60 by 60 meters and includes obstructions in the middle of the map that simulate obstructions such as a pillar or construction work. Each parking spot has a wall on each side. This is to simulate the parking spot having parked



(a) **(b)**

Figure 4.1: The two test environments. The initial position is marked with a visualization of the vehicle. In the Parking lot a goal state is placed randomly in one of the narrow parking spots. In the Maze the goal position is shown marked with $\boldsymbol{x}_{goal}$

cars or trailers on both sides, such that the vehicle trailer system must carefully navigate between them. The vehicle-trailer system starts with a random state in the lower-left corner. A random pose inside one of the parking spots is chosen as the goal state.

The second map is a Maze-like environment. The Maze tests the ability of the algorithm to navigate in an unstructured environment with many scattered obstructions. The Maze has many possible paths between the obstructions that can reach the goal. This will test the ability of our algorithm to not settle for a sub-optimal solution but to look for other possibilities. This particular scenario may not be very realistic; however, it represents environments that will be hard for humans to navigate and test the generality of the algorithm.

4.2 Evaluation of Exact Local Planner

This section will evaluate the performance of the exact local planner. Much of the properties of the Connect function can be analyzed by investigating the heuristic lookup tables from Section 3.3.

4.2.1 Success Rate

The lookup table contains the distance to each grid cell in 80 by 80 meter grid for $\theta_{2,to} \in \mathbb{S}$ and $\beta \in [-\pi/4, \pi/4]$. The grids are visualized in Fig. 4.2 toFig. 4.4. For each value of β_{to} the rate of successful connection between the $\boldsymbol{x}_{from}$ and $\boldsymbol{x}_{to}$ was calculated. The resulting success rates can be seen in Table 4.3.

β_{to}	Success Rate Approach Arch	Success Rate Dubins Path
0	99.39%	98.33%
$\pi/8$	95.46%	75.43%
$\pi/4$	81.46%	Not tested

Table 4.3: The exact local planner success rate with and without an approach arch added to the Dubins path.

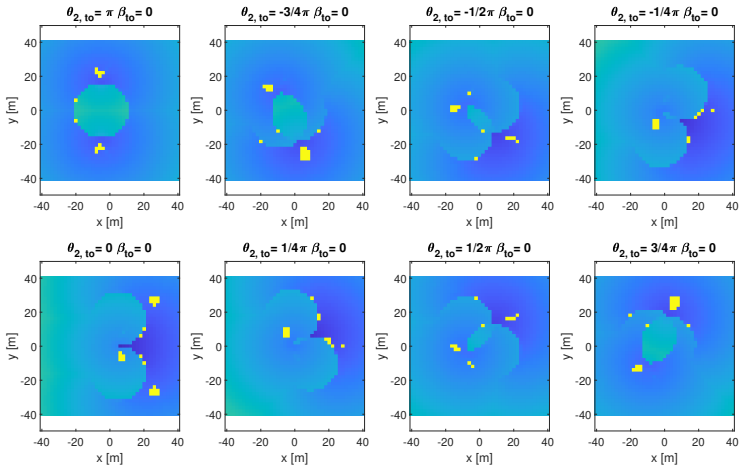


Figure 4.2: The distance heuristic for a $80m \times 80m$ grid with $\beta_g = 0$. Deeper blue means closer to the origin, and green means further away. Yellow indicates that the vehicle was not able to reach this state.

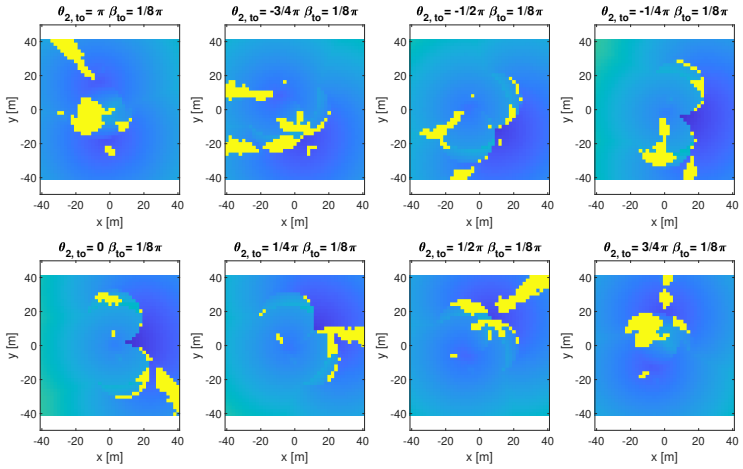


Figure 4.3: The distance heuristic for a $80m \times 80m$ grid with $\beta_g = \pi/8$.

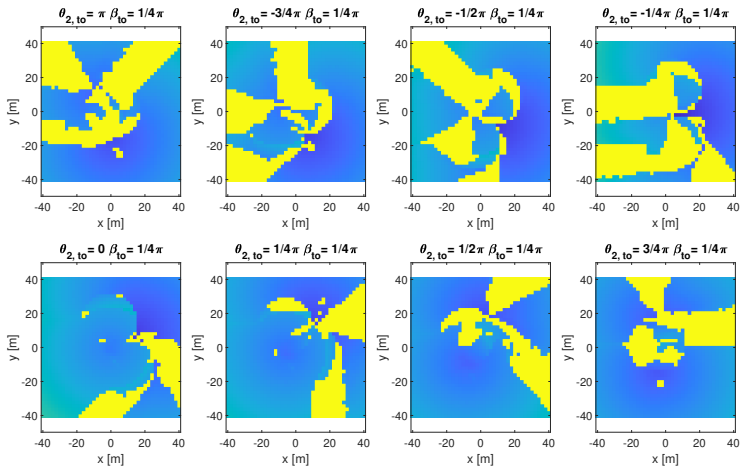


Figure 4.4: The distance heuristic for a $80m \times 80m$ grid with $\beta_g = \pi/4$.

It is clear that the exact local planner works best for small angles of β_{to}. For bigger values of β_{to} the performance goes considerably down.

4.2.2 Modification of Dubins Path

An evaluation of the exact local planner was also performed with and without the approach arch of the Dubins path. Fig. 4.5 and Fig. 4.6 shows an example of the difference between a connection attempt between two poses, with and without an approach arch. On Stage 2 of Fig. 4.5 it looks like the vehicle perfectly reaches $\boldsymbol{x}_{to}$, however β and θ_2 might not align with that of $\boldsymbol{x}_{to}$. In this case, these values are so far off that the reversing vehicle-trailer system cannot maintain tracking of the stage 2 path and the system can not recover from a jack-knife configuration. When a $5m$ approach arch is added, the end state of stage 2 is closer to $\boldsymbol{x}_{to}$, and the reversing vehicle-trailer system of stage 3 has no problem tracking the path of stage 2 and generating a successful local path.

From comparing the success rate of the heuristic function, which is reported in Table 4.3, it is clear that the approach arch greatly improved the success rate of the planner.

4.2.3 Accuracy

The Connect function does not create a perfect connection between two states. The endpoints of the path created by the Connect function will deviate a from the desired states $\boldsymbol{x}_{to}$ and $\boldsymbol{x}_{from}$.

The ρ_s distance for $\mathbb{R}^2 \times \mathbb{S}^2$, as described in Section 2.7, was used to measure how close the Connect function comes to connection two states. The distance

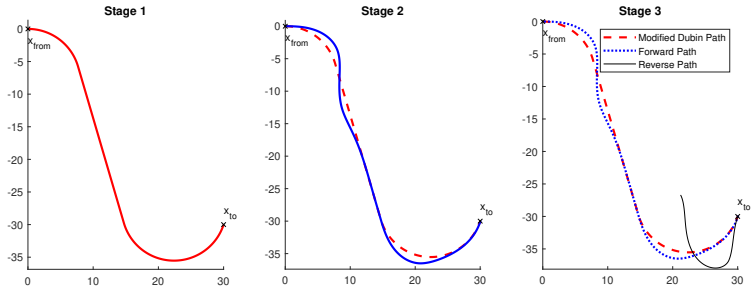


Figure 4.5: Fail attempt at exact local path planning between $\boldsymbol{x}_{from} = [0, 0, 0, 0]^T$ and $\boldsymbol{x}_{to} = [30, -30, 2\pi/5, \pi/8]^T$; without the approach arch. Stage 3 got to far away from the nominal path created by stage 2 and therefore lost tracking.

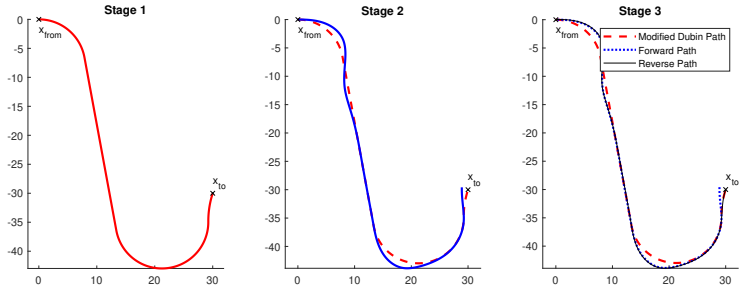


Figure 4.6: Successful attempt at exact local path planning between $\boldsymbol{x}_{from} = [0, 0, 0, 0]^T$ and $\boldsymbol{x}_{to} = [30, -30, 2\pi/5, \pi/8]^T$; with a approach length of 5 m.

is then measured as

$$d = \rho_s(\boldsymbol{x}_{from}, \boldsymbol{P}_{Connect}(0)) + \rho_s(\boldsymbol{x}_{to}, \boldsymbol{P}_{Connect}(T)) \tag{4.1}$$

We will use the unit meters for this distance, knowing this is a miss use of a meter since it also represent a deviation in orientation. However, meters are used since they give an upper bound on the deviation in euclidean distance of the position.

The Connect function was sampled within a grid of 60 times 60 grid with 8 values of θ_2 evenly distributed from 0 to 2π and two values of β; 0 and $\pi/8$. Negative value for β was not examined since the result will be symmetric to the positive result. A histogram of Connect path's deviations from $\boldsymbol{x}_{to}$ and $\boldsymbol{x}_{from}$ can be seen in Fig. 4.7.

Not including the distances above 0.5 meter, which will be discarded by the planner, 99.5% of the endpoints where within 0.20 m of their desired state.

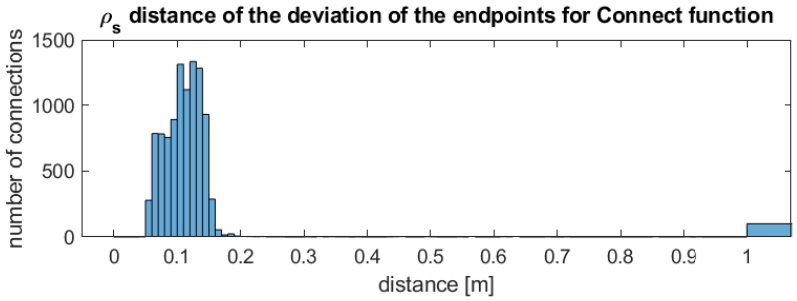


Figure 4.7: A histogram showing the ρ_s distance between the endpoints of the path created by the Connect function and x_{from} and x_{to} in the heuristic lookup table. The bin above 1 includes everything from 1 up to infinity.

4.3 Evaluation of the steer function

The heuristic lookup tables generated for the Steer function can be seen in Fig. 4.8 and Fig. 4.9. It is clear that the Steer function does not reach major path of the configuration space. Even though better coverage would likely increase the performance of the RRT algorithm, it is not critical for the algorithm to work. This is because the heuristic lookup table estimate based on the previous simulation if a state can be reached with the steer function, before an actual simulation with collision check is performed. Since the set of states which are reachable with the Steer function seam to mostly be continuous for most of the configuration space, it is reasonable to assume that the heuristic will be able to determine the outcome of a simulation in most cases.

4.4 CL-RRT*

4.4.1 Parking Lot

To test the CL-RRT* algorithm in a realistic scenario 100 simulations were performed in the Parking Lot as described in Section 4.1. For each randomly generated scenario, the planning algorithm was given 30 sec to find the best path possible.

To investigate the effectiveness of the stages implemented in this thesis, three different versions of the algorithm were tested. The base case is a simple basic implementation of CL-RRT which is the algorithm of Section 3.5 without stage 3 and stage 5. That is, the algorithm will not look for shortcuts whenever a new node is added, and it will not go through the final path and look for shortcuts. The second and third algorithms, are CL-RRT* with and without smoothing. The Connect function is used for goal connection in all cases, because not including any goal connection in CL-RRT would make the algorithm extremely slow. Both Kuwata et al. 2009 and Evestedt, Ljungqvist, and Axehill 2016 used goal connection in their implementation of CL-RRT.

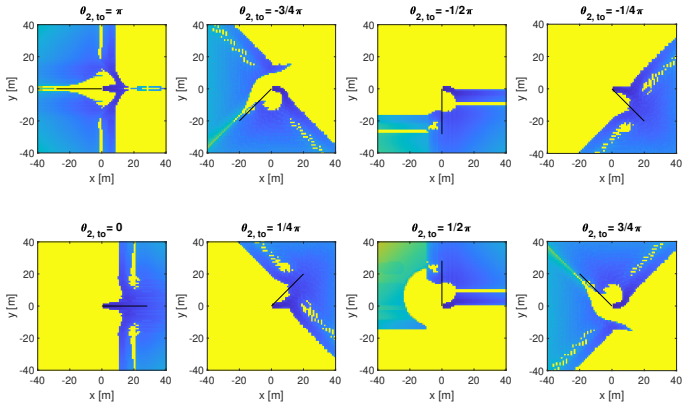


Figure 4.8: Distance to each grid state with the vehicle-trailer system controlled by the Steer function in forward travel direction. Deeper blue means closer to the origin, and green means further away. Yellow indicates that the vehicle was not able to reach this state.

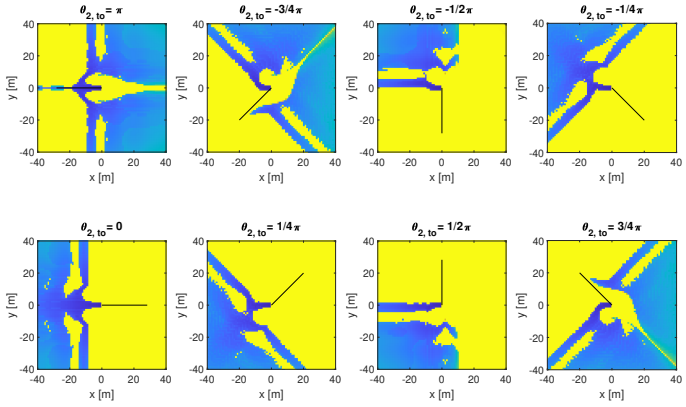


Figure 4.9: Distance to each grid state with the vehicle-trailer system controlled by the Steer function in reverse motion.

Description	**CL-RRT**	**CL-RRT*** without smoothing	**CL-RRT*** with smoothing
Average time to first path	1.24 s	1.22	1.21
Average length of first path	162.63 m	162.63 m	162.63 m
Success rate	99.0 %	99.0 %	99.0 %
Average length of final path	137.07 m	125.02 m	121.53 m

Table 4.4: The performance values for CL-RRT and CL-RRT* for the Parking Lot map.

The simulations were done in such as way that the randomly generated states $\boldsymbol{x}_{rand}$ as well as the $\boldsymbol{x}_0$ and $\boldsymbol{x}_{goal}$ would be the same for both CL-RRT and CL-RRT*. This was done to minimize variance in the performance between the two algorithms as a function of the randomly placed nodes in the configuration space. However, since CL-RRT and CL-RRT* will produce somewhat different trees, the variance between the two methods due to the random nodes will be increased as the algorithm expands the tree.

The average time it took to find the first path, the average length of the first path, and the average length of the path after 30 sec can be seen in Table 4.4 for both CL-RRT and CL-RRT*. The 16 first scenarios with the path visualized can be seen in Fig. 4.12 and Fig. 4.13.

The results from the simulations can be seen in Table 4.4. Since our implementation enables the connection part of CL-RRT* after the first goal connection is made, the time to the first path to goal as well the length of the goal path should be the same. Some variation in the time is expected due to variations in the scheduler of the computer used for simulation.

The CL-RRT* improves the shortest length of the path from 137.07m to 125.02m Mark that these lengths include the double penalty for reversing. Adding the smoothing stage, stage 5, after CL-RRT* has run improves the average shortest path to 121.53 m. From the 100 simulations that were performed, 23 solutions were improved by the smoothing stage, with an average improvement of 14.8 m.

From the visualizations in Fig. 4.12 and Fig. 4.13 we see that many of the shortest paths are exactly the same, or very similar, however, there are a few cases where CL-RRT* performs significantly better. These are the situations where CL-RRT has settled on a suboptimal solution where shortcuts can clearly be made. For example see 4a and 3c in Fig. 4.12 and Fig. 4.13. CL-RRT finds a solution by backing up around the obstruction, both taking a longer route and using the penalized baking maneuvers. CL-RRT* is able to find a shortcut by baking up a little, then driving forwards, and then back into the parking spot. It seems like both of the algorithms in general provide smooth, and intuitive solutions to the path-planning problem in the Parking Lot.

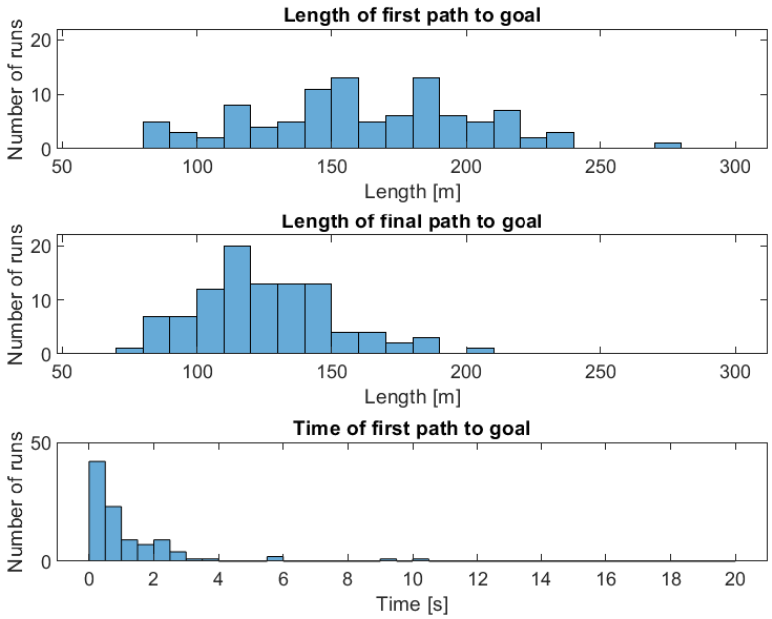


Figure 4.10: CL-RRT*

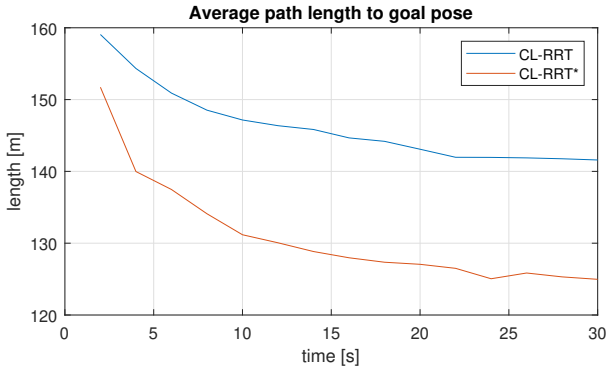


Figure 4.11: Plot of the average length of the shortest path over time for CL-RRT*.

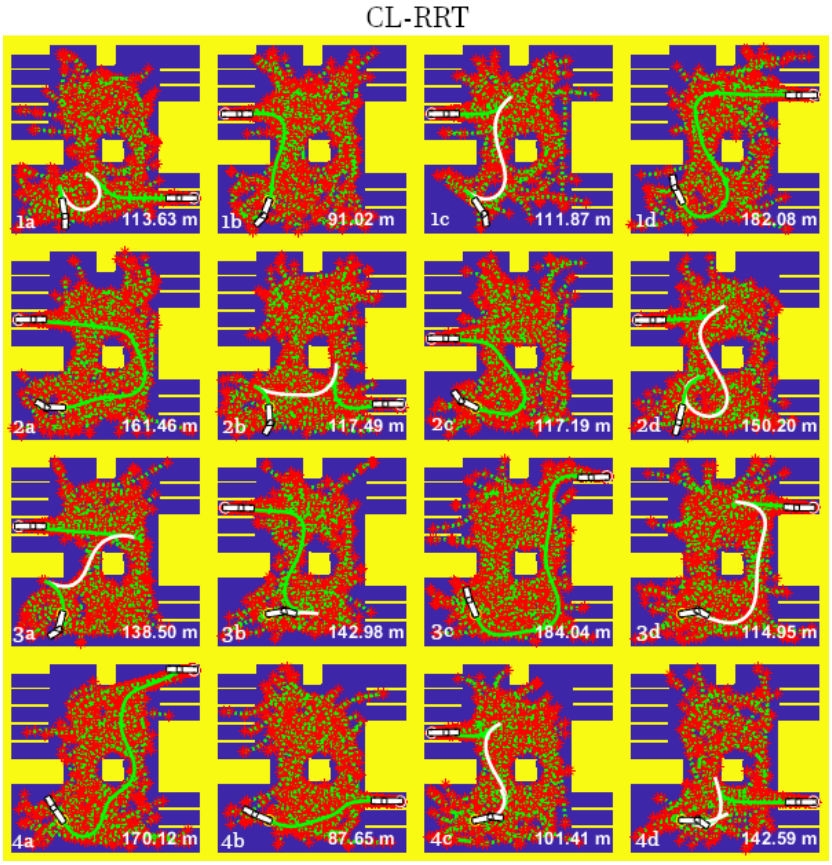


Figure 4.12: First 16 paths found in the Parking Lot simulation using CL-RRT after 30 sec. Yellow are obstructions, the green paths are trailer in reverse motion, white in forward motion and red are nodes in the tree.

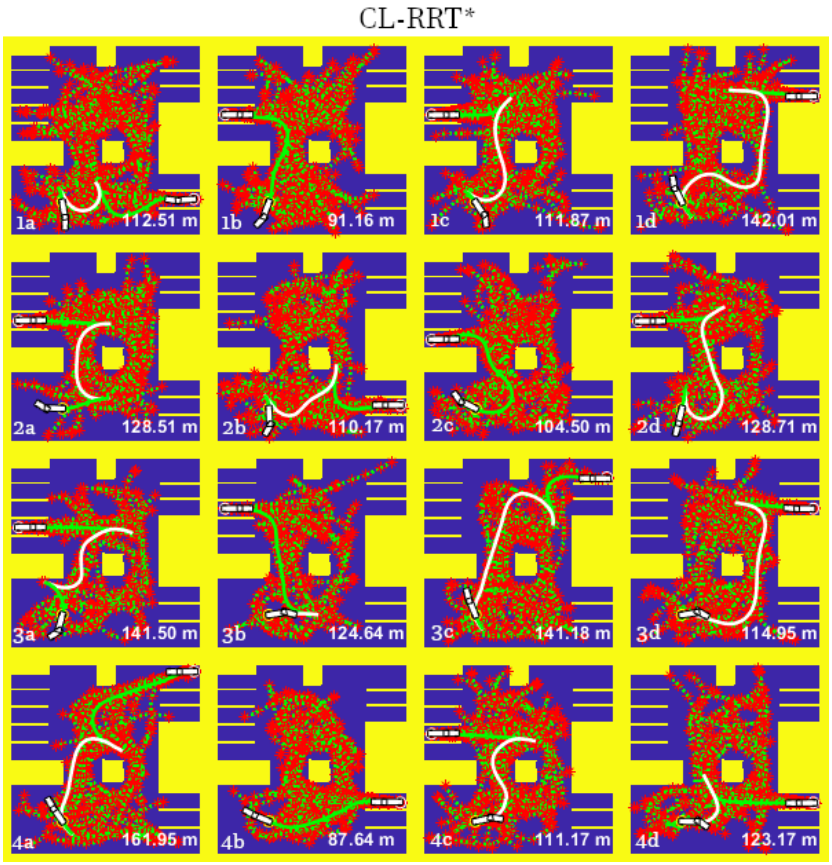
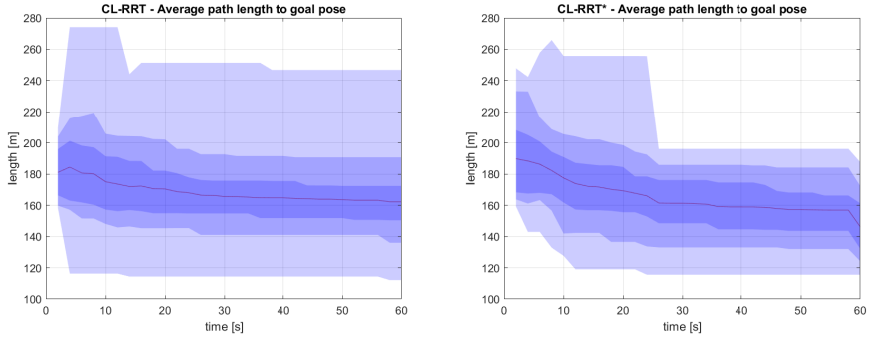


Figure 4.13: First 16 paths found in the Parking Lot simulation using CL-RRT* after 30 sec.



(a) **(b)**

Figure 4.14: The average length of the shortest path to the goal pose in the Maze as a function of time. The blue regions show the maximum and minimum length, the 10 to 90 percentile and 25 to 75 percentile.

Description	**CL-RRT**	**CL-RRT*** without smoothing	**CL-RRT*** with smoothing
Average time to first path	4.73 s	3.63	3.63
Average length of first path	194.31 m	193.67 m	193.67 m
Success rate	100.0 %	100.0 %	100.0 %
Average length of final path	162.27 m	153.76 m	143.27 m
Standard deviation of final path	23.5634 m	18.8164 m	18.7228 m

Table 4.5: The performance values for CL-RRT and CL-RRT* for the Maze environment.

4.4.2 Maze

The path planner was also tested in a more chaotic environment which we call the Maze. The scenario was simulated 50 times each for CL-RRT and CL-RRT*. To investigate the rate of convergence to the optimal solution the initial state and the goal state were the same through all the simulations. For each simulation, the planner was given 60 seconds to approach the optimal solution. The optimal solution to the problem was not found explicitly, however, the solution found by the planner with the shortest length can be used as an upper estimate of the optimal solution. If the algorithm is asymptotically optimal the average solution should at least approach the upper estimate of the optimal solution.

Fig. 4.14 shows the average length of the paths as a function of time. The average length is going down, and the spread of the solutions is getting smaller. The sudden improvement of the solutions found by CL-RRT* at the end in Fig. 4.14b is due to the smoothing of step 5 of the planning algorithm. The key values of the path found after 60 sec in shown in Table 4.5.

Function	Time used	Calls	Avg Time per Call
Connect	128.02s	6228	0.0206s
kNearSorted	74.71s	3588	0.0208s
Steer	41.9s	8796	0.0048s
kReachable	33.88s	1359	0.0249s
Others (including initializations)	32.22s		
Total	310.73s		

Table 4.6: The time used by the most time-consuming functions for 10 runs of 30 sec in the Parking Lot environment.

4.5 Run-time Analysis

It is important for many systems to run in real-time, preferably multiple times per second. This section will analyze the run-time of the algorithm to determine the bottlenecks for future work. The MATLAB profiler tool was used to time the algorithm. This slows the algorithm down by about 25%, however, assume that the relative time between the different functions stays approximately the same.

The profiler was tested on the Parking Lot environment as this scenario is seen as the most realistic scenario. 10 path-finding scenarios, each with different random seeds, were used to generate the analyze the performance. For each scenario, the planner was given 30 seconds to find a solution and improve upon it. The Steer, Connect, kNearSorted and kReachable functions take up 89.6 % of the computational time. The remaining time is utilized to initialize the vehicle-trailer's occupancy grid, calculate the LQR gain for the controllers, and build the tree during the tree expansion. The timing of each function is shown in Table 4.6. A chart visualizing the proportions of time for each function is shown in Fig. 4.15. On average, the Connect function used 4.3 times more time then the Steer function.

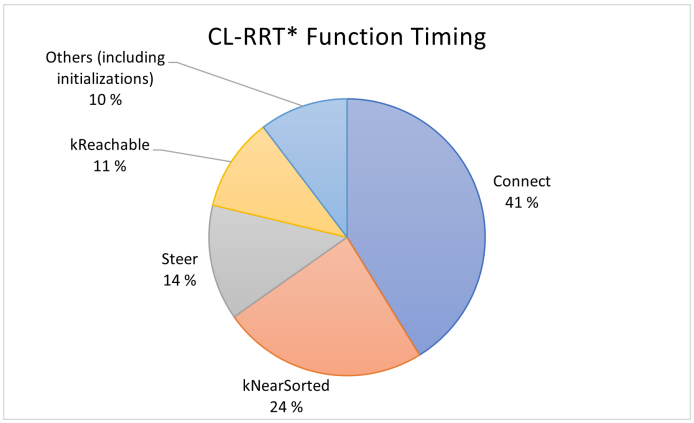


Figure 4.15: Pie chart showing the proportion of time taken by the most time-consuming functions.

5 Discussion

5.1 Evaluation and Future Work on the Connect Function

From the results in Section 4.2 we see that the Connect function can create a path between two states in a significant portion of the configuration space. Most of the connections are made such that the distance from the path endpoints to the states is within 0.2 m. We believe the endpoints of the path from the Connect function are close enough to the desired states that a vehicle using the path as a reference would not suffer in performance from the step between the endpoint of one path to another. This should be verified either with simulations of tire dynamics or with a real-life vehicle-trailer system.

The Connect function is about 4 times slower than the Steer function. Further tests should be performed to check this performance up against an exact local planner formulated as an optimal control problem. An interesting approach would be to do stage 1 and 2 of the Connect function and then do gradient descent on the last stage so that the endpoint aligned with the desired goal state. We suspect that the gradient descent method would be slower due to the relatively simple calculations performed in the Connect function.

The Connect function should also be measured up against the flat planner described in Section 2.9. Calculating a step of the flat planner requires both solving an integral and numerically finding a value through Newtons method. To increase the speed of the calculations, it might be beneficial to sacrifice some accuracy for speed by precomputing these values. This would be an interesting direction for future work.

The step length of the simulations in the Connect function was much shorter than that of the Steer function. The running time of the Connect function is directly proportional to the simulation step length. We have two suggestions for improving this. The first one could use the line-of-sight controller of Section 3.2 to generate the reference path from stage 2. Since the simulations with a line-of-sight controller allow for longer step lengths this would probably increase performance. However, one would lose the ability to do a collision check on the Dubins path before doing simulations. In addition, as seen in Section 4.3 is the line-of-sight controller stable in a smaller area then the Dubins path based controller. This could also hurt the performance of the overall algorithm. Therefore further investigations are needed on different methods of designing the Connect function.

A fixed-step Runge-Kutta method was used for the simulations. We believe that a variable step method could improve performance. This is because the dynamics of the systems is slow while following straight paths, and a larger step length can be used. In this thesis, the step length was set to a small value such that the simulation was stable for all cases.

The exact local planner only requires the system to be time reversible. It would

be interesting to investigate the possibility of using this method for other systems, such as a 2-trailer system.

5.2 Evaluation and Future Work on CL-RRT*

We see from the Parking Lot simulations that the CL-RRT* was successful at creating paths to the goal states. In terms of finding the shortest path, it was a bit better than CL-RRT. From Fig. 4.11 we see that path from CL-RRT* about 12 % shorter than that of CL-RRT. In the Maze environment, the convergence rate was very slow. For the CL-RRT* to be asymptotic optimal we would expect the solutions to at least approach the upper estimate of the optimal solution. From Fig. 4.14 we see that the average solution does not converge to the optimal solution or it does so at a rate that is impractical for most applications. We see a similar result when comparing the convergence rate of CL-RRT* with the convergence rate reported in Karaman and Frazzoli 2013. This might indicate that RRT* is not a viable solution for nonholonomic systems with real-time requirements.

The code was written in MATLAB. This might have been a big limiting factor on the computational time of the algorithm. We see it as very likely that implementing the algorithm in a language such as C++ would improve performance.

The simulations show that the first path was found within a few seconds in most cases. Then it would take an additional 10 to 30 sec before the iterative improvements of the algorithm would lower the path length. This execution time is too slow for many real-time applications. In an environment that is changing, the real-time requirements become even more challenging. Then the autonomous system would have to stop and recalculate the path again. In a few industrial settings, the algorithm could still be useful. For example take the use-case described in Section 1 of the vehicle-trailer operating at a sea port. In this case, the vehicle-trailer system must park a trailer in a controlled environment. 10 to 20 sec calculation time before parking the trailer can be seen as acceptable in this case.

For many scenarios, finding the optimal solution is not necessary. In these cases, the algorithm could be aborted early. Even toughed a better solution could be fund by spending an additional 20 sec on letting CL-RRT* expand the tree; this makes little sense if the path found only saves 2 sec of travel time.

An oversight that was made while implementing the CL-RRT* algorithm is that the time-reversible property of the vehicle-trailer system could be taken advantage of while updating the parents of a node. Say a shorter path has been found to node $\boldsymbol{x}_1$. In the current implementation, only root distance of the children and of $\boldsymbol{x}_1$ are recursively updated when the distance to $\boldsymbol{x}_1$ is updated. However, since the system is time-reversible, it is possible that a shorter path to the parent of $\boldsymbol{x}_1$ can be found by taking a route through $\boldsymbol{x}_1$. Implementing the update of distances through the parents would likely improve the converging rate of the algorithm.

As described in Section 2.6.3 there is a theoretical lower limit to the number of neighbors that must be checked during the shortcut stage of RRT*. In our case we are searching thought 4 dimension $[x_2, y_2, \theta_2, \beta]$ which gives $k_{Connect,min} = 106$. In our implementation we only used $k_{Connect} = 15$. We did not find any benefits from increasing $k_{Connect}$. This is likely due to the increased number of calls to the Connect functions which slowed the algorithm down. $k_{Connect,min}$ is derived in Karaman and Frazzoli 2011 for a vehicle without non-holonomic constraints. Although not reported in the results, multiple tests were performed by finding the nearest neighbors and reachable nodes using a fixed maximum distance instead of k-nearest. We found that this did not make any significant changes to the performance of the algorithm.

RRT-based methods for nonholonomic systems have a few issues that are hard to avoid. We believe there are three main issues with RRT for nonholonomic systems that are avoided by other methods:

- Doing k-nearest-search on every iteration is time consuming and inefficient compared to Hybrid A* and Lattice-graph search which can save states in a heap data structure..
- Many of the nodes generated by RRT are very close to each other, due to the nonholonomic constraints. We see it as unlikely that these nodes provide much in terms of exploring of the configuration space. Hybrid A* and Lattice-graph search both have mechanisms for ensuring that nodes are evenly spread out.
- Each time the tree is expanded a simulation of closed loop system must be performed. This is time consuming. Both Hybrid A* and Lattice-graph search uses precomputed paths when expanding.

We therefore think there are reason to believe that Hybrid A* or Lattice-graph search perform better when a short path is desired. The drawback of using Hybrid A* or Lattice-graph search is that these explore the configuration space densely. Meaning that all states, at the sampling resolution of the algorithms, are explored outwards from the initial state. For problems with wide open areas, RRT based methods might performer better then the two others. An interesting development in RRT is the use of learning based methods as described in Section 2.6.3 to find better states to expand the tree to. This might significantly reduce the number of expansions needed before the tree reached the goal state.

6 Conclusion

In this thesis the path planning problem for a vehicle-trailer system has been investigated. The goal has been to create a path planner for a 1-trailer system in a controlled industrial setting such as a sea port. A new method for generating a close to exact path between any two states of the vehicle-trailer system has been proposed. It was demonstrated that the new method for local planning was successful at generating paths between a wide range of states.

This thesis has demonstrated that CL-RRT* can be implemented for a vehicle-trailer system. To the authors knowledge this has not been shown before. This was achieved by using the exact local planner for rewiring of RRT-tree. The results showed that the planner was successful at creating feasible paths for the system in an realistic scenario. In the simulations the CL-RRT* algorithm gave solutions that where on average about 10% shorter then CL-RRT. When testing the CL-RRT* planner in a big and complicated environment, it was observed that the solution found by the planner converged very slowly to the optimal solution. For applications that require to find a the shortest path other methods such as Hybrid A* and Lattice-graph search might be better options.

References

Abbadi, Ahmad et al. (2014). "Spatial guidance to RRT planner using cell-decomposition algorithm". In: *20th international conference on soft computing, MENDEL*. Vol. 2014.

Atramentov, Anna and Steven M LaValle (2002). "Efficient nearest neighbor searching for motion planning". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*. Vol. 1. IEEE, pp. 632–637.

Bock, Hans Georg and Karl-Josef Plitt (1984). "A multiple shooting algorithm for direct solution of optimal control problems". In: *IFAC Proceedings Volumes* 17.2, pp. 1603–1608.

Bolu, Ali and Ömer Korçak (2021). "Adaptive Task Planning for Multi-Robot Smart Warehouse". In: *IEEE Access* 9, pp. 27346–27358.

Borisov, Alexey Vladimirovich and Ivan Sergeevich Mamaev (2005). "On the history of the development of the nonholonomic dynamics". In: *arXiv preprint nlin/0502040*.

Brandt, David (2006). "Comparison of a and rrt-connect motion planning techniques for self-reconfiguration planning". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 892–897.

Bresson, Guillaume et al. (2017). "Simultaneous localization and mapping: A survey of current trends in autonomous driving". In: *IEEE Transactions on Intelligent Vehicles* 2.3, pp. 194–220.

Cadena, Cesar et al. (2016). "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age". In: *IEEE Transactions on robotics* 32.6, pp. 1309–1332.

Chen, Chi-Tsong (1999). "Linear System Theory and Design". In:

Chitsaz, Hamidreza et al. (2009). "Minimum wheel-rotation paths for differential-drive mobile robots". In: *The International Journal of Robotics Research* 28.1, pp. 66–80.

Collins, Thomas, JJ Collins, and Donor Ryan (2007). "Occupancy grid mapping: An empirical evaluation". In: *2007 mediterranean conference on control & automation*. IEEE, pp. 1–6.

Dolgov, Dmitri et al. (2008). "Practical search techniques in path planning for autonomous driving". In: *Ann Arbor* 1001.48105, pp. 18–80.

Dubins, Lester E (1957). "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents". In: *American Journal of mathematics* 79.3, pp. 497–516.

Duleba, Szabolcs et al. (2021). "Ranking the key areas for autonomous proving ground development using Pareto Analytic Hierarchy Process". In: *IEEE Access* 9, pp. 51214–51230.

Egeland, Olav and Jan Tommy Gravdahl (2002). *Modeling and simulation for automatic control*. Vol. 76. Marine Cybernetics Trondheim, Norway.

Ericson, Christer (2004). *Real-time collision detection*. Crc Press.

Evestedt, Niclas, Oskar Ljungqvist, and Daniel Axehill (2016). "Motion planning for a reversing general 2-trailer configuration using Closed-Loop RRT". In:

2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 3690–3697.

Figueiredo, Mauro et al. (2010). "An efficient collision detection algorithm for point cloud models". In: *'2010*. Citeseer, pp. 30–37.

Fliess, M et al. (1997). "Controlling nonlinear systems by flatness". In: *Systems and Control in the Twenty-first Century*. Springer, pp. 137–154.

Fliess, Michel et al. (1995). "Flatness and defect of non-linear systems: introductory theory and examples". In: *International journal of control* 61.6, pp. 1327–1361.

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

Gammell, Jonathan D, Siddhartha S Srinivasa, and Timothy D Barfoot (2014). "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 2997–3004.

Golwala, Sunil (2014). "Lecture notes on classical mechanics for physics 106ab". In: *Publisher: CreateSpace Independent Publishing Platform*.

Hart, Peter E, Nils J Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Hejase, Mohammad et al. (2018). "Constrained backward path tracking control using a plug-in jackknife prevention system for autonomous tractor-trailers". In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, pp. 2012–2017.

Holmer, Olov (2016). *Motion planning for a reversing full-scale truck and trailer system*.

Ichter, Brian, James Harrison, and Marco Pavone (2018). "Learning sampling distributions for robot motion planning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 7087–7094.

Karaman, Sertac and Emilio Frazzoli (2011). "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7, pp. 846–894.

– (2013). "Sampling-based optimal motion planning for non-holonomic dynamical systems". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 5041–5047.

Kuffner, James J and Steven M LaValle (2000). "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE, pp. 995–1001.

Kuwata, Yoshiaki et al. (2009). "Real-time motion planning with applications to autonomous urban driving". In: *IEEE Transactions on control systems technology* 17.5, pp. 1105–1118.

Kvarnfors, Karl (2019). *Motion Planning for Parking a Truck and Trailer System*.

Lamiraux, Florent and J-P Laumond (2000). "Flatness and small-time controllability of multibody mobile robots: Application to motion planning". In: *IEEE Transactions on Automatic Control* 45.10, pp. 1878–1881.

Laumond, Jean-Paul, Sepanta Sekhavat, and Florent Lamiraux (1998). "Guidelines in nonholonomic motion planning for mobile robots". In: *Robot motion planning and control*. Springer, pp. 1–53.

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press.

Lee, Dasol, HanJun Song, and David Hyunchul Shim (2014). "Optimal path planning based on spline-RRT* for fixed-wing UAVs operating in three-dimensional environments". In: *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*. IEEE, pp. 835–839.

Lekkas, Anastasios M (2014). "Guidance and path-planning systems for autonomous vehicles". In:

Li, Bai et al. (2019). "Tractor-trailer vehicle trajectory planning in narrow environments with a progressively constrained optimal control approach". In: *IEEE Transactions on Intelligent Vehicles* 5.3, pp. 414–425.

Li, Yang et al. (2018). "Neural network approximation based near-optimal motion planning with kinodynamic constraints using RRT". In: *IEEE Transactions on Industrial Electronics* 65.11, pp. 8718–8729.

Ljungqvist, Oskar (2020). *Motion planning and feedback control techniques with applications to long tractor-trailer vehicles*. Vol. 2070. Linköping University Electronic Press.

Ljungqvist, Oskar, Daniel Axehill, and Anders Helmersson (2016). "Path following control for a reversing general 2-trailer system". In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, pp. 2455–2461.

Ljungqvist, Oskar, Daniel Axehill, and Johan Löfberg (2018). "On stability for state-lattice trajectory tracking control". In: *2018 Annual American Control Conference (ACC)*. IEEE, pp. 5868–5875.

Ljungqvist, Oskar, Niclas Evestedt, et al. (2019). "A path planning and path-following control framework for a general 2-trailer with a car-like tractor". In: *Journal of field robotics* 36.8, pp. 1345–1377.

Manav, Ahmet Canberk and Ismail Lazoglu (2021). "A Novel Cascade Path Planning Algorithm for Autonomous Truck-Trailer Parking". In: *IEEE Transactions on Intelligent Transportation Systems*.

Meyer-Delius, Daniel, Maximilian Beinhofer, and Wolfram Burgard (2012). "Occupancy grid models for robot mapping in changing environments". In: *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Nash, Alex et al. (2007). "Theta^*: Any-angle path planning on grids". In: *AAAI*. Vol. 7, pp. 1177–1183.

Nordeus, Erik (2015). *Self-driving-vehicle*. `https://github.com/Habrador/Self-driving-vehicle.git`.

Noreen, Iram, Amna Khan, Zulfiqar Habib, et al. (2016). "Optimal path planning using RRT* based approaches: a survey and future directions". In: *Int. J. Adv. Comput. Sci. Appl* 7.11, pp. 97–107.

Oliveira, Rui et al. (2018). "Trajectory generation using sharpness continuous dubins-like paths with applications in control of heavy-duty vehicles". In: *2018 European Control Conference (ECC)*. IEEE, pp. 935–940.

Palmieri, Luigi, Sven Koenig, and Kai O Arras (2016). "RRT-based nonholonomic motion planning using any-angle path biasing". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 2775–2781.

Pan, Jia et al. (2013). "Real-time collision detection and distance computation on point cloud sensor data". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE, pp. 3593–3599.

Pancanti, Stefania et al. (2004). "Motion planning through symbols and lattices". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004.* Vol. 4. IEEE, pp. 3914–3919.

Pivtoraiko, Mihail, Ross A Knepper, and Alonzo Kelly (2009). "Differentially constrained mobile robot motion planning in state lattices". In: *Journal of Field Robotics* 26.3, pp. 308–333.

Reeds, James and Lawrence Shepp (1990). "Optimal paths for a car that goes both forwards and backwards". In: *Pacific journal of mathematics* 145.2, pp. 367–393.

Rouchon, Pierre, Michel Fliess, Jean Lévine, et al. (Jan. 1992). "Flatness and motion Planning: the Car with n-trailers". In: *European Control Conference*.

– (1993). "Flatness, motion planning and trailer systems". In: *Proceedings of 32nd IEEE Conference on Decision and Control*. IEEE, pp. 2700–2705.

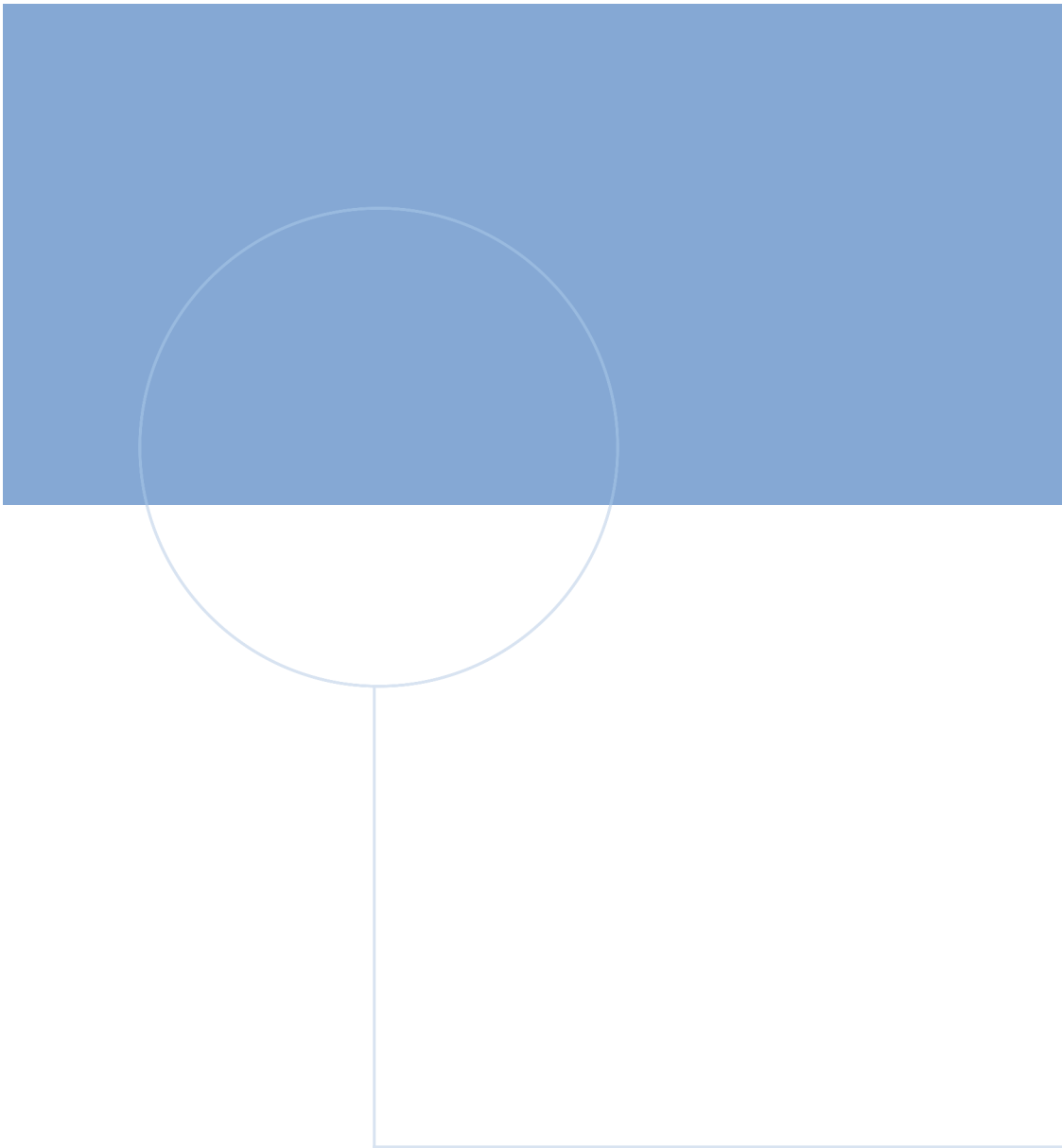
Schwarting, Wilko, Javier Alonso-Mora, and Daniela Rus (2018). "Planning and decision-making for autonomous vehicles". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1, pp. 187–210.

Sekhavat, Sepanta et al. (1998). "Multilevel path planning for nonholonomic robots using semiholonomic subsystems". In: *The international journal of robotics research* 17.8, pp. 840–857.

Svestka, Petr and Jules Vleugels (1995). "Exact motion planning for tractor-trailer robots". In: *Proceedings of 1995 IEEE International Conference on Robotics and Automation*. Vol. 3. IEEE, pp. 2445–2450.

Unity (2020). *Inner Workings of the Navigation System Kernel Description*. URL: `https://docs.unity3d.com/Manual/nav-InnerWorkings.html` (visited on 11/11/2021).

Yang, Fan et al. (2019). "Automatic indoor reconstruction from point clouds in multi-room environments with curved walls". In: *Sensors* 19.17, p. 3798.



NTNU

Kunnskap for en bedre verden