On Motion Planning Using Numerical Optimal Control

Kristoffer Bergman
On Motion Planning Using Numerical Optimal Control

Kristoffer Bergman
This is a Swedish Licentiate’s Thesis.

Swedish postgraduate education leads to a Doctor’s degree and/or a Licentiate’s degree. A Doctor’s Degree comprises 240 ECTS credits (4 years of full-time studies). A Licentiate’s degree comprises 120 ECTS credits, of which at least 60 ECTS credits constitute a Licentiate’s thesis.

Linköping studies in science and technology. Licentiate Thesis No. 1843

**On Motion Planning Using Numerical Optimal Control:**

Kristoffer Bergman

[kristoffer.bergman@liu.se](mailto:kristoffer.bergman@liu.se)

[www.control.isy.liu.se](http://www.control.isy.liu.se)

Department of Electrical Engineering

Linköping University

SE-581 83 Linköping

Sweden

ISBN 978-91-7685-057-2 ISSN 0280-7971

Copyright © 2019 Kristoffer Bergman

Printed by LiU-Tryck, Linköping, Sweden 2019
To my family and friends!
Abstract

During the last decades, motion planning for autonomous systems has become an important area of research. The high interest is not the least due to the development of systems such as self-driving cars, unmanned aerial vehicles and robotic manipulators. In this thesis, the objective is not only to find feasible solutions to a motion planning problem, but solutions that also optimize some kind of performance measure. From a control perspective, the resulting problem is an instance of an optimal control problem. In this thesis, the focus is to further develop optimal control algorithms such that they can be used to obtain improved solutions to motion planning problems. This is achieved by combining ideas from automatic control, numerical optimization and robotics.

First, a systematic approach for computing local solutions to motion planning problems in challenging environments is presented. The solutions are computed by combining homotopy methods and numerical optimal control techniques. The general principle is to define a homotopy that transforms, or preferably relaxes, the original problem to an easily solved problem. The approach is demonstrated in motion planning problems in 2D and 3D environments, where the presented method outperforms both a state-of-the-art numerical optimal control method based on standard initialization strategies and a state-of-the-art optimizing sampling-based planner based on random sampling.

Second, a framework for automatically generating motion primitives for lattice-based motion planners is proposed. Given a family of systems, the user only needs to specify which principle types of motions that are relevant for the considered system family. Based on the selected principle motions and a selected system instance, the algorithm not only automatically optimizes the motions connecting pre-defined boundary conditions, but also simultaneously optimizes the terminal state constraints as well. In addition to handling static a priori known system parameters such as platform dimensions, the framework also allows for fast automatic re-optimization of motion primitives if the system parameters change while the system is in use. Furthermore, the proposed framework is extended to also allow for an optimization of discretization parameters, that are are used by the lattice-based motion planner to define a state-space discretization. This enables an optimized selection of these parameters for a specific system instance.

Finally, a unified optimization-based path planning approach to efficiently compute locally optimal solutions to advanced path planning problems is presented. The main idea is to combine the strengths of sampling-based path planners and numerical optimal control. The lattice-based path planner is applied to the problem in a first step using a discretized search space, where system dynamics and objective function are chosen to coincide with those used in a second numerical optimal control step. This novel tight combination of a sampling-based path planner and numerical optimal control makes, in a structured way, benefit of the former method’s ability to solve combinatorial parts of the problem and the latter method’s ability to obtain locally optimal solutions not constrained to a discretized search space. The proposed approach is shown in several practically relevant path planning problems to provide improvements in terms of computation time, numerical reliability, and objective function value.

I denna avhandling är målet inte enbart att beräkna lösningar som tar hänsyn till systemens begränsningar, utan även att optierna ett användarvatt prestandamått. Några exempel på praktiskt användbara mått är minimering av energiförbrukningen, sträckan eller tiden. Ur ett reglertekniskt perspektiv är det här problemet en typ av optimalt styrbproblem (eng. optimal control problem). Fokus i den här avhandlingen ligger därför på hur algoritmer för att lösa optimala styrbproblem kan anpassas till att lösa rörelseplaneringsproblem.


Det andra bidraget är ett optimeringsramverk som ska användas inom en befintlig planeringsmetod från robotik som kallas lattice-baserade rörelseplanerare (eng. lattice-based motion planners). Dessa planerare söker efter lösningar till rörelseplaneringsproblem genom att kombinera delrörelser från ett bibliotek av förberäknade rörelser. Det föreslagna ramverket optimerar beräkningen av detta bibliotek automatiskt för en familj av system. Tidigare har detta arbete ofta utförts manuellt, något som är tidskrävande och kräver expertkunskap, eller genom en uttömmande sökning vilket bara går att applicera på vissa typer av system.

I den sista delen av den här avhandlingen kombineras metoder från robotik och optimal styrning för att lösa rörelseplaneringsproblem. Huvudmålet bygger på att utnyttja styrkorna hos både lattice-baserade rörelseplanerare och algoritmer för optimal styrning. I ett första steg löser lattice-baserade planeraren problemet i en diskretiserad sökrymd, där både prestandamått som optimeras samt rörelsemodell är valda till att vara samma som i det efterföljande optimala styrningssteget. Denna kombination utnyttjar, på ett strukturerat sätt, lattice-planerarens förmåga att lösa kombinatoriska delar av problemet (som att välja på vilken sida ett hinder ska passeras), samt förmågan att beräkna lösningar som inte är begränsade till en diskret sökrymd som algoritmer för optimal styrning besitter.
Acknowledgments

First of all, I would like to thank my supervisor Assoc. Prof. Daniel Axehill for your continuous support and encouragement during the last three years. Thank you for your never-ending enthusiasm, and for always motivating me to do my best! I would also like to thank my co-supervisor Prof. Torkel Glad for your assistance.

Thank you Prof. Svante Gunnarson, former head of Division at Automatic Control, and Assoc. Prof. Martin Enqvist, the current head of division. You both have done an excellent work to create a working environment that makes everyone feel welcome!

This thesis has greatly been improved by feedback from Daniel Arnström, Per Boström-Rost, Erik Hedberg, Herman Ekwall, Fredrik Ljungberg, Lic. Oskar Ljungqvist and Magnus Malmström. I really appreciate the time you spent on proofreading parts of this thesis. Furthermore, I thank Assoc. Prof. Gustaf Hendeby and Dr. Henrik Tidefelt for providing the template that has been used to write this thesis.

I would like to thank current and former colleagues at Automatic Control for contributing to such a friendly working atmosphere. A special thanks to my officemate and friend, Per Boström-Rost, for making time at the office more fun, and for teaching me how to properly use Google’s search engine. Thanks to Oskar Ljungqvist for our fruitful research collaborations and discussions during the past years. Thank you Andreas Bergström, Johan Dahlin, Angela Fontan, Robin Forsling, Erik Hedberg, Du Ho, Ylva Jung, Parinaz Kasebzadeh, Jonas Lindner, Martin Lindfors, Gustav Lindmark, Fredrik Ljungberg, Isak Nielsen, Magnus Malmström, Shervin Parvini Ahmadi, Kamiar Radnosrati, Zoran Sjanic, Martin Skoglund, Alberto Zenere, Clas Veibäck, Farnaz Adib Yaghmaie and all others for making fika breaks, lunch walks and other activities a pleasure!

This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation. WASP did not only contribute with funding, but also an invaluable network with people from all over the world. Thank you all AS batch one students for making courses, conferences and international study trips more fun, and to all the WASP seniors who have arranged them! Furthermore, I would like to thank my manager at Saab Dynamics AB, Lic. Torbjörn Crona, for believing in me and giving the chance to take this fantastic opportunity.

Finally, I would like to thank family: my sister Emelie, my father Lars and my mother Maria, for always being there for me. Matilda, you are not only the love of my life, but also my best friend. Thank you for your endless love and support, especially in times when I need it the most.

Linköping, May 2019
Kristoffer Bergman
# Contents

Notation xiii

1 Introduction 1
   1.1 Background and motivation .................................. 1
   1.2 Publications and contributions ................................ 3
   1.3 Thesis outline .................................................. 4

2 Background 5
   2.1 Nonlinear optimization ......................................... 5
      2.1.1 Nonlinear programming problem .......................... 5
      2.1.2 Optimality conditions .................................... 6
      2.1.3 Algorithms for solving nonlinear programs .............. 8
      2.1.4 Globalization techniques .................................. 10
   2.2 Optimal control .................................................. 13
      2.2.1 Direct methods for optimal control ....................... 14
   2.3 Optimal motion planning ........................................ 17
      2.3.1 Nonlinear system models .................................. 17
      2.3.2 Problem formulation ....................................... 18
      2.3.3 Sequential solution concept ............................... 19
      2.3.4 Sampling-based motion planning ........................... 21
      2.3.5 Lattice-based motion planning ............................ 24

3 Homotopy for motion planning 31
   3.1 Homotopy-based reformulation ................................. 31
   3.2 Sequential Homotopy Quadratic Programming .................. 35
   3.3 Obstacle classification ......................................... 37
      3.3.1 Examples of obstacle representations .................... 38
   3.4 Numerical results ............................................... 39
      3.4.1 Vehicle models ............................................. 40
      3.4.2 Experimental results ...................................... 41

4 Improved motion primitive optimization 45
   4.1 Motivation ....................................................... 45
## Contents

4.2 Maneuver-based motion primitive generation ................................................. 46
4.3 Extended maneuver optimization ................................................................. 51
  4.3.1 Selecting motion primitive weights ...................................................... 53
4.4 Numerical results .............................................................................................. 54
  4.4.1 Vehicle models ....................................................................................... 55
  4.4.2 State-space discretizations ..................................................................... 57
  4.4.3 Maneuver specifications ........................................................................... 57
  4.4.4 Experimental results .............................................................................. 59
  4.4.5 Extended maneuver optimization ............................................................ 62

5 Integrating lattice-based planning and numerical optimal control 65
  5.1 Problem formulation .................................................................................... 66
  5.2 Bilevel optimization preliminaries ............................................................... 67
  5.3 Bilevel optimization problem reformulation ............................................... 68
  5.4 Analysis of solution properties using bilevel arguments ............................... 69
  5.5 Improvement using numerical optimal control ............................................. 71
    5.5.1 Proposed path planning approach ....................................................... 72
    5.5.2 Variations of the proposed approach ................................................... 73
  5.6 Numerical results ............................................................................................ 73
    5.6.1 Vehicle models .................................................................................... 74
    5.6.2 State-lattice construction ...................................................................... 75
    5.6.3 Experimental results ............................................................................ 76

6 Conclusions and future work .............................................................................. 81
  6.1 Conclusions .................................................................................................. 81
  6.2 Future work .................................................................................................. 83

Bibliography .......................................................................................................... 85
**Notation**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbf{R}$</td>
<td>Set of real numbers</td>
</tr>
<tr>
<td>$\mathbf{Z}$</td>
<td>Set of integers</td>
</tr>
<tr>
<td>$\mathbf{Z}_{i,j}$</td>
<td>Set of integers ${i, i+1, \ldots, j-1, j}$</td>
</tr>
<tr>
<td>$\mathbb{E}(z)$</td>
<td>Expected value of random variable $z$</td>
</tr>
<tr>
<td>$[a, b]$</td>
<td>Interval of real numbers $x \in \mathbf{R}$ such that $a \leq x \leq b$</td>
</tr>
<tr>
<td>$\nabla f$</td>
<td>Gradient of the function $f: \mathbf{R}^n \to \mathbf{R}$</td>
</tr>
<tr>
<td>$\nabla_z f$</td>
<td>Gradient of the function $f(z,x): \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ with respect to $z$</td>
</tr>
<tr>
<td>$\nabla_{zz}^2 f$</td>
<td>Hessian of the function $f(z,y): \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}$ with respect to $z$</td>
</tr>
<tr>
<td>$z \triangleq y$</td>
<td>Definition of the variable $z$ as equal to $y$</td>
</tr>
<tr>
<td>$z \succeq y$</td>
<td>$z_i \geq y_i$ holds for each pair of components of $z$ and $y$</td>
</tr>
<tr>
<td>$Q \succeq 0$</td>
<td>Matrix $Q$ is positive semidefinite</td>
</tr>
<tr>
<td>$|z|_2$</td>
<td>Euclidean norm of vector $z$</td>
</tr>
<tr>
<td>$</td>
<td>S</td>
</tr>
<tr>
<td>$\mathcal{O}(g(z))$</td>
<td>Complexity growth is dominated by $g(z)$ when $z \to \infty$</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>BVP</td>
<td>Boundary Value Problem</td>
</tr>
<tr>
<td>HJB</td>
<td>Hamilton-Jacobi-Bellman</td>
</tr>
<tr>
<td>HLUT</td>
<td>Heuristic Look-Up Table</td>
</tr>
<tr>
<td>IP</td>
<td>Interior Point</td>
</tr>
<tr>
<td>KKT</td>
<td>Karush-Kuhn-Tucker</td>
</tr>
<tr>
<td>LICQ</td>
<td>Linear Independence Constraint Qualification</td>
</tr>
<tr>
<td>MINLP</td>
<td>Mixed-Integer Nonlinear Programming</td>
</tr>
<tr>
<td>MPC</td>
<td>Model Predictive Control</td>
</tr>
<tr>
<td>NLP</td>
<td>Nonlinear Programming</td>
</tr>
<tr>
<td>ODE</td>
<td>Ordinary Differential Equation</td>
</tr>
<tr>
<td>OCP</td>
<td>Optimal Control Problem</td>
</tr>
<tr>
<td>PMP</td>
<td>Pontryagin’s Maximum Principle</td>
</tr>
<tr>
<td>RRT</td>
<td>Rapidly exploring Random Tree</td>
</tr>
<tr>
<td>SHQP</td>
<td>Sequential Homotopy Quadratic Programming</td>
</tr>
<tr>
<td>SQP</td>
<td>Sequential Quadratic Programming</td>
</tr>
<tr>
<td>SST</td>
<td>Stable Sparse RRT</td>
</tr>
<tr>
<td>QP</td>
<td>Quadratic Programming</td>
</tr>
</tbody>
</table>
One of the main goals in the fields of robotics and control is to develop systems that are autonomous. These systems are able to interpret high-level descriptions of tasks and execute them without any human intervention. These high-level tasks only contain information of what to do, and it is up to the algorithms within the autonomous system to find solutions to the tasks. Today, the development of autonomous systems occurs for a wide variety of applications such as security and surveillance, aerospace, manufacturing, underwater operations, automotive and much more. For an autonomous system to work as intended, several technologies need to be integrated such as sensing and perception, automated reasoning and control [43]. Within the research of these technologies, several subproblems arise that need to be solved. One of them is motion planning, which considers the problem of finding a feasible motion plan that brings a system from its current state to a terminal state while avoiding obstacles in the surrounding environment.

1.1 Background and motivation

A common approach in classical motion planning has been to ignore the system dynamics during planning and instead focus on, e.g., how to get to the terminal state by translating and rotating a body without colliding with obstacles. This problem is commonly referred to as the piano mover’s problem [45]. For classical motion planning, there are mainly two general solution approaches that have been used, namely combinatorial and sampling-based planning approaches. Combinatorial approaches compute graph representations of the free space that exactly represent the original problems, while sampling-based approaches sample the state space and conduct discrete searches using these samples. In recent years, motion planning for systems that are subject to differential constraints has become an important area of research not the least due to the recent high interest in systems
Introduction

Figure 1.1: Examples of applications where motion planning is required.

such as self-driving cars, unmanned aerial vehicles and robotic manipulators. Two applications involving these type of systems are illustrated in Figure 1.1. Most of the successful motion planning methods for dynamical systems are sampling-based, because little can be done using combinatorial methods when constraints on the motion of the system in terms of differential equations are present [45]. The main strength of sampling-based approaches to motion planning problems are that they are good at solving combinatorial aspects of the problem, such as selecting which side to pass obstacles. Furthermore, they can be designed such that they are compatible with any representation of the surrounding environment [49]. The main drawback with these methods is that they suffer from the curse of dimensionality as the dimension of the search space to explore grows.

In this thesis, the objective is not to find any feasible motion plan between the initial and terminal state. Instead, the objective is to find solutions that also optimize some performance measure. Some measures of practical relevance are to find minimum distance or time solutions, a solution as far away from obstacles as possible, to minimize the total energy consumption for the system, or a measure that combines some or all of them. When a performance measure is included in the motion planning problem formulation for a dynamical system, the term optimal motion planning problem is commonly used [45]. From a control perspective, the optimal motion planning problem is an instance of an optimal control problem.

In optimal control theory, the fields of optimization and automatic control are combined to compute optimal control inputs to the system that is being controlled. It dates back to the development of calculus of variations, but regained interest during the 1960’s when it was successfully used for planning optimal trajectories in aerospace applications [76]. There exist many different methods to solve an optimal control problem, where perhaps the most common one is to use numerical optimization techniques. Many of these methods seek only a local solution, i.e., where the performance measure is lower than for any other feasible nearby solution. The main reason is that a global solution is usually both difficult to characterize and to locate [59], and hence intractable to find in reasonable time. In this thesis, we limit the search for a solution to locally optimal solutions of practical relevance.

In recent years, it has been increasingly popular to apply algorithms from numerical optimal control to compute optimized trajectories online such as in [54, 77, 87], or in a receding horizon fashion such as the work presented in [22, 31, 64, 74]. The increased interest is mainly due to the development of efficient and reliable
optimization algorithms, increased computational resources and the ability to system-
atically encode problem constraints such as restrictions on how the system can move and actuator limitations within an optimal control framework [53]. The main difficulty with applying these methods to motion planning problems is that the resulting problem formulation usually becomes a challenging nonconvex optimization problem. The performance of an optimization algorithm for solving such problems is strongly dependent on the provided initialization, i.e., where the algorithm starts the search for a solution [59]. Hence, there is a significant risk of computing a solution which is bad from a practical point of view, or not being able to find a solution at all, if the algorithms are naively applied to the motion planning problem using standard initialization strategies.

The focus in this thesis is to further develop optimal control algorithms such that they can reliably be applied on challenging motion planning problems. In this work, this is achieved by combining recent ideas from automatic control, numerical optimization and robotics.

1.2 Publications and contributions

The contributions in this thesis are presented in chapters 3, 4 and 5 and are based on published and unpublished material. The content in Chapter 3 is based on the publication:


This paper received the best application paper award at the conference. This contribution is a systematic approach for solving motion planning problems, where a homotopy method and numerical optimal control are combined to solve motion planning problems.

In Chapter 4, a framework for automatic generation of motion primitives for lattice-based motion planners is proposed. Given a family of systems, the user only needs to specify which principle types of motions that are relevant for the considered system family, which can then be reused for all system instances. The content is based on the publication:

Kristoffer Bergman, Oskar Ljungqvist, and Daniel Axehill. Improved optimization of motion primitives for motion planning in state lattices. Accepted for publication at the 2019 IEEE Intelligent Vehicles Symposium (IV), June 2019.

The chapter also contains unpublished material that extends the proposed framework for simultaneous optimization of both motion primitives and discretization parameters. These parameters are used by the lattice-based motion planner to define the state-space discretization. The proposed extension enables an optimized selection of the discretization parameters for a specific system instance.
Finally, Chapter 5 presents a unified optimization-based path planning approach, where a lattice-based path planner and numerical optimal control are tightly combined to solve advanced path planning problems. The material is mainly from the manuscript:


In the contributions listed in this section, the author of this thesis has driven the research, carried out the evaluations and done the main part of the writing. The co-authors have contributed with research ideas, technical discussions and improving the manuscripts.

### 1.3 Thesis outline

The remainder of this thesis is organized as follows:

- In Chapter 2, relevant theoretical background material is presented including nonlinear optimization, optimal control and motion planning. The main purpose with this chapter is to provide a theoretical foundation to the contributions presented in chapters 3, 4 and 5.

- In Chapter 3, a systematic approach for computing local solutions to motion planning problems by combining homotopy methods and numerical optimal control is proposed. The approach is demonstrated in motion planning problems in challenging 2D and 3D environments, where the proposed approach outperforms both a state-of-the-art numerical optimal control method and a state-of-the-art optimizing sampling-based planner based on random sampling.

- Chapter 4 presents a framework for automatic computation of motion primitives for lattice-based motion planners based on user-defined principle types of motions.

- Chapter 5 presents a unified optimization-based path planning approach to efficiently compute locally optimal solutions to advanced path planning problems. This is done by tightly integrating a lattice-based path planner and numerical optimal control.

- Finally, the thesis is concluded in Chapter 6, where also future research ideas are presented.
This chapter aims at providing the theoretical background required for the remainder of this thesis. First, some background in nonlinear optimization is presented, which forms the foundation of many numerical optimal control methods. Second, a general continuous-time optimal control problem (OCP) is posed and some commonly used solution strategies are discussed with a focus on direct methods. Finally, the relation between an OCP and an optimal motion planning problem is discussed, together with some commonly used solution strategies within the field of motion planning.

2.1 Nonlinear optimization

Nonlinear optimization algorithms are the basis of many numerical optimal control methods. This section contains a brief overview of the field and introduces some of the most important concepts in the area. The material in this section is inspired by [59] and [15], which are extensive references to nonlinear and convex optimization.

2.1.1 Nonlinear programming problem

Consider a nonlinear optimization problem given in the form:

\[
\begin{align*}
\text{minimize} & \quad L(z) \\
\text{subject to} & \quad f_i(z) \leq 0, \quad i \in Z_{1,p} \\
& \quad g_j(z) = 0, \quad j \in Z_{1,m},
\end{align*}
\]

where \( z \in \mathbb{R}^n \) is the optimization variable, \( L : \mathbb{R}^n \rightarrow \mathbb{R} \) is the objective function, and \( f_i : \mathbb{R}^n \rightarrow \mathbb{R}, \ i \in Z_{1,p} \) and \( g_j : \mathbb{R}^n \rightarrow \mathbb{R}, \ j \in Z_{1,m} \) are smooth functions.
representing the inequality and equality constraint functions, respectively. This type of optimization problem is usually referred to as a nonlinear programming (NLP) problem. An important subclass of optimization problems in the form of (2.1) are convex optimization problems, which all have a number of special properties. They are used in many areas such as automatic control systems, estimation and signal processing, statistics and finance [15]. Furthermore, they are also used within many algorithms for solving general NLPs [59]. Before defining a convex optimization problem, some definitions are required:

**Definition 2.1 (Feasible set).** The feasible set $\Omega$ is defined as the set 

$$\Omega := \{ z \in \mathbb{R}^n \mid f_i(z) \leq 0, \; i \in \mathbb{Z}_{1,p}; \; g_j(z) = 0, \; j \in \mathbb{Z}_{1,m} \}.$$  

**Definition 2.2 (Convex set).** A set $\Omega$ is convex if for any two points $z, x \in \Omega$ and any $\theta \in [0,1]$ it holds that 

$$\theta z + (1-\theta)x \in \Omega.$$  

**Definition 2.3 (Convex function).** A function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex function if its domain $D \subseteq \mathbb{R}^n$ is a convex set and if for all pairs of $z, x \in D$ and any $\theta \in [0,1]$ it holds that 

$$f(\theta z + (1-\theta)x) \leq \theta f(z) + (1-\theta)f(x).$$  

The optimization problem in (2.1) is referred to as a convex optimization problem, if the objective function $L$ and the inequality constraint functions $f_i$ are all convex functions and the equality constraint functions $g_j$ are affine [15].

### 2.1.2 Optimality conditions

In this section, first and second-order optimality conditions for $z^* \in \Omega$ to be a local minimizer of (2.1) are stated. To be able to pose these optimality conditions, the Lagrangian of the problem in (2.1) needs to be defined, which is given by 

$$\mathcal{L}(z, \lambda, \nu) \triangleq L(z) + \sum_{i=1}^{p} \lambda_i f_i(z) + \sum_{j=1}^{m} \nu_j g_j(z).$$  

The Lagrangian is an augmented representation of the objective function which contains weighted sums of the constraint functions. The weights are called Lagrange multipliers, or dual variables, and are given by $\lambda_i$ for the inequality constraint functions and $\nu_j$ for the equality constraint functions. Furthermore, the active set is defined as:

**Definition 2.4 (Active set).** The active set $\mathcal{A}(z)$ at a feasible point $z \in \Omega$ of (2.1) consists of the constraint indices corresponding to the constraints that hold with equality [59], i.e., 

$$\mathcal{A}(z) = \{ i \in \mathbb{Z}_{1,p}, \; j \in \mathbb{Z}_{1,m} \mid f_i(z) = 0, \; g_j(z) = 0 \}.$$
Finally, the linear independence constraint qualification (LICQ) is given by

**Definition 2.5 (Linear independence constraint qualification).** Given a point \( z \) and the corresponding active set \( A(z) \), the linear independence constraint qualification (LICQ) holds if the gradients of the active constraints,

\[
\nabla f_i(z), \ i \in A(z), \ \nabla g_j(z), \ j \in A(z),
\]

are linearly independent.

With these definitions, it is possible to pose the first-order optimality conditions as [59]:

**Theorem 2.1.** Suppose that \( z^* \) is a local solution to (2.1), that the functions \( L, f_i \) and \( g_j \) in (2.1) are continuously differentiable, and that LICQ holds at \( z^* \). Then, there exist Lagrange multiplier vectors \( \lambda^*, \nu^* \) with components \( \lambda^*_i \in \mathbb{Z}_{1,m} \) and \( \nu^*_j \in \mathbb{Z}_{1,p} \), such that the following conditions are satisfied at \((x^*, \lambda^*, \nu^*)\):

\[
\nabla_z L(z^*, \lambda^*, \nu^*) = 0, \\
f_i(z^*) \leq 0, \ i \in \mathbb{Z}_{1,p}, \\
g_j(z^*) = 0, \ j \in \mathbb{Z}_{1,m}, \\
\lambda^*_i \geq 0, \ i \in \mathbb{Z}_{1,p}, \\
\lambda^*_i f_i(z^*) = 0, \ i \in \mathbb{Z}_{1,p}.
\]

**Proof:** See Section 12.4 in [59].

The conditions in Theorem 2.1 are known as the Karush-Kuhn-Tucker (KKT) conditions and are necessary conditions for local optimality of (2.1). In the special case of a convex optimization problem, these conditions are not only necessary for a locally optimal solution, but also sufficient for a globally optimal solution. The conditions in (2.5e) are the complementarity conditions, which implies that either an inequality constraint is active, i.e., \( f_i(z^*) = 0 \), or that the corresponding Lagrange multiplier \( \lambda^*_i \) is zero, or possibly both [59]. The following special case of complementarity will be used later on in this thesis:

**Definition 2.6 (Strict complementarity).** Given a local solution \( z^* \) to (2.1) and Lagrange multipliers satisfying the KKT conditions (2.5), the strict complementarity condition is said to hold if exactly one of \( \lambda^*_i \) and \( f_i(z^*) \), \( i \in \mathbb{Z}_{1,p} \), is zero in (2.5e).

The following definition classifies the inequality constraints according to the value of the corresponding Lagrange multiplier:

**Definition 2.7 (Strongly active constraint).** Let \( z^* \) be an optimal solution to the optimization problem in (2.1), where the KKT conditions are satisfied with Lagrange multipliers \( \lambda^* \) associated with the inequality constraints \( f_i \), \( i \in \mathbb{Z}_{1,p} \). A constraint \( f_i(z) \) is then said to be strongly active if \( f_i(z^*) = 0 \) and \( \lambda^*_i > 0 \) [59].

To be able to pose the second-order optimality conditions, the set of linearized feasible directions is required. It is defined as:
Definition 2.8 (Linearized feasible directions). Given a point \( z \in \Omega \) and the corresponding active set \( A(z) \) (as defined in Definition 2.4), the set of linearized feasible directions \( F(z) \) is
\[
F(z) = \{ d \mid d^T \nabla g_j(z) = 0, \forall j \in A(z); d^T \nabla f_i(z) \leq 0, \forall i \in A(z) \}.
\]

The first-order optimality conditions in Theorem 2.1 tells us how the first-order derivatives of the objective function and the active constraint functions are related to each other at a solution \( z^* \). Hence, when the KKT conditions are satisfied, a step in any feasible direction \( w \in F(z^*) \) will result in either an increased or constant first-order approximation of the objective function value, i.e., \( w^T \nabla L(z^*) \geq 0 \).

For directions where \( w^T \nabla L(z^*) = 0 \), first-order information is not sufficient to determine if the objective function value decreases for a nonconvex optimization problem. With second-order information, it is sometimes possible to resolve this issue [59]. In the following theorem, sufficient second-order conditions are given to ensure that \( z^* \) is a local solution to (2.1):

Theorem 2.2. Suppose that for some feasible \( z^* \in \Omega \) of (2.1), there are Lagrange multipliers \( (\lambda^*, \nu^*) \) such that the KKT conditions in (2.5) are satisfied. Furthermore, suppose that
\[
w^T \nabla^2_{zz} L(z^*, \lambda^*, \nu^*) w > 0, \quad \forall w \in C(z^*, \lambda^*), \; w \neq 0.
\]
Then, \( z^* \) is a strict local solution to (2.1), i.e. there exists a neighborhood \( N \) of \( z^* \) such that \( L(z) > L(z^*) \) for all \( z \in N \cap \Omega, \; z \neq z^* \).

Proof: See Section 12.5 in [59].

In Theorem 2.2, \( C(z^*, \lambda^*) \) represents the critical cone which is defined as
\[
C(z^*, \lambda^*) \triangleq \{ w \in F(z^*) \mid w^T \nabla f_i(z^*) = 0, \forall i \in A(z^*) \text{ with } \lambda^*_i > 0 \}.
\]

From this definition and (2.5a) in the KKT conditions, it can be concluded that the critical cone is the set of directions from \( F(z^*) \) where first-order derivative information is insufficient to determine if the objective function value will increase or decrease [59].

2.1.3 Algorithms for solving nonlinear programs

In general, the solution to an NLP problem is found using iterative schemes. The major difference between different iterative solvers is how the search direction and step length are computed. When solving general NLP problems in the form of (2.1), there are mainly two families of optimization methods that are used: sequential quadratic programming (SQP) methods and nonlinear interior-point (IP) methods. They both aim at finding a (locally optimal) solution which satisfies the KKT conditions in Theorem 2.1, but approach the problem in different ways [59]. This section gives a brief introduction to these methods.
Sequential quadratic programming

One of the most commonly used methods for solving general NLPs are SQP methods, which are based on solving quadratic subproblems iteratively. In a basic SQP method, the search direction \( p_k \in \mathbb{R}^n \) is computed by approximating the NLP in (2.1) by a quadratic program (QP) at iterate \((z_k, \lambda_k, \nu_k)\). The iterate is given by the optimization variable \( z_k \) and the Lagrange multipliers \((\lambda_k, \nu_k)\) of the last iterate. By linearizing the NLP (2.1) at this iterate, the following QP is obtained [59]:

\[
\begin{align*}
\min_{p_k} & \quad \frac{1}{2} p_k^T B_k p_k + \nabla L(z_k)^T p_k \\
\text{subject to} & \quad \nabla f(z_k)^T p_k + f(z_k) \leq 0, \\
& \quad \nabla g(z_k)^T p_k + g(z_k) = 0.
\end{align*}
\]

Here, \( f : \mathbb{R}^n \to \mathbb{R}^p \) and \( g : \mathbb{R}^n \to \mathbb{R}^m \) are introduced to compactly represent the inequality and equality constraint functions from (2.1), respectively. \( B_k \) denotes the approximation of the Hessian of the Lagrangian to the NLP in (2.1). For an exact Newton method, the exact Hessian is used, i.e., \( B_k = \nabla^2_{zz} L_k \). The Jacobians of the inequality and equality constraint functions, respectively, are given by \( \nabla f \) and \( \nabla g \). They are defined as:

\[
\begin{align*}
\nabla g(z) &= [\nabla g_1(z), \nabla g_2(z), \ldots, \nabla g_m(z)], \\
\nabla f(z) &= [\nabla f_1(z), \nabla f_2(z), \ldots, \nabla f_p(z)].
\end{align*}
\]

Finally, \( \nabla L \) is the gradient of the objective function.

Due to the equivalence between SQP and Newton’s method, the search direction \( p_k \) can either be defined as the solution to the QP in (2.9), or as the search direction generated by Newton’s method for the corresponding NLP problem (2.1) for any fixed working set (i.e. the set of constraints that are imposed to hold with equality while all others are ignored) [59]. The following theorem assures under which conditions the SQP method is able to correctly identify the optimal active set:

**Theorem 2.3.** Suppose that \( z^* \) is a local solution to (2.1). At this point, the KKT conditions in Theorem 2.1 are satisfied for some Lagrange multipliers \((\lambda^*, \nu^*)\). Furthermore, suppose that LICQ (Definition 2.5), the strict complementary condition (Definition 2.6) and the second-order sufficient conditions (Theorem 2.2) hold at \((z^*, \lambda^*, \nu^*)\). Then, if the iterate \((z_k, \lambda_k, \nu_k)\) is sufficiently close to \((z^*, \lambda^*, \nu^*)\), there is a local solution to the SQP-subproblem (2.9) whose active set is the same as the active set \( A(z^*) \) of the NLP in (2.1).

**Proof:** See proof of Theorem 2.1 in [73].

Once the optimal active set is identified (and not changed in subsequent iterations), what remains is basically Newton’s method on a nonlinear system of equations with fast convergence to the solution. Furthermore, far from the solution, an SQP method is usually able to improve the estimate of the optimal active set and thus guide the iterates toward the solution. A major advantage with SQP methods based on active-set strategies is that they can be efficiently warm started.
if a good initial iterate is available. This can be useful if, e.g., a sequence of similar problems are to be solved [59].

**Nonlinear interior point**

Most nonlinear IP methods associate the problem of solving (2.1) with the following barrier problem:

$$\begin{align*}
\text{minimize} & \quad L(z) - \mu \sum_{i=1}^{p} \log(s_i) \\
\text{subject to} & \quad f_i(z) + s = 0, \quad i \in \mathbb{Z}_{1,p} \\
& \quad g_j(z) = 0, \quad j \in \mathbb{Z}_{1,m},
\end{align*}$$

(2.11)

where the inequality constraints from (2.1) are reformulated as equality constraints by introducing non-negative slack variables $s \succeq 0$ with $s \in \mathbb{R}^p$. The barrier parameter $\mu$ is given by a positive scalar, and $\log(\cdot)$ represents the natural logarithm function. The inequality constraints $s \succeq 0$ are not explicitly included in the problem since the additional term $\mu \sum \log s_i$ will force the components of $s$ from getting close to zero.

The solution to the barrier problem in (2.11) does not exactly coincide with the solution to the actual problem to be solved in (2.1), due to the modified objective function. The barrier approach instead aims at finding solutions to (2.11) for a sequence of positive barrier parameters $\{\mu_k\}$, with $\mu_k \to 0$ for increasing values of $k$. Hence, in the limit, the problem is equivalent to (2.1). The term interior point is derived from that early barrier methods did not introduce slack variables, but instead used the barrier function:

$$L(z) - \mu \sum_{i=0}^{p} \log(f_i(z)), \quad (2.12)$$

to prevent iterates from leaving the feasible region. However, this formulation requires an initial iterate $z_0$ that is feasible with respect to the inequality constraints. Most modern nonlinear IP methods can start from any initial iterate and remain interior only with respect to $s \succeq 0$ [59].

An advantage of using IP methods is that the combinatorial aspect of identifying the correct active set at the optimal solution is avoided. Hence, the linear system to be solved at each iteration always has the same structure and structure-exploiting techniques easily can be applied. Due to this property, IP methods usually outperform SQP methods on large-scale problems [59]. A drawback is that they are not warm-start friendly due to numerical sensitives for small perturbations on (2.11) when $\mu$ is close to zero [59].

**2.1.4 Globalization techniques**

In this section, a number of common approaches to increase the area of convergence are briefly outlined. It should not be confused with global optimization, which
aims at finding the global minimizer of a problem. To measure the progress of iterations, either merit functions or filter techniques are applied. The aim with these methods is to handle the trade-off between decreasing the objective function value and reducing the constraint violation. To ensure that progress is made in every iteration, either line-search or trust-region methods are used. The use of these methods is important for enabling NLP algorithms to converge not only from initializations close to a solution [59].

**Merit function**

A merit function \( \phi : \mathbb{R}^n \rightarrow \mathbb{R} \) is a function that produces a scalar measure of progress for the iterates computed during the search for a solution to (2.1). For an unconstrained optimization problem, the trivial choice of merit function is the objective function itself. When constraints are present, the merit function should also consider the constraint violation. One common choice of merit function is the \( \ell_1 \) penalty function given by:

\[
\phi(z) = L(z) + \sigma \sum_{i=1}^{p} [f_i(z)]^+ + \sigma \sum_{j=1}^{m} |g_j(z)|, \tag{2.13}
\]

where \([z]^+ = \max(0, z)\) [59]. The scalar \( \sigma \) represents the penalty parameter which determines the trade-off between the importance of reducing constraint violation or decreasing the objective function value. One important property of this choice of merit function is that it is an *exact merit function*, i.e., that there exists a large enough \( \sigma \) such that a local minimizer to (2.13) is also a local solution to the corresponding NLP in (2.1) [59]. Thus, in a standard descent procedure it is required that \( \phi(z_{k+1}) < \phi(z_k) \), otherwise the step is rejected or modified using either a line search procedure or a trust region, which are further described below in this section.

**Filter**

Filter techniques separate the goals of minimizing the objective function value and the constraint violation. By defining a measure of infeasibility according to:

\[
a(z) = \sum_{i=1}^{p} [f_i(z)]^+ + \sum_{j=1}^{m} |g_j(z)|, \tag{2.14}
\]

this separation can be written as the multi-objective optimization problem:

\[
\text{minimize } L(z) \quad \text{and minimize } a(z). \tag{2.15}
\]

A pair of the objective function value and measure of the constraint violation \((L(z_l), a(z_l))\) for a point \(z_l\) is said to *dominate* another pair \((L(z_k), a(z_k))\) if both \(L(z_l) \leq L(z_k)\) and \(a(z_l) \leq a(z_k)\). An iterate \(z_k\) is thus accepted by the filter if the pair \((L(z_k), a(z_k))\) is not dominated by any other pair that have previously been generated and added to the filter. If \(z_k\) is accepted, all pairs that are dominated by \(z_k\) are removed. To obtain good practical performance, several enhancements to this simple filter are needed, which are thoroughly described in [59].
Line search

An algorithm that employs a line-search strategy starts the solution procedure by computing a search direction $p_k$ using, e.g., one of the optimization algorithms described in Section 2.1.3. Then, the algorithm searches along this direction from the current iterate $z_k$ to a new iterate $z_{k+1} = z_k + \alpha_k p_k$ with a lower objective function (or merit function) value. Most line-search algorithms require that the search direction is a descent direction, which ensures that the objective function (or merit function) value can be reduced in this direction. Thus, the problem of computing a step length $\alpha_k \in \mathbb{R}$ becomes a one-dimensional optimization problem:

$$\min_{\alpha_k > 0} L(z_k + \alpha_k p_k). \quad (2.16)$$

Solving (2.16) exactly is usually too expensive. Instead, an inexact backtracking line-search procedure is commonly used. The procedure starts by trying to apply the full Newton step with $\alpha_k = 1$. If this is unsuccessful, $\alpha_k$ is iteratively reduced until a sufficient decrease in the objective function is obtained. One common approach to guarantee sufficient decrease is to ensure that the Wolfe conditions are satisfied for the candidate step length $\alpha_k$. The Wolfe conditions are satisfied when the following inequalities hold:

$$L(z_k + \alpha_k p_k) \leq L(z_k) + c_1 \alpha_k \nabla L(z_k)^T p_k,$$
$$\nabla (z_k + \alpha_k p_k)^T p_k \geq c_2 \nabla L(z_k)^T p_k, \quad (2.17)$$

where $c_1$ and $c_2$ are scalar parameters such that $0 < c_1 < c_2 < 1$ [59].

Trust region

In trust-region methods, an additional constraint is added to the optimization problem which limits the current step $p_k$ to a region close to the previous iterate $z_k$. In this region, the linearization of the problem at $z_k$ is a good approximation and the computed step can be trusted. Thus, in contrast to line-search methods, trust-region methods choose both search direction and step size simultaneously [59]. By using the QP-subproblem in (2.9) as an example, an SQP algorithm that employs a trust region instead solves the following optimization problem:

$$\min_{p_k} \frac{1}{2} p_k^T B_k p_k + \nabla L(z_k)^T p_k$$
subject to
$$\nabla f(z_k)^T p_k + f(z_k) \preceq 0,$$
$$\nabla g(z_k)^T p_k + g(z_k) = 0,$$
$$\|p_k\|_2 \leq \Delta_k, \quad (2.18)$$

where the scalar $\Delta_k > 0$ is called the trust-region radius. To obtain fast and reliable convergence, the trust-region radius $\Delta_k$ is typically modified after each iteration such that progress can be ensured [59].
2.2 Optimal control

The traditional optimal control problem is to compute an optimal control input to a dynamical system such that an objective function is minimized, subject to initial and terminal constraints [79]. By also enforcing the solution to satisfy certain inequality constraints, the problem is a constrained continuous-time optimal control problem, and is defined in its general form as:

$$\begin{align*}
\text{minimize} & \quad \Phi(x(t_f)) + \int_{t_i}^{t_f} \ell(x(t), u(t)) dt \\
\text{subject to} & \quad x(t_i) = x_{\text{init}}, \\
& \quad \dot{x}(t) = f(x(t), u(t)), \quad t \in [t_i, t_f] \\
& \quad c(x(t), u(t)) \leq 0, \quad t \in [t_i, t_f] \\
& \quad \Psi(x(t_f)) = x_{\text{term}},
\end{align*}$$

(2.19)

where $x_{\text{init}}$ and $x_{\text{term}}$ are the initial and terminal states, $x(\cdot) \in \mathbb{R}^n$ are the states, $u(\cdot) \in \mathbb{R}^m$ are the control inputs, $f: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ is the function that describes the system model, $c: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^p$ represents the path constraints and contains all the $p$ inequality constraint functions that determine the feasible region, $\Phi: \mathbb{R}^n \rightarrow \mathbb{R}$ and $\ell: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ form the objective function and $\Psi: \mathbb{R}^n \rightarrow \mathbb{R}^n$ is the terminal constraint function. The final time $t_f$ may, or may not, be an optimization variable [79]. For notational convenience, we use the term optimal control problem (OCP) when referring to a problem of this type further on in this thesis.

The solution to the OCP in (2.19) is a state and control trajectory $(x(t), u(t)), t \in [t_i, t_f]$ where the state trajectory is given by

$$x(t) = x(t_i) + \int_{t_i}^{t} f(x(\tau), u(\tau))d\tau, \quad t \in [t_i, t_f].$$

(2.20)

The OCP in (2.19) can be solved by applying any of the three approaches Hamilton-

![Figure 2.1: Different approaches to solve a constrained continuous-time optimal control problem [22].](image)
Jacobi-Bellman (HJB), indirect methods or direct methods [22] (as illustrated in Figure 2.1). In the first approach, the principle of optimality is used to obtain a partial differential equation in state space, denoted the HJB equation, which represents the continuous counterpart to the discrete-time dynamic programming approach. However, due to the curse of dimensionality, this approach is only applicable for small-scale problems [22]. Indirect methods use the necessary conditions of optimality of the OCP in (2.19), described by Pontryagin’s maximum principle (PMP), to obtain a boundary value problem (BVP) that is given by a set of ordinary differential equations (ODEs) [16]. In general, this BVP needs to be solved numerically, which makes it common to refer to this method as a “first optimize, then discretize” approach. Finally, direct methods solve the infinite OCP by first reformulating it into a finite NLP problem. After this reformulation, it is possible to use any of the methods described in Section 2.1 to solve the problem, making it a “first discretize, then optimize” approach [22].

In this thesis, direct methods for solving OCPs are used. The main reasons for this choice of method are that indirect formulations are often hard to solve in practice [13], and that there exist several software packages for solving the reformulated NLP problem with these methods, such as ACADO [35], IPOPT [83], SNOPT [32], WOHRP [17] and several more. The most common direct methods are briefly described in the following section. For a more detailed description of methods for solving OCPs using numerical optimal control techniques, see e.g. [13, 16, 69, 70].

2.2.1 Direct methods for optimal control

In direct methods for solving OCPs as the one in (2.19), the infinite decision variables are finitely parameterized such that it is possible to transform the problem in (2.19) into an NLP. In this section, a brief description of three well-known direct methods are given: direct single shooting, direct multiple shooting and direct collocation. The notation and material in this section is inspired by [22].

Direct single shooting

All shooting methods use an embedded ODE solver which makes it possible to rewrite, or even eliminate, the constraints of the continuous-time dynamics in (2.19) such that they can be handled by a general NLP solver. This is done by parameterizing the control signal \( u(t) \). One of the most widely used approaches is letting the control signal be constant on each phase of a uniform discretization of time with \( \{ t_k \}_{k=0}^{N} \), i.e., \( t_k = t_0 + k\Delta t \) and \( t_0 = t_i \), where \( \Delta t \) is an optimization variable. This parameterization can be written as:

\[
    u(t; z) = u_k, \quad t \in [t_k, t_{k+1}]
\]

with \( z = \{ u_0, u_1, \ldots, u_{N-1} \} \).

In direct single shooting the continuous-time dynamics constraints are removed by forward integration of the system, starting at the initial state \( x_{\text{init}} \) and applying
the parameterized control input in (2.21), which results in the state trajectory \( x(t; z) \). Thus, the OCP in (2.19) can be posed as:

\[
\begin{align*}
\minimize_{z, \Delta t} & \quad \Phi(x(t_N; z)) + \int_{t_i}^{t_f} \ell(x(t; z), u(t; z)) \, dt \\
\text{subject to} & \quad g(x(t_k; z), u(t_k; z)) \leq 0, \quad k \in \mathbb{Z}_{0,N} \\
& \quad \Psi(x(t_N; z)) = x_{\text{term}},
\end{align*}
\]

where both the state trajectory and the integral in the objective function can be computed using numerical integration techniques such as Runge-Kutta methods [13]. Hence, it is possible to use state-of-the-art ODE solvers with an adaptive step size for error control [22]. The problem is now in the form of a standard NLP and can be solved, e.g., using any of the methods in Section 2.1.3. The main drawbacks with this method are that the ODE solution \( x(t; z) \) can be severely non-linear with respect to \( z \), and that unstable systems are difficult to handle due to numerical sensitivities [22].

**Direct multiple shooting**

In direct multiple shooting, the problem is formulated as a collection of \( N \) phases. If we, as in the case of single shooting, assume piecewise constant control signals \( u_k \), the ODE is here separately solved on each interval, starting from an artificial initial state value \( s_k \)

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

This produces trajectory sequences \( x_k(t; s_k, u_k) \) for each phase, where the semicolon indicates that the trajectories are determined by the initial state values \( s_k \) and the constant control signals \( u_k \). The NLP decision variables are thus given by \( z = \{ s_0, u_0, s_1, u_1, \ldots, s_N \} \) [22], which gives the following NLP formulation:

\[
\begin{align*}
\minimize_{z, \Delta t} & \quad \Phi(s_N) + \sum_{k=0}^{N-1} \ell_k(s_k, u_k) \\
\text{subject to} & \quad s_0 = x_{\text{init}}, \\
& \quad s_{k+1} = f_k(s_k, u_k), \quad k \in \mathbb{Z}_{0,N-1} \\
& \quad g(s_k, u_k) \leq 0, \quad k \in \mathbb{Z}_{0,N} \\
& \quad \Psi(s_N) = x_{\text{term}}.
\end{align*}
\]

Here, the constraints \( s_{k+1} = f_k(s_k, u_k) \triangleq x_k(t_{k+1}; s_k, u_k) \) are the so-called continuity constraints that connect the subsequent phases. The objective function is here given by a sum of the numerically computed integrals for each phase [22]. One obvious difference in multiple shooting compared to single shooting is that the dimension of the problem increases. However, sparsity in the direct multiple
shooting formulation can be utilized for faster computations. Some of the benefits
of using direct multiple shooting instead of single shooting are that it is possible
to initialize the state trajectory, it is more numerically stable such that unstable
systems can be better handled and that it can robustly handle path and terminal
constraints [13].

Direct collocation

Another widely used approach is the direct collocation method. In this method,
the OCP is, similarly as in the shooting methods, discretized on a grid \( \{t_k\}_{k=0}^N \). Here, each interval \([t_k, t_{k+1}]\), \( t_{k+1} - t_k = \Delta t \) is denoted a collocation interval. On
each collocation interval, \( m \) collocation points \( \{t_k^{(1)}, \ldots, t_k^{(m)}\} \) are chosen and the
state trajectory is approximated by a polynomial:

\[
p_k(t; v_k) = \sum_{i=0}^{m} v_{k,i} t^i,
\]

where \( v_k \) is the coefficient vector. At the collocation points, equality constraints are
added to ensure that the derivative of the polynomial approximation of the state
trajectory coincides with the function \( f(x, u) \) that describes the system model. By
again representing the initial state values with \( s_k \), and assuming that the control
signal \( u_k \) is constant on each collocation interval, these constraints are given by:

\[
s_k = p_k(t_k; v_k) = \sum_{i=0}^{m} v_{k,i} t^i,
\]

\[
f(p_k(t_k^{(1)}; v_k), u_k) = p_k'(t_k^{(1)}; v_k) = \sum_{i=1}^{m} iv_{k,i} (t_k^{(1)})^{i-1},
\]

\[
\vdots
\]

\[
f(p_k(t_k^{(m)}; v_k), u_k) = p_k'(t_k^{(m)}; v_k) = \sum_{i=1}^{m} iv_{k,i} (t_k^{(m)})^{i-1}.
\]

By letting the constraints on each collocation interval in (2.26) be compactly rep-
resented as \( b_k(s_k, v_k, u_k) = 0 \), and the decision variables be represented by

\[
z = \{s_0, v_0, u_0, \ldots, s_{N-1}, v_{N-1}, u_{N-1}, s_N\},
\]

the following NLP is obtained:

\[
\begin{align*}
\text{minimize} & \quad \Phi(s_N) + \sum_{k=0}^{N-1} \ell_k(s_k, v_k, u_k) \\
\text{subject to} & \quad s_0 = x_{\text{init}}, \\
& \quad s_{k+1} = p_k(t_{k+1}; v_k), \quad k \in \mathbb{Z}_{0,N-1} \\
& \quad b_k(s_k, v_k, u_k) = 0, \quad k \in \mathbb{Z}_{0,N-1} \\
& \quad g(s_k, u_k) \leq 0, \quad k \in \mathbb{Z}_{0,N} \\
& \quad \Psi(s_N) = x_{\text{term}}.
\end{align*}
\]
where $\ell_k(s_k, v_k, u_k)$ denotes the numerical approximation of the objective function on interval $k$. The obtained NLP is a large, sparse problem where sparsity exploiting solvers can be used for faster computations. The major drawback is that the NLP needs to be re-gridded if adaptive discretization error control should be used. This means that the dimension of the NLP is required to be changed, which is not the case for single and multiple shooting since they can use adaptive ODE solvers [22].

### 2.3 Optimal motion planning

Depending on the field of subject and the applications, the meaning of the term *optimal motion planning* may vary [45]. In this thesis, the term will be used to denote the problem of finding a feasible and locally optimal trajectory for a nonlinear system from an initial state to a terminal state in an environment that includes obstacles. Hence, this is a quite broad notion which includes several applications such as finding trajectories for autonomous cars, unmanned aerial vehicles, autonomous underwater vehicles and trajectories for robotic manipulators. In the literature, there is a vast variety of different methods that can be used to solve motion planning problems and produce feasible or optimized trajectories [43, 45, 62]. Before we pose the optimal motion planning problem, some commonly used terms within nonlinear system modeling are discussed.

#### 2.3.1 Nonlinear system models

The function $f(x, u)$ that describes the system model in (2.19) can have several different properties which influence the difficulty of finding a solution to the motion planning problem. In this section, some commonly used system model characteristics in motion planning for nonlinear systems are briefly discussed.

**Dynamics and kinematics**

Kinematics is the branch of classical mechanics which describes the motion of a system without considering the forces and accelerations that cause a motion [56]. Hence, a system model is in this thesis denoted to be *kinematic* if it is derived only based on velocity constraints. On the other hand, the field of dynamics studies how applied forces and torques affect the motion of a system, such as Newton’s second law of motion [56]. Hence, it is reasonable that a dynamic model should consider these forces and torques. However, in motion planning, such as the definition of *kinodynamic motion planning* [23], it is also a commonly used term to denote that limitations on higher order derivatives, such as accelerations, are considered in the motion planning phase [45]. Using this definition, a system model is denoted *dynamic* in this thesis if any type higher order derivatives are considered in the system model.
Holonomic or nonholonomic

Holonomic and nonholonomic properties are related to what type of constraints on the velocity of the system that are present. A constraint is said to be holonomic (or integrable) if it restricts the attainable states of the system to a smooth hypersurface, and otherwise the constraint is said to be nonholonomic. This implies that holonomic constraints can be converted to constraints that do not involve any time derivatives of the system states [56]. One example of a nonholonomic system is a car. The car is unable to move sideways or rotate in place, but it is well known that it is possible to park a car at any position and orientation. Thus, the constraints imposed on the car are nonholonomic since the system is able to attain any feasible state, even though the motion is locally restricted. For a detailed description of holonomic and nonholonomic properties, the reader is referred to [56]. Motion planning for systems that are subject to nonholonomic constraints are often complicated, even in environments without obstacles, due to the locally restricted motion capability [47].

2.3.2 Problem formulation

The optimal motion planning problem is to find a feasible trajectory that moves the system from its initial state to a desired terminal state, while not colliding with any obstacles and minimizes a specified performance measure. From a control perspective, the problem can be formulated as an OCP according to:

\[
\begin{align*}
\text{minimize} & \quad J = \int_{t_i}^{t_f} \ell(x(t), u(t))dt \\
\text{subject to} & \quad x(t_i) = x_{\text{init}}, \quad x(t_f) = x_{\text{term}} \\
& \quad \dot{x}(t) = f(x(t), u(t)), \quad t \in [t_i, t_f] \\
& \quad x(t) \in \mathcal{X}_{\text{free}}(t) \cap \mathcal{X}_{\text{valid}}, \quad t \in [t_i, t_f] \\
& \quad u(t) \in \mathcal{U}, \quad t \in [t_i, t_f],
\end{align*}
\]

(2.29)

where \(x_{\text{init}}\) and \(x_{\text{term}}\) are the initial and terminal states, \(t_f\) the time when the terminal state is reached, \(\mathcal{X}_{\text{free}}(t)\) describes the obstacle free region, while \(\mathcal{X}_{\text{valid}}\) and \(\mathcal{U}\) describe the physical constraints on the states and controls, which typically are described by convex sets. Furthermore, it is assumed that \(x_{\text{init}} \in \mathcal{X}_{\text{free}} \cap \mathcal{X}_{\text{valid}}\) and \(x_{\text{term}} \in \mathcal{X}_{\text{free}} \cap \mathcal{X}_{\text{valid}}\) such that (2.29) is well defined, and that there exists a terminal control signal \(u(t_f) = u_{t_f}\) such that \(f(x(t), u_{t_f}) = 0, \ t \geq t_f\). In some applications, the equality constraint for the terminal state is represented by a terminal region \(x(t_f) \in \mathcal{X}_{\text{term}}\) [45], but in this thesis the focus is on problems where it is desired to reach the terminal state exactly.

Since the motion planning problem (2.29) is an OCP, it can be solved using numerical optimal control techniques described in Section 2.2. The advantages of using numerical optimal control are that state and control constraints can easily be incorporated in the problem formulation, and that it is straightforward to change the nonlinear model of the system, the performance measure (i.e objective
function), and to define and update problem parameters. These methods have in recent years been increasingly popular for real-time motion planning online due to increased computational resources and the development of robust nonlinear optimization algorithms [86].

The main challenge with optimization-based approaches for solving (2.29) is that the constraints representing the obstacle free region make the problem non-convex, and they introduce the combinatorial aspect of selecting which side to pass an obstacle. Hence, there usually exist several possible classes of solutions which make the choice of initialization strategy crucial for a reliable and fast convergence. Another challenge to handle is the nonlinear system model, which further complicates the problem. This combination makes the motion planning OCP (2.29) in general hard to solve by directly invoking a numerical optimal control solver. Instead, motion planning algorithms are often used to compute a suboptimal solution [45]. In the following sections, some common motion planning approaches are discussed.

2.3.3 Sequential solution concept

To compute a suboptimal solution to the problem in (2.29), many motion planning algorithms divide the problem into three phases: path planning, path optimization and velocity planning [45] which are solved sequentially. The inputs to the motion planner are a terminal state provided by a higher-level task planner and a description of the environment provided by the perception and localization subsystem. Based on these inputs, the motion planner computes a motion plan following the three steps illustrated in Figure 2.2.
The path planning step consists of computing a collision-free path from the initial state to the terminal state. By introducing the path parameter $s \geq 0$, where $s$ represents the progression along the path, this step traditionally aims at computing a path $x(s)$ using a classical planning algorithm. In this step, the constraints on the motion of the system are usually neglected and instead handled in later steps. In the second step, the path from the path planning step is transformed, or smoothened, such that the path satisfies constraints on the system based on, e.g., wheel motions for a nonholonomic vehicle [45]. In the path planning and optimization steps, dynamic obstacles are disregarded since time is not considered in these steps. The final step is performed by the velocity planner. Since the direction of motion is already decided by the previous steps, what remains for the velocity planner is to compute a velocity profile $\dot{s}(t) = |v(t)|$ along each path segment. It is computed such that the system does not collide with any dynamic obstacles while minimizing some performance measure, e.g., related to comfort or smoothness, and also respecting velocity and acceleration constraints. The relation between a path and a trajectory can thus be described as follows

**Definition 2.9 (Path trajectory relation).** A path is represented as $(x(s), u(s)), s \in [0, s_g]$ and a trajectory is a time-parameterized path $(x(s(t)), u(s(t))), t \in [t_i, t_f]$. 

In the literature, several approaches to solve these steps have been suggested. One common approach is to use B-splines or Bezier curves to compute smooth paths for differentially flat systems, either to smoothen a sequence of waypoints, for example generated by a classical path planner [65, 85] or as steering functions within a sampling-based motion planner [7, 60]. The use of these methods are computationally efficient since the model of the system can be described analytically. However, these methods are not applicable to non-flat systems, such as, e.g., many truck and trailer systems [75]. Furthermore, it is difficult to optimize the path with respect to a general performance measure. Another popular method, which is used in this thesis, is to formulate the problem as an OCP and use numerical optimal control techniques to transform the problem to an NLP. Due to non-convex constraints introduced by obstacles and the nonlinear system model, a proper initialization strategy is crucial to converge to a good local optimum [59]. A straightforward initialization strategy is to use linear interpolation between the initial state and terminal state [70], this can however often lead to convergence issues in cluttered environments, which is shown in both Chapter 3 and [87]. A more sophisticated initialization strategy is to use the solution from a sampling-based path planner, for which the path planner in Figure 2.2 aims at finding an initialization from where the NLP solver can converge to a locally optimal solution. In previous work in the literature, the vehicle models used by the path planner for initialization are simplified; either they disregard the system dynamics completely (classical motion planning) [19, 45], or partially (respecting only kinematic constraints) [5, 87]. The use of a classical motion planning initialization strategy is shown in [86] to cause problems for vehicles with nonholonomic constraints (illustrated here in Figure 2.3). Initializations based on a kinematic model will in general be infeasible in the actual problem to be solved, and potentially not homotopic to a feasible solution which will make it impossible for the NLP solver
2.3 Optimal motion planning

Figure 2.3: Feasibility issues. The solution from a path planner based on a simplified model that disregard the system dynamics completely (dashed yellow) provides an initialization from where it is impossible for an OCP solver to find a feasible solution. On the other hand, an initialization based on the full system model (dashed blue) enables reliable convergence to a locally optimal solution (blue).

to converge to a feasible solution [80]. Furthermore, the objective function in the sampling-based planner can only consider states that are represented in the chosen simplified model description, which might cause it to find an initialization far from a local minimum.

Ideally, a unified motion planning algorithm that handles all of the involved steps simultaneously is desirable, since the sequential approach often leads to sub-optimal solutions [45]. In Chapter 3, a unified approach to the problem is proposed where a systematic initialization procedure for the NLP solver is presented. In Chapter 5, we tightly couple the path planning and path optimization steps by integrating a deterministic sampling-based path planner and numerical optimal control for fast and reliable convergence to a solution. In the next section, sampling-based motion planners are further described.

2.3.4 Sampling-based motion planning

One commonly used approach for solving motion planning problems is to apply sampling-based motion planners. In this section, a general framework is outlined which corresponds to a generalization of sampling-based algorithms used for classical motion planning where the system dynamics and kinematics are neglected [45]. In general, the sampling-based algorithms perform the search for a solution to (2.29) by constructing a directed graph $G = (V,E)$ incrementally online, where a vertex $x_k \in V$ represents a state of the system and an edge $e_k \in E$ represents a motion that satisfies all the constraints imposed on the system given in (2.29). The involved steps are summarized in Algorithm 2.1, where several iterations may have to be made to complete a step.
Algorithm 2.1: Sampling-based motion planning

1: **Initialization**: Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent a directed graph, where $\mathcal{V}$ contains the initial state $x_{\text{init}}$ and $\mathcal{E}$ is empty.

2: **Expansion**: Choose a vertex $x_k \in \mathcal{V}$ to expand and select a desired system state $x_{k+1} \in \mathcal{X}_{\text{free}}(t) \cap \mathcal{X}_{\text{valid}}$ to connect with $x_k$.

3: **Steering function**: Try to compute a feasible motion $e_k$ between $x_k$ and $x_{k+1}$. If this stage fails, return to Step 2.

4: **Insertion**: Insert $x_{k+1}$ in $\mathcal{V}$ and $e_k$ in $\mathcal{E}$ if $x_{k+1} \in \mathcal{V}$ and the current cost-to-come to $x_{k+1}$ is higher than the candidate cost. Otherwise, return to Step 2.

5: **Solution check**: If $x_{k+1} = x_{\text{term}}$, mark the candidate trajectory from $x_{\text{init}}$ to $x_{\text{term}}$ as a feasible solution to the problem.

6: **Termination check**: If a termination condition is satisfied, return the solution with the lowest cost. Otherwise, return to Step 2.

The major difference in Algorithm 2.1 compared to classical sampling-based motion planning algorithms is that a steering function (Step 3) is required to be able to compute a feasible motion between two nearby states. Hence, the output from the steering function is a feasible motion $e_k$ from $x_k$ to $x_{k+1}$ with an associated cost $J(e_k)$. This cost is used in the graph search to compute the cost to reach states from $x_{\text{init}}$ and is in the end used to determine the optimal solution from $x_{\text{init}}$ to $x_{\text{term}}$. For general nonlinear systems with general performance measures, the steering function requires the solution of an OCP in the form of (2.29) for each pair of states to connect. This subproblem is often easier to solve than the complete problem since the distance between $x_k$ and $x_{k+1}$ is typically much shorter than from $x_{\text{init}}$ to $x_{\text{term}}$. Furthermore, many steering functions neglect the collision avoidance constraints and instead rely on a separate collision avoidance step [45] which further simplifies the OCP to be solved. This separation makes the sampling-based motion planning algorithms compatible with any representation of the environment, but unnecessary computations is likely to occur for computing solutions that are in collision with obstacles.

For linear systems without any other imposed constraints on the state and control signal, PMP can be applied to derive analytical solutions to the steering function OCPs [84]. Analytical solutions also exist for some nonlinear systems, such as shortest path solutions for car-like vehicles based on Dubins [24] or Reeds-Shepp [72] car models. Recently, these results have been extended to account for higher-order differentiability on the path curvatures such as the work in [29, 61]. Furthermore, there exists analytical solutions for differentially flat systems, such as unmanned aerial vehicles [54], where it is possible to compute polynomial functions in the flat outputs to generate candidate solutions between two states [45]. However, it is with these methods difficult to optimize a general performance measure, and any path constraint imposed on the system can be hard to encode when computing a solution.
The order in which states $x_k$ is selected for expansion in Step 2 of Algorithm 2.1 is a critical choice for the performance of a sampling-based motion planner [45]. Usually, a heuristic function $h(x_k, x_{term})$ is used to approximate the cost-to-go from $x_k \in \mathcal{V}$. This value, together with the cost-to-come from the initial state $x_{init}$ to $x_k$ guides the expansion step to select the most promising candidate to expand. If the heuristic function is a good approximation of the actual optimal cost-to-go from $x_k$ to $x_{term}$, here denoted $h^*(x_k, x_{term})$, the search conducted in Algorithm 2.1 will be able to find a solution faster [33]. To ensure that the algorithms are able to maintain optimality guarantees, the heuristic function needs to satisfy some properties. Let $J(e_k)$ denote the cost of a feasible motion $e_k$ from $x_k$ to $x_{k+1}$. Then, optimality guarantees are maintained if the heuristic function $h$ has the following properties [41]:

1. **Admissible:** $h(x_k, x_{term}) \leq h^*(x_k, x_{term}), x_k \in \mathcal{V}$.

2. **Consistent:** $h(x_k, x_{term}) \leq J(e_k) + h(x_{k+1}, x_{term}), x_k \in \mathcal{V}$ with successor state $x_{k+1} \in \mathcal{V}$.

Most admissible heuristic functions are also consistent. Furthermore, an inconsistent but admissible heuristic function can be transformed into a consistent heuristic function which is a more accurate approximation than the original one [41].

The two selections of which state to expand from, $x_k$, and the desired state to connect to, $x_{k+1}$, are done by either randomized sampling or deterministic sampling strategies [45]. Randomized sampling methods draw random samples of $x_{k+1}$ from $\mathcal{X}_{free} \cap \mathcal{X}_{valid}$ and then finds the nearest neighbor $x_k \in \mathcal{V}$. A steering function is then used to connect these states. A randomized sampling strategy typically leads to tree-shaped search spaces [45]. Many sampling-based motion planning algorithms with random sampling are based on the standard Rapidly exploring Random Tree (rrt) algorithm [44], which was originally developed for solving motion planning problems without considering system dynamics or kinematics. One example where this algorithm has been modified to account for a system model is rrt* for dynamical systems [38, 80]. In these methods, a rewiring step of the search tree is added such that asymptotic optimality guarantees can be obtained [38]. A major drawback of these methods is that an OCP is required to be solved between each pair of states that are selected for connection online, a process which for a general nonlinear system can be computationally demanding or even intractable. In [42], RRT is instead used to sample the input to a closed-loop system with an underlying controller (CL-rrt), i.e., a controller is acting as the steering function which removes the need of using an OCP solver online. Furthermore, the CL-rrt algorithm can reliably be applied to unstable systems, where the controller is used to stabilize the system [26]. On the downside, no optimization of the search tree by rewiring (as used in rrt*) can be performed with CL-rrt since states cannot be connected exactly with this choice of inexact steering function. Furthermore, the use of an inexact steering function also limits the CL-rrt algorithm to only being able to find solutions within a terminal region, i.e., $x_{term} \in \mathcal{X}_{term}$. For a more detailed description of sampling-based motion planning with random sampling, the reader is referred to the RRT related references [38, 42, 44, 80] and [25, 50].
In this thesis, we mainly use a deterministic sampling strategy, in terms of a lattice-based motion planner. This type of motion planner and the reason for choosing it is further described in the next section.

2.3.5 Lattice-based motion planning

The key idea with a lattice-based motion planner is to restrict the control to a discrete subset of the valid actions and hence transform the optimal motion planning problem in (2.29) to a discrete graph search problem [66]. There are mainly two different approaches that are used to generate a lattice for motion planning [37]:

- Control-sampling: The control space is sampled in a way such that the resulting sampling in the state space has desired properties in terms of discrepancy and dispersion. This typically leads to tree-shaped search spaces. This strategy relies on forward simulations of the system online.

- State lattice: First, a desired state-space discretization \( X_d \) is selected. Then, an OCP solver is used offline to connect several states in the discretized state space.

In this thesis, the state-lattice methodology will be used. The state-lattice formulation has mainly been used for position invariant systems operating in unstructured environments. The benefits of using a state lattice are that it is possible to design the state-space discretization depending on the application, and the potentially complex relation of how the control signal affects the states of the system is handled offline by the OCP solver. The latter also means reduced online computations since no forward simulations of the system are needed during online planning [67]. It is also much easier to create regular lattices, which will cover a larger volume of the search space in fewer samples [45]. The use of a regular state lattice will lead to that the graph \( G = (\mathcal{V}, \mathcal{E}) \) in Algorithm 2.1 becomes a cyclic graph, since many combinations of edges \( e \in \mathcal{E} \) will arrive to the same state \( x_k \in \mathcal{V} \). This enables the possibility of using cyclic graph search methods such as bidirectional search and exact pruning for a faster online planning [67].

State-lattice construction

The aim of the state-lattice construction is to compute a motion primitive set \( \mathcal{P} \) which contains feasible motions for the system between different discretized system states \( x_d \in X_d \) that can be used online to solve motion planning problems. Thus, the motion primitives act as precomputed steering functions within a sampling-based motion planner outlined in Algorithm 2.1, where the successor states are deterministically sampled offline. The state-lattice construction procedure can be divided into three different steps which are all performed offline [68] (illustrated in Figure 2.4):
2.3 Optimal motion planning

Figure 2.4: An illustration of the three steps in the state-lattice construction.
(1) Discretize the state space, (2) select connectivity, (3) solve resulting OCPs.

1. Discretize the state space to obtain a discrete search space $X_d$.

2. Select which pairs of states to connect in the discretized representation.

3. Solve the OCPs defined in the previous step to compute the motion primitive set $\mathcal{P}$.

The first step is to decide how the state space should be discretized, i.e., how to choose $X_d$. This is done by selecting which fidelity of the state space that should be used in the search space for the planner. As an example for car-like vehicles, the first step could be to determine the fidelity of the position, orientation and steering angle of the vehicle [68].

The second step is to select which pairs of states to connect in the discretized search space. If the system is position invariant, it is only required to compute motion primitives from states with a position in the origin. These motion primitives can then be translated and reused at all positions on the grid [68]. Furthermore, if the system is orientation invariant, rotational symmetries of the system can be exploited such that it is only required to compute motion primitives from a few initial headings. These motions can then be mirrored and/or rotated to the other initial headings to obtain the full motion primitive set [66]. Ideally, all possible initial states with a position in the origin to all terminal states in the discrete state space should be connected directly. However, this is intractable in practice since the required amount of memory usage, offline computation and online search time (with obstacles present) would be intractable. Instead, a smaller number of connections are chosen, such that the resulting reachability graph $\mathcal{G}_r(x_{\text{init}}, \mathcal{P})$...
Background

(which encodes the set of all possible trajectories from \(x_{\text{init}}\) given \(P\)) sufficiently well resembles the true forward reachable set of the system [45].

The procedure of how to select which states to connect is application dependent; systems that are more agile or systems working in environments with a lot of obstacles should have a dense connectivity selection. However, it is also a trade-off between optimality and online computation time, since the number of motion primitives defines the branching factor online, i.e., the number of edges that are explored in the graph for each node expansion. Hence, the complexity of the graph search problem to be solved online scales exponentially in the number of applicable motion primitives \(P(x)\) from each discrete state \(x \in X_d\) [68].

Finally, the OCPs defined by the second step needs to be solved to obtain the motion primitive set. In this thesis, a motion primitive \(m \in P\) is defined as:

\[
m = (x(t), u(t)) \in X_{\text{valid}} \times U, \quad t \in [0, T],
\]

and represents a feasible trajectory that moves the system from an initial state \(x(0) = x_{\text{start}} \in X_d\) to a terminal state \(x(T) = x_{\text{final}} \in X_d\) by applying the control inputs \(u(\cdot)\), where \(x_{\text{start}} \in X_d\) and \(x_{\text{final}} \in X_d\) are given by the connectivity selection in the previous step. The motion primitives are computed by solving the following OCP:

\[
\begin{align*}
\text{minimize} & \quad \int_{t=0}^{T} \ell(x(t), u(t))dt \\
\text{subject to} & \quad x(0) = x_{\text{start}}, \quad x(T) = x_{\text{final}}, \\
& \quad \dot{x}(t) = f(x(t), u(t)), \quad t \in [0, T] \\
& \quad x(t) \in X_{\text{valid}}, \quad t \in [0, T] \\
& \quad u(t) \in U, \quad t \in [0, T],
\end{align*}
\]

for each combination of \(x_{\text{start}}\) and \(x_{\text{final}}\) from the connectivity step. Since these calculations are performed offline, the constraints representing the obstacles, \(x(t) \in X_{\text{free}}(t)\) in (2.29) are so far unknown and hence disregarded in (2.31). Instead, they are considered during online planning. The cost function can be chosen as any smooth function of the states and control signals. For autonomous vehicles, one function that is commonly used is:

\[
\ell(x(t), u(t)) = 1 + ||[x(t), u(t)]||_Q^2,
\]

where the weight matrix \(Q \succeq 0\) is used to determine the trade-off between time duration (captured by the first term in (2.32)) and other measures such as smoothness of a motion [50].

One approach to solve the OCPs given in (2.31) is to use numerical optimal control, which is a commonly used method for computing motion primitives [4, 36, 51, 68]. In this thesis, we use state-of-the-art numerical optimal control software applicable for general nonlinear systems to solve the OCPs. In these methods, the continuous-time optimal control problem in (2.31) is reformulated as an NLP problem using methods described in Section 2.2. An example of a resulting motion primitive set for a car-like vehicle is shown in Figure 2.5.
Figure 2.5: One example of a motion primitive set computed offline that can be used to solve motion planning problems online. Each color corresponds to an initial heading direction. System symmetries are exploited such that calculations are only required to be performed from a few initial headings. Then, the obtained solutions are mirrored and/or rotated to the other headings.

Online planning

After the motion primitive set is computed, it is possible to approximate the OCP in (2.19) with the following discrete OCP:

\[
\begin{align*}
\text{minimize} & \quad J_{\text{lat}} = \sum_{k=1}^{N} L_u(m_k) \\
\text{subject to} & \quad x_1 = x_{\text{init}}, \quad x_{N+1} = x_{\text{term}}, \quad (2.33a) \\
& \quad x_{k+1} = f_m(x_k, m_k), \quad k \in \mathbb{Z}_{1,N} \quad (2.33b) \\
& \quad m_k \in \mathcal{P}(x_k), \quad k \in \mathbb{Z}_{1,N} \quad (2.33c) \\
& \quad c(x_k, m_k) \in \mathcal{X}_\text{free}, \quad k \in \mathbb{Z}_{1,N}. \quad (2.33d)
\end{align*}
\]

The decision variables are the motion primitive sequence \( \{m_k\}_{k=1}^{N} \) and its length \( N \). The stage cost for each motion primitive is \( L_u(m) = \int_0^T \ell(x(t), u(t))dt \). The state transition equation in (2.33c) defines the successor state \( x_{k+1} \) after \( m_k \) is applied from \( x_k \) and (2.33d) ensures that \( m_k \) is selected from the set of applicable motion primitives at \( x_k \). Finally, the constraint in (2.33e) ensures that the traveled path of the system when \( m_k \) is applied from \( x_k \) does not collide with any obstacles.

The use of the approximation in (2.33) enables the use of efficient graph search methods online such as \( A^* \) [33], anytime repairing \( A^* \) (ARA*) [48], etc., making it real-time applicable [51, 68]. Since standard graph search techniques can be used, the lattice-based motion planner inherits the properties and optimality guarantees.
of the chosen graph search method. Another benefit is that the solution is guaranteed to be feasible with respect to the system model since the motion primitives have been computed by solving (2.31). On the downside, the solutions from a lattice-based planner often suffer from discretization artifacts [5, 60], making it desirable to smoothen the state-lattice solution. Another limitation with graph search is that it is only possible to plan from and to states within the selected state-space discretization [45].

An A* search algorithm, inspired by Algorithm 1 in [25], tailored for a lattice-based motion planner is outlined in Algorithm 2.2. The algorithm maintains two sets of states: one to be able to keep track of which states that have been visited, \( V_{\text{closed}} \), and one which gathers all the states left for exploration, \( V_{\text{open}} \). The states in \( V_{\text{open}} \) are sorted according to an estimate of the objective function value represented by \( J_{\text{est}} \). This estimate is calculated by the sum of the cost-to-come defined by \( J_{\text{come}} \), and an estimate of the cost-to-go given by the heuristic function \( h(x, x_{\text{term}}) \). Hence, the most promising state is always selected for expansion. As stated in Section 2.3.4, the online computational performance relies on a well-informed heuristic function to guide the search towards the solution. One alternative that is compatible with lattice-based motion planning is to use a precomputed heuristic look-up table (HLUT). The HLUT typically represents a perfect estimate of the cost-to-go in free-space [40]. The HLUT is computed offline by solving a large number of motion planning problems (2.33) in an obstacle-free environment from all initial states \( x_{\text{init}} \in \mathcal{X}_d \) with a position at the origin to, e.g., all states \( x_{\text{term}} \in \mathcal{X}_d \) with a position within a square centered around the origin with a user-defined side length \( \rho \), where the size of the HLUT for a sufficient online planning performance is application dependent. In [40], some techniques for reducing the required memory usage and computation time for storing and generating the HLUT can be found.

Once a state \( x_k \in V_{\text{open}} \) has been selected for expansion, each possible feasible motion primitive from the selected state, i.e., \( m \in \mathcal{P}(x_k) \), is applied from \( x_k \) and the resulting state \( x_{\text{next}} \) is calculated (line 9 in Algorithm 2.2). Now, four possible scenarios can occur:

1. If \( x_{\text{next}} \) is already in the set of closed states \( V_{\text{closed}} \), the path in the graph between \( x_k \) and \( x_{\text{next}} \) can be pruned since the best solution from \( x_{\text{init}} \) to \( x_{\text{next}} \) is already found (line 10 in Algorithm 2.2).

2. If the path between \( x_k \) and \( x_{\text{next}} \) is in collision, represented by the constraint \( c(x_k, m) \not\in X_{\text{free}} \), the current path in the graph is pruned and the search continues, since the motion is infeasible (line 13 in Algorithm 2.2).

3. If \( x_{\text{next}} \) does not exist in \( V_{\text{open}} \), it is added as a new possible state for expansion with associated costs \( J_{\text{come}} \) and \( J_{\text{est}} \), its parent state \( x_k \) and the applied motion primitive \( m \) (lines 16-20 in Algorithm 2.2).

4. If \( x_{\text{next}} \) already exists in \( V_{\text{open}} \), but the new cost-to-come \( J_{\text{come}} \) is lower than the cost-to-come value in \( x_{\text{next}} \), the state is updated with new costs, parent state and motion primitive (lines 23-25 in Algorithm 2.2). Otherwise, the previous path in the graph to get to \( x_{\text{next}} \) has a lower cost-to-come and the current path in the graph is pruned.
2.3 Optimal motion planning

The search for a solution is terminated when \( x_{\text{term}} \) is selected for expansion, given by line 6 in Algorithm 2.2, where the solution is obtained by backtracking the parent states from \( x_{\text{term}} \) until \( x_{\text{init}} \) is reached. Optimality of the solution can be ensured since the list of open states is sorted according to the estimated objective function value. Hence, under the assumption that the heuristic function is both admissible and consistent, it is not possible to find any better solution with the current motion primitive set that brings the system from \( x_{\text{init}} \) to \( x_{\text{term}} \) [33].

---

Algorithm 2.2: A* search tailored for lattice-based motion planning

1: Initialization: \( \mathcal{V}_{\text{open}} \leftarrow x_{\text{init}}, \mathcal{V}_{\text{closed}} \leftarrow \emptyset \).
2: while \( \mathcal{V}_{\text{open}} \neq \emptyset \) do
3: \( x_k \leftarrow \mathcal{V}_{\text{open}}.\text{pop}() \)
4: \( \mathcal{V}_{\text{closed}}.\text{push}(x_k) \)
5: if \( x_k = x_{\text{term}} \) then
6: return back_track\((x_k)\)
7: end if
8: for each \( m \in \mathcal{P}(x_k) \) do
9: \( x_{\text{next}} \leftarrow f_m(x_k, m) \)
10: if \( x_{\text{next}} \in \mathcal{V}_{\text{closed}} \) then
11: continue
12: end if
13: if \( c(x_k, m) \notin \mathcal{X}_{\text{free}} \) then
14: continue
15: end if
16: \( J_{\text{come}} \leftarrow x_k.\text{come} + L_u(m) \)
17: \( J_{\text{est}} \leftarrow J_{\text{come}} + h(x_{\text{next}}, x_{\text{term}}) \)
18: if \( x_{\text{next}} \notin \mathcal{V}_{\text{open}} \) then
19: \( x_{\text{next}}.\text{come} \leftarrow J_{\text{come}} \)
20: \( x_{\text{next}}.\text{est} \leftarrow J_{\text{est}} \)
21: \( x_{\text{next}}.\text{parent} \leftarrow x_k \)
22: \( x_{\text{next}}.\text{motion} \leftarrow m \)
23: \( \mathcal{V}_{\text{open}}.\text{push}(x_{\text{next}}) \)
24: end if
25: if \( x_{\text{next}} \in \mathcal{V}_{\text{open}} \) and \( J_{\text{come}} < x_{\text{next}}.\text{come} \) then
26: \( \mathcal{V}_{\text{open}}.\text{update\_cost}(x_{\text{next}}, J_{\text{come}}, J_{\text{est}}) \)
27: \( \mathcal{V}_{\text{open}}.\text{update\_parent}(x_{\text{next}}, x_k) \)
28: \( \mathcal{V}_{\text{open}}.\text{update\_motion}(x_{\text{next}}, m) \)
29: end if
30: end for
31: end while
In this chapter, it is shown how a state-of-the-art method from numerical optimal control can be combined with a homotopy method into a method that solves motion planning problems with challenging geometry of the feasible set. Furthermore, the method is combined with a novel structure-exploiting constraint parameterization introduced in this chapter. It is illustrated that the proposed method manages to efficiently solve problems where a traditional state-of-the-art numerical optimal control method fails to compute any solution at all. Furthermore, it significantly outperforms an optimizing sampling-based motion planner both in terms of computational performance as well as trajectory quality.

3.1 Homotopy-based reformulation

Since solvers for a system of nonlinear equations are sensitive to the choice of initial iterate $z_0$, there is a risk of converging to a local minimum which does not coincide with a global minimum, or not converging at all, for difficult problems. Convergence to a globally optimal solution might be too much to ask for, but what is desired in practice is convergence to a local solution of practical relevance. One way to deal with this is to use so-called homotopy, or continuation methods [1]. Instead of solving the original problem directly, one can set up an easy problem for which the solution is more or less obvious, and then successively transform the problem towards the original and more difficult one.

--- Example 3.1: Homotopy map ---

If the problem is to find a root for the nonlinear equation $F(z) = 0$, $F : \mathbb{R}^n \to \mathbb{R}$, the homotopy method instead focuses on finding a root to the homotopy map

$$H(z, \gamma) = \gamma F(z) + (1 - \gamma) G(z), \quad (3.1)$$
where $\gamma \in [0,1]$, from now on denoted homotopy parameter, and $G : \mathbb{R}^n \to \mathbb{R}$ is a user-defined function. The basic idea is to solve a sequence of $n+1$ problems $H(z, \gamma) = 0$, for $\gamma = \gamma_0 < \gamma_1 < \ldots < \gamma_n = 1$. If the roots are relatively easy to find for $G(z)$, it should be easy to find a first iterate $z_0$. Then for any problem $k \in [0,n]$ along the homotopy path, given a starting point $z_k$ that is the approximate solution from $H(z, \gamma_k) = 0$, the next approximate solution $z_{k+1}$ is calculated by solving $H(x, \gamma_{k+1}) = 0$. This can be done using an iterative method such as Newton’s method, with $x_k$ as initial iterate. If the difference between $\gamma_k$ and $\gamma_{k+1}$ is small enough, the method will converge. In the last iteration, a root for $H(z, 1) = F(z) = 0$ is found, i.e., the original system of equations [1].

If we separate the inequality constraints $c$ in the optimal control problem (2.19) into two parts

- $c_C : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^r$, that contains all the convex inequality constraint functions and

- $c_H : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$, $r + q = p$, that contains all the non-convex inequality constraint functions,

it is possible to replace the non-convex inequality constraints $c_H$ (which for motion planning typically represents the obstacle avoidance constraints) with the homotopy map $h : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^q$, which represents a continuous transformation of $g_H$ with respect to the homotopy parameter $\gamma$. We require that $h$ is a homeomorphism\(^1\) with respect to the inequality constraint functions.

**Definition 3.1 (Homeomorphism).** A function $f : X \to Y$ between two topological spaces $X$ and $Y$ is called a homeomorphism if

1. $f$ is a one-to-one and onto mapping (bijection), and

2. both $f$ and $f^{-1}$ are continuous [45].

An illustrating example of homeomorphic transformations can be found in Figure 3.1. The reason to require that $h$ is a homeomorphism is easily motivated in the context of a motion planning problem.

**Definition 3.2 (Homotopy class).** Two trajectories are said to be in the same homotopy class if they can be continuously transformed from one to another without intersecting any obstacles, while keeping their endpoints fixed [14].

An example of an environment with two homotopy classes is shown in Figure 3.2: one class above the obstacle and one class below. Hence, the homeomorphism requirement on $h$ prevents the introduction of new, and removal of existing, homotopy classes, while continuously perturbing the available free space in each of the existing homotopy classes when $\gamma$ is updated.

\(^1\)The method can be generalized to handle a sequence of homeomorphic mappings.
Figure 3.1: An example illustrating homeomorphic and non-homeomorphic transformations of objects in 2D. The transformations from the top objects to the bottom objects are not homeomorphic due to the discontinuity introduced by the hole in the middle.

Figure 3.2: An example illustrating an environment with two trajectories belonging to different homotopy classes. The trajectories can not be continuously transformed into each other due to the obstacle.
Now, using the homotopy map, it is possible to write the OCP in (2.19) as

\[
\begin{align*}
\text{minimize} & \quad J = \int_{t_i}^{t_f} \ell(x(t), u(t)) \, dt \\
\text{subject to} & \quad x(t_i) = x_{\text{init}}, \quad x(t_f) = x_{\text{term}} \\
& \quad \dot{x}(t) = f(x(t), u(t)), \quad t \in [t_i, t_f] \\
& \quad c_C(x(t), u(t)) \leq 0, \quad t \in [t_i, t_f] \\
& \quad h(x(t), u(t), \gamma) \leq 0, \quad t \in [t_i, t_f].
\end{align*}
\]  

(3.2)

With this formulation, we can solve the problem iteratively by first solving a problem with simple or even convex inequality constraint functions, and then gradually perturb the non-convex inequality constraints by increasing the value of the homotopy parameter \(\gamma\). This method shares similarities with a barrier method (such as the IP method described in Section 2.1.3), where a sequence of problems is solved, starting from a simpler problem and then approaching the solution to the original one. In these methods, the barrier parameter corresponds to the homotopy parameter [59].

One important special type of perturbation structure is when the homotopy parameter corresponds to the level of relaxation of the optimization problem. Assume that the homotopy map is constructed such that the level of relaxation is gradually decreased as the homotopy parameter is increased from 0 to 1. Then, the optimal objective function value for \((x_k^*(\cdot), u_k^*(\cdot), \gamma_k)\) will always be greater than or equal to the optimal objective function value for a previous iterate \((x_{k-1}^*(\cdot), u_{k-1}^*(\cdot), \gamma_{k-1})\) as the problem (3.2) is solved for a strictly increasing value of \(\gamma\), i.e.,

\[
J(x_k^*(\cdot), u_k^*(\cdot), \gamma_k) \geq J(x_{k-1}^*(\cdot), u_{k-1}^*(\cdot), \gamma_{k-1}), \quad \gamma_k > \gamma_{k-1},
\]  

(3.3)

as the homotopy parameter is increased from 0 to 1. Furthermore, the following theorem holds when the homotopy map represents a gradual tightening for an increased value of \(\gamma\):

**Theorem 3.1.** Let the constraint \(h(x, u, \gamma) \leq 0\) in (3.2) be a homotopy map which represents a deceased level of relaxation (in optimization sense) as \(\gamma\) is increased from 0 to 1. Furthermore, let \(\mathcal{P}_k\) denote the problem (3.2) with homotopy parameter \(\gamma_k \in [0, 1]\), and let \(\mathcal{P}_{k+1}\) denote the problem (3.2) with homotopy parameter \(\gamma_{k+1} \geq \gamma_k\). If \(\mathcal{P}_k\) is infeasible, then also \(\mathcal{P}_{k+1}\) is infeasible for any \(\gamma_{k+1} \in [\gamma_k, 1]\).

**Proof:** It follows from that the feasible set of \(\mathcal{P}_{k+1}\) is a subset of the feasible set of \(\mathcal{P}_k\).

One implication of Theorem 3.1 is that if a subproblem is encountered where no solution exists for a value of the homotopy parameter \(\gamma_k < 1\), the search for a solution can be terminated since it is guaranteed that the actual problem to be solved in (3.2) is infeasible.
3.2 Sequential Homotopy Quadratic Programming

In this section, a prototype algorithm forming the back-bone of the main contribution in this chapter will be explained. As discussed in Section 2.2.1, the OCP in (2.19) can be transformed to an NLP by for example using a direct multiple shooting approach and then solved using SQP. Here, we extend the resulting SQP subproblem (in the form of (2.9)) to also include the homotopy map:

$$\begin{align*}
\min_{p_k} & \quad \frac{1}{2} p_k^T B_k p_k + \nabla L(z_k)^T p_k \\
\text{subject to} & \quad \nabla F(z_k)^T p_k + F(z_k) = 0, \\
& \quad \nabla G_c(z_k)^T p_k + G_c(z_k) \preceq 0, \\
& \quad \nabla H(z_k, \gamma)^T p_k + H(z_k, \gamma) \preceq 0. 
\end{align*}$$

(3.4)

In this compact formulation, $F$ contains all the discrete-time dynamic system constraints as well as initial and terminal constraints, $G_c$ contains all the convex inequality constraint functions, and $H$ contains all the homotopy mappings with corresponding Jacobian matrices $\nabla F$, $\nabla G_c$ and $\nabla H$, respectively. We observe that for a fixed $\gamma$, the problem can be solved, as usual, by a standard SQP method to local optimality. Furthermore, for a fixed working set, the corresponding KKT system is a system of nonlinear equations for which a standard homotopy approach as the one described in [1] can be used. Similarly to [71], we now propose to combine these two methods into one. We call the resulting method Sequential Homotopy Quadratic Programming (SHQP), which is described in Algorithm 3.1, using notations from (3.4). It is inspired by Algorithm 18.3 in [59], which describes an SQP method using line search combined with a merit function to globalize the search for a solution. For more details regarding SQP and globalization techniques, the reader is referred to Section 2.1 or [59].

The outer iteration in the algorithm corresponds to the homotopy mapping, while the inner iteration corresponds to the standard SQP algorithm. For every value of $\gamma$, the SQP algorithm iterates until the termination test is satisfied, or until a maximum number of $N_{\text{SQP}}$ SQP iterations has been performed. The termination test can vary with $\gamma$. It is natural to terminate the inner iterations already at a relatively low accuracy at the intermediate outer iterations. However, in the last outer iteration (with $\gamma = 1$), we would like the termination test to guarantee convergence. The same approach is used for the inexact centering in the barrier method [15], where the barrier parameter (homotopy parameter) is updated before Newton’s method has fully converged.

An important question to consider is how to update $\gamma$ in every outer iteration, i.e., how to choose the step size $\Delta \gamma$, to obtain the fastest possible convergence. We want to use as large steps as possible, such that the problem is solved with few homotopy iterations. However, if $\Delta \gamma$ is too large, the initialization from the last step could be far from the solution to the updated problem. This will require many SQP iterations before the termination test is satisfied in Algorithm 3.1, or it might not converge at all. An illustrating example of this can be seen in Figure 3.3. This trade-off is shared both with standard homotopy methods as well as barrier
Algorithm 3.1: Sequential Homotopy Quadratic Programming

1: Choose line search and merit function parameters [59]
2: \( \gamma \leftarrow \gamma_{\text{init}}, z_0 \leftarrow z_{\text{init}}, \lambda_0 \leftarrow \lambda_{\text{init}}, \nu_0 \leftarrow \nu_{\text{init}} \)
3: Choose an initial Hessian approx. \( B_0 \), or compute \( B_0 = \nabla^2_{zz} L_0 \)
4: Evaluate \( L(z_0), \nabla L(z_0), F(z_0), \nabla F(z_0), G_c(z_0), \nabla G_c(z_0) \)
5: \( \textbf{while} \ \gamma \leq 1 \ \textbf{do} \)
6: \( \textbf{for} \ k = 0 : N_{\text{sqp}} \ \textbf{do} \)
7: \( \textbf{if} \ \text{Termination test satisfied} \ \textbf{then} \)
8: \( \textbf{continue} \)
9: \( \textbf{end if} \)
10: Compute \( p_k \) by solving (3.4) at \( z_k, \lambda_k, \nu_k, \gamma \)
11: Calculate step size parameter \( \alpha_k \in [\alpha_{\text{min}}, 1] \) [59]
12: \( z_{k+1} \leftarrow z_k + \alpha_k p_k \)
13: Evaluate \( L(z_{k+1}), \nabla L(z_{k+1}), F(z_{k+1}), \nabla F(z_{k+1}), G_c(z_{k+1}), \nabla G_c(z_{k+1}), \)
14: \( H(z_{k+1}, \gamma), \nabla H(z_{k+1}, \gamma) \)
15: Compute new Lagrange multipliers \( \lambda_{k+1}, \nu_{k+1} \) [59]
16: Update Hessian using the exact Hessian or a quasi-Newton formula [59]
17: \( \textbf{end for} \)
18: \( \textbf{if} \ \text{Infeasible problem} \ \textbf{then} \)
19: \( \textbf{return} \ \text{infeasible problem detected} \)
20: \( \textbf{end if} \)
21: \( \textbf{if} \ \gamma = 1 \ \textbf{then} \)
22: \( \textbf{return} \ z_{k+1} \)
23: \( \textbf{end if} \)
24: \( \gamma \leftarrow \gamma + \Delta \gamma \)
25: \( z_0 \leftarrow z_{k+1}, \lambda_0 \leftarrow \lambda_{k+1}, \nu_0 \leftarrow \nu_{k+1}, B_0 \leftarrow B_{k+1} \)
26: \( L(z_0), F(z_0), G_c(z_0) \leftarrow L(z_{k+1}), F(z_{k+1}), G_c(z_{k+1}) \)
27: \( \nabla L(z_0), \nabla F(z_0), \nabla G_c(z_0) \leftarrow \nabla L(z_{k+1}), \nabla F(z_{k+1}), \nabla G_c(z_{k+1}) \)
28: \( \textbf{end while} \)
3.3 Obstacle classification

The total number of SQP iterations in Algorithm 3.1 depends on several factors, such as $\Delta \gamma$ and the nonlinear dynamics of the system. Other important factors are how large and what type of obstacles that are included in the optimal motion planning problem. We have categorized the obstacles in two main classes based on topological properties of the obstacle. We denote these classes Connected and Disconnected.

The free space in a motion planning problem is defined as the intersection of two sets. The first of these sets represents the convex free space and is specified by a number of convex inequality constraints. The second one represents the complement of the obstacles and is given by a set of nonconvex inequality constraints. The connected obstacles are, in the original problem, connected with at least one convex inequality constraint. From this, it follows in which direction the connected obstacle will grow during the homotopy phase, such that the space representing the feasible area does not introduce new homotopy classes. The name refers to the fact that in this case the infeasible area is topologically connected.

The disconnected obstacles are obstacles that are not intersecting any border of the convex free space. In this case, the homotopy map can, e.g., be introduced with a point representing the center of the obstacle. The existence of a disconnected obstacle will increase the number of homotopy classes in the problem.

Both these types of obstacles can be further divided into single and chained variants of the main classes. A single-type obstacle is represented only by one constraint, while the chain-type obstacles are represented by more than one constraint which are geometrically connected in a chain. The classification of obstacles will be of importance when choosing a homotopy map that satisfies the homeomorphism requirement. Examples of these types of obstacles can be found in Figure 3.4. The chosen homotopy maps for (a)-(c) gradually introduces the obstacles from the outside of the area represented by the convex part of the free space. In (b), the
Figure 3.3: An example of a large step (a), and a small step (b) in \( \gamma \), when an obstacle is perturbed by pushing it into the free space. The blue lines represent the simulated trajectory from every multiple shooting point. When a small step is used, the algorithm has already converged after one SQP iteration, while it is far from convergence using a large step.

long Chain-type obstacle is gradually extended like a “snake”. In (d), the radius of the torus is large in the beginning and gradually reduced.

### 3.3.1 Examples of obstacle representations

Algorithm 3.1 accepts all types of obstacle constraint representations that can be handled by SQP solvers, which means that the constraint functions should be smooth. One example of practical relevance is to represent them as super-ellipsoids:

\[
\left( \frac{x - x_c}{r_x} \right)^k + \left( \frac{y - y_c}{r_y} \right)^k + \left( \frac{z - z_c}{r_z} \right)^k - 1 \geq 0, \tag{3.5}
\]

for \( k = 2N, N \in \mathbb{Z}^+ \). \([x_c, y_c, z_c]\) represents the center-point of the obstacle, while \([r_x, r_y, r_z]\) defines the three radii for the super-ellipsoid in each axis. Higher values of \( k \) produces obstacles with sharper edges. However, an increasing \( k \) will cause the linear approximation of the constraints used in the SQP iterations to be more numerically sensitive. This will in turn lead to a slower convergence. Some examples of super-ellipsoid obstacles with \( k = 4 \) can be found in Figure 3.4.
### Example 3.2: Homotopy map for super-ellipsoid

The obstacle in Figure 3.3 is defined by the inequality

$$(x - 4)^4 + \left(\frac{y}{6}\right)^4 - 1 \geq 0. \quad (3.6)$$

If the obstacle is connected, e.g., when the feasible region is constrained by $y \geq 0$, this inequality can be replaced by the homotopy map

$$(x - 4)^4 + \left(\frac{y - 6(1 - \gamma)}{6}\right)^4 - 1 \geq 0, \quad (3.7)$$

which will continuously move the obstacle into the free space when $\gamma$ is increased from 0 to 1, without increasing the number of homotopy classes. For $\gamma = 1$, we have the original formulation (3.6).

If the obstacle is disconnected, the inequality in (3.6) can be replaced by a homotopy map in the form

$$(x - 4)^4 + \left(\frac{y}{6}\right)^4 - \gamma \geq 0, \quad (3.8)$$

which for $\gamma = 0$ is represented by a point in the center of the obstacle.

A second example we have used is to model an obstacle as a torus (illustrated in Figure 3.4 (d))

$$\left(R - \sqrt{(x - x_c)^2 + (y - y_c)^2}\right)^2 + (z - z_c)^2 - r^2 \geq 0. \quad (3.9)$$

Here, $R$ defines the distance from the center of the torus to the center of the tube, and $r$ is the radius of the tube. One homotopy map that can be used for this type of obstacle is to let

$$R(\gamma) = R_{true} + (1 - \gamma)r_{true}, \quad r(\gamma) = \gamma r_{true}, \quad (3.10)$$

where $R_{true}$ and $r_{true}$ are the true values for the torus. This transformation will keep the outer radius of the torus $R + r$ constant, while continuously increasing the radius of the tube when $\gamma$ is increased. This will lead to that the hole in the center of the torus will decrease until the original constraint in (3.9) is obtained.

### 3.4 Numerical results

In this section the numerical results from four different trajectory planning scenarios are presented. To produce the results, we have emulated Algorithm 3.1 for the sake of proof-of-concept. This is done using a script that iteratively calls ACADO for solving the OCP in (3.2) where the non-convex inequality constraint functions have been replaced by the homotopy map. The practical difference compared to Algorithm 3.1 is that ACADO needs to redo all the initialization when an
widely used open-source library which consists of several state-of-the-art sampling-based motion planning algorithms based on random sampling strategies. In these numerical results, the Stable Sparse RRT (SSR) planner [46] is used, since it is the only planner in OMPL that both accepts a dynamic system model and is able to optimize the trajectory according to a specified objective function. All simulations were performed on a laptop with an Intel Core i7-5600U processor.

### 3.4.1 Vehicle models

The numerical results presented in this section are based on two different types of vehicle models. The first one is based on a kinematic car-like vehicle model [45]

\[
\begin{align*}
\dot{x}(t) &= \begin{bmatrix} \dot{x}_1(t) \\ \dot{y}_1(t) \\ \dot{\theta}_1(t) \\ \dot{v}(t) \\ \dot{\alpha}(t) \end{bmatrix} = f(x(t), u(t)) = \\
&= \begin{bmatrix} v(t) \cos \theta_1(t) \\ v(t) \sin \theta_1(t) \\ v(t) \tan \alpha(t) \\ u_v(t) \\ u_\alpha(t) \end{bmatrix}. \tag{3.11}
\end{align*}
\]

Here, \( x = (x_1, y_1, \theta_1, v, \alpha) \) is the state vector of the car which represents the position, heading, speed, and steering angle, respectively. The control signals to the system are \( u = (u_v, u_\alpha) \) which represent the change rate in speed and steering angle, respectively. The steering angle is limited to \(-\frac{\pi}{4} \leq \alpha(t) \leq \frac{\pi}{4}\) and the control signals are limited to \(-2 \leq u_v(t) \leq 2\) and \(-\frac{\pi}{4} \leq u_\alpha(t) \leq \frac{\pi}{4}\). The speed of the car can be both positive (forward motion) and negative (backward motion), limited to \(-1 \leq v(t) \leq 1\).

The second model used in this section is an extension of (3.11) to allow movements in 3D. For this, we have used a model representing a simple fixed-wing aircraft

\[
\begin{align*}
\dot{x}(t) &= \begin{bmatrix} \dot{x}_1(t) \\ \dot{y}_1(t) \\ \dot{\theta}_1(t) \\ \dot{z}_1(t) \\ \dot{v}(t) \\ \dot{\alpha}(t) \end{bmatrix} = \begin{bmatrix} v(t) \cos \theta_1(t) \sin \phi(t) \\ v(t) \sin \theta_1(t) \sin \phi(t) \\ v(t) \tan \alpha(t) \\ u_v(t) \\ u_\alpha(t) \end{bmatrix} \\
&= \begin{bmatrix} v(t) \cos \phi(t) \\ u_\phi(t) \end{bmatrix}. \tag{3.12}
\end{align*}
\]

This model also contains states that represent the altitude, \( z(t) \), and polar angle, \( \phi(t) \), which is limited to \(0 \leq \phi(t) \leq \pi\). Here, \( \phi(t) < \pi/2 \) corresponds to an angle for which the altitude is increased, and \( \phi(t) > \pi/2 \) an angle where the altitude is decreased. The magnitude and direction of the velocity vector is controlled by changing \( u = [u_v, u_\alpha, u_\phi] \). The added control signal for controlling \( \phi(t) \) is limited
to $-\frac{\pi}{3} \leq u_\phi \leq \frac{\pi}{3}$. For this case, we require a speed that is strictly positive, limited to $0.2 \leq v(t) \leq 1$. Both these models have been implemented in OMPL by using the available `CompoundStateSpace` class.

The objective function in the motion planning OCP will be, for both models, to minimize the total path length. This can be achieved using the cost function $\ell(x(t), u(t)) = |v(t)|$ in (3.2). In ACADO, this non-smooth cost function is reformulated into a smooth epigraph formulation by introducing an auxiliary decision variable [15]. In OMPL, the objective function is approximated by $\sum_i |v_i| dt_i$, where $i$ represents the $i$th state in the sampled version of the trajectory computed by the SST planner, and $dt_i$ is the amount of time $i$th control signal is applied (which is set to vary randomly between 0.1, 0.2, 0.3, 0.4 and 0.5 seconds).

### 3.4.2 Experimental results

In this section, solutions from the following four trajectory planning problems will be presented

- **P1**: $x_{t_i} = [1,1,0,0,0]$, $x_{\text{term}} = [9,9,0,0,0]$, using the motion model (3.11) and environment specified by (a) in Figure 3.4.
  
  SHQP: $\Delta \gamma = 0.02$, prediction horizon $N = 30$.
  
  SST: Pruning radius $\delta_s : 0.3$, Selection Radius $\delta_v : 1.2$

- **P2**: $x_{t_i} = [1,1,\frac{\pi}{2},1,0]$, $x_{\text{term}} = [15.5,3.5,\frac{\pi}{2},0,0]$, using the motion model (3.11) and environment specified by (b) in Figure 3.4.
  
  SHQP: $\Delta \gamma = 0.01$, prediction horizon $N = 40$.
  
  SST: pruning radius $\delta_s : 0.3$, selection radius $\delta_v : 0.6$

- **P3**: $x_{t_i} = [1,1,\frac{\pi}{4},1,1,0,0]$, $x_{\text{term}} = [9,9,\frac{\pi}{2},1,1,0,0]$, using the motion model (3.12) and environment specified by (c) in Figure 3.4.
  
  SHQP: $\Delta \gamma = 0.02$, prediction horizon $N = 40$.
  
  SST: pruning radius $\delta_s : 0.3$, selection radius $\delta_v : 0.9$

- **P4**: $x_{t_i} = [6,6,0,9,1,0,\pi]$, $x_{\text{term}} = [6,6,0,9,1,0,\pi]$, using the motion model (3.12) and environment specified by (d) in Figure 3.4.
  
  SHQP: $\Delta \gamma = 0.01$, prediction horizon $N = 40$.
  
  SST: pruning radius $\delta_s : 0.25$, selection radius $\delta_v : 0.5$

In all examples, the area outside the box shown in Figure 3.4 is prohibited. Both standard ACADO and the emulated version of Algorithm 3.1 use the built-in initialization procedure in ACADO for the first iteration with homotopy parameter $\gamma = 0$, which means linear interpolation between the initial and terminal state. Note that the emulated version of Algorithm 3.1 does not get any extra external help with initialization. As a result of the method we propose, the initial problem in the homotopy is easy for ACADO to solve using its built-in initialization. The SST planner is initialized with the values of the parameters $\delta_s$ and $\delta_v$ presented above,
Table 3.1: The number of executions, out of 100, that returned a solution within 1000 seconds with a terminal state in the goal region. Standard ACADO was not able to produce a feasible solution to any of the problems.

<table>
<thead>
<tr>
<th>Planner</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACADO</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SHQP</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>SST</td>
<td>100</td>
<td>77</td>
<td>89</td>
<td>13</td>
</tr>
</tbody>
</table>

which are manually tuned to obtain good performance for each problem instance. For more information regarding $\delta_s$ and $\delta_v$, the reader is referred to [46]. We terminate the SST planner when a solution is found that satisfies $||x(t_f) - x_{\text{term}}|| < 1$, or after 1000 seconds, and returns the solution with $x(t_f)$ closest to the terminal state.

In Figure 3.4 (a) we have a trajectory planning problem with two homotopy classes, where our algorithm finds a solution in the homotopy class to the left of the disconnected obstacle. Figure 3.4 (b) is a challenging maze-type problem, which requires the largest amount of computation time from our method (see Table 3.2) due to the complex environment.

One problem which is challenging for the sampling-based methods is the torus example in Figure 3.4 (d), since it is required to draw samples which brings the platform through the hole in the middle to pass the obstacle [45]. This is confirmed by Table 3.1; only 13% of the executions of the sampling-based method SST were able to find a solution that satisfies $||x(t_f) - x_{\text{term}}|| < 1$ within 1000 seconds. It can also be seen that the SST method gets greedy and tries to sample straight towards the goal through the obstacle. Furthermore, Table 3.1 shows that our proposed method obtains a 100% success rate for all problems, while standard ACADO was not able to solve any of these problems due to the non-convex inequality constraints representing the obstacles. Moreover, it can be observed that SST achieves 100% success rate only on the simplest problem P1. The success rate for SST is significantly reduced on the maze problem P2 in 2D and is as low as 13% for the torus problem P4 in 3D.

Table 3.2 shows that the SHQP method completely outperforms the SST planner in terms of computation times, despite the previously described low-performance scripted implementation of Algorithm 3.1. In particular, it can be noted that for the difficult torus problem P4, already our rudimentary SHQP implementation shows convincing performance being more than 120 times faster than OMPL’s SST implementation. We also note that the proposed method scales well in higher dimension, which is one of the expected advantages.

Table 3.3 shows that SHQP produces paths that have a lower objective function value compared to the SST planner. Furthermore, the standard deviation of the objective function value is zero for the SHQP method, since the method is deterministic. For the SST method, the quality of the solution computed is not nearly as consistent. Finally, Table 3.4 shows another advantage with the proposed method
Figure 3.4: The resulting trajectories for the four trajectory planning problems after 100 simulations. The variance in the solutions from SST is illustrated with opacity; higher color intensity indicates more frequently visited areas. The only way to get from initial to terminal state in P4 is via the hole in the middle, which has a minimum radius of 0.5 meter.
Table 3.2: Mean ($\mu_t$) and standard deviation ($\sigma_t$) of the computation time (seconds) from 100 executions of the different planners, with a maximum allowed execution time of 1000 seconds.

<table>
<thead>
<tr>
<th>Planner</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHQP</td>
<td>$\mu_t$: 3.13</td>
<td>$\mu_t$: 28.5</td>
<td>$\mu_t$: 9.98</td>
<td>$\mu_t$: 7.64</td>
</tr>
<tr>
<td></td>
<td>$\sigma_t$: 0.02</td>
<td>$\sigma_t$: 0.23</td>
<td>$\sigma_t$: 0.06</td>
<td>$\sigma_t$: 0.04</td>
</tr>
<tr>
<td>SST</td>
<td>$\mu_t$: 42.4</td>
<td>$\mu_t$: 560</td>
<td>$\mu_t$: 320</td>
<td>$\mu_t$: 927</td>
</tr>
<tr>
<td></td>
<td>$\sigma_t$: 89.2</td>
<td>$\sigma_t$: 313</td>
<td>$\sigma_t$: 336</td>
<td>$\sigma_t$: 222</td>
</tr>
</tbody>
</table>

Table 3.3: Mean ($\mu_o$) and standard deviation ($\sigma_o$) of the objective function value (path length in [m]) from the executions that returned a solution with terminal state within the goal region (see Table 3.1).

<table>
<thead>
<tr>
<th>Planner</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHQP</td>
<td>$\mu_o$: 12.3</td>
<td>$\mu_o$: 27.5</td>
<td>$\mu_o$: 15.3</td>
<td>$\mu_o$: 25.1</td>
</tr>
<tr>
<td></td>
<td>$\sigma_o$: 0</td>
<td>$\sigma_o$: 0</td>
<td>$\sigma_o$: 0</td>
<td>$\sigma_o$: 0</td>
</tr>
<tr>
<td>SST</td>
<td>$\mu_o$: 20.1</td>
<td>$\mu_o$: 57.4</td>
<td>$\mu_o$: 24.9</td>
<td>$\mu_o$: 38.0</td>
</tr>
<tr>
<td></td>
<td>$\sigma_o$: 4.40</td>
<td>$\sigma_o$: 7.07</td>
<td>$\sigma_o$: 4.47</td>
<td>$\sigma_o$: 7.14</td>
</tr>
</tbody>
</table>

compared to sampling-based planners based on random sampling, which is that the terminal state can be reached with a much higher accuracy without dependencies to other methods such as steering functions.

From our experiments, we note that it is highly relevant in these applications to use QP solvers that efficiently can solve problems with long prediction horizons $N$, such as those presented in [30], [57], [6], which have a computational complexity that grows as $O(N)$. For challenging problems, the result in [58], which gives a computational complexity as low as $O(\log N)$ would be useful.

Table 3.4: Mean ($\mu_o$) and standard deviation ($\sigma_d$) of $||x(t_f) - x_{term}||$ from 100 executions of the different planners.

<table>
<thead>
<tr>
<th>Planner</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHQP</td>
<td>$\mu_d$: $3.35e^{-14}$</td>
<td>$\mu_d$: $7.78e^{-14}$</td>
<td>$\mu_d$: $3.56e^{-14}$</td>
<td>$\mu_d$: $3.35e^{-14}$</td>
</tr>
<tr>
<td></td>
<td>$\sigma_d$: 0</td>
<td>$\sigma_d$: 0</td>
<td>$\sigma_d$: 0</td>
<td>$\sigma_d$: 0</td>
</tr>
<tr>
<td>SST</td>
<td>$\mu_d$: 0.79</td>
<td>$\mu_d$: 1.01</td>
<td>$\mu_d$: 0.86</td>
<td>$\mu_d$: 1.30</td>
</tr>
<tr>
<td></td>
<td>$\sigma_d$: 0.11</td>
<td>$\sigma_d$: 0.57</td>
<td>$\sigma_d$: 0.15</td>
<td>$\sigma_d$: 0.37</td>
</tr>
</tbody>
</table>
In this chapter, it will be shown how optimal control techniques can be used to improve the motion primitive generation for a lattice-based motion planner. The procedure of manually specifying the connectivity, i.e., Step 2 in the state-lattice construction in Section 2.3, is both time consuming and non-trivial since it is heavily application dependent. In this chapter, we propose a novel framework which simultaneously selects an optimized connectivity for the system under consideration and computes optimal motions to obtain the motion primitive set. The optimization is performed based on a number of user-defined maneuvers for which the state-connectivity is at least partially unspecified. Furthermore, the proposed framework is extended to allow for an optimization of discretization parameters used to define the state-space discretization.

4.1 Motivation

In this chapter, we consider parameterized families of nonlinear systems:

\[ \dot{x}(t) = f(x(t), u(t), p_f), \]  

where \( x(t) \in \mathbb{R}^n \), \( u(t) \in \mathbb{R}^m \), and \( p_f \in \mathbb{R}^q \) denote the states, control signals, and the parameters that parameterizes the system family, respectively. Examples of parameters in common systems are platform dimensions or actuator limitations in cars, buses and trucks. The systems within the family are similar, but important parameters that affect optimal motions differ. If these variations are not handled in a structured way as proposed in this chapter, significant often manual work is necessary to develop lattice-based motion planners for each instance of these system families since the motion primitives need to be recomputed using the correct values of the parameters. Using model-based numerical optimal control, this is
4 Improved motion primitive optimization

fairly straight-forward given fixed boundary values. However, this work goes beyond that and introduces an efficient way to also optimize the boundary constraints in the set of OCPs to be solved for computing the motion primitives. Intuitively, the optimal choice of connectivity in the state-lattice construction (Step 2 in Section 2.3) for a vehicle with a long wheel-base is not the same as for a vehicle with short wheel-base, since the former is far less agile compared to the latter. As will be shown in this work, for good performance it does not only suffice to optimize the behavior between the boundary conditions, but also the conditions themselves.

One approach to select the boundary conditions in the set of OCPs to be solved in the motion primitive generation is by manual specification, which is typically done by an expert of the system [4, 51]. The manual procedure can be very time-consuming and if some of the system’s parameters are altered, the boundary conditions usually need to be carefully re-selected to not unnecessarily restrict the performance possible to obtain, and to maintain feasibility, during the following optimization of the motion primitives. Another widely used approach is to perform an exhaustive search to all neighboring states on the positional grid, either to the first feasible solution [68] (shortest path set), or to all positions within a user-defined distance [20, 66]. These methods perform well for systems where analytical solutions exist, such as for differentially flat systems [78] where efficient exact steering methods exist [39, 55]. However, for nonlinear systems that are not differentially flat, e.g., many truck and trailer systems [75], the exhaustive search will lead to an intractable computational burden. To avoid time consuming manual tuning or exhaustive search-based generation of the motion primitive set for each system instance within the family, this work proposes a novel framework that is automatically optimizing both the boundary constraint selection in the OCPs and the corresponding motions to compute a motion primitive set for general nonlinear systems. The proposed framework is described further in the next section.

4.2 Maneuver-based motion primitive generation

In this thesis, a maneuver is defined as a principle type of desired motion for a parameterized family of vehicle systems. A maneuver is computed by solving an OCP in a similar way as standard methods for computing motion primitives. The important difference is that in the OCP specifying the maneuver, freedom in selecting the terminal state is introduced by removing one or several of the terminal constraints. This means that a terminal state which is optimized for the selected system instance can be found using fast optimization techniques. The optimal choice of the terminal state is application dependent; for example the resulting optimal \( \pi/2 \) heading change maneuver for a truck is completely different from the solution for truck and trailer systems with different trailer lengths. This is illustrated in Figure 4.1.

From a maneuver specification and state-space discretization, it is possible to easily compute a solution to any desired instance in the family. Depending on which type of maneuver, the terminal state constraint \( x(T) = x_{\text{final}} \) in the motion primitive generation OCP (2.31) is replaced by a terminal manifold \( g(x(T)) = 0 \).
4.2 Maneuver-based motion primitive generation

![Graph showing optimal paths and steering angles]

**Figure 4.1:** The optimal paths and corresponding steering angles, using the same objective function, for a $\pi/2$ heading change maneuver (without constraints on terminal position). Clearly, the optimal solution changes depending on the system instance. The involved system instances are: a truck (green), a truck with a short dolly-steered trailer (red) and finally the same truck and trailer system with a longer trailer (blue).

where $g : \mathbb{R}^n \to \mathbb{R}^l$, $l < n$, and $n - l$ is the number of degrees of freedom for the terminal state constraint, which is a part of the maneuver specification. This modification generalizes the OCP in (2.31) for a traditional motion primitive to an OCP defining the more general maneuver:

$$
\begin{align*}
\text{minimize} & \quad \int_0^T \ell(x(t), u(t))dt \\
\text{subject to} & \quad x(0) = x_{\text{start}}, \quad g(x(T)) = 0, \\
& \quad \dot{x}(t) = f(x(t), u(t), p_f), \quad t \in [0, T] \\
& \quad x(t) \in \mathcal{X}_{\text{valid}}, \quad t \in [0, T] \\
& \quad u(t) \in \mathcal{U}, \quad t \in [0, T].
\end{align*}
$$

(4.2)

To illustrate how a maneuver can be defined, assume that the motion of the vehicle under consideration can be described by the states $x = (x_1, y_1, \theta_1)$ with $\dot{x} = [\cos(\theta_1), \sin(\theta_1), u_\theta]^T$. Assume that a discretized state space:

$$
x^d = \left(x_1^d, y_1^d, \theta_1^d\right),
$$

(4.3)
is given, where the position \((x_d^1, y_d^1)\) is discretized to a uniform grid with resolution \(r = 1\) m, and \(\theta_1^d \in \Theta = \{\frac{k\pi}{4}\}_{k=0}^7\).

--- Example 4.1: Heading change maneuver ---

A **Heading change maneuver** in (4.3) is given by a user-defined heading change to an adjacent heading in \(\Theta\). The initial state constraint is given by \(x_{\text{start}} = [0, 0, \theta_1,i]^T\), for all \(\theta_1,i \in \Theta\). The terminal state constraint is to arrive at heading \(\theta_{1,f} = \theta_{1,i} + \Delta_\theta\pi\), and \(\Delta_\theta\) is the user-defined parameter. The terminal constraint can hence be written as

\[
g(x(T)) = \theta_1(T) - \theta_{1,f} = 0,
\]

while the values of \(x_1(T)\) and \(y_1(T)\) are left as free variables to be optimized by the OCP solver. The red paths in Fig. 4.2 show one example of motion primitives for a heading change maneuver.

--- Example 4.2: Parallel maneuver ---

A **Parallel maneuver** in (4.3) is given by a user-defined lateral movement of \(c_{\text{lat}}\) to the same heading as the initial one. The initial state constraint is given by \(x_{\text{start}} = [0, 0, \theta_1,i]^T\), \(\forall \theta_1,i \in \Theta\). The terminal state constraint is to arrive at the same heading, i.e., \(\theta_1(T) = \theta_{1,i}\), with an additional terminal constraint that restricts \((x_1(T), y_1(T))\) to the line given by \(\cos(\theta_{1,f})y_1(T) + \sin(\theta_{1,f})x_1(T) = c_{\text{lat}}\). Hence, the total terminal state constraint for a parallel maneuver can be defined as

\[
g(x(T)) = \begin{bmatrix} \theta_1(T) - \theta_{1,f} \\ \cos(\theta_{1,f})y_1(T) + \sin(\theta_{1,f})x_1(T) - c_{\text{lat}} \end{bmatrix} = 0.
\]

(4.5)

The blue paths in Figure 4.2 show one example of motion primitives for a parallel maneuver.

---

A flow chart of the proposed framework is shown in Figure 4.3 and described in Algorithm 4.1. The inputs are given by the state discretization, the system model, an objective function and a set of user-defined maneuvers \(\mathcal{M}\). These inputs are used to set up the OCP solver as well as to define the initial constraint and terminal manifold for each maneuver (line 2 and 3 in Algorithm 4.1), which results in a number of OCPs to be solved (one for each maneuver).

One aspect to consider is that the resulting motion primitive generation problem, transcribed using, e.g., multiple shooting or direct collocation, becomes a mixed-integer nonlinear program (MINLP) due to the fact that formally the free terminal states are only allowed to take on the discrete values defined by the state discretization in \(X_d\). In the maneuver OCP in (4.2), a continuous relaxation of this MINLP is solved, since the terminal manifold is not restricted to states in \(X_d\). To ensure that the motion primitives are feasible with the given state discretization, any heuristic applicable for solving MINLPs can be utilized [12]. One commonly used alternative is to apply a *rounding heuristic* from the obtained continuous OCP.
solution $x_i(\cdot)$, where the closest terminal state alternatives (in Euclidean sense) represented in $X_d$ from $x_i(T_i)$ are computed and the result with lowest objective function value is chosen as resulting motion primitive $m_i$ (line 7 in Algorithm 4.1). Finally, if the system is orientation invariant, rotational symmetries of the system can be exploited to reduce the number of OCPs that need to be solved, in the same way as presented in [66] (here, line 8 in Algorithm 4.1). The symmetries of the system is something that should be accounted for already in the maneuver interpretation where the number of OCPs that need to be solved is decided.

The benefits of using the proposed method outlined in Algorithm 4.1 instead of completely manually specifying the connectivity are:

- The maneuver definitions can be **reused** to compute motion primitives for any desired instance of a parameterized family of systems, which reduces the manual workload of the operator since the re-connection of the state lattice will be handled automatically by the proposed framework.

- Both the connectivity and the motions will be **optimized** for the specific system instance which gives a lower total objective function value in the online planning phase.

- It is possible to automatically generate different levels of aggressiveness for the same maneuver by simply changing $Q$ in (2.32), which can be useful to obtain smooth final solutions while still being able to express the reachability of the vehicle sufficiently well.

Note that the proposed framework goes beyond solving an OCP with new parameters, it also automatically optimizes the connectivity in the graph since some
Algorithm 4.1: Maneuver-Based Motion Primitive Generation

1: **Input**: \( \mathcal{X}_d \), system instance \((\mathcal{X}, \mathcal{U}, f, p_f)\), cost function \(\ell(x,u)\) and a set of maneuvers \(\mathcal{M}\)

2: \( N_{\text{OCP}}, x_{\text{start}}, g(x_{\text{final}}) \leftarrow \text{maneuver_interpreter}(\mathcal{M}, \mathcal{X}_d) \)

3: \( \text{OCP\_solver} \leftarrow \text{setup\_problem}(\mathcal{X}, \mathcal{U}, f, p_f, \ell) \)

4: for \(i = 1 : N_{\text{OCP}}\) do

5: \( \text{OCP\_solver.set\_boundary\_constraints}(x_{i,\text{start}}, g_i(x_{i,\text{final}})) \)

6: \( x_i(\cdot), u_i(\cdot), T_i \leftarrow \text{OCP\_solver.solve}() \)

7: \( m_i \leftarrow \text{ensure\_connectivity}(x_i(\cdot), u_i(\cdot), T_i, \mathcal{X}_d, \text{OCP\_solver}) \)

8: \( P_i \leftarrow \text{exploit\_symmetries}(m_i) \)

9: \( P \leftarrow P \cup P_i \)

10: end for

maneuver specifications, e.g. the terminal position of the system, are left to be optimized by the framework. Industrially, this implies that motion planners can easily be prepared for new products with, e.g., another wheel-base. In that situation, the entire motion primitive set, including the state-connectivity, can be re-optimized only requiring a “single-click” from the developer.

Algorithm 4.1 shares similarities with the motion primitive set generation frameworks described in [66, 68]. The method in [68] generates a minimal set of motion primitives that connects all neighboring discrete states that do not represent the position of the vehicle by searching for the nearest feasible solution, and the method in [66] searches for a feasible terminal state within a user-defined distance. Both are performed by an exhaustive search for feasible terminal state constraints by cycling through all possible candidates starting at the origin. However, the framework proposed in this chapter has some major advantages compared to the previously suggested methods:

+ The computation time for the proposed method is orders of magnitude faster, and it scales better with state-space discretization and problem dimension.

This is since the search for a feasible terminal state in Algorithm 4.1 starts...
from an optimal terminal state not restricted to $X_d$, and not from the origin as in [66, 68].

+ Any user-defined (smooth) objective function can be minimized. In [68], the only objective function considered is minimum path length, since the search for a candidate terminal state is terminated as soon as a feasible candidate is found. Therefore, the smoothness term in (2.32) is always omitted in [68].

### 4.3 Extended maneuver optimization

The framework described in the previous section allows for an optimization of the terminal states for a user-defined discretization of the state space. In this section, the objective is to go even further and simultaneously optimize discretization parameters that are used in the definition of the state-space discretization. This is different from the framework presented in the previous section, where the complete state-space discretization is assumed to be manually specified. The extension of optimizing some of the discretization parameters is most easily illustrated by an example. Assume that the vehicle under consideration can be described by the states $x = (x_1, y_1, \theta_1, \alpha)$. Furthermore, assume that a discretized state space:

$$x^d = \left(x^d_1, y^d_1, \theta^d_1, \alpha^d\right),$$

(4.6)

is selected, where the position $(x^d_1, y^d_1)$ is discretized to a uniform grid with resolution $r = 1$ m, and the heading is $\theta^d_1 \in \Theta = \{k\pi/4\}_{k=0}^7$. Finally, the steering angle discretization is selected such that $\alpha^d \in \Phi = \{\alpha^d_1, \alpha^d_2, \alpha^d_3\}$, i.e., the cardinality of the steering angle discretization is chosen to three, but the actual values of the parameters $\alpha^d_i \in \Phi$ are not yet selected. Now, it is possible to define maneuvers in the same way as in Section 4.2, but with the difference that $\alpha^d_i$ are left as free parameters to be optimized over for a given system instance.

The benefits of optimizing over some of the discretization parameters, instead of predefining them, are twofold. First, the parameters will be selected automatically such that they are optimized for any given system instance. Second, deciding the actual values of these parameters in the state-space discretization are not required to be done manually by a human operator, which further reduces the time spent in the state-lattice construction phase. However, the extra freedom added by also optimizing the discretization parameters leads to that each maneuver OCP, in the form of (4.2), is dependent of other maneuver OCPs. This is the case since when sequential maneuvers are pieced together online by the lattice-based motion planner, the final state of one maneuver and the initial state of the next one need to be compatible, which is in the standard case ensured by manual work. These compatibility constraints limit the freedom of choice regarding the discretization parameters. Thus, to ensure that the maneuvers can be pieced together online, they need to be computed simultaneously in one large OCP, which can be posed as:
4 Improved motion primitive optimization

\begin{align}
\text{minimize} & \quad J_{\text{coup}} = \sum_{i=1}^{K} w_i \int_0^{T_i} \ell(x_i(t), u_i(t)) dt \\
\text{subject to} & \quad g_{\text{start}}(x_i(0), x_{\text{opt}}) = 0, \quad g_{\text{final}}(x_i(T_i), x_{\text{opt}}) = 0, \\
& \quad \dot{x}_i(t) = f(x_i(t), u_i(t)), \quad t \in [0, T_i], i \in \mathbb{Z}_{1,K} \\
& \quad u_i(t) \in U, \quad t \in [0, T_i], i \in \mathbb{Z}_{1,K} \\
& \quad x_i(t) \in X_{\text{valid}}, \quad t \in [0, T_i], i \in \mathbb{Z}_{1,K} 
\end{align} \tag{4.7}

where the solution is a set of maneuvers \( \{x_i(\cdot), u_i(\cdot), T_i\}_{i=1}^{K} \) that are used to compute the motion primitive set. The variable \( x_{\text{opt}} \in \mathbb{R}^p \) represents the parameters within the discretization that are left for optimization. These parameters are used to loosely couple the maneuvers by the initial and terminal constraints, which ensures that maneuvers can be combined online. The function \( g_{\text{start}} : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^n \) is used to define the initial constraints and \( g_{\text{final}} : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^l \) is used to define the terminal manifold, where again \( n-l \) is the number of degrees of freedom for the terminal state. The main difference in (4.7) compared to the maneuver OCP in (4.2) is that all maneuvers are optimized simultaneously, instead of separate OCPs being solved for each maneuver. Furthermore, the parameters \( w_i \) are added to the problem formulation. They represent weight parameters used in the objective function, which enable the possibility to indicate which maneuvers that are more important than others during the simultaneous motion primitive computation. The choice of these weight parameters are further discussed in Section 4.3.1.

The OCP in (4.7) is a much larger OCP than the maneuver OCP in (4.2), making it more computationally demanding to solve. However, due to the maneuvers only being loosely coupled through the added optimization variables \( x_{\text{opt}} \), the computational effort of solving the problem will still be tractable if solvers that utilize this sparsity is used, such as IPOPT and WORHP. Another possibility with the loosely coupled formulation is that it is likely that methods employing parallel computations can be utilized, such as the ones presented in [63].

The extension of also optimizing some of the discretization parameters when computing the motion primitive set is outlined in Algorithm 4.2, which is an extension of the previously introduced maneuver-based motion primitive generation in Algorithm 4.1. First, the maneuver interpreter is used in a similar way as in Algorithm 4.1, which outputs the functions used to define the initial constraints and terminal manifolds for the maneuvers in the coupled optimization problem (4.7). In the next step, given by line 3 in Algorithm 4.2, this coupled optimization problem is solved. Here, both the values of the discretization parameters \( x_{\text{opt}} \) in the discretized state space \( X_d \) as well as the terminal states in the maneuvers are optimized.

As described in Section 4.2, the motion primitive generation problem, transcribed using, e.g, multiple shooting, is formally an MINLP due to the fact that the terminal states are only allowed to take on the discrete values defined by \( X_d \). Hence, at line 5 in Algorithm 4.2, the values of \( x_{\text{opt}} \) are fixed to the values obtained in the previous optimization step. Then, a separate search for each maneuver is
Algorithm 4.2: Extended Maneuver-Based Motion Primitive Generation

1: **Input:** Partially specified state-space discretization \( \mathcal{X}_d \), states for optimization \( x_{\text{opt}} \), system instance \((\mathcal{X}, \mathcal{U}, f, p_f)\), cost function \( \ell(x, u) \) and a set of maneuvers \( \mathcal{M} \).
2: \( N_{\text{OCP}}, g_{\text{start}}(x_{\text{start}}), g_{\text{final}}(x_{\text{final}}) \leftarrow \text{maneuver_interpreter}(\mathcal{M}, \mathcal{X}_d, x_{\text{opt}}) \)
3: \( \{x_i(\cdot), u_i(\cdot), T_i\}_{i=1}^{N_{\text{OCP}}}, x_{\text{opt}} \leftarrow \text{solve_coupled_problem}(\mathcal{X}, \mathcal{U}, f, p_f, \ell, g_{\text{start}}(x_{\text{start}}), g_{\text{final}}(x_{\text{final}})) \)
4: \textbf{for} \( i = 1 : N_{\text{OCP}} \) \textbf{do}
5: \( x_i(\cdot), u_i(\cdot), T_i \leftarrow \text{ensure_connectivity}(x_i(\cdot), u_i(\cdot), T_i, \mathcal{X}_d, x_{\text{opt}}) \)
6: \textbf{end for}
7: \( \{m_i\}_{i=1}^{N_{\text{OCP}}}, x_{\text{opt}} \leftarrow \text{reoptimize_values}(\mathcal{X}, \mathcal{U}, f, p_f, \ell, \{x_i(\cdot), u_i(\cdot), T_i\}_{i=1}^{N_{\text{OCP}}}) \)
8: \textbf{for} \( i = 1 : N_{\text{OCP}} \) \textbf{do}
9: \( P_i \leftarrow \text{exploit_symmetries}(m_i) \)
10: \( P \leftarrow P \cup P_i \)
11: \textbf{end for}

---

performed (similar to what is done in Algorithm 4.1), where a heuristic is applied from each solution to find a terminal state constraint for the maneuver that is compatible with \( \mathcal{X}_d \). After this step, the values of \( x_{\text{opt}} \) might not represent the best possible choice anymore since a subset of the terminal state constraints in the maneuvers have been modified. Thus, in a subsequent step, given by line 7 in Algorithm 4.2, the terminal state constraints in the maneuvers are fixed, and the value of \( x_{\text{opt}} \) is re-optimized. This re-optimization ensures that the best discretization parameter values with respect to the final optimized connectivity is obtained. Finally, symmetries are exploited in the same way as in Algorithm 4.1 to produce the complete motion primitive set \( P \).

4.3.1 Selecting motion primitive weights

For a given motion planning problem, the objective function value of a solution produced by a lattice-based motion planner (by solving (2.33)) can also be formulated as:

\[
J_{\text{lat}} = \sum_{i=1}^{K} k_i L_u(m_i),
\]

where \( k_i \) represents the number of times the motion primitive \( m_i \) is applied in the solution. Furthermore, \( \sum k_i \) is equal to the length of the motion primitive sequence \( N \). Moreover, the objective function minimized in the coupled maneuver optimization problem (4.7) is given by:

\[
J_{\text{coup}} = \sum_{i=1}^{K} w_i \int_0^{T_i} \ell(x_i(t), u_i(t))dt = \sum_{i=1}^{K} w_i L_u(m_i).
\]
It is relatively easy to realize that the best choice of weight parameters $w_i$, for a given motion planning problem with a solution which objective function value can be written as (4.8), is to use $w_i = k_i$ in (4.7). However, due to the parameters $k_i$ in (4.8) are given as a result of applying the lattice-based motion planner online, the optimal choice of $k_i$ is not known prior to solving the maneuver OCP in (4.7). Instead, the values of the weight parameters need to be appropriately selected such that the motion primitive generation can be performed offline. One reasonable choice is to treat the number of times each motion primitive is used as a random variable, and use an objective function in (4.7) which represents the expected motion primitive usage:

$$J_{\text{coup}} = \sum_{i=1}^{K} E(k_i)L_u(m_i) = \sum_{i=1}^{K} E(k_i) \int_0^{T_i} \ell(x_i(t), u_i(t))dt, \quad (4.10)$$

where $E(k_i)$ is the expected number of times that the motion primitive $m_i$ is used during online planning. If usage statistics are available from solving motion planning problems online using the lattice-based motion planner, these can be used as weights in (4.7). Before any usage statistics are available, the most simple choice of weights is to use $w_i = 1$ for all maneuvers. This can be interpreted as that all maneuvers are expected to be used equally many times online by the lattice-based motion planner. Another possibility is to approximate the expected usage by solving, in simulations, a series of problems within a certain class. A practically relevant problem class is to solve motion planning problems in a model of the actual (static) environment where the system operates. The solutions, or usage statistics, from these simulations could then be used to select the weights in the objective function of (4.7) when computing the next set of improved motion primitives.

When usage statistics from online motion planning problems are available, it is possible to repeatedly solve the coupled maneuver optimization problem in (4.7), with the continuously collected usage statistics as weights. How often the motion primitive set needs to be recalculated, and how much of the past statistics that is useful for determining values of the weight parameters are both expected to be application dependent. Furthermore, it might be of interest to add a forgetting factor on the usage statistics if the system is operating within an environment that involves dynamic obstacles.

### 4.4 Numerical results

In this section, it is illustrated how the proposed frameworks can be applied to compute motion primitives for a diverse family of systems including cars and trucks with and without trailers. The implementation is done in Python, where CasADi combined with the NLP solver IPOPT [83] is used as a backbone to solve the OCPs that define the motion primitives.
4.4 Numerical results

\[\begin{align*}
\dot{x}_1(t) &= v_1 \cos \theta_1(t), \\
\dot{y}_1(t) &= v_1 \sin \theta_1(t), \\
\dot{\theta}_1(t) &= v_1 \tan \alpha(t)/L_1, \\
\dot{\alpha}(t) &= \omega(t), \\
\dot{\omega}(t) &= u_\omega(t),
\end{align*}\] (4.11)

Here, \(x_c = (\bar{x}_c, \alpha, \omega) = (x_1, y_1, \theta_1, \alpha, \omega)\) is the state vector which represents the position, heading, steering angle and steering angle rate, and \(L_1\) is the wheel-base of the car-like vehicle. The control signals are \(u = (v_1, u_\omega)\), where \(v_1\) is the longitudinal velocity of the rear axle of the vehicle and \(u_\omega\) the steering angle acceleration. In these numerical results, we consider path planning and therefore constrain the longitudinal velocity as \(v_1 = \{1, -1\}\) to represent the possibility for both forward and backward motions.

The truck and trailer system is a general 2-trailer with car-like truck (illustrated in Figure 4.4). With the state vector

\[\begin{align*}
\bar{x}_t &= (x_3, y_3, \theta_3, \beta_3, \beta_2), \\
x_t &= (\bar{x}_t, \alpha, \omega),
\end{align*}\] (4.12)

where \((x_3, y_3)\) and \(\theta_3\) represent the center of the rear axle of the trailer and the heading of the trailer, respectively. Moreover, \(\beta_3\) and \(\beta_2\) denote the joint angles between the trailer and the truck. The kinematic model of this system is [2]:

Figure 4.4: A schematic illustration of the truck and trailer system [52].

4.4.1 Vehicle models

The model for a car and a truck without any trailers is based on a kinematic car-like vehicle model [45]:
\[
\begin{align*}
\dot{x}_3 &= v_1 \cos \beta_3 \cos \beta_2 \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right) \cos \theta_3, \\
\dot{y}_3 &= v_1 \cos \beta_3 \cos \beta_2 \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right) \sin \theta_3, \\
\dot{\theta}_3 &= v_1 \frac{\sin \beta_3 \cos \beta_2}{L_3} \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right), \\
\dot{\beta}_3 &= v_1 \cos \beta_2 \left( \frac{1}{L_2} \left( \tan \beta_2 - \frac{M_1}{L_1} \tan \alpha \right) - \frac{\sin \beta_3}{L_3} \left( 1 + \frac{M_1}{L_1} \tan \beta_2 \tan \alpha \right) \right), \\
\dot{\beta}_2 &= v_1 \left( \frac{\tan \alpha}{L_1} - \frac{\sin \beta_2}{L_2} + \frac{M_1}{L_1 L_2} \cos \beta_2 \tan \alpha \right), \\
\dot{\alpha} &= \omega, \\
\dot{\omega} &= u_\omega.
\end{align*}
\] (4.13)

The control signals for the truck and trailer system coincide with the control signals for the car-like vehicle (4.11), i.e., \( u = (v_1, u_\omega) \). The system parameters for the general 2-trailer with car-like truck are the geometric lengths (in meters) \( L_3, L_2, L_1 \) and \( M_1 \), which relate the distances and connections between the vehicle bodies. The truck and trailer system can attain \textit{circular equilibrium configurations}, where a constant steering angle \( \alpha_e \) results in constant joint angles \( \beta_{3,e} \) and \( \beta_{2,e} \) given by

\[
\begin{align*}
\beta_{3,e} &= \text{sign} \left( \alpha_e \right) \arctan \left( \frac{L_3}{R_3} \right), \tag{4.14a} \\
\beta_{2,e} &= \text{sign} \left( \alpha_e \right) \left( \arctan \left( \frac{M_1}{R_1} \right) + \arctan \left( \frac{L_2}{R_2} \right) \right), \tag{4.14b}
\end{align*}
\]

where \( R_1 = L_1 / |\tan \alpha_e|, R_2 = \sqrt{R_1^2 + M_1^2 - L_2^2} \) and \( R_3 = \sqrt{R_2^2 - L_3^2} \) [2]. If such configurations are kept constant, the axles of the vehicle will move along circles with radii determined by \( R_1, R_2 \) and \( R_3 \). For more details regarding the model of the truck and trailer system, the reader is referred to [2].

The physical constraints on the steering angle, the steering angle rate and acceleration are the same for both systems, and are given by

\[
\mathcal{X}_{\text{valid}} = \left\{ x(t) : -\alpha_{\text{max}} \leq \alpha(t) \leq \alpha_{\text{max}}, -\omega_{\text{max}} \leq \omega(t) \leq \omega_{\text{max}} \right\},
\]

and \( U = \{ u_\omega(t) : -u_{\alpha,\text{max}} \leq u_\omega(t) \leq u_{\alpha,\text{max}} \} \). In the numerical examples, the bounds are given by \( \alpha_{\text{max}} = \pi/4 \text{ rad}, \omega_{\text{max}} = 0.5 \text{ rad/s} \) and \( u_{\alpha,\text{max}} = 40 \text{ rad/s}^2 \).

For the truck and trailer system, the direction of motion determines the stability properties of the system; in backward motion the dynamics for the joint angles \( \beta_3 \) and \( \beta_2 \) are unstable, while they are stable in forward motion. To obtain a more reliable convergence for the NLP solver, we have observed that it can be
beneficial to compute the motion primitives for backward motion from the goal state to the initial state in forward motion for which the system is stable. Since the truck and trailer system is on the form of Lemma 1 in [52], it is possible, according to Theorem 1 in [52], to obtain an optimal backward motion \((x^*_t,b(\tau),u^*_b(\tau)), \tau \in [0,T]\) that brings the system from \(x_{\text{start}}\) to \(x_{\text{final}}\), from the optimal forward motion \((x^*_t,f(t),u^*_f(t)), t \in [0,T]\). This can be achieved by the state and control transformation:

\[
\begin{align*}
x^*_t,b(\tau) &= \left(\bar{x}_t,f(T-\tau), \alpha^*_f(T-\tau), -\omega^*_f(T-\tau)\right), \\
u^*_b(\tau) &= \left(-1, u^*_f(T-\tau)\right), \quad \tau \in [0,T],
\end{align*}
\]

where the optimal forward motion is obtained by solving the motion primitive OCP (4.2) from \(x_{\text{final}}\) to \(x_{\text{start}}\). Here, it is assumed that the cost function to be minimized in (4.2) satisfies \(\ell(\bar{x},\alpha,\omega,u_\omega) = \ell(\bar{x},\alpha,-\omega,u_\omega)\) and that \(\omega = 0\) in both \(x_{\text{start}}\) and \(x_{\text{final}}\). Note that it is only the steering angle rate \(\omega\) and the velocity \(v_1\) that change signs, while the other states and controls are fully time-reversed. For a detailed description of this property, the reader is referred to [52].

### 4.4.2 State-space discretizations

In the state-lattice formulation, the state space of the vehicles has to be discretized. The state space for the car-like vehicle is discretized as \(x^d = (x^d_1, y^d_1, \theta^d_1, \alpha^d, \omega^d)\), and for the truck and trailer system as \(x^d_t = (x^d_3, y^d_3, \theta^d_3, \beta^d_3, \beta^d_2, \alpha^d, \omega^d)\). The position for both vehicles \((x^d_i, y^d_i)\) are discretized to a uniform grid with resolution \(r = 1\) meter. The headings are irregularly discretized \(\theta^d_i \in \Theta = \{\theta_k\}_{k=0}^{N-1}\) to be able to generate short, straight motions from all headings [68]. The steering angle is discretized according to \(\alpha^d_i \in \Phi = \{-0.1, 0.0.1\}\). For the truck and trailer case, in analogy with [51], \(\alpha^d_c\) is chosen to implicitly determine the discretization of the joint angles \(\beta^d_{3,e}\) and \(\beta^d_{2,e}\), where the vehicle is constrained to a circular equilibrium configuration at each discrete state in the state lattice. This reduces the state dimension from seven to five states for the truck and trailer system, which imply that the state-space discretization for both vehicles can be described by \(x^d\). Finally, to ensure that \(\alpha\) is a continuously differentiable function in time when multiple motions are combined during online planning, the steering angle rate \(\omega^d\) is constrained to zero in the state-space discretization. Note however that on the path between two discretized states, the systems can take any feasible vehicle configuration.

### 4.4.3 Maneuver specifications

The maneuvers that are typically used in a motion primitive set for the considered systems are straight, heading change, parallel or circular maneuvers. A straight maneuver is computed by connecting to the closest position on the grid that is aligned with the initial heading of the vehicle. The heading change and parallel

\[\text{It is trivial to extend the result to any feasible value of } \omega \text{ in } x_{\text{start}} \text{ and } x_{\text{final}}.\]
Figure 4.5: The resulting paths from an optimized heading change maneuver of $\pi/2$ for a car-like vehicle using different values of $\lambda$ in the objective function (4.16). Clearly, the choice of $\lambda$ has a significant impact on the resulting maneuver.

Maneuvers are defined in a similar way as in Example 4.1–4.2. The circular maneuvers are here used in combination with the heading change maneuver defined in Example 4.1, where each heading change maneuver is allowed to start from $\alpha_i^e \in \Phi$ and end in a neighboring $\alpha_f^e \in \Phi$. Since the state-space discretization for the truck and trailer system is chosen such that $\alpha_i^d$ determines the values of $\beta_{3,e}^d$ and $\beta_{2,e}^d$, all maneuvers apply directly on both systems in (4.11) and (4.13).

The cost function that is used to compute the motion primitives for the vehicles in this section is given by:

$$\ell(x(t), u(t)) = 1 + \lambda \left( \alpha^2 + 10 \omega^2 + u_{\omega}^2 \right).$$  \hspace{1cm} (4.16)

When computing backward motions for the truck and trailer case, additional quadratic penalties for the joint angles $\beta_3$ and $\beta_2$ are added to promote solutions that have a small risk of leading to a jackknife state\(^2\) during path execution, i.e,

$$\ell_b(x(t), u(t)) = 1 + \lambda \left( \beta_3^2 + \beta_2^2 + \alpha^2 + 10 \omega^2 + u_{\omega}^2 \right).$$  \hspace{1cm} (4.17)

One choice of value for the trade-off parameter $\lambda$, which will be used in this section, is to use $\lambda = 1$. The impact of $\lambda$ on the optimal solution for the car-like vehicle can be seen in Figure 4.5 for a heading change maneuver of $\pi/2$.

\(^2\)In a jackknife state, the vehicle system attains an undesirable folding of the vehicle bodies, which can lead to physical damage of the vehicle.
4.4.4 Experimental results

To illustrate the computational effectiveness of the proposed framework, four different sets of maneuvers are considered, which are defined and described in Table 4.1. The computation times for generating the complete set of motion primitives for several system instances are found in Table 4.2. The system instances are a truck \((L_1 = 4.66)\), a small car \((L_1 = 2.5)\) and a truck and trailer system \((4.13)\) with two different trailer lengths \((L_3 = 8\) and \(L_3 = 6)\). For all settings using the proposed framework in this chapter \((P_1-P_2\) in Table 4.2), the complete set of motion primitives are generated within minutes. This time would have been significantly higher if the explicit state-space connectivity for all maneuvers would have been chosen manually for each system instance. These results illustrate that the proposed framework could even be used online to enable automatic re-optimization of, e.g., an autonomous truck and trailer system if a new trailer with different length is to be connected to the truck.

The proposed framework is also shown to outperform the suggested method in [68] \((P_3-P_4\) in Table 4.2), especially for the truck and trailer system. For this system instance, the algorithm in [68] failed to compute the motion primitive set within ten hours. The main reason for the drastic change in computation times is that the method in [68] spends most of the computations on trying to solve infeasible problems \((N_{\text{inf}}\) in Table 4.2) since it does not start the search from a feasible terminal state, but instead from the origin.

To demonstrate why it is important to choose the state-space connectivity based on the platform to be controlled, a planning scenario from several initial states in an environment with obstacles is used (illustrated in Figure 4.6) using a standard A* search to solve the motion planning problems. To ensure collision avoidance in the obtained solution, the obstacles are represented in the A* search using axis aligned bounding boxes, and the vehicles are represented by bounding circles [45]. In the comparison, the optimized connectivity for each one of the tested system instances are computed using the proposed framework and then reused on each one of the remaining system instances. The results obtained from the planning scenario can be found in Table 4.3. When using a connectivity optimized for a system instance of smaller dimension, the motion primitive generation becomes infeasible, which is the case for the truck and trailer system, but also when using a connectivity optimized for a small car on a truck. In the opposite case, when using a connectivity optimized for a system instance of larger dimension, the motion primitive generation still finds feasible solutions. However, in Table 4.3 it can be seen that the average objective function value for the obtained solutions increases due to that the full potential of the maneuverability for the specific system instance is not fully utilized. One example of this is shown in Figure 4.6 where the car needs to travel an unnecessarily long route when using a connectivity optimized for a truck. Clearly, what is proposed in this work also has a value in practice.
Figure 4.6: Planning example with obstacles (red boxes) solved from several initial positions with $\theta_i^{\text{init}} = \pi/2$ (indicated by the area within the dotted lines) to a terminal state in $(x_i^{\text{final}}, y_i^{\text{final}}, \theta_i^{\text{final}}) = (13, 4, 0)$. The systems used in the example are (from left to right) a truck and trailer system, a truck ($L_1 = 4.66$) and a small car ($L_1 = 2.5$). The solutions illustrated as examples in the figure are for the car. Blue: Connectivity optimized for the car. Green: Connectivity optimized for the truck. A summary of the results can be found in Table 4.3.
4.4 Numerical results

Table 4.1: A description of the different used motion primitive sets. $|\Theta|$ defines the number of $\theta^d$ in $\Theta$, $\Delta_{\theta}^{\max}$ defines which heading change maneuvers to generate (from $\pm 1$ to $\pm \Delta_{\theta}^{\max}$) and $n_{\text{par}}$ defines the number of parallel maneuvers from each initial heading. $P_3$ and $P_4$ contains no parallel maneuvers since they cannot be represented using the algorithm in [68]. Finally, $n_{\alpha_e}$ defines the number of $\alpha_e^d \in \Phi$ and $n_{\text{prim}}$ the total number of motion primitives after exploiting system symmetries.

| $\mathcal{P}$ | Alg. Model | $|\Theta|$ | $\Delta_{\theta}^{\max}$ | $n_{\text{par}}$ | $n_{\alpha_e}$ | $n_{\text{prim}}$ |
|--------------|-------------|--------------|-----------------|----------------|----------------|---------------|
| $P_1$        | Alg. 4.1 (4.11) | 16           | 4               | 4              | 3              | 1312          |
| $P_2$        | Alg. 4.1 (4.13) | 16           | 4               | 4              | 3              | 1312          |
| $P_3$        | [68] (4.11)   | 16           | 4               | 0              | 3              | 1056          |
| $P_4$        | [68] (4.13)   | 16           | 4               | 0              | 3              | 1056          |

Table 4.2: The total computation time, $T_{\text{tot}}$, for generating and storing the complete set of motion primitives $P_i$ defined in Table 4.1. The values of $L_1$, $L_2$, $L_3$ and $M_1$ represent the system parameter values used in (4.11) and (4.13). $N_{\text{inf}}$ represents the number of infeasible problems encountered during the motion primitive generation.

<table>
<thead>
<tr>
<th>$\mathcal{P}$</th>
<th>$L_1$ [m]</th>
<th>$L_2$ [m]</th>
<th>$L_3$ [m]</th>
<th>$M_1$ [m]</th>
<th>$T_{\text{tot}}$ [s]</th>
<th>$N_{\text{inf}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>4.66</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>25.4</td>
<td>18</td>
</tr>
<tr>
<td>$P_1$</td>
<td>2.5</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>28.6</td>
<td>36</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4.66</td>
<td>3.75</td>
<td>8.0</td>
<td>1.67</td>
<td>314.2</td>
<td>208</td>
</tr>
<tr>
<td>$P_2$</td>
<td>4.66</td>
<td>3.75</td>
<td>6.0</td>
<td>1.67</td>
<td>177.8</td>
<td>41</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.66</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>3947</td>
<td>21062</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4.66</td>
<td>3.75</td>
<td>6.0</td>
<td>1.67</td>
<td>$&gt;36000$</td>
<td>$&gt;46630$</td>
</tr>
</tbody>
</table>

Table 4.3: Resulting relative objective function value for the motion planning scenario in Figure 4.6 (comparable row-by-row). The conclusions drawn are that when a connectivity optimized for a system instance with larger dimensions is used, the vehicles are not able to utilize its maneuverability. In the opposite case, the motion primitive generation fails due to infeasibility.

<table>
<thead>
<tr>
<th>Actual $P_2$, $L_3 = 6$ m</th>
<th>Opt. for $P_2$, $L_3 = 6$ m</th>
<th>$P_2$, $L_3 = 6$ m</th>
<th>$P_1$, $L_1 = 4.66$ m</th>
<th>$P_1$, $L_1 = 2.5$ m</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_2$, $L_3 = 6$ m</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$P_1$, $L_1 = 4.66$ m</td>
<td>1.18</td>
<td>1.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$P_1$, $L_1 = 2.5$ m</td>
<td>1.31</td>
<td>1.14</td>
<td>1.0</td>
<td></td>
</tr>
</tbody>
</table>
4.4.5 Extended maneuver optimization

The aim of this section is to illustrate the benefits of using the extended maneuver-based motion primitive generation in Section 4.3. This is done by leaving the discretization parameters which concerns the steering angle as free variables, i.e, $\alpha_e^d \in \Phi = \{-\bar{\alpha}, 0, \bar{\alpha}\}$. The difference compared to the previous state-space discretization in Section 4.4.2 is that $\bar{\alpha}$ is introduced as a decision variable to be optimized for each system instance, instead of being manually set to 0.1. Note that one value of $\alpha_e^d$ is still explicitly set to zero such that short, straight motions can be computed. The maneuvers from Section 4.4.3 affected are hence the combined circular and heading change maneuvers, since the straight and parallel maneuvers are computed with initial and terminal values of $\alpha$ constrained to zero.

The combined circular and heading change maneuvers, together with the value of $\bar{\alpha}$, are optimized for the system instances used in Table 4.2, i.e., a truck without trailer ($L_1 = 4.66$ m), a small car ($L_1 = 2.5$ m) and a truck and trailer with a long ($L_3 = 8$ m) or short ($L_3 = 6$ m) trailer. Since the maneuvers are coupled through the dependency on the decision variable $\bar{\alpha}$, the framework presented in Algorithm 4.2 is here used to compute the motion primitive set. For the truck and trailer system, the pairs of alpha angle and joint angles, i.e., $(\alpha, \beta_2, \beta_3)$, in the state-space discretization are here not required to represent a circular configuration anymore. Hence, fulfilling (4.14) is no longer a necessity. Instead, $\beta_2^d \in \{-\bar{\beta}_2, 0, \bar{\beta}_2\}$ and $\beta_3^d \in \{\bar{\beta}_3, 0, \bar{\beta}_3\}$ are introduced, where the variables $\beta_2$ and $\beta_3$ are to be optimized by the framework in the same way as $\bar{\alpha}$. The maneuvers obtained earlier from Algorithm 4.1 with $\bar{\alpha} = 0.1$ are used as initialization to the solver of the resulting coupled OCP in (4.7). They are also used to compare the difference in the objective function value of (4.7) with the maneuvers computed by the extended framework in Algorithm 4.2. In this numerical example, the weight parameters $w_i$ are equal to one, and the same cost functions as in Section 4.2 are used to form the objective function in (4.7).

When both the terminal state constraints and $\bar{\alpha}$, $\bar{\beta}_2$ and $\bar{\beta}_3$ are optimized, the rounding heuristic described in Section 4.2 proved to be insufficient for a satisfactory performance. Therefore, the rounding heuristic is in this section augmented with a search that starts from the solution of a problem where two half-plane constraints are added. The aim with adding these constraints is to restrict the search to an area where it is likely to find the best choice of terminal state within $\mathcal{X}_d$. The constraints are determined from the solution of the coupled maneuver problem in (4.7) and the terminal heading in the maneuver. The augmented heuristic is illustrated in Figure 4.7 for one heading change maneuver for the truck and trailer system with a short trailer. First, the coupled problem is solved to compute an optimal maneuver with no restrictions on the terminal position in the maneuver. This is followed by a step where a constrained solution is computed, where the heuristic constraints are added to the problem formulation. In the specific case illustrated in Figure 4.7, the constrained solution is significantly different from the coupled solution, which is indicated by the infeasible and feasible regions for a terminal position within $\mathcal{X}_d$. In a final step, the closest terminal states within the state-space discretization from the coupled and constrained solutions are eval-
Table 4.4: Extended maneuver-based motion primitive optimization. Here, total objective function value for all heading change maneuvers with optimization of $\bar{\alpha}$ in the state-space discretization is compared with the solutions obtained with Algorithm 4.1 using the user-defined value of $\bar{\alpha} = \bar{\alpha}^{\text{init}} = 0.1$. The reduction in objective function value is represented by $\delta J_{\text{coup}}$. The used system instances are a truck without trailer ($L_1 = 4.66$ m), a small car ($L_1 = 2.5$ m), and a truck and trailer with long ($L_3 = 8$ m) and short trailer ($L_3 = 6$ m).

<table>
<thead>
<tr>
<th>$\mathcal{P}$</th>
<th>$L_1$ [m]</th>
<th>$L_3$ [m]</th>
<th>$\bar{\alpha}^{\text{init}}$</th>
<th>$\bar{\alpha}^{\text{opt}}$</th>
<th>$\beta_3^{\text{opt}}$</th>
<th>$\beta_2^{\text{opt}}$</th>
<th>$\delta J_{\text{coup}}$ (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>4.66</td>
<td>N/A</td>
<td>0.1</td>
<td>0.67</td>
<td>N/A</td>
<td>N/A</td>
<td>10.7%</td>
</tr>
<tr>
<td>$\mathcal{P}_1$</td>
<td>2.5</td>
<td>N/A</td>
<td>0.1</td>
<td>0.1</td>
<td>0.5</td>
<td>N/A</td>
<td>3.6%</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>4.66</td>
<td>8.0</td>
<td>0.1</td>
<td>0.07</td>
<td>0.33</td>
<td>0.21</td>
<td>7.4%</td>
</tr>
<tr>
<td>$\mathcal{P}_2$</td>
<td>4.66</td>
<td>6.0</td>
<td>0.1</td>
<td>0.03</td>
<td>0.37</td>
<td>0.28</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

The candidate with lowest objective function value is selected. As a benchmark, the optimal choice of terminal state in $X_d$ is computed using exhaustive search and is marked with a brown asterisk in Figure 4.7. In this example, the optimal choice is not found since the search for a solution is limited to the closest neighbors within $X_d$. One possible extension is to extend the area and not only search for a solution among the most adjacent neighbors. However, such an extension comes at the cost of a higher computational burden since more candidate solutions need to be computed.

For the car-like systems, the results presented in Table 4.4 show that the objective function value is reduced by more than 10% for the truck and 3.6% for the small car when the value of $\bar{\alpha}$ is optimized, compared to using the user-defined value $\bar{\alpha} = 0.1$ combined with optimization of the terminal constraint obtained from Algorithm 4.1. For the truck and trailer system, a decrease of 7.4% is observed for the parameterization with the longer trailer. For the shorter trailer, the objective function value is not significantly improved, even though the resulting parameters differ significantly. However, it is important to notice that the comparison is done relative to maneuvers that are already optimized using Algorithm 4.1.
Improved motion primitive optimization

Figure 4.7: Illustration of the heuristic used to search for a feasible terminal state. In this example, a heading change maneuver from $x_t = (0,0,\arctan(1/2),0.37,0.24,0.08,0)$ to $x_t = (x^f_3, y^f_3, \pi/4,0.37,0.24,0.08,0)$ is calculated. The red path represents the optimal maneuver, computed from the coupled problem in (4.7). The dashed lines represent the introduced heuristic constraints, obtained from the coupled optimal maneuver. When these constraints are added, the optimal path given by the green line is obtained. From the coupled and constrained solutions, we search for a maneuver that is feasible with respect to the resolution in the state-space discretization (in this case $r = 1 \text{ m}$). The green and red regions represent the areas (obtained from exhaustive search) with feasible and infeasible terminal states within the resolution, respectively. Finally, the closest neighbors are evaluated, and the obtained solution is the one with lowest objective function value (orange). Here, the optimal choice of terminal position obtained from exhaustive search is indicated by the brown asterisk.
In this chapter, a lattice-based path planner and numerical optimal control is combined to compute locally optimal solutions to path planning problems. It is done by first introducing a bilevel optimization formulation, which provides a new tool to analyze and understand the relation between the original optimal path planning problem and the lattice-based approximation. It is shown how the solution from a lattice-based path planner can be improved by initializing a numerical optimal control solver that efficiently solves the problem to local optimality. The main idea behind the proposed approach is that the lattice-based path planner solves the combinatorial aspects (e.g. which side to pass an obstacle) while numerical optimal control is used to improve the continuous aspects of the solution keeping the combinatorial part fixed. Furthermore, this work goes beyond previous path planning initialization strategies since the initialization is not only feasible, but also optimized using the same objective function as in the improvement step. It is shown in several numerical examples that the proposed approach results in significantly reduced computation time, reliable convergence of the numerical optimal control solver and generally improved solutions compared to previously used initialization strategies or lattice-based planners alone.
5.1 Problem formulation

In this chapter, the optimal path planning problem is defined as follows:

\[
\begin{align*}
\text{minimize} \quad & J = \int_0^{S_g} \ell(x(s), u(s), q(s)) ds \\
\text{subject to} \quad & x(0) = x_{\text{init}}, \quad x(S_g) = x_{\text{term}}, \\
& x'(s) = f_q(x(s), u(s)), \quad s \in [0, S] \\
& x(s) \in X_{\text{free}} \cap X_{\text{valid}}, \quad s \in [0, S] \\
& q(s) \in Q, \quad u(s) \in U, \quad s \in [0, S].
\end{align*}
\] (5.1)

Here, \( s > 0 \) is defined as the distance traveled by the system, \( x \in \mathbb{R}^n \) is the state vector and \( u \in \mathbb{R}^m \) is the continuous control input for the system. The derivative of the state with respect to the distance traveled is defined as \( \frac{dx}{ds} = x'(s) \). \( X_{\text{free}} \) represents the obstacle-free region and \( X_{\text{valid}} \) and \( U \) the feasible sets of states and control signals, respectively. There is also a discrete input signal \( q(s) \in Q = \{1,2,\ldots,N\} \) which enables the selection between \( N \) modes of the system. For the problem (5.1) to be well-defined, we assume a finite number of system mode switches in a finite distance.

The system mode determines the vector field \( f_q \in \mathcal{F} \) that describes the current equation of motion [34]. The system mode can for example represent the direction of motion (which is the main use in this chapter). However, the results presented in Section 5.2-5.5 also hold for a set \( \mathcal{F} \) representing a more general switched dynamical system. One such example is morphing aerial vehicles [28]. The objective function \( J \) to be minimized in (5.1) is specified by the cost function \( \ell(x,u,q) \). This function can depend on the continuous variables as well as the system mode, where the latter enables the possibility of associating each system mode with a unique cost function.

Since the problem (5.1) contains the combinatorial aspects of selecting the system mode and the route to avoid obstacles, as well as continuous nonlinear system dynamics, finding a feasible and locally optimal solution is a difficult problem. Hence, approximate methods aiming at feasible, suboptimal solutions are commonly used. In Section 2.3 some alternatives are listed, where the lattice-based path planner provides one alternative.

### Lattice-based path planner

As described in Section 2.3.5, a lattice-based motion planner computes suboptimal solutions of motion planning problems online by using a set of precomputed motion primitives \( \mathcal{P} \) offline. A lattice-based motion planner can also be used to solve path planning problems such as the one in (5.1). To be able to account for the system mode in a motion primitive \( m \in \mathcal{P} \), it is in this chapter defined as:

\[
m = (x(s), u(s), q) \in X_{\text{valid}} \times U \times Q, \quad s \in [0, S].
\] (5.2)
Here, a motion primitive represents a feasible path of length $S$ in a fixed system mode $q \in Q$ that moves the system from an initial state $x(0) = x_{\text{start}} \in \mathcal{X}_d$ to a terminal state $x(S) = x_{\text{final}} \in \mathcal{X}_d$ (where $\mathcal{X}_d$ represents the state-space discretization), by applying the control inputs $u(\cdot)$. The motion primitives are computed by solving OCPs in the form of:

$$\begin{align*}
\text{minimize} & \quad \int_{s=0}^{S} \ell(x(s), u(s), q) ds \\
\text{subject to} & \quad x(0) = x_{\text{start}}, \quad x(S) = x_{\text{final}} \\
& \quad x'(s) = f_q(x(s), u(s)), \quad s \in [0, S] \\
& \quad x(s) \in \mathcal{X}_{\text{valid}}, \quad s \in [0, S] \\
& \quad u(s) \in \mathcal{U}, \quad s \in [0, S],
\end{align*}$$

(5.3)

for the user-defined selections of $x_{\text{start}}$, $x_{\text{final}}$ and $q$. After the motion primitive set is computed, the original path planning problem (5.1) can be approximated by the discrete OCP in (2.33), where the combinatorial aspect of selecting the system mode sequence $\{q_k\}_{k=1}^N$ is implicitly encoded in the selection of motion primitives. Finally, the objective function in (2.33) used to approximately solve (5.1) is given by:

$$L_u(m) = \int_{0}^{S} \ell(x(s), u(s), q) ds.$$  

(5.4)

## 5.2 Bilevel optimization preliminaries

In this chapter, the optimal path planning problem in (5.1) and the lattice-based approximation (2.33) will be related by rewriting the original problem formulation (5.1) into a bilevel optimization problem [21]. It will be shown that this new formulation of the problem allows for an insightful interpretation of the standard lattice solution methodology. In particular it will be used to connect the methodology to parametric optimization, highlight suboptimality properties, and discuss the choice of objective function used at different parts of the lattice-based framework.

A bilevel optimization problem is an optimization problem where a subset of the variables are constrained to be an optimal solution to another optimization problem called the lower-level problem. Analogously, the problem on the first level is called the upper-level problem. A general bilevel optimization problem can be written as [21]:

$$\begin{align*}
\text{minimize} & \quad F(x, y) \\
\text{subject to} & \quad (x, y) \in \Upsilon, \\
& \quad y \in \arg\min_z \{f(x, z) : (x, z) \in \Omega\},
\end{align*}$$

(5.5)
with upper-level optimization variable \( x \in \mathbb{R}^{n_x} \) and lower-level optimization variable \( y \in \mathbb{R}^{n_y} \). The functions \( F(x, y) \) and \( f(x, z) \) represent the upper and lower-level objective functions, respectively, and

\[
\Upsilon = \{(x, y) \mid G_i(x, y) \leq 0, \ i \in \mathbb{Z}_{1,C}\},
\]

\[
\Omega = \{(x, z) \mid g_i(x, z) \leq 0, \ i \in \mathbb{Z}_{1,D}\},
\]

represent the upper and lower-level feasible sets, which are represented by \( C \) and \( D \) inequality constraints, respectively. Typically, a subset of the optimization variables in the upper-level problem are considered as parameters to the lower-level problem. Seen from the upper-level problem, the optimality requirement of the lower-level problem is in general a non-convex constraint. Comparably simple examples of bilevel problems, e.g., where the problems on both levels are quadratic programming problems, can be solved by representing the solution to the lower level problem by, e.g., encoding the KKT conditions using mixed integer optimization [21] or explicitly representing the lower-level solution parametrically using a solution obtained from parametric programming [27]. It will be shown in this work that the lattice planner can be interpreted as a way of solving a bilevel formulation of (5.1) using the latter alternative, i.e., representing the lower-level solution explicitly as a (sampled) parametric solution.

### 5.3 Bilevel optimization problem reformulation

It will now be shown how the path planning problem in (5.1) can be reformulated as a bilevel optimization problem. Let \( L_u(m) \) from (5.4) represent the upper-level objective function and introduce lower-level cost function \( \ell_l(x, u, q) \). Assume that

\[
L_u(m) = \int_0^S \ell_l(x(s), u(s), q) \, ds.
\]  

(5.6)

After dividing the path planning problem in (5.1) in \( M \) path segments where along each one the system mode is kept constant, it is possible to cast it as an equivalent bilevel (dynamic) optimization problem in the form:

\[
\begin{align*}
\text{minimize} & \quad \left\{ x_k^s, x_k^f, q_k, m_k \right\}_{k=1}^M \\
\text{subject to} & \quad x_1^s = x_{\text{init}}, \quad x_M^f = x_{\text{term}}, \\
& \quad x_k^s = x_{k-1}^f, \quad k \in \mathbb{Z}_{2,M} \\
& \quad q_k \in \mathbb{Q}, \quad k \in \mathbb{Z}_{1,M} \\
& \quad m_k \in \text{arg}\min_{(x, u, q, S)} (5.8), \quad k \in \mathbb{Z}_{1,M},
\end{align*}
\]

(5.7)

where the initial state \( x_k^s \), terminal state \( x_k^f \) and system mode \( q_k \) for phase \( k \) are the upper-level optimization variables considered as parameters to the lower-level optimization problem, and \( m_k \) is defined in the same way as a motion primitive in (5.2).
The constraints given by \( x^s_k = x^f_{k-1} \) ensure that the path is continuous between adjacent path segments. Furthermore, the corresponding lower-level optimization problem in (5.7) can formally be specified as the following multi-parametric OCP (mp-OCP):

\[
J_*^l(x^s,x^f,q) = \min_{u(\cdot),x(\cdot),q,S} \int_0^S \ell_l(x(s),u(s),\bar{q}) ds 
\]

subject to

\[
\begin{align*}
&x(0) = x^s, \quad x(S) = x^f, \\
x'(s) = f(q(x(s),u(s))), \quad s \in [0,S] \\
x(s) \in X^\text{free} \cap X^\text{valid}, \quad s \in [0,S] \\
u(s) \in U, \quad s \in [0,S] \\
\bar{q} = q,
\end{align*}
\]

where \( x^s, x^f, \) and \( q \) are considered as parameters from the upper-level problem. Note the similarities between this problem and (5.1). Here, the main difference is that the system mode is fixed and the path length \( S \) is typically shorter than \( S_g \).

Above, it was assumed that the objective functions are related as in (5.6), which was necessary in order for the equivalence between (5.1) and (5.7) to hold. An alternative is to select the objective functions in the two levels more freely in a way that does not satisfy (5.6), with the price of breaking the equivalence between (5.1) and (5.7). If such a choice is still made, the solution to (5.7) with (5.8) will in general no longer be an optimal solution to (5.1). However, the use of different objective functions allows in practice for a division of the specification of the problem such as finding a minimum time solution by combining, e.g., low-lateral-acceleration solutions from the lower-level problem [51]. A bilevel interpretation of this is that the lower-level problem restricts the family of solutions the upper-level problem can use to compose an optimal solution.

### 5.4 Analysis of solution properties using bilevel arguments

From a practical point of view, the bilevel optimization problem consisting of (5.7) and (5.8) is in principle harder to solve than the standard formulation of the optimal control problem in (5.1). However, the formulation as a bilevel problem introduces possibilities to approximate the solution by sampling the solution to the lower-level mp-OCP as a function of its parameters. The result of this sampling is that the solution to (5.8) is only computed for \( K \) predefined parameter combinations \((x^s_i,x^f_i,q_i) \in \mathcal{A}, \ i \in \mathbb{Z}_{1,K}\), where \( \mathcal{A} \) is the user-defined set of combinations. These motion segments obtained by solving the mp-OCP for \( K \) parameter combinations together constitute the motion primitive set \( \mathcal{P} \) used in (2.33). An interpretation of this procedure is hence that \( \mathcal{P} \) used in a lattice planner is a coarsely sampled parametric solution to the mp-OCP in (5.8) which can be used to represent the optimal solution to the lower-level problem when the upper-level
problem is solved. The sampling introduces the well-known suboptimality of only being able to find solutions within the selected discretization [45]. However, this approximation makes it possible to solve the bilevel problem in real-time in the form of a lattice planner.

To be able to compute the motion primitives offline, the obstacle avoidance constraints in (5.8) are disregarded in the lower-level problem and are instead handled during online planning in the upper-level problem. After this rearrangement, the bilevel optimization problem in (5.7) is equivalent to the lattice formulation (2.33). In the following results, it is shown that this rearrangement of constraints makes it impossible to obtain an optimal solution within the selected discretization since the lower-level problems are not required to satisfy the obstacle avoidance constraints.

**Theorem 5.1.** Let $\mathcal{P}_1$ denote the bilevel optimization problem

$$\begin{align*}
\text{minimize} & \quad F(x, y) \\
\text{subject to} & \quad y \in \arg\min_z \{ F(x, z) : (x, z) \in \Omega \},
\end{align*}$$

with optimal solution $(x_1^*, y_1^*)$ and optimal objective function value $F(x_1^*, y_1^*)$. Furthermore, let $\mathcal{P}_2$ denote the bilevel optimization problem

$$\begin{align*}
\text{minimize} & \quad F(x, y) \\
\text{subject to} & \quad (x, y) \in \Omega, \\
& \quad y \in \arg\min_z \{ F(x, z) \},
\end{align*}$$

with optimal solution $(x_2^*, y_2^*)$ and optimal objective function value $F(x_2^*, y_2^*)$. It then holds that $F(x_1^*, y_1^*) \leq F(x_2^*, y_2^*)$.

**Proof:** The feasible set of $\mathcal{P}_1$ is $Z_1 = \{(x, y) \mid y \in \arg\min_z \{ F(x, z) : (x, z) \in \Omega \} \}$, and the feasible set of $\mathcal{P}_2$ is $Z_2 = \{(x, y) \mid (x, y) \in \Omega; y \in \arg\min_z \{ F(x, z) \} \}$. Hence, any point in $Z_2$ is also in $Z_1$, i.e., $Z_2 \subseteq Z_1 \implies F(x_1^*, y_1^*) \leq F(x_2^*, y_2^*)$.

**Corollary 5.1.** Assume that the optimal solution $(x_1^*, y_1^*)$ to $\mathcal{P}_1$ in (5.9) is unique. Then, if there exists an index $i$ such that $g_i(x_1^*, y_1^*)$ is strongly active (Definition 2.7) in the lower-level problem, it holds that $F(x_1^*, y_1^*) < F(x_2^*, y_2^*)$ where $(x_2^*, y_2^*)$ is the optimal solution to $\mathcal{P}_2$ in (5.10).

**Proof:** The feasible sets $Z_1$ of $\mathcal{P}_1$ and $Z_2$ of $\mathcal{P}_2$ are defined in the same way as in the proof of Theorem 5.1. Since there exists at least one constraint which is strongly active at the lower level, it follows that $(x_1^*, y_1^*) \notin Z_2$, since $(x_1^*, \arg\min_z \{ F(x_1^*, z) \}) \notin \Omega$. Hence, $(x_1^*, y_1^*) \in Z_1 \setminus Z_2$. Since $(x_1^*, y_1^*)$ is the unique optimal solution to $\mathcal{P}_1$ over $Z_1 \supseteq Z_2$, it follows that $\not\exists (x_2^*, y_2^*) \in Z_2 : F(x_2^*, y_2^*) \leq F(x_1^*, y_1^*)$. Hence, $F(x_1^*, y_1^*) < F(x_2^*, y_2^*)$. 

An interpretation of Corollary 5.1 is that if the optimal solution to (5.7) is “strongly” in contact with the environment, then it is not in general possible to obtain an optimal solution using solutions to the lower-level problem (i.e., motion
primitives) computed without considering obstacles. Note that these effects are beyond the fact that lower-level problems are sampled on a grid. The lower-level family of solutions is no longer optimal, instead the solutions need to adapt to the surrounding environment to become optimal, which is not explicitly a part of the standard lattice-planning framework.

It will now be shown that the consequences of the suboptimality aspects discussed in this section of the approximate solution to (5.1) obtained by solving (2.33) using a lattice-based path planner, can be efficiently reduced using numerical optimal control employing the solution from the lattice planner as a good warm-start.

5.5 Improvement using numerical optimal control

In this section, we propose to use numerical optimal control to improve the approximate solution computed by the lattice planner. By letting the system mode sequence \( \sigma = \{q_k\}_{k=1}^M \) be fixed to the solution from the lattice planner, the following OCP is obtained:

\[
\begin{align*}
\min_{\{x_k(\cdot), u_k(\cdot), S_k\}_{k=1}^M} & \quad J_{\text{imp}} = \sum_{k=1}^M \int_0^{S_k} \ell(x_k(s), u_k(s), \sigma[k]) \, ds \\
\text{subject to} & \quad x_1(0) = x_{\text{init}}, \quad x_M(S_M) = x_{\text{term}}, \\
& \quad x_k(0) = x_{k-1}(S_k), \quad k \in \mathbb{Z}_{2,M} \\
& \quad x'_k(s) = f_{\sigma[k]}(x_k(s), u_k(s)), \quad s \in [0, S_k], \quad k \in \mathbb{Z}_{1,M} \\
& \quad x_k(s) \in X_{\text{free}} \cap X_{\text{valid}}, \quad s \in [0, S_k], \quad k \in \mathbb{Z}_{1,M} \\
& \quad u_k(s) \in U, \quad s \in [0, S_k], \quad k \in \mathbb{Z}_{1,M} 
\end{align*}
\]

where the optimization variables are the control signals \( u_k(\cdot) \) and lengths \( S_k \) of the \( M \) phases. The difference compared to the optimal path planning problem (5.1) is that the combinatorial aspect of selecting the system mode sequence is already specified. However, since the lengths of the phases are optimization variables, it is possible that redundant phases introduced by the lattice planner are removed by selecting their lengths to zero. Furthermore, the second combinatorial aspect of selecting how to pass obstacles is implicitly encoded in the warm-start solution from the lattice planner.

The problem in (5.11) is in the form of a standard multiphase OCP, where the subsequent phases are connected using equality constraints. This problem can be solved using numerical optimal control according to what is described in Section 2.2. Today, there exist high-performing open-source NLP software such as IPOPT [83], WOHRP [17], etc., that can be used to solve these types of problems. Common for such NLP solvers is that they aim at minimizing both the constraint violation and the objective function value [59]. Hence, a good initialization strategy should consider both the objective function and feasibility. In this work, the resolution optimal solution \( \{m_k\}_{k=1}^M \) from the lattice planner is used to initialize
the NLP solver. It represents a path that is not only dynamically feasible, but also where each phase in the path, i.e., each motion primitive, has been computed by optimizing the same cost function $\ell(x, u, q)$ as in (5.11). Hence, the NLP solver is provided with a well-informed warm-start, which in general will decrease the time for the NLP solver to converge to a locally optimal solution [59]. Furthermore, when the same objective function is used both in (2.33) and (5.11), the NLP solver will in general be initialized close to a good local minimum. Finally, a benefit of using a feasible initialization is that it is always guaranteed that a dynamically feasible solution exists that is homotopic with the provided initialization (at the very least the initialization itself), making it reliable to use online. Due to all these properties, the step of solving (5.11) is referred to as an improvement step in this work, which is somewhat in contrast to previous work where this step is commonly denoted “smoothing”. Its primary aim is to improve the solution obtained from the lattice planner in terms of improving the objective function value.

\[ \text{Remark 5.1.} \quad \text{Note that the improvement step also can be used to enable path planning from and to arbitrary initial and terminal states that are not within the specified state-space discretization } \mathcal{X}_d. \text{ Here, the lattice planner can be used to find a path from and to the closest states in } \mathcal{X}_d, \text{ and the improvement step can then adapt the path such that it starts at the initial state and reaches the terminal state exactly. However, in this case the warm-start cannot be guaranteed to be feasible.} \]

After the improvement step is applied, a solution with lower objective function value compared to the solution from the lattice planner will in general be obtained since, relating back to Section 5.4, the discretization constraints in the bilevel formulation are removed, and the paths are constructed while explicitly considering obstacles.

### 5.5.1 Proposed path planning approach

A solution to the path planning problem is found using a preparation step offline and a two-step procedure online according to Algorithm 5.1. In the preparation step, the objective functions used in the motion primitive generation and graph search in the lattice planner and the improvement step are specified. Furthermore, the system modes with associated vehicle models (used in the motion primitive generation and improvement step) are defined. Then, the motion primitive set $\mathcal{P}$ is computed by solving the OCPs defined by the user without considering obstacles. For a detailed explanation of this step, the reader is referred to Section 2.3.5 and Chapter 4.

The first step online is called whenever a new path planning problem from $x_{\text{init}}$ to $x_{\text{term}}$ is to be solved. In this step, a lattice planner is used to solve the approximate path planning problem (2.33) by using the precomputed motion primitive set $\mathcal{P}$ and the current description of the available free space $\mathcal{X}_{\text{free}}$. The solution is a resolution optimal path, where the system mode is kept constant in each phase. This solution is used as a well-informed warm-start to the final improvement step, where the multiphase OCP in (5.11) is solved to local optimality by improving the continuous aspects of the solution from the lattice planner.
Algorithm 5.1: Proposed path planning approach

1: **Offline:**
2: **Input:** \( x(s), X_{\text{valid}}, u(s), U, q \in Q, f_q(x(s), u(s)) \in \mathcal{F} \) and \( L_u(m), \ell_1(x, u, q) \) and \( \ell(x, u, q) \).
3: Choose \( X_d \) and select \( (x_i^s, x_i^f, q_i), i = Z_{1:K} \)
4: \( \mathcal{P} \leftarrow \text{solve } K \text{ ocps in the form of (5.8) disregarding obstacle constraints.} \)
5: **Online:**
6: **Input:** \( x_{\text{init}}, x_{\text{term}}, X_{\text{free}}. \)
7: **Lattice planner:** : \( \{m_k, q_k\}_{k=1}^{M} \leftarrow \text{Solve (2.33) from } x_{\text{init}} \text{ to } x_{\text{term}} \text{ with } \mathcal{P}. \)
8: **Improvement step:** : \( \{x_k(\cdot), u_k(\cdot), S_k\}_{k=1}^{M} \leftarrow \text{Solve (5.11) with } \sigma = \{q_k\}_{k=1}^{M}, \)
   warm-started with \( \{m_k\}_{k=1}^{M}. \)

5.5.2 Variations of the proposed approach

The benefits of the introduced improvement step come at the cost of additional computation time. If the computational budget is not enough to cover the cost of improving the complete path from the initial state to the terminal state, there are variants of the approach presented in this work that can reduce the computational effort required.

For a faster improvement step, it is possible to fix some of the states along the solution from the lattice planner. This enables the possibility to perform computations in parallel since each phase in the problem is decoupled from the rest of the phases. Another variant is to start the improvement step \( M \) samples into the future of the solution from the lattice-based planner and commit to executing the first \( M - 1 \) steps without applying the improvement step. If \( M \) is chosen large enough, the improvement step will be provided with sufficient computations such that it will be possible to execute the improved part once the vehicle arrives at the selected state.

If the environment is expected to be constantly updated, e.g., if dynamic obstacles are present, the initial part of the solution from the lattice-based planner should be prioritized for improvement since it is likely that re-planning will be required. This can be achieved by only improving the initial part of the solution (at minimum longer than a motion primitive) in a receding horizon fashion. Finally, another variant is to use feasible sequential optimization methods, which means that a feasible solution is available for execution after every iteration in the improvement step. One such method is described in [82], which is based on a trust region feasibility-perturbed SQP approach. Since each iteration is feasible, the algorithm can run as long as the computational budget allows for, with an improved solution after each step.

5.6 Numerical results

In this section, the proposed path planning approach is applied to two different vehicular systems; a car and a truck and trailer system. During online planning,
the lattice planner is implemented using A* graph search, where a precomputed free-space HLUT is used as heuristic function to guide the search process. The HLUT is computed by solving path planning problems in an obstacle-free environment from all initial states \( x_{\text{init}} \in X_d \) with a position at the origin to all terminal states \( x_{\text{term}} \in X_d \) with a position within a square centered around the origin with side length \( \rho \) (in the experiments, \( \rho = 40 \) m for the car and \( \rho = 80 \) m for the truck and trailer system are used). The motion primitive generation and the improvement step are both implemented in Python using CasADi [3], where the warm-start friendly SQP method WORHP is used as NLP solver. All simulations are performed on a laptop computer with an Intel Core i7-5600U processor.

### 5.6.1 Vehicle models

The model of the car is based on the kinematic bicycle model in Section 4.4.1, with state vector \( x_c(s) = (\bar{x}_c(s), \alpha(s), \omega(s)) \), where \( \bar{x}_c(s) = (x_1(s), y_1(s), \theta_1(s)) \). Here, \( (x_1, y_1) \) is the center of the rear axle of the car, \( \theta_1 \) is the car’s orientation, \( \alpha \) is the front-wheel steering angle and \( \omega \) is the steering angle rate. The vehicle model is

\[
\bar{x}'_c(s) = q \left( \cos \theta_1(s), \sin \theta_1(s), \frac{\tan \alpha(s)}{L_1} \right)^T, \tag{5.12}
\]

\[\alpha'(s) = \omega(s), \quad \omega'(s) = u_\omega(s),\]

where \( u_\omega \) is the continuous control signal to the system which represents the steering angle acceleration, \( L_1 = 2.9 \) m the wheel-base of the car and \( q \in \{1, -1\} \) is the discrete decision variable representing the direction of motion. The constraints imposed on the states and control signal are given by \( |\alpha(s)| \leq \pi/4, |\omega(s)| \leq 0.5 \) and \( |u_\omega(s)| \leq 40 \). The cost function used for the car is given by:

\[
\ell_c(x_c, u_\omega, q) = 1 + \lambda (\alpha^2 + 10\omega^2 + u_\omega^2), \tag{5.13}
\]

where the variable \( \lambda \) represents the trade-off between path length and smoothness of the solution. The truck and trailer system is a general 2-trailer with a car-like truck given in Section 4.4.1. This system is composed of three interconnected vehicle segments; a car-like truck, a dolly and a semitrailer (see Figure 4.4). The state vector for this system is given by \( x_t(s) = (\bar{x}_t(s), \alpha(s), \omega(s)) \) where \( \bar{x}_t(s) = (x_3(s), y_3(s), \theta_3(s), \beta_3(s), \beta_2(s)) \). Here, \( (x_3, y_3) \) is the center of the axle of the semitrailer, \( \theta_3 \) is the orientation of the semitrailer, \( \beta_3 \) is joint angle between the semitrailer and the dolly, \( \beta_2 \) is joint angle between the dolly and the car-like truck. The truck’s steering angle \( \alpha \) and its derivatives, \( \omega \) and \( u_\omega \), are subject to the same constraints as in the car-case. The model of this system can compactly be represented as (see Section 4.4.1 or [51] for details):

\[
\bar{x}'_t(s) = q f_t(\bar{x}_t(s), \alpha(s)),
\]

\[\alpha'(s) = \omega(s), \quad \omega'(s) = u_\omega(s), \tag{5.14}\]

where \( q \in \{1, -1\} \) also in this case represents the direction of motion. The system parameters for the truck and trailer system used in this section are \( L_1 = 4.66, \)

$L_2 = 3.75$, $L_3 = 7.59$ and $M_1 = 1.668$, which coincide with the parameters used in [51]. Finally, the cost function used for this system is given by:

$$
\ell_t(x_t, u_\omega, q) = \begin{cases} 
1 + \lambda(\alpha^2 + 10\omega^2 + u_\omega^2), & q = 1, \\
1 + \lambda(\beta_3^2 + \beta_2^2 + \alpha^2 + 10\omega^2 + u_\omega^2), & q = -1,
\end{cases}
$$

(5.15)

i.e., quadratic penalties for large joint angles $\beta_3$ and $\beta_2$ are added to the cost function for paths in backward motion to avoid so-called jackknife configurations. Unless stated otherwise, $\lambda = 1$ is used in both (5.13) and (5.15).

As described in Section 4.4.1, it is possible to obtain the motion primitives for the truck and trailer system in backward motion by computing them in forward motion from the terminal state to the initial state. This is a useful result to obtain a more reliable convergence for the NLP solver used to compute the motion primitives, since the system is stable in forward motion and unstable in backward motion. In this section, this result is also utilized in the improvement step (5.11) where all path segments in the solution from the lattice-based path planner are computed in forward motion. To ensure that the path segments in backward motion are connected according to the solution from the lattice planner, the equality constraints that connect subsequent phases in (5.11) are modified accordingly.

### 5.6.2 State-lattice construction

To illustrate the full potential of the proposed approach, three different motion primitive sets for each vehicle are used by the lattice planner, where the sets use either simplified or complete vehicle models. The first motion primitive sets $P_{\text{dyn}}$ use the complete vehicle models. The second sets $P_{\text{kin}}$ disregard the steering dynamics and hence neglects the associated state $\omega$ and control signal $u_\omega$. Instead, the steering angle $\alpha$ is considered as the control signal (i.e., purely kinematic models), which is similar to the initialization strategy used in [87]. The third sets $P_{\text{geo}}$ are computed by completely neglecting the system dynamics, further referred to as a geometric model, where instead linear interpolation is used between the initial and terminal states for each motion primitive.

Before computing the motion primitive sets, the state spaces of the vehicles need to be discretized. The positions, $(x_1, y_1)$ for the car and $(x_3, y_3)$ for the semitrailer, are discretized onto a uniform grid with resolution $r = 1$ m and the orientations $\theta_1 \in \Theta$ and $\theta_3 \in \Theta$ are irregularly discretized as proposed in [68]. The discretization of the steering angle $\alpha$ is only applicable for the complete models. For simplicity, it is here constrained to zero and its rate $\omega$ is also constrained to zero to ensure that $\alpha$ is continuously differentiable, even when motion segments are combined online [51]. For the truck and trailer system, the joint angle $\beta_3$ and $\beta_2$ are also constrained to zero at each discretized state in the state lattice. Note however that on the path between two discretized states, the systems can take any feasible vehicle configuration.

The motion primitive sets are automatically computed using the approach described in Chapter 4, where the sets are composed of heading changes and parallel maneuvers according to Table 5.1. These maneuvers are optimized using the cost
5 Integrating lattice-based planning and numerical optimal control

Table 5.1: A description of the different motion primitive sets used. $|\Theta|$ defines the number of heading discretization points, $\Delta_{\theta}^{\text{max}}$ defines which of the neighboring headings to connect (from $\pm 1$ to $\pm \Delta_{\theta}^{\text{max}}$), and $n_{\text{par}}$ defines the number of parallel maneuvers (per heading). Finally, $n_{\text{prim}}$ defines the resulting total number of motion primitives.

| $\mathcal{P}$ | $|\Theta|$ | $\Delta_{\theta}^{\text{max}}$ | $n_{\text{par}}$ | $n_{\text{prim}}$ |
|--------------|-----------|-----------------|-------------|-------------|
| $\mathcal{P}_{\text{geo}}$ | 16 | 2 | N/A | 224 |
| $\mathcal{P}_{\text{kin}}$ | 16 | 4 | 3 | 480 |
| $\mathcal{P}_{\text{dyn}}$ | 16 | 4 | 3 | 480 |

functions defined in (5.13) and (5.15). For the simplified vehicle models, the neglected states are disregarded in the cost functions. A more detailed description of the state-lattice construction can be found in Section 2.3.5 and Chapter 4.

5.6.3 Experimental results

For the car model, two different path planning scenarios are considered: a parallel parking problem (Figure 5.1) and one with multiple routes to avoid obstacles (Figure 5.2). For the truck and trailer system, a loading site area is used (Figure 5.3). The obstacles and vehicles are implemented using bounding circles [45]; the area of the car is described by three circles, while the truck is described by one circle and the trailer by two circles. This choice of obstacle representation can be used in all steps since the constraints can be described by smooth functions. An alternative object representation that is perfectly compatible with the approach presented in this work is proposed in [86], where vehicles and obstacles can be represented by general convex sets.

The path planning problems are first solved by the lattice planner, using the three different motion primitive sets described in Table 5.1. Thereafter, the obtained solutions are used to initialize the improvement step. For the simplified models, all states that are not represented are initialized to zero.

For the car scenarios in Figure 5.1-5.2, the results in Table 5.2-5.3 show that the lattice planner achieves the lowest computation times if the geometric model is used, compared to the kinematic and the complete model. However, using this simple initialization strategy results in a decreased reliability (only 62 % and 54.5 % success rate) and the total average computation time becomes higher than the two other cases due to a more computationally demanding improvement step. The kinematic initialization performs better than the geometric in terms of reliability, but in a cluttered environment (Table 5.3) the success rate is only 75.3 %. When the complete model is used in the lattice planner, the computation time for the improvement step is significantly reduced compared to when the simpler initialization strategies are used. In particular, the total computation time including the lattice planner is as much as halved and the success rate is always 100 %. Furthermore, the mean objective function value $\bar{J}_\text{opt}$ decreases significantly compared to the solution from the lattice planner $\bar{J}_\text{lat}$. For the two simpler initialization
5.6 Numerical results

Table 5.2: Results from parallel parking scenario (Figure 5.1, 150 problems).

The motion primitive set used in the lattice planner. \( \bar{t}_{\text{lat}} \) is the average time for the lattice planner to find a solution. \( r_{\text{imp}} \) and \( t_{\text{imp}} \) are the success rate and average time for the improvement step to converge. \( t_{\text{tot}} \) is the average total time. Finally, \( \bar{J}_{\text{lat}} \) and \( \bar{J}_{\text{imp}} \) is the average objective function value for the solutions from the lattice planner and improvement step, respectively.

<table>
<thead>
<tr>
<th>( \mathcal{P} )</th>
<th>( t_{\text{lat}} ) [s]</th>
<th>( t_{\text{imp}} ) [s]</th>
<th>( t_{\text{tot}} ) [s]</th>
<th>( r_{\text{imp}} )</th>
<th>( \bar{J}_{\text{lat}} )</th>
<th>( \bar{J}_{\text{imp}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{P}_{\text{geo}} )</td>
<td>0.0011</td>
<td>1.12</td>
<td>1.12</td>
<td>62 %</td>
<td>N/A</td>
<td>30.8</td>
</tr>
<tr>
<td>( \mathcal{P}_{\text{kin}} )</td>
<td>0.025</td>
<td>1.03</td>
<td>1.06</td>
<td>90.7 %</td>
<td>N/A</td>
<td>28.7</td>
</tr>
<tr>
<td>( \mathcal{P}_{\text{dyn}} )</td>
<td>0.014</td>
<td>0.88</td>
<td>0.894</td>
<td>100 %</td>
<td>35.7</td>
<td>27.5</td>
</tr>
</tbody>
</table>

Table 5.3: Results from multiple routes scenario (Figure 5.2, 77 problems).

See Table 5.2 for a description of the variables.

<table>
<thead>
<tr>
<th>( \mathcal{P} )</th>
<th>( t_{\text{lat}} ) [s]</th>
<th>( t_{\text{imp}} ) [s]</th>
<th>( t_{\text{tot}} ) [s]</th>
<th>( r_{\text{imp}} )</th>
<th>( \bar{J}_{\text{lat}} )</th>
<th>( \bar{J}_{\text{imp}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{P}_{\text{geo}} )</td>
<td>0.03</td>
<td>4.82</td>
<td>4.85</td>
<td>54.5 %</td>
<td>N/A</td>
<td>60.5</td>
</tr>
<tr>
<td>( \mathcal{P}_{\text{kin}} )</td>
<td>0.36</td>
<td>3.81</td>
<td>4.17</td>
<td>75.3 %</td>
<td>N/A</td>
<td>50.0</td>
</tr>
<tr>
<td>( \mathcal{P}_{\text{dyn}} )</td>
<td>0.31</td>
<td>2.14</td>
<td>2.45</td>
<td>100 %</td>
<td>59.9</td>
<td>50.2</td>
</tr>
</tbody>
</table>

strategies, no comparable objective function values from the lattice planner exist since the solutions are infeasible with respect to the actual path planning problem.

The results for the truck and trailer system (Figure 5.3) are summarized in Table 5.4. In this experiment, using a dynamically feasible initialization (as proposed in this work) has an even larger impact on the time spent in the improvement step; the average time using the geometric and kinematic models has dropped from 18.4 s and 15 s, respectively, down to 9.5 s for the dynamic model. The reason why such a large computational performance gain is obtained in this more advanced scenario is mainly due to the complicated system dynamics, which also affect the reliability using a geometric initialization strategy where the success rate is less than 50 \%

Finally, the reliability for the kinematic initialization is higher compared to the car scenarios. This is mainly due to a less cluttered environment, which enables a higher success rate for the kinematic initialization strategy.

In Table 5.5 the impact of using the complete vehicle model and the same or different objective functions at the three steps: the motion primitive generation, the graph search in the lattice planner and the improvement step is analyzed. The results on row 1 \((\lambda_u = \lambda_l = \lambda_i = 10)\) represent the baseline where the same objective function is used in all steps. When shortest path is used as objective function in the graph search \((\lambda_u = 0, \text{row 2 in Table 5.5})\), the average cost for a path is increased by roughly 10 \%, due to that the improvement step converges to a worse local minimum. However, the total computation time decreases, as a result of a faster graph search. This is mainly due to that Euclidean distance is used as heuristic function outside the range of the HLUT, which is a better estimator of cost to go when solely shortest path is used as objective function. The computation
Figure 5.1: Parallel parking scenario solved from several initial states with \( \theta_1^i = \{0, \pi/4\} \) (indicated by area within the dotted lines). Solutions from one problem are illustrated for the three initialization strategies (using the motion primitive sets described in Table 5.1) with corresponding solutions from the improvement step.

Figure 5.2: A problem with multiple routes solved from several initial states with \( \theta_1^i = 0 \) to get to the terminal state \((x_1^f, y_1^f, \theta_1^f, \alpha_f) = (13, 3, \pi, 0)\). See Figure 5.1 for a further description of the content.
Table 5.4: Results from loading site scenario (Figure 5.3, 270 problems). See Table 5.2 for a description of the variables.

<table>
<thead>
<tr>
<th>$\mathcal{P}$</th>
<th>$\bar{t}_{lat}$ [s]</th>
<th>$\bar{t}_{s}$ [s]</th>
<th>$\bar{t}_{tot}$ [s]</th>
<th>$r_{imp}$</th>
<th>$J_{lat}$</th>
<th>$J_{imp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_{geo}$</td>
<td>0.69</td>
<td>18.4</td>
<td>19.1</td>
<td>43.7 %</td>
<td>N/A</td>
<td>236</td>
</tr>
<tr>
<td>$\mathcal{P}_{kin}$</td>
<td>6.5</td>
<td>15.0</td>
<td>21.5</td>
<td>98.9 %</td>
<td>N/A</td>
<td>164</td>
</tr>
<tr>
<td>$\mathcal{P}_{dyn}$</td>
<td>5.35</td>
<td>9.45</td>
<td>14.8</td>
<td>100 %</td>
<td>184</td>
<td>164</td>
</tr>
</tbody>
</table>

Table 5.5: Results from loading site scenario (Figure 5.3, 270 problems). $\lambda_u$, $\lambda_l$ and $\lambda_i$ are the values of $\lambda$ in (5.15) used in the graph search, the motion primitive generation and improvement step, respectively. See Table 5.2 for a description of the other variables.

<table>
<thead>
<tr>
<th>$\mathcal{P}$</th>
<th>$\lambda_u$</th>
<th>$\lambda_l$</th>
<th>$\lambda_i$</th>
<th>$\bar{t}_{lat}$ [s]</th>
<th>$\bar{t}_{imp}$ [s]</th>
<th>$\bar{t}_{tot}$ [s]</th>
<th>$r_{imp}$</th>
<th>$J_{imp}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_{dyn}$</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>3.2</td>
<td>6.2</td>
<td>9.4</td>
<td>100 %</td>
<td>190</td>
</tr>
<tr>
<td>$\mathcal{P}_{dyn}$</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>2.2</td>
<td>5.9</td>
<td>8.1</td>
<td>100 %</td>
<td>208</td>
</tr>
<tr>
<td>$\mathcal{P}_{dyn}$</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>1.9</td>
<td>12.6</td>
<td>14.5</td>
<td>100 %</td>
<td>212</td>
</tr>
</tbody>
</table>

time for the improvement step is similar to using $\lambda_u = 10$, which is reasonable since each phase (i.e., motion primitive) in the warm-start is optimized using the same objective function as in the improvement step. When also the motion primitives are generated using shortest path as objective function ($\lambda_u = \lambda_l = 0$, row 3 in Table 5.5), not only the average solution cost increases, but also the convergence time for the improvement step. The reason is that each phase in the initialization is far from a local minimum in terms of the objective function used in the improvement step. This clearly illustrates the importance of using the same objective function in the motion primitive generation and improvement step for fast convergence in the latter step.
Figure 5.3: Loading site scenario using the truck and trailer system, solved from and to several initial and terminal states. See Figure 5.1 for a further description of the content.
In this thesis, optimal motion planning for autonomous systems is studied, where the focus is to improve numerical optimal control algorithms such that they can reliably be applied on motion planning problems in challenging unstructured environments. Here, the main contributions are concluded and possible future research directions are presented.

6.1 Conclusions

The main contribution in this thesis is to efficiently compute improved solutions to optimal motion planning problems. This is mainly achieved by combining recent ideas from automatic control, numerical optimization and robotics, where the strengths of the developed methods within each field are utilized. Motion planning algorithms from the robotics community are good at solving combinatorial aspects that are present in the problem formulation, but suffer from the curse of dimensionality when the search space to explore grows. Numerical optimal control algorithms on the other hand, scales better in higher dimensions. However, they are in danger of getting stuck in a bad local minimum when naively applied to nonconvex problems, such as general motion planning problems.

In Chapter 3, the problem of initializing a numerical optimal control solver for motion planning was addressed. It was shown that by combining the homotopy method and the SQP method, a method that has promising properties to solve motion planning problems involving environments with a challenging geometry of the feasible set is obtained. A proof-of-concept implementation of the algorithm was evaluated on a number of motion planning problems. The results showed that the proposed method significantly extends the applicability of state-of-the-art numerical optimal control methods by solving problems that could not be solved at all by these methods before when initialized using standard strategies.
Furthermore, it was also shown that not only a numerical optimal control method was outperformed, but also a state-of-the-art sampling-based motion planner based on random sampling.

The work presented in Chapter 4 proposed an optimization-based and automatic motion primitive generation framework for motion planning using a lattice-based planner. Based on user-defined principle motion types, the suggested framework computes the motion primitive set for any user-selected system instance in a parameterized family of systems. The computations are performed by simultaneously optimizing the motion and the selection of the states to connect in the state-space discretization. It was shown that this new framework enables the use of the same maneuver definitions for all instances in a parameterized, fairly diverse, family of systems, which enables fast generation of the motion primitive set for any desired instance of the system. In numerical experiments the new method was shown to clearly outperform existing related methods, both in terms of performance as well as generality. The capabilities of the proposed framework both have applications at the level of industrial production of, e.g., several similar but different vehicles as well as online as a response to changes in the system to control. Furthermore, the proposed framework was shown to increase the overall quality of the solutions generated in the online planning phase in several numerical examples. Finally, the proposed framework was extended to allow for simultaneous optimization of both the motion primitives and discretization parameters that are used in the lattice-based planner to define the discrete search space. This extension enables automatic selection of optimized discretization parameters for a specific system instance. The extension was shown to improve the quality of the computed motion primitives in terms of reduced objective function value, compared to motion primitives only optimized with respect to terminal state constraints using a pre-defined state-space discretization.

Chapter 5 presented a unified optimization-based path planning approach to efficiently compute high-quality locally optimal solutions to path planning problems. The approach was motivated by first showing that a lattice-based path planner can be cast and analyzed as a bilevel optimization problem. This information was then used to motivate a novel, tight combination of a lattice-based path planner and numerical optimal control. The first step of the proposed approach consists of using a lattice-based path planner, where the motion primitive set is optimized using the framework presented in Chapter 4, to find a resolution-optimal solution to the path planning problem using a discretized search space. This solution is then used as a well-informed warm-start in a second improvement step where numerical optimal control is used to compute a locally optimal solution to the path planning problem. In order to tightly couple the two steps, the lattice-based path planner uses a vehicle model and objective function that are chosen to coincide with those used in the second improvement step. This combination of a path planner and numerical optimal control makes, in a structured way, benefit of the former method’s ability to solve combinatorial parts of the problem and the latter method’s ability to obtain locally optimal solutions not restricted to a discretized search space. The value of this tight combination was thoroughly investigated with successful results in several practically relevant path planning problems. It was shown to outperform commonly used initialization strategies in terms of computation time, numerical reliability, and objective function value.
6.2 Future work

The results presented in this thesis can be extended in several ways. One possible extension of the results in Chapter 3 is to integrate and implement the proposed algorithm in a state-of-the-art SQP solver suited for OCPs with long prediction horizons. Another important future work is to develop an obstacle detection and classification framework, and to automatically select suitable homotopy maps for the obstacles (i.e., how the obstacles are “introduced”). A final extension to these results is to investigate if it is possible to integrate the homotopy map, representing the obstacle constraints, within the homotopy-based functionality that is already implemented for the barrier parameter in IP-based optimization solvers.

Future work for the results presented in Chapter 4 includes to further develop the extension that also optimizes the discretization parameters. Some possible alternatives are to improve the heuristics used to search for a feasible terminal state, and to optimize a larger amount of discretization parameters. Another extension is to evaluate the extended framework on motion planning problems where real-world usage statistics are available. Finally, an additional extension is to compute motion primitives that are also optimized with respect to expected potential obstacles in the environment.

For the results in Chapter 5, future work includes to further decrease the online planning time by, e.g., applying the improvement step in a receding horizon fashion or to only optimize parts of the solution obtained from the lattice-based path planner. Another extension is to apply the approach to systems with more distinct system modes, such as morphable drones \[28\]. Finally, one interesting research direction is to investigate how the proposed path planning framework can be extended, in a systematic way, for computing trajectories instead of paths.
6 Conclusions and future work


Licentiate Theses
Division of Automatic Control
Linköping University


