Linköping Studies in Science and Technology
Licentiate Thesis No. 1981

On Optimal Integrated Task and Motion Planning with Applications to Tractor-Trailers

Anja Hellander

LINKÖPING UNIVERSITY

Linköping studies in science and technology. Licentiate Thesis
No. 1981

On Optimal Integrated Task and Motion Planning with Applications to Tractor-Trailers

Anja Hellander

LINKÖPING
UNIVERSITY

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. 1981

On Optimal Integrated Task and Motion Planning with Applications to Tractor-Trailers

Anja Hellander

anja.hellander@liu.se
www.control.isy.liu.se
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping
Sweden

ISBN 978-91-8075-463-7 (print)
ISBN 978-91-8075-464-4 (PDF)
ISSN 0280-7971

Printed by LiU-Tryck, Linköping, Sweden 2023

To my family and friends!

Abstract

An important aspect in autonomous systems is the ability of a system to plan before acting. This includes both high-level task planning to determine what sequence of actions to take in order for the system to reach a goal state, as well as low-level motion planning to detail how to perform the actions required.

While it is sometimes possible to plan hierarchically, i.e., to first compute a task plan and then compute motion plans for each action in the task plan, there are also many problem instances where this approach fails to find a feasible plan as not all task plans lead to motion-planning problems that have feasible solutions. For this reason, it is desirable to solve the two problems jointly rather than sequentially. Additionally, it is often desirable to find plans that optimize a performance measure, such as the energy used, the length of the path travelled by the system or the time required. This thesis focuses on the problem of finding joint task and motion plans that optimize a performance measure.

The first contribution is a method for solving a joint task and motion planning problem, that can be formulated as a traveling salesman problem with dynamic obstacles and motion constraints, to resolution optimality. The proposed method uses a planner comprising two nested graph-search planners. Several different heuristics are considered and evaluated.

The second contribution is a method for solving a joint task and motion planning problem, in the form of a rearrangement problem for a tractor-trailer system, to resolution optimality. The proposed method combines a task planner with motion planners, all based on heuristically guided graph search, and uses branch-and-bound techniques in order to improve the efficiency of the search algorithm.

The final contribution is a method for improving task and motion plans for rearrangement problems using optimal control. The proposed method takes inspiration from finite-horizon optimal control and decomposes the optimization problem into several smaller optimization problems rather than solving one larger optimization problem. Compared to solving the original larger optimization problem, it is demonstrated that this can lead to reduced computation time without any significant decrease in solution quality.

Populärvetenskaplig sammanfattning

Ett stort forskningsområde under de senaste årtiondena är utvecklingen av autonoma system, system som på egen hand utan mänsklig påverkan kan lösa och genomföra olika uppdrag. Två viktiga delproblem som behöver lösas för att kunna uppnå det är uppgiftsplanering (eng. *task planning*) och rörelseplanering (eng. *motion planning*). Såväl uppgiftsplanering som rörelseplanering handlar om att beräkna hur ett system ska ta sig från sitt nuvarande tillstånd till ett måltillstånd, men på olika abstraktionsnivåer. Uppgiftsplanering görs på en högre abstraktionsnivå och kan sägas lösa problemet med *vad* som ska göras, medan rörelseplanering görs på en lägre nivå och kan sägas lösa problemet med *hur* det ska göras.

Ett exempel kan vara en robotarm som har i uppgift att stapla ett antal klossar på varandra. Uppgiftsplanering används då för att bestämma i vilken ordning klossarna ska lyftas upp och staplas på varandra, medan rörelseplanering anger hur robotarmen ska flyttas för att greppa en kloss eller för att flytta en kloss från en position till en annan.

Många problem har aspekter av såväl uppgiftsplanering som rörelseplanering. Ett sätt att lösa sådana problem är att först lösa uppgiftsplaneringsproblemet och därefter lösa rörelseplaneringsproblem. Det är dock inte säkert att det resulterar i en lösning till det ursprungliga problemet, eftersom systemet kan ha rörelsebegränsningar som inte fångas av uppgiftsplaneringen. Det är därför önskvärt att integrera uppgifts- och rörelseplanering tätare genom att ta hänsyn till rörelsebegränsningarna i rörelseplaneringsproblemet redan när uppgiftsplaneringen görs så att de båda delproblemen kan lösas samtidigt i stället för i sekvens.

I denna avhandling är målet inte enbart att beräkna en kombinerad uppgifts- och rörelseplan som tar hänsyn till systemens begränsningar, utan även att optimera ett prestandamått. Exempel på sådan optimering kan vara att minimera energiförbrukning, förflyttad sträcka eller tid.

Det första bidraget är en metod för att lösa en typ av uppgifts- och rörelseplanering som uppkommer vid planering av borrning i gruvor. Den föreslagna metoden använder sig av grafsökning och resulterar i lösningar som är optimala med avseende på ett prestandamått, givet en diskretisering av problemet.

Det andra bidraget är en metod för att gemensamt lösa en typ av uppgifts- och rörelseplaneringsproblem för en dragbil som ska omarrangera ett antal släp. Den presenterade metoden ger lösningar som givet en diskretisering av problemet är optimala med avseende på ett prestandamått.

Det sista bidraget är en metod för att med hjälp av optimal styrning förbättra en given lösning med avseende på ett prestandamått. I stället för att lösa ett större optimeringsproblem så presenteras här en lösning till problemet där en serie av mindre optimeringsproblem löses, vilket kan leda till att tiden det tar att lösa problemet kraftigt reduceras samtidigt som kvaliteten på de funna lösningarna bibehålls.

Acknowledgments

First of all, I would like to thank my supervisor Assoc. Prof. Daniel Axehill for your constant support and encouragement. Thank you for our technical discussions and your research ideas, and for always staying up as long as it takes to finish an article before the deadline. I promise to finish before midnight at least once.

I would also like to thank my co-supervisors Assoc. Prof. Martin Enqvist and Dr. Kristoffer Bergman. Martin, thank you for maintaining a welcoming work environment in your role as head of division. Kristoffer, thank you for your support with technical insight and for having laid the foundation that my research is built on.

This thesis has been proofread by Hans Gunnarsson, Dr. Gustaf Hendeby, Dr. Magnus Malmström, Joel Nilsson, and Lic. Shamisa Shoja. I really appreciate the time you have spent and the suggestions you have made for improvement. An extra thanks to Gustaf for technical support with the LaTeXtemplate used to write the thesis.

I would like to thank all my current and former colleagues at the division of Automatic Control for contributing to a friendly atmosphere. Thank you for the fun we have had at conferences, pub nights at Villevalla, interesting discussions about Dante in the book circle, and fierce Advent of Code competitions. I would also like to thank all current and former colleagues at Guidance, Navigation and Control at Saab Dynamics for the friendly atmosphere. In particular, I want to thank my manager Lic. Torbjörn Crona for your constant belief in me and for providing me with this fantastic opportunity. I would also like to thank Dr. Henrik Jonson for agreeing to be my industrial supervisor even though you are technically retired.

This work has been supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP), funded by the Knut and Alice Wallenberg Foundation. I am grateful not only for the funding, but also for the opportunities to network with other WASP researchers.

Finally, I would like to thank my family. Mum, thank you for always being there for me. Hans, thank you for being you and for putting up with me even when I am at my most stressed. I love you.

Linköping, December 2023
Anja Hellander

Contents

Notation

Some sets

Notation	Meaning
$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers

Abbreviations

Abbreviation	Meaning
HLUT	Heuristic look-up table
MILP	Mixed-integer linear programming
MINLP	Mixed-integer nonlinear programming
NLP	Nonlinear programming
OCP	Optimal control problem
PDDL	Planning Domain Definition Language
PRM	Probabilistic roadmap
RRT	Rapidly-exploring random tree
TAMP	Task and motion planning

Part I

Background

1

Introduction

This chapter introduces the area of task and motion planning, provides a summary of the contributions, and presents the outline of the thesis.

1.1 Background and motivation

During the last decades, much research has been performed with the goal of developing systems that are autonomous, i.e., able to operate without human intervention. This requires several capabilities such as sensing and perception, planning and reasoning, as well as control. Each such capability leads to different research problems that must be solved in order to achieve the goal of autonomous systems.

For an autonomous system to be able to carry out a task defined in the form of an abstract goal, it is necessary that the system is capable of planning what to do before it acts. This planning needs to be done at several different levels of abstraction. At a high level of abstraction, task planning is needed to compute a sequence of actions that when executed will take the system from its current state to a desired state. At a lower level, motion planning is needed to detail exactly how the actions should be executed. As an example, a manufacturing robot might be tasked with manufacturing a product from separate parts. Task planning can then be used to plan in what order to add different parts, and motion planning can be used to compute the trajectory that the arm of the robot must take in order to successfully pick up or move a part or another object.

The areas of task planning and motion planning have been studied separately for a long time and share some similarities while also having differences. At their core, both task and motion planning are concerned with finding a sequence of valid states or configurations of a system that will move the system from its current state to a given terminal state, together with the actions or control inputs that will cause the system to move between these states [20, 43]. This requires a description of the system and the world it acts in, as well as how the state of the system changes as a result of the actions taken.

The largest difference between task planning and motion planning is perhaps that task planning usually considers a discrete world whereas motion planning considers a continuous world [43]. This difference affects the methods that can be used for the different problems. Task planning is often solved using graph search, guided by some heuristic function, and planners are often deterministic, i.e., they return the same result for the same problem every time. While there are some special cases of motion planning for non-holonomic systems that can be solved analytically without discretization, such as the Dubins car [14] or the Reeds-Shepp car [56], many motion-planning approaches are based on discretizing the problem in order to reduce it to a problem that can be solved by graph search. This discretization can be done in a deterministic way, or by random sampling of the configuration space.

As many problems contain aspects of both task planning and motion planning, the interest in integrated task and motion planning has increased in the last decade [8, 13, 17, 18, 31, 58]. This is motivated by the fact that a hierarchical solution where a task plan is computed first, and a motion plan is then computed for each action in the task plan, is not guaranteed to result in a feasible solution even if one exists to the overall problem. The main focus has been on finding feasible solutions, which is a difficult problem in itself.

Recently, there has been an increased interest in optimal task and motion planning [16, 40, 60, 63], where the objective is to find a solution that is not only feasible but that optimizes some performance measure as well. This is the focus of this thesis.

1.2 Research questions

This thesis aims to investigate how methods for task planning and motion planning that are based on graph search can be integrated, and how optimal control can be utilized in order to compute joint task and motion plans that are not only feasible but also optimized. In particular, the thesis aims to answer the following questions:

- How can optimal task and motion planning problems, especially for non-holonomic systems, be solved efficiently?
- How can methods from optimal control be used to improved the quality of joint task and motion plans?

1.3 Publications and contributions

The content of this licentiate thesis is based on the following published and unpublished work.

Paper A: On a Traveling Salesman Problem with Dynamic Obstacles and Integrated Motion Planning

> Anja Hellander and Daniel Axehill. On a traveling salesman problem with dynamic obstacles and integrated motion planning. In *2022 American Control Conference (ACC)*, pages 4965–4972, Atlanta, June 2022. IEEE.

Paper A investigates a task and motion planning problem where the task planning part consists of determining in which order to visit a set of determined positions, and where the chosen order affects the constraints for the motion planning subproblems. Paper A proposes a planner consisting of two nested graph-search planners. Several different heuristics for the planner are proposed and investigated.

Paper B: On Integrated Optimal Task and Motion Planning for a Tractor-Trailer Rearrangement Problem

> Anja Hellander, Kristoffer Bergman, and Daniel Axehill. On integrated optimal task and motion planning for a tractor-trailer rearrangement problem. In *62nd IEEE Conference on Decision and Control (CDC)*, Singapore, December 2023. IEEE.

Paper B proposes a combined task and motion planner for a rearrangement problem for a tractor and a set of trailers. The proposed planner combines a task planner and a motion planner that are both based on heuristically guided graph search. The planner is shown to be resolution complete and resolution optimal, i.e., given the discretization used by the planner it is complete and optimal. The proposed planner further uses the heuristic functions in order to maintain upper and lower bounds that are used in order to prune the search, which is shown to increase the efficiency of the algorithm without sacrificing neither resolution completeness nor resolution optimality.

Paper C: Improved Task and Motion Planning for Rearrangement Problems using Optimal Control

> Anja Hellander, Kristoffer Bergman, and Daniel Axehill. Improved task and motion planning for rearrangement problems using optimal control. *To be submitted*.

Paper C proposes a method for improving task and motion plans for rearrangement problems by formulating and solving optimal control problems. Building on ideas from finite-horizon optimal control and block coordinate descent, Paper C proposes a method where instead of solving the original optimization problem, several smaller optimization problems are posed and solved. It is shown that, compared to solving the original problem, this can lead to reduced computation time while resulting in solutions of similar quality.

In all the contributions listed in this section, the author of this thesis has performed the main part of the research, including theoretical derivations, numerical experiments, evaluations, and writing. The co-authors have contributed with research ideas, technical discussions and by improving the manuscripts.

1.4 Thesis outline

The thesis is divided into two parts. The first part presents the relevant background and consists of Chapters 1–6. Chapter 1 gives a background to the research problem and presents the contributions of the thesis. Chapter 2 gives an introduction to graph-search techniques. Chapter 3 provides a theoretical background to task planning. Chapter 4 presents background on motion planning, particularly approaches that are based on graph search. Chapter 5 gives an introduction to the field of joint task and motion planning. Chapter 6 concludes the first part of the thesis and presents some ideas for future work. The second part of the thesis contains edited versions of the publications listed under Section 1.3.

2

Graph search

This chapter gives a background on graph search and the problem of finding the shortest path between two nodes in a graph. The main focus is on heuristically guided search and A* in particular.

2.1 Preliminaries

Before defining the shortest path problem, some definitions are required:

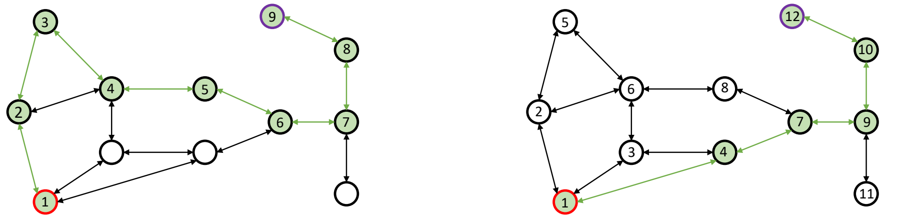
Definition 2.1. A *directed graph* $\mathcal{G}$ is a pair $(\mathcal{V}, \mathcal{E})$ consisting of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. An edge is an ordered pair of distinct vertices, i.e., $\mathcal{E} \subset \{(v_1, v_2) | v_1, v_2 \in \mathcal{V} \text{ and } v_1 \neq v_2\}$.

Definition 2.2. On a directed graph $(\mathcal{V}, \mathcal{E})$, $v_1 \in \mathcal{V}$ is a *predecessor* of $v_2 \in \mathcal{V}$ and v_2 is a *successor* of v_1 if $(v_1, v_2) \in \mathcal{E}$.

Definition 2.3. A *weighted directed graph* is a directed graph $(\mathcal{V}, \mathcal{E})$ together with a weight function $w : \mathcal{E} \mapsto \mathbb{R}$.

Definition 2.4. A *path* of length $N - 1$ is a sequence $P = (v_1, v_2, \ldots, v_N)$ of n vertices such that $(v_i, v_{i+1}) \in \mathcal{E}$ for $i = 1, 2, \ldots N - 1$.

Given a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with weight function w, the shortest path problem from $v_{\text{start}} \in \mathcal{V}$ to $v_{\text{goal}} \in \mathcal{V}$ can be formulated as the optimization problem



(a) *Depth-first search.* *(b)* *Breadth-first search.*

Figure 2.1: *Examples of depth-first and breadth-first search. The initial node is shown with a red border, and the final node with a purple border. Nodes are numbered according to the order in which they are examined, and the resulting path is shown in green.*

$$
\begin{aligned}
\underset{N,\{v\}_{i=1}^{N}}{\text{minimize}} \quad & \sum_{i=1}^{N-1} w((v_i, v_{i+1})) \\
\text{subject to} \quad & v_1 = v_{\text{start}} \\
& v_N = v_{\text{goal}} \\
& (v_i, v_{i+1}) \in \mathcal{E} \quad i = 1, \ldots, N-1.
\end{aligned}
\tag{2.1}
$$

A general graph-search algorithm maintains a list of nodes (vertices) to explore, called the frontier or the open list. In each iteration, a node is chosen from the frontier for exploration and its unexplored successors are added to the frontier. During the search, the algorithm maintains the previous node for each node. The search continues until the goal node has been reached and the resulting path is then extracted by starting at the goal node and moving to the previous node until the initial node is reached.

Different graph-search algorithms differ mainly in how the next node to explore is chosen from the frontier. Depth-first and breadth-first search sort the nodes in the frontier based on the order in which they were added to the frontier. Depth-first search chooses the node that was added last for expansion, i.e., the frontier is a stack, whereas breadth-first search chooses the node that was added first, i.e., the frontier is a queue. An illustration of depth-first and breadth-first search is shown in Figure 2.1. In the special case where all edges have the same (positive) weight, breadth-first search finds the shortest path, otherwise neither breadth-first nor depth-first search is guaranteed to find the shortest path. To find the shortest path, it is therefore necessary to continue the search even after a solution has been found in order to enumerate all possible solutions and keep track of the shortest solution that has currently been found.

To find the shortest path on a graph with positive weights, Dijkstra's algorithm [11] can be used. Pseudocode for the algorithm is shown in Algorithm 1.

Algorithm 1 Dijkstra's algorithm

$g(n_{\text{start}}) = 0$
Q.insert(n_{start}, $g(n_{\text{start}})$)
while not Q.empty() **do**
  $n = Q$.pop()
  if $n = n_{\text{goal}}$ **then**
    return ExtractPath(n_{goal})
  end if
  for $n' \in \text{succ}(n)$ **do**
    if no previous(n') or $g(n') > g(n) + w((n, n'))$ **then**
      previous(n') $= n$
      $g(n') = g(n) + w((n, n'))$
      Q.insert(n', $g(n')$)
    end if
  end for
end while
return FAILURE

For each node n, Dijkstra's algorithm keeps track of the cost-to-come $g(n)$, i.e., the cost of a path from the initial node. This cost is updated during the search whenever a new path with a lower cost is found. The frontier, denoted Q in Algorithm 1, is sorted based on cost-to-come, with the node with the lowest cost-to-come chosen for exploration in every iteration. For this reason, the function for inserting a node n to the frontier has an additional argument for the priority $g(n)$ of the node. Unlike breadth-first or depth-first where the first path that is found and returned is in general not the shortest, Dijkstra's algorithm is guaranteed to find a shortest path. Dijkstra's algorithm can also be run without any particular goal node. In that case the shortest path from the initial node to any other node in the graph will be computed.

2.2 Heuristically guided search

Heuristically guided search algorithms use a heuristically evaluated function to sort the nodes in the frontier. Greedy best-first search uses a heuristic function that estimates the cost of a path from the current node to the goal node in order to sort the frontier. One of the most common heuristically guided search algorithm is A*.

2.2.1 A*

A*, first described in [23], is one of the most used graph-search algorithms due to its efficiency. Pseudocode for the algorithm is shown in Algorithm 2. The frontier is sorted based on $f(n) = g(n) + h(n)$ where $g(n)$ is the cost-to-come and $h(n)$ is a non-negative heuristic function that estimates the cost-to-go, i.e., the cost of a

Algorithm 2 The A* algorithm

$g(n_{\text{start}}) = 0$
$f(n_{\text{start}}) = g(n_{\text{start}}) + h(n_{\text{start}})$
Q.insert(n_{start}, $f(n_{\text{start}})$)
while not Q.empty() **do**
    $n = Q$.pop()
    if $n = n_{\text{goal}}$ **then**
        return ExtractPath(n_{goal})
    end if
    for $n' \in \text{succ}(n)$ **do**
        if no previous(n') or $g(n') > g(n) + w((n, n'))$ **then**
            previous(n') $= n$
            $g(n') = g(n) + w((n, n'))$
            $f(n') = g(n') + h(n')$
            Q.insert(n', $f(n')$)
        end if
    end for
end while
return FAILURE

shortest path from n to the goal. The heuristic function should be chosen so as to be admissible and consistent, both of which are defined below.

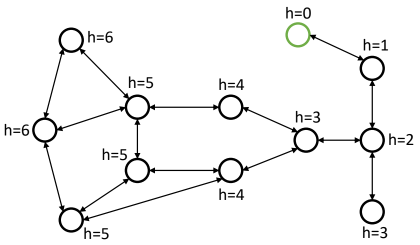
Definition 2.5. A heuristic function $h(n)$ is *admissible* if $h(n) \leq h^*(n)$ for all n, where $h^*(n)$ is the true cost-to-go.

Definition 2.6. A heuristic function $h(n)$ is *consistent* if for all nodes n and all successors m of n it holds that $h(n) \leq h(m) + w((n, m))$.

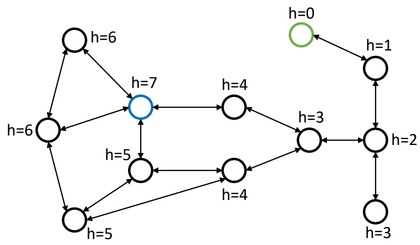
If h is admissible, the path returned by A* has optimal cost. If h is consistent, A* explores each node no more than once, and the f-values of the explored nodes are monotonically non-decreasing. Consistency implies admissibility, however, the reverse does not hold as it is possible to construct heuristic functions that are admissible but not consistent. Examples of some heuristic functions with varying properties are shown in Figure 2.2.

It can be noted that with the choice $h(n) = 0$ (which is trivially admissible and consistent) A* reduces to Dijkstra's algorithm. If a perfect heuristic, i.e., $h(n) = h^*(n)$ is used, then A* expands only the nodes along an optimal path.

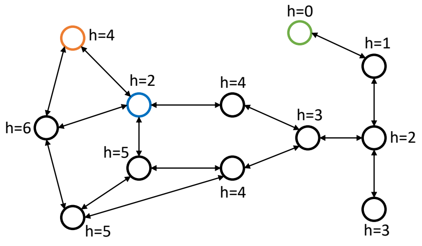
It is also possible to inflate the heuristic function with a factor $\epsilon > 1$, i.e., to sort the frontier based on $f(n) = g(n) + \epsilon h(n)$. This sacrifices optimality as the resulting heuristic is no longer admissible but may speed up the search. The level of suboptimality of the resulting solution is upper bounded by the factor ϵ, i.e., the resulting cost is at most ϵ times the optimal cost [52]. This has given rise to anytime A* methods, that repeatedly run A* with an inflated heuristic, gradually decreasing the inflation. This quickly finds a (suboptimal) solution, and then gradually improves the solution. The idea is that the algorithm should



(a) *Admissible and consistent heuristic. (The perfect heuristic as $h(n) = h^*(n)$ for all nodes n.)*



(b) *Neither admissible nor consistent heuristic. The heuristic value of the blue node is higher than the true cost-to-go.*



(c) *Admissible but not consistent heuristic. For all nodes n it holds that $h(n) \leq h^*(n)$, but the heuristic value of the orange node is higher than the sum of the heuristic value of the blue node and the cost of the edge from the orange to the blue node.*

Figure 2.2: *Some examples of heuristics with varying properties. The goal node is shown in green, and all edges have weight 1.*

be able to return a feasible solution even if it is aborted prematurely. Anytime repairing A* (ARA*) [46] is one such algorithm where results from previous runs with higher ϵ are reused rather than restarting the search from scratch for each inflation constant.

2.2.2 LPA*

Lifelong Planning A* (LPA*) [39] is an extension of A* that can be used to repeatedly solve shortest path problems between the same pair of nodes when the underlying graph changes between subsequent calls to the algorithm. It maintains two estimates of the cost-to-come, $g(n)$ and $\text{rhs}(n)$. The estimate $g(n)$ is a direct equivalent of the cost-to-come from the A* algorithm and remains unchanged between searches. The second estimate, $\text{rhs}(n)$ is based on the g-values of the predecessors of a node, $\text{rhs}(n) = \min_{m \in \text{predecessors}(n)} g(m) + w((m,n))$. A node n is said to be locally consistent if $g(n) = \text{rhs}(n)$, otherwise locally inconsistent. If $g(n) > \text{rhs}(n)$ it is said to be overconsistent, and if $g(n) < \text{rhs}(n)$ it is underconsistent. The frontier consists of the nodes that are locally inconsistent, sorted based on the (2-dimensional) key function $k(n) = [\min(g(n), \text{rhs}(n)) + h(n), \min(g(n), \text{rhs}(n))]$.

Pseudocode for the algorithm is shown in Algorithm 3, Algorithm 4, and Algorithm 5. The frontier is denoted with Q, and it is assumed to have methods $\text{insert}(node, key)$ for inserting a node with a given key, $\text{remove}(node)$ for removing a node from the frontier, and $\text{pop}(node)$ for removing and returning the node with the lowest key in the frontier. The backbone of the algorithm is the subroutine ComputeShortestPath, which is called to compute the new shortest path when there have been changes in the edge costs. The subroutine repeatedly selects the node in the frontier with the lowest key value for examination, until the goal node is locally consistent and no node in the frontier has a lower key value than the goal node. A shortest path can then be traced back from the goal node by for each node n moving to the predecessor n' that minimizes $g(n') + w((n', n))$.

Whenever a node n is chosen for expansion, the two estimates for the cost-to-come are compared. If $g(n) > \text{rhs}(n)$, then n is made to be locally consistent by setting $g(n) = \text{rhs}(n)$ and the rhs and key values of its successors are updated. If instead $g(n) < \text{rhs}(n)$, then $g(n)$ is updated as $g(n) = \infty$ and the rhs and key values of its successors as well as n itself are updated. If $\text{rhs}(n) = \infty$ as well, the node has become locally consistent. Otherwise, it is still locally inconsistent and is added to the frontier again, this time with a higher key value.

Under the condition that the heuristic used is admissible and consistent, LPA* is guaranteed to find a shortest path. Each node n will then be visited at most twice, at most once when it is underconsistent and at most once when it is overconsistent [39].

In the worst-case scenario where there are large changes to the graph, LPA* is not more efficient than A* search from scratch and may even be less efficient [39]. However, in cases where changes to the graph are small, LPA* can increase the efficiency compared to running an A* search from scratch.

Algorithm 3 The main loop of the LPA* algorithm

```
procedure MAIN
    Initialize()
    while true do
        ComputeShortestPath()
        Wait for changes in edge costs
        for all edges (n, n′) with changed cost do
            Update w((n, n′))
            UpdateNode(n′)
        end for
    end while
end procedure
```

Algorithm 4 The ComputeShortestPath procedure

```
procedure COMPUTESHORTESTPATH
    while Q.TopKey()< CalculateKey(n_goal) or rhs(n_goal) ≠ g(n_goal) do
        n = Q.pop()
        if g(n) > rhs(n) then
            g(n) = rhs(n)
            for n′ ∈ succ(n) do
                UpdateNode(n′)
            end for
        else
            g(n) = ∞
            for n′ ∈ succ(n) ∪ {n} do
                UpdateNode(n′)
            end for
        end if
    end while
end procedure
```

2.3 Branch and bound

Branch and bound (B&B) is a method, or a family of related methods, for solving optimization problems [42, 45]. It is often used in particular for optimization problems with discrete or combinatorial aspects. The algorithm finds the optimal solution by searching in a tree where each node n corresponds to a set of candidate solutions X_n. The root node corresponds to the entire set of feasible solutions, and for each node n and set of successor nodes $\text{succ}(n)$ it holds that $\cup_{m \in \text{succ}(n)} X_m = X_n$ and $X_m \cap X_{m'} = \emptyset$ for all $m, m' \in \text{succ}(n)$ such that $m \neq m'$.

A general version of a B&B algorithm is shown in Algorithm 6. An upper bound $\bar{J}$ on the optimal value and the corresponding feasible solution $\bar{x}$ for which the upper bound is obtained is maintained by the algorithm. For each node n that is visited, a relaxed optimization problem is solved with optimal objective func-

Algorithm 5 Procedures used by the LPA* algorithm

procedure CALCULATEKEY(n)
    return $[\min(g(n), \mathrm{rhs}(n)) + h(n); \min(g(n), \mathrm{rhs}(n))]$
end procedure
procedure INITIALIZE
    $Q = \emptyset$
    for $n \in V$ **do**
        $\mathrm{rhs}(n) = g(n) = \infty$
    end for
    $\mathrm{rhs}(n_{\mathrm{start}}) = 0$
    Q.insert(n_{start}, CalculateKey(n_{start}))
end procedure
procedure UPDATENODE(n)
    if $n \neq n_{\mathrm{start}}$ **then**
        $\mathrm{rhs}(n) = \min_{n' \in \mathrm{pred}(n)}(g(n') + w((n', n)))$
    end if
    if $n \in Q$ **then**
        Q.remove(n)
    end if
    if $g(n) \neq \mathrm{rhs}(n)$ **then**
        Q.insert(n, CalculateKey(n))
    end if
end procedure

Algorithm 6 A general B&B algorithm

$\bar{J} = \infty$
$\bar{x} = \emptyset$
Q.insert(n_0)
while not Q.empty() **do**
    $n = Q$.pop()
    $\underline{J}, \underline{x}$ = solveRelaxation(n)
    if feasibleCandidate($\underline{x}$) and $\underline{J} < \bar{J}$ **then**
        $\bar{J} = \underline{J}$
        $\bar{x} = \underline{x}$
    end if
    if feasibleCandidate($\underline{x}$) or $\underline{J} = \infty$ or $\underline{J} \geq \bar{J}$ **then**
        continue
    end if
    for $n' \in \mathrm{succ}(n)$ **do**
        Q.insert(n')
    end for
end while
return $\bar{J}, \bar{x}$

tion value $\underline{J}$ together with the solution $\underline{x}$ for which the optimal relaxed solution is obtained. The solution $\underline{x}$ is not necessarily feasible for the original optimization problem, but the obtained value $\underline{J}$ gives a lower bound on the optimal value for the set of candidate solutions that the node n corresponds to. If $\underline{x}$ is a feasible solution to the original problem, there is no need to examine the successors of the node, and the upper bound and corresponding solution can be updated if it is better than the best previously known solution.

The upper and lower bounds can be used to prune the search space. If $\underline{J} \geq \bar{J}$, the node can be pruned (cut), i.e., the node is not further explored, and its successors are not generated. If a node cannot be pruned, its successors are generated by dividing the corresponding solution set into disjoint sets and adding their corresponding nodes to the frontier. An example of how this can be used to solve an integer programming optimization problem is shown in Example 2.7.

Example 2.7: Branch and bound

Consider the optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & -3x_1 - 2x_2 \\ \text{subject to} \quad & x_1 + 2x_2 \leq 7 \\ & 4x_1 + 2x_2 \leq 15 \\ & x_1, x_2 \in \mathbb{N}. \end{aligned} \tag{2.2}$$

The resulting search tree when applying the B&B algorithm to this problem is shown in Figure 2.3. In each node, the constraints $x_1, x_2 \in \mathbb{N}$ are relaxed and the resulting optimization problem is solved. The relaxed problems for n_2 and n_4 lack feasible solutions, and the nodes can therefore be pruned. The solution to the relaxed problem for n_5 is a feasible solution to (2.2), so the global upper bound $\bar{J}$ and best solution $\bar{x}$ can be updated and the node pruned. For n_6, the relaxed solution is such that $\underline{J} \geq \bar{J}$ and the node can be pruned. As there are no nodes left to examine, the algorithm then terminates and returns $\bar{J} = -11, \bar{x} = [3, 1]$.

There are many similarities between B&B and graph-search algorithms such as A*. In [49], a generalization of B&B is presented that encompasses both B&B and A* as special cases. Ideas from B&B can be used in other graph search methods such as depth-first search as well by using a heuristic function to give a lower bound on the optimal value instead of solving a relaxation of an optimization problem. This can speed up the search compared to having to do an explicit enumeration of possible solutions.

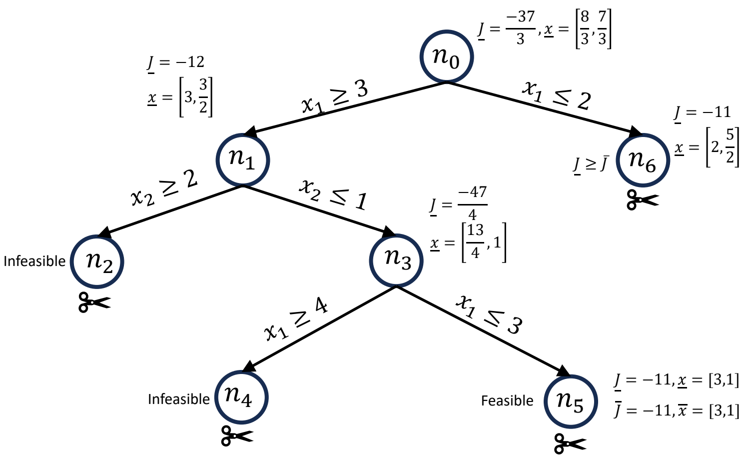


Figure 2.3: *The resulting search graph from Example 2.7. Nodes are numbered in the order they are examined.*

3

Task planning

This chapter introduces the topic of task planning. The classical representation of a task-planning problem is presented, as well as the optimal task-planning problem which can be solved using graph-search techniques. Different domain-independent heuristic functions that can be used to guide a graph search are presented.

3.1 Preliminaries

Task planning, as defined in this thesis, is the problem of finding a feasible plan in the form of a sequence of actions that transform a discrete system from an initial state to a goal state. For an introduction to task planning, see [20, 21].

Task planning problems are commonly modelled using state transition systems, which are defined as follows [20]:

Definition 3.1. A *state transition system* is a tuple $\Sigma = (S, A, E, \gamma)$ where:

- S is a finite (or recursively enumerable) set of states,
- A is a finite (or recursively enumerable) set of actions,
- E is a finite (or recursively enumerable) set of events,
- $\gamma : S \times A \times E \mapsto 2^S$ is a state transition function, where 2^S denotes the power set of S.

Some common assumptions that are often used in classical task planning [20], and that will be used in this thesis as well are:

- S is finite.

- Σ is fully observable, i.e., complete knowledge about the state of the system is assumed.
- Σ is deterministic. For each state $s \in S$ and each action-event pair $(a, e) \in A \times E$, $\gamma(s, a, e)$ is either the empty set (if the action is not applicable in the state) or it is a singleton set (if the action is applicable).
- Σ is static, i.e., $E = \emptyset$. The system will remain in the same state until an action is applied as there are no internal dynamics.
- The goal used as input to the planner is a set of goal states S_g with one or more elements.
- A solution plan to a planning problem is a linearly ordered finite sequence of actions $a_0, \ldots, a_k$.
- Actions and events have no duration and all state transitions happen instantaneously.
- Planning is done offline, and any changes in Σ that may occur during the planning are not considered by the planner.

Since Σ is assumed to be deterministic and static, it is possible to simplify the notation for the state transition function and write $\gamma(s, a) = s'$ rather than $\gamma(s, a) = \{s'\}$.

A task planning problem can be defined as a tuple $(\Sigma, s_{\text{init}}, S_g)$, where Σ is a state transition system, $s_{\text{init}} \in S$ is the initial state, and $S_g \subseteq S$ is a subset of goal states. A solution plan is an action sequence $a_0, \ldots, a_{N-1}$ that gives rise to a sequence of states $s_0, \ldots, s_N$ such that $s_0 = s_{\text{init}}$, $s_N \in S_g$, and $\gamma(s_k, a_k) = s_{k+1}$ for all $k = 0, 1, \ldots, N-1$.

3.2 Representation of states and actions

Task planning problems often use logic to represent the state space and the action space. The most common logic representation is a so-called STRIPS-like representation [20], that uses a simplified version of first-order logic. This is done by using a first-order language $\mathcal{L}$ consisting of finitely many predicate symbols and finitely many constant symbols. The constant symbols, called instances in [43], represent the objects that exist in the world and are of relevance to the planning. These objects could be, e.g., robots, cars, cranes, blocks, or locations. A predicate is a binary-valued function used to indicate properties or relations between objects. Predicates can be applied to one or more terms, i.e., variables or constants, or to none. An example could be at(robot, place), where the predicate at is applied to the variables *robot* and *place*, and can be used to indicate that the robot *robot* is at the location *place*. Another example could be a predicate snowing that does not require any terms, and can be used to indicate if it is snowing or not. To separate between variables and constants, this thesis will use names in italics such as *robot* and *block* to refer to variables, and names without italics such as robot1 and blockA to refer to constants. Some useful definitions are provided below:

Definition 3.2. An *atom* is a predicate applied to the correct number of terms. An atom that contains no variables, i.e., only constants, is said to be a *ground atom* or a *fact*.

Definition 3.3. A *literal* is an atom (positive literal) or the negation of an atom (negative literal).

As an example, at(*robot*, *place*) is an atom, and at(robot1, place2) is a ground atom. Both are examples of positive literals, and their negations ¬at(*robot*, *place*) and ¬at(robot1, place2) are negative literals.

A state is represented as a set of ground atoms of $\mathcal{L}$. Often the closed-world assumption is used, so that only positive literals are included and any atom that is not explicitly included is assumed to not hold in that state. A set of goal states S_g can therefore be represented by a set of ground atoms g as $S_g = \{s \in S | g \subseteq s\}$. This representation will be used later in this chapter, and the task planning problem is then represented as $(\Sigma, s_{\text{init}}, g)$, where g represents such a set of states. An example of predicates and states for a world where a robot moves blocks around is shown in Example 3.4.

Example 3.4: States

Consider a world, with three blocks (blockA, blockB and blockC), a table, and a robot manipulator that can pick up and place the blocks on top of each other or on the table. To describe the state, the following predicates can be defined:

- on(*obj1*, *obj2*), which indicates that the object *obj1* is on top of the object *obj2*,
- clear(*block*), which indicates that there is no object on top of the block *block*,
- holding(*block*), which indicates that the robot is holding the block *block*,
- empty_hand, which indicates that the robot is currently not holding any block.

An illustration of possible initial and goal states is shown in Figure 3.1 together with the predicates that describe them.

Actions are specified through the use of *operators*. In addition to its name, an operator is specified by the following three components:

- The variables it operates on. One example could be an operator move(*robot*, *place1*, *place2*) that operates on the three variables *robot*, *place1*, and *place2*.
- The preconditions pre(o) of the operator, i.e., a set of literals that must hold in order for the operator to apply. For the move operator this could

be at(*robot*, *place1*). The preconditions can be divided into $\text{pre}^+(o)$ and $\text{pre}^-(o)$, the positive and negative preconditions, respectively. Positive preconditions are positive literals, and negative preconditions are negative literals.

- The effects of the operator $\text{eff}(o)$, that describe the changes to the state when the operator is applied. The effects can be divided into $\text{eff}^+(o)$ and $\text{eff}^-(o)$, the positive and negative effects, respectively.

An action is a ground instance of an operator. An action a is applicable in a state s if $\text{pre}^+(a) \subseteq s$ and $\text{pre}^-(a) \cap s = \emptyset$, and the result when the action is applied is $\gamma(s, a) = (s - \text{eff}^-(a)) \cup \text{eff}^+(a)$. In Example 3.5, examples of operators for the world in Example 3.4 is shown.

Example 3.5: Operators

Consider the same world as in Example 3.4. To move the blocks around, four operators pickup(*block*), putdown(*block*), unstack(*block1*, *block2*) and stack(*block1*, *block2*) can be defined according to:

pickup(*block*) - pick up *block* from the table
Precondition: clear(*block*), on(*block*, table), empty_hand
Effect: ¬on(*block*, table), ¬clear(*block*), ¬empty_hand, holding(*block*)

putdown(*block*) - put down *block* on the table
Precondition: holding(*block*)
Effect: ¬holding(*block*), on(*block*, table), clear(*block*), empty_hand

unstack(*block1*, *block2*) - pick up *block1* from *block2*
Precondition: clear(*block1*), on(*block1*, *block2*), empty_hand
Effect: ¬on(*block1*, *block2*), ¬clear(*block1*), ¬empty_hand, holding(*block1*), clear(*block2*)

stack(*block1*, *block2*) - put down *block1* on *block2*
Precondition: holding(*block*), clear(*block2*)
Effect: ¬holding(*block1*), on(*block1*, *block2*), clear(*block1*), ¬clear(*block2*), empty_hand

A plan that solves the planning problem with initial and goal states as in Figure 3.1 is π = unstack(blockB, blockA), stack(blockA, blockC), pickup(blockA), stack(blockA, blockB).

3.3 Optimal task planning

Consider a state transition system $\Sigma = (S, A, \gamma)$ with a cost function $c : S \times A \mapsto [0, \infty)$ that assigns a non-negative cost to each instance of applying an action in a

(a) *Example of an initial state: {on(blockA, table), on(blockB, blockA), on(blockC, table), clear(blockB), clear(blockC), empty_hand}.*

(b) *Example of a goal state: {on(blockA, blockB), on(blockB, blockC), on(blockC, table), clear(blockA), empty_hand}.*

Figure 3.1: *Examples of initial and goal state for the world in Example 3.4.*

state. Given an initial state $s_{\text{init}} \in S$ and a subset of goal states $S_g \subseteq S$, the optimal task planning problem can be formulated as the following optimization problem:

$$\begin{aligned} \underset{\{a_k\}_{k=0}^{N-1}, N}{\text{minimize}} \quad & \sum_{k=0}^{N-1} c(s_k, a_k) \\ \text{subject to} \quad & s_0 = s_{\text{init}}, \quad s_N \in S_g \\ & s_{k+1} = \gamma(s_k, a_k), \quad k = 0, \ldots, N-1 \\ & s_k \in S, \quad k = 0, \ldots, N \\ & a_k \in A, \quad k = 0, \ldots, N-1. \end{aligned} \tag{3.1}$$

The problem in (3.1) can be solved by posing the problem as a graph-search problem on a graph where each vertex corresponds to a state, and edges correspond to actions. The problem can then be solved using, e.g., the graph-search techniques described in Chapter 2. Domain-independent optimal planners often use A* with an admissible heuristic.

3.4 Domain-independent heuristics

An important part for a search problem is the heuristic that is used to guide the search. While good heuristics can be dependent on the domain, i.e., the task planning problem, there has also been a lot of work on domain-independent heuristics that work well on many different task planning problems. Such heuristics are often based on solving a relaxed planning problem where some constraints are relaxed. This could be constraints that restrict, e.g., what a state, action or plan is, what actions are applicable, or what effects are produced when applying an action [21]. Doing so will result in a relaxed planning problem, such that any solution to the original problem is a solution to the relaxed planning problem as well. This guarantees that the cost of an optimal solution to the relaxed planning problem is a lower bound on the cost of an optimal solution to the original planning problem.

3.4.1 The max-cost and the additive cost heuristics

One early example of a domain-independent heuristic is the max-cost heuristic $h^{max}(s)$, which is the cost of an optimal solution to a relaxed planning problem where a goal in the form of a set of literals (could be either a goal state or the preconditions of an action) is achieved as long as one of its literals (the one that is most expensive to achieve) is achieved. For a planning problem $(\Sigma, s_{\text{init}}, g)$, where g is a set of literals, the max-cost heuristic is defined as [5, 21]

$$\begin{aligned} h^{max}(s) &= \Delta^{max}(s, g) = \max_{g_k \in g} \Delta^{max}(s, g_k) \\ \Delta^{max}(s, g_k) &= \begin{cases} 0 & \text{if } g_k \in s \\ \min\{\Delta^{max}(s, a) | a \in A \text{ and } g_k \in \text{eff}(a)\} & \text{otherwise} \end{cases} \\ \Delta^{max}(s, a) &= c(a) + \Delta^{max}(s, \text{pre}(a)) \end{aligned} \tag{3.2}$$

where $\text{pre}(a)$ denotes the preconditions and $\text{eff}(a)$ the effects of an action a. This is an admissible heuristic, but in practice, better results have been achieved using the related additive cost heuristic $h^{add}(s)$, which is inadmissible [21]. This heuristic is defined as

$$\begin{aligned} h^{add}(s) &= \Delta^{add}(s, s_g) = \sum_{g_k \in s_g} \Delta^{add}(s, g_k) \\ \Delta^{add}(s, g_k) &= \begin{cases} 0 & \text{if } g_k \in s \\ \min\{\Delta^{add}(s, a) | a \in A \text{ and } g_k \in \text{eff}(a)\} & \text{otherwise} \end{cases} \\ \Delta^{add}(s, a) &= c(a) + \Delta^{add}(s, \text{pre}(a)). \end{aligned} \tag{3.3}$$

3.4.2 Delete-relaxation heuristics

Another way to relax a planning problem is through delete-relaxation. In a delete-relaxation, actions can never remove atoms from a state, only add new ones. Under the assumption that preconditions and goals can only be positive, which is common in classical planning, this corresponds to removing the negative effects of an action so that actions only have positive effects. An admissible heuristic could be $h^+(s)$, where $h^+(s)$ denotes the cost of an optimal solution plan to the delete-relaxed problem. While finding a feasible solution to the delete-relaxed problem is easier than finding a feasible solution to the original problem, it is still NP-hard to find an optimal solution. Instead, the Fast-Forward planner [28, 29] uses the inadmissible Fast-Forward heuristic $h^{FF}(s)$ which computes an approximation to the optimal relaxed solution. The Fast-Forward heuristic is described

Algorithm 7 The Fast-Forward heuristic $h^{FF}(s)$

1: $\hat{s}_0 = s; A_0 = \emptyset$

2: $k = 0$

3: **while** $g \not\subseteq \hat{s}_k$ **do**

4: $\quad k = k + 1$

5: $\quad A_k = \{a|\text{pre}(a) \subseteq \hat{s}_{k-1}\}$

6: $\quad \hat{s}_k = \hat{s}_{k-1} \cup \bigcup_{a\in A_k} \text{eff}^+(a)$

7: $\quad$ **if** $\hat{s}_k = \hat{s}_{k+1}$ **then**

8: $\qquad$ **return** ∞

9: $\quad$ **end if**

10: **end while**

11: $\hat{g}_k = g$

12: $h = 0$

13: **while** $k > 0$ **do**

14: $\quad$ choose a minimal set $\hat{a}_k \subseteq A_k$ such that $\hat{g}_k \subseteq \hat{s}_{k-1} \cup \bigcup_{a\in\hat{a}_k} \text{eff}^+(a)$

15: $\quad \hat{g}_{k-1} = \{g_i \in \hat{g}_k | g_i \notin \bigcup_{a\in\hat{a}_k} \text{eff}^+(a)\} \cup \bigcup_{a\in\hat{a}_k} pre(a)$

16: $\quad h = h + \sum_{a\in\hat{a}_k} \text{cost}(a)$

17: **end while**

18: **return** h

in Algorithm 7. Starting from the current state s it constructs a sequence of relaxed states $\hat{s}_k$ and sets of actions A_k, with $\hat{s}_0 = s$ and $A_0 = \emptyset$. The next set of actions A_{k+1} is constructed as all (relaxed) actions that are applicable in the relaxed state $\hat{s}_k$ (line 5), and the next relaxed state $\hat{s}_{k+1}$ is then constructed as the union of $\hat{s}_k$ and the (positive) effects of the actions in A_{k+1} (line 6). Once a relaxed state has been reached that contains the set of goal literals g, a relaxed plan is extracted in the form a subset of each set of actions (lines 14–15). The total cost of these actions is taken as the heuristic value. As there is no guarantee that the selection of which actions to include is optimal, the heuristic is inadmissible.

3.4.3 Landmark heuristics

The idea behind landmark heuristics is to find and exploit so-called landmarks. A landmark for a given planning problem is a fact (or a disjunction of facts) that must hold in some state along every plan that solves the problem [30]. There are also action landmarks: an action is an action landmark if it is included in every plan that solves the planning problem [34, 64]. Finding landmarks is typically PSPACE-complete [55], but for many cases there exist efficient methods for finding and ordering landmarks [30, 37, 57].

Once landmarks have been extracted and ordered they can be used to construct heuristics. The LAMA planner [57] uses an inadmissible pseudo-heuristic that estimates the distance to the goal by counting the number of landmarks that are yet to be achieved. The estimate depends both on the current state s and the path taken from s_{init} to s. Building upon this, an admissible (still path-dependent) heuristic is constructed in [34] and used in the optimal planner BJOLP [12].

3.4.4 Pattern databases

Pattern database heuristics [7, 15] are based on relaxations. The underlying idea is to relax the planning problem by using an abstraction to transform the planning problem $P = (\Sigma, s_0, g)$ to another planning problem $P' = (\Sigma', s'_0, g')$ that is smaller and simpler to solve. For a given pattern p, which is a selection of ground facts, the abstracted problem is created by ignoring all facts that are not in the pattern. A state s is abstracted to $s' = s \cap p$, and an action a is abstracted to an action a' with preconditions $\mathrm{pre}(a') = \mathrm{pre}(a) \cap p$ and effects $\mathrm{eff}(a') = \mathrm{eff}(a) \cap p$. If $\pi = (a_1, \dots, a_n)$ is a solution to P, then $\pi' = (a'_1, \dots, a'_n)$ is a solution to P', so the cost of an optimal solution to P' is an admissible heuristic for P. The heuristic values are computed by solving the abstracted planning problem for all possible abstract states in advance and storing the resulting values in a so-called pattern database which acts as a lookup table.

As pattern databases require solving a planning problem to optimality for each possible abstract state, it is necessary to keep the number of possible states low, which limits the informativeness of the heuristic [27]. One possibility to alleviate this is to use independent patterns and use an additive heuristic that is the sum of several such heuristic functions [35], or to consider a generalization of pattern databases called merge-and-shrink [27] which consider a larger class of abstractions.

4

Motion planning

This chapter gives an introduction to the topic of motion planning, with a particular focus on path planning for nonlinear and nonholonomic systems. The optimal path-planning problem is defined, and sampling-based motion-planning methods in general are presented, with lattice-based motion planning being described in more detail.

4.1 Preliminaries

Motion planning is the problem of finding a feasible path or trajectory for a system from an initial state to a terminal state in an environment that may contain obstacles. It is often desirable that the path or trajectory, in addition to being feasible, should minimize some performance measure such as path length, time or energy consumption. The problem of finding such a path or trajectory is referred to as optimal motion planning.

Motion planning can refer to either path planning or trajectory planning. A *path* describes the geometric motion in space and is represented as $\mathbf{x}(s)$, $s \in [0, s_g]$ where s is the path parameter representing progression along the path. A *trajectory* $\mathbf{x}(s(t))$ is a time-parametrized path, i.e., a path with a velocity profile. A common method for trajectory planning is to first solve a path-planning problem and then solve a velocity-planning problem [43]. In the remainder of this chapter, focus therefore lies on path planning, although the definitions and methods mentioned can be used for trajectory planning as well.

The difficulty of motion planning depends on the properties of the system. In many cases, it is not possible to solve a motion-planning problem analytically and numerical methods are used instead [43]. There are, however, some exceptions, such as finding the shortest path in an obstacle-free environment for a Dubins car [14] or a Reeds-Shepp car [56]. It is also more challenging for non-holonomic systems than for holonomic systems [51], and more challenging for systems that are not differentially flat, i.e., systems where there is no output such that all states and inputs can be described using the output and a finite number of its derivatives [3]. This chapter focuses on methods for nonholomic systems that are not necessarily differentially flat. An overview of such methods can be found in [43, 51].

4.2 Problem formulation

Consider a continuous-time (nonlinear) system in the form

$$\dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \mathbf{u}(s), q(s)), \quad \mathbf{x}(0) = \mathbf{x}_{\text{init}} \tag{4.1}$$

where the parameter s denotes the length of the path travelled by the system, $\mathbf{x}(s) \in \mathbb{R}^n$ denotes the system state, $\mathbf{u}(s) \in \mathbb{R}^m$ denotes the control input of the system, $q(s)$ is the mode of the system, and $\mathbf{x}_{\text{init}}$ is the initial state of the system at path length $s = 0$. Some systems have only a single mode, while other systems may have several different modes, such as, e.g., forward and reverse motion. The system mode is subject to the constraint

$$q \in \mathcal{Q} \tag{4.2}$$

where $\mathcal{Q}$ is the set of possible system modes. The state and control input of the system are subject to the constraints

$$\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^n, \quad \mathbf{u} \in \mathcal{U} \subseteq \mathbb{R}^m \tag{4.3}$$

where $\mathcal{X}$ and $\mathcal{U}$ describe the physical constraints on the state and input, respectively. In addition to the physical constraints, there are additional constraints arising from obstacles in the environment. Denote the region occupied by obstacles with $\mathcal{X}_{\text{obst}}$. The free space is then defined as $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obst}}$.

The optimal path-planning problem can be defined as the problem of finding a feasible path $(\mathbf{x}(\cdot), \mathbf{u}(\cdot), q(\cdot))$ from an initial state $\mathbf{x}_{\text{init}} \in \mathcal{X}_{\text{free}}$ to a terminal state $\mathbf{x}_{\text{term}} \in \mathcal{X}_{\text{free}}$ such that a performance measure J is minimized. This can be posed as the following continuous-time optimal control problem (OCP):

$$\begin{aligned}
\underset{\mathbf{x}(\cdot),\mathbf{u}(\cdot),q(\cdot),S_f}{\text{minimize}} \quad & J = \int_0^{S_f} l(\mathbf{x}(s), \mathbf{u}(s), q(s))\, \mathrm{d}s \\
\text{subject to} \quad & \mathbf{x}(0) = \mathbf{x}_{\text{init}}, \quad \mathbf{x}(S_f) = \mathbf{x}_{\text{term}} \\
& \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \mathbf{u}(s), q(s)), \quad s \in [0, S_f] \\
& \mathbf{x}(s) \in \mathcal{X}_{\text{free}}, \quad s \in [0, S_f] \\
& \mathbf{u}(s) \in \mathcal{U}, \quad s \in [0, S_f] \\
& q(s) \in \mathcal{Q}, \quad s \in [0, S_f],
\end{aligned} \tag{4.4}$$

where $l(\mathbf{x}, \mathbf{u}, q) > 0$ is a performance measure. With the choice $l(\mathbf{x}, \mathbf{u}, q) = 1$ the resulting problem is that of finding a shortest path.

Algorithm 8 Single-query sampling-based motion planning.

1: Initialization: Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a directed graph where $\mathcal{E}$ is empty and $\mathcal{V}$ contains at least $\mathbf{x}_{\text{init}}$ and possibly $\mathbf{x}_{\text{term}}$.
2: Vertex selection: select a vertex $\mathbf{x} \in \mathcal{V}$ for expansion.
3: Extension: Select a configuration $\mathbf{x}_{\text{new}} \in \mathcal{X}_{\text{free}}$ and attempt to compute a feasible and collision-free path e from $\mathbf{x}$ to $\mathbf{x}_{\text{new}}$. If this fails, return to Step 2.
4: Insertion: If $\mathbf{x}_{\text{new}} \notin \mathcal{V}$, insert $\mathbf{x}_{\text{new}}$ in $\mathcal{V}$. Insert e in $\mathcal{E}$.
5: Solution check: Determine if a solution is found, and mark it as a candidate solution.
6: Termination check: If a termination condition is satisfied, return the solution with the lowest cost or return failure if there is no candidate solution. Otherwise, return to Step 2.

4.3 Sampling-based motion planning

A commonly used group of motion planning methods is sampling-based motion planning. The main idea of these methods is to sample the free configuration space and incrementally construct a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where the vertices correspond to configurations in the free configuration space, and the edges correspond to feasible and collision-free motions between configurations.

Sampling-based motion planning methods can be either single-query, where motion planning is performed for only one pair of initial and terminal state $(\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{term}})$ for each obstacle set, or multiple-query where motion planning is performed multiple times with varying $(\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{term}})$ for each obstacle set.

A general single-query sampling-based motion-planning algorithm is given in Algorithm 8. The key points to determine are how to select a vertex for expansion (line 2), how to select a configuration to extend to, as well as how to compute a feasible and collision-free path from the vertex to the configuration (line 3).

The vertex selection can be done either deterministically or randomly. Strategies based on random sampling often draw a random sample $\mathbf{x}_{\text{rand}} \in \mathcal{X}_{\text{free}}$ and select the nearest vertex. This is the strategy used by the Rapidly-exploring Random Tree (RRT) algorithm [44], that many sampling-based motion-planning algorithms are based on.

The extension step can be done in several different ways. One approach is to choose $\mathbf{x}_{\text{new}}$ first (possibly based on $\mathbf{x}_{\text{rand}}$ if applicable) and compute a feasible and collision-free path from $\mathbf{x}$ to $\mathbf{x}_{\text{new}}$ which is a motion-planning problem in itself, though easier to solve since the distance between $\mathbf{x}$ and $\mathbf{x}_{\text{new}}$ is typically small. Often, a path that is feasible with respect to the motion constraints is computed first and then checked to see if it is collision-free. Such a path can be found by solving an OCP numerically, or in some special cases analytically [43]. It is also possible to ignore the motion constraints and choose the path to be a straight line, in which case the solution path will be a sequence of waypoints. A separate smoothing step is then added after a path has been found where the straight lines are smoothed into feasible curves. Another approach to the extension step is to use a closed-loop controller to steer the system from $\mathbf{x}$ towards the

Algorithm 9 A general algorithm for roadmap construction.

```
1: Let 𝒢 = (𝒱, ℰ) be a directed graph where ℰ is empty and 𝒱 consists of n
   sampled points in 𝒳_free
2: for v ∈ 𝒱 do
3:     for n ∈ neighbours(v) do
4:         Attempt to compute a feasible and collision-free path e from v to n. If
           this fails, continue.
5:         Insert e in ℰ.
6:     end for
7: end for
```

next state, either until it is sufficiently close to the desired state, or for a given amount of time after which $\mathbf{x}_{\text{new}}$ is taken as the resulting state [41]. This does not allow for connecting states exactly, but can be computationally less demanding than solving an OCP in every step.

An extension to RRT is RRT* [32], which has asymptotic optimality guarantees. In RRT* a rewiring step is added, where vertices within a neighborhood of the newly-added vertex $\mathbf{x}_{\text{new}}$ are considered and edges can be rewired between this set of vertices and $\mathbf{x}_{\text{new}}$ if this results in a lower total cost of the path from $\mathbf{x}_{\text{init}}$. It is possible to use RRT* while considering motion constraints [33]. However, as the rewiring step requires finding a path that connects states exactly, it is necessary to solve an OCP rather than using closed-loop steering, which is computationally demanding.

For multi-query planning it is common to divide the motion planning into two phases: an offline preprocessing phase in which the graph $\mathcal{G}$ is constructed, and an online query phase in which a path between the given $(\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{term}})$ is computed. Many such methods are based on the probabilistic roadmap (PRM) method introduced in [36]. A general algorithm for constructing the so-called roadmap $\mathcal{G}$ is shown in Algorithm 9. To construct the roadmap, it is necessary to be able to sample points in $\mathcal{X}_{\text{free}}$. A function neighbours$(v)$ that returns a set of neighbour vertices is also required. As in the single-query case, it is also necessary to be able to compute feasible and collision-free paths between the sampled points. The same methods that are used in the single-query case can be used here as well.

In the query phase, the initial and terminal states $\mathbf{x}_{\text{init}}$ and $\mathbf{x}_{\text{term}}$ are added as vertices to the graph, and connected to the other vertices by following line 3–6 in Algorithm 9 for $v = \mathbf{x}_{\text{init}}, \mathbf{x}_{\text{term}}$. After that, graph search methods such as those described in Chapter 2 can be used to find a path from $\mathbf{x}_{\text{init}}$ to $\mathbf{x}_{\text{term}}$ in $\mathcal{G}$.

4.4 Lattice-based motion planning

Lattice-based motion planning can be seen as a special case of sampling-based motion planning where deterministic sampling is used [51]. The underlying idea is to transform the optimal motion-planning problem in (4.4) to a discrete graph-search problem. This is done by constructing a lattice structure consisting of a

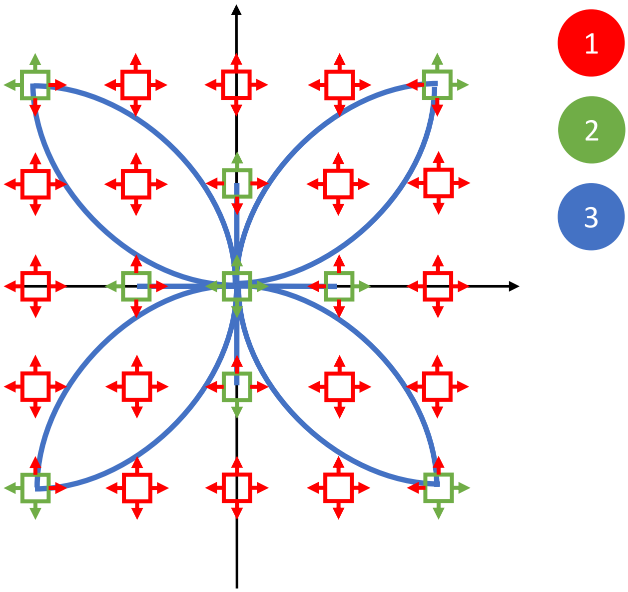


Figure 4.1: An illustration of the steps performed to construct the state-lattice. (1) Discretize the search space, (2) select which states to connect, (3) solve the OCPs to connect the states.

discrete set of states together with a discrete set of motion segments that detail allowed motions between the states in the lattice. Such a lattice can be generated using either a control-sampling approach or a state-lattice approach [3]. This thesis will consider only the state-lattice approach in which the state-space is discretized and states in the discretized state-space are connected by solving OCPs.

4.4.1 Constructing the state-lattice

The state-lattice construction is illustrated in Figure 4.1 and is performed offline in three steps [54]:

1. Discretization of the state space to obtain the discrete search space $\mathcal{X}_d$.

2. Selection of which discrete states to connect.

3. Computation of the motion primitive set by solving the OCPs defined by the previous step.

In the first step, the state space is discretized to obtain $\mathcal{X}_d$, which requires selecting the fidelity of the state space that should be used.

The second step is to select which pairs of states in $\mathcal{X}_d$ that should be connected. If the system is able to operate in different modes, the choice of which mode $q \in \mathcal{Q}$ to operate in can also be included in this step. If the system is position invariant, as is the case for many systems, it is sufficient to compute motion primitives from states that are positioned in the origin as these motion primitives can then be translated [54]. If the system is orientation invariant, the number of motion primitives that must be computed can be further reduced by mirroring and/or rotating motion primitives from a few initial headings [53].

In the last step, the motion primitive set $\mathcal{P}$ is constructed by computing the motion primitives that are required in order to connect the pairs of states that were chosen in the second step. This can be done using numerical optimal control [3, 47, 54]. A motion primitive $m \in \mathcal{P}$ is in this thesis defined as

$$m = (\mathbf{x}_m(s), \mathbf{u}_m(s), q_m) \in \mathcal{X} \times \mathcal{U} \times \mathcal{Q}, \quad s \in [0, S_m], \tag{4.5}$$

and represents a feasible path from an initial state $\mathbf{x}_m(0) = \mathbf{x}_{\text{start}} \in \mathcal{X}_d$ to a terminal state $\mathbf{x}_m(S_m) = \mathbf{x}_{\text{final}} \in \mathcal{X}_d$, the control inputs $u_m(s)$ required to move the system along the path, and the mode q_m of the system during the motion. The mode q_m is assumed to have been chosen during the second step. For each combination of $(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{final}}, q_m)$ as determined by the second step, the corresponding motion primitive is computed by solving the OCP

$$\begin{aligned}
\underset{\mathbf{x}(\cdot),\mathbf{u}(\cdot),S_f}{\text{minimize}} \quad & J = \int_0^{S_f} l(\mathbf{x}(s), \mathbf{u}(s), q_m)\,\mathrm{d}s \\
\text{subject to} \quad & \mathbf{x}(0) = \mathbf{x}_{\text{start}}, \quad \mathbf{x}(S_f) = \mathbf{x}_{\text{final}} \\
& \dot{\mathbf{x}}(s) = \mathbf{f}(\mathbf{x}(s), \mathbf{u}(s), q_m), \quad s \in [0, S_f] \\
& \mathbf{x}(s) \in \mathcal{X}, \quad s \in [0, S_f] \\
& \mathbf{u}(s) \in \mathcal{U}, \quad s \in [0, S_f]
\end{aligned} \tag{4.6}$$

where $l(\mathbf{x}, \mathbf{u}, q)$ is a cost function. A common choice for the cost function is

$$l(\mathbf{x}(s), \mathbf{u}(s), q) = 1 + \|\mathbf{x}(s)\|^2_{\mathbf{Q}(q)} + \|\mathbf{u}(s)\|^2_{\mathbf{R}(q)} \tag{4.7}$$

where the weight matrices $\mathbf{Q}(q) \in \mathbb{R}^{n\times n}$ and $\mathbf{R}(q) \in \mathbb{R}^{m\times m}$ are used to give a trade-off between path length and other measures such as the smoothness of the motion [3, 47]. The weight matrices can have different values for different system modes, as in [4] where different values of $\mathbf{Q}(q)$ are used for forward and reverse motion for a tractor-trailer in order to avoid large joint angles when reversing. An example of a resulting set of motion primitives for a car-like vehicle is shown in Figure 4.2.

4.4.2 Planning

After the state-lattice has been constructed offline, it can be used to simplify the motion-planning problem (4.4). Let the state transition function $\mathbf{f}_\mathbf{m}(\mathbf{x}, m)$ define

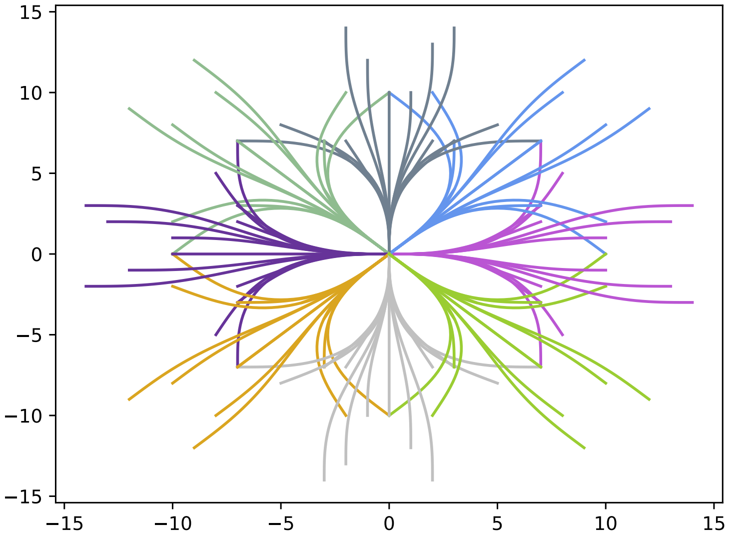


Figure 4.2: *Example of a set of motion primitives for a car-like vehicle. Each colour corresponds to a different initial heading of the vehicle.*

the resulting state when the motion primitive m is applied starting from the state $\mathbf{x}$. The path-planning problem can now be reduced to

$$\begin{aligned} \underset{\{m_k\}_{k=1}^{N},N}{\text{minimize}} \quad & \sum_{k=1}^{N} L_m(m_k) \\ \text{subject to} \quad & \mathbf{x}_1 = \mathbf{x}_{\text{init}}, \quad \mathbf{x}_{N+1} = \mathbf{x}_{\text{term}} \\ & \mathbf{x}_{k+1} = \mathbf{f}_{\mathbf{m}}(\mathbf{x}_k, m_k), \quad k = 1, \ldots, N \\ & m_k \in \mathcal{P}, \quad k = 1, \ldots, N \\ & \tau(\mathbf{x}_k, m_k, s) \in \mathcal{X}_{\text{free}}, \quad k = 1, \ldots, N, \quad s \in [0, S_{m_k}] \end{aligned} \tag{4.8}$$

where $\tau(\mathbf{x}_k, m_k, s)$ represents the path parametrized by s that is obtained when the motion primitive m_k is applied starting from the state $\mathbf{x}_k$, and

$$L_m(m) = \int_0^{S_m} l(\mathbf{x}_m(s), \mathbf{u}_m(s), q_m)\, \mathrm{d}s. \tag{4.9}$$

The resulting problem in (4.8) can be solved using graph-search methods described in Chapter 2, such as A*.

One possible choice of heuristic is to use a heuristic lookup table (HLUT) [38]. The HLUT is a pre-computed table of heuristic values. Typically, the optimal cost-

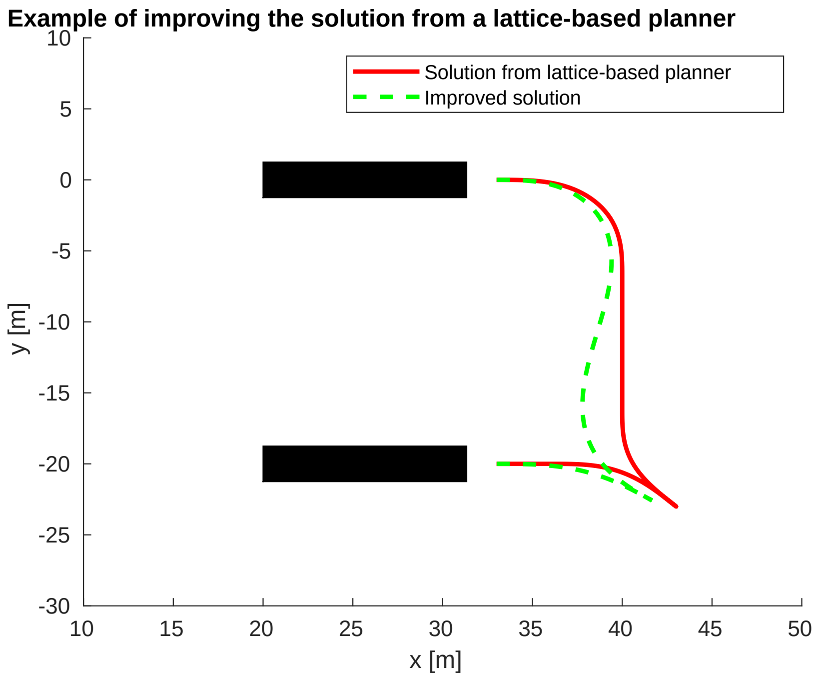


Figure 4.3: *Example of a path for a car-like vehicle computed by a lattice-based motion planner, and the resulting improved path. Black shapes denote obstacles.*

to-go value in free space is used as the heuristic value. This can be computed by solving the motion-planning problem for each combination of $\mathbf{x}_{\text{init}}, \mathbf{x}_{\text{term}} \in \mathcal{X}_d$. To reduce the number of motion-planning problems to solve, a length ρ is often chosen and only motion-planning problems with $\mathbf{x}_{\text{init}}$ positioned at the origin and $\mathbf{x}_{\text{term}} \in \mathcal{X}_d$, such that $\mathbf{x}_{\text{term}}$ is within a square centred at the origin with side length ρ are computed. By using a motion planner based on Dijkstra's rather than A*, it is possible to compute many heuristic values at once, increasing the efficiency of the computation.

4.5 Improvement using optimal control

The solutions computed by lattice-based motion-planning algorithms as described in Section 4.4 often suffer from discretization artefacts [1, 50]. It is therefore often desirable to smooth the solution in order to obtain smooth and continuously differentiable paths [3].

In [4], a method for using numerical optimal control to improve upon a solution that has been computed by a lattice-based motion planner is presented. Unlike traditional smoothing, which aims only to produce smooth paths, this improvement method also improves the path with respect to the objective function value. Given a solution in the form of a sequence of motion primitives $\{m_k\}_{k=1}^M$, the system mode sequence $\{q_k\}_{k=1}^M$ is fixed. This results in the OCP:

$$
\begin{aligned}
\underset{\{\mathbf{x}_k(\cdot),\mathbf{u}_k(\cdot),S_k\}_{k=1}^M}{\text{minimize}} \quad & J_{\text{imp}} = \sum_{k=1}^{M} \int_0^{S_k} l(\mathbf{x}(s), \mathbf{u}(s), q_k)\, \mathrm{d}s \\
\text{subject to} \quad & \mathbf{x}_1(0) = \mathbf{x}_{\text{init}}, \quad \mathbf{x}_M(S_M) = \mathbf{x}_{\text{term}} \\
& \mathbf{x}_k(0) = \mathbf{x}_{k-1}(S_{k-1}) \quad k = 2, \dots, M \\
& \dot{\mathbf{x}}_k(s) = \mathbf{f}(\mathbf{x}_k(s), \mathbf{u}_k(s), q_k), \quad s \in [0, S_k] \\
& \mathbf{x}_k(s) \in \mathcal{X}_{\text{free}}, \quad s \in [0, S_k] \\
& \mathbf{u}_k(s) \in \mathcal{U}, \quad s \in [0, S_k].
\end{aligned}
\tag{4.10}
$$

This problem can be solved using numerical optimal control. For an overview of numerical optimal control methods, see [3, 10]. Nonlinear programming (NLP) solvers such as IPOPT [62] or WORHP [6] can be used to solve the problem to local optimality. Such solvers typically need to be warm-started with a good initial solution, where a good solution means good both with respect to feasibility and objective function value [2]. According to the method in [4], the solver is warm-started with the solution obtained from the lattice-based motion planner, which is guaranteed to be feasible. The improvement step is also tightly integrated with the lattice-based motion planner as the same cost function is used both for the primitive generation and the improvement step. An example of a path for a car-like vehicle generated by a lattice-based motion planner together with the resulting path after an improvement step are shown in Figure 4.3.

5

Task and motion planning

This chapter introduces the integrated task and motion planning (TAMP) problem and gives a brief presentation of the methods that have been proposed in the literature.

5.1 Problem formulation

The field of integrated task and motion planning extends the fields of task planning and motion planning as described in Chapter 3 and Chapter 4, respectively. For an overview, see [19].

Task and motion planning is an extension of task planning that considers geometric and kinematic constraints as well. In task planning, all action parameters are discrete, but in task and motion planning the action parameters are allowed to be continuous as well. This also means that in addition to the preconditions and effects, actions may also include additional constraints on the continuous parameters [19]. As an example, consider a move action, which in a discrete task-planning problem might be defined as move(*robot*, *location1*, *location2*), where *location1* and *location2* are discrete variables. In a TAMP problem the move action might instead be move(*robot*, $\mathbf{x}_1$, $\mathbf{x}_2$, τ) where $\mathbf{x}_1, \mathbf{x}_2$ are continuous parameters for the state of the robot, and τ is a continuous parameter for the path or trajectory to take between $\mathbf{x}_1$ and $\mathbf{x}_2$. Constraints on the continuous parameters could be that the path or trajectory τ is feasible and collision-free.

The input to a TAMP problem is a state transition system $\Sigma = (\mathcal{S}, \mathcal{A}, \gamma)$ together with a continuous-time model of the system as in (4.1), the initial discrete and continuous states of the system $(s_{\text{init}}, \mathbf{x}_{\text{init}})$ and the set of discrete and continuous goal states S_g, X_g. A solution plan is a sequence of actions $a_0, \ldots, a_{N-1}$ with corresponding action parameters $\theta_0, \ldots, \theta_{N-1}$ such that $s_0 = s_{\text{init}}$, $\mathbf{x}_0 = \mathbf{x}_{\text{init}}$, $\gamma(s_k, \mathbf{x}_k, a_k, \theta_k) = (s_{k+1}, \mathbf{x}_{k+1})$ for all $k = 0, \ldots, N-1$ and $s_N \in S_g, \mathbf{x}_N \in X_g$. Clearly,

Algorithm 10 A general sequencing first algorithm.

1: Compute plan skeleton.
2: Attempt to find a valid parameter configuration. If fail, return to Step 1.
3: Return plan

this a more challenging problem than the task-planning problem as it requires finding feasible values of the continuous action parameters as well, requiring the solution of motion-planning problems.

5.2 TAMP approaches

Many different approaches to TAMP have been proposed. For a more thorough overview of existing methods, see [19, 22]. In general, methods can be grouped into one of three classes of methods depending on how they combine the search for an action sequence with the search for valid parameter configurations: sequencing first, satisfaction first or interleaved [19].

Sequencing first

One approach is to use sequencing first. A general algorithm for this group of methods is shown in Algorithm 10. First, a so-called plan skeleton is computed, which is an action sequence where the continuous parameters are free variables. These free variables are constrained, both by constraints that are part of each action, as well as by additional constraints that are required for the plan skeleton to achieve the desired goal. Once a plan skeleton is computed, the algorithm attempts to find a valid parameter configuration. If no such configuration exists, the algorithm computes a new plan skeleton. To make sure that the new plan skeleton is different from previously computed plan skeletons it is necessary to be able to backtrack or update the task-planning problem as done in [58]. Other sequencing first approaches include the task-motion kit [9] and PDDLStream [18].

Satisfaction first

The second approach is satisfaction first. A general algorithm for such methods is shown in Algorithm 11. Algorithms that use this strategy first sample parameter values that satisfy constraints, and then attempt to find an action sequence using those sampled values that solves the problem. If no such action value is found, new parameter values are sampled. This approach can be more efficient than sequencing first if sampling is efficient, action sequencing can be performed without much overhead, and/or sampled values are unlikely to satisfy critical constraints [19]. An example of one such algorithm is FFRob [17].

Algorithm 11 A general satisfaction first algorithm.

1: Sample new values that satisfy constraints.
2: Attempt to find an action sequence using the sampled parameter values. If fail, return to Step 1.
3: Return plan

Interleaved

The last group of algorithms interleaves the search for an action sequence with the search for valid parameter configurations. This can be done in various ways.

One approach is to sample some variable values, e.g., robot configurations and object poses, and leave others, e.g., paths and trajectories, as free variables during the search for an action sequence [19]. Examples of this include the semantic attachments approach [13].

Another approach is that in [59], where a sampling-based motion-planning algorithm is used to plan in a space that includes both the geometric and the symbolic, i.e., continuous and discrete, states of the system.

5.3 Optimal task and motion planning

Most of the proposed TAMP methods consider only the problem of finding a feasible plan, which is a difficult problem in itself, but there have also been some research effort directed toward optimal task and motion planning, where the goal is to find a plan that minimizes some performance measure such as path length, or time duration.

One of the first works to consider the optimal task and motion planning problem was [63], where the problem is formulated as a three-level optimization problem. The top-level problem is formulated as a travelling salesman problem, and the lower-level planners are used to iteratively refine and improve plans, passing the resulting costs upward.

The work in [61] considers a generalization of the TAMP problem called logic-geometric programming, where the goal is to optimize a cost function over the final geometric state, such as placing an object as high as possible over the ground.

In [60] an almost-surely asymptotically optimal planner is presented. The proposed planner integrates a symbolic planner based on Satisfiability Modulo Theories with sampling-based motion planning.

6

Concluding remarks

This chapter concludes the first part of this thesis. Here, the main contributions of the publications in Part II are summarized, and possible directions for future research are discussed.

6.1 Summary of contributions

In Paper A, a task and motion-planning problem with applications to open-pit mine drilling has been investigated. The problem consists of determining the order in which to drill holes at given locations, as well as finding feasible paths for the drill rig that do not pass over drilled holes. The problem is formulated as a variation of the travelling salesman problem that includes dynamic obstacles. A planner consisting of two nested graph-search planners has been proposed to solve the problem, where the top-level planner solves the ordering problem, and the low-level planner solves motion planning problems. Several different heuristics, admissible as well as inadmissible, have been proposed and successfully evaluated.

In Paper B, a rearrangement problem in which a tractor is tasked with rearranging a set of trailers has been investigated. A combined task and motion planner has been proposed that combines an LPA*-based task planner with a lattice-based motion planner to iteratively compute and update action costs. Both planners are based on A* search with an admissible heuristic, and the resulting planner has been shown to be resolution complete as well as resolution optimal. The proposed planner also uses ideas from branch and bound, and maintains upper and lower bounds on motion plan costs that are used to prune the search, which has been shown in the paper to increase the efficiency of the search without sacrificing optimality.

In Paper C, a method for improving task and motion plans for rearrangement problems in which a manipulator rearranges moveable objects was proposed. The method takes as input a feasible solution, which can be computed using a method such as the one from Paper B, and improves the plan by formulating and solving optimal control problems. Two different approaches were proposed and investigated: one that formulates and solves one larger OCP, and one that decomposes the optimal control problem into a sequence of smaller OCPs and optimizes parts of the solution at a time. The second approach was shown to maintain a feasible solution to the full problem at all times and that the quality of the solution is non-decreasing. Numerical experiments were conducted that indicate that this second approach can reduce the computation time compared to the first approach while still resulting in solutions of similar quality.

6.2 Future work

There are several possible interesting directions for future research. Some of these are presented here.

Domain independency

The planner that was proposed in Paper B is somewhat tailored to the rearrangement problem for tractor-trailers. It is fairly straightforward to apply it to other rearrangement problems, provided that the motion planner that is used has the same properties as the motion planner that was used, i.e., it is resolution complete and resolution optimal, and an underestimate of the cost of the plan can be extracted at all times. One possible direction of future research could therefore be to adapt the planner so that it can be used for other domains as well. This could include adding the possibility to solve problems specified using Planning Domain Definition Language (PDDL) [48], which would make it necessary to extend PDDL so as to be able to specify motion models and other constraints needed to define TAMP problems.

Heuristics

An interesting line of research is to investigate what heuristics, whether problem specific or domain independent, can be used to guide a search for a joint task and motion plan. Several domain-independent heuristics for task planning have been proposed in literature, and are presented in Section 3.4. However, many suffer from drawbacks such as not being admissible or being difficult to compute. They also assume that the cost of an action is known in advance and typically also that the cost depends only on the action and not on other aspects. This does not always hold for the task and motion planning setting, where the cost of a motion plan depends on the environment which depends on previous actions, and computing the costs of all actions might be intractable. In Paper B a problem-specific heuristic is used that uses the heuristic function used by the motion planner. It would be interesting to generalize this in order to find heuristics, preferably domain independent, that work well in a TAMP setting.

Mixed-integer nonlinear programming (MINLP)

Task and motion planning shows some similarities to mixed-integer nonlinear programming (MINLP) as it contains both discrete aspects (what actions to take and in which order), and continuous aspects (what continuous path to take). In [40] the TAMP problem is formulated as a mixed-integer linear programming (MILP) problem and solved using B&B techniques. However, this requires linear motion dynamics which is not always realistic. A possible line of research is therefore to explore how TAMP can be formulated and solved as an MINLP problem, both when solved from scratch as in Paper B, or when optimizing a plan that has already been found as in Paper C. Future work could also investigate if heuristics that have been successful for solving MINLP problems can also be applied to TAMP problems.

Improving efficiency

While the approach in Paper C that is inspired by finite-horizon optimal control reduces the computation time required compared to the first proposed method, there is still room for improvement. In Paper C, two different methods for handling collision-avoidance constraints for movable objects were investigated: representing obstacles with circles, and finding safety envelopes in the form of convex polygons. While the safety envelopes could improve the computation time, they reduced the performance due to being too conservative. Alternative methods could be investigated in order to find a method that speeds up the solution time without sacrificing performance. It could also be of interest to investigate if there are other ways to pose the optimization problem or other solvers to use that could make the methods in Paper C more efficient.

Bibliography

[1] Henrik Andreasson, Jari Saarinen, Marcello Cirillo, Todor Stoyanov, and Achim J Lilienthal. Fast, continuous state path smoothing to improve navigation accuracy. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 662–669, 2015.

[2] Kristoffer Bergman. *On motion planning using numerical optimal control*. Licentiate thesis no 1843, Department of Electrical Engineering, Linköpings universitet, Sweden, June 2019.

[3] Kristoffer Bergman. *Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments*. Dissertations no 2133, Linköping Studies in Science and Technology, May 2021.

[4] Kristoffer Bergman, Oskar Ljungqvist, and Daniel Axehill. Improved path planning by tightly combining lattice-based path planning and optimal control. *IEEE Transactions on Intelligent Vehicles*, 6(1):57–66, 2020.

[5] Blai Bonet and Héctor Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[6] Christof Büskens and Dennis Wassel. The ESA NLP Solver WORHP. In Giorgio Fasano and János D. Pintér, editors, *Modeling and Optimization in Space Engineering*, volume 73, pages 85–110. Springer New York, 2013.

[7] Joseph C Culberson and Jonathan Schaeffer. Pattern databases. *Computational Intelligence*, 14(3):318–334, 1998.

[8] Neil T Dantam, Zachary K Kingston, Swarat Chaudhuri, and Lydia E Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12, page 00052. Ann Arbor, MI, USA, 2016.

[9] Neil T Dantam, Swarat Chaudhuri, and Lydia E Kavraki. The task-motion kit: An open source, general-purpose task and motion-planning framework. *IEEE Robotics & Automation Magazine*, 25(3):61–70, 2018.

[10] Moritz Diehl, Hans Georg Bock, Holger Diedam, and P-B Wieber. Fast direct multiple shooting algorithms for optimal robot control. *Fast motions in biomechanics and robotics: optimization and feedback control*, pages 65–93, 2006.

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

[12] Carmel Domshlak, Malte Helmert, Erez Karpas, Emil Keyder, Silvia Richter, Gabriele Röger, Jendrik Seipp, and Matthias Westphal. BJOLP: The big joint optimal landmarks planner. In *IPC 2011 Planner Abstracts*, pages 91–95, 2011.

[13] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Trüg, Michael Brenner, and Bernhard Nebel. Semantic attachments for domain-independent planning systems. *Towards Service Robots for Everyday Environments: Recent Advances in Designing Service Robots for Complex Tasks in Everyday Environments*, pages 99–115, 2012.

[14] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[15] Stefan Edelkamp. Planning with Pattern Databases. In *ECP-01. Sixth European Conference on Planning*, pages 13–24, Toledo, September 2001.

[16] Marco Faroni, Alessandro Umbrico, Manuel Beschi, Andrea Orlandini, Amedeo Cesta, and Nicola Pedrocchi. Optimal Task and Motion Planning and Execution for Multiagent Systems in Dynamic Environments. *IEEE Transactions on Cybernetics*, 2023.

[17] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFrob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 179–195. Springer, 2015.

[18] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 440–448, 2020.

[19] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[20] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[21] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning and acting*. Cambridge University Press, 2016.

[22] Huihui Guo, Fan Wu, Yunchuan Qin, Ruihui Li, Keqin Li, and Kenli Li. Recent trends in task and motion planning for robotics: A Survey. *ACM Computing Surveys*, 2023.

[23] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[24] Anja Hellander and Daniel Axehill. On a traveling salesman problem with dynamic obstacles and integrated motion planning. In *2022 American Control Conference (ACC)*, pages 4965–4972, Atlanta, June 2022. IEEE.

[25] Anja Hellander, Kristoffer Bergman, and Daniel Axehill. Improved task and motion planning for rearrangement problems using optimal control. *To be submitted*.

[26] Anja Hellander, Kristoffer Bergman, and Daniel Axehill. On integrated optimal task and motion planning for a tractor-trailer rearrangement problem. In *62nd IEEE Conference on Decision and Control (CDC)*, Singapore, December 2023. IEEE.

[27] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM (JACM)*, 61(3):1–63, 2014.

[28] Jörg Hoffmann. FF: The fast-forward planning system. *AI magazine*, 22(3): 57–62, 2001.

[29] Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[30] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research*, 22:215–278, 2004.

[31] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011.

[32] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7): 846–894, 2011.

[33] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *2013 IEEE international conference on robotics and automation*, pages 5041–5047. IEEE, 2013.

[34] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1728–1733. Pasadena, CA, 2009.

[35] Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 174–181, 2008.

[36] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[37] Emil Keyder, Silvia Richter, and Malte Helmert. Sound and Complete Landmarks for And/Or Graphs. 2010.

[38] Ross A Knepper and Alonzo Kelly. High Performance State Lattice Planning Using Heuristic Look-Up Tables. In *IROS*, pages 3375–3380. Citeseer, 2006.

[39] Sven Koenig and Maxim Likhachev. Incremental A*. *Advances in neural information processing systems*, 14, 2001.

[40] Takuma Kogo, Kei Takaya, and Hiroyuki Oyama. Fast MILP-based Task and Motion Planning for Pick-and-Place with Hard/Soft Constraints of Collision-Free Route. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1020–1027. IEEE, 2021.

[41] Yoshiaki Kuwata, Justin Teo, Sertac Karaman, Gaston Fiore, Emilio Frazzoli, and Jonathan How. Motion planning in complex environments using closed-loop prediction. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, page 7166, 2008.

[42] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[43] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[44] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[45] Eugene L Lawler and David E Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[46] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in neural information processing systems*, 16, 2003.

[47] Oskar Ljungqvist, Niclas Evestedt, Daniel Axehill, Marcello Cirillo, and Henrik Pettersson. A path planning and path-following control framework for a general 2-trailer with a car-like tractor. *Journal of field robotics*, 36(8): 1345–1377, 2019.

[48] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL–the planning domain definition language. Technical report, Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[49] Dana S Nau, Vipin Kumar, and Laveen Kanal. General branch and bound, and its relation to A* and AO*. *Artificial Intelligence*, 23(1):29–58, 1984.

[50] Rui Oliveira, Pedro F Lima, Marcello Cirillo, Jonas Mårtensson, and Bo Wahlberg. Trajectory generation using sharpness continuous dubins-like paths with applications in control of heavy-duty vehicles. In *2018 European Control Conference (ECC)*, pages 935–940. IEEE, 2018.

[51] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[52] Judea Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[53] Mihail Pivtoraiko and Alonzo Kelly. Efficient constrained path planning via search in state lattices. In *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, pages 1–7. Munich Germany, 2005.

[54] Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[55] Julie Porteous, Laura Sebastia, and Jörg Hoffmann. On the extraction, ordering, and usage of landmarks in planning. In *6th European Conference on Planning*, 2001.

[56] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific Journal of Mathematics*, 145(2):367–393, 1990.

[57] Silvia Richter and Matthias Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177, 2010.

[58] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[59] Wil Thomason and Ross A Knepper. A unified sampling-based approach to integrated task and motion planning. In *The International Symposium of Robotics Research*, pages 773–788. Springer, 2019.

[60] Wil Thomason, Marlin P Strub, and Jonathan D Gammell. Task and motion informed trees (TMIT*): Almost-surely asymptotically optimal integrated task and motion planning. *IEEE Robotics and Automation Letters*, 7(4): 11370–11377, 2022.

[61] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1930–1936, 2015.

[62] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

[63] Chongjie Zhang and Julie A Shah. Co-optimizing task and motion planning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4750–4756. IEEE, 2016.

[64] Lin Zhu and Robert Givan. Heuristic planning via roadmap deduction. *4th. Int. Planning Competition Booklet (ICAPS-04)*, 2004.

Part II

Publications

Papers

The papers associated with this thesis have been removed for copyright reasons. For more details about these see:

https://doi.org/10.3384/9789180754644

Licentiate Theses
Division of Automatic Control
Linköping University

P. Andersson: Adaptive Forgetting through Multiple Models and Adaptive Control of Car Dynamics. Thesis No. 15, 1983.

B. Wahlberg: On Model Simplification in System Identification. Thesis No. 47, 1985.

A. Isaksson: Identification of Time Varying Systems and Applications of System Identification to Signal Processing. Thesis No. 75, 1986.

G. Malmberg: A Study of Adaptive Control Missiles. Thesis No. 76, 1986.

S. Gunnarsson: On the Mean Square Error of Transfer Function Estimates with Applications to Control. Thesis No. 90, 1986.

M. Viberg: On the Adaptive Array Problem. Thesis No. 117, 1987.

K. Ståhl: On the Frequency Domain Analysis of Nonlinear Systems. Thesis No. 137, 1988.

A. Skeppstedt: Construction of Composite Models from Large Data-Sets. Thesis No. 149, 1988.

P. A. J. Nagy: MaMiS: A Programming Environment for Numeric/Symbolic Data Processing. Thesis No. 153, 1988.

K. Forsman: Applications of Constructive Algebra to Control Problems. Thesis No. 231, 1990.

I. Klein: Planning for a Class of Sequential Control Problems. Thesis No. 234, 1990.

F. Gustafsson: Optimal Segmentation of Linear Regression Parameters. Thesis No. 246, 1990.

H. Hjalmarsson: On Estimation of Model Quality in System Identification. Thesis No. 251, 1990.

S. Andersson: Sensor Array Processing; Application to Mobile Communication Systems and Dimension Reduction. Thesis No. 255, 1990.

K. Wang Chen: Observability and Invertibility of Nonlinear Systems: A Differential Algebraic Approach. Thesis No. 282, 1991.

J. Sjöberg: Regularization Issues in Neural Network Models of Dynamical Systems. Thesis No. 366, 1993.

P. Pucar: Segmentation of Laser Range Radar Images Using Hidden Markov Field Models. Thesis No. 403, 1993.

H. Fortell: Volterra and Algebraic Approaches to the Zero Dynamics. Thesis No. 438, 1994.

T. McKelvey: On State-Space Models in System Identification. Thesis No. 447, 1994.

T. Andersson: Concepts and Algorithms for Non-Linear System Identifiability. Thesis No. 448, 1994.

P. Lindskog: Algorithms and Tools for System Identification Using Prior Knowledge. Thesis No. 456, 1994.

J. Plantin: Algebraic Methods for Verification and Control of Discrete Event Dynamic Systems. Thesis No. 501, 1995.

J. Gunnarsson: On Modeling of Discrete Event Dynamic Systems, Using Symbolic Algebraic Methods. Thesis No. 502, 1995.

A. Ericsson: Fast Power Control to Counteract Rayleigh Fading in Cellular Radio Systems. Thesis No. 527, 1995.

M. Jirstrand: Algebraic Methods for Modeling and Design in Control. Thesis No. 540, 1996.

K. Edström: Simulation of Mode Switching Systems Using Switched Bond Graphs. Thesis No. 586, 1996.

J. Palmqvist: On Integrity Monitoring of Integrated Navigation Systems. Thesis No. 600, 1997.

A. Stenman: Just-in-Time Models with Applications to Dynamical Systems. Thesis No. 601, 1997.

M. Andersson: Experimental Design and Updating of Finite Element Models. Thesis No. 611, 1997.

U. Forssell: Properties and Usage of Closed-Loop Identification Methods. Thesis No. 641, 1997.

M. Larsson: On Modeling and Diagnosis of Discrete Event Dynamic systems. Thesis No. 648, 1997.

N. Bergman: Bayesian Inference in Terrain Navigation. Thesis No. 649, 1997.

V. Einarsson: On Verification of Switched Systems Using Abstractions. Thesis No. 705, 1998.

J. Blom, F. Gunnarsson: Power Control in Cellular Radio Systems. Thesis No. 706, 1998.

P. Spångéus: Hybrid Control using LP and LMI methods – Some Applications. Thesis No. 724, 1998.

M. Norrlöf: On Analysis and Implementation of Iterative Learning Control. Thesis No. 727, 1998.

A. Hagenblad: Aspects of the Identification of Wiener Models. Thesis No. 793, 1999.

F. Tjärnström: Quality Estimation of Approximate Models. Thesis No. 810, 2000.

C. Carlsson: Vehicle Size and Orientation Estimation Using Geometric Fitting. Thesis No. 840, 2000.

J. Löfberg: Linear Model Predictive Control: Stability and Robustness. Thesis No. 866, 2001.

O. Härkegård: Flight Control Design Using Backstepping. Thesis No. 875, 2001.

J. Elbornsson: Equalization of Distortion in A/D Converters. Thesis No. 883, 2001.

J. Roll: Robust Verification and Identification of Piecewise Affine Systems. Thesis No. 899, 2001.

I. Lind: Regressor Selection in System Identification using ANOVA. Thesis No. 921, 2001.

R. Karlsson: Simulation Based Methods for Target Tracking. Thesis No. 930, 2002.

P.-J. Nordlund: Sequential Monte Carlo Filters and Integrated Navigation. Thesis No. 945, 2002.

M. Östring: Identification, Diagnosis, and Control of a Flexible Robot Arm. Thesis No. 948, 2002.

C. Olsson: Active Engine Vibration Isolation using Feedback Control. Thesis No. 968, 2002.

J. Jansson: Tracking and Decision Making for Automotive Collision Avoidance. Thesis No. 965, 2002.

N. Persson: Event Based Sampling with Application to Spectral Estimation. Thesis No. 981, 2002.

D. Lindgren: Subspace Selection Techniques for Classification Problems. Thesis No. 995, 2002.

E. Geijer Lundin: Uplink Load in CDMA Cellular Systems. Thesis No. 1045, 2003.

M. Enqvist: Some Results on Linear Models of Nonlinear Systems. Thesis No. 1046, 2003.

T. Schön: On Computational Methods for Nonlinear Estimation. Thesis No. 1047, 2003.

F. Gunnarsson: On Modeling and Control of Network Queue Dynamics. Thesis No. 1048, 2003.

S. Björklund: A Survey and Comparison of Time-Delay Estimation Methods in Linear Systems. Thesis No. 1061, 2003.

M. Gerdin: Parameter Estimation in Linear Descriptor Systems. Thesis No. 1085, 2004.

A. Eidehall: An Automotive Lane Guidance System. Thesis No. 1122, 2004.

E. Wernholt: On Multivariable and Nonlinear Identification of Industrial Robots. Thesis No. 1131, 2004.

J. Gillberg: Methods for Frequency Domain Estimation of Continuous-Time Models. Thesis No. 1133, 2004.

G. Hendeby: Fundamental Estimation and Detection Limits in Linear Non-Gaussian Systems. Thesis No. 1199, 2005.

D. Axehill: Applications of Integer Quadratic Programming in Control and Communication. Thesis No. 1218, 2005.

J. Sjöberg: Some Results On Optimal Control for Nonlinear Descriptor Systems. Thesis No. 1227, 2006.

D. Törnqvist: Statistical Fault Detection with Applications to IMU Disturbances. Thesis No. 1258, 2006.

H. Tidefelt: Structural algorithms and perturbations in differential-algebraic equations. Thesis No. 1318, 2007.

S. Moberg: On Modeling and Control of Flexible Manipulators. Thesis No. 1336, 2007.

J. Wallén: On Kinematic Modelling and Iterative Learning Control of Industrial Robots. Thesis No. 1343, 2008.

J. Harju Johansson: A Structure Utilizing Inexact Primal-Dual Interior-Point Method for Analysis of Linear Differential Inclusions. Thesis No. 1367, 2008.

J. D. Hol: Pose Estimation and Calibration Algorithms for Vision and Inertial Sensors. Thesis No. 1370, 2008.

H. Ohlsson: Regression on Manifolds with Implications for System Identification. Thesis No. 1382, 2008.

D. Ankelhed: On low order controller synthesis using rational constraints. Thesis No. 1398, 2009.

P. Skoglar: Planning Methods for Aerial Exploration and Ground Target Tracking. Thesis No. 1420, 2009.

C. Lundquist: Automotive Sensor Fusion for Situation Awareness. Thesis No. 1422, 2009.

C. Lyzell: Initialization Methods for System Identification. Thesis No. 1426, 2009.

R. Falkeborn: Structure exploitation in semidefinite programming for control. Thesis No. 1430, 2010.

D. Petersson: Nonlinear Optimization Approaches to $\mathcal{H}_2$-Norm Based LPV Modelling and Control. Thesis No. 1453, 2010.

Z. Sjanic: Navigation and SAR Auto-focusing in a Sensor Fusion Framework. Thesis No. 1464, 2011.

K. Granström: Loop detection and extended target tracking using laser data. Thesis No. 1465, 2011.

J. Callmer: Topics in Localization and Mapping. Thesis No. 1489, 2011.

F. Lindsten: Rao-Blackwellised particle methods for inference and identification. Thesis No. 1480, 2011.

M. Skoglund: Visual Inertial Navigation and Calibration. Thesis No. 1500, 2011.

S. Khoshfetrat Pakazad: Topics in Robustness Analysis. Thesis No. 1512, 2011.

P. Axelsson: On Sensor Fusion Applied to Industrial Manipulators. Thesis No. 1511, 2011.

A. Carvalho Bittencourt: On Modeling and Diagnosis of Friction and Wear in Industrial Robots. Thesis No. 1516, 2012.

P. Rosander: Averaging level control in the presence of frequent inlet flow upsets. Thesis No. 1527, 2012.

N. Wahlström: Localization using Magnetometers and Light Sensors. Thesis No. 1581, 2013.

R. Larsson: System Identification of Flight Mechanical Characteristics. Thesis No. 1599, 2013.

Y. Jung: Estimation of Inverse Models Applied to Power Amplifier Predistortion. Thesis No. 1605, 2013.

M. Syldatk: On Calibration of Ground Sensor Networks. Thesis No. 1611, 2013.

M. Roth: Kalman Filters for Nonlinear Systems and Heavy-Tailed Noise. Thesis No. 1613, 2013.

D. Simon: Model Predictive Control in Flight Control Design — Stability and Reference Tracking. Thesis No. 1642, 2014.

J. Dahlin: Sequential Monte Carlo for inference in nonlinear state space models. Thesis No. 1652, 2014.

M. Kok: Probabilistic modeling for positioning applications using inertial sensors. Thesis No. 1656, 2014.

J. Linder: Graybox Modelling of Ships Using Indirect Input Measurements. Thesis No. 1681, 2014.

G. Mathai: Direction of Arrival Estimation of Wideband Acoustic Wavefields in a Passive Sensing Environment. Thesis No. 1721, 2015.

I. Nielsen: On Structure Exploiting Numerical Algorithms for Model Predictive Control. Thesis No. 1727, 2015.

C. Veibäck: Tracking of Animals Using Airborne Cameras. Thesis No. 1761, 2016.

N. Evestedt: Sampling Based Motion Planning for Heavy Duty Autonomous Vehicles. Thesis No. 1762, 2016.

H. Nyqvist: On Pose Estimation in Room-Scaled Environments. Thesis No. 1765, 2016.

Y. Zhao: Position Estimation in Uncertain Radio Environments and Trajectory Learning. Thesis No. 1772, 2017.

P. Kasebzadeh: Parameter Estimation for Mobile Positioning Applications. Thesis No. 1786, 2017.

K. Radnosrati: On Timing-Based Localization in Cellular Radio Networks. Thesis No. 1808, 2018.

G. Lindmark: Methods and Algorithms for Control Input Placement in Complex Networks. Thesis No. 1814, 2018.

M. Lindfors: Frequency Tracking for Speed Estimation. Thesis No. 1815, 2018.

D. Ho: Some results on closed-loop identification of quadcopters. Thesis No. 1826, 2018.

O. Ljungqvist: On motion planning and control for truck and trailer systems. Thesis No. 1832, 2019.

P. Boström-Rost: On Informative Path Planning for Tracking and Surveillance. Thesis No. 1838, 2019.

K. Bergman: On Motion Planning Using Numerical Optimal Control. Thesis No. 1843, 2019.

M. Klingspor: Low-rank optimization in system identification. Thesis No. 1855, 2019.

A. Bergström: Timing-Based Localization using Multipath Information. Thesis No. 1867, 2019.

F. Ljungberg: Estimation of Nonlinear Greybox Models for Marine Applications. Thesis No. 1880, 2020.

E. Hedberg: Control, Models and Industrial Manipulators. Thesis No. 1894, 2020.

R. Forsling: Decentralized Estimation Using Conservative Information Extraction. Thesis No. 1897, 2020.

D. Arnström: On Complexity Certification of Active-Set QP Methods with Applications to Linear MPC. Thesis No. 1901, 2021.

M. Malmström: Uncertainties in Neural Networks: A System Identification Approach. Thesis No. 1902, 2021.

K. Nielsen: Robust LIDAR-Based Localization in Underground Mines. Thesis No. 1906, 2021.

H. Haghshenas: Time-Optimal Cooperative Path Tracking for Multi-Robot Systems. Thesis No. 1915, 2021.

A. Kullberg: On Joint State Estimation and Model Learning using Gaussian Process Approximations. Thesis No. 1917, 2021.

J. Nordlöf: On Landmark Densities in Minimum-Uncertainty Motion Planning. Thesis No. 1927, 2022.

S. A. Zimmermann: Data-driven Modeling of Robotic Manipulators—Efficiency Aspects. Thesis No. 1963, 2023.

S. Shoja: On Complexity Certification of Branch-and-Bound Methods for MILP and MIQP with Applications to Hybrid MPC. Thesis No. 1967, 2023.

FACULTY OF SCIENCE AND ENGINEERING

Linköping Studies in Science and Technology, Licentiate Thesis No. 1981, 2023
Department of Electrical Engineering

Linköping University
SE-581 83 Linköping, Sweden

www.liu.se

LINKÖPING
UNIVERSITY