THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

# Trajectory Optimization including Robot Controller Emulation

*Optimizing trajectories and generating robot code for industrial applications*

Daniel Gleeson

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2025

**Trajectory Optimization including Robot Controller Emulation**

*Optimizing trajectories and generating robot code for industrial applications*

Daniel Gleeson

ISBN 978-91-8103-282-6

Acknowledgements, dedications, and similar personal statements in this thesis, reflect the author's own views.



Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5740
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Cover:
A simulated and a physical image of the KUKA KR30 KRC2, and its surrounding workspace, in the Production Systems Lab at Chalmers University of Technology. Read more about accurately emulating the behavior of a physical robot in Chapter 4.

Printed by Chalmers Digital Printing
Gothenburg, Sweden, October 2025

**Trajectory Optimization including Robot Controller Emulation**

*Optimizing trajectories and generating robot code for industrial applications*

Daniel Gleeson
Department of Electrical Engineering
Chalmers University of Technology

# Abstract

The manufacturing industry has seen an ever-increasing level of automation with fully automated robotic assembly lines being commonplace in a wide variety of manufacturing industries. The automotive industry has had a prominent position in driving this development, and increasing the extent of automation has been a natural way to handle complex manufacturing processes and high throughput production. With industrial robots becoming cheaper and more available and product complexity and customization increasing, the use of robotics in industrial settings shows no signs of slowing down. With automated production lines, the use of simulation models and virtual commissioning has a place in every modern factory. A driving overall goal is to produce a virtual representation of the physical world, with a digital representation of every step in the manufacturing process, including CAD-file descriptions of both the production cell and the final product, specific process information, and full factory layouts. The work presented in this thesis is all related to bridging the gap between the digital and physical world, optimizing both the quality of the product and the efficiency of the robotic manufacturing process, while simplifying the work needed to be done to realize these solutions. One main contribution is modelling and optimizing collision free industrial robot trajectories in a cluttered environment, with trajectories defined by instantaneous torque values or as robot code equivalent parametrizations. To implement these solutions for a wide variety of robots, a robot controller emulator that executes robot code and accurately calculates the resulting robot trajectories has been developed. Improved and automated robot trajectories have been used in several applications, including in manufacturing processes that require advanced modelling techniques, such as the accurate modelling and optimization of robotic spray-painting trajectories.

**Keywords:** Industrial robots, trajectory optimization, robot controller, automatic code generation, manufacturing automation, spray painting.

*To Sofia, Seve and Linnea.*

# List of Publications

This thesis is based on the following publications:

[A] Staffan Björkenstam, **Daniel Gleeson**, Robert Bohlin, Johan S. Carlson, and Bengt Lennartson, "Energy Efficient and Collision Free Motion of Industrial Robots using Optimal Control".
Published in *Proceedings of the 2013 IEEE International Conference on Automation Science and Engineering (CASE)*, Madison, Wisconsin, USA, pp. 510–515, 2013. DOI: 10.1109/CoASE.2013.6654025.

[B] **Daniel Gleeson**, Staffan Björkenstam, Robert Bohlin, Johan S. Carlson, and Bengt Lennartson, "Optimizing Robot Trajectories for Automatic Robot Code Generation".
Published in *Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering (CASE)*, Gothenburg, Sweden, pp. 495–500, 2015. DOI: 10.1109/CoASE.2015.7294128.

[C] **Daniel Gleeson**, Staffan Björkenstam, Robert Bohlin, Johan S. Carlson, and Bengt Lennartson, "Towards Energy Optimization using Trajectory Smoothing and Automatic Code Generation for Robotic Assembly".
Published in *Proceedings of the 6th CIRP Conference on Assembly Technologies and Systems (CATS), Procedia CIRP*, Gothenburg, Sweden, Volume 44, pp. 341–346, 2016. DOI: 10.1016/j.procir.2016.02.099.

[D] **Daniel Gleeson**, Christian Larsen, Johan S. Carlson, and Bengt Lennartson, "Implementation of a Rapidly Executing Robot Controller".
Published in *Proceedings of the 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, Vancouver, BC, Canada, pp. 1341–1346, 2019. DOI: 10.1109/COASE.2019.8843254.

[E] **Daniel Gleeson**, Stefan Jakobsson, Raad Salman, Fredrik Ekstedt, Niklas Sandgren, Fredrik Edelvik, Johan S. Carlson, and Bengt Lennartson, "Generating Optimized Trajectories for Robotic Spray Painting".
Published in *The IEEE Transactions on Automation Science and Engineering (T-ASE)*, Volume 19, Issue 3, pp. 1380–1391, July 2022.
DOI: 10.1109/TASE.2022.3156803.

Other publications by the author, not included in this thesis, are:

[F] Staffan Björkenstam, Domenico Spensieri, Johan S. Carlson, Robert Bohlin, and **Daniel Gleeson**, "Efficient sequencing of industrial robots through optimal control".
Published in *Proceedings of the 5th CIRP Conference on Assembly Technologies and Systems (CATS), Procedia CIRP*, Dresden, Germany, Volume 23, pp. 194–199, 2014. DOI: 10.1016/j.procir.2014.10.094.

[G] **Daniel Gleeson**, Stefan Jakobsson, Raad Salman, Niklas Sandgren, Fredrik Edelvik, Johan S. Carlson, and Bengt Lennartson, "Robot spray painting trajectory optimization".
Published in *Proceedings of the 2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, Hong Kong, China, pp. 1135–1140, 2020. DOI: 10.1109/CASE48305.2020.9216983.

# Contents

# Acknowledgments

I would first like to extend my gratitude to my two main supervisors. Bengt Lennartson is Professor of Automation and the head of the Division of Systems and Control in the Electrical Engineering Department at Chalmers University of Technology. I want to thank you for being a kind and supportive mentor throughout my long PhD journey. I always genuinely felt you wanted my best and I appreciate everything you have done for me. Johan S. Carlson is the Director of Fraunhofer-Chalmers Centre of Industrial Mathematics (FCC). I am very grateful for how supportive you have been and how you have always believed in me, and I appreciate your style of leadership where you trust and see the best in everyone around you.

I am very thankful to my great colleagues at FCC, who all help to foster an inclusive and inspiring nature. I really enjoy all interesting discussions about work, life and everything. I would like to specially thank Staffan Björkenstam for all the years both leading and exploring together with me all the way back to my Master's Thesis work.

From my time employed as a part-time PhD-student at Chalmers University of Technology I would like to thank my fellow PhD-students for the friendly and warm atmosphere and all collaboration, discussions about optimization and joint work in the robotics lab.

To my parents, Ann and John, I have always felt your love and that "you are on my team", thank you for everything! You two are incredibly strong human beings that both have handled and dealt with everything life has had come your way and by doing so you have taught me so much! To my brother Christopher I want to say thank you for being the best big little brother anyone could wish for. I always enjoy talking to you and I cherish the connection we have always had. I have always had reason to be the proud big brother.

To my two children Seve and Linnea, who are the most amazing little humans I have ever met. I am constantly filled with immense warmth in my heart and pride in my chest at how kind and thoughtful you are, and how much you care for everyone around you.

And finally, to my wife Sofia, I am so grateful for all days I have had the fortune of spending with you, we have been through so much and there is nowhere I would rather be. You are a true inspiration to me and the reason I have a constant wish to improve myself. I feel so much love on so many levels for you, you are the love of my life!

Daniel Gleeson
Gothenburg, October 2025

# Acronyms

| | |
|---|---|
| BVP: | Boundary value problem |
| CAD: | Computer-aided design |
| DOF: | Degrees of freedom |
| FCC: | Fraunhofer-Chalmers Centre of Industrial Mathematics |
| FK: | Forward kinematics |
| FONC: | First order necessary conditions for optimality |
| IK: | Inverse kinematics |
| IPOPT: | Interior Point Optimizer (software library) |
| IPS: | Industrial Path Solutions (software) |
| KKT: | Karush-Kuhn-Tucker, first-order optimality conditions |
| LICQ: | Linear independence constraint qualification |
| LP: | Linear programming |
| NLP: | Non-linear programming |
| PLC: | Programmable logic controller |
| PMP: | Pontryagin's maximum principle |
| QP: | Quadratic programming |
| RCS: | Robot Controller Software |
| RRS: | Realistic Robot Simulation |
| RRT: | Rapidly-exploring Random Tree |
| SOSC: | Second order sufficient conditions for optimality |
| SQP: | Sequential quadratic programming |
| TCP: | Tool Center Point |

# Part I

# Overview

# CHAPTER 1

# Introduction

Historically, automation as a concept has had a profound impact on defining the shape of the modern industrialized world. It can be viewed as the foremost enabler for the industrial revolution and has been the go-to solution to achieve a wide variety of goals and improvements, such as saving labor and material, reducing waste, and improving quality, accuracy, and precision. By formulating the ironies of automation [1] in 1983, Bainbridge pointed out the fragility of automated systems, and that overreliance on skilled workers in an automated system may exacerbate the risks related to human factors. As a response to alleviate these issues, research has been conducted on formulating robust systems that fail gracefully, and developing tools that improve the working conditions of engineers and operators controlling the automated system.

Not only can a highly automated system be both complex and sensitive, and require highly skilled workers to maintain. Setting up such an automated system is also a labor-intensive undertaking, which requires domain specific knowledge to set up correctly, as well as extensive testing and physical prototyping. It is an iterative procedure where problems are discovered and corrected, often becoming increasingly costly to rectify if not found until

the later stages of the commissioning process. There is a large incentive to include more and more testing, assurances and optimization earlier on in the commissioning of a new automated production line. Not only to increase the performance of the system, but also to minimize or avoid unwelcome surprises through all stages of the process, and especially in the final stages just before production starts.

## 1.1 Background and motivation

An overall vision for production automation research at the Wingquist Laboratory at Chalmers University of Technology has been to make physical prototyping and testing obsolete [2]. It has been concisely formulated as: the first product to be produced should be sellable. The work presented in this thesis can on one hand be of use in this pre-production stage, by optimizing and verifying robot motions during the setup of an automated production line, or even for feasibility analysis before any physical products are finalized. On the other hand, by achieving a close correspondence to the real world, and implementing solutions as executable robot code, it can also be used when modeling and optimizing existing production lines.

### Automation and robotics in industry

The degree of automation and robotics in industrial manufacturing has been steadily increasing for decades. The main industries driving the field of automization forward, have from the very beginning been characterized by low variability, large batch-size, and high throughput production. Examples include automotive, packaging and consumer electronics industries. These industries are continuously working towards a more automated and robust production system, but automation and robotics have also been seen to revolutionize other industries such as logistics centers, and are increasingly used even in small batch-size production.

According to a report [3] in 2022 from the non-profit organization International Federation of Robotics, the year of 2021 saw an all-time high of newly commissioned industrial robots, with over half a million units installed, setting a new record of about 3.5 million operational industrial robots world wide.

The shift towards more flexible production lines with smaller batch-sizes

creates a need of streamlining the setup and commissioning period of robotic production lines. This includes all steps and problems to be solved when going from a definition of a task to be performed, to obtaining executable robot code. This thesis presents work related to the later stages of this chain of problems to be solved. The earlier stages include defining a task, performing reachability analysis for the robot, path planning to find a feasible collision free path, and load balancing as well as scheduling the tasks to be performed. From this point in the chain and onward we arrive at different types of problems, questions, and considerations that are addressed in this thesis. Even after a collision free piecewise linear path has been found, there is still a need to select or fine tune robot code parameters to produce executable robot code. Will the robot's internal limits affect the execution of a given trajectory? Is it possible to modify the found solution locally in order to improve the solution? How will the robot controller and its control system affect the executed trajectory, and will this depend on the brand and model of the robot? If more aspects of the problem at hand are included in the formulated optimization problem, then how will this affect the optimal trajectory?

## Automation and robotics research

If historically, the main focus of trajectory optimization was purely on minimizing distance and cycle time, there has now been a shift towards considering multiple and more complex objectives, when determining the optimal load balancing of a production line. One objective commonly introduced is to include a term aimed at reducing energy consumption [4]–[6]. In one example this is achieved by formulating an optimal control problem and obtaining a solution using a combination of direct collocation and indirect multiple shooting [7]. In another it is included together with an obstacle avoidance potential function in the optimal control formulation [8]. Other objectives include minimizing equipment cost [9], or increasing the robustness and reliability of the production, for example by reducing the wear and tear on robots and their dress packs, and thereby minimizing production down time [10].

In multiple robot trajectory optimization, examples include motion planning and coordination between multiple robots with independent performance measures [11], and globalization schemes for the optimization of multiple mobile robots' trajectories [12]. By handling the optimization of single robot trajectory optimization separately, the multiple robot case is regarded as a

scheduling problem [13], [14], including energy optimal trajectories [15], [16].

A prerequisite for considering more complex trajectory constraints or cost functions is to have robust methods for collision free path planning [17]–[22], and trajectory planning [23]. The work presented in this thesis will often make use of an initial trajectory found using a path planning algorithm, and this is a common globalization technique to guide a subsequent more localized search [24]. Including control signals in the problem formulation, and optimizing over them, we arrive at an optimal control formulation [25]. An optimal control problem is a well-suited formulation when optimizing to find robot control variables, and this is a formulation which has been extensively used and researched in a wide variety of fields such as chemistry [26], economics [27] and aeronautics [28]. Collision avoidance has also been included in stochastic [29], [30] and optimal control formulations of the trajectory optimization problem [31]. By decoupling the problem into a path planning step and a subsequent path tracking optimization problem, the latter can be formulated [32]–[37] without the need to include collision avoidance constraints.

## Industrial workflow and bottlenecks

The workflow of an operator or engineer who wishes to automate a certain task using industrial robots, typically consists of a number of well-defined and extensively studied steps.

The first step could be said to be a setup phase, where the task to be performed needs to be defined. This could for example be positions in space to be visited to perform a pointwise task, such as for example a stud weld, a spot weld, a laser measurement, applying a fastener, picking up or dropping off a part. For point wise tasks, defined as a single position in space, there could be a uniquely defined rotation of the tool needed to perform the task, or there could be multiple ways to position the tool for example in tasks that are rotationally symmetric. The task to be performed could also be defined as a one dimensional curve to be followed, this could for example be to lay down a sealing material, arc welding a seam, cutting into material along a specified curve, applying a string of glue, 3D-printing an object using a feeding nozzle, or solving factory logistics along specified delivery paths.

Examples of tasks defined over two-dimensional surfaces that occur in industry are spray painting a specified surface such as a sheet metal part, mapping out a factory layout or cleaning or disinfecting a work area. The task could

even be defined in three spatial dimensions as in scanning an industrial production line to get a 3D-representation of it, for example in the form of a point cloud.

The second step is a robot path planning step, which also might include load balancing and sequencing. The path planning problem is to find a collision free path to connect all tasks to be performed. There might be a necessary order in which the tasks are to be executed, or there might be an opportunity for improving the cycle time by finding an optimal sequence in which to perform the tasks. For robot stations with multiple robots, it might be possible to further reduce the cycle time by solving the assignment problem where we want to determine which robot should perform what task. When robots are working in the same space, with similar tasks, they might interfere with their respective work-area. This is where a synchronization method, perhaps making use of triggers and signals, could be used to make sure that the robots do not collide or interfere with each other.

The third step is the transition from a robot path to a robot trajectory and to executable robot code. In this trajectory planning step the robot path should be time stamped in the time domain and perhaps smoothed out in the spatial domain. Spatial smoothing will not only reduce the length of the path but also reduce accelerations and forces along the trajectory. When these solutions have been found, the operator typically would like to test the solutions in a physical robot line by generating and then running robot code that corresponds to the optimized trajectories.

## Problem focus

The main focus of the work presented in this thesis is in some way connected to improving the industrial workflow described above. The goal can either be to improve some bottleneck in the workflow, so that the workflow is further automated. Or it can be to improve the found solution, either by finding a better solution as defined by some objective function, or that the solution is a closer match to the reality of later stages in the workflow, as we get closer to a physically realizable solution.

There are a number of different types of problems encountered when trying to fulfill these goals. How can collision avoidance be included in trajectory optimization in an efficient way? How are competing and conflicting objectives handled best? Examples include the competing goals of cycle time versus en-

ergy consumption or how to balance the quality metrics of a painted surface with the parameters determining the executability of the robot trajectory of the painting robot. How can found results be generalized for different robot manufacturers with differences both on the level of robot code semantics and functionality, as well as inter-manufacturer discrepancies on the robot trajectory level even when issuing similar robot commands? What level of effort should be put into optimizing the different stages of the trajectory generation? Where are there substantial improvements to be found? Do we need to include a large set of optimization variables and solve the whole system simultaneously to get the best solution, or is it more beneficial to take a more incremental approach and focus on a few of the variables at a time?

## Funding

While working on the papers presented in this thesis I have been employed at the mathematical research institute Fraunhofer-Chalmers Centre (FCC). I have also had a part-time employment at Chalmers University of Technology where I was part of the Automation group at the Department of Signals and Systems and later the Department of Electrical Engineering. I started my work at FCC by writing my master's thesis in the area of trajectory optimization and optimal control of a manikin, a mathematical model of the human body. After that, my work pivoted over towards the robotics module of the FCC software platform Industrial Path Solutions (IPS), with the goal of including similar trajectory optimization procedures for robot trajectories as had been done for the manikin.

Funding for the research has been provided by a number of different internal and external projects and research initiatives. It has been supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the Wingquist Laboratory VINN Excellence Centre, as part of the Sustainable Production Initiative and the Production Area of Advance at Chalmers University of Technology. Research has been conducted within a number of different projects. One project is the EU-project Automation and Robotics for European Sustainable Manufacturing (AREUS), concerning eco-friendly design and programming of robotized factories. Research has also been conducted within the VINNOVA project Sustainable motions - SmoothIT, where optimization algorithms were used to reduce the energy use and increase the service life of moving equipment in industry. Research related to opti-

mization of robotic painting trajectories has been supported in part by the project SelfPaint, funded by the Fraunhofer Gesellschaft and the internal program for business oriented strategic alliances (WISA), and in part by Formas, a Swedish Research Council for Sustainable Development, and the project RoboClean, which optimizes and automates a robotic solution for cleaning processes in the food industry. This work was also during the last year supported in part by the Swedish Governmental Agency for Innovation Systems, VINNOVA, through the Advanced Digitalization program and the projects The Virtual PaintShop – AI-Boosted Automated Painting (AUTOPAINT) and Simulation-driven plasma optimization for improved paint adhesion and bonding (SIMPL).

## 1.2 Research approach

As an Industrial PhD-student working on problems where the goal is to solve real world industrial cases, the goal is to develop physically accurate models and algorithms that not only find a high quality solution, but also scale well with the problem size so that it can be solved in a reasonable amount of time. For complex problems where accuracy, optimality, and reasonable solving times cannot be simultaneously achieved, there is a need for making a trade-off between the different objectives. This can be done by either tailoring the problem formulation, introducing meaningful heuristics or simplifying the model. When it comes to the scientific contribution, given a sufficiently complex problem, a scientifically interesting solution typically coincides with solving for industrial goals and needs in cases where all criteria are fulfilled. Furthermore, pushing the boundaries and improving each of these criteria can even have its own research merits. But in cases where a trade-off is needed, the scientific and industrial interests might diverge. As an example, in an industrial setting, by prioritizing model accuracy and solving speed, and settling for a good enough solution by industrial standards, the problem might be solved using a simplified set of optimization routines. On the other hand, the main focus in a scientific contribution might be to include multi-physics modeling in the optimization problem for increased accuracy, or to provide a proof for an improved lower bound of the optimal solution.

To state it succinctly, the research approach has been to develop, improve, and use state-of-the-art models of physical systems, and to include these mod-

els in industrially valuable large-scale optimization problems.

The physical systems studied include robot controller internals and spray paint droplet simulation, but all contributions and implementations presented in this thesis have an overreaching goal of simplifying the engineering workflow in the set-up of a robotic production line. Improving the workflow of production lines has unsurprisingly received a lot of interest, and in order to be able to automate and make this process more efficient, there are a number of core technologies and algorithms that have been developed. These are constantly being improved both by improving the algorithms and by making use of increasing computational power. It is possible to divide the workflow into three main stages, where each stage typically makes use of results obtained in preceding stages. The first two categories, path planning and load balancing, will briefly be introduced in this thesis, to set the scene and to explain the starting point of most of the presented work. This thesis and the appended papers are all mainly related to the third stage in the automated workflow, which could be denoted time synchronization and post processing.

To summarize, the research has been fueled by perceived needs in industrial applications, the ambition of knowledge creation, and the will to improve and generalize solutions. To increase understanding of the problem at hand, there is often a need to implement an accurate model. However, models of increased complexity are often cumbersome or impossible to incorporate in an optimization routine. This typically leads to an iterative process of formulating generalized mathematical models that capture relevant behavior, and then simplifying the models while trying to retain the most important aspects of the underlying behavior.

## 1.3 Research questions

The research presented in this thesis has been centered around the following three research questions.

**RQ1:** *For a robot with freely moving and accurately controlled actuators, how can the optimal collision free trajectory be found when optimizing for cycle time and/or energy consumption?*

The first research question considers an industrial robot that is free to move all joints and other actuators between target configurations and is not bound

to follow a specified path. It is assumed that the robot can be accurately and instantaneously controlled at each joint, instead of providing the robot with target commands and letting the robot controller calculate the resulting trajectory.

**RQ2:** *How can robot controllers from different manufacturers be accurately emulated and included in an optimization scheme for automatic robot code generation?*

Different robot manufacturers have their unique set of robot commands that are used to execute a robot trajectory. Their proprietary robot controller parses and executes the robot code by calculating the trajectory and feeding the robot with the desired control values. By emulating the robot controller behavior, and including robot code parameters as variables in the optimization problem, the resulting optimal solution can be converted to directly executable robot code.

**RQ3:** *How can optimal robotic spray painting trajectories be found for an arbitrary CAD-modeled surface?*

The goal is to find a spray painting trajectory, which when executed will produce a paint coverage where all points on the geometry have a paint thickness sufficiently close to a specified target thickness. It is desirable to quickly find solution trajectories even for small batch-size production, without relying on extensive manual input.

## 1.4 Contributions

The main contributions presented in this thesis can be stated as follows with regard to how they connect to the five included papers.

- *Modeling and optimizing collision free industrial robot trajectories in a cluttered environment, with trajectories defined by instantaneous torque values or as robot code equivalent parametrizations.* Paper A describes an optimization scheme for robot trajectories in a cluttered environment where the robot is controlled by using the torque values in each time instance as control variables. In Paper B and Paper C the optimization uses a parametrization of the trajectory matching robot controller input

parameters, which makes it possible to export the found solution as executable robot code. As starting point for the optimization, a piecewise linear trajectory from a path planning algorithm is used as a feasible initial solution.

- *The implementation of a robot controller emulator that executes robot code and accurately calculates the resulting robot trajectories.* Paper D describes how controllers from different robot manufacturers are accurately simulated. An efficient algorithm is formulated, with a computation time orders of magnitude faster than the emulated time, making it possible to include it in optimization schemes. The emulator has been implemented in the Robotics module of the IPS (Industrial Path Solutions) software under the name of Controller Light.
- *The accurate modeling and optimization of robotic spray painting trajectories.* Paper E describes the generation of an initial trajectory, as well as the optimization with regard to paint thickness and the experimental setup used for model calibration. The optimization routine has been developed to be included as an integral part of achieving the automated workflow of scanning and painting incoming parts in an automated robotic paint booth.

## 1.5 Outline

This chapter, **Chapter 1**, is the introduction of the thesis and covers background, motivation, research questions and contributions. This chapter is now concluded by outlining the structure of the following chapters of Part I, the Outline of the thesis. The included publications are found in Part II of the thesis.

**Chapter 2:** Concepts related to mathematical optimization in a broader sense are introduced and described, with an explicit focus on **optimization methods**. These include how an optimization problem is formulated, its optimality conditions, and the connection between the continuous and discrete formulation of the problem.

**Chapter 3:** The **trajectory optimization** chapter covers robot trajectory representations, path planning, and a presentation of theory and examples related to trajectory optimization.

**Chapter 4:** This chapter on **robot controller emulation** presents how robot controllers from robot manufacturers can be interfaced, differences between trajectory representations, and the work done on developing an internal representation of a robot controller.

**Chapter 5:** The chapter is a brief introduction to robotic **spray painting** with respect to simulation, verification, and optimization.

**Chapter 6:** In this chapter all included publications are briefly summarized.

**Chapter 7:** The final chapter starts out with some concluding remarks that also reconnect the stated research questions with the presented work, and ends by laying a path forward for further research in related areas.

# CHAPTER 2

# Optimization methods

In general, an optimization problem is any problem where we want to find the optimal value, be it either a minimum or maximum value. This is something we might encounter daily, solving the problem on the fly, without giving it much thought, or it might be a highly complex problem that is challenging or even impossible to solve in practice. Examples from the first category might be calculations performed subconsciously to be able to catch a thrown ball mid-air. Or it might be a problem that requires more conscious effort to formulate and solve. Where, when, and for how long should we charge our car on our long road trip, in order to arrive at our destination as quickly as possible without ever running lower than a 5% battery charge level?

Optimization problems of different kinds and complexity are commonly encountered in industry. How many blades should our new wind turbine have to maximize efficiency and minimize noise and vibrations? How many robots are needed in the robot cell to be able to weld the sheet metal parts together within the specified cycle time of the production line? In this chapter we will in Section 2.1 introduce notation to describe these types of optimization problems mathematically, along with theory for conditions of optimality in Section 2.2. Thereafter, in Section 2.3, two different classes of solution

methods are described, and an example problem is solved using both of these methods to highlight differences and similarities between active set and interior point methods. In Section 2.4 we will dive deeper into theory on optimal control, which is the underlying problem formulation for most of the work presented in this thesis.

## 2.1 Optimization problem formulation

An unconstrained optimization problem might be formulated as

$$\min_x f(x),$$

where $f(x)$ is a function over the single or multi-variable input $x$.

Constraining the search space by including equality constraints, $g(x) = 0$, and inequality constraints, $h(x) \leq 0$, the problem formulation changes to

$$\begin{aligned} \min_x \quad & f(x), && \text{(2.1a)} \\ \text{subject to} \quad & g(x) = 0, && \text{(2.1b)} \\ & h(x) \leq 0. && \text{(2.1c)} \end{aligned}$$

Both $g(x)$ and $h(x)$ are in general vector-valued functions over the single or multi-variable input $x$. It should be noted that for a given constrained optimization problem either $g(x)$ or $h(x)$ might be absent.

The examples of optimization problems given in the introduction of this chapter, are problems where the optimal value is to be found over a discrete or continuous function of one or more variables.

A formulation often encountered in optimal control problems is to instead optimize over a set of control functions $u(x, t)$. The cost is now given by a cost functional instead of a cost function, since we are optimizing over a space of functions. But in practice, for problems presented in this thesis, this distinction is not always obvious or necessary, since we typically discretize the problem before optimizing. With the discretized version of the problem, we are back to optimizing over a set of input variables. The alternative, to optimize before discretizing, is also possible to do and results in a calculus of variations formulation [38].

## 2.2 Optimality conditions

To determine if a point is optimal for an unconstrained single variable function, we can check the first order derivative of the cost function at the point. If it is zero, the point is a stationary point, and by also examining the second order derivative it can give information about if the point is a maximum, minimum or a saddle point. These conditions can be generalized and concisely stated as follows for multi-variable unconstrained optimization problems:

**First order necessary conditions:** If the function $f(x)$ is continuously differentiable in an open neighborhood of the local minimizer $x^*$, then the following holds:

$$\nabla f(x^*) = 0.$$

**Second order necessary conditions:** If $\nabla^2 f(x)$ is continuous in an open neighborhood of the local minimizer $x^*$, then the following two statements hold:

$$\nabla f(x^*) = 0,$$
$$\nabla^2 f(x^*) \text{ is positive semi-definite.}$$

**Second order sufficient conditions:** If $\nabla^2 f(x)$ is continuous in an open neighborhood of $x^*$, then $x^*$ is a strict local minimizer of $f(x)$ if the following two conditions hold:

$$\nabla f(x^*) = 0,$$
$$\nabla^2 f(x^*) \text{ is positive definite.}$$

For a multi-variable constrained problem, (2.1), where equality and inequality constraints have been added, similar conditions of optimality can still be formulated.

### KKT conditions

First-order necessary conditions for optimality of a solution to a constrained nonlinear programming problem were formulated by Kuhn and Tucker [39], and independently by Karush [40]. Using a problem formulation as stated in (2.1), with $m$ number of equality constraints and $n$ number of inequality con-

straints, the Karush-Kuhn-Tucker (KKT) conditions can be stated as points fulfilling the following set of equations [41]

$$g_i(x^*) = 0, \quad i = 1, \ldots, m, \tag{2.2a}$$
$$h_j(x^*) \leq 0, \quad j = 1, \ldots, n, \tag{2.2b}$$
$$\mu_j^* \geq 0, \quad j = 1, \ldots, n, \tag{2.2c}$$
$$\mu_j^* h_j(x^*) = 0, \quad j = 1, \ldots, n \tag{2.2d}$$
$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) + \sum_{j=1}^{n} \mu_j^* \nabla h_j(x^*) = 0. \tag{2.2e}$$

The new variables introduced here, one for each of the constraints, are referred to as Lagrange multipliers and are used to formulate a relationship between the gradient of the cost and the gradient of the constraints. For each of the equality constraints of $g(x)$ a multiplier, $\lambda$, is introduced, while $\mu$ is used for the inequality constraints of $h(x)$.

The Karush-Kuhn-Tucker (KKT) conditions can be more compactly stated after introducing the Lagrangian

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^T g(x) + \mu^T h(x).$$

and grouping the equations with respect to what condition they are related to. A point $(x^*, \lambda^*, \mu^*)$ is called a KKT point if it satisfies the following conditions

$$\text{Stationarity:} \quad \nabla_x \mathcal{L}(x^*, \lambda^*, \mu^*) = 0, \tag{2.3a}$$
$$\text{Primal feasibility:} \quad g(x^*) = 0, \quad h(x^*) \leq 0, \tag{2.3b}$$
$$\text{Dual feasibility:} \quad \mu^* \geq 0, \tag{2.3c}$$
$$\text{Complementary slackness:} \quad \mu^* \odot h(x^*) = 0. \tag{2.3d}$$

Here $\odot$ denotes the elementwise Hadamard product, which makes the complementary slackness condition equivalent to $\mu_j^* h_j(x^*) = 0, \forall j$.

## Constraint qualifications

**Linear independence constraint qualification (LICQ):** For a point $x$, we denote $\mathcal{A}(x)$ to be the set of active constraints at $x$. Linear indepen-

dence constraint qualification holds at $x$ if the gradients of the set of active constraints are linearly independent. LICQ holds at points where the matrix $[\nabla g(x) \quad \nabla h_{\mathcal{A}}(x)]$ of gradients of the equality constraints and the active inequality constraints has full rank. Points where LICQ holds are regular points.

**First order necessary conditions (FONC):** If $x^*$ is both a local optimum and is regular, then there exists a unique vector of multiplier values $\lambda^*$ and $\mu^*$ such that $(x^*, \lambda^*, \mu^*)$ is a KKT point.

**Second order sufficient conditions, (SOSC):** We consider here functions $f$, $g$ and $h$ that are $C^2$, which means that the functions, and their first and second derivatives are continuous. We also define the set of feasible directions at a point $x^*$ as

$$\mathcal{F} = \{v \,|\, \nabla g(x^*)^\top v = 0, \quad \nabla h_i(x^*)^\top v \leq 0, \quad \forall i \in \mathcal{A}\}.$$

For a regular point $x^*$ where there exists multiplier values $\lambda^*$ and $\mu^*$ such that $(x^*, \lambda^*, \mu^*)$ is a KKT point, $x^*$ is a local minimum if the following inequality holds

$$d^\top \nabla^2 \mathcal{L}(x^*, \lambda^*, \mu^*) d > 0$$

for any direction $d \in \mathcal{F} \setminus \{0\}$ with $\nabla h_i(x^*)^\top d = 0$ for $\mu_i^* > 0$.

## 2.3 Active set methods versus interior point optimization

When faced with a constrained optimization problem as formulated in (2.1), there are a number of different solution methods that can be used to hopefully find an optimal solution to the problem. One way of trying to solve the problem, is to first focus on trying to remove the inequality constraints. This can be done by dividing the problem into two parts. First selecting a set of constraints with the goal of finding all constraints that are active at the optimum, and then solving the problem using only these constraints as equality constraints. Methods of solving the optimization problem that make use of this idea, are commonly referred to as active set methods. Areas where active set methods are commonly used range from linear programming (LP), the special case of (2.1) where the cost function and all constraints are linear,

and sequential quadratic programming (SQP) where a non-linear problem is solved as a sequence of quadratic programming (QP) problems.

In the linear case, the optimal value will be found at the boundary of the feasible region. Solving the problem using an active set method, will iterate through intersection points of constraints as the set of active constraints are updated. An example of a method using active set updating is the classic Simplex algorithm for linear programming [42]. As long as the optimal set of constraints remains to be found, these intersection points will continue to jump around the boundary of the feasible region. More generally though, for a given set of active constraints, their intersection will not necessarily lie within the feasible region. This behavior of possibly violating some of the constraints during intermediate steps of the minimization search will happen also for active set methods for nonlinear optimization.

Interior point methods on the other hand are named so because they often try to keep the evaluated points of the iterative search in the interior of the feasible region. One quite straight forward way of making sure only feasible points are evaluated during the search, is to assign an infinite cost to all infeasible points. The way this is typically done, is to introduce so-called barrier functions in the cost function. These barrier functions should preferably be differentiable in the feasible region, but grow to infinity as constraints are approached from the interior of the feasible region, pushing the search away from the constraints. These barrier functions are meant to keep the solution in each step of the search iterations from violating the constraints.

It is not necessarily beneficial to force every search iteration to be strictly feasible, but instead allow some constraint relaxation during intermediate steps. Of the KKT-conditions, the most problematic condition for an iterative solver to handle, is the complementary slackness criteria (2.3d). It is a criteria with two components, the Lagrangian multiplier $\mu^*$, and the value of the inequality constraint function $h(x)$, where one or both of the components need to be zero for the criteria to be fulfilled. We say the criteria has two branches, one where the constraint is active and $h(x) = 0$, and the other when it is inactive and $\mu^* = 0$. The sign of each non-zero component is determined by the primal feasibility (2.3b) and the dual feasibility (2.3c), respectively. A plot of $h(x)$ against $\mu^*$ would show that points that fulfill the complimentary slackness criteria will lie on the two half-axes of negative $h(x^*)$ and positive $\mu^*$, with a sharp corner at the origin. This is problematic for any algorithm

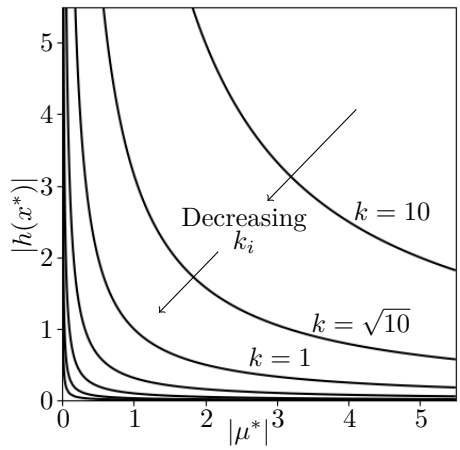



**Figure 2.1:** Relaxing the complementarity slackness condition $\mu^* h(x^*) = 0$ by introducing a constant $k$ on the right hand side of the equation. The figure shows the level curves, $k_i$, of $|\mu^* h(x^*)|$ for decreasing values of $k_i$.

that includes constraint linearizations. By relaxing this criteria with some constant, $k = \mu^* h(x^*)$, we smooth out the corner, see Figure 2.1, making it possible to switch from one solution branch to the other. The idea is to navigate around the corner to the correct solution branch before lowering the relaxation constant and homing in on a solution.

## Optimization example with two non-linear constraints

As an illustrative example, consider the following constrained minimization problem

$$\min_x \quad x_1^2 + x_1 x_2 + x_2^2 - 3x_1 - 3x_2, \tag{2.4a}$$

$$\text{subject to} \quad (x_1/2 - 2)^3 - x_2 + 4 \leq 0, \tag{2.4b}$$

$$x_1^3 + x_2 - 4 \leq 0. \tag{2.4c}$$

The minimization problem has a quadratic cost function, $f(x)$, over two variables, $x = [x_1, x_2]^\top$, and is constrained by two third degree polynomials. The level curves of the cost function and the two constraints can be seen in Fig-

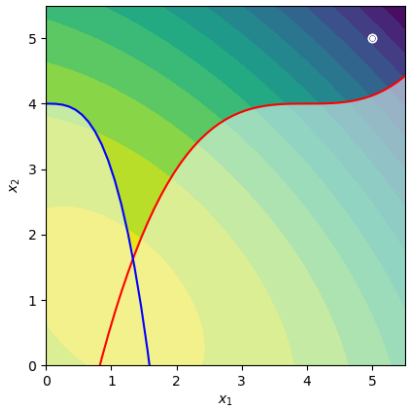



**Figure 2.2:** Level curves shown for the cost function $f(x)$ along with the two non-linear constraints in red and blue, that define the feasible region by excluding the shaded area below them.

ure 2.2 along with a given starting point for the search at $x_0 = [5, 5]^\top$. If we only consider the minimization term of (2.4a), the unconstrained problem would have a solution at $x^* = [1, 1]^\top$ where the objective function takes the value $f(x^*) = -3$. Turning our attention to the two constraints, we see that they define the feasible region by excluding the shaded area below the curves, and the optimum should be found down in the crevasse where the two constraints intersect.

One way to find the optimal solution is to use an active set method and iteratively update our current point until it converges to a solution. If we at the starting point linearize the two constraints, we get the two dotted lines in the top left figure of Figure 2.3. Solving this problem with linear constraints gives us a new point from where we can linearize the constraints and continue our search. Note that even if the initial point was feasible, the fact that only the linearized constraints are considered, has resulted in a situation where the found solution of the quadratic programming problem is not in the feasible set of the original problem. In the following iterations shown in Figure 2.3 we can see how the search first updates the point back into the feasible region and then rather quickly converges to the optimal value.

To demonstrate the differences between the solution methods we will also solve the same optimization problem using an interior point method. One way

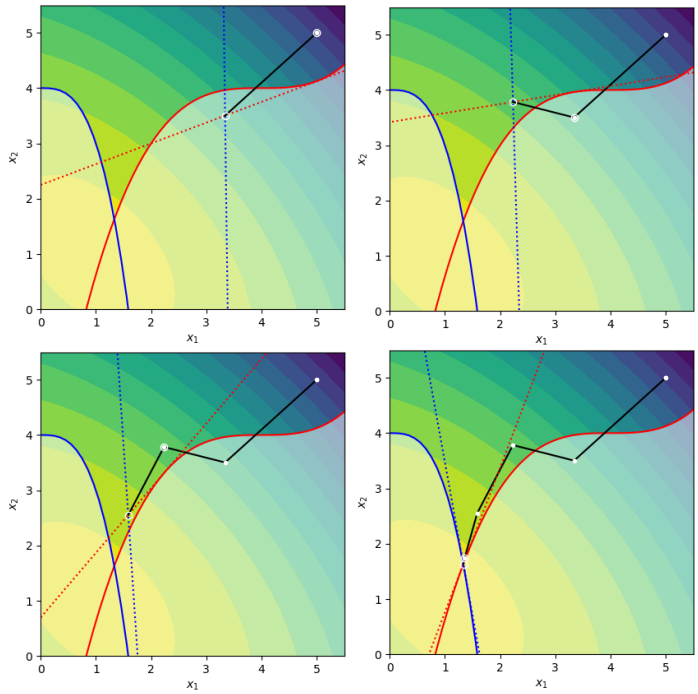



**Figure 2.3:** The first, second, third, and fifth iterations taken by the SQP solver on a problem with third degree constraints. In each iteration the linearizations of the constraints at the current point are also shown.

to do this is to introduce a logarithmic barrier function $B(x)$, which is infinite for positive values of $h_i(x)$ and otherwise is defined by

$$B(x) = \sum_i -\log(-h_i(x)), \tag{2.5}$$

and adding it to the original cost function. The benefit of this is that it now is possible to formulate and solve an unconstrained problem

$$\min_x \quad f(x) + kB(x). \tag{2.6}$$

As the parameter $k$ is lowered the solution to this unconstrained problem

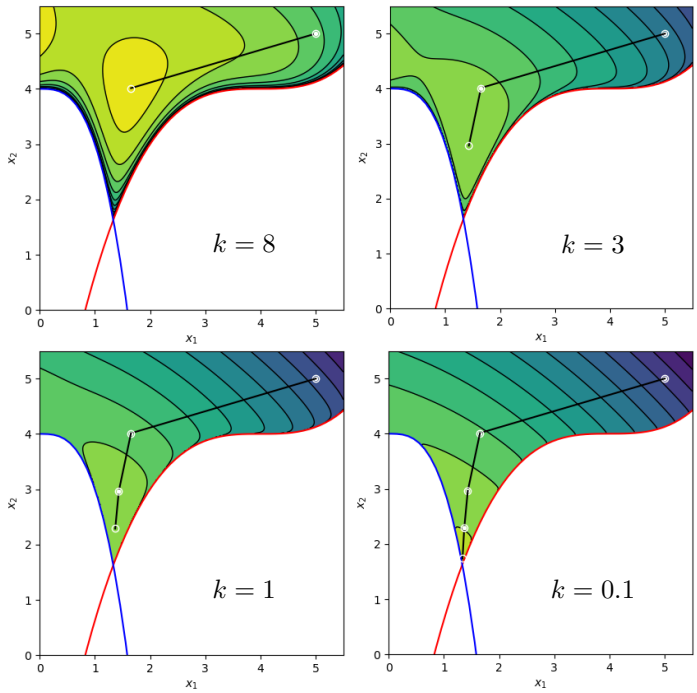



**Figure 2.4:** Level curves shown for the cost function combined with a logarithmic barrier function for each of the nonlinear constraints. As the barrier weight parameter $k$ is lowered, the optimum of the unconstrained problem improves with respect to the original constrained problem.

becomes a better approximation of the solution to the original constrained optimization problem. The barrier functions restrict the search to the interior of the feasible region, but the downside is that the simple quadratic term of the objective function is turned into a more complex and non-linear function. This new function can both have local minima, and will necessarily have a highly non-linear behavior close to the constraints, especially for low values of $k$.

Using this solution method to solve the problem of (2.4), we can add the barrier functions and use the same initial point as was used with the active set method. By iteratively solving for lower values of $k$ the solution approaches the optimal solution of the original problem as can be seen in Figure 2.4.

## 2.4 Optimal control

In an optimal control problem, the goal is to find the control of a system with its corresponding state that will optimize some cost over a period of time, subject to constraints. The constraints of the optimal control problem will include dynamic constraints that define the state evolution of the dynamical system. They will also often include path constraints that limit the variables along the trajectory, as well as endpoint constraints specifying initial and terminal conditions.

The time evolution of the state depends on both the current state, $x(t)$, and control, $u(t)$, according to the so-called state equation. This is formulated as a differential equation as follows

$$\dot{x} = f(x, u). \tag{2.7}$$

In particular, the full trajectory of the state, $x(t)$, can be determined for a given initial state $x_0$ and a particular applied control $u(t)$ over the interval $t \in [0, T]$.

The overall cost, $J$, is often stated to be composed of two terms. The first term includes costs related to endpoint values, $\Phi(x(t_s), t_s, x(t_f), t_f)$, while the second term is a running cost, $L(x(t), u(t), t)$, which is integrated up over time. For a specified time interval, $t \in [t_s, t_f]$, the optimal control problem formulation can then be stated as a minimization of the cost functional

$$J = \Phi(x(t_s), t_s, x(t_f), t_f) + \int_{t_s}^{t_f} L(x(t), u(t), t)dt, \tag{2.8a}$$

while satisfying the following additional constraints.

$$\text{State equations:} \quad \dot{x}(t) = f(x(t), u(t), t) \tag{2.8b}$$

$$\text{Path constraints:} \quad g(x(t), u(t), t) \geq 0 \tag{2.8c}$$

$$\text{Terminal conditions:} \quad h(x(t_s), t_s, x(t_f), t_f) = 0 \tag{2.8d}$$

These state equations and path constraints need to be fulfilled along the trajectory for all times $t \in [t_s, t_f]$. This general formulation of an optimal control problem can be found explicitly stated in Paper A, Paper B, and Paper C.

## Pontryagin's maximum principle

Pontryagin's maximum principle (PMP) states a number of conditions that are necessary to fulfill for a control input to be optimal. The principle was formulated by Pontryagin [43] and has since then been restated with modernized notation [44], [45]. These conditions are stated using the control Hamiltonian. For a specified cost functional, $J$, as in (2.8a) we formulate the control Hamiltonian, $H$, of the system as

$$H(x(t), u(t), \lambda(t), t) = \lambda^\top(t) f(x(t), u(t)) + L(x(t), u(t)). \tag{2.9}$$

The optimal state trajectory, $x^*$, optimal control, $u^*$, and Lagrange multipliers $\lambda^*$ must minimize the Hamiltonian for all times $t \in [0, T]$ and all permissible control inputs $u \in \mathcal{U}$,

$$H(x^*(t), u^*(t), \lambda^*(t), t) \leq H(x(t), u, \lambda(t), t). \tag{2.10}$$

Restating the inequality of (2.10) in terms of a minimization problem to be solved when finding the optimal control solution for a given problem we have

$$u^*(t) = \arg\min_u H(x^*(t), u, \lambda^*(t), t). \tag{2.11}$$

Just as the trajectory of the state is governed by the state equation (2.7), the time evolution of the Lagrangian vector, $\lambda$, should equal the state derivative of the Hamiltonian according to the costate equation,

$$\dot{\lambda}^\top(t) = -\frac{\partial H}{\partial x}, \tag{2.12}$$

and should fulfill the terminal conditions

$$\lambda^\top(T) = \left.\frac{\partial \Phi}{\partial x}\right|_{x(T)}, \tag{2.13a}$$

$$\left.\frac{\partial \Phi}{\partial T}\right|_{x(T)} + H(T) = 0, \tag{2.13b}$$

where the second terminal condition (2.13b) only applies for non-fixed final states.

The stated conditions are necessary to fulfill for optimality, but they are not

sufficient. They can be thought of as linking the time evolution of the state, as described by the state equation (2.8b), with a condition of optimality with respect to the cost functional (2.8a). There might also be terminal conditions (2.8d) that should be fulfilled, and these could be included to formulate a two-point boundary value problem (BVP). If there are also inequality constraints that should be fulfilled along the path as in (2.8c), the switch from active to inactive constraints will create junction points. This might happen at multiple points along the trajectory, and the junction points will have their own set of boundary constraints, which will need to be included in a multi-point BVP.

## Double integrator example

As an example to show how Pontryagin's maximum principle can be used to solve an optimal control problem, we consider a minimum time control problem for a double integrator. The states of the double integrator are the position, $x_1(t)$, and its time derivative, $x_2(t) = \dot{x}_1(t)$, and the system is controlled using the second-order time derivative as control signal,

$$u(t) = \dot{x}_2(t) = \ddot{x}_1(t).$$

The goal is to bring both states to zero as quickly as possible from a given initial state $x_0$, which can be stated as follows using the standard optimal control problem formulation of (2.8)

$$\min_u \int_0^{t_f} 1 dt = \min_u t_f, \tag{2.14a}$$

while satisfying

$$\dot{x}_1(t) = x_2(t), \tag{2.14b}$$
$$\dot{x}_2(t) = u(t), \tag{2.14c}$$
$$x(0) = x_0, \tag{2.14d}$$
$$\Phi(x(t_f)) = (x_1(t_f), x_2(t_f))^\top = (0, 0)^\top, \tag{2.14e}$$

where the control signal, $u(t)$, is limited to values in the symmetric interval $[-u_{\max}, u_{\max}]$. For simplicity we will use $u_{\max} = 1$ in the exemplified solutions.

### Solution method 1. Solving the continuous problem

One way to solve the optimal control problem is to formulate the optimality conditions for the continuous problem. To do this we first formulate the Hamiltonian, $H$, of the system according to the definition given in (2.9)

$$H(x(t), u(t), \lambda(t), t) = \lambda_1(t)x_2(t) + \lambda_2(t)u + 1. \tag{2.15}$$

The optimal control solution is found by minimizing the Hamiltonian as stated in (2.11)

$$u^*(t) = \arg\min_u 1 + \lambda_1(t)x_2^*(t) + \lambda_2(t)u.$$

Since $u$ only enters the expression in the last term, the optimal control value must obtain its maximum positive or negative value depending on the value of $\lambda_2(t)$. This type of optimal solution is referred to as bang-bang control

$$u^*(t) = \begin{cases} u_{\max}, & \text{if } \lambda_2(t) < 0, \\ -u_{\max}, & \text{if } \lambda_2(t) \geq 0. \end{cases}$$

The time evolution of the Lagrangian vector must fulfill the costate equation of (2.12). This means that the time derivative of the Lagrange multipliers are $\dot{\lambda}_1(t) = 0$ and $\dot{\lambda}_2(t) = -\lambda_1(t)$, which gives us $\lambda_1(t) = c_1$, and $\lambda_2(t) = c_2 - c_1 t$.

In a similar manner, by using the fact that the control variable takes one of the two constant values of $u(t) = \pm u_{\max}$, we have

$$x_1(t) = x_1(0) + x_2(0)t \pm u_{\max}t^2/2, \tag{2.16a}$$
$$x_2(t) = x_2(0) \pm u_{\max}t. \tag{2.16b}$$

By squaring the equation for $x_2(t)$, multiplying by $\pm u_{\max}/2$, and subtracting the two equations, we can eliminate the time dependency and find the relation

$$x_1(t) \pm u_{\max}x_2^2(t)/2 = \text{constant}. \tag{2.17}$$

This relation defines a switching curve with two different solutions regions. A plot of this switching curve can be seen in Figure 2.5.

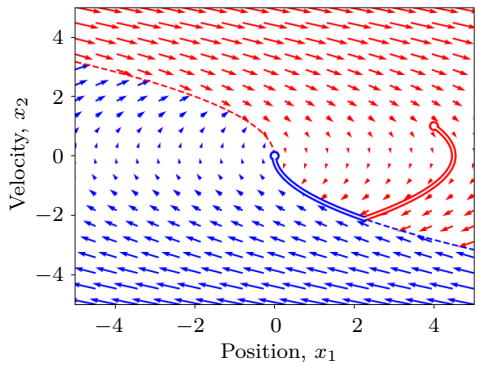



**Figure 2.5:** Switching curve for optimality for the double integrator example problem. The vector field of arrows show a small integration forward in time for different starting conditions. Optimal curve shown for a given set of initial conditions, first in red and then in blue after the control value switch.

### Solution method 2. Solving the problem by first discretizing and then optimizing.

The same formulation of the double integrator example makes it possible to compare two different solution methods. It is also possible to find a solution by first discretizing the problem, and then solving the discrete problem. Here the position, $x_1$, the velocity, $x_2$, and the acceleration which in this case is the control signal, $u$, are all discretized at $n+1$ discretization points.

$$x_1 = \begin{bmatrix} x_1[0] \\ x_1[1] \\ \vdots \\ x_1[n] \end{bmatrix}, \quad x_2 = \begin{bmatrix} x_2[0] \\ x_2[1] \\ \vdots \\ x_2[n] \end{bmatrix}, \quad u = \begin{bmatrix} u[0] \\ u[1] \\ \vdots \\ u[n] \end{bmatrix}.$$

These are together with the final time, $T$, the variables, $w$, of our optimization problem

$$w = \begin{bmatrix} T & x_1[0] & x_2[0] & u[0] & \dots & x_1[n] & x_2[n] & u[n] \end{bmatrix}.$$

We have upper and lower bounds on our control variable as well as initial and terminal constraints on our two states

$$u_{\min} \leq u[i] \leq u_{\max}, \quad i \in \{0, 1, \dots n\}, \tag{2.18a}$$
$$x_1[0] = x_{10}, \tag{2.18b}$$
$$x_2[0] = x_{20}, \tag{2.18c}$$
$$x_1[n] = 0, \tag{2.18d}$$
$$x_2[n] = 0. \tag{2.18e}$$

The system dynamics are discretized using a first order approximation and added as equality constraints to the optimization problem

$$0 = x_1[i+1] - x_1[i] - \frac{T}{n}\frac{x_2[i+1] + x_2[i]}{2}, \tag{2.18f}$$
$$0 = x_2[i+1] - x_2[i] - \frac{T}{n}\frac{u[i+1] + u[i]}{2}. \tag{2.18g}$$

The optimization problem can then be stated as a minimization over $T$, subject to the constraints (2.18).

For an initial position of $x_{10} = 4$, initial velocity of $x_{20} = 1$ and using 40 discretization points, ($n = 39$), we obtain the optimal solution seen in Figure 2.6. Comparing the solution to the plot of the switching curve of Figure 2.5 we can see that they both describe the same solution. Specifically, we can see that the optimal solution switches from a minimal control value to a maximal control value, and this happens at a point where the position and velocity values match the values of the switching point in Figure 2.5.

In this chapter different solution methods have been presented for solving non-linear optimization problems and optimal control problems. The main solution method used in Paper A, Paper B, Paper C, and Paper E is to first model the problem continuously, then possibly simplifying the model, before discretizing the continuous problem into a non-linear optimization problem. The final step is to solve the resulting problem with a general-purpose non-linear solver, for example an interior point solver such as IPOPT[46].

There are a number of ways to do these steps as we already have seen in this chapter. Choices made in the modeling and discretization steps will affect the solvability of the resulting optimization problem and the optimality of the found solution. It is therefore important to keep the later stages in mind as the

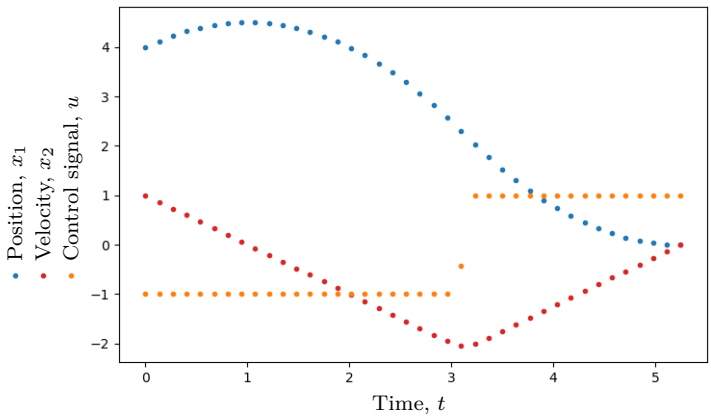



**Figure 2.6:** Optimal solution found for the discretized version of a double integrator example problem. The position $x_1$ (blue), velocity $x_2$ (red), and control signal $u$ (yellow) are shown are shown at each discrete time value.

problem is formulated. The following chapter describes and breaks down robot trajectory optimization in a number of distinct steps. Simplified problems are used to exemplify what might go wrong if the modeling and discretization leads to weaknesses in the problem formulation that the solver most often will find and exploit. This can in turn lead to unwanted characteristics of the solution or convergence towards infeasible solutions.

## 2.5 Summary

This chapter has described theory and introduced nomenclature related to optimization problems and the solution methods used to solve them. A general mathematical formulation of an optimization problem was introduced in Section 2.1. In Section 2.2 conditions of optimality were introduced. Both necessary and sufficient conditions were stated for an unconstrained optimization problem. After introducing the KKT-conditions, the corresponding conditions were also stated for a constrained NLP problem.

In Section 2.3 active set methods and interior point methods were described, and similarities and differences were discussed using simplified example problems. Section 2.4 took the step from a finite dimensional problem to an infinite

dimensional problem by formulating an optimal control problem, where the goal is to find an optimal control function, subject to constraints. This is a type of problem formulation that is used in most of the papers presented in this thesis. We saw how an example problem with a double integrator could be solved analytically using Pontryagin's maximum principle, or how it could first be discretized and solved using a general-purpose NLP solver.

The following chapter will go into the specifics of what we need to consider when an optimal control problem formulation is to be used within the field of robot trajectory optimization.

# CHAPTER 3

# Trajectory optimization

When aiming to solve a trajectory optimization problem, it is quite natural to first focus on solving a path planning problem. This is because it is quite common to have a problem formulation with a set of tasks to be performed, where every task is a specified target point in space. Each task could in turn be performed using a specific set of robot configurations. Even when the task is specified by both a fixed position and orientation of the TCP, there are typically still multiple ways for a robot to fulfill the task, see Figure 3.1 for an example of two different robot configurations with the same TCP transformation. These configurations can be differentiated in the robot code using some sort of configuration data or by using more informal descriptions for the placement of for example the second joint which is referred to as the elbow of the robot. In the figure we see the robot in a configuration with the elbow up and the blue shadow alternative shows the robot in an elbow down axis configuration.

The resulting path does not hold information about the time it takes to reach the destination. But minimizing the length of the path is often tightly correlated with lower execution time. For complex geometric obstacles it can also be exceedingly difficult to even determine that there exists a collision

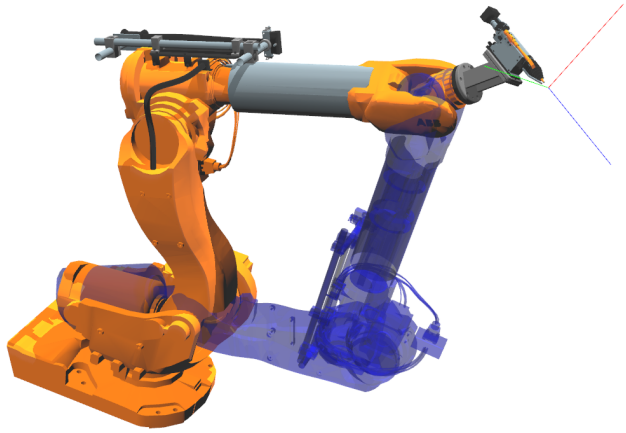


**Figure 3.1:** Two different configurations of the robot that meet the demands of a task with a specified position and orientation of the robot Tool Center Point (TCP). The TCP is in the figure represented by the three red, green and blue lines that together define the local coordinate frame of the tool.

free path between specified target points [17]–[22]. If the target points are specified as points in space that the robot must reach, then the very first problem to solve is the inverse kinematics problem. This is the problem of finding combinations of joint values that fulfill target point constraints of the end effector.

## 3.1 Joint space coordinates and inverse kinematics

When a robot is tasked with for example spot welding at a specific point, the tool of the robot should be placed in the correct position, with correct orientation. The spot weld position is a target point for the TCP of the robot. It is defined in the TCP-space of the robot. The problem of determining joint values of the robot, that matches the robots TCP to a target point, is a problem which is referred to as inverse kinematics (IK). This term is in contrast with forward kinematics (FK) where the position of the TCP is calculated for given values of each robot joint. There might be a few different

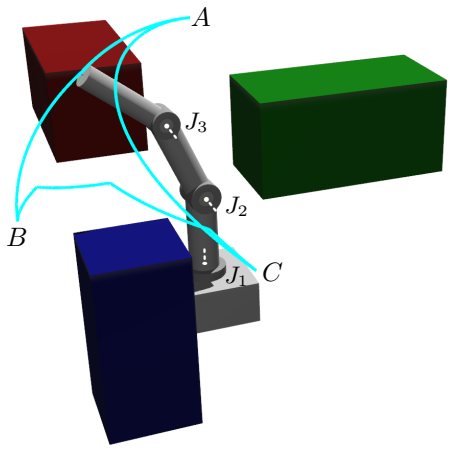



**Figure 3.2:** Simplified representation of a three degree of freedom robot arm using cylinders to represent joints and links. The three rotational axes of the robot, denoted $J_1$, $J_2$ and $J_3$, are shown as white dashed lines. The surrounding geometry is made up of three rectangular blocks in red, green and blue. A collision free trajectory moving between the three defined points $A$, $B$ and $C$ while avoiding the blocks, is exemplified by showing its TCP trace in cyan.

configurations for which the robot can perform a specific task. If we select any one of these configurations, we now have a task defined in the joint space of the robot. Moving between tasks in joint space, any configuration that leads to collision with the surrounding geometry should be avoided. This mapping between geometric obstacles and corresponding joint space regions can be exemplified using a simple three degree of freedom robot arm.

Such a simplified 3DOF robot can be seen in Figure 3.2, where one meter long cylinders are used for its three links and smaller cylinders are used to show the rotation directions of its joints. Rectangular blocks in red, green and blue of dimensions one by one by two meters are used to represent the surrounding geometry. These simple geometric obstacles are used to exemplify how the state space changes when transforming from the TCP-space to the robot's joint space. The robot is then tasked to reach three different target points, $A$, $B$ and $C$, without colliding with the surrounding geometry.

Even a simple surrounding geometry of the tree rectangular blocks shown

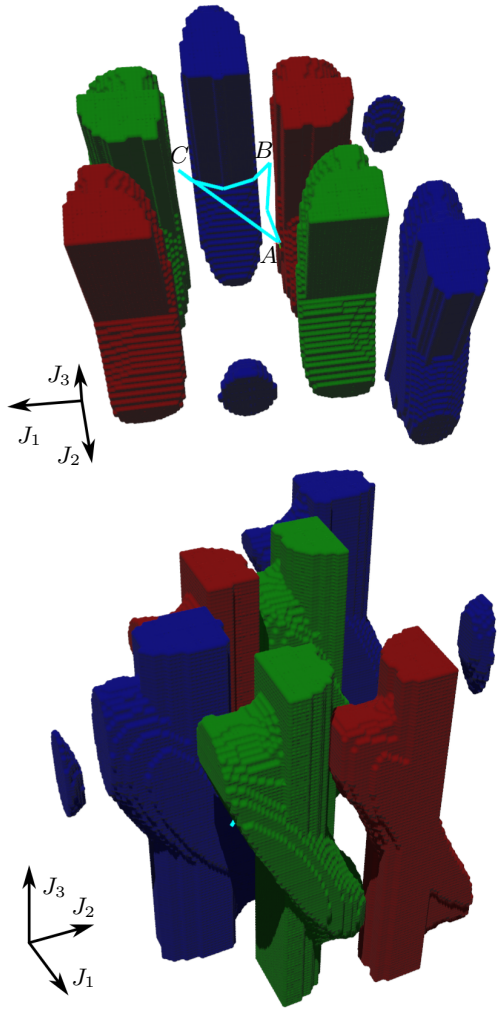



**Figure 3.3:** Three-dimensional joint space plot showing joint value combinations that lead to collisions with the surrounding geometry. The top and bottom figures show two different views of the same joint space plot, as indicated by the $J_1$, $J_2$ and $J_3$ coordinate directions. The collision free joint space trajectory between the points $A$, $B$ and $C$ is in the top figure shown in cyan. The trajectory, with inserted intermediate waypoints, is a piecewise linear path which curves around the blue and red obstacles.

in Figure 3.2 are morphed into unrecognizable geometries when viewed in the joint space of the robot, as is shown in Figure 3.3. The shape of the joint space obstacles seen in the figure is found by exhaustively sampling the full three-dimensional joint space of the robot, using a grid-based sampling with 100 sample points along each dimension. By setting the robot joint values to every possible combination of sampled values, and calculate if the robot is colliding with either of the three rectangular blocks of Figure 3.2, each point can be color coded accordingly. It can be noted firstly that these joint space obstacles are no longer convex, nor are they necessarily connected, and the repeated similar shapes are due to periodic similarities in the rotational space. For example, rotating the first joint by one half revolution gives a mirrored joint-space for the second and third joints. Secondly, even though the original rectangular blocks do not intersect, the joint space obstacles can do so, as this corresponds to combinations of joint values where the robot is colliding with more than one of the blocks.

Even these simple examples of rectangular blocks spaced quite far apart, hint at the complexity of solving a path planning problem in a cluttered environment. In the next section we will define the path planning problem more precisely and discuss a few solution methods used to find a collision free path.

## 3.2 Path planning

In the field of path planning and trajectory optimization, a path is what is mathematically more commonly referred to as a curve. A continuous path can intuitively be thought of as the trace left by a moving point. The function that defines the path is called the parametrization of the parametric path $x(s)$. This is an $n$-dimensional function of the one-dimensional $s$ defined on an interval $a \leq s \leq b$,

$$x(s) = [x_1(s), x_2(s), \ldots, x_n(s)] .$$

The description here focuses on continuous paths, and in robotics the paths will most often be continuous, especially as paths often are expressed in joint space coordinates of the robot. But one practical example of when a path might not be continuous, is when tracing the active Tool Center Point (TCP)

of a robot with multiple defined TCP:s. Performing a robot motion which includes TCP changes will produce a piecewise continuous path with discrete jumps at each switching point.

To solve a path planning problem, we want to find a collision free parametric path $x(s)$, expressed in the joint space of the robot, from a starting configuration to a final configuration. This means there are regions in the joint space the path must avoid, since the robot should at no point along the path collide with the surrounding geometry. The goal is to find the shortest possible path.

When the parametrized path $x(s)$ uses the time $t$ as the parametrization variable, we talk about a trajectory $x(t)$. The trajectory holds information, not only about which path in space will be followed, but also at what time each point along the path will be visited.

One classic class of graph traversal algorithms which can be useful in path finding methods, are variants and generalizations of Dijkstra's algorithm, such as the A*-algorithm. The problem then becomes how to build up the search graph in the first place. For problems of low dimensionality which can be exhaustively sampled with acceptable resolution, for example in two-dimensional path finding, these algorithms are commonly used together with grid-based sampling [47]. But for many applications, simple exhaustive grid-based sampling will often be too costly, especially as the dimensionality of the problem increases and the cost of performing a collision check with an increasingly complex surrounding geometry rises.

A class of methods used to sample the search space and build a search graph are referred to as probabilistic roadmap (PRM) methods. They first find waypoint candidates using random sampling, and then check if there are collision free straight lines between pairs of waypoints using a local planner. This builds up a graph problem which can be solved to try to link waypoints together from start to end configuration [19].

Another class of solution methods make use of so called rapidly exploring random trees (RRT) in their search. By branching out from one or more of the endpoints and choosing sampling direction and distance for each new sampling point tree-like graphs are constructed that efficiently cover the search space [48].

## 3.3 Trajectory planning

One way of going from a path to a trajectory is to use the exact same path and find a mapping $s(t)$ that will make the trajectory $x(s(t))$ fulfill any dynamic constraints such as velocity and acceleration constraints. This time stamping problem formulation is referred to as a velocity tuning problem since it is the velocities along the path that are modified. It might however be beneficial to modify the initial path to more easily be able to fulfill dynamic constraints. That there is a potential gain in modifying the initial path can easily be understood when considering that most of the path planning solution methods covered previously return a piecewise linear trajectory visiting a set of waypoints. If the path is not modified and we must follow this trajectory exactly, then the robot will have to come to a complete stop at each waypoint to not exceed a finite acceleration limit.

In Paper A the initial path is used as an initial feasible solution and the trajectory is found by solving the full optimization problem. The goal is to limit the trajectory as little as possible, trying to make use of the large search space to find a faster and shorter solution. Collision avoidance is handled by an iterative scheme that updates variable limits to pointwise fulfill the collision avoidance criteria. In Section 3.4 some illustrative examples of solving a trajectory planning problem in two dimensions are presented, and in for example Figure 3.4 we can see how even for very simple problem formulations, pointwise fulfillment of collision constraints might still cause problems and give an unrealizable trajectory if not handled carefully.

In Paper B and Paper C, a formulation matching robot controller trajectory parametrizations is used, with via-point zones used to smooth out the initial piecewise linear path. Paper E formulates the full problem of first finding an initial trajectory that adequately covers a surface and then optimizes the trajectory with respect to spray paint coverage.

## 3.4 Illustrative examples

In this section we will make use of a simple two-dimensional example to describe concepts connected to trajectory optimization, and to exemplify issues that can arise when trying to solve even a simplified problem.

When optimizing the robot trajectory and formulating it as a discrete op-

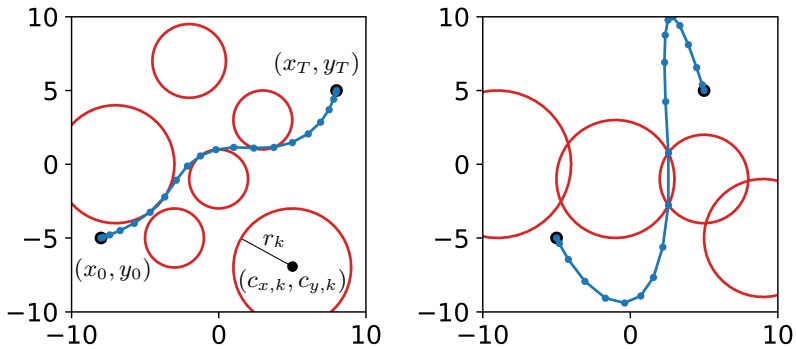



**Figure 3.4:** Solution trajectory reaching target points while avoiding red circles.

timization problem there are a number of different considerations that have to be made, and potential problems that need to be handled. One important issue is to define the discrete optimization problem in such a way that the solution to the problem describes a valid solution to the continuous problem. In order to exemplify a case where this might be a problem, a simple two-dimensional example problem is introduced. Despite its simplicity, when solving a few different instances of the problem we encounter a few different types of difficulties that commonly are encountered also in more complex problem formulations.

## Two-dimensional simple optimal control problem

A point mass system in two dimensions is an example that has the beneficial property that it is easy to visualize the problem as well as its solutions, while still being complex enough to exhibit a lot of the problems and characteristics of a more general problem formulation. It might be beneficial to consider a graphical representation of the problem shown in Figure 3.4 as we introduce the variables and states of the optimization problem.

This example problem is formulated as follows: A point mass system in two dimensions is controlled by an applied force. The time evolution of the point mass can be limited by lower and upper bounds on its position, velocity and acceleration in each dimension. In the examples in this section, we will only limit the position to a square and leave the velocities and accelerations

unbounded. What limits the movement instead, is an upper bound on the magnitude of the applied force at each time step. A collision avoidance constraint is also included in the problem formulation. By setting a lower bound on the distances to each point in a specified collection of points we introduce circular obstacles to be avoided at all times. The goal is to get from an initial configuration, $x_0$, to a target configuration, $x_T$, in the least amount of time. This problem has a lot of similarities with the second order integrator studied in Section 2.4. As was the case there, the cost functional of the standard optimal control formulation (2.8) simplifies to minimizing the final time as in (2.14a). The two configurations, $x_0$ and $x_T$, could in general have specified velocities and accelerations, but in these examples we will for simplicity only consider configurations where they both are at rest. The trajectory between these two points is discretized into a number of points with constant time difference.

The constants of the optimization problem are the number of discretization points, $n$, the midpoints of each circular obstacle, $c_k = [c_{x,k}; c_{y,k}]$, the radius of each obstacle, $r_k$, the mass of the particle, $m$, and the maximum control signal value, $u_{max}$. The notation used for potential variables to use in the optimization problem can be listed as follows,

$$
\begin{aligned}
q_i = [x_i; y_i] &= \text{The position of each discretization point,} \\
\dot{q}_i &= \text{Velocity at the midpoint of each discretization interval,} \\
\ddot{q}_i &= \text{Acceleration at each discretization point,} \\
u_i &= \text{Value of the force control at each discretization point,} \\
T &= \text{Duration of movement,}
\end{aligned}
$$

where index $i$ iterates over the number of discretization points, $i \in [1, n]$. Letting the velocities be defined at the midpoints of each interval is just for convenient use of the midpoint method for time derivatives.

Of these variables, the selected state variables are the positions and velocities at all discretization points, the duration of the movement is included as a free variable, while the control variables are the applied forces. With these variables, the dynamics of the system are described by the following time

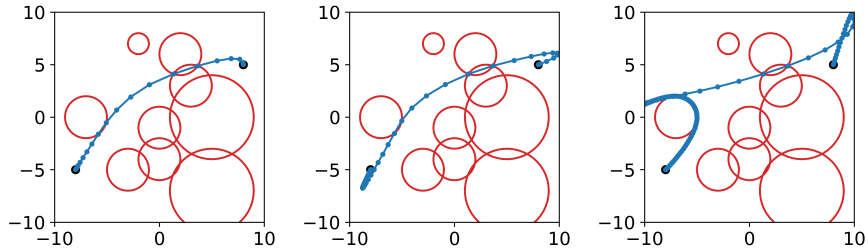



**Figure 3.5:** Jumping bounds despite problem being solvable. Increasing the number of discretization points from $n = 20$ to $n = 40$ and then to $n = 100$ does not solve the problem.

evolution of the state variables:

$$\begin{aligned} x_1 &= q_1, & \dot{x}_1 &= x_3, \\ x_2 &= q_2, & \dot{x}_2 &= x_4, \\ x_3 &= \dot{q}_1, & \dot{x}_3 &= u_1/m, \\ x_4 &= \dot{q}_2, & \dot{x}_4 &= u_2/m, \end{aligned}$$

where the mass, $m$, is set to unity.

Additionally, we need to include a few constraints in the optimization problem. The first set of constraints handles the collision avoidance of the circular obstacles, $r_k^2 - (q_i - c_k)^2 \leq 0$. And finally, we limit the maximum magnitude of the control variables, $u_i^2 - u_{max}^2 \leq 0$, so that the controlling forces always lie within a circle of radius $u_{max}$.

### Problem solutions

A typical problem instance and its corresponding solution is as previously stated shown in the left figure of Figure 3.4. The areas to avoid are shown as red circles and black dots mark the start and target point of the trajectory. The found solution navigates through the maze of obstacles and arrives at the target point as quickly as possible. The initial solution is as a first approximation chosen to be the linear interpolation between the target points. The number of discretization points along the trajectory are chosen to be $n = 20$.

### Jumping constraint bounds

Modifying the problem to be a bit more complex, the number of obstacles are increased and overlapping obstacles are introduced. In the right figure of Figure 3.4 these overlapping obstacles are placed in such a way that there is no valid solution to the continuous problem. Despite this, the solver produces a solution to the discretized version of the problem. The found solution shown in the figure has an obvious problem in that it is not physically realizable. The optimizer has exploited the discretization of the problem and spaced out the discretization points in such a way that all of the points fulfill the stated constraints. But the resulting trajectory jumps over a forbidden region between two of the discretization points. Notably, the solution even needs to back up and gather speed to be able to make the jump in one time step.

A similar example, which also includes constraints with overlapping circles, is seen in Figure 3.5. Ideally, the optimal solution should navigate through the upper region of the figure and as quickly as possible move through the maze-like structure. In particular it is worth noting that any solution of the continuous problem which successfully navigates the maze of obstacles, will necessarily be quite far from the straight line used as initial guess. This contrasts with the first example we saw in Figure 3.4, where the initial linear trajectory was a much better approximation of the final solution.

### Exploiting free end time

The fact that the discretized solution only is feasible if the discrete trajectory makes the jump over the forbidden region in a single time step hints at a probable remedy to the problem. If the number of discretization points are increased, it should get harder to clear the obstacle in a single jump and we might be able to force the optimizer into a physically feasible solution. This would leverage any globalization technique included in the solver to force it away from infeasible solutions. The problem with this strategy is that since the final time of the trajectory is a variable of the system, it can be increased as needed by the optimizer. Given enough room to speed up for the jump, a discretized trajectory, spanning the forbidden region in a single time step, can be found even as the number of discretization points are increased. Solutions for increasing number of discretization points can be seen in Figure 3.5. We can see how the solver always converges to the same type of infeasible solution,

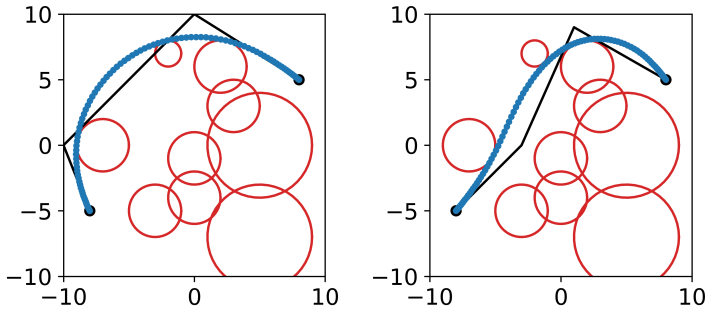



**Figure 3.6:** Finding a solution to the problem by using a new initial trajectory.

even taking the time to back up all the way to the boundary of the allowed area, turn around, and accelerate as quickly as possible towards the forbidden region in order to make the jump over the obstacle in a single discretization interval.

### Guiding the search using an initial solution

Often a more successful strategy of avoiding unwanted behavior of the obtained solution is to make use of the solver's sensitivity to the initial values of the optimization variables. By finding and supplying appropriate initial values to the solver, we can guide the search towards desirable search spaces where the solver will converge to a feasible solution. This is as true when solving one-dimensional root-finding problems using the Newton method as when solving large scale nonlinear optimization problems using state-of-the-art solvers. In Figure 3.6 a comparison between the solutions found using different initial guesses can be seen. The linear interpolation from start to end configuration ran into problems of jumping bounds. Modifying it to a piecewise linear initial guess, which is continuously feasible, the solver can be guided to a feasible discretized solution. This initial trajectory could in turn be the output of a path planning problem. The two solutions shown for the two different initial guesses also show that for these types of problems we typically cannot guarantee that we have found a global optimum. Instead, we often need to rely on the quality of the solution to the path planning problem, to successfully guide the subsequent local search for optimality.

## 3.5 Summary

This chapter has introduced and described aspects related to robot trajectory optimization. Section 3.1 aimed to shed some light on the connection between our regular three-dimensional space and the joint space coordinates of a robot. An example with a 3-DOF robot was used to show the surrounding geometry and a collision free trajectory in both spaces. In Section 3.2 the path planning problem was described, and a few examples were given of solution algorithms. In Section 3.3 we go from optimizing a path parametrized in space to optimizing a trajectory as a function of time. It is described how trajectory optimization is used in the included papers. Section 3.4 introduces a simple two-dimensional problem to exemplify issues that might arise when trying to solve a discretized trajectory optimization problem.

The next chapter will focus on robot trajectory parametrizations and on how the robot trajectory is calculated when sending robot commands to a robot controller from a specific manufacturer.

# CHAPTER 4

# Robot controller emulation

We have in Chapter 3 seen examples of how robot trajectory optimization problems can be formulated and solved, along with example cases highlighting potential difficulties encountered when searching for a physically realizable solution. It has been stated that we want to find a solution to the problem expressed in terms of parameters found in robot code, so that the solution can be directly used to generate executable robot code. In this chapter we will explore what these parameters are and how we can approximate the executed trajectory for the robot, given a set of robot commands. Determining the trajectory from the robot commands is the task of the robot controller.

To accurately be able to predict how the robot will move when executing a robot program is important for a number of different reasons. It is important to make sure that the robot performs its assigned tasks as expected. This often corresponds to ensuring that the target points of the robot are correctly expressed. If we instead look at simulating the robot motion between tasks, perhaps the most common reason to do so, is to make sure the given robot program will not make the robot collide with surrounding geometry or other robots while moving between its specified tasks.

Complex robot stations with low specified clearance will necessarily have

a high accuracy requirement for calculating the robot trajectory. There will however always be a discrepancy between the virtual emulator and the real-world robot execution of the program. For example in ABB:s product specification [49] of their controller software IRC5, in the section covering their Absolute Accuracy calibration concept, they state that

*"The difference between an ideal robot and a real robot can typically be up to 10 mm, resulting from mechanical tolerances and deflection in the robot structure."*

To handle this problem, they offer different options of individually calibrating a specific robot and its control system to improve this accuracy.

When it comes to collision avoidance for static geometry, it is enough to consider the path of the robot in the spatial domain. If we also want to consider dynamic obstacles such as other robots, the time evolution of the trajectory would also need to be considered. One way to handle these obstacles could be to include blocking signals to ensure the robots will not collide, even if one of them suddenly malfunctions and halts their execution.

There are other reasons where it would be important to consider the time evolution of the robot trajectory. The reasons could be quite general, with a focus on total execution time and an interest in fulfilling cycle time criteria. They could also be more specific, where it is important to determine where the TCP is at any given time, for example in cases with time-dependent processes, such as spray painting or when applying a bead of sealant material.

When manually teaching a new robot path, the operator can run the robot in a reduced mode to ensure the program runs as expected. When the robot program is generated offline and uploaded to the robot or sent through a remote connection, it is for safety reasons recommended to first dry run the program in a virtual robot controller emulator. All major robot manufacturers provide such a virtual tool. Examples include RobotStudio [50] from ABB, KUKA.Sim [51] from KUKA and URSim [52] from Universal Robots. These emulators from robot manufacturers are of course a helpful tool for a robot operator, and a lot of robot programs are prepared using the available tools within a virtual simulator like this. For someone interested in performing standalone offline robot programming, these virtual environments often include a brand specific interface, or can be accessed through RSS, which is a general-purpose robot interface.

The industry standard Realistic Robot Simulation (RRS) enables offline

programming software to interface with a propriety Robot Controller Software (RCS) module of a robot manufacturer. This makes it possible to send robot commands to the RCS module and in return get an accurate simulation including cycle time estimates. This interface supports useful functionality, and in the Industrial Path Solutions (IPS) software, there is similar functionality available with respect to ABB robots, using their virtual robot controller VRC/RobotStudio interface. Including an RRS compatible interface in the IPS software could be a good way to increase the number of supported robot manufacturers. This would provide a general way for an end user to simulate the trajectories for a given set of robot instructions, given that they have access to the RCS module of the robot manufacturer.

## 4.1 Controller parameter identification

When performing offline robot trajectory optimization as in Paper B and Paper C, with the goal of generating executable robot code, it is desirable to have an accurate description of what happens within a controller from a robot manufacturer, and not only treat it as a black-box. An accurate mathematical description of the controller enables us to derive analytical derivatives of the controller, which can be supplied to the solver to improve the convergence and decrease the number of necessary function evaluations. This was the focus of Paper D, to accurately simulate robot trajectories from different robot manufacturers. The controller model also allows us to quickly evaluate the trajectory, while the controllers from robot manufacturers are more focused on simulating at, or near, real time evaluation. However, it is not always straight forward to derive what happens internally in a proprietary robot controller. Even if the robot code between manufacturers share a lot of functionality, different manufacturers vary in how they calculate and interpolate the resulting trajectory.

The problem a robot controller is implemented to solve can be generally stated, even if there are differences in how robot controllers from different manufacturers implement their controllers. When a robot controller is given a set of robot commands to execute, it should calculate the resulting trajectory and determine what control values to apply to the robot. These are the power levels of the servo motors which affect the actuator torques. Using a set look-ahead, the controller determines what torque to apply at the current time for

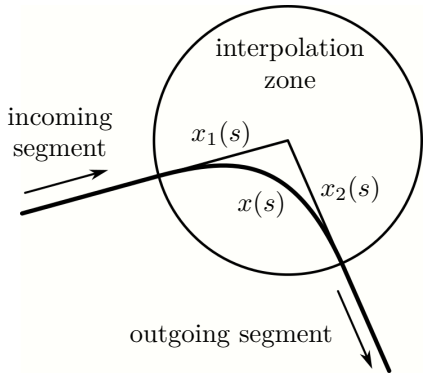



**Figure 4.1:** Sketch of the interpolation zone of a trajectory waypoint with linear incoming and outgoing segments. The interpolated trajectory $x(s)$ within the zone is calculated from $x_1(s)$ and $x_2$. These describe the path the robot would follow if there is no zone interpolation, and the waypoint is reached exactly.

each joint, and as the robot starts moving, it will monitor sensor values to make sure the robot closely follows the calculated path and shut down if it encounters any deviations.

There is a clear motivation for developing a lightweight and generic robot controller, that can emulate controllers from different manufacturers. This generic controller emulator will be referred to as *Controller Light*, and this chapter describes the main steps involved in implementing it.

When developing Controller Light, an important step in the process has been to use model fitting, to get manufacturer specific parametrized robot trajectories. This step consists of selecting a few probable parametrizations, looking at combinations of robot commands, and performing physical testing and registering sampled joint values along the trajectory, especially within the zones or blending areas. Test trajectories have been formulated where the angle between incoming and outgoing segments is varied, which makes it possible to find parameter values that show a good fit for a wide range of possible robot command combinations. The sampled physical trajectories are in a parameter estimation step compared to parametrized analytical trajectories. The parameters are optimized to achieve a close correspondence to the physical trajectories for the full range of test trajectories.

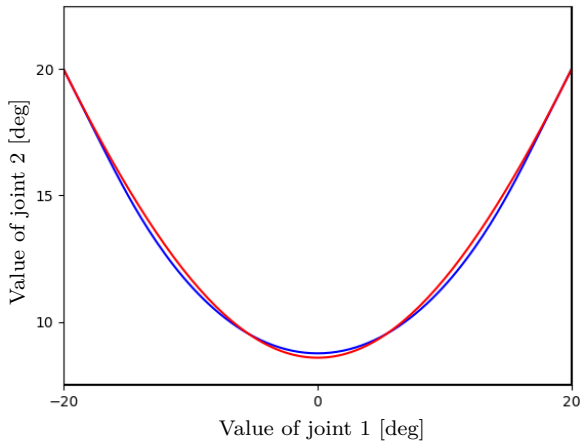



**Figure 4.2:** A single test trajectory with a zoomed in section highlighting a part of the blending zone and the type of deviations that might occur between the calculated (blue) and the sampled (red) trajectory.

As an example of a zone interpolation, we can take the following expression from Paper D using polynomials for $\alpha(s)$ and $\beta(s)$

$$x(s) = x_1(s) + \alpha(s)(x_2(s) - x_1(s)) + \beta(s)u,$$

to blend an incoming segment $x_1(s)$ with an outgoing segment $x_2(s)$, see Figure 4.1. The selection of the type of expression used, and the parameter optimization needed to tune the parametric models to specific robot manufacturers, will determine how accurately the resulting trajectory can be modeled.

In Paper D good agreement of the analytical expression to robot trajectories generated by controllers from different manufacturers was achieved. In Figure 4.2 we can see a comparison between a trajectory from Controller Light and a physically sampled trajectory. The chosen parametrization follows the trajectory as closely as possible given the chosen order of the polynomials. The differences that can be seen between the two trajectories in the figure are a consequence of higher order terms not being fully captured.

## 4.2 Time stamping

Calculating an accurate robot path in space is important, not only to ensure a collision free trajectory but also in continuous processes such as spray painting, where the trace followed by the end effector directly affects the process. The parameter identification described in Section 4.1 focuses on obtaining an accurate spatial representation of the robot path. A full representation of the robot trajectory also requires accurate time stamps of each sample point along the robot path. Accurate trajectory times are essential for most applications as it gives cycle time estimations, and a good prediction of when each point of the path is reached is important for continuous processes.

The path should be time stamped in such a way that all limits on time derivatives are fulfilled. These include TCP-velocity, joint space velocity and joint space acceleration. All of these limitations are typically stated either in the data sheet of the robot or specified in the robot code to apply to a specific segment of the trajectory.

The velocity constraints can be fulfilled locally by considering each pair of consecutive sample points and selecting a time duration between them which will fulfill also the most constraining of constraints. The joint acceleration constraints use a local calculation for three consecutive points, where the second time interval can be determined in relation to the first. This is done by matching the three points to a quadratic polynomial, $f(t) = At^2 + Bt + C$, where the constants $A$, $B$ and $C$ are calculated from the acceleration limits.

Making use of the fact that the robot is at rest at the start and at the end of the trajectory, the local expression can be used to sweep first forward through all samples and modifying time stamps so that all accelerations are fulfilled, and then again backwards to fulfill all decelerations. These sweeps are exemplified in Figure 4.3 and the main benefit of this method is that a time stamped trajectory can be quickly calculated, without the need of formulating an optimization problem to find the value of each time stamp. It should be noted that in general, with more complex constraints on the trajectory it might be necessary to formulate a full optimization problem. The time stamping calculations are independent for each joint of the robot along with any controllable external axes and are as previously mentioned combined by selecting the most limiting constraint. Focusing on the simplified one-dimensional case shown in Figure 4.3, we can see how the time stamps of the trajectory are shifted forward in time to fulfill the acceleration constraints,

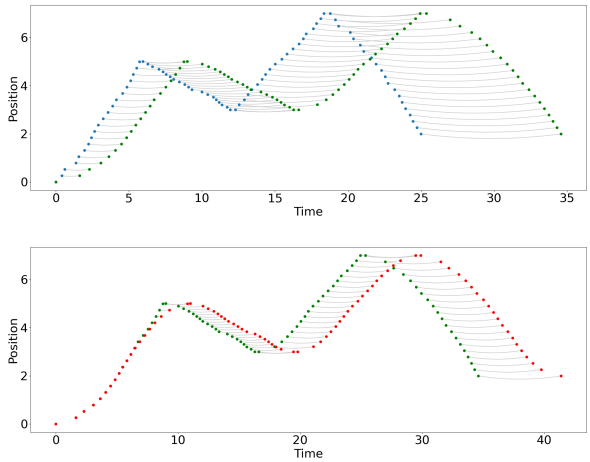



**Figure 4.3:** Exemplifying how running a time stamping algorithm over any given trajectory can quickly generate a trajectory that fulfills velocity and acceleration bounds. The top figure shows in blue dots a one-dimensional trajectory going back and forward with non-uniform times between samples. The green dots represent the updated samples after a single sweep of the algorithm. The lower figure represents the backward sweep, where the red dots show the final trajectory after again being shifted forward to account for all decelerations.

and the final trajectory is smooth and gently comes to a standstill at the endpoints as well as the intermediate switching points.

## 4.3 Controller Light

Controller Light is not a real time controller, in that it is not primarily used to calculate instantaneous control values for the robot. It is rather a simulation tool that can be used to quickly calculate a nominal path that closely follows the actual path of the robot.

A lightweight robot controller simulator is also useful in cases where we do not have a specific robot in mind, as we perform some optimization of an external process, such as spray painting. It is useful both to get an approximation of a generic robot path or select a specific robot manufacturer to get

a more precise representation.

One limitation of the structure of Controller Light is that the path is speed independent since the trajectory is first sampled in space, and then time stamped. This is often true and helpful for operators that want to test run solutions at lower speeds, however speed dependent blending could in some cases be more efficient and there are examples of robot manufacturers that have implemented this. It is also necessary to have a speed dependent trajectory in cases where the robot movement should be synchronized with an external moving object such as a conveyor.

## Types of movement commands

The three different types of movement commands seen in Figure 4.4 can be handled by Controller Light and represent movement primitives commonly used by different robot manufacturers in a similar way. They represent linear movement in joint space, linear movement in TCP space and circular movement in TCP space, where the shorthand notation MoveJ, MoveL and MoveC follows the ABB naming convention.

Moving the robots joints linearly in joint space, MoveJ, is a highly efficient movement. This movement type can be used to quickly move between target points in cases where the movement of the TCP is unimportant. The limiting factors of such a movement will relate to physical limits of the motors driving each joint. An example of a linear joint space movement was described in Section 3.1 where the solution to the path planning problem is the piecewise linear trajectory shown in the joint space plot of the robot workspace.

Specifying a movement in joint space is also necessary when moving between different inverse configuration spaces or when moving close to kinematic singularities.

A linear movement in TCP-space, MoveL, is used where it is more important to keep track of the position and orientation of the robot tool. This is a commonly used movement type in industrial applications, perhaps even overly so. When programming a robot, the main focus is on positioning the tool of the robot and it can therefore feel quite natural to program the robot with respect to the position and orientation of the tool, and to move between specified positions in straight lines. Especially since it can be hard to get an intuitive feeling about how the robot will move in space as it moves in joint space between different configurations. Linear TCP movement could also be

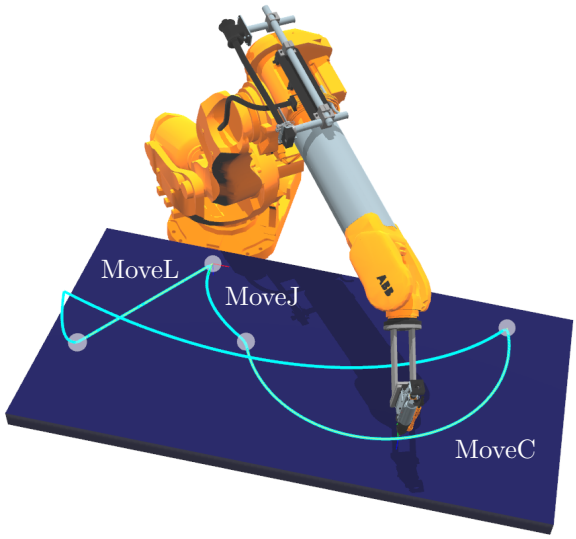



**Figure 4.4:** Robot performing a simple trajectory involving three different types of movement commands.

a useful way to find a collision free trajectory when for example moving close to a work piece and the biggest collision risk comes from the tool colliding with the workpiece. Additionally, if the robot is holding a heavy tool, it could be an efficient strategy to move the tool the shortest possible distance.

A circular movement in TCP space, MoveC, can be used in curve following processes to get the robot to follow a smooth trajectory curve using fewer robot commands compared to trying to follow the curve using MoveL.

All of these three movement types are used by most robot venders and a simple point-to-point trajectory using one of these commands will typically look very similar irrespective of the type of robot model used. Despite this seemingly generic way of expressing movement commands, there will be differences in the resulting trajectory for different robot manufacturers as we string these movement commands together. Looking at what happens whenever two of these movement commands are blended together, there is not a single unique solution for how the transition should be made, for example between a MoveC and the MoveJ command.

### External axis and static TCP

External axes include additional actuators that can move synchronously with the robot. These could be controlled by the robot controller or by the programmable logic controller (PLC) of the robot station. Examples include a linear track that adds a translational degree of freedom to the base of the robot, increasing its available workspace. Or a work-piece turn-table that adds a rotational degree of freedom to the work-piece, to improve reachability. In the controller these axes are handled in the same way as when the robot is controlled with MoveJ commands. All external axis are calculated using linear interpolation, before any inverse kinematics (IK) calculations are performed to determine the desired position of the robot TCP.

The most common setup for an industrial robot is to have the TCP defined as a point on a tool which is mounted on the face plate of the robot. There are however cases where a bulky tool such as a spot-welding gun or a precise measurement device is mounted in a fixed location. The TCP is still defined as a point on the tool and from the robot's perspective this is an external TCP and the robot instead manipulates the work-piece. In this case the inverse kinematics calculations close the kinematic loop just as in the standard setup, the only difference is the choice of coordinate system and the order in which interpolations are calculated.

## 4.4 Zone size maximization

With an accurate representation of the robot trajectory for a given set of robot commands, it is possible to explore how changes in the parameters of the robot code affect the trajectory. We have seen examples of optimizing multiple robot code parameters simultaneously in Paper B and Paper C. A more narrow focus on optimizing only a subset of these parameters, would simplify the optimization scheme. In particular, if we only consider the sizes of the waypoint zones we can see that the initial piece-wise linear trajectory found by the path planner will have a zone size of zero, and to obtain a smooth and fast trajectory we want to increase the zone sizes as much as possible. This would both reduce peak accelerations and shorten the length of the trajectory. This is the basis of the zone size maximization algorithm implemented in IPS using Controller Light.

The algorithm works by looking at each waypoint, and maximizing its cor-

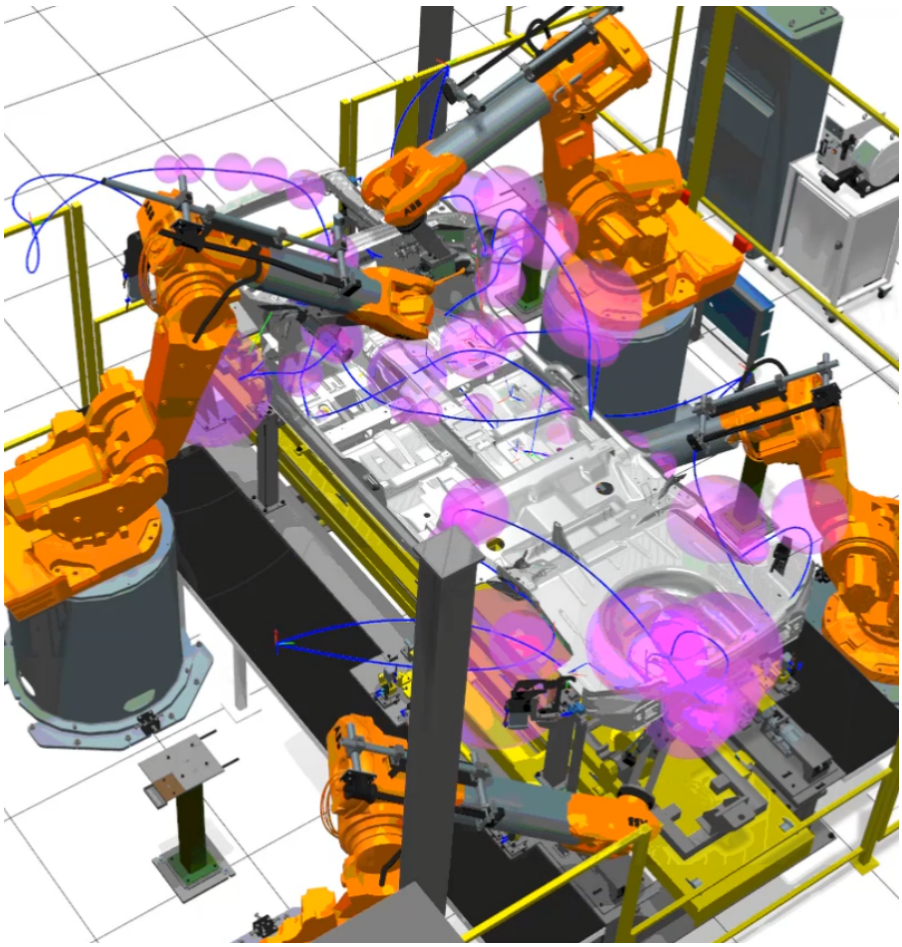


**Figure 4.5:** Four robots in a robot cell executing their robot programs in parallel. The TCP trace of each robot is shown in blue, with the individually maximized waypoint zones represented as spheres. CAD model of workpiece and robot cell courtesy of Volvo Car Corporation.

responding zone size using the bisection method, while checking the resulting trajectory for collisions with the surrounding geometry. The resulting robot programs produced by running the zone size maximization algorithm on all robots in a robot cell are shown in Figure 4.5. The figure shows the TCP trajectory of each of the four robots in blue. Any waypoint a robot passes when moving between tasks is shown as a purple interpolation sphere. There is a wide range of zone sizes seen in the figure. The size of the spheres are limited by either the proximity to the previous or next waypoint, or that increasing the zone size further will lead to a collision.

Since the initial path is known to be collision free, it is only possible to collide within the interpolation spheres. This makes it possible to check for collision more efficiently by only sampling the trajectory within the sphere we are currently trying to maximize. The efficient bisection method uses interval halving to further reduce the number of computationally expensive collision checks.

## 4.5 Summary

This chapters has covered aspects related to robot controller emulation. Section 4.1 described how parameter identification is used to ensure the emulated robot path closely follows the physical robot path for a range of combinations of movement commands. The path is converted to a trajectory using an efficient time stamping algorithm described in Section 4.2. In Section 4.3 we see how this has been packaged within the IPS software as Controller Light, and are presented a number of different use cases that controller is able to handle. Finally in Section 4.4 we see an example of how the implemented controller is used in the zone size maximization algorithm. This is an algorithm which is available to commercial users of the IPS software in industry.

The next chapter covers robot trajectory optimization in the context of robotic spray painting. We will see how the spray painting process is described and modeled with different levels of complexity and accuracy.

# CHAPTER 5

# Spray painting

In previous chapters the goal has been to find optimal robot trajectories between well-defined and precisely specified tasks, even if the tasks themselves have a certain number of parameters to vary. We have discussed spot welding tasks where the target point is set in space, as well as applying a bead of sealant material by following a specified parametric path in space.

For robotic spray painting, exemplified by Figure 5.1, the task is specified in terms of the desired result, an even paint thickness. But from the point of view of the robot it is not necessarily clear what motion the robot should perform to achieve this desired result. One way to specify the robot task requires good knowledge of the painting process, and to know that if we move the paint applicator along a specified path, we have empirically found this to produce an even paint thickness. If we are able to simulate the paint deposition onto the geometry, it would be possible to verify candidate trajectories, to determine if they produce an adequate paint coverage with respect to paint thickness, transfer efficiency, and cycle time.

The fact that the tasks in some sense have a higher dimensionality also affects the design of painting robots. The robot's kinematic structure can be different compared to a typical 6-DOF industrial robot arm, with extra joints

**Figure 5.1:** An example of an automated painting booth, where robots are painting parts of a Volvo truck suspended on a rack. Image courtesy of AB Volvo.

added for increased workspace flexibility. Another example can be seen in Figure 5.1, where the painting robots have an extra dependent joint which makes it possible to run paint tubes through the hollow wrist of the robot. There are also examples of more experimental designs tailored to spray painting applications [53].

In the following sections some aspects and considerations of paint deposition modeling will be brought up, with Section 5.1 describing a multi-physics simulation while Section 5.2 describes how this paint deposition model can be simplified in order to be included in a painting trajectory optimization routine. Section 5.3 shows some results from Paper E where an optimized applicator

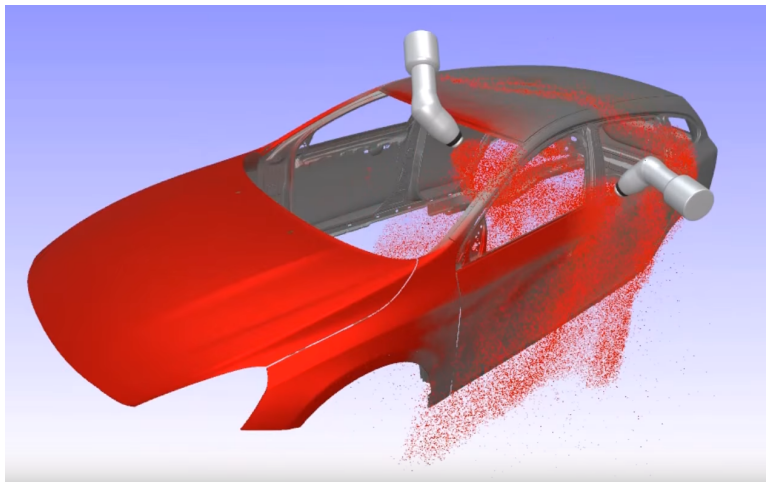


**Figure 5.2:** Simulation of two robots spray painting a Volvo car body. The snapshot shows the simulated positions of each droplet in the paint cloud. CAD model courtesy of Volvo Car Corporation.

trajectory is found using the described paint projection method. Section 5.4 covers aspects related to the obtained solution and how the found applicator trajectory can be physically realized.

## 5.1 Spray painting multi-physics simulation

In order to realistically simulate the paint deposition of an industrial spray painting process, the different physical processes that affect the paint droplets all need to be accurately modeled.

To exemplify these processes, we can consider an electrostatically charged rotary bell paint applicator. This is a commonly used setup for a robotic paint booth in the automotive industry. The first physical process to consider here is the atomization of the paint into tiny droplets at the edge of the quickly spinning rotary bell. Each paint droplet then has to be simulated forward in time while considering how the air-flow and the electric field affect the charged paint droplets. By coupling the simulations for air-flows, electrostatic fields and charged paint particles, each paint particle can be traced from when it is

ejected out of the applicator until it collides with the target geometry. The applied electric potential pulls the paint droplets towards the desired surfaces, thereby improving the transfer efficiency of paint. It also allows paint to swirl around corners and land on points that are not within a direct line of sight of the applicator. This can have a positive effect on the consistency of the paint thickness, but the increased electrostatic attraction of edges and protrusions can also create difficulties when trying to apply an even amount of paint on the whole surface. Integrating the contribution of the droplet landing on the surface produces a paint thickness estimate for each triangle of the target geometry to be painted [54]–[57]. A snapshot of a multi-physics simulation of each paint droplet can be seen in Figure 5.2, where each rotary bell applicator follows a specified trajectory.

## 5.2 Projection method for paint deposition

A calibrated multi-physics simulation can produce a highly accurate approximation of the resulting paint thickness. It is a resource intensive simulation, which can be invoked in the validation step of finding an adequate painting trajectory, or when comparing a limited set of candidate trajectories. For parameters where the relation is well understood of how they affect the resulting paint thickness, the simulation result can also be used to perform a local optimization over these parameters. This is for example the case for the speed of the applicator and the paint flow rate, which both have an approximately linear effect on the paint thickness. Speed optimization of a given trajectory makes use of this by performing a full multi-physics simulation for the trajectory and modifying the speed along each trajectory segment to obtain a more even paint thickness.

In cases where it is desirable to modify a larger set of variables, it soon becomes infeasible to directly include results from the multi-physics simulation in the optimization. Here it would instead be beneficial to use a simplified paint deposition model [58] to be able to optimize the paint thickness over the surface mesh or point cloud representation [59]. In Paper E, which is an extended and improved journal version of the work presented in an earlier conference submission [60], painting trajectories are optimized by making use of a paint projection deposition model. The main features of the projection model can be seen in Figure 5.3. The paint applicator is modeled as a point

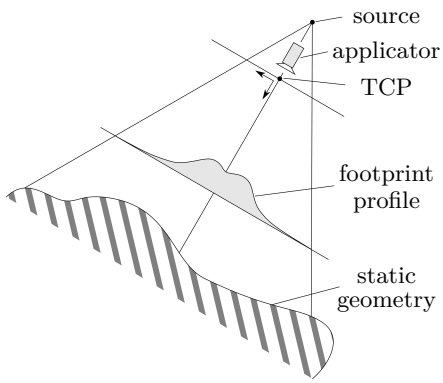



**Figure 5.3:** Representation of the main components of the paint projection. The modeled point source determines how the footprint profile is projected onto the static geometry. The figure also shows a TCP-centered coordinate system relative to the applicator.

source which projects a radially symmetric footprint profile down onto the geometry to be painted.

In one of the figures of Paper E, Figure 6, we can see a representation of a footprint profile, and how it is calibrated to physical experiments. The experimental setup involves sweeping the applicator over flat surfaces at different distances and measuring the resulting thickness profile in lines orthogonal to the sweep direction. The paint deposition model is formulated as an analytical expression of radially symmetric spline functions. By projecting this so-called brush function back onto a geometry, the resulting paint thickness can be analytically calculated. As the model is to be used in an optimization routine, it is important that all partial derivatives of the analytical expression can be calculated with respect to each optimization variable.

## 5.3 Applicator trajectory optimization

Since the projection method treats the paint deposition process as a pure projection of paint onto the geometry, it can be expected to perform well in situations where this is a sufficiently accurate representation of the process. This is true for low curvature surfaces, which are similar to the flat plates

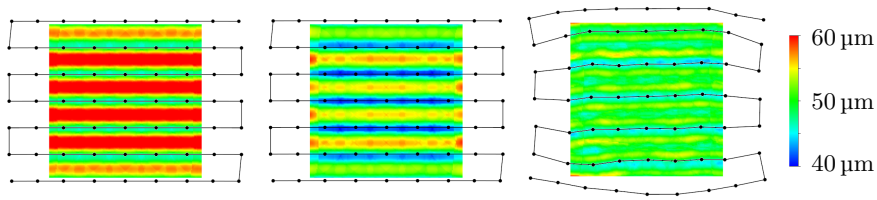
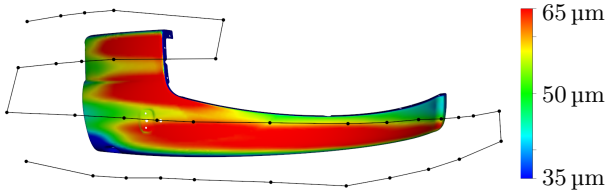


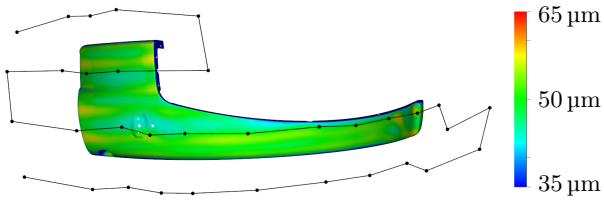

**Figure 5.4:** Test case of a square plate with a side length of 1.0 m, an initial path with a sweep width of 180 mm, and a target thickness of 50 µm. The figures show from left to right the resulting simulated paint thickness for the initial trajectory with a constant velocity, the thickness after applying a speed optimization but leaving the path unchanged, and after performing a full optimization where the trajectory points are also allowed to move.

used to calibrate the method. But for cases with high curvature, or along edges, this method will not fully capture the resulting painting pattern arising from inhomogeneous airflow and electric field. The optimization problem formulation and initial path generation algorithms presented in Paper E are formulated with this limitation in mind. The goal is to find a formulation which optimizes trajectories in a range of values where the model gives a good approximation of the paint deposition process. Specifically, the optimization routine should not be able to exploit a weakness in the paint deposition model to push the trajectory into a physically infeasible solution.

The results from the presented painting trajectory optimization, along with initial trajectory generation algorithms, are promising and show how an initial trajectory can be first generated, and then improved upon in the optimization step, to provide an adequate paint result in terms of thickness and consistency. Figures 5.4 and 5.5 taken from Paper E show examples of optimized trajectories. In Figure 5.4 it is clear that speed optimization by itself can improve the resulting paint thickness, but even in this simple two-dimensional case the modification of trajectory waypoints is a clear improvement over the speed optimized trajectory. For the industrial case shown in Figure 5.5 the simulated results are again promising. The two figures show how the optimization is able to take the automatically generated trajectory seen in Figure 5.5a and smooth out the projected profile over the surface by modifying the waypoints and obtaining the even paint result seen in Figure 5.5b.




**(a)** Initial path with constant velocity.




**(b)** Optimized trajectory.

**Figure 5.5:** The figures show the simulated paint thickness on a tractor fender that is obtained when using the same paint projection as is used in the optimization. The results are for an initial path with constant velocity, and an optimized trajectory, where the target thickness is 50 µm. CAD model courtesy of John Deere.

## 5.4 Spray painting trajectory

In Chapter 4, where robot controller trajectories are covered in detail, the trajectory is linked to, and affected by, the particular robot performing the task. The painting trajectories optimized in Paper E are formulated in more generic terms, only capturing the movement of the applicator or TCP of the robot. The path of the applicator is defined by a number of waypoints that specify position and orientation of the applicator. These waypoints are here used as optimization variables, but depending on application, it might be possible to reduce the number of decision variables to a smaller set of trajectory generating parameters [61]–[63]. Even after finding a solution for the TCP trajectory, there might be issues encountered when trying to physically realize this trajectory. It might not be possible to place the workpiece relative to the robot in such a way that the whole trajectory is reachable and within the robot's workspace. Even if the robot can reach each waypoint of the trajectory, it

might not be possible to continuously connect them all using the same inverse configuration. The kinematic singularity that would be reached when trying to move between different inverse configurations, could also cause problems by just being in the vicinity of the trajectory. This type of reorientation during a robot motion typically requires large velocity and acceleration values for a number of limiting joints. This could in turn produce a motion where the required target TCP velocity is not obtained, which will affect the calculated paint thickness. If this deviation is too large, it could be necessary to brake up the trajectory in multiple parts or perhaps rerun the optimization with additional constraints.

Physical limitations of the robot could also be included in the optimization to directly handle these issues even in the problem formulation step and Chapter 7 covers ongoing work related to this type of formulation.

## 5.5 Summary

This chapter has described the modeling and simulation of a spray painting process, with the stated goal of including the model in trajectory optimization for robotic spray painting. In Section 5.1 the multi-physics simulation of paint deposition, and the physical phenomena it aims to simulate, were described in general terms. Section 5.2 focused on the simplified projection method for paint deposition, which has been used in trajectory optimization for a paint applicator, as discussed in Section 5.3. The results from the optimization were exemplified by included results from Paper E. The final section, Section 5.4, compared the resulting applicator trajectories to robot trajectories, with a focus on differences that might result in executability problems.

The next chapter contains summaries of the included papers in the thesis, followed by conclusions and future work in the final chapter.

# CHAPTER 6

# Summary of included papers

This chapter provides a summary of the included papers.

## 6.1 Paper A

Staffan Björkenstam, **Daniel Gleeson**, Robert Bohlin, Johan S. Carlson, and Bengt Lennartson
Energy Efficient and Collision Free Motion of Industrial Robots using Optimal Control
Published in *Proceedings of the 2013 IEEE International Conference on Automation Science and Engineering (CASE)*, Madison, Wisconsin, USA, pp. 510–515, 2013.

DOI: 10.1109/CoASE.2013.6654025.

The paper covers trajectory optimization for an industrial robot, which is fully controllable on all joints, with respect to both cycle time and energy norm, while maintaining a collision free trajectory. Collision avoidance along the trajectory, even in highly cluttered environments, is achieved through an

iterative process based on a local sensitivity analysis. Optimal solutions are found by discretizing an optimal control formulation of the problem, and finding solutions using a general-purpose interior point solver. Results from an industrial case with two robots in an automotive stud welding station, show that the optimized smoothed trajectory cut down the cycle travel time by 19.2%, while at the same time decreasing the energy consumption by 12.8%, compared with the velocity tuned piecewise linear solution.

The work presented in the paper can be seen as a continuation of the Master thesis work of Daniel Gleeson (DG), supervised by Staffan Björkenstam (SB), about optimizing the trajectories of human model manikins. In this paper similar methods are used and adapted to the context of the relatively lower dimensional problem of industrial robots. The method, model and optimization routine were developed by DG and SB. Robert Bohlin (RB) provided valuable input regarding trajectory optimization and developed the path planning algorithm used to find initial paths. Johan S. Carlson (JSC) and Bengt Lennartson (BL) are PhD and Master's thesis head supervisor and examiner, and have provided guidance and input. All authors participated in reviewing and revising the manuscript.

*Context:* Sections 2.4, 3.1, 3.2, 3.3 and 3.4.

## 6.2 Paper B

**Daniel Gleeson**, Staffan Björkenstam, Robert Bohlin, Johan S. Carlson, and Bengt Lennartson
Optimizing Robot Trajectories for Automatic Robot Code Generation
Published in *Proceedings of the 2015 IEEE International Conference on Automation Science and Engineering (CASE)*, Gothenburg, Sweden, pp. 495–500, 2015.

DOI: 10.1109/CoASE.2015.7294128.

In the work presented in Paper A, the robot trajectory optimization considered a robot with double integrator joint dynamics, which is a generalized and non-specific parametrization of a robot trajectory. To realize such a trajectory in practice, requires exact control of the robot trajectory, using an interface to send sampled trajectory values directly to the robot. In real world industrial cases, robots are instead most commonly controlled by sending robot com-

mands to be executed, leaving the exact trajectory to be followed up to the internal robot controller. In this work the trajectory is optimized using such robot code parameters as optimization variables, making it possible to export the found solution to executable robot code. Results show that in an industrial test case with two robots, optimized zone sizes reduce the cycle time by 21,8% compared to the initial piecewise linear trajectory. By optimizing the trajectory zones a faster solution is found even when compared to a manually fine-tuned solution with zone sizes selected from a predefined set of Robot Studio zone sizes.

DG and SB jointly formulated the optimization problem and the iterative procedure to maintain a collision free trajectory. RB provided valuable input regarding distance measure calculations. JSC was involved in formulating the problem and together with BL guided the work and provided input on how it was presented. All authors participated in reviewing and revising the manuscript.

*Context:* Sections 2.4, 3.1, 3.2 and 4.3.

## 6.3 Paper C

**Daniel Gleeson**, Staffan Björkenstam, Robert Bohlin, Johan S. Carlson, and Bengt Lennartson
Towards Energy Optimization using Trajectory Smoothing and Automatic Code Generation for Robotic Assembly
Published in *Proceedings of the 6th CIRP Conference on Assembly Technologies and Systems (CATS), Procedia CIRP*, Gothenburg, Sweden, Volume 44, pp. 341–346, 2016.

DOI: 10.1016/j.procir.2016.02.099.

This paper can be seen as a continuation and refinement of the work presented in Paper B. Here the parametrization of the robot controller trajectory is more explicitly stated, to simplify the formulation and calculations of the resulting trajectory and all partial derivatives with respect to the robot code parameters. The trajectory optimization problem is formulated using additional robot code parameters as optimization variables. The resulting trajectory is compared to velocity tuned solutions using measures of both cycle time and energy consumption approximations. Results show that by modi-

fying waypoints and zone interpolation variables, a faster and more energy efficient solution can be found. The cycle time is reduced by 14.3% and an energy reduction of up to 10% on the load bearing links of the robot.

DG formulated the new parametrization of the robot trajectory. DG and SB reformulated and improved the discretized optimization problem. RB provided valuable input regarding parametric trajectory optimization. JSC and BL guided the work and provided input on how it is presented. All authors participated in reviewing and revising the manuscript.

*Context:* Sections 2.4, 3.1, 3.2 and 4.3.

## 6.4 Paper D

**Daniel Gleeson**, Christian Larsen, Johan S. Carlson, and Bengt Lennartson
Implementation of a Rapidly Executing Robot Controller
Published in *Proceedings of the 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, Vancouver, BC, Canada, pp. 1341–1346, 2019.

DOI: 10.1109/COASE.2019.8843254.

The work presented here aims to emulate robot controllers from different robot manufacturers and find parametrizations of the robot trajectories that can be quickly evaluated given a set of robot commands. One important motivation and benefit with this type of formulation is that it enables robots from multiple manufacturers to be used in the work presented in Paper B and Paper C, where robot trajectories are optimized in a cluttered environment using robot code parameters as optimization variables. A fast and accurate robot controller is formulated and implemented. It first analytically generates a spatial trajectory from robot commands based on manufacturer-specific parameters found by solving a parameter identification optimization problem. The trajectory is time stamped to fulfill velocity and acceleration bounds using a double-sweep algorithm, which finds the optimal time stamps without explicitly formulating an optimization problem. Results show good agreement with experimental values, with maximum trajectory deviations of less than 0.1° for the two industrial robots ABB 6640 and the KUKA KR 30-3, and a somewhat larger maximum deviation of 0.3° for the smaller KUKA IIWA

robot.

DG and Christian Larsen (CL) performed robot trajectory measurements, with DG focusing on the KUKA KR 30-3 and CL on the KUKA IIWA. CL investigated differences in a range of robot code from different brands. DG implemented the lightweight robot controller, formulated the test trajectories and performed the parameter identification optimization. JSC and BL provided guidance and input on the work. All authors participated in reviewing and revising the manuscript.

*Context:* Sections 4.1, 4.2 and 4.3.

## 6.5 Paper E

**Daniel Gleeson**, Stefan Jakobsson, Raad Salman, Fredrik Ekstedt, Niklas Sandgren, Fredrik Edelvik, Johan S. Carlson, and Bengt Lennartson
Generating Optimized Trajectories for Robotic Spray Painting
Published in *The IEEE Transactions on Automation Science and Engineering (T-ASE)*, Volume 19, Issue 3, pp. 1380–1391, July 2022.

DOI: 10.1109/TASE.2022.3156803.

The painting trajectory optimization presented in this paper finds an applicator trajectory that produces an even paint thickness. This is an important step for reaching the overall goal of automating the full process of spray-painting parts in a painting booth. An initial trajectory is found by projecting and modifying a raster-curve of parallel sweeps onto the triangulated surface. A paint thickness optimization problem is formulated making use of a paint projection deposition model which has been calibrated to match experimental results from spray painting rectangular test plates. The optimization variables are the spatial positions and orientations of the applicator along with segment time durations, in contrast to the joint space robot code parameters used as optimization variables in Paper B and Paper C. Testing the algorithm on industrial cases show promising results, with the resulting trajectory producing a paint thickness close to the target thickness for most experimentally measured points on the geometry.

Stefan Jakobsson (SJ) formulated the first version of the optimization problem including implementing partial derivative calculations. DG, Raad Salman

(RS) and Niklas Sandgren (NS) further developed the formulation, with DG focused on the trajectory optimization formulation, RS on improving constraints and parameters by running test cases and NS on the projection deposition model. Fredrik Ekstedt (FEk) worked on the algorithm for generating an initial trajectory. Fredrik Edelvik (FEd), JSC and BL provided input and guidance throughout the work. All authors participated in reviewing and revising the manuscript.

*Context:* Sections 5.1, 5.2, 5.3 and 5.4.

# CHAPTER 7

# Conclusions and future work

The work presented in this thesis has been focused on different aspects of robot trajectory optimization. The main focus, which has returned through all different applications, has been on modeling, simulation and optimization. In particular, special attention has often been given to the model formulation step. It is important to formulate a model that captures the main behavior and interesting aspects of the studied physical process, while at the same time being simplified enough such that it is practically useful for simulation, especially in a context where it is used in an optimization problem formulation.

## 7.1 Conclusions

In Paper A the trajectory optimization problem was formulated in quite general terms, with a relatively unconstrained trajectory. The only included limiting factors for the time evolution of the state variables, were the physical limitations of the robot. The main focus was to be able to include collision avoidance constraints in such a way that it was feasible to use the optimization problem formulation even in complex and cluttered environments. The trajectory itself was formulated without specific constraints defining the

parametrization of the trajectory. In such a relaxed formulation of the time evolution of the trajectory, the trade-off in the amount of restriction applied, instead becomes a matter of restricting the optimizer from exploiting weaknesses or inconsistencies in the formulation to arrive at infeasible solutions. Examples of this kind of exploitations are shown in Chapter 3, where the fact that all constraints are only fulfilled pointwise in the sampled trajectory can lead to the final trajectory jumping over infeasible regions in the form of physical barriers. The importance of using a good initial guess to guide the optimization was also highlighted based on simplified examples.

The work presented in Paper B and Paper C was driven by an industrial need where the robot trajectory should be restricted to follow a type of trajectory produced by an industrial robot controller, and that the resulting trajectory should be directly exportable in the form of robot code. In terms of how this affects the optimization problem formulation, this additional constraint is a restriction on the solution space compared to the formulation in Paper A. It also includes the complexity of possibly nested optimization problems in the case where we want to optimize with respect to for example time and energy consumption. This is because the limitation to use robot code parameters introduce an inner optimization, where the robot controller generates a time optimal trajectory given the robot code parameters and its internal constraints.

Inspired by the industrial interest and feedback received for the work on automatic code generation of optimized trajectories, the work presented in Paper D was motivated by the need for expanding previous work to additional robot manufacturers. By finding common types of parametrizations that are possible to apply to robot code generated by different manufacturers, the previous work on robot trajectory optimization would become increasingly more applicable to real world industrial cases. Finding an analytical optimization-free method of generating robot trajectories from given robot code, has additional benefits. It enables the unwinding of the nested optimization mentioned previously for more general cost functions, removing the need to include internal optimality conditions in the optimization problem formulation.

In Paper E, the close coupling to robot code generation is relaxed into a more simplified model of a robot trajectory. Here the trajectory only models the motion of the end effector, while considering a more complex objective

function in the form of paint optimization. The main focus here is to sufficiently simplify the modeled process to make it useable in an optimization routine, while retaining enough information that the resulting trajectory produces an accurate enough result. This is the case both while modeling the robot trajectory and when modeling the paint deposition process.

The research questions stated in Chapter 1 concern trajectory optimization, robot controller modeling and optimization of robotic spray painting. Considering each of the questions, and restating them here for clarity, to what degree have the research questions been answered by the work presented in this thesis?

**RQ1:** *For a robot with freely moving and accurately controlled actuators, how can the optimal collision free trajectory be found when optimizing for cycle time and/or energy consumption?*

The first question is mainly answered in Paper A since this problem formulation placed few constraints on the shape of the robot trajectory, while at the same time optimizing for a combination of both cycle time and an approximation of the energy consumption. In Paper B and Paper C the focus was shifted slightly from the initial research question formulation. Here the main result is that we have optimized robot trajectories subject to robot controller constraints. It has been shown that it is possible to formulate a parametrization of the problem such that the model formulation can be used in an optimization scheme even in cluttered environments. However, it should be noted that this restriction to solutions that are directly exportable to robot code reduces the domain of the problem, excluding solutions that produce lower cost function values by more closely controlling the time evolution of the state vector.

**RQ2:** *How can robot controllers from different manufacturers be accurately emulated and included in an optimization scheme for automatic robot code generation?* The second question is partly answered in Paper B and Paper C,

where the feasibility of the method is proven for a single robot manufacturer trajectory formulation. Accurate parametrizations of the robot controller trajectory are formulated in the two papers and are included directly in the optimization problem. The main difference in Paper C is that the parametrization is formulated using a set of differentiable basis functions, to separate parameters related to determining the shape of the trajectory from the via point

values of the trajectory. By using robot code parameters as variables in the optimization problem, the found solutions are directly exportable as robot code. This work is then possible to expand to multiple robot manufacturers by making use of the work presented in Paper D, where similar parametrizations are found using parameter fitting.

**RQ3:** *How can optimal robotic spray painting trajectories be found for an arbitrary CAD-modeled surface?* This question is answered in Paper E where a spray cone projection model and a paint applicator trajectory approximation are used together to formulate an optimization problem for finding a trajectory that will produce a paint thickness within specified limits. Constraints are added to the problem to increase the probability of being able to physically realize the obtained robot trajectory. The robot trajectory optimization presented in Papers A, B and C make use of a feasible initial collision free trajectory which is found using a path planning search algorithm. Paper E instead describes a broader view of the problem at hand, where the solution is not only focused on optimizing a paint applicator trajectory given a feasible initial solution, but also formulates an algorithm for obtaining this initial solution.

## 7.2 Future work

Future work expanding on the methods presented here can be divided into a few different categories broadly based on the different applications presented.

Starting out, it would be of interest to continue working on improving the trajectory optimization, for example by including more general optimization formulations in the trajectory optimization, such as accurate energy consumption models. The initial path used in the optimization is created by a path planning algorithm. By more closely linking the trajectory optimization to planning, sequencing and load-balancing algorithms, it could be possible to find more efficient solutions in multi-robot stations, especially in cases where robots need to perform tasks in close proximity to each other.

There is also a clear interest, especially from industry partners, in the importance of continually developing, improving and generalizing Robot Controller Light. Different tracks of interest include generalizing the robot controller simulator to include additional robot manufacturers, implementing more gen-

eral trajectory representations, for example speed dependent spatial trajectories, and handling more complex constraints or objectives such as torque minimization. For specific applications it would be beneficial if the controller was able to handle multi-robot stations, with one robot holding the workpiece. There are also possibilities for improving the accuracy of the controller by implementing more general blending patterns and improving the blending of certain combinations of movement commands, for example increasing the accuracy of MoveC-MoveC interpolation.

One area where my colleagues and I plan to work extensively the coming couple of years is improving the painting optimization. One important improvement we wish to achieve is to include robotic kinematic constraints directly in the paint optimization. One way to do this would be to make use of derivative-free optimization algorithms to be able to make use of simulation models which are not easily differentiable. We will also work on improving the simplified paint deposition modeling to more closely resemble the deposition of paint onto real world geometries, without a significant increase in computation time. If successful, this would significantly increase the number of use cases where the optimization is able to produce physically accurate and realizable solutions.

# References


[1] L. Bainbridge, "Ironies of automation," in *Analysis, design and evaluation of man–machine systems*, Elsevier, 1983, pp. 129–135.

[2] *Wingquist laboratory*, `https://www.chalmers.se/en/centres/wingquist/about-us/`, [Online; accessed 2025-09-23], 2025.

[3] *World robotics report*, `https://ifr.org/downloads/press2018/2022_WR_extended_version.pdf`, [Online; accessed 2025-09-23], 2022.

[4] J. Gregory, A. Olivares, and E. Staffetti, "Energy-optimal trajectory planning for robot manipulators with holonomic constraints," *Systems Control Letters*, vol. 61, no. 2, pp. 279–291, 2012.

[5] K. Paes, W. Dewulf, K. V. Elst, K. Kellens, and P. Slaets, "Energy efficient trajectories for an industrial ABB robot," *Procedia CIRP, 21st CIRP Conference on Life Cycle Engineering*, vol. 15, pp. 105–110, 2014.

[6] S. Riazi, K. Bengtsson, O. Wigström, E. Vidarsson, and B. Lennartson, "Energy optimization of multi-robot systems," in *Automation Science and Engineering (CASE), 2015 IEEE International Conference on*, Gothenburg, Sweden, Aug. 2015, pp. 1345–1350.

[7] O. von Stryk and M. Schlemmer, "Optimal control of the industrial robot manutec r3," *Computational optimal control, International series of Numerical Mathematics*, vol. 115, pp. 367–382, 1994.

[8] A. Müller, "Energy optimal control of serial manipulators avoiding collisions," in *Mechatronics, 2004. ICM'04. Proceedings of the IEEE International Conference on*, Istanbul, Turkey, 2004, pp. 299–304.

[9] J. Bukchin and M. Tzur, "Design of flexible assembly line to minimize equipment cost," *IIE Transactions*, vol. 32, no. 7, pp. 585–598, 2000.

[10] T. Hermansson, J. S. Carlson, J. Linn, and J. Kressin, "Quasi-static path optimization for industrial robots with dress packs," *Robotics and Computer-Integrated Manufacturing*, vol. 68, p. 102 055, 2021.

[11] S. LaValle and S. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *Robotics and Automation, IEEE Transactions on*, vol. 14, no. 6, pp. 912–925, 1998, ISSN: 1042-296X.

[12] C. Rösmann, F. Hoffmann, and T. Bertram, "Planning of multiple robot trajectories in distinctive topologies," in *2015 European Conference on Mobile Robots (ECMR)*, IEEE, Lincoln, UK, 2015, pp. 1–6.

[13] M. Gombolay, R. Wilcox, and J. Shah, "Fast scheduling of multi-robot teams with temporospatial constraints," *Robotics: Science and Systems IX*, 2013.

[14] D. Spensieri, J. S. Carlson, F. Ekstedt, and R. Bohlin, "An iterative approach for collision free routing and scheduling in multirobot stations," *IEEE Transactions on Automation science and Engineering*, vol. 13, no. 2, pp. 950–962, 2015.

[15] O. Wigström, B. Lennartson, A. Vergnano, and C. Breitholtz, "High-level scheduling of energy optimal trajectories," *Automation Science and Engineering, IEEE Transactions on*, vol. 10, no. 1, pp. 57–64, Jan. 2013.

[16] O. Wigström, N. Murgovski, S. Riazi, and B. Lennartson, "Computationally efficient energy optimization of multiple robots," in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, Xi'an, China, 2017, pp. 515–522.

[17] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[18] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[19] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[20] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, vol. 1, San Francisco, California, USA, 2000, pp. 521–528.

[21] L. Jaillet, J. Cortés, and T. Siméon, "Transition-based rrt for path planning in continuous cost spaces," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, 2008, pp. 2145–2150.

[22] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[23] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," in *1999 IEEE International Conference on Robotics and Automation*, vol. 1, Detroit, Michigan, USA, 1999, pp. 473–479.

[24] G. Field and Y. Stepanenko, "Iterative dynamic programming: An approach to minimum energy trajectory planning for robotic manipulators," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, vol. 3, Minneapolis, Minnesota, USA, 1996, pp. 2755–2760.

[25] R. Bordalba, T. Schoels, L. Ros, J. M. Porta, and M. Diehl, "Direct collocation methods for trajectory optimization in constrained robotic systems," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 183–202, 2023.

[26] J. E. Cuthrell and L. T. Biegler, "Simultaneous optimization and solution methods for batch reactor control profiles," *Computers & Chemical Engineering*, vol. 13, no. 1-2, pp. 49–62, 1989.

[27] R. Dorfman, "An economic interpretation of optimal control theory," *The American Economic Review*, vol. 59, no. 5, pp. 817–831, 1969.

[28] F. Fahroo and I. M. Ross, "Advances in pseudospectral methods for optimal control," in *AIAA guidance, navigation and control conference and exhibit*, Honolulu, Hawaii, USA, 2008, p. 7309.

[29] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2011, pp. 4569–4574.

[30] W. Han, A. Jasour, and B. Williams, "Non-gaussian risk bounded trajectory optimization for stochastic nonlinear systems in uncertain environments," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, Philadelphia, Pennsylvania, USA, 2022, pp. 11 044–11 050.

[31] M. Gerdts, R. Henrion, D. Hömberg, and C. Landry, "Path planning and collision avoidance for robots," *Numerical Algebra, Control and Optimization*, vol. 2, no. 3, pp. 437–463, 2012.

[32] J. M. Hollerbach, "Dynamic scaling of manipulator trajectories," in *1983 American Control Conference*, IEEE, San Francisco, California, USA, 1983, pp. 752–756.

[33] J. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[34] J. Park, "Motion profile planning of repetitive point-to-point control for maximum energy conversion efficiency under acceleration conditions," *Mechatronics*, vol. 6, no. 6, pp. 649–663, 1996.

[35] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.

[36] T. Lipp and S. Boyd, "Minimum-time speed optimisation over a fixed path," *International Journal of Control*, vol. 87, no. 6, pp. 1297–1311, 2014.

[37] S. Riazi, O. Wigström, K. Bengtsson, and B. Lennartson, "Energy and peak power optimization of time-bounded robot trajectories," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 646–657, 2017.

[38] I. M. Gelfand and S. V. Fomin, *Calculus of variations*. Prentice-Hall, 1963.

[39] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," *Proceedings of 2nd Berkeley Symposium*, pp. 481–492, 1951.

[40] W. Karush, "Minima of functions of several variables with inequalities as side conditions," *Master thesis, University of Chicago*, 1939.

[41] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[42] G. B. Dantzig, A. Orden, P. Wolfe, *et al.*, "The generalized simplex method for minimizing a linear form under linear inequality restraints," *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 183–195, 1955.

[43] L. Pontryagin, *The Mathematical Theory of Optimal Processes* (International series of monographs in pure and applied mathematics). Pergamon Press; [distributed in the Western Hemisphere by Macmillan, New York], 1964.

[44] A. E. Bryson and Y.-C. Ho, "Applied optimal control, revised printing," *Hemisphere, New York*, vol. 10, 1975.

[45] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*, 2nd ed. Society for Industrial and Applied Mathematics, 2010.

[46] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[47] G. Tang, C. Tang, C. Claramunt, X. Hu, and P. Zhou, "Geometric a-star algorithm: An improved a-star algorithm for agv path planning in a port environment," *IEEE Access*, vol. 9, pp. 59 196–59 210, 2021.

[48] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings*, vol. 2, San Francisco, California, USA, 2000, pp. 995–1001.

[49] ABB, *Product specification Controller software IRC5, RobotWare 5.07, Revision 3*, `https://library.e.abb.com/public/536adf0dbe55f9aec125766d003e8e4c/3HAC022349-001_rev3_en_library.pdf`, [Online; accessed 2025-09-23], 2004.

[50] ABB, *RobotStudio API Reference*, `https://developercenter.robotstudio.com/api/robotstudio/api/index.html`, [Online; accessed 2025-09-23], 2025.

[51] KUKA, *KUKA.Sim simulation software*, `https://www.kuka.com/en-de/products/robot-systems/software/planning-project-engineering-service-safety/kuka_sim`, [Online; accessed 2025-09-23], 2025.

[52] Universal Robots, *UR Sim offline simulator*, `https://www.universal-robots.com/download/software-e-series/simulator-linux/offline-simulator-e-series-ur-sim-for-linux-5126-lts/`, [Online; accessed 2025-09-23], 2024.

[53] F. Xu, B. Zi, Z. Yu, J. Zhao, and H. Ding, "Design and implementation of a 7-dof cable-driven serial spray-painting robot with motion-decoupling mechanisms," *Mechanism and Machine Theory*, vol. 192, p. 105 549, 2024.

[54] A. Mark, B. Andersson, S. Tafuri, *et al.*, "Simulation of electrostatic rotary bell spray painting in automotive paint shops," *Atomization and sprays*, vol. 23, no. 1, 2013.

[55] B. Andersson, V. Golovitchev, S. Jakobsson, *et al.*, "A modified tab model for simulation of atomization in rotary bell spray painting," *Journal of Mechanical Engineering and Automation*, vol. 3, no. 2, pp. 54–61, 2013.

[56] T. Johnson, S. Jakobsson, B. Wettervik, B. Andersson, A. Mark, and F. Edelvik, "A finite volume method for electrostatic three species negative corona discharge simulations with application to externally charged powder bells," *Journal of Electrostatics*, vol. 74, pp. 27–36, 2015.

[57] F. Edelvik, A. Mark, N. Karlsson, T. Johnson, and J. S. Carlson, "Math-based algorithms and software for virtual product realization implemented in automotive paint shops," in *Math for the Digital Factory*, Springer, 2017, pp. 231–251.

[58] D. Hegels, T. Wiederkehr, and H. Müller, "Simulation based iterative post-optimization of paths of robot guided thermal spraying," *Robotics and Computer-Integrated Manufacturing*, vol. 35, pp. 1–15, 2015.

[59] S. Nieto Bastida and C.-Y. Lin, "Autonomous trajectory planning for spray painting on complex surfaces based on a point cloud model," *Sensors*, vol. 23, no. 24, p. 9634, 2023.

[60] D. Gleeson, S. Jakobsson, R. Salman, *et al.*, "Robot spray painting trajectory optimization," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, Hong Kong, China, 2020, pp. 1135–1140.

[61] H. Chen, W. Sheng, N. Xi, M. Song, and Y. Chen, "Automated robot trajectory planning for spray painting of free-form surfaces in automotive manufacturing," in *Proceedings 2002 IEEE International Conference on Robotics and Automation*, vol. 1, Washington, DC, USA, 2002, pp. 450–455.

[62] Q. Yu, G. Wang, and K. Chen, "A robotic spraying path generation algorithm for free-form surface based on constant coating overlapping width," in *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, Shenyang, China, 2015, pp. 1045–1049.

[63] P. N. Atkar, H. Choset, and A. A. Rizzi, "Towards optimal coverage of 2-dimensional surfaces embedded in r3: Choice of start curve," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 4, Las Vegas, Nevada, USA, 2003, pp. 3581–3587.