Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CRL

Puppeteering using a two-armed Manipulator Robot

Simon Zimmermann

Master Thesis
April 2018

Computational Robotics Lab
Department of Computer Science
ETH Zürich

Supervisors:
Prof. Dr. Stelian Coros
Dr. Roi Poranne

Abstract

Using manipulator robots for tasks like pick-and-place of rigid objects or assembling of different parts has been the standard in many branches of the industry for quite a while. In order to fulfil more complex tasks, it is becoming a requirement for robots to manipulate objects as dexterously as humans do. To have such capabilities, the robots need to possess a deep understanding of the physical world of dynamical systems. To investigate this grand challenge, this thesis examines a particular problem, namely robotic puppeteering, which requires expert control of very high-dimensional, underactuated dynamical systems.

We present a step by step modelling procedure for a simplified planar marionette. In order to plan trajectories for the marionette's handles such that its body executes desired movements, we solve a nonlinear optimization problem using the derived physical model as constraints. To handle the initial configurations and the parameter settings of the optimization, an interactive MATLAB framework was implemented in the scope of this thesis. It is used to generate movements that avoid obstacles and self-collision, and to identify system parameters of the physical model. All generated trajectories are tested on a real physical system using the two-armed ABB robot YuMi for movement execution. The paths of the systems are tracked in order to compare them to the simulated values.

We show that our approach can be used to generate trajectories in simulation, which can directly be applied to real-world systems. The systems are able to follow these trajectories accurately.

Zusammenfassung

Dass Industrieroboter für Tätigkeiten wie beispielsweise das Verschieben und Sortieren von festen Objekten oder das Zusammenbauen von verschieden Bauteilen eingesetzt werden, gehört heutzutage zum Standardequipment vieler Industriezweige. Um komplexere Tätigkeiten ausführen zu können ist es für Roboter unumgänglich, mit Objekten so geschickt umzugehen wie wir Menschen. Damit solche Fähigkeiten entwickelt werden können, müssen Roboter ein tiefes Verständnis für die physikalische Welt von dynamischen Systemen besitzen. Wir untersuchen diese grosse Herausforderung indem wir uns einem bestimmten Problem widmen: der Puppenspielerei. Das Manipulieren von Marionetten erfordert grosse Expertise in der Kontrolle über dieses hochdimensionale, unteraktuierte dynamische System.
In dieser Arbeit präsentieren wir eine Schritt-für-Schritt Modellierungsprozedur für eine vereinfachte, planare Marionette. Um die Trajektorien für die "Griffe" der Marionette zu planen, sodass ihr Körper eine gewünschte Bewegung ausführt, lösen wir ein nicht-lineares Optimierungsproblem. Dafür nutzen wir das hergeleitete Modell der Marionette. Die Anfangsbedingungen sowie die Parametereinstellungen des Optimierungsproblems können in einer interaktiven Benutzeroberfläche, programmiert in MATLAB, eingestellt und verändert werden. Es kann ausserdem genutzt werden, um Bewegungen zu erstellen, welche Hindernisse oder Kollisionen vermeiden, und um die Systemparameter des physikalischen Modells zu berechnen. Alle generierten Trajektorien wurden auf echten physikalischen Systemen getestet, wobei der ABB Roboter YuMi genutzt wurde, um die zuvor berechneten Bewegungsabläufe auszuführen. Die zurückgelegten Wege der Systeme wurden gemessen, um sie anschliessend mit den simulierten Werten vergleichen zu können.
Wir zeigen, dass unser Ansatz genutzt werden kann, um Trajektorien für dynamische Systeme zu erzeugen, welche direkt auf den entsprechenden echten Systemen ausgeführt werden können. Es wird demonstriert, dass diese Wege präzise denen der Simulation entsprechen.

Acknowledgements

I would like to express my gratitude towards Stelian Coros, my supervisor and professor. Working with you on this project was a great inspiration for me. Each of our meetings served to enhance my motivation even more. I'm very grateful for being fully included in the research team from the first moment onward.

For all of his assistance regarding optimization I would like to thank Roi Poranne. Your guidance was greatly appreciated.

Lastly a big thank you to the whole CRL group for providing fresh perspectives during our troubleshooting and brain storming sessions.

Contents

List of Figures v

List of Tables vi

1 Introduction 1

2 Related Work 3
2.1 Manipulation of Cable-Suspended Payload . . . 3
2.2 Marionettes . . . 5

3 Materials and Methods 7
3.1 Framework . . . 7
3.2 Robot . . . 9
3.3 Systems . . . 10
3.4 Position Tracking . . . 10

4 Trajectory Optimization 11
4.1 Space Ship . . . 11
4.2 Pendulum . . . 14
4.3 String Model . . . 16
4.3.1 Unilateral Model . . . 17
4.3.2 Conversion to Maximum Coordinates . . . 18
4.4 System Parameter Identification . . . 19
4.5 Obstacle and Collision Avoidance . . . 20
4.6 Extension to more complex Dynamical Systems . . . 21
4.7 Simplified Marionette in 2D . . . 22

5 Results 24
5.1 System Parameter Identification . . . 24
5.2 Test Trajectories . . . 27

6 Conclusion and Outlook 30

Bibliography 34

A Robot Controller Functions 35

List of Figures

3.1 MATLAB GUI for trajectory optimization 8
3.2 The two-armed manipulator robot *YuMi* 9

4.1 Figure of pendulum model 15
4.2 Image of simplified planar marionette in simulation and reality 22

5.1 Result plots of positions of parameter identification - Oscillation experiment . . . 25
5.2 Result plots of positions of parameter identification - Drop experiment 26
5.3 Plots of positions of test trajectory 1: Pendulum - Side move 27
5.4 Plots of positions of test trajectory 2: Double pendulum - Jump into cup 28
5.5 Plots of positions of test trajectory 3: Simplified marionette - Waving motion . . 29

List of Tables

5.1 Overview of parameters used for different test cases 24

Chapter 1

Introduction

Robots of all kinds have been a part of daily life for quite some time now. They are designed to help us with tasks where high accuracy and/or a high rate of repeatability is required, or with jobs where the working environment would be too dangerous for a human. Robots are often used in industry in order to increase productivity. Manipulator robots for tasks like pick-and-place of rigid objects or assembling of different parts have become the standard and a lot of effort has been put into the optimization and fine tuning of these processes.
As the objects the robots have to manipulate become more and more complex due to technical progress, the requirements of the complexity of the tasks the robots have to fulfil is also increasing. The expectations of robots have risen to being able to manipulate not only rigid, but also dynamic systems as dexterously as humans are capable of. To gain such capabilities, the robots need to possess a deep understanding of the physical world of these kinds of systems.
To investigate this grand challenge, this thesis examines a particular problem, namely robotic puppeteering. This means that a robot is supposed to manipulate a marionette: a puppet that is supported and controlled by manipulating strings attached to the limbs of its body. Puppeteering is an entertainment form that has been practiced for centuries. As marionettes are underactuated, nonlinear and highly coupled dynamical systems, they build a very interesting research platform. The problem serves as a good test environment for many current issues in robotics, such as systematic modeling of systems with a high number of degrees of freedom, semantics for high-level motion planning and control, and numerical optimization strategies for motion generation. This is why this platform cannot only be used for entertainment, but also for analyzing problems appearing in industry and on construction sites. A few examples are the control of objects hanging from a crane or robots carrying parts through production sites. Furthermore, it can be useful for studying mapping techniques of movements from humans or robots to puppets, and generalizing them to other robotic systems.

In this thesis, we address the problem of planning trajectories for complex dynamical systems such that they behave in a desired way. Starting with the simple dynamic system of a pendulum, we increase the complexity step by step towards a simplified version of a marionette. Within the scope of this thesis, we remain within the two-dimensional space.
The trajectories are planned by solving a nonlinear constrained optimization problem. To realize this, we implemented a planning framework in MATLAB using its GUI and solver functionalities as presented in section 3.1. In order to verify if the planned trajectories for the models of the different dynamical systems are close to reality, their physical counterparts were built, using the materials described in section 3.3, and given to our YuMi robot, which is briefly introduced in section 3.2. While the robot executed the precalculated trajectories, we tracked its end-effectors and the point masses of the dynamical systems using tracking methods described in section 3.4.

The results of this analysis are presented in chapter 5 in form of some test trajectories generated for different dynamical systems.
The details of the optimization problem that needs to be solved in order to plan the trajectories are presented in chapter 4. The process explanations begin in section 4.1 with the idea of a simple space ship, which is modeled as a point mass that can freely move along one dimension. This formulation can later be used to model the robot's end-effectors. The dynamical description of the pendulum is introduced in section 4.2. As a marionette's limbs are typically manipulated by a puppeteer by non-rigid strings, this must be taken into account in the model. This is done in section 4.3, where the string between the handle and the point mass of the pendulum is modeled using a spring-damper system. This leaves us with some unknown system parameters such as the spring and damper constants. In order to determine them for this particular system, a parameter identification procedure is presented in section 4.4, using the same optimization environment as for the trajectory generation.
While planning the trajectories, we want to make sure that neither the robot's end-effectors nor the dynamical system are colliding with an obstacle or with itself. This is why we introduce an obstacle and collision avoidance procedure described in section 4.5.
In order to move towards a full model of a marionette, we extend the complexity of the dynamical system to a double pendulum and a system with a rigid connection, which is presented in section 4.6. We then have all the ingredients to develop a first version of a marionette in 2D as described in section 4.7.
A conclusion and outlook of the presented work is given in the final chapter 6 of the thesis.

Chapter 2

Related Work

This chapter gives an overview of the research that has already been done, relating to the topic of this thesis. It is divided into two sections: 2.1 covers research in the area of manipulating cable-suspended payloads and section 2.2 gives an impression of what has been done in the field of manipulating marionettes using robotics.

2.1 Manipulation of Cable-Suspended Payload

Plenty of research has been done in the area of manipulating cable-suspended payloads in a desirable way using all kinds of machines and robots. In the earlier years in which this topic has been in focus of researchers, large efforts have been put into reducing residual oscillations in high speed rest to rest motions of freely suspended payloads, as this is a common problem appearing in many industry sectors such as automotive and construction. A few examples of such problems are gantry and crane systems, cooperative control with multiple cable systems and aerial transportation of payloads. A possible approach that generates optimal swing-free motion trajectories for the manipulation of such a dynamical system was presented by Zameroski et al. [1]. They outlined the development of a Dynamic Programming algorithm for discrete time systems and described its application with a rapid maneuver on a long slender payload suspended by a cable at each end, which was held by two independent robot manipulators. The approach was shown to be advantageous in comparison to former developed methods using closed-loop control, impulse convolution and parameter optimization methods, especially when dealing with intrinsically nonlinear applications.
Another strongly related topic in this area is helicopters carrying these kinds of loads. It has been pushed mostly by the aerospace research community due to the stability problems from which such systems suffer. In order to design stability augmenting techniques, a good model of the system is necessary, which required research effort. One example was provided in the work of Bisgaard et al. [2]. They presented the annotated result of the modeling and verification of a generic slung-load system using a small-scale helicopter. They used a redundant coordinate formulation based on Gauss's Principle of Least Constraint using the Udwadia-Kalaba equation to derive the model. It contained detection of wire slackening and tightening as well as aerodynamic coupling between helicopter and load. In addition, they introduced a numerical stabilization algorithm to compensate for drift in the simulation.
Around the same time, another work was published by Bernard and Kondak [3], focusing on the control aspect and on the movement of the rope connecting small size unmanned helicopters and load. Their approach is based on two control loops: an outer loop to control the translation of each helicopter in compound and an inner loop to control the orientation of the helicopters. In addition, the usage of force sensors in the ropes was introduced, which leads to a simplification of

the inner loop controller and makes the system robust against variations of the system parameters. They addressed the problem of oscillations in the robes caused by external disturbances as well and proposed a solution based on a load state observer. They verified their results in real flight experiences where one and then three helicopters transported a load.
A more general approach on how to address the problem of trajectory generation for underactuated control of a suspended mass was presented by Schultz and Murphey [4]. They used a magnetically-suspended, differential drive robot in combination with an additional winch system for controlling the length of a string suspending a mass. First, they constructed a simplified dynamic model of the system using the principle of kinematic reduction. This produced a mixed kinematic-dynamic model that isolated the modeling of the system actuators from the rest of the system and led to the advantage that the inputs become generalized velocities instead of generalized forces. These generalized velocities can directly be sent as commands to the robot. Using this model, they presented a nonlinear, infinite-dimensional optimization algorithm. This algorithm automatically deals with the complexities introduced by nonlinear dynamics and underactuation to synthesize dynamically feasible system trajectories for a wide array of trajectory generation problems.
A lot of related publications have been done by a group from the University of Pennsylvania in collaboration with various other universities. In their early work [5], [6], a quadrotor with a cable-suspended load with eight degrees of freedom and four degrees underactuation was established to be a differentially-flat hybrid system with the load position and the quadrotor yaw serving as the flat outputs. Using the flatness property, they presented a trajectory generation method that enables finding nominal trajectories with various constraints that either result in minimal load swing or can be used to generate dynamically agile motions. They modeled the quadrotor with load as a hybrid system taking into account that the dynamics of the system switch when the tension in the cable drops to zero or when the slack cable becomes taut when the tension gets reestablished. This means that two models were derived: one for the case with nonzero cable tension and one for zero cable tension. They considered the switching dynamics by inducing a unilateral tension constraint. Furthermore, they developed a nonlinear geometric control design that enables tracking of outputs defined by either the quadrotor attitude, the load attitude or the load's position. They presented almost global exponential stability proofs for the controller design of their system.
As a next step, the findings were extended to the problem of cooperative transportation of a cable-suspended payload by multiple quadrotors [7]. They showed numerical and experimental results illustrating the superior performance of this method compared to earlier derivations based on quasi-static models.
In their later work, they presented a trajectory planning method to navigate the quadrotor with load through known obstacle-filled environments by formulating the problem as a Mixed Integer Quadratic Program (MIQP) [8]. They formulated the problem assuming that waypoint specifications and obstacle locations were given. They showed that the method is practical for generating trajectories including aggressive obstacle avoidance maneuvers.
In their most recent work [9], they added an estimation component to the system by observing the payload using a downward-facing onboard camera. It can be used to estimate the load's state relative to the quadrotor using an extended Kalman filter. They demonstrated closed-loop payload control in the 3D space with a planning, estimation and control pipeline implemented on an onboard processor. The results showed that the method can be applied for agile slung-load maneuvers and large payload angles to the vertical axis.
There have also been approaches to solve the problem using machine learning. For example Faust et al. [10] presented a motion planning method for generating trajectories with minimal residual oscillations for rotorcraft carrying a suspended load, where they relied on a finite-sampling batch reinforcement learning algorithm to train the system for a particular load. Similarly, Palunko et al. [11] presented a problem where they specified a reference trajectory for the suspended load and a

quadrotor needed to learn its own trajectory ensuring that the suspended load tracks this reference path. The proposed method is based on least-square policy iteration (LSPI), which is a type of reinforcement learning. In general, the advantage of using machine learning to solve these kinds of problems is that no model of the underlying system is needed. A disadvantage comes from the fact that the learning procedure needs to be applied again as soon as the system parameters change. One possible design method for learning motion control consists of using a model-based optimal control algorithm to initialize the policy of a sampling based reinforcement learning algorithm. This was addressed by the work of Crousaz et al. [12]. They presented an approach where the initial control trajectory design was performed for a quadrotor with a cable-suspended load using the iterative LQG (iLQG) algorithm. Afterwards, the generated controller was used to initialize the policy of a learning algorithm. Additionally, they introduced a hybrid model of the system, meaning that depending on the cable being taut or free, the system has two dynamics modes with a different number of states and degrees of underactuation. They showed that their approach can be used for more aggressive maneuvers with the quadrotor.

There are some similarities between the work presented in this section and the one this thesis is about. We start our investigation of how to manipulate dynamical systems with a pendulum (as presented in sections 4.2 and 4.3), which is - considering our model - essentially a cable-suspended load. We are focusing on applying fast maneuvers while avoiding obstacles with our system rather than trying to generate swing-free trajectories. The main difference to the previous work is that we are using a manipulator robot instead of a flying platform for manipulation. This is one of the reasons why we don't consider the control part of the robot in this work. Our focus is on deriving models of the dynamical systems such that their behavior fit the one of their real world counterparts as closely as possible.

2.2 Marionettes

Some research has been done using the same platform as in this thesis: marionettes. In 2002, Xing and Chen [13] started investigating this topic by trying to generate expressive gestures on a robotic puppet controlled by a behavior-based method. They used the platform in order to study the similarity between gesture and speech.
Two years later, Chen et al. [14] introduced the evolution and the engineering aspects of traditional marionette design and manipulation skills. In their paper, they gave a detailed overview of the anatomy of marionettes and how conventional marionette manipulation works. Additionally, they introduced their own prototype of a robotic marionette system.
In the same year, Yamane et al. [15] presented a method to control a motorized marionette using motion capture data from a human actor as well as from a traditional marionette operated by a professional puppeteer. They investigated whether there is a way to easily create new performances for the marionette using this motion capture data. They addressed the problem of adapting the human motion data to the marionette as their kinematic and dynamic properties extensively differ. They then applied a feedforward controller to prevent extraneous swings of the marionette's hands.
Later on, Kim et al. [16] proposed the traditional marionette control as an interface to control articulated characters in interactive computer games. In their work they explained how to implement a virtual marionette based on physically-based modeling. The presented system used haptic interfaces to model the control of a real-world marionette and provided responsive forces as a result of the created motions. They were able to create reasonably complicated motions at highly interactive rates with the limitation that their system could not handle inter-string and body-string collisions.

Johnson and Murphey [17] considered the problem of modeling a robotic marionette with more precision. They presented a mixed dynamic-kinematic modeling technique that removed the controller dynamics from the marionette. They derived examples by modeling a single arm moving in a plane as well as a 3D marionette using generalized coordinates and Lagrange's equation. Additionally, they used an expensive-space tree (EST) motion planner to generate a path from an input configuration to a goal for a puppet arm with seven degrees of freedom.
The same authors published another paper [18] where they focused on automatically morphing trajectories from one mechanical system into trajectories of another system with potentially different dynamics. They demonstrated the process using an example based on a marionette, as the trajectory morphing process allowed them to specify trajectories for the puppet using human motion capture data or animation tools without considering the puppet's abilities. The method consists of a projection operator-based trajectory optimization which finds a dynamically-admissible trajectory for the target system that approximates the desired trajectory of the source system.
Another approach was introduced in the work of Martin and Egerstedt [19]. They presented an optimal timing control formulation of the problem of controlling autonomous marionettes. To do so, they laid out a formal *motion description language* that allowed them to specify what a puppet should do at a high level of abstraction. Afterwards they used optimal control techniques for compiling these high-level specifications into executable control code, meaning that the high-level commands were parsed by a dynamical system that produced optimized, hybrid control laws corresponding to motions, locations and temporal durations for each motion primitive.
A further improvement of the work described above was shown in the publications of Martin et al. [20], Murphey and Egerstedt [21] and Murphey and Johnson [22]. These papers presented a project aimed at the creation of fully automated marionettes for puppet plays, focusing on the design of software for embedded control of robotic marionettes using choreographies to specify the marionette's motions.

In the work presented in this thesis, we do not focus on creating marionette motions such that they look realistically like a human movement. Our focus lies on the modelling side of the problem. We want to create models of marionettes such that their movements in simulation fit the ones of the real puppets as good as possible. We plan to use these results to create more high-level control commands for marionettes in future work.

Chapter 3

Materials and Methods

In this chapter, the tools, materials and methods used in this thesis are presented. Section 3.1 introduces the framework that was developed in the scope of this thesis, which handles the trajectory planning. The next section 3.2 gives some technical details about the robot that is used to verify the results generated in simulation on a physical platform. The materials utilized to build the physical dynamical systems are introduced in section 3.3. Furthermore, two different tracking systems for data acquisition are briefly presented in 3.4.

3.1 Framework

We use our own small framework developed in *MATLAB* in order to plan trajectories for different dynamical systems. It is briefly introduced in the following two paragraphs: the Graphical User Interface (GUI) and the solver that is used to solve the optimization problem.

GUI:

In order to be able to flexibly manipulate and change all the parameters that have to be set for the optimization problem, we implemented a GUI using the standard tools that MATLAB provides [23]. A screenshot is shown in figure 3.1. It allows the user to easily generate figures, text and check boxes, pop-up menus, sliders and buttons. All of which can trigger a specific predefined callback function when pressed, activated or changed. It is also able to track the mouse input, which is used for the implementation of a drag-and-drop functionality of objects depicted in an interactive visualization. This is helpful for changing the initial conditions for the trajectory optimization quickly.
The trajectory generation environment is implemented as an *interactive* optimization framework. This means that the GUI runs in parallel to the optimization solver, which permits the user to interactively manipulate the parameters of the optimization like objective weights, enable or disable constraints, or drag and drop obstacles around while the optimization is running. Features like continuous trajectory plotting and display of objective and constraint violation values allow us to monitor the optimization progress and directly interfere with it. In order to enable the communication between GUI and optimization, the number of iterations is set to a small value such that the optimization breaks regularly. It is then reinitialized using the updated GUI parameters and its current state as initial conditions.

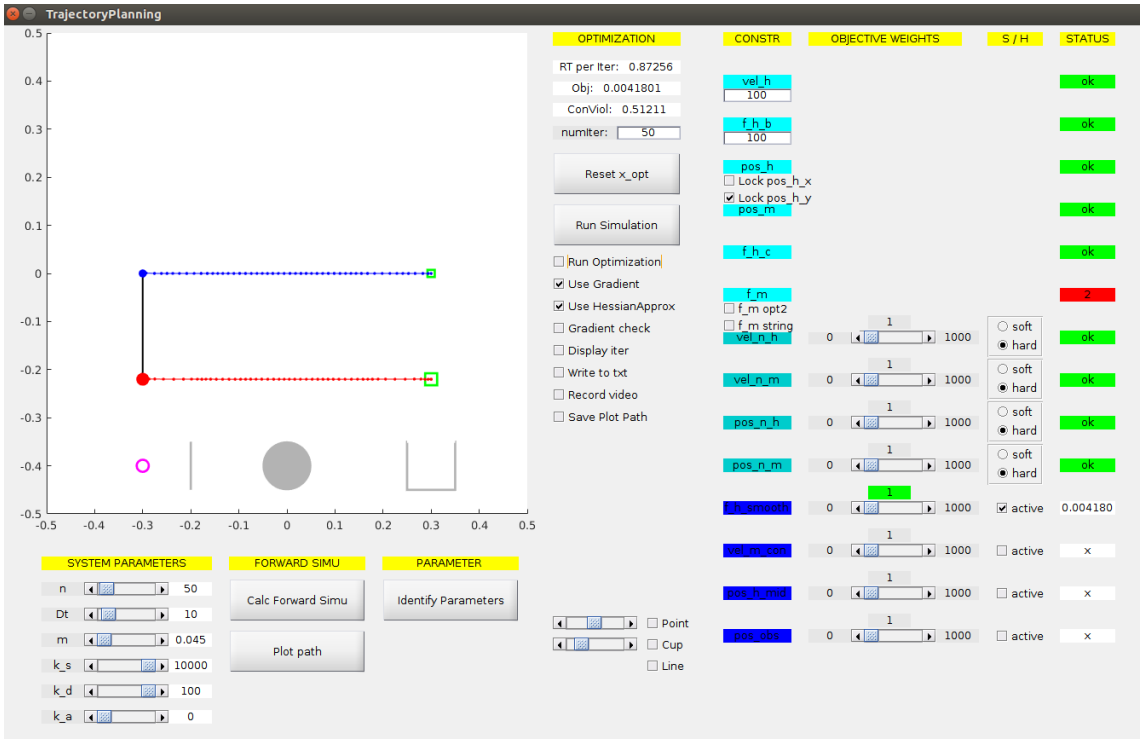

Figure 3.1: MATLAB GUI for trajectory optimization

Solver:

In order to solve our trajectory optimization problem, we use the nonlinear programming solver *fmincon* provided by MATLAB [24]. It takes objectives, boundary constraints, linear as well as nonlinear equality and inequality constraints and tries to find a solution which minimizes the objectives by prioritizing them by their weights. Different optimization algorithms can be chosen to find a solution. For our problem, we are using the *interior-point* algorithm [25]. It follows a barrier approach that employs sequential quadratic programming and trust regions to solve subproblems occurring in the iteration. The method shows good convergence for our purpose. To speed up the computation time, we provide the algorithm the analytically derived gradients of the objectives and the nonlinear constraints, so that they don't have to be estimated. For this case, fmincon provides a helpful tool: *CheckGradients* [24]. When activated, it compares the user-supplied gradients to derivatives computed using finite differences [26] and outputs an indication whether the results of the analytic gradients are similar to the finite-difference solutions. This offers us the possibility to check if the implementation of our gradients is correct. fmincon also allows us to choose in which way the Hessian, the second derivative of all objectives and constraints, shall be approximated. We use *L-BFGS* to do so [27]. It is the limited-memory version of the BFGS algorithm, as it stores less data throughout the progress of the optimization. BFGS belongs to the quasi-Newton methods, where the Hessian matrix does not need to be evaluated directly, but is approximated using updates specified by gradient evolutions.

3.2 Robot

In order to test the trajectories generated in simulation on a physical system, we use the robot *YuMi* (depicted in figure 3.2) from the Swiss robot manufacturer *ABB Robotics* [28]. It is a two-armed manipulator that has been developed for small component assembly in the electronics industry. YuMi is what is known in industry as a *collaborative robot*, meaning that it has been designed to work hand in hand with humans. The advantage in using such a robot is that it has very high safety standards such that no additional protective measures are necessary.
YuMi possesses seven degrees of freedom per arm and has a range of 559 mm. One of its main advantages is that its movements are very precise. According to [29] it has a position repeatability of 0.02 mm. In addition, with its maximum tool center point (TCP) velocity of 1.5 $^{m}/_{s}$ and a maximum TCP acceleration of 11 $^{m}/_{s^2}$ it can move quite fast. One of the drawbacks is that it cannot lift heavy loads, as its maximum handling capacity lies at 0.5 kg per arm.

Figure 3.2: The two-armed manipulator robot *YuMi*

Communication:

The interface we are using to control the YuMi consists of two parts: On one hand, Berkeley AutoLab's *yumipy* module [30] is used. It is a *Python* interface for sending diverse commands to the YuMi. It is still under development and can be used as a ROS [31] as well as a Python-Only installation. We use the Python-Only version with some adaptions to our needs. On the other hand, controllers running directly on the robot are necessary. They are implemented in the programming language *RAPID*, developed by ABB [32], which provides multiple functions to steer the robot. The most important RAPID functions used for this thesis are presented in appendix A.
These two interfaces can be connected via Ethernet (more details about YuMi's *E/A interface* can be found in [33]) to exchange data over a TCP/IP socket connection.
We use this interface in the following way: First, we generate optimal trajectories for the robot's end-effectors in MATLAB (more details in chapter 4). Then, these trajectories are given in form of discretized position commands and time steps to the yumipy module. From there, the data is sent to the robot controllers using socket communication. The end-effector and timing commands are sent to the robot one by one, but are stored in a buffer on the robot controller first. A separate command causes all buffered commands to be executed in order. This ensures that the robot follows the trajectories smoothly without any time-delays.

3.3 Systems

To build the physical systems to test our simulated trajectories with the help of our robot, we use the following materials:

- **Point masses:** We use stainless steel spheres with a mass of 45 g and a diameter of approximately 20 mm [34].
- **Strings:** For "massless" strings, we use fishing line of the brand *FORTEC strong*, which has a diameter of 0.2 mm and can carry a maximum load of 3.2 kg. For our systems, a string length of approximately 200 mm is chosen.
- **Rigid Connections:** All rigid connections of the different systems presented in the following chapter are 3D printed, using a *stratasys* 3D printer with ASA material [35].

3.4 Position Tracking

To track the position and movements of our dynamical systems, we use the following two methods:

OptiTrack: This tracking system is typically used in research areas like virtual reality, movement sciences, robotics and animation [36]. We use it to generate data for the system parameter identification part presented in the chapters 4.4 and 5.1. It is an optical-passive motion capture system, which means that is uses retroreflective markers that are tracked by infrared cameras. These markers can be attached to individual spots on an object. It is an easy-to-use system and provides very accurate tracking data in 3D.

OpenCV Object Tracking: As the configuration of our OptiTrack system does not allow for easy placement of the robot within the tracking area, we additionally use a vision system to track the positions of the different systems during a movement. The resulting data is used for accuracy evaluation as presented in chapter 5.2. The movements are first recorded on a video camera. Afterwards, we use the object tracking functionality of the open source computer vision library *OpenCV* in python. The *MIL Tracker* as presented in [37] has been found to give the best results for our purpose.

Chapter 4

Trajectory Optimization

In order to plan the path for the robot's end-effectors to manipulate a given dynamical system in a desired way, we formulate and solve optimization problems. This chapter describes how these problems are devised. It starts with the basic problem formulation to generate trajectories for the robot's end-effectors in section 4.1. Afterwards we add a comparatively simple dynamical system in the form of a pendulum to the problem, which is presented in section 4.2. Then the system's complexity is raised step by step in the following sections. 4.3 presents the chosen model of the pendulum's string. After that, we present a system parameter identification procedure in section 4.4. Obstacle and collision avoidance is added to the optimization problem in chapter 4.5. In addition, we present the extension to slightly more complex dynamical systems in 4.6. Finally we have all the "ingredients" to describe our first model of a simplified marionette in 2D in section 4.7.

4.1 Space Ship

To formulate our basic path planning problem, we first consider the simple example of a space ship, which can freely move in one dimension. We want to find a path $x(t)$ this space ship should follow in an optimal way, in the sense of a predefined objective, while fulfilling some constraints. We discretize time into equally spaced time intervals $t_0, ..., t_n$ and use a shorthand notation of $x_i = x(t_i)$. In general, we can formulate this simple trajectory planning problem in 1D as a quadratic program

$$\begin{aligned} \underset{x_1,...,x_{n-1},f_0,...,f_{n-1}}{\text{minimize}} \quad & \sum_{i=0}^{n-1} f_i^2 \\ \text{subject to} \quad & f_i = m \cdot a_i, \quad \forall i = 0, ..., n-1 \\ & l_f \leq f_i \leq u_f, \quad \forall i = 0, ..., n-1 \\ & l_x \leq x_i \leq u_x, \quad \forall i = 1, ..., n-1 \\ & l_v \leq v_i \leq u_v, \quad \forall i = 1, ..., n-1, \end{aligned} \tag{4.1}$$

where x_i, f_i, a_i and v_i describe the position, the acting force, the acceleration and the velocity of the space ship at time i, respectively. n denotes the number of steps that are taken, m is the space ship's mass and l_* and u_* represent the lower and the upper bound of the corresponding unknowns, respectively.
By looking at this formulation we can see that we are solving for the positions x_i as well as for the forces f_i the space ship has to apply in order to move. As an objective we choose the sum of all forces squared. This penalizes high forces, which is equivalent to penalizing high accelerations of

the space ship. The constraints ensure that the positions, velocities and forces remain within valid ranges. As we want to use this example to model the robot's end-effectors, we need these bounds, as the robot's range, speed and applicable force are limited. In addition, we want the space ship to obey physics by fulfilling Newton's second law, which is why we add the equality constraint $f_i = m \cdot a_i$.
The velocity v_i can be approximated using *backward finite differences*

$$v = \frac{dx}{dt} \approx \frac{x(t) - x(t - \Delta t)}{\Delta t} \rightarrow v_i \approx \frac{x_i - x_{i-1}}{\Delta t}, \tag{4.2}$$

where the constant Δt represents the difference between two time instances. In a similar way we choose to approximate the acceleration with *2nd order central finite differences*

$$a = \frac{dv}{dt} \approx \frac{x(t + \Delta t) - 2 \cdot x(t) + x(t - \Delta t)}{\Delta t^2} \rightarrow a_i \approx \frac{x_{i+1} - 2 \cdot x_i + x_{i-1}}{\Delta t^2}. \tag{4.3}$$

These approximations mean that f_i represents the force that has to be applied such that the space ship can move from position x_i to x_{i+1}, while v_i is the velocity the object possesses at position x_i. In addition to the constraints presented in equation 4.1, we also want to impose boundary conditions on the velocity, which can be expressed by

$$v_0^* = \frac{x_0 - x_{-1}}{\Delta t}, \quad v_n^* = \frac{x_n - x_{n-1}}{\Delta t}, \tag{4.4}$$

where v_0^* and v_n^* represent the desired velocities of the space ship at the beginning and at the end of the trajectory, respectively. This is why we add x_{-1}, x_0 and x_n to our optimization variables, which results in the following vector of variables:

$$\boldsymbol{x} = [x_{-1}, x_0, x_1, ..., x_{n-1}, x_n, f_0, ..., f_{n-1}]^T. \tag{4.5}$$

Additionally, we need to impose boundary conditions for the space ship's start and end position

$$x_0 = x_0^*, \quad x_n = x_n^*. \tag{4.6}$$

To summarize, this results in the following *quadratic program*:

$$\begin{aligned}
\underset{\boldsymbol{x}}{\text{minimize}} \quad & \sum_{i=0}^{n-1} f_i^2 \\
\text{subject to} \quad & f_i = m \cdot a_i, \quad \forall i = 0, ..., n-1 \\
& l_f \leq f_i \leq u_f, \quad \forall i = 0, ..., n-1 \\
& l_x \leq x_i \leq u_x, \quad \forall i = -1, ..., n \\
& l_v \leq v_i \leq u_v, \quad \forall i = 1, ..., n-1 \\
& v_0 = v_0^*, \quad v_n = v_n^* \\
& x_0 = x_0^*, \quad x_n = x_n^*.
\end{aligned} \tag{4.7}$$

As a first step, we solve this optimization problem with *MATLAB* using the solver function *quadprog* [38]. It finds a minimum for a problem specified in the form

$$\begin{aligned}
\underset{\boldsymbol{x}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{x}^T \boldsymbol{H} \boldsymbol{x} + \boldsymbol{f}^T \boldsymbol{x} \\
\text{subject to} \quad & \boldsymbol{A} \cdot \boldsymbol{x} \leq \boldsymbol{b} \\
& \boldsymbol{A}_{eq} \cdot \boldsymbol{x} = \boldsymbol{b}_{eq} \\
& \boldsymbol{l}_b \leq \boldsymbol{x} \leq \boldsymbol{u}_b.
\end{aligned} \tag{4.8}$$

For our case, the matrix $\boldsymbol{H}$ and the vector $\boldsymbol{f}$ have to describe the objective function $\sum_{i=0}^{n-1} f_i^2$, which is why they have to be chosen as

$$\boldsymbol{H} = \begin{pmatrix} \mathbf{0}_{(n+2)\times(n+2)} & \mathbf{0}_{(n+2)\times n} \\ \mathbf{0}_{n\times(n+2)} & \boldsymbol{I}_{n\times n} \end{pmatrix}, \quad \boldsymbol{f} = \mathbf{0}_{(2n+2)\times 1}, \tag{4.9}$$

where $\mathbf{0}$ presents a zero matrix and $\boldsymbol{I}$ is the identity matrix. The dimensions can be derived from the fact that we have $n+2$ position and n force variables to optimize for.
The bounding constraints of f_i and x_i can be handled using the vectors $\boldsymbol{l}_b$ and $\boldsymbol{u}_b$ such that

$$\boldsymbol{l}_b = \begin{pmatrix} l_x \cdot \mathbf{1}_{(n+2)\times 1} \\ l_f \cdot \mathbf{1}_{n\times 1} \end{pmatrix}, \quad \boldsymbol{u}_b = \begin{pmatrix} u_x \cdot \mathbf{1}_{(n+2)\times 1} \\ u_f \cdot \mathbf{1}_{n\times 1} \end{pmatrix}, \tag{4.10}$$

where $\mathbf{1}$ represents a vector whose entries consist of ones.
The constraints on the velocities v_i are taken into account using matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$. To do so, we reformulate these constraints as

$$-x_i + x_{i-1} \leq -\Delta t \cdot l_v, \quad x_i - x_{i-1} \leq \Delta t \cdot u_v, \tag{4.11}$$

which results in:

$$\boldsymbol{A} = \begin{pmatrix} 0 & -1 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & -1 & 1 & 0 & \cdots & 0 \\ & & \vdots & & & \vdots & \\ 0 & 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & -1 & 0 & \cdots & 0 \\ & & \vdots & & & \vdots & \end{pmatrix} \in \mathbb{Z}^{(2(n-1))\times(2(n+1))}, \tag{4.12}$$

$$\boldsymbol{b} = \begin{pmatrix} -\Delta t \cdot l_v \cdot \mathbf{1}_{(n-1)\times 1} \\ \Delta t \cdot u_v \cdot \mathbf{1}_{(n-1)\times 1} \end{pmatrix}. \tag{4.13}$$

Finally, the equality constraints are handled using matrix $\boldsymbol{A}_{eq}$ and vector $\boldsymbol{b}_{eq}$. Again, we reformulate the constraints for f_i by using equation 4.3 as

$$\frac{\Delta t^2}{m} \cdot f_i - x_{i+1} + 2 \cdot x_i - x_{i-1} = 0, \tag{4.14}$$

so that we can build:

$$\boldsymbol{A}_{eq} = \begin{pmatrix} -1 & 2 & -1 & 0 & 0 & \cdots & 0 & \frac{\Delta t^2}{m} & 0 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & 0 & \cdots & 0 & 0 & \frac{\Delta t^2}{m} & 0 & \cdots & 0 \\ & & \vdots & & & & & 0 & \vdots & & & \\ -1 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & & 0 \\ 0 & 0 & 0 & 0 & \cdots & -1 & 1 & 0 & 0 & \cdots & & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & & 0 \end{pmatrix} \in \mathbb{R}^{(n+4)\times(2(n+1))} \tag{4.15}$$

$$\boldsymbol{b}_{eq} = \begin{pmatrix} 0 & \cdots & 0 & \Delta t \cdot v_0^* & \Delta t \cdot v_n^* & x_0^* & x_n^* \end{pmatrix}^T \in \mathbb{R}^{(n+4)\times 1}. \tag{4.16}$$

The last four rows of this system come from the boundary conditions on velocity and position.

The problem can easily be expanded to the 2D case. When assuming that both directions are independent of each other (which is a reasonable assumption for our robot's end-effectors as long as there are no rotational mechanics involved), the two systems are completely decoupled, which is why they can either be solved as independent optimization problems, or by stacking all parameters together in the corresponding matrices.

4.2 Pendulum

In order to get an understanding of how to manipulate dynamical systems, we start with a comparatively simple system: a pendulum. As a first step we model it as a point mass which is attached to a handle by a rigid connection. The handle is what the robot is supposed to manipulate in a further step. It has the same constraint and objective configuration as the space ship presented in chapter 4.1. In order to embed the dynamical system into our existing optimization problem derived in the last section, we need to add:

- position variables of the pendulum's point mass $\boldsymbol{x}_{-1}^m, \boldsymbol{x}_0^m, \boldsymbol{x}_1^m, ..., \boldsymbol{x}_{n-1}^m, \boldsymbol{x}_n^m$ to the vector of optimization variables $\boldsymbol{x}$, where $\boldsymbol{x}_i^m$ is a 2D vector containing the x and y coordinates of the point mass;
- the boundary conditions for these position variables and the corresponding velocities $\boldsymbol{x}_0^{m,*}$, $\boldsymbol{v}_0^{m,*}$, $\boldsymbol{x}_n^{m,*}$, $\boldsymbol{v}_n^{m,*}$ similarly to how it is presented in chapter 4.1;
- the equations of motion of the dynamic system to the constraints, such that the law of physics of the system are obeyed.

We will now derive the equations of motion for our pendulum system and reformulate them as hard constraints. There are several ways to do so. In this thesis we present the derivation using the *Lagrange equations* [39]. In contrast to Newton's second law, they use the system's kinetic and potential energies instead of forces. The resulting formulation gives us an advantage when deriving the string model as presented in section 4.3. The general Lagrange equation for non-constrained coordinates with external generalized forces is

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}}\right) - \frac{\partial L}{\partial q} - Q_q = 0, \tag{4.17}$$

where the *Lagrangian function*

$$L(q, \dot{q}, t) = T(q, \dot{q}, t) - V(q, t) \tag{4.18}$$

expresses the difference between the kinetic energy $T(q, \dot{q}, t)$ and the potential energy $V(q, t)$ of the system. $q(t)$ denotes the generalized coordinates of the system. The term Q_q represents the generalized force acting on the system.
The definitions of the used coordinate systems are depicted in figure 4.1, where $\{x_I, y_I\}$ expresses the fixed world coordinate system, which will be referred to as *maximum coordinates* and $\{x_h(t), y_h(t)\}$ denotes the position of the handle. As we are only interested in how our point mass moves in time, our only unknown in this system is the angle $\theta(t)$ (as introduced in figure 4.1).

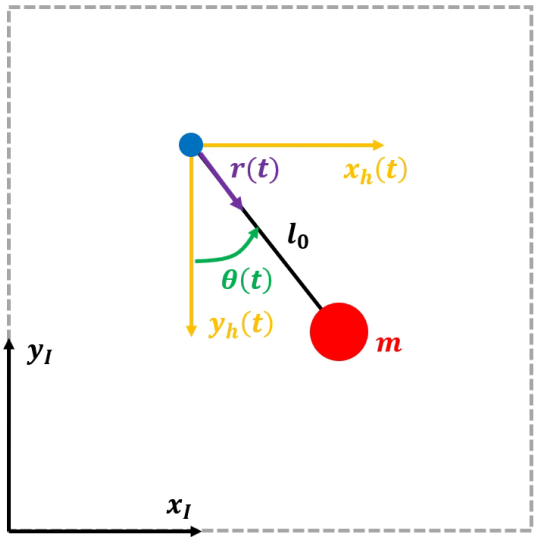

Figure 4.1: Figure of pendulum model

We want to find an expression for the position of the point mass in maximum coordinates, which is why we derive the position vector and its derivative with respect to time as

$$\boldsymbol{r}_m(t) = \begin{pmatrix} x_h(t) \\ y_h(t) \end{pmatrix} + l_0 \begin{pmatrix} \sin(\theta(t)) \\ -\cos(\theta(t)) \end{pmatrix} \tag{4.19}$$

$$\boldsymbol{v}_m(t) = \dot{\boldsymbol{r}}_m(t) = \begin{pmatrix} \dot{x}_h(t) \\ \dot{y}_h(t) \end{pmatrix} + l_0\dot{\theta}(t) \begin{pmatrix} \cos(\theta(t)) \\ \sin(\theta(t)) \end{pmatrix} \tag{4.20}$$

where l_0 expresses the constant length of the pendulum's connection to the handle. For readability reasons we will drop the time-dependency from now on. The kinetic and potential energy can be derived as

$$T(\theta, \dot{\theta}) = \frac{1}{2} m \boldsymbol{v}_m^T \boldsymbol{v}_m = \frac{1}{2} m \left(\dot{x}_h^2 + \dot{y}_h^2 + l_0^2\dot{\theta}^2 + 2l_0\dot{\theta} \left(\dot{x}_h \cos(\theta) + \dot{y}_h \sin(\theta) \right) \right) \tag{4.21}$$

$$V(\theta) = m\boldsymbol{g}^T \boldsymbol{r}_m = mg \left(y_h - l_0 \cos(\theta) \right), \tag{4.22}$$

where g expresses the gravity constant and m is the pendulum's mass. As there are damping forces acting on the point mass due to air friction, we find the generalized force to be

$$Q_\theta(\dot{\theta}) = -k_a\dot{\theta}, \tag{4.23}$$

where k_a represents the air damping constant.
This results in the following Lagrangian function:

$$\begin{aligned} L(\theta, \dot{\theta}) &= T(\theta, \dot{\theta}) - V(\theta) \\ &= \frac{1}{2} m l_0^2 \dot{\theta}^2 + m l_0 \dot{\theta} \left(\dot{x}_h \cos(\theta) + \dot{y}_h \sin(\theta) \right) + \frac{1}{2} m \left(\dot{x}_h^2 + \dot{y}_h^2 \right) + mgl_0 \cos(\theta) - mgy_h. \end{aligned} \tag{4.24}$$

Using the Lagrange equation for $q = \theta$ and simplifying the equation, we can derive the equation of motion of the pendulum to be

$$\begin{aligned}
0 &= \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} - Q_\theta \\
&= \frac{d}{dt}\left(ml_0^2\dot{\theta} + ml_0\left(\dot{x}_h\cos(\theta) + \dot{y}_h\sin(\theta)\right)\right) \\
&\quad - \left(ml_0\dot{\theta}\left(-\dot{x}_h\sin(\theta) + \dot{y}_h\cos(\theta)\right) - mgl_0\sin(\theta)\right) + k_a\dot{\theta} \\
&= ml_0^2\ddot{\theta} + ml_0\left(\ddot{x}_h\cos(\theta) + \ddot{y}_h\sin(\theta)\right) + mgl_0\sin(\theta) + k_a\dot{\theta} \\
&= ml_0^2\ddot{\theta} + ml_0\left(\ddot{y}_h + g\right)\sin(\theta) + ml_0\ddot{x}_h\cos(\theta) + k_a\dot{\theta} \\
&= l_0\ddot{\theta} + \left(\ddot{y}_h + g\right)\sin(\theta) + \ddot{x}_h\cos(\theta) + \frac{k_a}{ml_0}\dot{\theta}
\end{aligned} \tag{4.25}$$

We add this equation as equality constraints into our optimization problem, where we use the same formulation to approximate the velocities and accelerations as presented in equations 4.2 and 4.3.

4.3 String Model

As we are aiming to control a marionette which is brought into motion using strings, we do not want the connection between handle and point mass to be rigid, but to fit the real string as closely as possible. This is why we choose to model the string as a *spring-damper system*. To do so we introduce an additional generalized coordinate $r(t)$, which defines the distance between the handle and the point mass (as introduced in figure 4.1). This changes the position and velocity vectors of the point mass in maximum coordinates as follows:

$$\boldsymbol{r}_m(t) = \begin{pmatrix} x_h(t) \\ y_h(t) \end{pmatrix} + r(t)\begin{pmatrix} \sin(\theta(t)) \\ -\cos(\theta(t)) \end{pmatrix} \tag{4.26}$$

$$\boldsymbol{v}_m(t) = \dot{\boldsymbol{r}}_m(t) = \begin{pmatrix} \dot{x}_h(t) \\ \dot{y}_h(t) \end{pmatrix} + \begin{pmatrix} \dot{r}(t)\sin(\theta(t)) + r(t)\dot{\theta}(t)\cos(\theta(t)) \\ -\dot{r}(t)\cos(\theta(t)) + r(t)\dot{\theta}(t)\sin(\theta(t)) \end{pmatrix} \tag{4.27}$$

Again we drop the time dependency notation and derive kinetic and potential energy of our new system:

$$T(\theta,\dot{\theta},r,\dot{r}) = \frac{1}{2}m\left(\left(\dot{x}_h + \sin(\theta)\dot{r} + \cos(\theta)r\dot{\theta}\right)^2 + \left(\dot{y}_h - \cos(\theta)\dot{r} + \sin(\theta)r\dot{\theta}\right)^2\right) \tag{4.28}$$

$$V(\theta,r) = gm\left(y_h - \cos(\theta)r\right) + \frac{1}{2}k_s(r-l_0)^2 \tag{4.29}$$

The resulting Lagrange function is

$$\begin{aligned}
L(\theta,\dot{\theta},r,\dot{r}) &= T(\theta,\dot{\theta},r,\dot{r}) - V(\theta,r) \\
&= \frac{1}{2}m\left(\dot{x}_h^2 + \dot{y}_h^2 + \dot{r}^2 + r^2\dot{\theta}^2\right) + m\dot{x}_h\left(\sin(\theta)\dot{r} + \cos(\theta)r\dot{\theta}\right) \\
&\quad + m\dot{y}_h\left(-\cos(\theta)\dot{r} + \sin(\theta)r\dot{\theta}\right) - gm\left(y_h - \cos(\theta)r\right) - \frac{1}{2}k_s\left(r-l_0\right)^2.
\end{aligned} \tag{4.30}$$

Note that in order to represent the spring-damper part in the equations of motion, we have to add the energy of the spring in our potential energy, where k_s represents the *spring constant*. As the damping part is dissipative, we need to take it into account by adding the generalized force

$$Q_r(\dot{r}) = -k_d \dot{r} \tag{4.31}$$

to our Lagrange equation, where k_d is the *damping constant*. As we have two generalized coordinates now, we have to calculate the Lagrange equation for both of them in order to derive the equations of motion to be

$$\begin{aligned} 0 &= \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} - Q_\theta \\ &= r\ddot{\theta} + (\ddot{y}_h + g)\sin(\theta) + \ddot{x}_h \cos(\theta) + 2\dot{r}\dot{\theta} + \frac{k_a}{mr}\dot{\theta} \end{aligned} \tag{4.32}$$

$$\begin{aligned} 0 &= \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{r}}\right) - \frac{\partial L}{\partial r} - Q_r \\ &= \ddot{r} + \ddot{x}_h \sin(\theta) - (\ddot{y}_h + g)\cos(\theta) - r\dot{\theta}^2 + \frac{k_s}{m}(r - l_0) + \frac{k_d}{m}\dot{r} \end{aligned} \tag{4.33}$$

We have derived the equations of motion of a pendulum whose connection between handle and point mass is modeled by a spring-damper system. We integrate this into our optimization problem by replacing the EoM derived in chapter 4.2 with these new equality constraints.

4.3.1 Unilateral Model

As we want our string model to fit its physical counterpart as closely as possible, we have to do an additional adjustment. In its current state, our spring-damper model applies forces in cases where the string is stretched as well as when it is compressed. Our physical string can only take forces along the direction of the string when it is at nominal length l_0 or in case it is stretched, but not when it is compressed. We take this into account by introducing an *unilateral model*. It can also be used for contact models [40]. We consider the spring term in our equations of motion:

$$\frac{k_s}{m}(r(t) - l_0) := \frac{k_s}{m}\Delta r(t). \tag{4.34}$$

We can see that the spring component influences the motion of the pendulum only if $\Delta r(t) \neq 0$. In order to formulate the unilateral model, we construct a function $F_s(\Delta r)$ which takes Δr as an input and models the behavior of the real-world string as good as possible. To do so, we want to design $F_s(\Delta r)$ in such a way that it is zero below a certain small value $\epsilon < 0$ - which covers the case when the string is "compressed" - and grows with respect to a quadratic function when it is larger than epsilon. We do this because we want the spring term to start acting on the system slightly before Δr switches from a negative to a positive value. Additionally, we want the spring the become very stiff with growing Δr. As we want to calculate the derivatives of this function in order to derive the gradients in our optimization problem, we want it to be continuously differentiable. This is why we introduce an additional cubic function for a certain range of our unilateral function. As a consequence, the function is formulated as

$$F_s(\Delta r) = \begin{cases} \frac{1}{2}a_1 \Delta r^2 + b_1 \Delta r + c_1 & \text{if } \Delta r \geq 0 \\ \frac{1}{3}a_2 \Delta r^3 + \frac{1}{2}b_2 \Delta r^2 + c_2 \Delta r + d_2 & \text{if } \epsilon \leq \Delta r < 0 \\ 0 & \text{else.} \end{cases} \tag{4.35}$$

We can determine the seven unknown coefficients by using the following conditions:

- We set $a_1 = s$, where s denotes the stiffness of the function. It defines how fast the function value "rises" with increasing Δr and we can choose it to match the physical properties of the real-world string.
- We want the value of the cubic and the quadratic function to be the same at point $\Delta r = 0$. This should not only be true for the two functions, but also for their first and second derivatives. This gives us three additional conditions.
- The cubic function has to be zero at point $\Delta r = \epsilon$. Again, this has to be true for the first and the second derivatives as well, which leaves us with three additional equations.

Using these conditions, the function coefficients can be determined to be

$$\begin{aligned} a_1 &= s \\ b_1 &= -\frac{1}{2}a_1\epsilon \\ c_1 &= \frac{1}{6}a_1\epsilon^2 \\ a_2 &= -\frac{a_1}{2\epsilon} \\ b_2 &= a_2 \\ c_2 &= b_1 \\ d_2 &= c_1. \end{aligned} \tag{4.36}$$

We apply this function to the previously derived equations of motion by replacing the former spring term $\frac{k_s}{m}(r(t) - l_0)$ by the new one $\frac{k_s}{m}F_s(\Delta r)$.
The same kind of model is applied for the damping term $\frac{k_d}{m}\dot{r}(t)$ as well, using $\dot{r}$ instead of Δr.

4.3.2 Conversion to Maximum Coordinates

The derived EoM are formulated with respect to the generalized coordinates θ and r. This means that the calculation of the position of the point mass in maximum coordinates always depends on the position of the handle $\{x_h, y_h\}$. In applications such as obstacle avoidance, it is beneficial to optimize directly for the mass positions in maximum coordinates, as the problem formulation becomes much easier. This is why we reformulate the derived EoM in terms of maximum coordinates. Essentially, this means that we want to reformulate θ, r and all their derivatives in terms of x_m, y_m and their derivatives, representing the position of the point mass in maximum coordinates. This formulation for both generalized coordinates can be found to be

$$r = \sqrt{(x_h - x_m)^2 + (y_h - y_m)^2} := \bar{r} \tag{4.37}$$

$$\sin(\theta) = -\frac{(x_h - x_m)}{\bar{r}} \rightarrow \theta = \arcsin\left(\frac{x_m - x_h)}{\bar{r}}\right) \tag{4.38}$$

To avoid confusion while having a simpler notation, we additionally introduce the variable $\bar{r}$, which is equal to r but represents its formulation in terms of x_m and y_m.

In addition, we calculate their first and second derivatives with respect to time:

$$\dot{r} = \frac{(x_h - x_m)(\dot{x}_h - \dot{x}_m) + (y_h - y_m)(\dot{y}_h - \dot{y}_m)}{\bar{r}} \tag{4.39}$$

$$\dot{\theta} = \frac{\dot{x}_m - \dot{x}_h}{\bar{r}\sqrt{1 - \left(\frac{x_m - x_h}{\bar{r}}\right)^2}} \tag{4.40}$$

$$\begin{aligned} \ddot{r} = & \frac{1}{((x_h - x_m)^2 + (y_h - y_m)^2)^{\frac{3}{2}}} \\ & - ((x_h - x_m)(\dot{x}_h - \dot{x}_m) + (y_h - y_m)(\dot{y}_h - \dot{y}_m))^2 \\ & + ((x_h - x_m)^2 + (y_h - y_m)^2)((\dot{x}_h - \dot{x}_m)^2 + (\dot{y}_h - \dot{y}_m)^2) \\ & + ((x_h - x_m)^2 + (y_h - y_m)^2)((x_h - x_m)(\ddot{x}_h - \ddot{x}_m) + (y_h - y_m)(\ddot{y}_h - \ddot{y}_m)) \end{aligned} \tag{4.41}$$

$$\ddot{\theta} = \frac{(-x_h + x_m)(\dot{x}_h - \dot{x}_m)^2 + (\bar{r} + x_h - x_m)(\bar{r} - x_h + x_m)(-\ddot{x}_h + \ddot{x}_m)}{\bar{r}^3 \left(1 - \frac{(x_h - x_m)^2}{\bar{r}^2}\right)^{\frac{3}{2}}} \tag{4.42}$$

4.4 System Parameter Identification

The mass m and the nominal string length l_0 of our pendulum are easily measurable. This is not the case for the rest of the system parameters, which are:

- the spring constant k_s
- the damping constant k_d
- the air damping constant k_a.

In order to identify them, we use position tracking data which is generated using the OptiTrack system as briefly described in section 3.4. To get results that fit those of our simulation environment as closely as possible, we use our existing optimization framework with the following adjustments:

- We disable the end-point boundary conditions for velocity $v_n^m = v_n^{m,*}$ and position $x_n^m = x_n^{m,*}$ of the point mass.
- We disable all the objectives.
- We add the three system parameters to our optimization vector $\boldsymbol{x}$
- We add an objective which penalizes the difference between the taken tracking data of the physical pendulum and the simulated positions of the point mass:

$$f = \frac{1}{2}\sum_{i=1}^{n+1}\left((x_i^m - x_i^{track})^2 + (y_i^m - y_i^{track})^2\right). \tag{4.43}$$

  This objective is introduced to the optimization by adding it to the others with a weighting factor that can be changed in the MATLAB GUI.

$\{x_{track}(i), y_{track}(i)\}$ represents the OptiTrack position measurement of the real-world system at time step i. Adding this objective means that we are minimizing the difference between the tracking data and the positions of the point mass in simulation while still taking into account the constraints given by the equations of motion of the system, as those are still present in our optimization problem as equality constraints.
Note that the framerate used to collect the data with OptiTrack might vary from the framerate we are choosing by setting the parameter Δt for our optimization. If this is the case, we prepare the data such that it is applied in the right instant in time. As the framerate in simulation is usually higher as the one used for data collection, this can be done by applying the objective only for time instances i where a corresponding tracked position has been measured.

4.5 Obstacle and Collision Avoidance

To generate useful trajectories for our dynamical system, it is crucial that we are also able to introduce obstacles into our trajectory optimization procedure. We do so by adding another objective to our optimization problem. We want to be able to avoid two kinds of obstacles: *circles* with a certain radius R_c and *lines* with a certain thickness t_l.
The idea behind this objective is the same for both kinds of obstacles: First, we calculate the shortest distances d_i between the obstacle and the trajectory of a desired point of our dynamical system (for the example of the pendulum this could either be the handle or the point mass) at each point in time. Afterwards we can use these distances to create a unilateral objective function $p(d_i)$ in a similar way as presented in section 4.3.1. The difference is that this constructed function has its highest value at $d_i = 0$ and decreases the larger d_i gets, as we want to penalize when the object and the system point are too close together. Our obstacle avoidance objective can be formulated as

$$f = \frac{1}{2} \sum_{i=0}^{n+1} p(d_i)^2. \tag{4.44}$$

In case we have multiple obstacles, we add such an objective for each one of them.
Now we will have a closer look at how to calculate the distances d_i to the individual obstacles.

Circular Obstacles:

In a first attempt, we calculate the distance between the center of the circular obstacle and our system point for each discretized position of the trajectory and subtract the circle radii from the resulting distance. This leads to the following problem: As the trajectory of the system point is discretized in time, the obstacle avoidance does not apply for places inbetween two points of the trajectory. This can result in behavior where the system speeds up beforehand so that it is able to go right through the obstacle instead of avoiding it. As this is not what we wish to achieve, we need to adjust the formulation: Instead of calculating the distance between two points, we determine the shortest distance between a point and a line segment. The point is still the center point of our circular obstacle. For the line segment we choose the path between two subsequent positions of the trajectory of our system point. Examples of how the distance calculation between a point and a line segment can be implemented are presented in [41]. Again, this is done for each line segment of the trajectory. In this way we penalize the path the system point is taking rather than certain points along the way. This implementation shows much better performance in terms of obstacle avoidance.

Line Obstacles:

For similar reasons as described in the paragraph about circular obstacles, we calculate the shortest distance between two line segments in order to avoid line obstacles, in which one line segment is again the system's trajectory between two subsequent positions and the other one is the line obstacle itself (implementation examples can be found in [42]). Afterwards, we subtract the point radius and the line thickness from the calculated distance and penalize it with the objective as described in equation 4.44.
We can extend this kind of obstacle to model different obstacle shapes such as for example a cup (used in chapter 5.2) by connecting three lines together and adding one objective to our optimization for each of them.

Remark: For the case where we introduce more complex dynamical systems (as introduced in section 4.6), this kind of objective can also be used to avoid the collision of several point masses of the dynamical system, multiple handles or its connections by strings or rigid bodies.

4.6 Extension to more complex Dynamical Systems

Using all the findings presented in the former sections, the modeling of more complex dynamical systems is relatively straightforward. In this section, two systems which are part of the experiments presented in chapter 5.2 will be explained briefly.

Double Pendulum:

To model a double pendulum in our framework, we add another point mass to our optimization variables and apply the same linear constraints as for the first one. Additionally, we implement the same nonlinear constraints as for point mass one based on the EoM as presented in section 4.3, but connect the new point mass to the first one instead of the handle. This results in a double pendulum, where the two point masses are connected by strings.

Rigid Bar on Two Strings:

Another system we are using for the evaluation of our algorithm is one consisting of two pendulums on strings, where the point masses are rigidly connected. To realize this, we add another pendulum to our framework, subject to the same constraints and objectives as already presented. In addition, we apply the equality constraint

$$0 = \sqrt{\left(x_i^{m,1} - x_i^{m,2}\right)^2 + \left(y_i^{m,1} - y_i^{m,2}\right)^2} - l_{b,0} \quad \forall i, \tag{4.45}$$

where $l_{b,0}$ represents the length of the bar connection between point masses 1 and 2 at time instant 0. This *distance constraint* demands that the distance between the two point masses has to be preserved for all time steps.

4.7 Simplified Marionette in 2D

Now we have all the ingredients needed to derive our first simplified model of a marionette in 2D. Pictures of the puppet in simulation and its physical counterpart are presented in figure 4.2. To model and generate trajectories for this system, we introduce the following optimization variables:

- The positions and corresponding forces of four handles in both x and y direction (as described in section 4.1, corresponding handles are labeled in figure 4.2 as h_L, h_{CL}, h_{CR} and h_R, respectively)
- The positions of six point masses in both x and y direction, representing the left and right hands and elbows as well as the body of the marionette (labeled in figure 4.2 as m_{HL}, m_{EL}, m_{BL}, m_{BR}, m_{ER} and m_R, respectively)

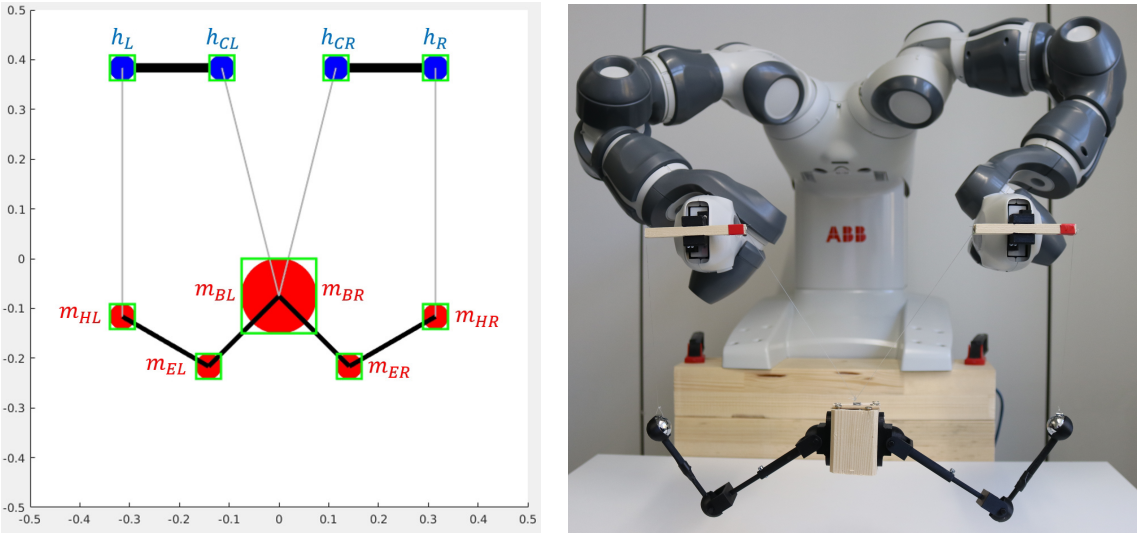

Figure 4.2: Image of simplified planar marionette in simulation (left) and reality (right)

In addition to all linear equality and inequality constraints corresponding to each of these positions and forces as introduced in sections 4.1 and 4.2, we add the following nonlinear equality constraints:

- Pendulum with spring-damper system constraint (as presented in section 4.3) for all the strings connecting the handles and the point masses, concretely for:
  - left handle h_L and left puppet hand m_{HL}
  - central left handle h_{CL} and left puppet body m_{BL}
  - central right handle h_{CR} and right puppet body m_{BR}
  - right handle h_R and right puppet hand m_{HR}
- Distance constraint (as presented in section 4.6) to take care of the rigid connections between the handles and between the point masses of the puppet. This means that we apply the constraints between:
  - handles h_L and h_{CL}
  - handles h_R and h_{CR}

- point masses m_{HL} and m_{EL}
- point masses m_{EL} and m_{BL}
- point masses m_{BL} and m_{BR} (which should remain close together)
- point masses m_{ER} and m_{BR}
- point masses m_{HR} and m_{ER}

To give our marionette a motivation to move, we add another objective to our framework: We want a certain point mass (for example the left hand) to follow a predefined trajectory. To realize this, we want the point mass $\{x_{t_j}^{m_{HL}}, y_{t_j}^{m_{HL}}\}$ to be at a certain position $\{x_{t_j}^{p}, y_{t_j}^{p}\}$ at a specific point in time t_j and this should be true for $j = 1, ..., J$, where J presents the number of positions. We formulate the objective to be

$$f = \frac{1}{2} \sum_{j=1}^{J} \left((x_{t_j}^{m_{HL}} - x_{t_j}^{p})^2 + (y_{t_j}^{m_{HL}} - y_{t_j}^{p})^2 \right). \tag{4.46}$$

In addition, we add objectives to avoid collisions between the individual system points and connections as presented in section 4.5.

Chapter 5

Results

Within the process of this thesis, a lot of different trajectories for the various systems presented in chapter 4 have been generated and tested using their physical counterparts. In this chapter, we present some representative results. Section 5.1 covers the findings received when applying the system parameter identification method as presented in section 4.4. Two different experiments to identify the parameters are introduced. Section 5.2 provides the results when measuring the accuracy of the trajectories generated in simulation in comparison to the tracking data of the physical system. To get an idea of the performance of the method, the results for three different trajectories on three different dynamical systems are presented. After the generation in simulation, the trajectories of the handles are sent as end-effector position commands to the robot. While executing the motion, the robot's handles as well as the point masses of the systems are tracked using a vision algorithm as explained in section 3.4. The chosen values for time step Δt and the number of steps n for the different tests are presented in table 5.1.

Table 5.1: Overview of parameters used for different test cases

Test Cases	n	Δt
Oscillation Experiment - Not Optimized	300	1 / 90
Oscillation Experiment - Optimized	300	1 / 90
Drop Experiment - Large Δt	100	1 / 90
Drop Experiment - Small Δt	1000	1 / 900
Test Trajectory 1: Pendulum - Side Move	200	1 / 120
Test Trajectory 2: Double Pendulum - Jump into Cup	60	1 / 30
Test Trajectory 3: Marionette - Waving Motion	50	1 / 30

5.1 System Parameter Identification

Oscillation Experiment:

As a first experiment, we collect data using our physical pendulum by releasing it at a 61 degrees offset from its zero position and record its oscillation. This experiment does not give us much information about the spring and damper constants k_s and k_d, but it can be used to gather some statements about the air damping term k_a and the time step Δt. The spring constant k_s needs to be high enough in order to keep the string close enough to its nominal length. If it is chosen to be too small, the simulated string gets longer and longer over time.

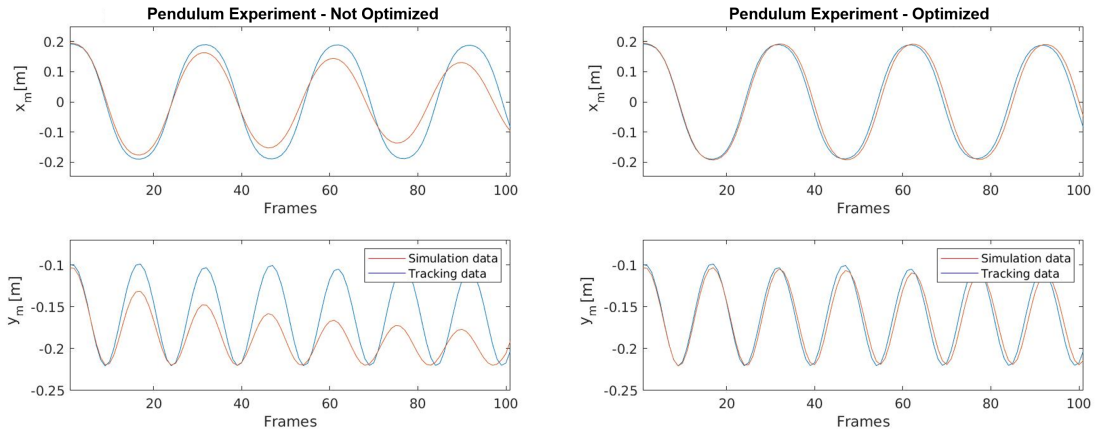

Figure 5.1: Result plots of positions of the parameter identification oscillation experiment; Simulation data in red, tracking data in blue; $\{x_m, y_m\}$ express the position of the point mass over time; Left side: results without optimizing the system parameters; Right side: results with system parameters included in the optimization; The parameters used to generate the results are listed in table 5.1

We use Δt to approximate the velocity and the acceleration of the system as explained in section 4.1. In general, we can say that the smaller we choose Δt to be, the smaller the errors of these approximations are. Choosing Δt small comes with the cost of having longer calculation times, as more variables are introduced to the optimization. Furthermore, the approximation errors get larger when we are dealing with high velocities and high accelerations. This behavior is observed when comparing the simulated data with the real-world tracking data, as it is done in the plots shown in figure 5.1. The plot on the left side shows the results when applying the optimization procedure without optimizing the system parameters. We can see that in comparison to the tracking data (in blue), the simulated oscillations (in red) damp down much faster. We could confirm that this behavior appears due to the approximation errors by repeating the optimization using smaller Δt's. The smaller we choose Δt, the closer the simulation gets to the tracking data at the cost of longer computation time.
As an additional check, we add the system parameters to the optimization variables (as presented in section 4.4) and solve the resulting optimization problem. We receive the expected result: The optimization compensates for the approximation errors by setting the air damping constant k_a to a negative value. Again, this value approaches zero and ends up in a very small positive term the more we decrease Δt. The resulting plot is depicted in figure 5.1 on the right side, where the air damping constant approached a value of $k_a = -0.006$. It can be seen that in this case, the simulated data match the tracking data very well.
As a conclusion of these findings, we can say that depending on how fast our system has to move and to accelerate, we have to choose the time step Δt small enough such that the simulation fits the reality closely. Another approach would be to introduce higher order approximation methods to make the approximation errors smaller.

Drop Experiment:

The second experiment addresses the estimation of the spring and damper constants k_s and k_d. To do so, we lift the point mass of our physical pendulum at zero degrees in the vertical direction such that its initial position is 45 percentage of its nominal string length. Then we let go and record the position data until the system has stopped bouncing up and down. This data is depicted in figure 5.2 by the blue curves. We use it in order to solve for k_s and k_d while the system's constraints are still active. We do this again for different values of Δt. As the left plot of figure 5.2 shows, we are not able to reach the desired system behavior when a relatively large Δt is chosen. On the other hand, by choosing a Δt which is small enough, we can approximate the physical system's behavior quite well (as shown in the right plot of figure 5.2). In this case, the system parameters as part of the optimization converged to the values $k_s = 1527$ and $k_d = 0.566$.
We observe that the damping term in the equations of motion does not have a huge influence in how the system behaves, but it helps the algorithm numerically to faster converge to a good solution.

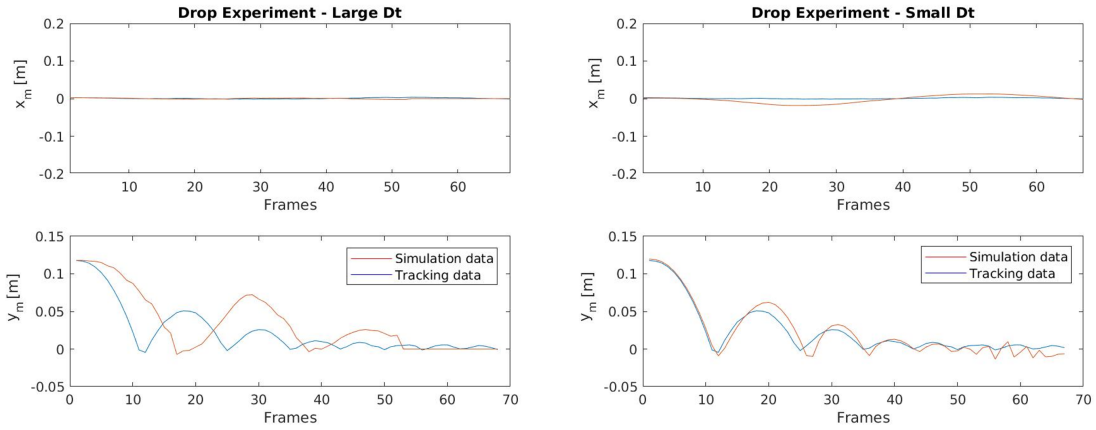

Figure 5.2: Result plots of positions of the parameter identification drop experiment;
Simulation data in red, tracking data in blue; $\{x_m, y_m\}$ express the position of the point mass over time; Left side: optimization with relatively large Δt; Right side: optimization with small Δt;
The parameters used to generate the results are listed in table 5.1

5.2 Test Trajectories

Test 1: Pendulum - Side Move

As a first test case in order to check the performance of our physical system in comparison to the simulated data, we choose our pendulum model as presented in section 4.3 and let it execute a simple side movement. To realize this, we give the handle the objective to be at a specific point half-way of its trajectory. Additionally, both the handle and the point mass must be back at their initial positions with zero velocity at the end of the time horizon. To regularize the optimization, we add an objective that penalizes high point mass velocities. This results in a trajectory plan as depicted in figure 5.3. By looking at the blue curve depicting the tracking data, we can see that the real-world system is capable of sticking to the precomputed simulation positions very accurately with the given parameter settings. The noise in the blue curves comes from the slight inaccuracy of the vision tracking algorithm.
Although this movement seems to be fairly easy to execute, it turns out that it is close to impossible for a human to put the system in motion and back to rest without having a much larger amount of "control effort" than this optimal trajectory that has been generated for the robot's end-effector.

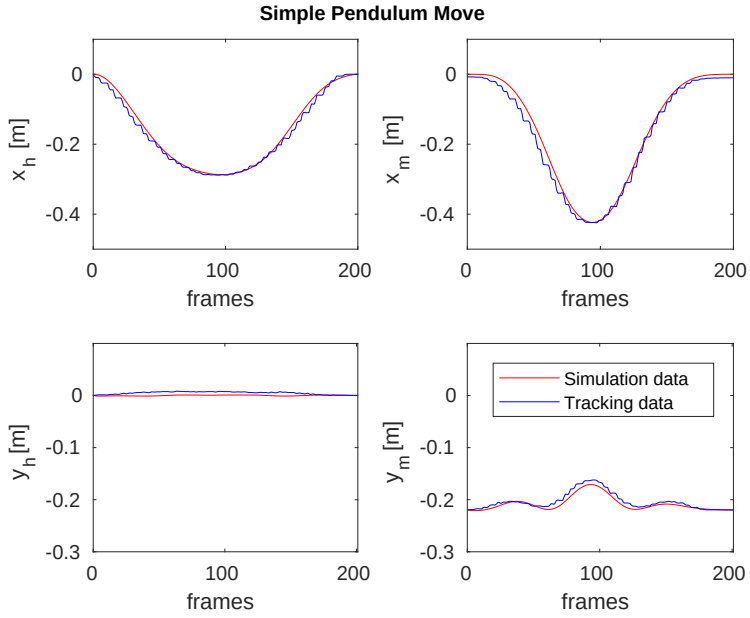

Figure 5.3: Plots of positions of test trajectory 1: Pendulum - Side move;
Simulation data in red, tracking data of the physical system in blue; $\{x_h, y_h\}$ and $\{x_m, y_m\}$ express the positions of the handle and the point mass over time, respectively; The parameters used to generate the results are listed in table 5.1

Test 2: Double Pendulum - Jump into Cup

We also tested our method using more complex dynamical systems. We demonstrate representative results using a double pendulum whose lower point mass is supposed to jump into a cup. The double pendulum is modeled as presented in section 4.6 and the cup obstacle is introduced as three line obstacles as explained in section 4.5. As the task would be quite easy to fulfill by just lifting the system up, we lock the vertical direction of the robot's end-effector, such that it is only allowed to move along a horizontal line. This setup forces the optimization to come up with a smarter motion plan in order to let the point mass jump into the cup. Figure 5.4 shows the comparison between the simulated and the tracked positions of this test. We can see that the robot's end-effector is able to keep up with the simulation well. Looking at the trajectories of the point masses, we can see that our physical system is approximately doing what it is supposed to do, despite the fact that we are dealing with a more chaotic dynamical system. The behavior shows that the two point masses are influencing each other, ending up in a delayed or early reaction in comparison to the simulation. Also, both point masses are slightly oscillating instead of being at rest at the end of the move, which could not be taken into account by our simple model. Nevertheless, we are able to control the system in such a way that the lower point mass jumps into the cup as planned by the optimization.

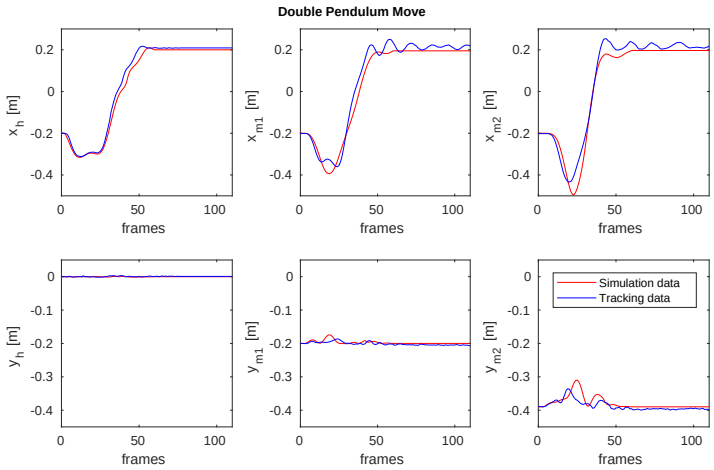

Figure 5.4: Plots of positions of test trajectory 2: Double pendulum - Jump into cup;
Simulation data in red, tracked data of the physical system in blue; $\{x_h, y_h\}$, $\{x_{m1}, y_{m1}\}$ and $\{x_{m2}, y_{m2}\}$ express the positions of the handle and the first and second point mass over time, respectively; The parameters used to generate the results are listed in table 5.1

Test 3: Simplified Marionette in 2D - Waving Motion

In order to test the simplified model of our first marionette prototype, we let the puppet fulfill a waving motion using both arms. Note that in order to control the physical puppet, we attach the robot's end-effectors to the center of the rigid handle connections (as described in section 4.7). In addition to these center position commands, we send orientation commands of the individual handles to the corresponding robot arm.
By looking at the results depicted in figure 5.5, we can see that the puppet is able to fulfil the waving motion. Nevertheless, it is made visible that our physical model does not match the simulated one entirely. There are some modeling errors due to the fact that we are not able to attach the string exactly at the center of mass of the point masses, as required by our model in simulation. This is the reason why the movement of the body in x direction ($x_{m,B}$) shows an oscillating behavior. Furthermore, according the simulation, the point masses should be freely rotatable within the rigid connections. Although we tried to take that into account for our physical puppet, this is not true for the whole range of motion. This is why the the puppets left and right hand are still oscillating a bit at the end of the motion, visible in the plots of $x_{m,HL}$ and $x_{m,HR}$. We can improve this behavior by either taking this into account in our optimization problem, or by designing a puppet that fits the model more accurately.

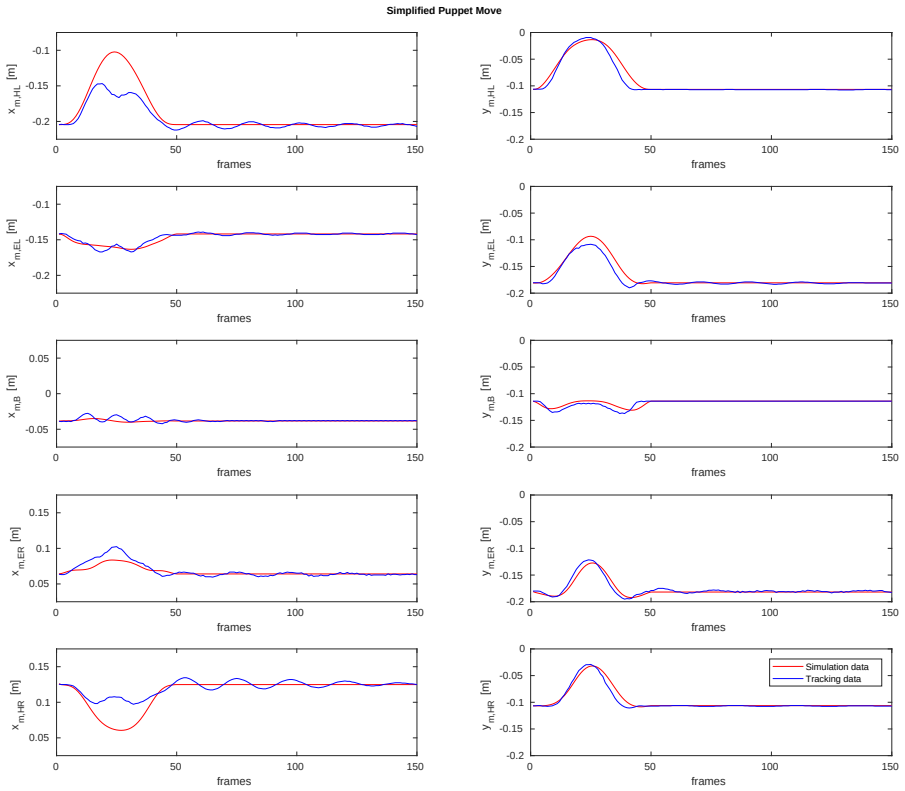

Figure 5.5: Plots of positions of test trajectory 3: Simplified 2D marionette - Waving motion; Simulation data in red, tracking data of the physical system in blue; $\{x_{m,HL}, y_{m,HL}\}$, $\{x_{m,EL}, y_{m,EL}\}$, $\{x_{m,B}, y_{m,B}\}$, $\{x_{m,ER}, y_{m,ER}\}$ and $\{x_{m,HR}, y_{m,HR}\}$ express the positions of the puppet's left hand, left elbow, body, right elbow and right hand, respectively; The parameters used to generate the results are listed in table 5.1

Chapter 6

Conclusion and Outlook

In this thesis, we presented a trajectory planning method for complex dynamical systems by solving nonlinear constrained optimization problems. We developed a model of a simplified planar marionette and showed some planned trajectories for different dynamical systems. Our framework includes system parameter identification and is able to take obstacles and self-collision into account.
To conclude, we can say that our method works accurately for planning trajectories for a large range of different dynamical systems. In our current implemented framework in MATLAB, we have the advantage of being able to directly interact with the optimization parameters such that we can help the process to converge to a desired optimal solution. In the current state of our work, we did not focus on the runtime of the algorithms, which is one reason why - depending on the parameter settings - it can take quite some time until convergence. Another point worth mentioning is that depending on how we set parameters like the number of steps n or the step size Δt, the resulting trajectories can differ from each other. Getting a deeper understanding of how these settings influence the trajectory generation will be an important aspect to consider for future work. This comes in combination with the question what time step Δt we have to choose in order to approximate the system well enough while keeping reasonable calculation times. As there are often several local minima for a particular problem, the initial guesses of the optimization variables play an important role as well.

At this stage of the project, we have completed the "prototype phase". We have shown that the presented method combined with the used robot platform works for the manipulation of dynamical systems in 2D. One of our next steps will be to extend the problem to 3D and take not only point masses but also rigid body dynamics into account. To realize this, we will transfer the optimization environment into C++. This gives us several advantages: First of all, we expect the optimization problem to be solved in a sufficiently accurate way much faster than with MATLAB. This becomes a crucial point in case we want to adjust the trajectory in real time while the robot is already executing a movement. Second, a 3D model of the robot in a physical simulation environment in C++ already exists. It can be used to test the algorithm and the resulting trajectories on the simulated robot beforehand. Additionally, a connection interface to the physical YuMi robot in C++ also already exists. It has been implemented in addition to the work presented in this thesis.
Another step which we want to take is sending joint position commands to the robot instead of end-effector commands. This means that we would not optimize the handle positions of our system, but all of YuMi's joint angles directly. To achieve this, we have to add the kinematic configuration of the robot as constraints into our optimization environment. This gives us the advantage that we won't just find trajectories so that our dynamical system does not collide with an obstacle or with itself, but also that the robot's arms do not collide with each other. We will also be able to tell

beforehand if a trajectory is feasible for the robot to execute, or if it ends up on a collision course or in a singularity. We can get this information without having to rely on the checks executed on the level of the robot controller after the commands have already been sent to it.
Former experiences of our group have shown that these kinds of optimization problems can be solved faster when it is given only objectives instead of both objectives and hard constraints. This is why we are probably going to try to switch all of our hard constraints to soft constraints. One challenge when doing this is that the weights of the individual objectives have to be tuned very carefully in order to achieve the desired results. It has to be made sure that the constraints implied by the physics of the system are sufficiently fulfilled such that it fits its physical counterpart as closely as possible.
An interesting feature to have would be to apply some kind of feedback control. For now, we precompute the trajectories and send them to the robot in a feed-forward way. An improvement of performance in terms of accurate manipulation of the dynamical system would be to add a feedback loop such that the trajectory can be adjusted in real time. For that we would have to add some sensors (like for example tracking cameras or force sensors) to our robot.
We want our focus to remain on working on the puppeteering problem. While building the prototype of our simplified planar marionette, we realized how crucial the design choice for performance is. When designing a puppet, the question arises of how many strings to add and where to put them on the marionette. This is why it would be very interesting to have an incorporated design for the puppet. The underlying question is if it is possible to solve an optimization problem which can tell us the dimensions of the marionette and the locations of the strings in some optimal way. One possible example would be to try to minimize the oscillating movements due to the dynamical system's eigen modes, as these are generally hard to control.

Bibliography

[1] D. Zameroski, G. Starr, J. Wood, and R. Lumia, "Swing-free trajectory generation for dual cooperative manipulators using dynamic programming," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, May 2006, pp. 1997–2003.

[2] M. Bisgaard, J. Bendtsen, and A. L. Cour-Harbo, "Modelling of Generic Slung Load System," in *AIAA Modeling and Simulation Technologies Conference and Exhibit*. American Institute of Aeronautics and Astronautics, Aug. 2006.

[3] M. Bernard and K. Kondak, "Generic slung load transportation system using small size helicopters," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 3258–3264.

[4] J. Schultz and T. Murphey, "Trajectory generation for underactuated control of a suspended mass," in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 123–129.

[5] K. Sreenath, N. Michael, and V. Kumar, "Trajectory generation and control of a quadrotor with a cable-suspended load - A differentially-flat hybrid system," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 4888–4895.

[6] K. Sreenath, T. Lee, and V. Kumar, "Geometric control and differential flatness of a quadrotor UAV with a cable-suspended load," in *52nd IEEE Conference on Decision and Control*, Dec. 2013, pp. 2269–2274.

[7] K. Sreenath and V. Kumar, "Dynamics, Control and Planning for Cooperative Manipulation of Payloads Suspended by Cables from Multiple Quadrotor Robots," Jun. 2013.

[8] S. Tang and V. Kumar, "Mixed Integer Quadratic Program trajectory generation for a quadrotor with a cable-suspended payload," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 2216–2222.

[9] S. Tang, V. Wüest, and V. Kumar, "Aggressive Flight With Suspended Payloads Using Vision-Based Control," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1152–1159, Apr. 2018.

[10] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Learning swing-free trajectories for UAVs with a suspended load," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 4902–4909.

[11] I. Palunko, A. Faust, P. Cruz, L. Tapia, and R. Fierro, "A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 4896–4901.

[12] C. D. Crousaz, F. Farshidian, and J. Buchli, "Aggressive optimal control for agile flight with a slung load," in *in IROS 2014 Workshop on Machine Learning in Planning and Control of Robot Motion*, Aug. 2014.

[13] S. Xing and I.-M. Chen, "Design expressive behaviors for robotic puppet," in *7th International Conference on Control, Automation, Robotics and Vision, 2002. ICARCV 2002.*, vol. 1, Dec. 2002, pp. 378–383 vol.1.

[14] I.-M. Chen, R. Tay, S. Xing, and S. H. Yeo, "Marionette: From Traditional Manipulation to Robotic Manipulation," in *International Symposium on History of Machines and Mechanisms*. Springer, Dordrecht, 2004, pp. 119–133.

[15] K. Yamane, J. K. Hodgins, and H. B. Brown, "Controlling a motorized marionette with human motion capture data," *International Journal of Humanoid Robotics*, vol. 01, no. 04, pp. 651–669, Dec. 2004.

[16] S. Kim, X. Zhang, and Y. J. Kim, "Haptic Puppetry for Interactive Games," in *Technologies for E-Learning and Digital Entertainment*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Apr. 2006, pp. 1292–1302.

[17] E. Johnson and T. D. Murphey, "Dynamic Modeling and Motion Planning for Marionettes: Rigid Bodies Articulated by Massless Strings," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, Apr. 2007, pp. 330–335.

[18] E. R. Johnson and T. D. Murphey, "Automated Trajectory Morphing For Marionettes Using Trajectory Optimization," in *2007 IEEE International Conference on Robotics and Automation*, 2007, p. 6.

[19] P. Martin and M. Egerstedt, "Optimal timing control of interconnected, switched systems with applications to robotic marionettes," in *2008 9th International Workshop on Discrete Event Systems*, May 2008, pp. 156–161.

[20] P. Martin, E. Johnson, T. Murphey, and M. Egerstedt, "Constructing and Implementing Motion Programs for Robotic Marionettes," *IEEE Transactions on Automatic Control*, vol. 56, no. 4, pp. 902–907, Apr. 2011.

[21] T. D. Murphey and M. Egerstedt, "Choreography for Marionettes: Imitation, Planning, and Control," in *IROS*, Nov. 2007, p. 7.

[22] T. D. Murphey and E. R. Johnson, "Control aesthetics in software architecture for robotic marionettes," in *Proceedings of the 2011 American Control Conference*, Jun. 2011, pp. 3825–3830.

[23] MathWorks, "MATLAB GUI - Create Apps with Graphical User Interfaces in MATLAB," https://ch.mathworks.com/discovery/matlab-gui.html, February 2018.

[24] ——, "fmincon - Find minimum of constrained nonlinear multivariable function," https://ch.mathworks.com/help/optim/ug/fmincon.html, February 2018.

[25] R. H. Byrd, M. E. Hribar, and J. Nocedal, "An Interior Point Algorithm for Large-Scale Nonlinear Programming," *SIAM Journal on Optimization*, vol. 9, no. 4, pp. 877–900, Jan. 1999.

[26] P. Wolfe, "Checking the Calculation of Gradients," *ACM Trans. Math. Softw.*, vol. 8, no. 4, pp. 337–343, Dec. 1982.

[27] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1-3, pp. 503–528, Aug. 1989.

[28] ABB, "YuMi - Zusammen in die Zukunft der Automatisierung. You and Me." http://new.abb.com/products/robotics/de/industrieroboter/yumi, February 2018.

[29] ——, "YuMi - Datasheet," https://library.e.abb.com/public/848462e58ba244f38532d23534bb4067/Datenblatt%20YuMi_lowres.pdf, February 2018.

[30] Berkeley Automation, "Berkeley AutoLab yumipy Documentation," https://berkeleyautomation.github.io/yumipy/, February 2018.

[31] Robot Operating System, "ROS," http://www.ros.org/, April 2018.

[32] ABB Robotics, "Technical reference manual - RAPID Instructions, Functions and Data types," https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf, April 2018.

[33] ABB, "YuMi - IRB 14000 Überblick," https://library.e.abb.com/public/bf60a2bd5fcb4d9590555be163d865ab/Produktpraesentation_IRB14000.pdf, February 2018.

[34] PEARL, "Infactory Newtons's Cradle," https://www.pearl.ch/ch-a-PE5234-3422.shtml, March 2018.

[35] stratasys, "ASA," http://www.stratasys.com/de/materials/search/asa, March 2018.

[36] A LEYARD Company, "OptiTrack," http://www.optitrack.com/, March 2018.

[37] Learn OpenCV, "Object Tracking using OpenCV," https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/, April 2018.

[38] MathWorks, "quadprog - Quadratic programming," https://ch.mathworks.com/help/optim/ug/quadprog.html, February 2018.

[39] M. Borneas, "The Lagrange function in a general problem," *Il Nuovo Cimento (1955-1965)*, vol. 16, no. 5, pp. 806–810, Jun. 1960.

[40] P. R. Pagilla and M. Tomizuka, "Contact Transition Control of Nonlinear Mechanical Systems Subject to a Unilateral Constraint," *Journal of Dynamic Systems, Measurement, and Control*, vol. 119, no. 4, pp. 749–759, Dec. 1997.

[41] stack overflow, "Shortest distance between a point and a line segment," https://stackoverflow.com/questions/849211/shortest-distance-between-a-point-and-a-line-segment, April 2018.

[42] ——, "Shortest distance between two line segmentst," https://stackoverflow.com/questions/2824478/shortest-distance-between-two-line-segments, April 2018.

Appendix A

Robot Controller Functions

This chapter gives an overview of the most important functions used to control the ABB robot YuMi. The listed functions are part of the ABB programming language *RAPID* and are running on the YuMi controller. A more detailed description of the individual functions can be found in [32]. The page numbers in the following lists are related to this source. A more detailed description of the functions used to control the grippers can be found in the product manual delivered with the robot.

Movement:

- *MoveL*: used to move the tool center point (TCP) linearly to a given absolute destination; It can also be used to reorientate the tool (p. 264)
- *MoveC*: used to move the TCP circularly to a given absolute destination, while the orientation remains unchanged relative to the circle during the movement (p. 236)
- *MoveJ*: used to move the robot's joints from one point to another; All axes reach the destination position at the same time, which results in a nonlinear path for the end-effector (p. 253)
- *MoveAbsJ*: used to move the robot to an absolute joint configuration; Again, all axes reach the destination at the same time (p. 230)

Request Information:

- *CRobT*: reads the current position and orientation of the robot's end-effector and its axes configuration (p. 807)
- *CalcRobT*: can be used to check if a target end-effector position is reachable before executing the movement (p. 789)
- *CJointT*: reads the current angles of all the joints (p. 800)
- *CalcJointT*: can be used to check if a target joint configuration is reachable before executing the movement (p. 786)

Control Grippers:

- *g_ Init*: used to initialize the gripper by calibrating it and setting its maximum speed and holding force (p. 74)
- *g_ JogIn, g_ JogOut*: used to move the gripper inward or outward until it reaches a mechanical limit or a timeout (p. 76, 77)
- *g_ MoveTo*: used to move the gripper to a specified position (p. 78)
- *g_ GripIn, g_ GripOut*: used to indicate the gripper to grip inward or outward (p. 80, 83)
- *g_ Stop*: used to stop any action of the gripper (p. 87)

Other Useful Functions:

- *ClkReset, ClkStart, ClkStop, ClkRead*: used to measure how long a process takes to be executed (p. 51, 52, 54, 802)
- *SyncMoveOn, SyncMoveOff*: used to execute synchronized movements of both arms of the YuMi (p. 534, 528)
- *NumToStr*: used to convert numeric value to a sting (helpful for socket communication) (p. 904)

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CRL

Master Thesis for Simon Zimmermann

Puppeteering using a two-armed Manipulator Robot

Introduction

Using manipulator robots for tasks like pick-and-place of rigid objects or assembling of different parts is nowadays state of the art in industry. In order to fulfill more complex tasks, it is becoming a requirement for robots to manipulate objects as dexterously as humans do. To have such capabilities, the robots need to possess a deep understanding of the physical world of deformable systems. To start the investigation of this grand challenge, this thesis examines a particular problem, namely puppeteering, which requires expert control of very high-dimensional, underactuated dynamical systems.

Tasks

The purpose of this thesis is to start with a simple dynamical system and increase the complexity towards a model of a puppet.
In particular, the following tasks will be carried out:

- Perform a literature review to get familiarized with the area and find related work
- Review and compare different robot platforms
- Setup best suited robot platform
- Model dynamical systems (with increasing complexity)
- Identify system parameters
- Investigate path planning of the robot's end-effectors, including the control tasks needed to perform them on the platform
- Write thesis

Remarks

This thesis is overseen and supervised by Prof. Dr. Stelian Coros, Computational Robotics Lab.
A written report and an oral presentation conclude the thesis.

Timeline

Issue: 09.10.2017
Submission: 23.04.2018

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Puppeteering using a two-armed Manipulator Robot

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Zimmermann	Simon Andreas

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 23.04.2018	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.