K-Order Markov Optimization and Sequential Convex Programming for Multirotor Motion Planning

by

Welf Rehberg

Student-ID: 378985

Submitted to the Department of Transport and Machine Systems
in partial fulfillment of the requirements for the degree of

Master of Science in Computational Engineering Science

at the

TECHNICAL UNIVERSITY OF BERLIN

November 2022

Statement of Authorship

I hereby declare that the work presented in this thesis, entitled

K-Order Markov Optimization and Sequential Convex Programming for Multirotor Motion Planning,

is entirely my own and that I did not use any sources or auxiliary means other than those referenced.

December 7, 2022, Berlin

Welf Rehberg

K-Order Markov Optimization and Sequential Convex Programming for Multirotor Motion Planning

by

Welf Rehberg

Submitted to the Department of Transport and Machine Systems
on November 16, 2022, in partial fulfillment of the
requirements for the degree of
Master of Science in Computational Engineering Science

Abstract

Multirotors have many exciting applications, including entertainment, cooperative construction, inspection of power lines and off-shore wind parks as well as aerial additive manufacturing. Since the motion planning problem for multirotors is known to be very difficult, approximate solutions are frequently employed. A common approximation is the use of the differential-flatness property, which allows to compute and follow polynomial splines as trajectories. Unfortunately, such a model cannot take the limited motor forces of real multirotors into account and produces conservative motions. Other methods use search- or sampling-based approaches, which are not applicable when considering the full dynamics. In this work, k-order Markov optimization (KOMO) and successive convexification (SCvx) were applied to multirotor motion planning problems considering the full dynamic model and compared extensively. KOMO makes use of the short-term dependency of trajectories to reduce the dimensionality of the optimization problem and has already proven its maturity for robotic manipulation. SCvx, on the other hand, is claimed to be well suited for highly-nonlinear problems of dynamic systems. Both algorithms were used to solve various problem scenarios, including flying in cluttered environments and recovering from upside-down positions. The evaluation results show that KOMO and SCvx have a high success rate for problems requiring solutions close to geometric path constraints (e.g., obstacle constraints). In highly dynamic cases where the solution hits the input constraints, only SCvx finds solutions reliably.

Zusammenfassung

Multikopter haben in den letzten Jahren sowohl für akademische als auch industrielle Anwendungen zunehmend an Bedeutung gewonnen. Sie werden zum Beispiel erfolgreich für Lichtshows, kooperatives Aufbauen von komplexen Strukturen und zur Inspektion von Hochspannungsleitungen und off-shore Windanlagen eingesetzt. Die

Trajektorienplanung für Multikopter gestaltet sich jedoch kompliziert aufgrund ihrer hochgradig nicht-linearen Bewegungsdifferenzialgleichungen und der hohen Anzahl an Optimierungsvariablen. Um diese Probleme zu umgehen, werden häufig Approximationen verwendet. Eine Möglichkeit ist auszunutzen, dass das dynamische Model eines Multikopters differenziell flach ist. Dies erlaubt Flugbahnen direkt als polynomiale Splines vorzugeben. Da dieser Ansatz in der Optimierung die Beschränkungen der Motordrehzahl außer Acht lässt, sind die resultierenden Flugbahnen meist konservativ. Ansätze, die eine probabilistische oder vollständige Repräsentation des kollisionsfreien Raumes erstellen, brauchen meist lange, um eine Lösung zu finden, wenn die nicht-linearen Differenzialgleichungen beachtet werden. Methoden, die die Trajektorienplanung als mathematisches Problem formulieren hingegen sind in der Lage, schnell Lösungen zu finden. In dieser Arbeit werden Markov Optimierung k-ter Ordnung (KOMO) und sukzessive konvexe Approximation (SCvx) als Trajektorienplanungsalgorithmen verwendet. KOMO wird bereits für die Bahnplanung von Industrierobotern erfolgreich eingesetzt, und für SCvx wird davon ausgegangen, dass sich der Algorithmus besonders für nicht-lineare dynamische Systeme eignet. Beide Algorithmen wurden anhand von unterschiedlichen Szenarien evaluiert und es stellte sich heraus, dass KOMO und SCvx eine hohe Erfolgsrate für Probleme aufweisen, bei denen der Fokus auf der Kollisionsvermeidung liegt. Probleme, bei denen die Grenzen der Motordrehzahl voll ausgenutzt werden, sind für KOMO deutlich schwieriger zu lösen. SCvx hingegen findet zuverlässig Lösungen für diese Probleme.

First Thesis Examiner : Prof. Dr. Marc Toussaint
Title: Head of Learning and Intelligent Systems Laboratory

Second Thesis Examiner : Prof. Dr. Oliver Brock
Title: Head of Robotics and Biology Laboratory

Thesis Supervisor: Dr. Wolfgang Hönig
Title: Independent Junior Research Group Leader

Thesis Supervisor: Joaquim Ortiz-Haro
Title: PhD Student

Acknowledgement

I would like to thank Dr. Wolfgang Hönig and Joaquim Ortiz-Haro for their guidance and support throughout this thesis. Especially for their patience during the many fruitful discussions we had. I would also like to thank the entire LIS team at TU-Berlin for their constant support and the countless joyful lunch breaks. Last but not least, I would like to thank Arved Wintzer and Sebastian Gerstberger for proofreading parts of this work.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ACADO Automatic Control and Dynamic Optimization

BVP Boundary Value Problem

CHOMP Covariant Hamiltonian Optimization for Motion Planning

CoG Center of Gravity

DDP Differential Dynamic Programming

FCL Flexible Collision Library

iLQR Iterative Linear Quadratic Regulator

KOMO K-Order Markov Optimization

NLP Non-Linear Problem

PRM Probabilistic Roadmap

RAI Robotic AI

RRT Rapidly exploring Random Tree

SCP Sequential Convex Programming

SCvx Successive Convexification

SLP Sequential Linear Programming

SQP Sequential Quadratic Programming

STOMP Stochastic Trajectory Optimization for Motion Planning

SUAV Small Unmanned Aerial Vehicle

Chapter 1

Introduction

In recent years, multirotors have risen in popularity in academia and industry due to their exceptional agility while being technically simple aerial vehicles. Multirotors are successfully applied in various fields e.g., entertainment [1], cooperative construction [2], inspection of power lines [3] and off-shore wind parks [4] and aerial additive manufacturing [5]. Many of these applications require highly autonomous behavior, raising the need for motion planners that can reliably solve problems requiring complex trajectories. While the mechanical structure of multirotors is simple, controlling them is challenging due to nonlinear dynamics, complex aerodynamic effects, and actuation constraints. To plan their movements according to limitations imposed by their dynamics and surrounding, modern motion planners have to solve complex problems. While sampling- and search-based approaches to motion planning have strong theoretical guarantees regarding completeness, optimization-based methods often provide speed advantages or better quality of the found solutions and will be used throughout this work. Since the motion planning problem is notoriously difficult to solve, approximate solutions are frequently applied. A common approximation is the use of the differential-flatness property of the multirotor model, which allows to compute and follow splines as trajectories. Unfortunately, such a model cannot take the limited motor forces of a real multirotor into account and produces conservative motions. In this work, two algorithms are considered that find solutions making use of the full dynamics of the model. While successive convexification (SCvx) is claimed to perform well for highly nonlinear problems for dynamic systems, k-order Markov optimization (KOMO) has proven its maturity for robotic manipulation. This thesis aims to evaluate the performance and suitability of KOMO and SCvx for problems related to multirotor motion planning. Therefore, throughout the work, KOMO and SCvx are extensively compared.

Contributions:

In this thesis, the following scientific contributions are presented:

1. Extensive experimental evaluation, discussion and comparison of the perfor-

mance of k-order Markov optimization and successive convexification for multirotor motion planning

2. Implementation and validation of a successive convexification algorithm using Python

The thesis is structured as follows. First, a short introduction to multirotors and their dynamic model is given in chapter 2. Subsequently, related work in the field of motion planning is presented, where sampling-, search- and optimization-based methods are discussed. In chapter 3, the concrete motion planning problem is introduced and the used optimization methods are presented. Additionally, details on implementing the motion planning problem in the presented algorithmic frameworks are given. The experimental results are presented and discussed in chapter 4 and the work is summarized in chapter 5.

Chapter 2

Background

The following chapter briefly introduces multirotors and the foundations of trajectory generation.

2.1 Multirotor

Throughout this section, multirotors are compared to other aerial vehicles and the dynamics of multirotors are described.

2.1.1 Small Unmanned Aerial Vehicles

Common small unmanned aerial vehicles (SUAV) are divided into three different classes depending on the type of lift generation and the arrangement of the lift generating structures [6]:

1. Fixed-wing aircraft,
2. Single rotor blade helicopter,
3. Multirotor.

The three different types of SUAV are shown in figure 2-1. In fixed-wing aircrafts [6], the lift-generating structures (wings) are firmly attached to the vehicle's airframe. The airflow generates lift over the special shape of the wings. In order to generate a lift that counteracts the gravitational force of the vehicle, the vehicle must maintain a certain airspeed. To achieve this speed, an additional propulsion system is needed. To take off or land, the aircraft must accelerate to or decelerate from take-off speed, which prevents a vertical landing or take-off. On the other hand, the advantage of the fixed-wing structure is its energy efficiency and high possible payload.

Single-bladed helicopters [6], on the other hand, can generate lift directly through rotors. The lift-generating structure is not fixed to the airframe and can rotate relative to it. The lift is, therefore, not dependent on the vehicle's translational speed but on the rotor's rotational speed, which allows the aircraft to take off without airspeed (Vertical Take-Off and Landing) and hover in place. Due to the air resistance of the rotor, additional structures are needed to compensate for the torque generated by the air resistance. The overall complexity of a helicopter with only one rotor blade is higher than that of a fixed-wing aircraft and that of a multirotor and therefore has higher maintenance costs.

Multirotors [6] are helicopters with three or more rotors used to generate lift. The lift-generating structures are, therefore, not attached to the vehicle's airframe. By using multiple rotors, the torque generated by the drag of the rotors can be balanced by opposite rotation directions. The simple structure of the multirotor results in high maneuverability, the ability to hover in place and a simple control strategy using only the rotor speeds as control inputs. This is an advantage when it comes to solving highly dynamic problems. In terms of payload, energy efficiency and maintenance costs due to their low complexity, multirotors are a good compromise between helicopters with one rotor blade and fixed-wing aircrafts. Another useful property of multirotors is redundancy for rotor numbers greater than four, leading to advantages regarding safety aspects. Therefore, multirotors are well-suited for a variety of tasks.

Figure 2-1: Different types of SUAV: fixed-wing aircraft (left) [7], single rotor blade helicopter (middle) [8], multirotor (right) [9].

2.1.2 Dynamic Model of a Multirotor

In this section, the dynamic model of a multirotor will be described in detail. The presented model will later be used to formulate dynamic constraints for the trajectory optimization problem.

Figure 2-2 shows the schematics of a multirotor for the example of 4, 6 and 8 rotors. Here ω_i describes the rotational speed of the rotor. To counteract the torque caused by the air resistance of the rotor, the motors next to each other rotate in opposite directions. The distance of the rotors to the center of gravity (CoG) is determined by the arm length l and the angular offset from the i-th arm to x_B is described by β_i. Two right-handed coordinate systems are used: the inertial system $\mathcal{F}_\mathcal{I} : \{x_I, y_I, z_I\}$, which refers to the origin of the flying volume, and the body system $\mathcal{F}_\mathcal{B} : \{x_B, y_B, z_B\}$, which describes the reference of the multirotor. Here z_I is permanently aligned with gravity

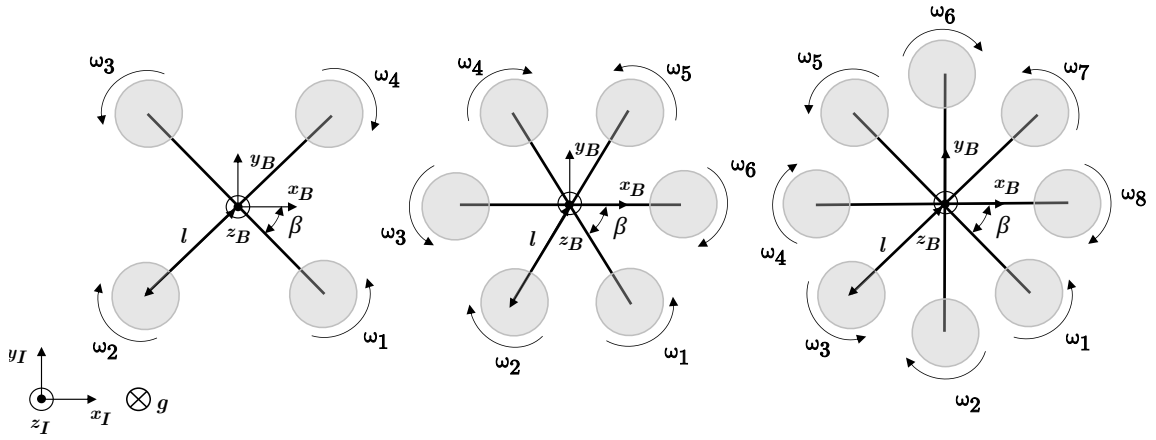


Figure 2-2: Multirotor model with different rotor numbers.

$g = [0, 0, 9.81m/s^2]^T$ and z_B is permanently aligned with the direction of the combined thrust f_T generated by the rotors. To describe the state of the multirotor at time t the position of the CoG $p(t) \in \mathbb{R}^3$, velocity $v(t) \in \mathbb{R}^3$, unit quaternion rotation $q(t) \in \mathbb{H}$ (parametrising the rotation matrix $\mathbf{R}(q)$) and the rotational velocity in the body frame $\omega_B(t) \in \mathbb{R}^3$ are expressed in the state vector $x(t) = [p(t), v(t), q(t), \omega_B(t)]^T \in \mathbb{R}^{13}$. Where $p(t)$ and $q(t)$ together represent transformations of $\mathcal{F}_\mathcal{I}$ to $\mathcal{F}_\mathcal{B}$ in $SE(3)$. The quaternion orientation representation has several advantages, such as solving the gimble-lock problem of the Euler angle representation [6] and being numerically more stable [10] than the representation with rotation matrices. The dynamic model of the multirotor is obtained by applying the Newton-Euler equations (2.1) for rigid bodies with 6 degrees of freedom (DoF) [11]. This results in the following relations:

$$\begin{pmatrix} \mathbf{R}(q) f_T \\ \tau \end{pmatrix} = \begin{pmatrix} mI_3 & 0 \\ 0 & \mathbf{J} \end{pmatrix} \begin{pmatrix} \dot{v} \\ \dot{\omega}_B \end{pmatrix} + \begin{pmatrix} -g \\ \omega_B \times \mathbf{J}\omega_B \end{pmatrix}. \tag{2.1}$$

Here $\mathbf{J}$ denotes the time constant inertia matrix of the entire multirotor, m its mass and τ the acting torque. The dynamic equations of the multirotor can therefore be written as:

$$\dot{p} = v \tag{2.2}$$

$$\dot{v} = \frac{1}{m}\mathbf{R}(q) f_T + g \tag{2.3}$$

$$\dot{q} = \frac{1}{2} q \otimes \begin{pmatrix} 0 \\ \omega_B \end{pmatrix} \tag{2.4}$$

$$\dot{\omega}_B = \mathbf{J}^{-1}(\tau - \omega_B \times \mathbf{J}\omega_B). \tag{2.5}$$

Where the derivative of the quaternion $\dot{q}$ is taken from [10] and $\otimes$ denotes the quaternion multiplication.

The total thrust vector f_T and the acting torque τ result from the multirotor geometry and the acting forces generated by the motors as follows:

$$f_T = \begin{pmatrix} 0 \\ 0 \\ \sum_i^{n_m} f_i \end{pmatrix} \qquad (2.6) \qquad \tau = \begin{pmatrix} l \sum_i^{n_m} \sin(\beta_i) f_i \\ l \sum_i^{n_m} \cos(\beta_i) f_i \\ \kappa \sum_i^{n_m} \alpha_i f_i \end{pmatrix}. \qquad (2.7)$$

The effective torque around z_B is caused by the rotor air resistance and is coupled to the acting motor forces by the torque constant κ. Depending on the rotor's rotation direction, the rotor resistance torque acts in a positive or negative direction. This is taken into account by α_i, which is either +1 or -1 depending on the rotor. The maximal combined thrust is defined by the thrust-to-weight ratio $r_{t2w} = f_{T,max}/m$. The total number of forces depends on the type of the multirotor and is described by the number of motors n_m. The forces f_i are correlated to the rotor speed by the thrust coefficient κ_f according to:

$$f_i = \kappa_f \omega_i^2, \qquad (2.8)$$

where ω_i is the rotation speed of the i-th rotor.

2.2 Motion Planning

Throughout the next section, common approaches for motion planning will be presented. Since there is no clear guideline for planning problems terminology, the following classification will be used. Geometric motion planning usually describes planning considering the workspace limitations and task constraints such as initial and final configuration. Kinodynamic motion planning or trajectory planning, on the other hand, also considers differential and other constraints of the system. Optimal control extends the planning problem to search for optimal system inputs that consider certain constraints and lead to a feasible trajectory. In the course of this work, no distinction is made between trajectory planning and optimal control problems, and both terms refer to planning problems, including inputs. Usually, planning algorithms aim not only to find a feasible solution but also to find an optimal solution with respect to a certain objective. Optimal trajectory planning in general aims for solving an optimal control problem by searching for an optimal control input trajectory $u^*(t) : \mathbb{R} \to \mathbb{R}^{n_u}$, corresponding optimal state trajectory $x^*(t) : \mathbb{R} \to \mathbb{R}^{n_x}$ and optimal time-invariant parameters $p^* \in \mathbb{R}^{n_p}$ (e.g. final time for free-final-time problems). Where the goal is to optimize an objective with respect to a set of constraints [12] on a defined time interval $t \in [0, T]$. The general optimization problem can then be formulated as follows:

$$\min_{x(t),u(t),p} \quad J(x(t),u(t),p) \tag{2.9}$$

$$\text{subject to} \quad \dot{x}(t) = f(x(t),u(t),p),\ \forall t \in [0,T] \tag{2.10}$$

$$h_{ic}(x(0),p) = 0,\ h_{tc}(x(T),p) = 0 \tag{2.11}$$

$$g_p(x(t),u(t),p) \leq 0,\ \forall t \in [0,T]. \tag{2.12}$$

Here $x(t) : \mathbb{R} \to \mathbb{R}^{n_x}, u(t) : \mathbb{R} \to \mathbb{R}^{n_u}, p \in \mathbb{R}^{n_p}$ describe the state trajectory, input trajectory and the time-invariant parameters. The general optimal control problem contains the objective function $J : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}$ (2.9) which should be minimized and the constraints. Usually, constraints are divided into dynamic constraints $f : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}$ (2.10), initial constraints $h_{ic} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_{ic}}$ and final constraints $h_{tc} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_{tc}}$ (2.11) and path constraints $g_p : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_g}$ (2.12). While the dynamic constraints ensure that the resulting trajectory respects the differential equations representing the dynamic model, the initial and final constraints represent requirements such as fixed initial and final configurations. Path constraints, on the other hand, enforce collision avoidance or restrict states and inputs to a certain set.

These problems are known to be at least PSPACE hard to solve, high dimensional, and often non-convex [13]. The most common approaches to solving trajectory generation problems rely on either building graphs or search trees or directly following the gradients of the optimization problem. These approaches lead to three classes of algorithms: optimization-based, search-based and sampling-based. These will be discussed in detail in the following.

An essential property of a motion planner is completeness. Therefore completeness and its weaker forms will be defined briefly. A planner is considered *complete* if it returns a solution, if one exists, in a finite amount of time and returns a failure otherwise. For *resolution completeness*, a planner has to be able to find a solution to the spatially discretized version of the problem and otherwise return that no solution exists. The discretization of the problem introduces a resolution up to which problems can be solved. *Probabilistic completeness* is given if the probability of a planner finding an existing solution converges to one.

2.2.1 Optimization-Based Trajectory Generation

Optimization-based trajectory planning algorithms aim to find a state-control sequence that minimizes a certain cost function locally. Since they have significant computational speed advantages, they are increasingly used as stand-alone algorithms even though they are prone to local minima. Especially in aerospace [14, 15] and robotics applications [16, 17, 18], purely optimization-based algorithms are widely used for trajectory planning. Optimization-based algorithms aim to solve the continuous-time optimal control problem (2.9) - (2.12). In general, existing ap-

proaches can be divided into direct and indirect methods. While indirect approaches first analytically derive the necessary optimality conditions and then discretize these to find a root that solves the problem, direct methods first discretize and then solve the nonlinear problem (NLP) [19] by minimizing the objective function. Although this work will be focused on direct methods for solving NLPs, indirect methods will also be briefly discussed.

Indirect Methods

Indirect methods derive a boundary value problem (BVP) using the necessary optimality conditions of the continuous-time problem derived from Pontryagin's maximum principle. The resulting system of ordinary differential equations is, in general, solved numerically [20]. However, since differential equations are often highly nonlinear and unstable, they are often difficult to solve in practice [21].

Direct Methods

Direct methods, on the contrary, formulate a tractable optimization problem that can be solved directly [21]. Since the original problem formulation (2.9) - (2.12) is an infinite dimensional optimization problem and therefore intractable, the continuous dynamics, controls and cost function have to be discretized to apply numerical optimization methods. Resulting in the following discrete optimization problem:

$$\begin{aligned}
\min_{x,u,p} \quad & J(x,u,p) && (2.13)\\
\text{subject to} \quad & x_{k+1} = \hat{f}(x_k, u_k, p), \ \ \forall k \in \{0,...,N\} && (2.14)\\
& h_{ic}(x_{k=0}) = 0, \ \ h_{tc}(x_{k=N}) = 0 && (2.15)\\
& g_p(x_k, u_k, p) \leq 0, \ \ \forall k \in \{0,...,N\}. && (2.16)
\end{aligned}$$

Here, the state and input constraints are often non-convex, and the system dynamics are often nonlinear. In general, this problem formulation leads to high-dimensional and non-convex problems, which result in high computational complexity and loss of guarantees for obtaining a solution [22]. For this, various methods for solving the discretized nonlinear optimization problem have been developed using different problem formulations.

For example, methods such as CHOMP [17] and STOMP [23] have proven their maturity for solving a wide variety of motion planning problems. For CHOMP, the cost function consists of two parts: the first part measures dynamic quantities of the trajectory, while the second part deals with obstacle constraints by applying potential functions based on distance fields. The update of the discrete configurations is then calculated via covariant gradient descent, making the update invariant to the trajectory parameterization. Therefore, the gradients of the resulting cost function

must be available and smooth. While the original implementation could not handle additional constraints, the extension described in [16] was able to incorporate a wider variety of constraints into the planning process. To overcome weak local minima, [17] incorporates gradient information into Monte Carlo sampling to make the planner probabilistically complete ([16]). For updating the configurations along the trajectory, STOMP [23] uses a similar approach but extends the method to arbitrary cost functions by not relying on analytical gradients. Instead, [23] uses a stochastic estimation of the gradient and performs stochastic gradient descent. This results in STOMP being able to escape local minima more efficiently due to the stochastic nature of its update rule. Since both methods only treat the configurations as optimization variables, the inputs have to be calculated using the system's inverse kinematics.

KOMO [18] and SCP [12], have shown their significant potential in the last years. SCP methods, are more general NLP solvers and used in a variety of different optimal control problems [14, 24], but do not make use of the special structure and properties of these problems, whereas KOMO can exploit these efficiently. Toussaint [18], like Kalakrishnan et al. [23] and Ratliff et al. [17], represents the trajectory only in configuration space and deals with dynamics by imposing constraints on consecutive configurations. The resulting optimization problem is then solved by classical optimization algorithms such as Gauss-Newton, augmented Lagrangian and log-barrier. KOMO exploits the resulting structure of the Jacobian of the cost function to efficiently compute the pseudo-Hessian and store it in a matrix representation beneficial for further calculations [18]. SCP implementations like SCvx and GuSTO [12], on the other hand, are more general and can also treat velocities and accelerations as decision variables. Therefore, they are well suited for arbitrary constraints and cost functions. Since SCP and KOMO are used extensively in this thesis, they are explained in detail in section 3.2 to 3.4.

Differential dynamic programming (DDP) [25] and iterative LQR (iLQR) [26] are also widely used in robotic optimal control [27, 28, 29]. Although these methods are fast and have small memory requirements, they are less well suited for considering nonlinear state and input constraints [29].

Application for Multirotors

Although the aforementioned algorithms are widely used in Robotics, only SCP [12, 30], DDP [27] and iLQR [31, 32] are actually used for multirotor trajectory generation and only DDP methods and implementations using generic NLP-solvers (e.g. ACADO [33]) consider the full dynamics of the system and non-convex constraints.

2.2.2 Trajectory Generation for Flat Systems

A common method for generating trajectories for multirotors is to leverage the system dynamics instead of simply imposing them as constraints, making use of the

differential flatness property of the full dynamics of the multirotor [34, 35, 36]. These approaches are discussed in detail as a special case of optimization-based trajectory generation, which is important for multirotors. In system theory, a dynamic system is called differentially flat if a set of outputs exists such that all states and inputs of the system can be fully described by smooth functions of these outputs and their time derivatives [37]. Here, the flat output's dimensionality must match the original input's dimensionality. The trajectory in the flat output is defined by:

$$\psi(t) = h(x(t), u(t), \dot{u}(t), ..., u^{(p)}(t)) \in \mathbb{R}^{n_u} \tag{2.17}$$

and is called flat if the smooth functions g_x and g_u exist such that:

$$x(t) = g_x(\psi(t), \dot{\psi}(t), ..., \psi^{(p)}(t)) \in \mathbb{R}^{n_x} \tag{2.18}$$

$$u(t) = g_u(\psi(t), \dot{\psi}(t), ..., \psi^{(p)}(t)) \in \mathbb{R}^{n_u}. \tag{2.19}$$

Taking advantage of this property, a reference trajectory can be planned in output space while the actual states and inputs are obtained by mapping the flat output through g_x and g_u. For multirotors, the planning of the reference trajectory can be done for every component independently. Where the flat output becomes $\psi(t) = [x, y, z, \phi]^T \in \mathbb{R}^4$, with x, y, z being the positional components and ϕ being the yaw angle. Generating the reference trajectory in output space can be simplified by using parametric splines. For example, using piecewise polynomials [36, 38] or defining spline curves as the linear combination of smooth basis functions, e.g. B-spline basis functions [39] or Bernstein polynomials [40]. These segments are parameterized by a vector of control points $\mathbf{P} = [p_0, p_1, ..., p_n] \in \mathbb{R}^{n_u \times (n+1)}$. Here the number of control points is determined by the desired degree n of the resulting spline. For a spline segment using smooth basis functions $b_{i,n} : \mathbb{R} \to \mathbb{R}$, this can be written as follows:

$$\psi(t) = \sum_{i=0}^{n} p_i b_{i,n}(t) = \mathbf{P} B_n(t),\ t \in [0, 1]. \tag{2.20}$$

With $B_n(t) \in \mathbb{R}^n$ being the vector concatenating all basis functions up to degree n. When using Bezier curves as basis functions, the resulting spline has the following useful properties [41]:

1. *Convex Hull Property*: $\psi(t)$ lies completely in the convex hull of the control points $p_0, ..., p_n$ (control polygon). This is illustrated by figure 2-3.

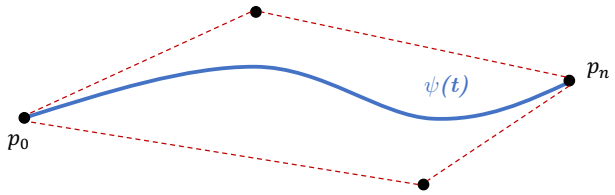


Figure 2-3: Convex hull of the control points $p_0, ..., p_n$.

2. C^n *Smoothness*: all derivatives of $\psi(t)$ up to degree n are smooth if $\psi(t)$ is composed of basis functions of degree n.

3. *End Point Interpolation*: the curve at the first time point and last time point coincides with the first and the last control points

$$\psi(0) = p_0 \ , \ \psi(1) = p_n. \tag{2.21}$$

To construct the trajectory planning problem, a vector of $N+1$ waypoints $W = [w_0, ..., w_N]$ and timestamps $T = [t_0, ..., t_N]$ is defined through which the trajectory must pass (e.g. initial state and final state). These waypoints are connected by N different splines parameterized by N different vectors of control points and knots. The degree n of the basis functions must be chosen such that the highest order derivative appearing in g_x and g_u is still smooth (property 2). The trajectory connecting all waypoints is thus composed of the following:

$$\begin{aligned} &\psi_1(t) = \mathbf{P_1} B_{1,n}(t - t_0), && t_0 \leq t \leq t_1 \\ &\psi_2(t) = \mathbf{P_2} B_{2,n}(t - t_1), && t_1 \leq t \leq t_2 \\ &\vdots \\ &\psi_N(t) = \mathbf{P_N} B_{N,n}(t - t_{N-1}), && t_{N-1} \leq t \leq t_N. \end{aligned} \tag{2.22}$$

The original planning problem (2.9) - (2.12) then translates to:

$$\min_{P_1,...,P_N} \quad J(\psi(t)^{(d)}, T) \tag{2.23}$$

$$\text{subject to} \quad \psi_i(t_i) = \psi_{i+1}(t_i) = w_{i-1}, \ \ i = 1, ..., N-1 \tag{2.24}$$

$$\psi_N(t_N) = w_N \tag{2.25}$$

$$\psi_1(t_0) = w_0 \tag{2.26}$$

$$\psi_i^{(p)}(t_i) = \psi_{i+1}^{(p)}(t_i), \ \ i = 1, ..., N-1, p = 0, ..., n. \tag{2.27}$$

There (2.24) to (2.26) ensure that the defined way points are passed and (2.27) guarantees that the combined trajectory is C^n continuous.

The cost function J normally penalizes the square of the derivatives of $\psi(t)$, e.g., minimizing the snap of the trajectory in the flat output [36], which derivatives to incorporate into the cost function is determined by the mappings g_x and g_u and the desired states or inputs that should be penalized. This leads to a quadratic optimization problem with a global optimum that can be solved in polynomial time [38]. However, this scales poorly with increasing numbers of spline segments and control points. For polynomial splines, a possible solution is proposed by Richter et al. [38]. The problem is reformulated as an unconstrained quadratic optimization problem, optimizing over the endpoint derivatives instead of polynomial coefficients.

Time Optimization

Since the execution time is crucial for trajectory generation, following Richter et al. [38], a possible method for minimizing flight time for polynomial splines is to adapt the problem formulation by introducing the segment times as optimization variables and adding the weighted sum to the cost function as follows:

$$\tilde{J}(\psi(t)^{(d)}, T) = J(\psi(t)^{(d)}, T) + k_T \sum_{i=1}^{N} t_i. \tag{2.28}$$

Here, the weight k_T influences the trade-off between minimizing the trajectory snap and the execution time. Note that this is still a quadratic optimization problem.

Collision Avoidance

Additionally, it is often required to calculate collision-free trajectories. Since, for multirotors, the position is part of the flat output, a constraint limiting the trajectory to free space can be enforced by making use of the convex hull property of splines composed of Bezier basis functions and B-splines. Therefore, a set of convex obstacles $\mathcal{O} = \{O_1, ..., O_{no}\}$ is defined and the trajectory is considered collision-free if for each Obstacle $O_l \in \mathcal{O}$ and any time point $t \in [t_0, t_N]$ the condition $r(t) \notin O_l$ holds. Here $r(t) \in \mathbb{R}^3$ is the part of the trajectory $\psi(t)$ that represents only the position in Cartesian coordinates. Using the separating hyperplane theorem, the constraint can be formulated as follows: there has to exist a separating hyperplane between the control polygons of all sub-splines and any convex obstacle. This was, e.g., used by Stoical et al. [39] for B-splines. Compared to other formulations, it is therefore not necessary to make sure that the trajectory is also collision free between discretized time steps. For polynomial splines, Richter et al. [38] proposes a method where additional waypoints are added to make colliding sub-splines feasible.

Input Constraints

Since the inputs are not freely selectable in reality but are restricted by constraints, the trajectory in the output space must be mapped onto a feasible set of inputs. Since the mapping from the trajectory in output space to input is generally nonlinear, this would lead to non-convex constraints, losing the advantage of a quadratic optimization problem. Therefore, Richter et al. [38] propose to first optimize the ratio of segment times independently. Since the optimal ratio between the sub-intervals is invariant to the total time, the segment times can be scaled in a separate optimization problem until the input constraints are satisfied, where increasing the total time T used causes the inputs to converge to the hover state of the multirotor. However, this leads to conservative trajectories because the input constraints do not influence the positions of the control points.

Application for Multirotors

Differential flatness-based approaches are commonly used for multirotor trajectory generation, for example, in [42, 36, 34]. Here the use of the flatness property can lead to conservative motions since the input constraints are not handled explicitly in the optimization.

2.2.3 Search-Based Trajectory Generation

Especially for trajectory planning in cluttered environments, search-based approaches are used to find feasible trajectories [43],[44] since they provide strong theoretical properties such as resolution completeness [43]. To generate a searchable graph, the configuration space is discretized by a state lattice discretization that already incorporates the system dynamics. The relative positions of the discrete states s with respect to each other are defined by motion primitives. Motion primitives, in general, define sequences of states and actions for a defined time period that are consistent with the system's dynamics and have an associated cost. These motion primitives describe the transition from one state to a set of possible successor states. The motion primitives e are constructed in two ways. Either by applying a constant input u_m to the system for a short period of time τ, where the input is chosen from the discretization $\mathcal{U}_M = \{u_1, ..., u_M\}$ of the control set $\mathcal{U}$ [43], or by using an inverse planner to connect two neighboring nodes of the state lattice [45]. To search an already piecewise optimal graph, the computation of the motion primitive must itself be an optimization problem that guarantees that the trajectory between two successive states is optimal with respect to the chosen cost function. For motion primitives calculated by applying a constant input this can be done as follows. For multirotors, the optimization problem can be simplified by taking advantage of the differential flatness property of the system and planning with independent polynomials ($p(t) \in \mathbb{R}^3$) [43] as state trajectories in output space. For the system in output space, the

control input can then be chosen as any derivative of the polynomial trajectory (e.g., the jerk $\dddot{p}(t)$) with respect to which the trajectory should be optimal. Since the system's dynamics become linear in output space, they can be written in state space form, which can be used to calculate the optimal successor state given an applied input and a cost function that penalizes the input. The linear dynamics in state space form are given by:

$$\psi(t) = [p(t)^T, \dot{p}(t)^T, ..., p^{(n-1)}(t)^T]^T \in \mathbb{R}^{3n} \tag{2.29}$$

$$u(t) = p^n(t) \tag{2.30}$$

$$\dot{\psi}(t) = \mathbf{A}\psi(t) + \mathbf{B}u(t). \tag{2.31}$$

Where $\mathbf{A} \in \mathbb{R}^{3(n+1)\times 3(n+1)}$ denotes the state matrix and $\mathbf{B} \in \mathbb{R}^{3(n+1)\times 3}$ the input matrix. The optimal trajectory connecting a state and its successor state in the flat output space can then be obtained using the standard solution for a linear, time-invariant system:

$$\psi(t) = \underbrace{e^{\mathbf{A}t}}_{\mathbf{F}(t)} y_0 + \underbrace{\left[\int_0^t e^{\mathbf{A}(t-\sigma)}\mathbf{B}d\sigma\right]}_{\mathbf{G}(t)} u_m. \tag{2.32}$$

That the obtained solution is optimal with respect to the chosen cost function is shown by Liu et al. [43]. Moreover, since the duration of the motion primitive is constant, the cost of each primitive can be calculated in advance. An example of how the transition from a state with a non-zero velocity to its successor states may look is shown in figure 2-4, where the red squares denote the system states, the pink lines denote the motion primitives and the black arrows denote the applied elements of the discretized control set $\mathcal{U}_M$.

Searchable Graph

To build a searchable graph, the output state space is mostly discretized in two ways [45]. Firstly, a grid discretization can be chosen directly. Secondly, discretizing the control set leads to an implicit state space discretization.

To build the searchable graph directly on a discretized grid, an inverse planner has to be invoked to plan the trajectories connecting an initial state with its neighbors. If the discretization is regular, the obtained motions can be applied to any node to connect it with its neighbors [45]. To store all states of a possibly infinite lattice imposes significant memory requirements [45]. Therefore Pivtoraiko and Kelly proposes using a minimal set of feasible motions, from which all possible paths can be obtained by concatenation. This set is obtained by calculating the motions to each node in

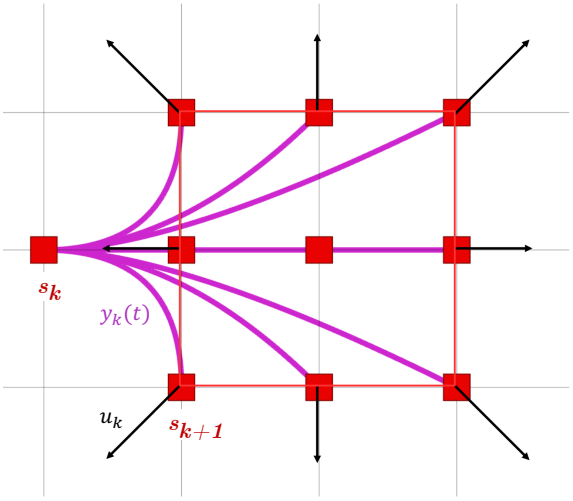


Figure 2-4: Transition of an initial state with nonzero velocity to possible successor states defined by motion primitives (source: figure 3 (a) in [43]).

a limited radius around the initial state while excluding connections that can be composed of already obtained motions. During the search, only the minimal subset of motions is considered.

To build a searchable graph using the discrete control set, the state space is explored by starting with the initial state and calculating the motion primitives that lead to the successor states. For each successor state, this procedure is repeated until a node of the state lattice lies within the goal region $\mathcal{X}_{goal}$. Changing the terminal constraint from an exact goal state to a goal region in state space is necessary since the state lattice has a finite resolution determined by the duration τ of the motion primitives. Therefore, the duration must be chosen such that the resolution of the state lattice is smaller than the actual goal region [44].

The obtained states and motion primitives have to be checked for collisions and state constraints feasibility. Enforcing state constraints can be done by computing the maximum and minimum of the associated derivatives over the trajectory connecting two consecutive states [43]. Enforcing obstacle constraints is more difficult as it theoretically requires a collision check between the swept volume of the multirotor along the trajectory and each obstacle, i.e., ensuring that $x_k(t) \in \mathcal{X}_{free}$. Here $\mathcal{X}_{free}$ denotes the collision-free subset of the state space. This can be simplified by approximating the shape of the multirotor as an ellipsoid and representing the obstacles as point clouds. Intermediate positions are then sampled along the motion primitive, at which it is checked whether any points lie within the ellipsoid [44]. A different approach is to check traveled and occupied grid cells [43].

Since all nodes lie in feasible regions of the subspace and all graph edges consist of optimal trajectories, the original planning problem (2.9) - (2.12) translates to [43]:

$$\min_{N, u_{0:N-1}} \quad (\sum_{k=0}^{N-1} ||u_k||^2 + \rho N)\tau \tag{2.33}$$

$$\text{subject to} \quad \psi_k(t) = \mathbf{F}(t)s_k + \mathbf{G}(t)u_k, \;\; t \in [0, \tau] \tag{2.34}$$

$$\psi_k(t) \subset \psi_{free}, \;\; \forall k \in \{0, ..., N-1\}, t \in [0, \tau] \tag{2.35}$$

$$s_{k+1} = x_k(\tau), \;\; \forall k \in \{0, ..., N-1\} \tag{2.36}$$

$$s_0 = \psi_0, \;\; s_N \in \psi_{goal} \tag{2.37}$$

$$u_k \in \mathcal{U}_M, \;\; \forall k \in \{0, ..., N-1\}. \tag{2.38}$$

Where $N\tau$ additionally penalizes the overall flight time and ρ determines the weighting between input cost and time cost. Note that the flatness property is assumed and $\psi_k(t)$ is in flat output space. The stated problem can be solved by graph search. Where the searchable graph $\mathcal{G}(\mathcal{S}, \mathcal{E})$ is composed of the set of reachable states $\mathcal{S}$ and the set of motion primitives $\mathcal{E}$ connecting these states.

Heuristic Function

Graph search problems can be solved efficiently with informed search algorithms like A*. To efficiently guide the graph search, an approximation of the real cost-to-go function has to be found that assigns an approximation of the cost-to-go to each node. Liu et al. [43] propose two candidates that meet the requirements. An intuitive solution for a heuristic that only considers time as cost-to-go is to calculate the minimum possible traveling time for the distance from the state to the goal region. Since this heuristic discards the control effort, which is part of the original objective, Liu et al. provide a more sophisticated heuristic derived from a relaxed version of problem 2.8. Here, the input and obstacle constraints are removed and the problem becomes a quadratic optimization problem with an analytical solution and a closer approximation of the real cost.

Trajectory Refinement

Guaranteeing the smoothness of the trajectory in higher dimensions introduces higher dimensional states (position, velocity, acceleration, jerk, ...) and, thus, a longer planning time. This can be avoided by planning in lower dimensions and refining the trajectory around the solution found. Liu et al. propose to either solve an unconstrained quadratic problem [43], keeping the previously found waypoints and time steps the same, or resolve the graph search problem in higher dimensions and incorporating the previously found solution into a new heuristic function [44]. The final trajectory obtained by the first method is not guaranteed to be collision-free and feasible any longer. For the second method, the new heuristic is not admissible anymore since it is not necessarily an underestimate of the cost-to-go and, therefore,

the resulting trajectory will not be optimal.

Application for Multirotors

Search-based motion planning methods for multirotors are, e.g., applied in [43] and [44]. However, the need to reduce the dimensionality of the state lattice requires the use of approximations using the differential flatness property leading to the mentioned limitations.

2.2.4 Sampling-Based Trajectory Generation

Sampling-based motion planning algorithms are considered practical solutions to a variety of motion planning problems due to their theoretical properties regarding probabilistic completeness [46] and the fact that they scale well (still exponentially) with high dimensions in configuration space (C-space) [47].

These algorithms explore the free C-space of a dynamic system by generating and connecting samples to find a feasible path between an initial and a goal configuration. Sampling-based methods can be divided into multi-query and single-query planners. Multi-query planners aim to represent the connectivity of the free C-space as a searchable graph that can be used to solve multiple motion planning problems efficiently. These approaches are useful when numerous tasks need to be performed in the same environment without changing the position and shape of the obstacles. A single-query planner, on the other hand, connects only an initial and a goal configuration [48]. The most common algorithms are based on the different variants of rapidly exploring random trees (RRT) and probabilistic roadmaps (PRM). Since RRTs and PRMs are widely used, their basic operating principle is described in detail in the following sections.

Probabilistic Roadmap Planner

Probabilistic roadmap methods [49] are multi-query planners that operate in two phases: exploration and exploitation phase. In the exploration phase, a probabilistic roadmap is built by sampling configurations and connecting them into a searchable graph. The algorithm starts with an empty graph $\mathcal{G}(\mathcal{S}, \mathcal{E})$ and then draws samples c, checking each sample for feasibility regarding collisions. Each feasible sample is added to the nodes $\mathcal{S}$, and a fast local planner is used to connect them to their nearest neighbors. If the planner finds a feasible path that connects the sample to one of the nodes s of the roadmap, the edge (c, s) is added to $\mathcal{E}$. Since connecting a sampled configuration to the existing graph is computationally costly, it is important to choose wisely when to connect a sample. For example, samples in a connected part of the free C-space are not as useful for representing the connectivity as samples in narrow passages in between connected parts [50]. Generating "useful" samples can be

done by exploiting the knowledge about the configuration space obtained during the sampling procedure [50, 51, 52]. During the exploitation phase, the generated graph is used to find feasible paths between two configurations by connecting them to the graph and searching for the shortest path from the initial to the goal configuration. Regarding completeness, this algorithm is considered probabilistic complete [46].

Rapidly Exploring Random Trees

Methods that use rapidly exploring random trees [47] are considered single query planners. Unlike PRM methods, RRT methods do not aim to generate a representation that can be exploited but connect an initial configuration directly to a goal configuration/region. The RRT algorithm starts with the initial configuration and samples in each iteration a new configuration. The tree's growth is guided by the so-called Voronoi bias and favors extending in unexplored regions [47], leading to an equal exploration of the C-space. In the next step, the algorithm aims to connect the obtained sample to the existing tree by extending the nearest node in the direction of the sample. Once the tree has reached the goal region, the algorithm terminates and returns the resulting tree. This algorithm is considered probabilistic complete [46]. A notable extension that significantly increases the performance of the original version is using two separate rapidly exploring random trees, one growing from the initial and one from the goal configuration [53]. This can be useful to escape so-called bug trap scenarios significantly faster.

Kinodynamic Planning

Up to this point, discussed RRT and PRM versions only find paths without considering the system's dynamics. Since non-holonomic systems cannot follow arbitrary paths, kinodynamic methods have been derived that are able to incorporate the system dynamics [47]. For RRT's, LaValle already addressed this problem in his original RRT implementation by extending a node of a tree in a dynamically feasible fashion. Connecting a node to a sample or extending it to a particular point in C-space while respecting differential constraints leads to a BVP that is difficult or impossible to solve [48]. Therefore, the new node can be obtained by propagating the dynamics $\dot{x} = f(x, u)$ for a randomly sampled time Δt for different inputs u and choosing the configuration that gets closest to the actual sample. This leads to a tree where dynamically feasible edges connect all nodes. PRM's result in a graph representation of the free space where each node has multiple connections. Since propagating the node could only be done for one connection, kinodynamic PRM-methods require solving multiple BVPs to obtain feasible connections.

Optimal Planning

Up to this point, all the described sampling-based methods either have no guarantees regarding optimality or lead to suboptimal solutions. For RRT, it has even been proven that the algorithm converges to a suboptimal solution with probability 1 [46]. A common method for optimizing a path tries to connect two non-consecutive nodes along the path with a feasible link, shortcutting the path at this point [49]. Since this does not guarantee optimality, Karaman and Frazzoli propose variants of PRMs and RRTs (PRM*, RRT*) that are shown to be asymptotically optimal and computationally efficient. Since certain PRM versions are already asymptotically optimal [46], PRM* becomes computationally efficient by limiting the number of connections from each new node according to the total number of nodes in the graph. RRT* extends the original version by connecting a new node not to its nearest neighbor, but to the neighbor that minimizes a given cost function. Afterward, the graph is rewired so that the added node is chosen as the parent for all neighboring nodes, resulting in a lower-cost path.

Application for Multirotors

For the trajectory planning of multirotors, kinodynamic RRT* implementations are often used [54, 55]. These methods either use the linearized dynamics of the multirotor model or make use of the differential flatness property. This is necessary to ensure computational efficiency, as RRT* versions are used that connect the tree to a newly obtained sample instead of just propagating a node in the direction of the sample. This may lead to suboptimal solutions for highly aggressive maneuvers as the real dynamic could deviate strongly from the linearization.

2.2.5 Discussion

Throughout the following section, the methods introduced up to this point will be discussed to motivate the choice of optimization-based methods for motion planning. The properties regarding completeness, optimality and run time of the discussed algorithms are presented in table 2.1. Sampling- and search-based methods are especially well suited for problems with a low dimensional state space and scenarios where gradient information alone will seldom guide to feasible solutions. Such scenarios can be, e.g., "bug-trap" type of problems, where random exploration for finding a solution or the complete exhaustion of all possible solutions is necessary. Since multirotor applications seldom have these properties and, in most cases, do not require the globally optimal solution, optimization-based methods are applicable. Furthermore, optimization-based methods show several significant benefits, such as speed advantages for high dimensional state spaces, and their optimality is not just asymptotically or tied to a state lattice resolution. Therefore optimization-based methods were used throughout this work.

	Completeness	**Optimality**	**Computation Time**
Sampling	Probabilistically Complete	Asymptotically Optimal	does not scale well with state space dimensions (exponential)
Search	Resolution Complete	Globally Optimal in Resolution	does not scale well with state space dimensions (exponential)
Optimization	Incomplete	locally Optimal	scales well with state space dimensions (polynomial)

Table 2.1: Comparison of motion planning approaches.

Chapter 3

Approach

In this chapter, the used approaches are described and their implementations are presented.

3.1 Formulation of the Multirotor Motion Planning Problem

The following section describes the concrete problem formulation for multirotor motion planning. For the dynamic model, two different levels of approximation were used. The first being the full nonlinear dynamics (3.2) already introduced in section 2.1.2 with state x_{nl} and control input u_{nl}. The second is a double integrator model (3.3), representing the multirotor as a single point with state x_l including position and velocity and with control input u_l being the acceleration of the multirotor. These approaches lead to the following states and inputs:

$$x_{nl} = \begin{pmatrix} p \\ v \\ q \\ \omega \end{pmatrix} \in \mathbb{R}^{13}, u_{nl} = \begin{pmatrix} f_0 \\ \vdots \\ f_{n_m} \end{pmatrix} \in \mathbb{R}^{n_m}, x_l = \begin{pmatrix} p \\ v \end{pmatrix} \in \mathbb{R}^6, u_l = a \in \mathbb{R}^3 \quad (3.1)$$

and the dynamic equations:

$$\dot{x}_{nl} = \begin{pmatrix} v \\ \frac{1}{m}\mathbf{R}(q)f_T + g \\ \frac{1}{2}q \otimes \begin{pmatrix} 0 \\ \omega_B \end{pmatrix} \\ J^{-1}(\tau - \omega_B \times J\omega_B) \end{pmatrix} \quad (3.2) \qquad \dot{x}_l = \begin{pmatrix} v \\ a \end{pmatrix} \quad (3.3)$$

$$\dot{x}_{nl} = f_{nl}(x, u) \tag{3.4}$$

$$\dot{x}_{l} = f_{l}(x, u). \tag{3.5}$$

Using the double integrator model is a common approximation of the dynamics for multirotor trajectory generation (e.g., [12]) and can be used to generate more accurate reference trajectories than produced by naive approaches like straight line interpolation.

The motion planning problem considered in this work places the following requirements on the obtained solution:

1. the multirotor trajectory starts in an initial state x_0 and ends in a final state x_T,
2. the whole trajectory $x(t)$ is collision-free, meaning that the multirotor approximated by the sphere $S(t)$ does not intersect with the obstacles $O_i \in \mathcal{O}$, $i \in \{1, ..., n_{obs}\}$ at any time point,
3. the multirotor movement corresponds to the dynamic equations (3.2) and (3.3),
4. the maximum and minimum position (p_{max}) / velocity (v_{max}) / orientation (q_{max}) / rotational velocity ($\omega_{B,max}$) / input ($f_{i,max}$ or a_{max}) is not exceeded,
5. the trajectory minimizes control effort i.e. minimizes $J(u) = \int_0^T \|u(t)\|^2 dt$,

leading to the following problem formulation for the two models:

$$\begin{aligned}
\min_{x(t),u(t)} \quad & J(u(t)) && (3.6)\\
\text{subject to} \quad & \dot{x}(t) = f_{nl/l}(x(t), u(t)) && (3.7)\\
& S(t) \cap O_i = \emptyset, \forall i \in \{1, ..., n_{obs}\} && (3.8)\\
& x(0) = x_0, x(T) = x_T && (3.9)\\
& x(t) \leq x_{max}, x(t) \geq x_{min} && (3.10)\\
& u(t) \leq u_{max}, u(t) \geq u_{min}. && (3.11)\\
& && (3.12)
\end{aligned}$$

3.2 Sequential Convex Programming (SCP)

Throughout this section, SCP-methods and their advantages, guarantees and limitations will be described to motivate the use as a baseline to compare KOMO against. Sequential convex programming methods approximate the original non-convex optimization problem iteratively as a convex sub-problem updating the approximation

according to newly obtained sub-problem solutions. Modern SCP methods are considered fast, flexible and efficient local optimization algorithms for trajectory generation. Probably the first SCP method sharing the structure of modern methods was sequential linear programming SLP [12]. Which has grown in popularity, especially for large-scale problems [56], due to well matured linear problem solvers such as the Simplex-Algorithm. Over time constraining the convex sub-problems to SLPs has become unnecessary since solvers for more general problems were sufficiently advanced. Nevertheless, linear approximations are still accurate enough for various optimization problems and are often used in practice. Since their gain in popularity in the 1970s, sequential quadratic programming SQP algorithms are arguably the most mature class of SCP algorithms and comparably well analyzed (e.g., [57]). Even though they benefit from their close relation to Newton Methods and the fact that the initial guess does not have to be feasible, they have clear downsides, such as the need to reliably estimate the Hessian of the non-convex program and requiring affine constraints in the solution variable (trajectory generation problems often contain non-affine constraints such as second-order cone constraints) [12]. These limitations are overcome by methods using more general convex problems, such as sequential semi-definite programming, which will be described in detail later. These methods are widely used in a variety of engineering domains, among others in aerospace [24], mechanical design [58] and power grid technology [59]. An application worth mentioning is the use of SCP-algorithms in the New Shephard rocket of NASA and BlueOrigin [12]. Following the extension by Malyuta et al. of the hierarchy of optimization algorithms originally proposed by Boyd and Vandenberghe, SCP-algorithms can be placed at the top of standard optimization algorithms in terms of complexity, since they use complex algorithms such as interior point methods as an inner loop for solving convex sub-problems.

As already mentioned, all SCP implementations revolve around iteratively solving convex approximations of the non-convex problem around an updated initial guess. Malyuta et al. describe the general procedure as shown in figure 3-1 and point out different properties and advantages, which will be discussed briefly. To solve the non-convex trajectory optimization problem, SCP is initialized with the problem formulation P defined by the objective function and the constraints and the initial guess x_{init}. To create a convex sub-problem, the non-convex parts of the problem definition are convexified (1.). Approximating the non-convex problem can lead to an unbounded or infeasible sub-problem (a more detailed explanation follows in 3.2.1), which has to be relaxed to guarantee its feasibility (2.). Then the continuous time problem is discretized (3.) and subsequently solved (4.). If the obtained solution has converged SCP stops and returns the optimal solution x_{opt}. If the obtained solution has not converged (5.), the algorithm updates the trust region according to an update rule. It uses the found solution as an initial guess for the next iteration (6.). This is repeated until the solution has converged. One major advantage of SCP-methods is that they are agnostic to the choice of the convex solver in (4.). Therefore the solver can be

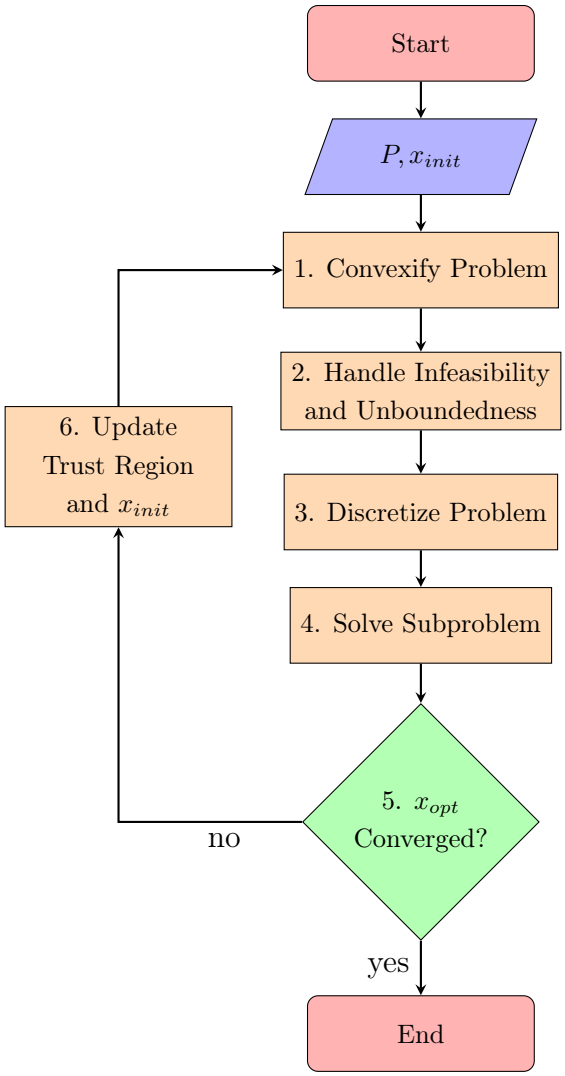


Figure 3-1: SCP procedure (based on [12], figure 11).

chosen from a variety of mature, application-ready packages such as Gurobi[1], ECOS[2] or MOSEK[3]. Another useful property of SCP-methods is that they have meaningful theoretical guarantees regarding algorithmic complexity and performance, in contrast to general NLP-optimization [12].

3.2.1 Artificial Unboundedness and Infeasibility

Convexifying constraints can render the originally feasible problem infeasible by making it unbounded or lead to an empty set of feasible points regarding the convex constraints (figure 3-2 right-hand side). In figure 3-2 on the left-hand side, the possible

[1]https://www.gurobi.com/
[2]https://github.com/embotech/ecos
[3]https://www.mosek.com/

unboundedness of the convex approximation is visualized. The originally non-convex objective is shown in blue, while the dashed orange line describes the convex approximation. It becomes clear that in the non-convex case the optimal point x^* lies in the right valley of the objective, while in the convex case the objective function $J(x)$ becomes unbounded below. Artificial unboundedness can be addressed by introducing a trust region limiting the area in which the approximation is assumed accurate. However, limiting the region in which the optimizer finds feasible solutions can, in a similar fashion as approximating the non-convex constraints, lead to artificial infeasibility. The figure in the middle of figure 3-2 shows how the convex approximation can render an originally feasible problem infeasible. Here the equality constraints are represented by the blue and green line, the inequality constraint is described by the red shaded area and the convexified equality constraint is represented by the orange dashed line. Linearizing the non-convex blue equality constraint leads to a shift of the intersection of the equality constraints into the infeasible region of the inequality constraint, rendering the problem infeasible. On the right-hand side, it is shown how constraining the validity of the approximation to a trust region can make a problem infeasible. Here the introduced trust region constraint (green) does not cover the intersection of the equality constraints (blue and orange). The problem is therefore artificially infeasible. To tackle infeasibility introduced by the convexification of the problem, SCP methods relax the problem guaranteeing that the convex sub-problem is feasible. How this is done in particular can change from implementation to implementation.

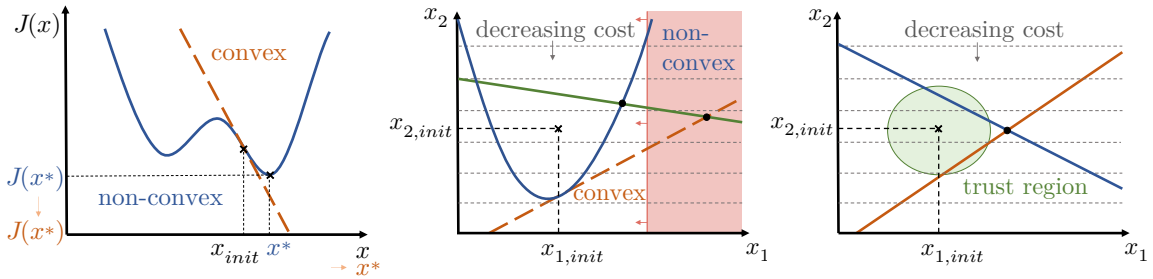


Figure 3-2: Visualization of artificial unboundedness (left-hand side) and artificial infeasibility (middle, right-hand side (based on [12] figure 13)).

3.2.2 Successive Convexification (SCvx)

This section will describe the SCP-algorithm implemented as a baseline for benchmarking the KOMO-framework. Throughout this work sequential convexification (SCvx) is used as variant of SCP. The algorithm was first introduced by Mao et al. [61], rigorously analyzed regarding convergence behavior in [62] and compared to similar SCP-methods in [12]. Since the authors only use first-order approximations of the non-convex problem, the algorithm can be seen as part of the SLP-methods. Compared to general nonlinear optimizers, SCvx has a series of solid theoretical guarantees regarding its convergence behavior and does not assume the feasibility of the initial

guess [62]. Compared to SQP-algorithms, which have, at best linear convergence behavior with perfect hessian approximation, SCvx is guaranteed to have a superlinear convergence behavior and does not require computationally expensive techniques for approximating the hessian and making sure that it is positive semi-definite [61] since the hessian is not required.

This chapter's notation and problem formulation mostly follow the work of Malyuta et al. on the SCvx-algorithm. To simplify the notation later on, the path constraints in the general trajectory planning problem (2.9)-(2.12) are split up into the convex sets $\mathcal{X} \subseteq \mathbb{R}^{n_x}$ and $\mathcal{U} \subseteq \mathbb{R}^{n_u}$ and in non-convex constraints represented by the continuously differentiable function $s : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_s}$. Deviating from the algorithm proposed by Malyuta et al. the constraints $h_{ic}(x(0)) = 0, h_{tc}(x(T)) = 0$ are assumed to already be convex, since a wide variety of trajectory generation problems can be formulated completely relying on convex initial and final constraints. Without loss of generality, the problem is formulated on the time interval $[0, 1]$, since the final time can be incorporated into the parameter vector p. The resulting optimization problem has the following form:

$$\begin{aligned}
\min_{x(t),u(t),p} \quad & J(x(t), u(t), p) && (3.13)\\
\text{subject to} \quad & \dot{x}(t) = f(x(t), u(t), p), \ \ \forall t \in [0, 1] && (3.14)\\
& h_{ic}(x(0)) = 0, \ \ h_{tc}(x(1)) = 0 && (3.15)\\
& s(x(t), u(t), p) \leq 0, \ \ \forall t \in [0, 1] && (3.16)\\
& (x(t), p) \in \mathcal{X}(t), (u(t), p) \in \mathcal{U}(t), \ \ \forall t \in [0, 1]. && (3.17)
\end{aligned}$$

Here the objective function is assumed to be in Bolza form [63]:

$$J(x, u, p) = \phi(x(T), p) + \int_0^T \Gamma(x(t), u(t), p)dt. \tag{3.18}$$

With the terminal cost $\phi : \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}$ and the running cost $\Gamma : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_p} \to \mathbb{R}$. Note that the objective is already assumed to be convex since all non-convex terms in the cost can be reformulated and offloaded into the constraints [12]. To handle the challenges mentioned in section 3.2.1, the following algorithmic choices were made following [12]:

Variable Scaling:

Optimization variables with different magnitudes can lead to numerical issues during optimization (i.e., bad quality of the optimal solution or slow convergence). Therefore Malyuta et al. propose a method that shifts and scales the decision variables x, u, p such that all their elements take values in the same interval $[0, 1]$. This leads to the following affine transformation with x', u', p' being the new scaled variables:

$$x = \mathbf{S}'_x x + c_x, \tag{3.19}$$
$$u = \mathbf{S}'_u u + c_u, \tag{3.20}$$
$$p = \mathbf{S}'_p p + c_p. \tag{3.21}$$

Where S_x, S_u, S_p and c_x, c_u, c_p are matrices and vectors with appropriate content.

Constraint Convexification:

The originally non-convex problem is linearized to obtain a convex sub-problem. While first-order approximations have several advantages, using first-order approximations is less accurate than higher-order approximations. In the SCvx-Algorithm, the introduced linearization error is adequately managed by the trust region update [62] that is described in detail later. To obtain the first Taylor series approximation, the Jacobians of h_{ic}, h_{tc}, s and f must be calculated around the reference solution $(\bar{x}, \bar{u}, \bar{p})$. These are defined as follows [12]:

$$\mathbf{A}(t) \triangleq \nabla_x f(\bar{x}(t), \bar{u}(t), \bar{p}) \tag{3.22}$$
$$\mathbf{B}(t) \triangleq \nabla_u f(\bar{x}(t), \bar{u}(t), \bar{p}) \tag{3.23}$$
$$\mathbf{F}(t) \triangleq \nabla_p f(\bar{x}(t), \bar{u}(t), \bar{p}) \tag{3.24}$$
$$\mathbf{C}(t) \triangleq \nabla_x s(\bar{x}(t), \bar{u}(t), \bar{p}) \tag{3.25}$$
$$\mathbf{D}(t) \triangleq \nabla_u s(\bar{x}(t), \bar{u}(t), \bar{p}) \tag{3.26}$$
$$\mathbf{G}(t) \triangleq \nabla_p s(\bar{x}(t), \bar{u}(t), \bar{p}). \tag{3.27}$$

This leads to the following approximations of the non-convex constraints:

$$f(x(t), u(t), p) \approx \mathbf{A}(t)x(t) + \mathbf{B}(t)u(t) + \mathbf{F}(t)p + r_f(t) \tag{3.28}$$
$$s(x(t), u(t), p) \approx \mathbf{C}(t)x(t) + \mathbf{D}(t)u(t) + \mathbf{G}(t)p + r_s(t). \tag{3.29}$$

To simplify the notation, the quantities r_f and r_s are introduced as:

$$r_f(t) = f(\bar{x}(t), \bar{u}(t), \bar{p}) - \mathbf{A}\bar{x}(t) - \mathbf{B}\bar{u}(t) - \mathbf{F}\bar{p} \tag{3.30}$$
$$r_s(t) = s(\bar{x}(t), \bar{u}(t), \bar{p}) - \mathbf{C}\bar{x}(t) - \mathbf{D}\bar{u}(t) - \mathbf{G}\bar{p}. \tag{3.31}$$

Relaxation of the Linearized Problem:

Since the linearization can render the original problem infeasible (as described in chapter 3.2.1), a relaxed version of the convex problem is formulated following [12]. Therefore virtual control terms ($\nu_d : \mathbb{R} \to \mathbb{R}^{n_x}, \nu_s : \mathbb{R} \to \mathbb{R}^{n_s}, \nu_{ic} \in \mathbb{R}^{n_{ic}}, \nu_{tc} \in \mathbb{R}^{n_{tc}}$) are added to the path, initial, final and dynamic constraints. If the linearized constraints now lead to an infeasible optimization problem, the constraint violation

can be "absorbed" by the introduced buffer. Since these constraints are no longer strictly enforced, SCvx can converge to a non-feasible solution regarding the linearized constraints. Even though this is a known limitation of SCP-methods in general [12], it can often be overcome by slightly tuning the algorithm parameters based on insight about the problem gained from the magnitude of the constraint violations. Since the initial and final constraints are already convex, they do not need to be relaxed. Still, experience has shown that introducing virtual control terms for them also has numerical advantages. For example, even a slight violation irrelevant to real-world applications can give the optimizer more leeway to find a feasible solution. The relaxed linearized constraints then have the following form [12]:

$$\dot{x}(t) = \mathbf{A}(t)x(t) + \mathbf{B}(t)u(t) + \mathbf{F}(t)p + r_f(t) + \nu_d(t) \tag{3.32}$$
$$\mathbf{C}(t)x(t) + \mathbf{D}(t)u(t) + \mathbf{G}(t)p + r_s(t) + \nu_s(t) \leq 0 \tag{3.33}$$
$$h_{ic}(x(0)) + \nu_{ic} = 0 \tag{3.34}$$
$$h_{tc}(x(T)) + \nu_{tc} = 0. \tag{3.35}$$

To discourage constraint violations the $\nu_d, \nu_s, \nu_{ic}, \nu_{tc}$ are penalized in the objective function. Where $\lambda \in \mathbb{R}$ with $\lambda > 0$ is a user-defined weight and must be chosen sufficiently large. The objective function, including the introduced slack variables, results in the following:

$$\begin{aligned} J_\lambda(x, u, p, \nu_d, \nu_s, \nu_{ic}, \nu_{tc}) = \phi(x(T), p) + \lambda(\|\nu_{ic}\|_1 + \|\nu_{tc}\|_1) \\ + \int_0^T \Gamma(x(t), u(t), p) + \lambda(\|\nu_d(t)\|_1 + \|\nu_s(t)\|_1)dt. \end{aligned} \tag{3.36}$$

Trust Region Formulation:

A key element of SCP-methods is to make sure that the obtained solution is feasible with respect to the original non-convex constraints. Therefore the obtained approximation has to be sufficiently accurate, which is closely related to the proximity of the reference solution and the obtained solution. To make sure that the newly obtained solution is sufficiently close to the reference solution, the following trust region constraint is introduced [12]:

$$\|x(t) - \bar{x}(t)\|_\infty + \|u(t) - \bar{u}(t)\|_\infty + \|p - \bar{p}\|_\infty \leq \eta, \tag{3.37}$$

with η being the trust region radius.

Discretization:

To turn the relaxed and linearized problem into a finite-dimensional optimization problem that can be solved numerically, it has to be discretized on the interval $[0, 1]$. Since SCvx works with arbitrary discretization schemes, the method can be selected freely. In this work, a forward Euler integration scheme was chosen. This leads to the following optimization problem:

$$\min_{x,u,p,\nu_d,\nu_s,\nu_{ic},\nu_{tc}} \quad L_\lambda(x, u, p, \nu_d, \nu_s, \nu_{ic}, \nu_{tc}) \tag{3.38}$$

$$\text{subject to} \quad x_{k+1} = \mathbf{A}_k x_k + \mathbf{B}_k u_k + \mathbf{F}_k p + r_{f,k} + \nu_{d,k} \tag{3.39}$$

$$\mathbf{C}_k x_k + \mathbf{D}_k u_k + \mathbf{G}_k p + r_{s,k} + \nu_{s,k} \leq 0 \tag{3.40}$$

$$(x_k, p) \in \mathcal{X}_k, (u_k, p) \in \mathcal{U}_k \tag{3.41}$$

$$h_{ic}(x(0)) + \nu_{ic} = 0 \tag{3.42}$$

$$h_{tc}(x(N)) + \nu_{tc} = 0 \tag{3.43}$$

$$\|x_k - \bar{x}_k\|_q + \|u_k - \bar{u}_k\|_q + \|p - \bar{p}\|_q \leq \eta, \tag{3.44}$$

incorporating the convexified constraints (3.28) - (3.29), trust region constraints (3.37) and virtual control terms (3.32) - (3.35) applied on the discrete nodes $k \in \{0, ..., N-1\}$ [12] and the discrete objective function:

$$\begin{aligned} L_\lambda(x, u, p, \nu_d, \nu_s, \nu_{ic}, \nu_{tc}) = \phi(x_N, p) + \lambda(\|\nu_{ic}\|_1 + \|\nu_{tc}\|_1) \\ + \Delta t \sum_{k=0}^{N-1} \Gamma(x_k, u_k, p) + \lambda(\|\nu_{d,k}\|_1 + \|\nu_{s,k}\|_1). \end{aligned} \tag{3.45}$$

Update of the Trust Region and the Reference Solution:

Since the trust region constraint limits the progress that can be made in one iteration, the trust region radius greatly affects the trade-off between the speed at which the algorithm converges and the accuracy of the obtained solution with respect to the original constraints. The accuracy of the approximation can change depending on the reference solution. Therefore, the trust region is iteratively updated based on the accuracy of the convex approximation, as described by Malyuta et al. A valid metric to quantify the approximation accuracy is comparing the approximated cost improvement and the actual non-convex cost improvement in one iteration. To calculate the optimal value of the non-convex problem the violations of the non-convex constraints for each node $\delta_d \in \mathbb{R}^{N-1 \times n_x}, \delta_s \in \mathbb{R}^{N \times n_s}, \delta_{ic} \in \mathbb{R}^{n_{ic}}, \delta_{tc} \in \mathbb{R}^{n_{tc}}$ have to be calculated. Where δ_{ic} and δ_{tc} can be obtained by mapping the solution of the convex problem x^*, u^*, p^* through the convex initial and terminal constraints. The violations of the non-convex path constraints can be obtained similarly by clipping the values of s such that only the violation of the inequality constraint remains:

$\boldsymbol{\rho \leq \rho_0}$:	$\boldsymbol{\rho_0 \leq \rho \leq \rho_1}$:	$\boldsymbol{\rho_1 \leq \rho \leq \rho_2}$:	$\boldsymbol{\rho_2 \leq \rho}$:
$\eta \leftarrow \max(\eta_{min}, \frac{\eta}{\beta_{sh}})$	$\eta \leftarrow \max(\eta_{min}, \frac{\eta}{\beta_{sh}})$	$\eta \leftarrow \eta$	$\eta \leftarrow \min(\eta_{max}, \beta_{gr}\eta)$
$\bar{x}, \bar{u}, \bar{p} \leftarrow \bar{x}, \bar{u}, \bar{p}$	$\bar{x}, \bar{u}, \bar{p} \leftarrow x^*, u^*, p^*$	$\bar{x}, \bar{u}, \bar{p} \leftarrow x^*, u^*, p^*$	$\bar{x}, \bar{u}, \bar{p} \leftarrow x^*, u^*, p^*$

Table 3.1: Update of the trust region and reference solution based on the approximation accuracy.

$$\delta_{s,k} = \max(0, s(x_k^*, u_k^*, p_k^*)). \tag{3.46}$$

To obtain the violations of the dynamic constraints δ_d, the successor state for the first $N-1$ nodes has to be calculated by propagating u_k^* through the nonlinear dynamics starting at x_k^*. The obtained state $\hat{x}_{k+1}^*$ can then be used to calculate the difference to the optimal solution of the approximated problem x_{k+1}^*. Propagating the control input through the nonlinear dynamics is done using a forward Euler integration scheme as follows:

$$\hat{x}_{k+1}^* = x_k^* + f(x_k^*, u_k^*, p^*)\Delta t \tag{3.47}$$

$$\delta_{d,k} = x_{k+1}^* - \hat{x}_{k+1}^*. \tag{3.48}$$

The obtained non-convex constraint violations are used to define the nonlinear augmented cost function to calculate the value of the non-convex problem as:

$$\begin{aligned}\mathcal{J}_\lambda(x, u, p) = \phi(x_N, p) + \lambda(\|\delta_{ic}\|_1 + \|\delta_{tc}\|_1) \\ + \Delta t \sum_{k=0}^{N-1} \Gamma(x_k, u_k, p) + \lambda(\|\delta_{d,k}\|_1 + \|\delta_{s,k}\|_1).\end{aligned} \tag{3.49}$$

Using $\mathcal{J}_\lambda$, the proposed accuracy metric $\rho \in \mathbb{R}$ is defined by the ratio between the approximated improvement ΔL_λ of the reference solution to the new solution and the actual improvement $\Delta \mathcal{J}_\lambda$ as:

$$\rho = \frac{\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p}) - L_\lambda(x^*, u^*, p^*, \nu_d^*, \nu_s^*, \nu_{ic}^*, \nu_{tc}^*)}{\mathcal{J}_\lambda(\bar{x}, \bar{u}, \bar{p}) - \mathcal{J}_\lambda(x^*, u^*, p^*)} = \frac{\bar{\mathcal{J}}_\lambda - L_\lambda^*}{\bar{\mathcal{J}}_\lambda - \mathcal{J}_\lambda^*} = \frac{\Delta L_\lambda}{\Delta \mathcal{J}_\lambda}. \tag{3.50}$$

According to this metric the trust region and the reference solution is updated as described in table 3.1 [12].

The thresholds $\rho_0, \rho_1, \rho_2 \in \mathbb{R}$ with $0 < \rho_0 < \rho_1 < \rho_2 < 1$, the parameter $\beta_{gr}, \beta_{sh} \in \mathbb{R}$ with $\beta_{sh}, \beta_{gr} \geq 1$ and the minimal and maximal trust region $\eta_{min}, \eta_{max} \in \mathbb{R}$ are user-defined and control the update behavior of the algorithm. If the approximation accuracy is deemed poor, i.e., smaller than a threshold ρ_0, the new solution is rejected and the trust region is shrunk by β_{sh}. The reference solution is replaced with the new optimal solution for an accuracy above ρ_0 and below ρ_1. With an accuracy above ρ_1 and below ρ_2, the trust region is kept the same and above ρ_2, the trust region will be increased by β_{gr}.

Stopping Criterion:

Determining if the obtained solution is sufficiently close to a local optimum is done by monitoring the convergence of the predicted cost improvement and the convergence of the optimal state trajectory and optimal parameters (x^*, p^*). The second criterion was proposed by Malyuta et al., deviating from the original implementation of Mao et al., to avoid unnecessary conservative behavior for problems where large changes in the control trajectory imply only small changes in the state trajectory. If either the change in the cost improvement or the change in the optimal trajectory from iteration to iteration falls under the user-defined threshold ($\epsilon \in \mathbb{R}, \epsilon > 0$ for predicted improvement and $\epsilon_t \in \mathbb{R}, \epsilon_t > 0$ for trajectory change), the algorithm stops. The introduced criteria then have the following form:

$$\frac{\Delta L_\lambda}{L_\lambda^*} \leq \epsilon \tag{3.51}$$

$$\|p^* - \bar{p}\|_\infty + \max_{k \in \{1,\dots,N\}} \|x_k^* - \bar{x}_k\|_\infty = \Delta p_\infty + \Delta x_{\infty,max} \leq \epsilon_t. \tag{3.52}$$

Algorithm:

The algorithm resulting from combining the aforementioned algorithmic choices is shown in algorithm 1.

Algorithm 1 SCvx Algorithm

procedure SCvx($\bar{x}, \bar{u}, \bar{p}, \lambda, \rho_0, \rho_1, \rho_2, \beta_{sh}, \beta_{gr}, \eta_{min}, \eta_{max}$)

Given: Reference solution $\bar{x}, \bar{u}, \bar{p}$; weight for virtual control λ;
thresholds for the approximation accuracy ρ_0, ρ_1, ρ_2;
parameter for shrink and growth rate β_{sh}, β_{gr};
initial, minimal and maximal trust region $\eta, \eta_{min}, \eta_{max}$

while True **do**

  $x^*, u^*, p^* \leftarrow$ SolveProblem($\bar{x}, \bar{u}, \bar{p}$) ▷ solve problem (eq. 3.38 - 3.43)

  $\Delta\mathcal{J}_\lambda, \Delta L_\lambda \leftarrow$ ComputeImprovement($x^*, u^*, p^*, \bar{x}, \bar{u}, \bar{p}$)

  if $\Delta L_\lambda / L_\lambda^* \leq \epsilon$ or $\Delta p_\infty + \Delta x_{\infty,max} \leq \epsilon_t$ **then** ▷ check stopping criteria

    return x^*, u^*, p^*

  compute $\rho = \frac{\Delta L_\lambda}{\Delta \mathcal{J}_\lambda}$ ▷ calculate approximation accuracy

  if $\rho \leq \rho_0$ **then**

    $\bar{x}, \bar{u}, \bar{p} \leftarrow \bar{x}, \bar{u}, \bar{p}$ ▷ reject solution

    $\eta \leftarrow \max(\eta_{min}, \eta/\beta_{sh})$ ▷ shrink trust region

  else

    $\bar{x}, \bar{u}, \bar{p} \leftarrow x^*, u^*, p^*$ ▷ update reference solution

    $$\eta = \begin{cases} \max(\eta_{min}, \eta/\beta_{sh}), & \text{if } \rho_0 \leq \rho \leq \rho_1 \\ \eta, & \text{if } \rho_1 \leq \rho \leq \rho_2 \\ \min(\eta_{max}, \beta_{gr}\eta), & \text{if } \rho_2 \leq \rho \end{cases}$$ ▷ update trust region

3.3 SCvx Python Implementation

The algorithm discussed in section 3.2.2 has been implemented in Python for this project. The implementation was based on a rudimentary SCP-version provided by Wolfgang Hönig[4], which already included a trust region mechanism and was able to handle nonlinear dynamic constraints as well as box-constraints on inputs and initial constraints. A solver, using the same algorithm as foundation, was already implemented by Malyuta et al. in [12][5]. However, this solver only implements collision avoidance for ellipsoid obstacles and a double-integrator model that would not include the full nonlinear dynamics of the multirotor model was implemented. Also, the solver was implemented in Julia and, therefore, difficult to integrate into the existing software architecture of the LIS group[6] written mostly in Python and C++. Furthermore, implementing the solver from scratch improves the understanding of the inner workings and was therefore considered necessary.

In the following, the implementation of the motion planning problem within the SCvx-framework in Python will be described in detail and the convergence behavior of the implementation will be validated. CVXPY[7] is used to formulate the convex subproblem and provide an interface to sophisticated convex solvers. CVXPY is a Python framework developed for formulating and solving convex optimization problems. As a convex solver Gurobi is used since it outperformed comparable convex solvers in practice.

3.3.1 Problem Implementation

In the following, the implementation details to the algorithmic components described in section 3.2.2 are given.

Discrete Multirotor Model:

The dynamic equations are discretized using a forward Euler integration scheme, obtaining the state at $k+1$ as follows:

$$x_{k+1} = x_k + \Delta t T f(x_k, u_k) \tag{3.53}$$

$$x_{k+1} = \hat{f}(x_k, u_k, T). \tag{3.54}$$

Here the step size of the discretization Δt is calculated from the desired number of time steps N with $\Delta t = 1/N$ and the final time T is incorporated into the parameter vector. The final time can either be treated as an optimization variable for time-

[4]Independent junior research group leader at TU-Berlin and co-supervisor of this thesis
[5]https://github.com/UW-ACL/SCPToolbox.jl/tree/csm
[6]Learning and Intelligent Systems group at TU-Berlin
[7]https://github.com/cvxpy/cvxpy

optimal problems or be constrained for fixed final-time problems. In this work, only fixed final-time problems will be considered. The discrete dynamics $\hat{f}(x_k, u_k, T)$ are implemented in the shown form as a step-function in the dynamic models of the framework. An implementation of the nonlinear dynamic equations of a multirotor with a number of motors $n_m = 4$ was already provided. This model was extended to rotor numbers $n_m \in \{2, 3, 4, 6, 8\}$. Additionally, the model only considering the double integrator dynamics was added. For the nonlinear dynamic model, the inertia matrix J of the multirotor was approximated by assuming the motors as point masses. The weights (m_i) of the point masses were calculated from the overall weight of the multirotor provided by the user, leading to the following inertia matrix:

$$\mathbf{J} \approx \begin{pmatrix} \sum_i^{n_m} m_i l_{i,x}^2 & 0 & 0 \\ 0 & \sum_i^{n_m} m_i l_{i,y}^2 & \\ 0 & 0 & \sum_i^{n_m} m_i l_{i,z}^2 \end{pmatrix}. \tag{3.55}$$

Where $l_{i,x}, l_{i,y}, l_{i,z}$ are the distance of the i-th point mass to the x-,y- and z-axis.

Initial Guess:

SCP-methods are locally optimal, only. Therefore, the quality of the provided reference solution $\bar{x}, \bar{u}, \bar{p}$ is crucial to the convergence behavior of the algorithm and the quality of the final solution. In this work, a straight-line interpolation was used as reference solution $\bar{x}$ for the first iteration for the state trajectory and a constant input corresponding to the hover state where used for the control trajectory $\bar{u}$. For the full model, this results in:

$$\hat{p}_k = (1 - \frac{k}{N})p_0 + \frac{k}{N}p_T, \quad k \in 0, ..., N \tag{3.56}$$

$$\hat{v}_k = (1 - \frac{k}{N})v_0 + \frac{k}{N}v_T, \quad k \in 0, ..., N \tag{3.57}$$

$$\hat{\omega}_k = (1 - \frac{k}{N})\omega_0 + \frac{k}{N}\omega_T, \quad k \in 0, ..., N \tag{3.58}$$

$$\hat{u}_k = \begin{pmatrix} f_1 \\ \vdots \\ f_{n_m} \end{pmatrix}, f_i = \frac{mg}{n_m}, \quad i \in \{1, ..., n_m\}. \tag{3.59}$$

Since quaternions require non-equidistant steps in their components to interpolate between two orientations continuously, quaternion spherical line interpolation [64] is used to calculate the reference solution. Resulting in:

$$\hat{q}_k = q_0(q_0^{-1}q_T)^{\frac{k}{N}}, \quad k \in 0, ..., N. \tag{3.60}$$

For the double integrator model, only (3.56) and (3.57) are used and the input reference trajectory is composed of zeros.

For the final-time reference solution $\bar{T}$, the middle of the defined interval is chosen as:

$$\hat{T} = \frac{T_{max} + T_{min}}{2}. \tag{3.61}$$

To increase the numerical stability of the convergence behavior, the noise z, drawn from a normal distribution, was added to the reference solution. Where the mean of the distribution from which the noise was drawn is zero and the standard deviation depends on the feasible range of the states, inputs, and parameters leading to:

$$z \sim \mathcal{N}(0, y_{max} - y_{min}) \tag{3.62}$$

$$\bar{y}_k = \hat{y}_k + \alpha_n z. \tag{3.63}$$

y denotes states and inputs and α_n is a user-defined factor to control the added noise.

Dynamic Constraints:

To make the linearization of the implemented dynamics $\hat{f}(x_k, u_k, T)$ apply to different dynamic models, by avoiding calculating the Jacobians analytically, the Jacobians $\mathbf{A}_k, \mathbf{B}_k, \mathbf{F}_k$ are obtained numerically using JAX[8] (Autograd). JAX can differentiate native Python and NumPy functions, such as the step function implemented in the dynamic model. The discrete dynamic constraints at every time step are then implemented as

$$x_{k+1} = \hat{f}(x_k, u_k, p) + \mathbf{A}_k(x_k - \bar{x}_k) + \mathbf{B}_k(u_k - \bar{u}_k) + \mathbf{F}_k(T - \bar{T}) + \nu_{d_k}. \tag{3.64}$$

Where $T - \bar{T} = 0$ for fixed final time problems.

Collision Avoidance (Non-Convex Path Constraints):

In order to impose constraints regarding collision avoidance, it is necessary to calculate the signed distances d_i between the multirotor and the elements of the set of stationary obstacles $\mathcal{O}$ in the environment. To do this efficiently, the multirotor is approximated at its current position by the sphere $S(t)$, while the obstacles are represented by arbitrary shapes $O_i \in \mathcal{O}, i \in \{1, ..., n_{obs}\}$. The sphere's radius r_S is determined by the arm length l of the multirotor for the full model. For the double integrator model, $r_S \neq 0$ is chosen to avoid numerical problems. $r_S = 1^{-4}$ turned out to be a suitable

[8] https://github.com/google/jax

choice in practice. To prevent the multirotor from intersecting with the obstacles along its trajectory $x(t)$, the general obstacle constraints:

$$S(t) \cap O_i = \emptyset, \quad \forall i \in \{1, ..., n_{obs}\}, \tag{3.65}$$

can be reformulated using the signed distances d_i as:

$$d_i(t) \geq 0 \quad , \forall i \in \{1, ..., n_{obs}\}. \tag{3.66}$$

The signed distance becomes negative for intersecting shapes and positive for non-intersecting shapes. To formulate the constraints in SCvx, an approach is used that was originally proposed by Virgili-Llop et al. in [65]. Since (3.66) are in general non-convex path constraints, they are convexified as described in (3.29). Therefore the constraint gives the following convex approximation (consistent with (3.25) - (3.27)) of the signed distance around the reference trajectory $\bar{x}(t)$:

$$d_i(t) \approx \bar{d}_i(t) + \underbrace{\left[\hat{z}_i(t)^T \mathbf{J}_i(t)\right]}_{\mathbf{C}(t)} (x(t) - \bar{x}(t)) \tag{3.67}$$

$$z_i(t) = \begin{cases} p_{i,S}(t) - p_{i,O}(t), & \text{if} \quad \bar{d}_i(t) > 0 \\ p_{i,O}(t) - p_{i,S}(t), & \text{if} \quad \bar{d}_i(t) < 0 \end{cases} \tag{3.68}$$

$$\hat{z}_i(t) = \frac{z_i(t)}{\|z_i(t)\|_2}. \tag{3.69}$$

Here $p_{i,S}(t) \in S(t)$ and $p_{i,O}(t) \in O_i$ (supporting points) are the closest points on the surface of the multirotor and the i-th obstacle in the reference solution. Moreover $\mathbf{J}_i(t) \in \mathbb{R}^{3\times3}$ denotes the analytical Jacobian of $p_{i,S}(t)$ only depending on the position. Therefore the Jacobian is the identity matrix since $S(t)$ is a sphere. The signed distance and the closest supporting points of the reference solution are obtained using the flexible collision library (FCL)[9] and the FCL Python-bindings[10]. $\hat{z}_i(t)$ can intuitively be understood as the direction in which the nearest point $p_{i,O}$ lies on the obstacle. Therefore $\hat{z}_i(t)$ is not defined if the reference trajectory is in contact with an obstacle ($\bar{d}_i(t) = 0, p_{i,S}(t) = p_{i,O}(t)$), leading to a partly unconstrained trajectory in the next iteration. To ensure constrained behavior in practice, the radius of the spherical approximation $S(t)$ for the time point where $\bar{d}_i(t) = 0$ is decreased by an infinitesimal amount such that the direction information can be obtained again. The used discrete constraints formulation is implemented as follows:

[9] https://github.com/flexible-collision-library/fcl
[10] https://github.com/BerkeleyAutomation/python-fcl

$$d_{i,k+1} = \bar{d}_{i,k} + \hat{z}_{i,k}(t)^T \mathbf{I}_3(x_k - \bar{x}_k) \geq 0, \quad \forall i \in \{1, ..., n_{obs}\}. \tag{3.70}$$

A drawback of this formulation is the possibility of colliding connections between the discrete non-colliding states. Especially when turning sharp corners, this effect plays a role. In practice, this problem is approached by increasing $S(t)$ such that the trajectory becomes collision free again.

Box-Constraints (Convex Path Constraints):

The convex path constraints are formulated as simple box-constraints on the states, inputs and parameters:

$$x_k \geq x_{min}, \quad x_k \leq x_{max} \tag{3.71}$$
$$u_k \geq u_{min}, \quad u_k \leq u_{max} \tag{3.72}$$
$$T \geq T_{min}, \quad T \leq T_{max}. \tag{3.73}$$

where $T_{min} = T_{max}$ for fixed final time problems.

Initial and Final State Constraints:

The initial and final state constraints are already convex and implemented as follows:

$$x_{k=0} = x_0, \quad x_{k=N} = x_N. \tag{3.74}$$

Objective:

The minimized objective should represent the control effort, i.e., the inputs. Therefore the sum of squares of the inputs is chosen. Due to the introduced slack variables used to avoid artificial infeasibility, the objective function has to be augmented to penalize these to avoid constraints violations. The final cost function then results in

$$\begin{aligned} L_{\lambda,\beta,\gamma}(u, T, \nu_d, \nu_s, \nu_{ic}, \nu_{tc}) = \lambda(\|\nu_{ic}\|_1 + \|\nu_{tc}\|_1) \\ + \beta(1-\gamma)[\sum_{k}^{N-1} \sum_{i}^{n_m} u_{k,i}^2 + \lambda(\|\nu_{d,k}\|_1 + \|\nu_{s,k}\|_1)]\Delta t + \gamma T. \end{aligned} \tag{3.75}$$

Where β is the weight of the inputs and γ is used to tune the trade-off between consumed energy and final time when free final time problems are considered. Therefore throughout this work, $\gamma = 0$ is chosen.

Helper Variables:

In addition to the algorithmic components described in section 3.2.2, further decision variables are used by Malyuta et al. These are added to reformulate the trust region constraints and objective function. Where for the trust region the vectors of decision variables $\delta_x \in \mathbb{R}^N, \delta_u \in \mathbb{R}^N, \delta_p \in \mathbb{R}$ are introduced to augment the constraints as:

$$\|x_k - \bar{x}_k\|_\infty \leq \delta_{x,k}, \tag{3.76}$$
$$\|u_k - \bar{u}_k\|_\infty \leq \delta_{u,k}, \tag{3.77}$$
$$\|p_k - \bar{p}_k\|_\infty \leq \delta_{p,k}, \tag{3.78}$$
$$\delta_{x,k} + \delta_{u,k} + \delta_{p,k} \leq \eta. \tag{3.79}$$

For the objective function $P \in \mathbb{R}^{N-1}, P_i \in \mathbb{R}, P_f \in \mathbb{R}$ are introduced as new decision variables. These are used to augment the objective function to

$$\|\nu_{d,k}\|_1 + \|\nu_{s,k}\|_1 \leq P_k, \quad \forall_k \in \{0, ..., N-1\}, \tag{3.80}$$
$$\|\nu_{ic}\|_1 \leq P_i, \tag{3.81}$$
$$\|\nu_{tc}\|_1 \leq P_f, \tag{3.82}$$
$$L_{\lambda,\gamma}(u, T, \nu_d, \nu_s, \nu_{ic}, \nu_{tc}) = \lambda(P_i + P_f) + (1-\gamma)[\sum_{k}^{N-1} \sum_{i}^{n_m} u_{k,i}^2 + P_k]\Delta t + \gamma T \tag{3.83}$$

Note that all introduced augmentations do not change the local or global minima. Nevertheless, the augmented optimization problem has additional dimensions changing its structure and giving the optimizer more flexibility in its convergence. Even though these are not based on a theoretical consideration, they proved to lead to better solutions in specific scenarios by helping the optimizer avoid getting stuck in infeasible local minima.

3.3.2 Structure of the Python Framework

The framework optimizes a given trajectory generation problem following the flowchart shown in figure 3-3. The implementation has the following key components:

- **Dynamic Model:** In the dynamic models, two things are defined: Firstly, the physical parameters, such as the geometry and inertia of the multirotor and secondly the discrete dynamic equations of the system. The models are initialized with user-defined parameters such as the number of rotors of the multirotor.
- **Problem Definition:** It contains the actual problem specifications like the

initial state x_0, the end state x_T, the final time T and the shape and position of the obstacles.

- **Collision Handler:** The collision module is used to calculate the signed distances $\bar{d}_i$ and the closest supporting points $p_{i,m}, p_{i,o}$ on the obstacles O_i and the model approximation S_i at all discrete waypoints. It is initialized with the robot model and provides methods to register objects that should be avoided.
- **SCvx:** Here the actual SCvx-Algorithm is implemented and its parameters, such as the weights on slack and input and the initial trust region, are defined. Also, the described constraints and the objective are implemented here. The SCvx module additionally provides a method to solve the defined motion planning problem.
- **Check:** To check if the obtained solution is feasible with respect to the originally imposed constraints, the violations of the non-convex constraints are evaluated.

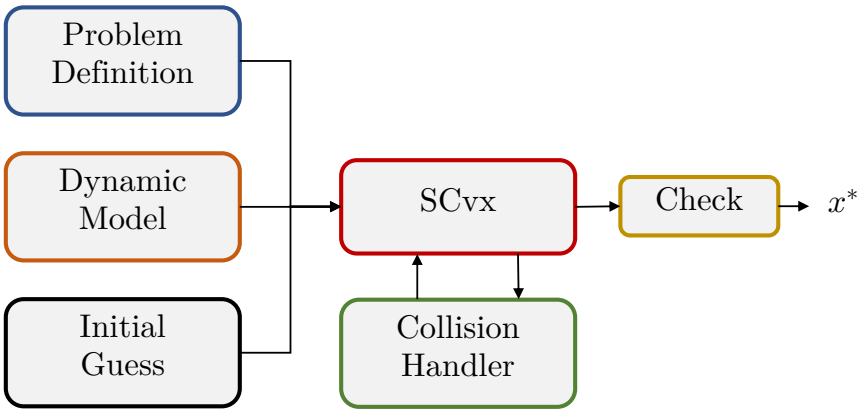


Figure 3-3: SCvx-framework flowchart.

3.3.3 Validation

In the following, our SCvx-framework in Python (SCvx-Python) will briefly be compared to the original implementation in Julia (SCvx-Julia). Therefore the multirotor motion optimization problem implemented by Malyuta et al. was adapted to only one spherical obstacle and new initial and final states while keeping the remaining constraints the same (compare [12, page 48–50]). Also, the same SCvx specific parameters, such as the initial trust region and the weight on the slack cost, were used (compare [12, page 49], table 2, Algorithmic parameters). The problem used to compare the implementations is based on the double integrator model, which already leads to linear dynamics. The collision avoidance is formulated in an analytical form, which can only handle ellipsoid objects ([12, page 49], equation (113)). As a convex solver, ECOS was used since the implementation of Malyuta et al. did not support Gurobi. SCvx-Python and SCvx-Julia were run for 15 iterations and the following four quantities were obtained at each iteration: The approximation accuracy ρ, the nonlinear cost $\mathcal{J}_\lambda$, the trust region radius η and the distance to the optimal solution Δ_x of each of the implementations. The latter is closely related to the trajectory convergence criterion discussed in section 3.2.2 and defined as follows:

$$\Delta_x = \frac{\|x_i - x^*\|_2}{\|x^*\|_2}. \tag{3.84}$$

Here x_i is the trajectory at the i-th iteration and x^* is the converged solution returned by the frameworks. The history of the quantities mentioned above is shown in figure 3-4 and the convergence behavior of the positional part of the trajectory is shown in figure 3-5. As presented in the former, the convergence of SCvx-Python is, similar to SCvx-Julia, with a slight offset. Experience has shown that convergence behavior is prone to numerical imprecision. For example, changing the numerical precision of the calculation of $\mathbf{A}_k, \mathbf{B}_k, \mathbf{F}_k$ from single-precision (32 bit) to double-precision (64 bit) has a significant influence on the convergence behavior. The remaining slight difference in the convergence behavior is considered unimportant since the distance to the optimal solution lies in roughly the same region and the algorithms will converge in the same number of iterations. Also, the non-convex cost follows nearly the same path, resulting in just a slightly worse solution for SCvx-Python. The trust region update is the same for both implementations and the approximation accuracy shows only insignificant differences, with slightly better accuracy for SCvx-Python. Evaluating the total time needed for a solution would not provide a fair comparison due to Julia's significant speed advantages. Since this was an accepted downside of implementing the algorithm in Python, SCvx-Python was not evaluated based on total run time.

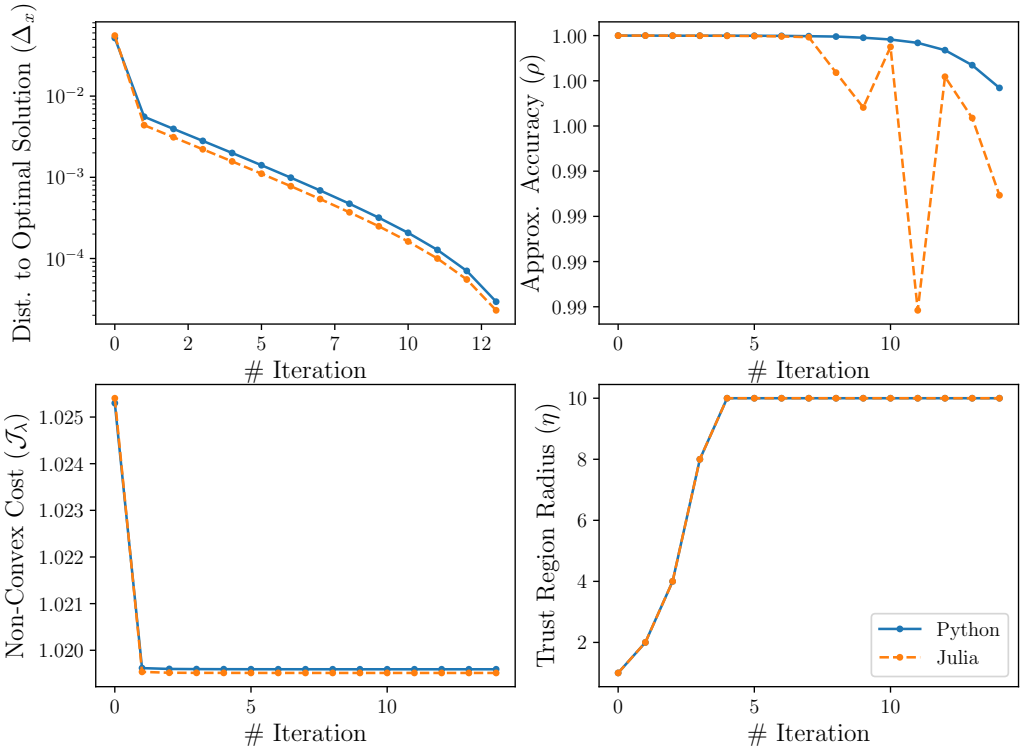


Figure 3-4: History of characteristic quantities of SCvx-Python and SCvx-Julia.

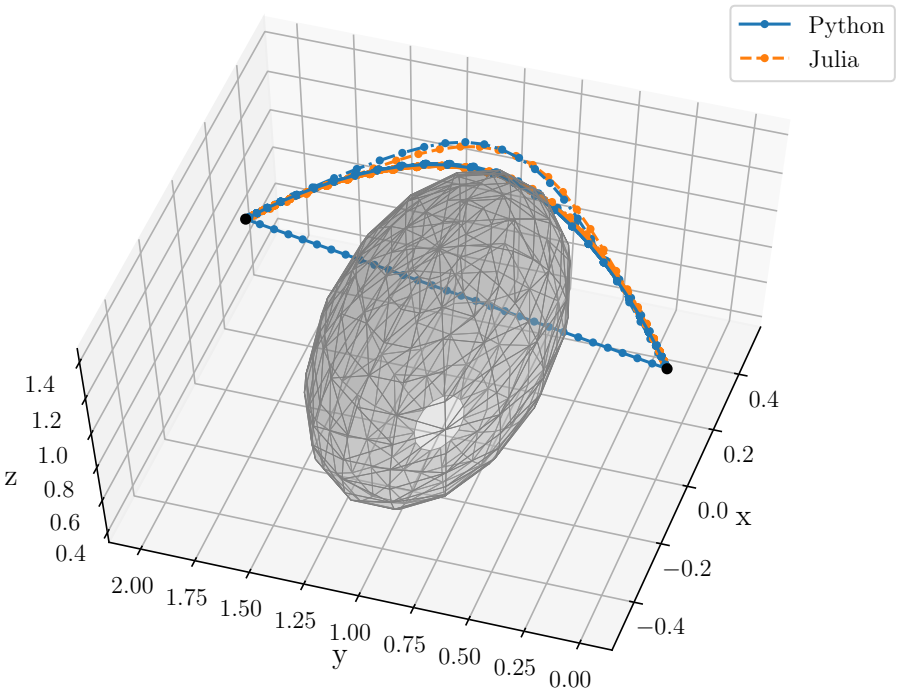


Figure 3-5: Convergence of the trajectories obtained by SCvx-Python and SCvx-Julia.

3.4 K-Order Markov Optimization (KOMO)

In this section, the second motion planning method used in this work will be described in detail. Therefore the mathematical foundation of KOMO (K-Order Markov Optimization) and the core methods of the RAI-framework (Robotic AI) in which KOMO is implemented will be described. KOMO is a sophisticated method for efficiently solving robot motion planning problems originally introduced in 2014 [66]. Even though it belongs to the class of optimization-based methods and is considered an optimal local planner, experience has shown that KOMO is able to find feasible solutions for a wide variety of different kinematic problems. Furthermore, in comparison to other motion planning methods, KOMO represents the trajectory only in configuration space (x) instead of phase space ($x, \dot{x}$), handling differential quantities by finite differences of consecutive configurations. This leads to a structure of the approximate hessian that can be exploited for efficient computation. Another advantage of the chosen representation is that it renders discretization schemes necessary for solving dynamical problems obsolete [67]. In general, the RAI-framework aims to separate the nonlinear optimizer and the abstraction of the motion planning problem into the standard form of an optimization problem:

$$\min_x J(x) \quad s.t. \quad g(x) \leq 0, \quad h(x) = 0. \tag{3.85}$$

With optimiaztion variables $x \in \mathbb{R}^n$, objective function $J : \mathbb{R}^n \to \mathbb{R}$, inequality constraints $g : \mathbb{R}^n \to \mathbb{R}^m$ and equality constraints $h : \mathbb{R}^n \to \mathbb{R}^l$.

3.4.1 Mathematical Formulation

The foundation of KOMO's mathematical problem formulation is, as the name implies, the k-order Markov assumption, describing the short-term dependency of configurations along the trajectory. Let $x \in \mathbb{R}^{N \times n_x}$ be a trajectory consisting of N configurations in an n_x-dimensional configuration space and the objective function $J_t : \mathbb{R}^{n_x} \to \mathbb{R}$, the inequality constraints $g_t : \mathbb{R}^{n_x} \to \mathbb{R}^{n_g}$ and the equality constraints $h_t : \mathbb{R}^{n_x} \to \mathbb{R}^{n_h}$ be smooth. Making use of the Markov assumption the standard problem (3.85) can be formulated as a combination of terms each only depending on $k+1$ configurations choosing $J(x), g(x)$ and $h(x)$ to:

$$J(x) = \sum_t J_t(x_{t-k:t}), \quad g(x) = \bigotimes_t g_t(x_{t-k:t}), \quad h(x) = \bigotimes_t h_t(x_{t-k:t}). \tag{3.86}$$

The outer product $\bigotimes$ denotes the constraints at each time step t being stacked into the constraint functions over the full path. This leads to the following optimization problem:

$$\min_x \sum_t J_t(x_{t-k:t}) \quad s.t. \qquad g_t(x_{t-k:t}) \leq 0, \quad h_t(x_{t-k:t}) = 0, \quad \forall t. \tag{3.87}$$

Here the tuples $x_{t-k:t} = (x_{t-k}, x_{t-k+1}, ..., x_t)$ describe the configurations considered at each time point and the initial configurations $x_{k-1:0}$ are assumed to be known. Each instance of $J_t(x_{t-k:t}), g_t(x_{t-k:t})$ and $h_t(x_{t-k:t})$ is called a feature and encodes e.g. penalties on the distance to a goal configuration or constraints on the velocity of the system calculated by finite differences of the configurations. All resulting features are stacked into a vector of appropriate size. Due to the structure introduced by the short-term dependency of the Markov property, the Jacobian and the pseudo-Hessian of the stacked features result in banded and banded-symmetric matrices which are efficient to compute, store and factorize [66]. This is done by using row-shifted matrix packing, which only stores the $(k + 1)n$ non-zero elements of the Jacobian.

3.4.2 Framework / API

Formulating a motion planning problem for complex dynamic systems like robots requires condensing abstract goals like obstacle avoidance into an efficiently solvable problem formulation. The RAI-framework[11], described in [67], simplifies this by converting user-defined goals into the aforementioned general form of an optimization problem, storing the Jacobian and the Hessian efficiently. With this, the framework decouples the problem formulation from actually solving it, allowing for the use of generic NLP-Solvers. Specific data structures and methods are used to represent the obstacles, the dynamic model, the constraints, and the objectives of the planning problem. These are briefly explained as they form the core of the framework.

Configurations: Configurations are the basic representation of the scenery used by the framework to define a motion planning problem [67]. In these, all objects (frames) from a 3D environment are stored and it is defined how they are kinematically linked. The single frames in a configuration are connected by joints with certain degrees of freedom and the vector of all combined degrees of freedom of one configuration is called joint state. Configurations also store information about acting contact forces between frames in a scenery. The mentioned data is stored in a structure $C(x)$, where a joint configuration x is mapped through a kinematic engine C.

Frames: Frames are the elementary structures within a configuration [67]. Each frame has different properties, such as degrees of freedom describing its orientation and position relative to its parents. The 3D shape of a frame is represented by a mesh that can be used for proximity checking. Frames also include the inertia of a link.

[11] https://github.com/MarcToussaint/rai

Features: Features are the interface between configurations and numerics, representing any numerical quantity that might be calculated for a configuration [67] (e.g., the distance between two frames or the system's total energy). Features often represent errors such as the distance between a desired and an actual position or errors in Newton-Euler equations. The actual optimization objective is then conveniently formulated as bringing all features to zero. A feature vector $\phi_t : (C_{t-k:t}) \to (y, \mathbf{J})$ maps $k+1$ consecutive configurations to the desired numerical quantities $y \in \mathbb{R}^d$ and its Jacobian $\mathbf{J} \in \mathbb{R}^{d \times n}$. Where $d = dim(\phi)$ is the dimension of the calculated numerical quantities. Additionally, a linear transformation can be defined by the parameters ρ and y^*, where ρ acts as a weight and y^* shifts the target value of the features away from zero. The transformed feature vector then has the following form:

$$\hat{\phi}_t(C_{t-k:t})_y = diag(\rho)(\phi_t(C_{t-k:t})_y - y^*), \tag{3.88}$$

where $\phi_t(C_{t-k:t})_y$ and $\hat{\phi}_t(C_{t-k:t})_y$ denote only the part returning y. The transformed feature then still gives the Jacobian of the shifted y.

3.4.3 Solver

As mentioned, KOMO formulates the motion planning problem as a nonlinear constrained optimization problem. This section describes the augmented Lagrangian method used internally to solve the nonlinear problem formulated with KOMO. Since the KOMO-framework handles the formulation of the problem into a NLP in standard form (3.85), this formulation will be used throughout the next section.

Karush-Kuhn-Tucker Conditions

The necessary optimality conditions (Karush-Kuhn-Tucker conditions (KKT-conditions)) for x^* being a local solution will be briefly discussed to motivate the augmented Lagrangian. Therefore the Lagrangian of the problem in standard form is defined as:

$$\mathcal{L}(x, \lambda, \kappa) = J(x) + \lambda^T g(x) + \kappa^T h(x). \tag{3.89}$$

with the vectors of Lagrangian multipliers $\lambda \in \mathbb{R}^{n_g}, \kappa \in \mathbb{R}^{n_h}$ and $\lambda_i \geq 0$. Making use of the Lagrangian function, the KKT-conditions [68] are defined as follows:

If x^* is a local solution to problem (3.85)*, the objective and constraint functions are continuously differentiable and the linear independence constraint qualification [68] holds, then there exist vectors of Lagrangian multipliers λ^* and κ^* such that the conditions are satisfied:*

$$\textbf{1.} \quad \nabla_{x^*}\mathcal{L}(x^*, \lambda^*, \kappa^*) = \nabla J(x^*) + \sum_{i=1} \lambda_i^* \nabla g_i(x^*) + \sum_{j=1} \kappa_j^* \nabla h_j(x^*) = 0 \tag{3.90}$$

$$\textbf{2.} \quad g_i(x^*) \leq 0, \;\; h_j(x^*) = 0, \;\; \forall i, j \tag{3.91}$$

$$\textbf{3.} \quad \lambda_i^* \geq 0, \;\; \forall i \tag{3.92}$$

$$\textbf{4.} \quad \lambda_i^* g_i(x^*) = 0, \;\; \forall i. \tag{3.93}$$

$$\tag{3.94}$$

Intuitively, the first condition can be interpreted as a stationarity condition. This implies that at the optimum x^* has to be an equilibrium between the gradients of the active inequality and equality constraints and the gradient of the objective function. The second condition indicates primal feasibility meaning the problem is feasible regarding the constraints. Enforcing all λ_i^* to be positive ensures that the directional dependency of the inequality constraints is taken into account. The fourth condition elegantly implies that either the optimal solution lies on the boundary of the feasible region ($g_i(x^*) = 0$) or inside it. In the first case, the constraint's gradient prevents the objective's gradient from pushing the solution x^* outside of the boundary ($\lambda_i^* \nabla g_i(x^*) \neq 0$). In the second case, the optimal solution lies within the feasible region and therefore the search is only guided by the gradients of the objective and equality constraints, leading to $\lambda_i^* = 0$ and ($\lambda_i^* \nabla g_i(x^*) = 0$). These conditions use only first-order information and are necessary for x^* being an optial solution but not sufficient. To determine if a solution fulfilling the KKT-conditions is a local optimum second order optimality conditions also have to apply [68]. Still, these are not further discussed in this work since they seldomly play a role in numerical methods.

Augmented Lagrangian

The applied constraints are often incorporated into the objective function to use gradient-based methods to solve a constrained optimization problem. As the name already implies, the Augmented Lagrangian uses a formulation closely related to the Lagrangian of the optimization problem to handle equality and inequality constraints. The used formulation is based on the squared penalty method, which will therefore be briefly discussed for problems with only equality constraints:

$$\min_x J(x) \quad s.t. \;\; h_j(x) = 0, \forall j. \tag{3.95}$$

Squared Penalty Methods:

The squared penalty method includes constraints into the objective by adding the squared constraints violations to it, leading to the following unconstrained optimization problem:

$$\min_x J(x) + \frac{\nu}{2} \sum_j h_j^2(x). \tag{3.96}$$

Where $\nu \in \mathbb{R}, \nu \geq 0$ is a weight on the constraints violations. The solution of the modified problem for $\mu < \infty$ approximates the original problem and only respects the constraints to a certain degree. By iteratively increasing μ and re-solving the problem using the former solution as an initial guess, the obtained solution converges to the actual solution of the original problem. For large μ, the Hessian of the objective becomes increasingly ill-conditioned and leads to performance issues when using algorithms for unconstrained problems such as quasi-Newton methods [68].

Augmented Lagrangian for Equality Constrained Problems:

Since the last element of a finite series of solutions obtained by the iterative approach of the squared penalty method will still violate the constraints by a small amount, it would be beneficial to approximate this systematic violation and tackle it in the objective directly. The Augmented Lagrangian achieves this by adding an approximation of the Lagrangian multipliers κ_j to formulation (3.96) based on the solution of the last iteration [68]. The augmented objective to minimize then has the following form:

$$\mathcal{L}_A(x, \kappa, \nu) = J(x) + \frac{\nu}{2} \sum_j h_j^2(x) + \sum_j \kappa_j h_j(x). \tag{3.97}$$

Where the solution x_k^* minimizing (3.97) in the k-th iteration fulfills the optimality condition for unconstrained optimization $\nabla \mathcal{L}_A(x, \kappa, \nu) = 0$ [68] as follows:

$$x_k^* = \operatorname*{argmin}_x J(x) + \frac{\nu_k}{2} \sum_j h_j^2(x) + \sum_j \kappa_{j,k} h_j(x) \tag{3.98}$$

$$0 = \nabla J(x_k^*) + \nu_k \sum_j h_j(x_k^*) \nabla h_j(x_k^*) + \sum_j \kappa_{j,k} \nabla h_j(x_k^*). \tag{3.99}$$

The update of the approximation κ_j in every iteration can then be formulated from condition (3.99) as:

$$\sum_j \kappa_{j,k+1} \nabla h_j(x_k^*) = \nu_k \sum_j h_j(x_k^*) \nabla h_j(x_k^*) + \sum_j \kappa_{j,k} \nabla h_j(x_k^*) \tag{3.100}$$

$$\kappa_{j,k+1} = \kappa_{j,k} + \nu_k h_j(x_k^*). \tag{3.101}$$

The Augmented Lagrangian method tackles one of the significant downsides of the squared penalty method by avoiding the ill-conditioning of the Hessian and also largely preserves the smoothness of the objective function when incorporating the constraints [68]. Also, it provides a much better constraint approximation [69].

Augmented Lagrangian for Inequality Constrained Problems:

To incorporate inequality constraints in the Augmented Lagrangian formulation, the constraints are added to the objective such that only constraint violations are penalized. This is done by introducing the following indicator function [69]:

$$I_{g_i} = \begin{cases} 1, & \text{if } g_i(x) \geq 0 \vee \lambda_i > 0 \\ 0, & \text{if } g_i(x) < 0 \wedge \lambda_i = 0. \end{cases} \tag{3.102}$$

The indicator function makes sure that constraint penalty $g_i(x)^2$ pushes the solution to the boundary $g_i(x) = 0$, if the constraint is active. If the constraint is not active, the indicator function sets the introduced penalty to 0. Using the indicator function, the augmented objective then becomes:

$$\mathcal{L}_A(x, \lambda, \mu, \kappa, \nu) = J(x) + \frac{\mu}{2} \sum_i I_{g_i} g_i(x)^2 + \sum_i \lambda_i g_i(x) + \frac{\nu}{2} \sum_j h_j^2(x) + \sum_j \kappa_j h_j(x). \tag{3.103}$$

With the Augmented Lagrangian dual updates:

$$\lambda_{i,k+1} = \max(0, \lambda_{i,k} + \mu_k g_i(x_k^*)) \tag{3.104}$$

$$\kappa_{j,k+1} = \kappa_{j,k} + \nu_k h_j(x_k^*). \tag{3.105}$$

Taking the $max(\cdot, \cdot)$ makes sure that the Lagrangian multiplier still fulfills $\lambda_i \geq 0$ in the next step. Converting the described method into an algorithmic framework leads to algorithm 2.

Algorithm 2 Augmented Lagrangian Method

procedure AUGMENTEDLAGRANGIAN(x_0, ρ_μ, ρ_ν, μ_0, ν_0, Θ, ϵ)
  Given: Initial guess x_0; penalty weight update parameters ρ_μ, $\rho_\nu > 1$; initial penalty weights μ_0, ν_0; convergence tolerances Θ, ϵ
  $\lambda_0 \leftarrow 0$, $\kappa_0 \leftarrow 0$ ▷ initiate Lagrangian multiplier
  $k \leftarrow 0$
  while True **do**
    $x_{k+1} \leftarrow$ minimize $\mathcal{L}_A(x_k, \lambda_k, \mu_k, \kappa_k, \nu_k)$ ▷ solve sub-problem
    if $|\Delta_x| < \Theta$ and $g_i(x_k) < \epsilon$ and $|h_j(x_k)| < \epsilon$ **then**
      return x_{k+1} ▷ return solution when converged
    $\lambda_{i,k+1} \leftarrow \max(0, \lambda_{i,k} + \mu_k g_i(x_{k+1}))$, $\forall i$ ▷ update Lagrangian multiplier λ
    $\kappa_{j,k+1} = \kappa_{j,k} + \nu_k h_j(x_{k+1})$, $\forall j$ ▷ update Lagrangian multiplier κ
    $\mu_{k+1} \leftarrow \rho_\mu \mu_k$, $\nu_{k+1} \leftarrow \rho_\nu \nu_k$ ▷ update penalty weights
    $k = k + 1$

3.5 KOMO Motion Planning Problem in Python

Throughout the following section the implementation of problem (3.6) - (3.11) using the KOMO-Framework is described. A convenient way to incorporate the KOMO-Framework in Python projects is using the provided bindings[12]. These allow the use of a large number of native methods of the C++ implementation. As well as for SCvx, in KOMO only fixed final time problems are considered.

Kinematic Model:

The implementation of the full multirotor model in the KOMO-framework is done in two steps. In the first step, a geometric representation of the multirotor is defined. This includes specifying the position of each frame, such as motors and arms, relative to either the world-coordinate system or its parent frame. In addition, it is defined for each frame how it can move relative to its parent, i.e., what degrees of freedom the frame has. In the implemented model, all frames were rigidly connected and therefore, the combined kinematic model has 6 degrees of freedom. For collisions, a sphere with the same radius chosen as for the SCvx-framework was used as a model approximation. The kinematic model of the multirotor for four rotors without the collision sphere is shown in figure 3-6 on the left-hand side. This implementation was based on a model provided by Marc Toussaint[13] which was extended to rotor numbers

[12] https://github.com/MarcToussaint/rai-python

[13] Head Learning and Intelligent Systems Laboratory at TU-Berlin and examiner of this thesis

$n_m \in \{2, 3, 4, 6, 8\}$.

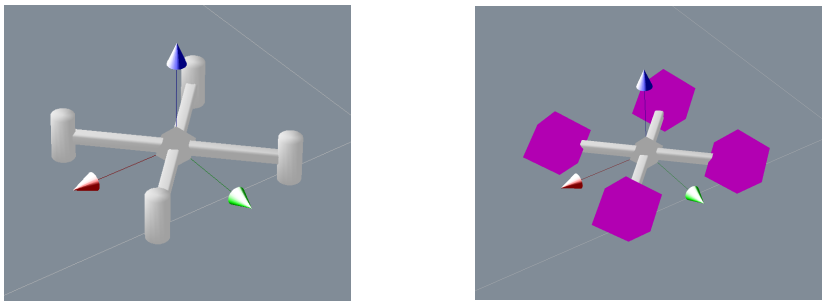


Figure 3-6: Multirotor model with 4 rotors with (right) and without (left) thrust implemented in KOMO-framework.

In the second step, the interaction between the multirotor and its environment caused by the thrust of the rotors is modeled. The KOMO-framework provides methods to add force exchanges between frames which are used to model the force acting between the motor frame and the world frame. Note that the torque introduced by the rotor drag is also included in this step. Adding a force exchange adds a new decision variable to the optimization problem. Therefore for each motor force, a decision variable is added representing the inputs that should be optimized (figure 3-6 right-hand side).

Only a spherical approximation with the radius $r_S = 1^{-4}$ used for collision checking is implemented for the double integrator model. There is no need for force exchanges in this case since the control input for the double integrator model is its acceleration.

Initial Guess:

The initial guess for KOMO was generated in the same fashion as the initial guess for SCvx described in section 3.3.

Constraints and Objective:

The constraints of problem (3.6)-(3.11) are defined in KOMO using features. As already mentioned, features are the interface between configurations and numerics and can define any differentiable mapping ϕ_k. They can represent quantities such as distances between configurations and can approximate the k-th derivative when using order k. For example, a feature representing the position of a frame can represent its velocity by setting the order of the feature to $k = 1$. Initial and final constraints, as well as the box-constraints representing minimal and maximal states, are implemented by using features on positions/orientations and their derivatives. The corresponding target values (e.g., p_T, v_{max}) are incorporated to specify the goal of the equality constraint or the limit of the inequality constraint. For the input constraints, the features are implemented using the newly introduced optimization variables. In contrast to the SCvx-algorithm, the initial configuration is not part of the decision variables. The collision constraints in KOMO are handled internally and use a similar formulation as already described in section 3.3. The collision feature

maps a configuration to a real number indicating a collision if positive. Again the signed distances are calculated, with the Jacobian of the distances depending on the shape of the involved frames. The linear dynamics of the double integrator system are implicitly enforced by the box-constraints on the accelerations. For the full model, KOMO provides a convenient way to formulate the dynamic constraints by making use of the Newton-Euler equations (2.1), which were described in section 2.1.2 expressed in the inertial frame. Therefore the states and inputs are constrained to be consistent with the following:

$$f_{T,I} = \dot{v}_I m - g, \tag{3.106}$$

$$\tau_I = \frac{d}{dt}(\mathbf{J}_I \omega_I). \tag{3.107}$$

Since the equation is now expressed in the inertial frame, the fictitious forces $\omega \times \mathbf{J}\omega$ vanish and the inertia matrix $\mathbf{J}_I$ becomes time-dependent. The accelerations $\dot{v}_I, \dot{\omega}_I$ are approximated using the finite differences of the configurations at k, $k-1$ and $k-2$. For example, for the linear acceleration, this leads to the following approximation and implicated integration scheme

$$\dot{v}_{I,k} \approx \frac{v_{I,k} - v_{I,k-1}}{\Delta t} \quad \rightarrow \quad v_{I,k} = v_{I,k-1} + \Delta t \dot{v}_{I,k}. \tag{3.108}$$

In contrast to SCvx, this corresponds to an implicit Euler integration scheme, since $\dot{v}_{I,k}$ is used to calculate $v_{I,k}$. The features used to implement the motion planning problem for the full model and for the double integrator implementation are listed in appendix A.

Chapter 4

Results

Three specific scenarios were chosen to evaluate the performance of the KOMO-framework compared to the SCvx implementation. The selected scenarios include geometrically complex problems and problems requiring highly dynamic solutions. Trajectories for each scenario were obtained with different motor numbers and each experiment was repeated 15 times to account for the variance in the computation time and evaluate robustness regarding different initial guesses. All results were obtained on a workstation with 32 GB RAM and an AMD Ryzen 9 3900x 12-core processor.

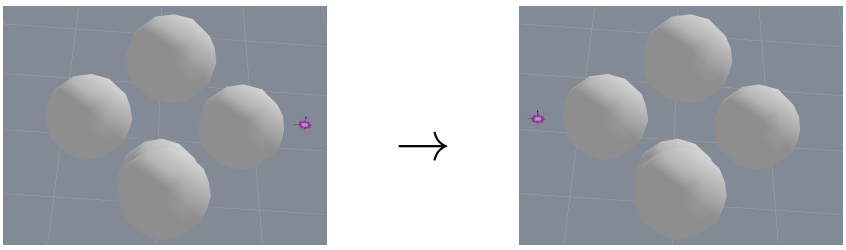
4.1 Scenario Descriptions

In all scenarios, the model parameters of the full model (arm length of the multirotor l, mass m and torque constant κ) were specified as:

$$l = 0.046m \quad , \quad m = 0.034kg \quad , \quad \kappa = 0.006\frac{Nm}{N}. \tag{4.1}$$

Multirotors with motor numbers $n_m = \{4, 6, 8\}$ were considered and three different scenarios were chosen for evaluation.

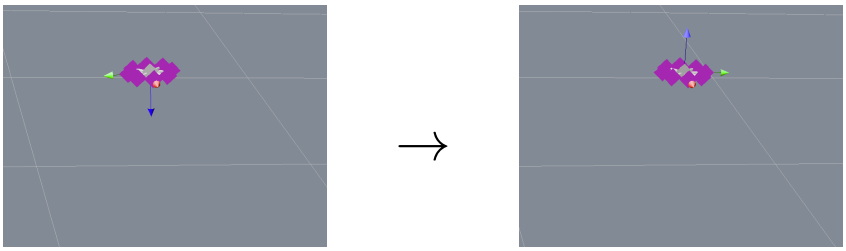
1. **Scenario 1** considers flying through an environment cluttered with spherical obstacles using the double integrator model.
2. **Scenario 2** considers flying through an environment cluttered with spherical obstacles using the full nonlinear dynamics (shown in figure 4-1).



(a) Start (b) Goal

Figure 4-1: Start and goal scenario 2.

3. **Scenario 3** considers recovering the multirotor from an upside-down position (shown in figure 4-2).



(a) Start (b) Goal

Figure 4-2: Start and goal scenario 3.

In the first two scenarios, the initial and the final states are set such that the optimal trajectory circumvents six spherical obstacles. The double integrator model was used in scenario one, while the full dynamics were considered in scenario two. The time horizon was chosen such that the optimal input sequence is not in saturation, i.e., the input does not reach the constraints. This shifts the focus to the obstacle avoidance problem. The second scenario does not include obstacles but is dynamically challenging. Here the multirotor starts nearly upside down and has to recover from that position. The time horizon and the thrust-to-weight ratio were chosen such that the optimal inputs are in saturation. Two different thrust-to-weight ratios were chosen to solve two differently hard-constrained problems. The scenario parameters are stated in table 4.1.

Checking for Feasibility:

To check if a computed solution is feasible, it has to be evaluated if the implemented constraints are respected. The box-constraints on states and inputs can easily be verified by simply applying them. Using FCL ensures that the trajectory is collision-free at every time step. The optimal input series is propagated through the nonlinear dynamics using the corresponding integration scheme to determine if the solution is dynamically feasible. At each $k \in \{0, ..., N-1\}$ time step the propagated state x_{k+1}

and the state of the computed solution x^*_{k+1} are compared. A solution is considered feasible if the difference between these two falls under a user-defined threshold.

Complexity of KOMO:

Since KOMO is using a configuration space representation of the trajectory only implicitly considering higher order derivatives without introducing them as optimization variables, the dimensionality of the optimization problem is drastically reduced with respect to SCvx, as can be seen in table 4.1. Note that KOMO is not introducing more constraints compared to SCvx. The number of optimization variables influences the run time to a high degree. Still, it is not the only decisive factor since the dimensionality only influences the computation time of the inner loop of the augmented Lagrangian method in KOMO and the inner loops of Gurobi. On the other hand, the outer loops are influenced by the problem structure, its conditioning and its convexity. Still, a smaller number of optimization variables can significantly influence the speed of the inner loops leading to a potentially lower computation time for KOMO.

	Scenario 1	**Scenario 2**	**Scenario 3**
Obstacles	$r_O = 0.4$ $p_1 = (0, -0.7, 1)$, $p_2 = (1, 0, 1)$ $p_3 = (-1, 0, 1)$, $p_4 = (0, 0, 1.7)$ $p_5 = (0, 0, 0.3)$, $p_6 = (0, 0.7, 1)$		-
Initial State (x_0)	$p_0 = (0.1, -1.3, 1)$ $q_0 = (1, 0, 0, 0)$ $v_0, \omega_0 = (0, 0, 0)$	$p_0 = (0.1, -1.3, 1)$ $v_0 = (0, 0, 0)$	$p_0 = (0, 0, 1)$ $q_0 = (0.04, 0, -0.9, 0)$ (175° rotation y-axis) $v_0, \omega_0 = (0, 0, 0)$
Final State (x_T)	$p_T = (-0.1, 1.3, 1)$ $q_T = (1, 0, 0, 0)$ $v_T, \omega_T = (0, 0, 0)$	$p_T = (-0.1, 1.3, 1)$ $v_T = (0, 0, 0)$	$p_T = (0, 0.15, 1)$ $q_T = (1, 0, 0, 0)$ $v_T, \omega_T = (0, 0, 0)$
Final Time (T) [s]	2.7	2.7	1.8
#Time Steps (N)	30	30	100
Noise Factor (α_n)	0.05	0.05	0.01
Thrust to Weight Ratio (r_{t2w})	1.4	1.4	{1.4 , 1.5}
# Motors (n_m)	-	{4 , 6, 8}	{4 , 6, 8}
# of Decision Variables SCvx	730	$n_m = 4 : 1187$ $n_m = 6 : 1247$ $n_m = 8 : 1307$	$n_m = 4 : 3417$ $n_m = 6 : 3617$ $n_m = 8 : 3817$
# of Decision Variables KOMO	203	$n_m = 4 : 330$ $n_m = 6 : 390$ $n_m = 8 : 450$	$n_m = 4 : 1100$ $n_m = 6 : 1300$ $n_m = 8 : 1500$

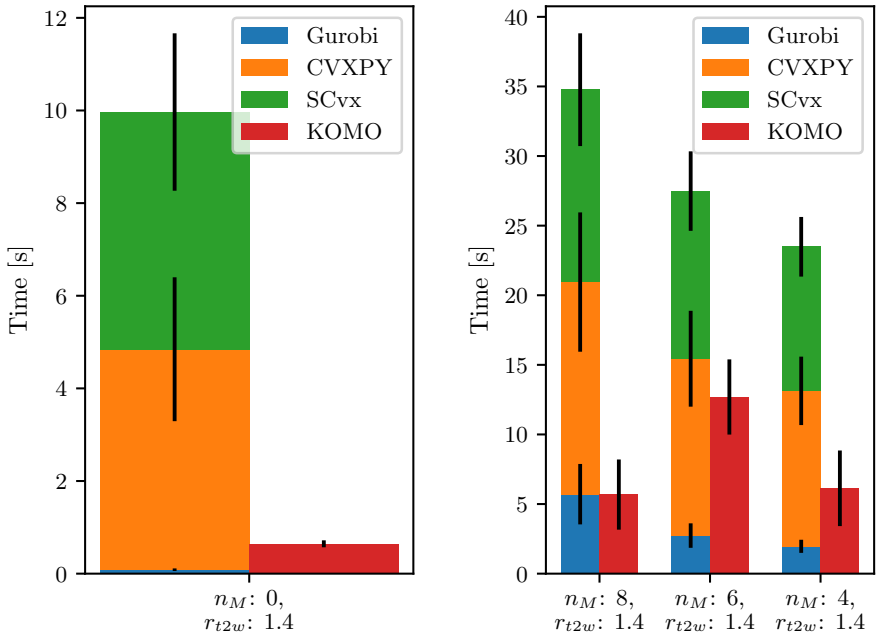
Table 4.1: Scenario parameters.

4.2 Experiments

In the following section, the results of the conducted experiments are presented, where all plots only contain trials considered feasible.

4.2.1 Computation Time

Figure 4-3 and 4-4 summarize the computation time for all three scenarios. Since the SCvx-implementation is done in Python, a direct run time comparison would not be fair. Nevertheless, the convex solver (Gurobi) used in every iteration to obtain the solution to the convex sub-problem is written in C and run time optimized. Therefore the time accumulated over all iterations used by Gurobi can be seen as a lower bound on time taken by an optimized SCvx version. The time it took SCvx to find a solution is split into three parts: The accumulated time spent in the algorithm itself, the accumulated time spent in the CVXPY interface and the accumulated time used by Gurobi to solve the convex sub-problem. The colored bars represent the average time over all trials, while the black lines indicate the standard deviation of the computation time. In figure 4-3a the computation time for scenario 1 is shown. The time it took KOMO to find a solution is substantially smaller than the time needed by SCvx and the accumulated time of Gurobi is again substantially smaller than the computation time of KOMO. For scenario 2 (figure 4-3b), the difference between the computation time of KOMO and the lower bound provided by Gurobi becomes significantly smaller. Using a multirotor model with $n_m = 8$, the computation time becomes even close to equal. Figure 4-4, on the left-hand side, shows the time needed to solve scenario 3 with a tight bound on the inputs ($r_{t2w} = 1.4$). Here KOMO takes comparatively longer to find a feasible solution, even nearly closing the gap to the Python implementation of SCvx for $n_m = 8$. The computation time for scenario 3 for $r_{t2w} = 1.5$ is shown in figure 4-4 on the right-hand side. In the relaxed version of scenario 3 for $n_m = 8$, the run time of KOMO becomes significantly lower while staying the same for other motor numbers. The deviation of the computation time of SCvx for scenarios 1 and 2 is higher than the time deviation of KOMO. This is reversed in scenario 3, leading to a higher deviation of computation time for KOMO. In summary, KOMO converges clearly slower than an optimized SCvx version for scenario 3 and scenario 2 with $n_m = \{4, 6\}$. Whereas for scenario 1 and scenario 2 with $n_m = 8$, no clear statement can be made.



(a) Scenario 1 (b) Scenario 2

Figure 4-3: Comparison of the computation time of KOMO and SCvx.

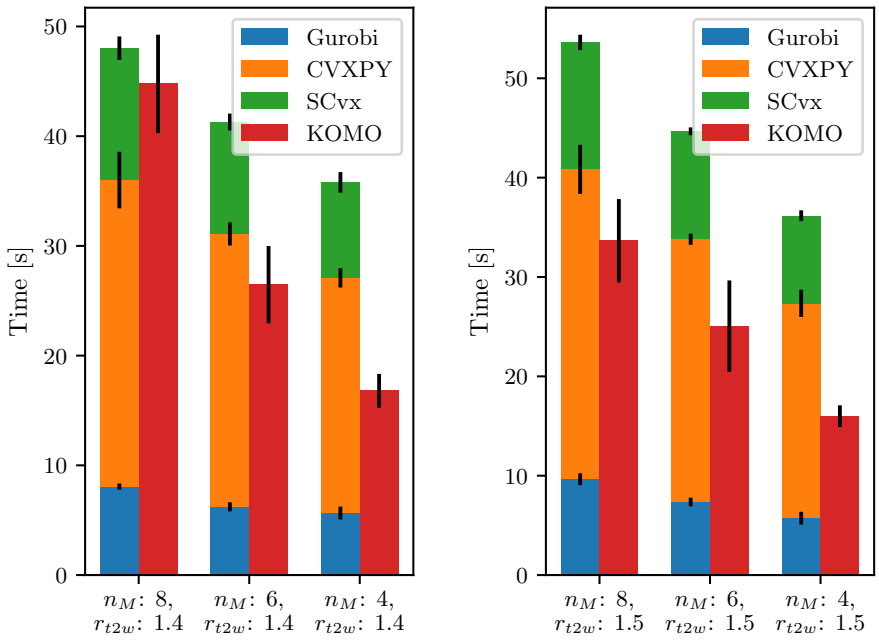


Figure 4-4: Computation time of scenario 3.

4.2.2 Converged Trajectories

Figures 4-5, 4-6 and 4-7 show the obtained trajectories with the minimal costs over all trials, where the arrows indicate the z-axis of the multirotor. Note that for scenarios 1 and 2, there are four equally optimal solutions. Passing on the right side over and under the spheres lying between the initial and the goal position and passing on the left side over and under the spheres results in the same theoretical cost on the used input for the double integrator model and the full model. Figures 4-5 and 4-6 show that KOMO and SCvx both find solutions following trajectories in these group. While SCvx converges for scenario 2 for all rotor numbers to roughly the same trajectory, KOMO's solution differs in the range of the equally optimal solutions. The trajectory obtained from KOMO is shown in detail in appendix B. The trajectories obtained for scenario 3 (figure 4-7) differ between both algorithms. Where the trajectory obtained with KOMO for $n_m = 6, r_{t2w} = 1.4$ is in detail shown in appendix B. SCvx converges throughout all different motor numbers and thrust to weight ratios to similar trajectories, while KOMO finds clearly different solutions for $n_m = 4$. Note that no statement can be made whether one of the algorithms has found the globally optimal solution or not. In summary, the by KOMO computed trajectories change more with changing motor numbers when compared to SCvx.

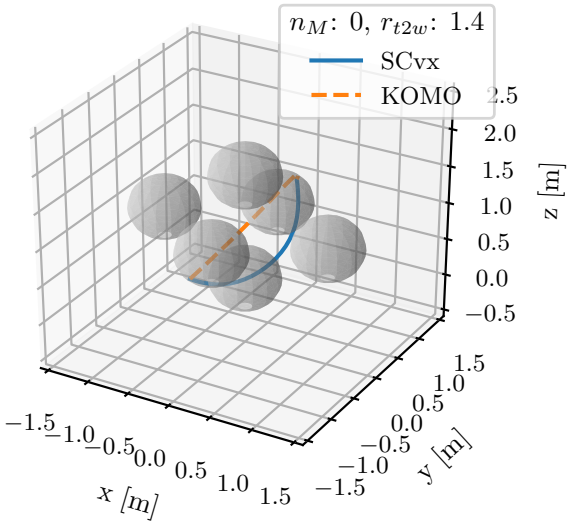


Figure 4-5: Trajectories with minimal cost over all 15 trials for scenario 1.

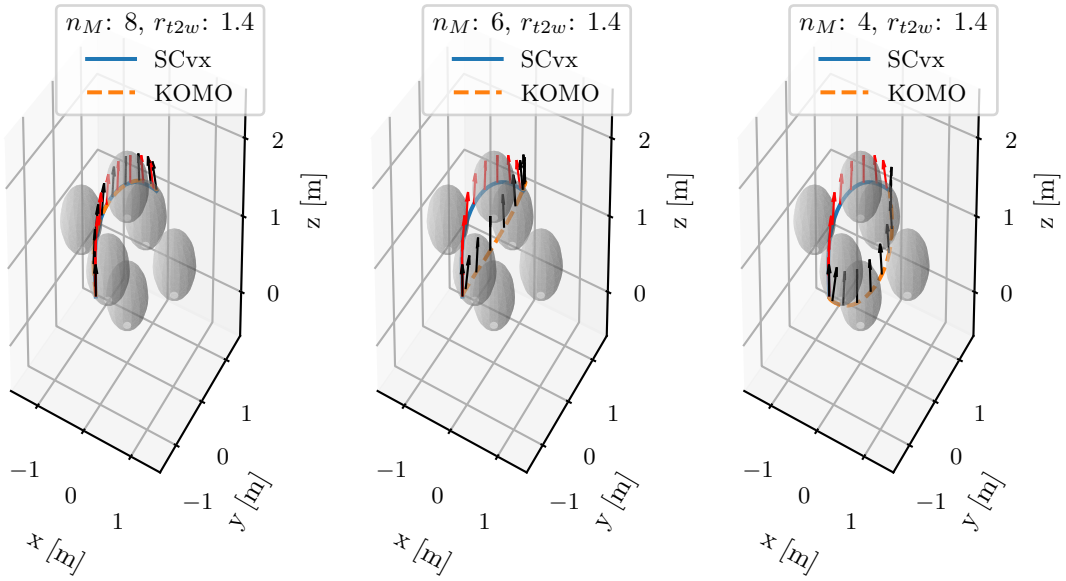


Figure 4-6: Trajectories with minimal cost over all 15 trials for scenario 2.

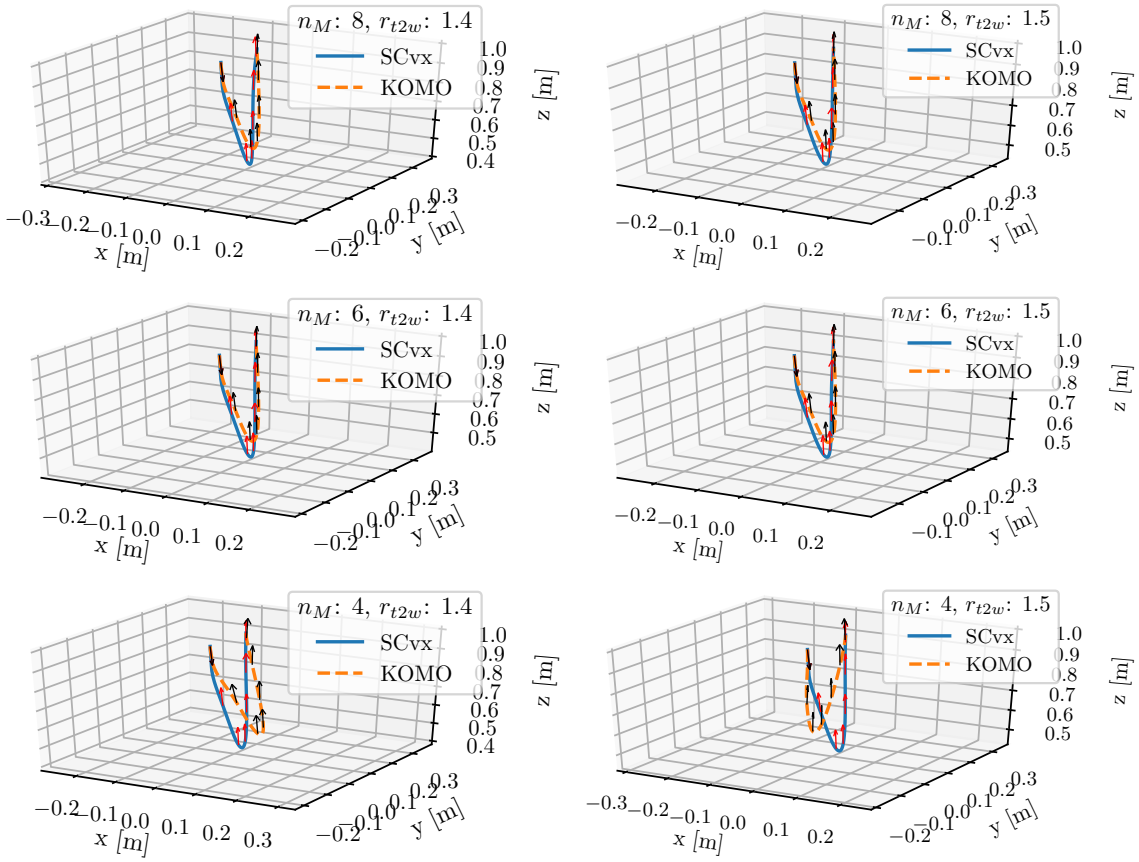
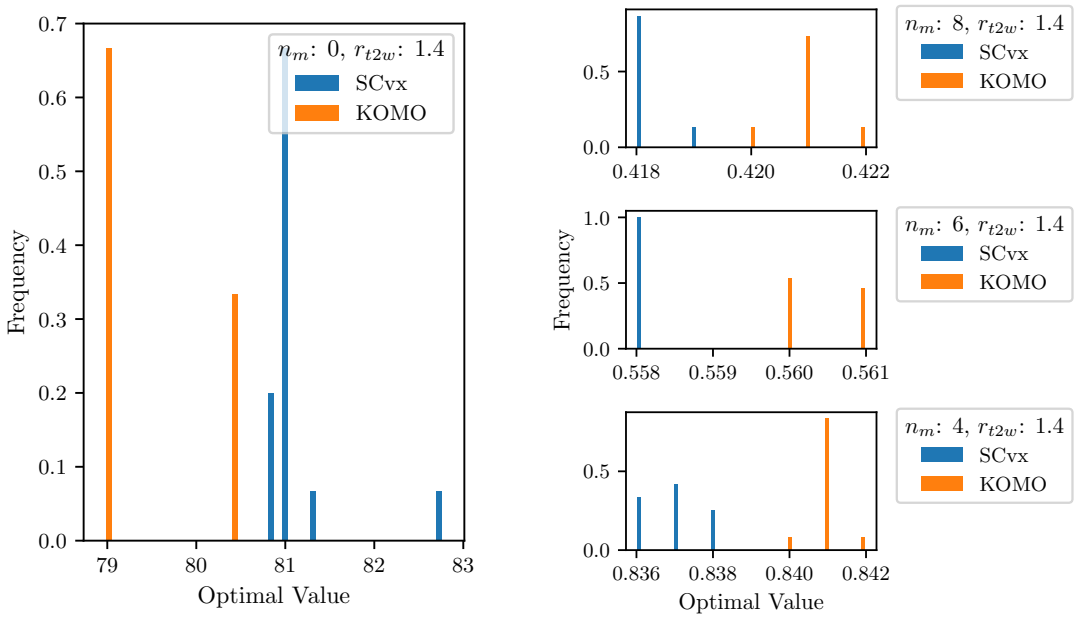


Figure 4-7: Trajectories with minimal cost over all 15 trials for scenario 3.

4.2.3 Distribution of Computed Solutions

In figure 4-8 and 4-9 the distributions of the optimal values for all 15 trials are shown. For scenario 1 (figure 4-8a), KOMO converges for all feasible trials to solutions with lower cost than SCvx with a relative cost difference between both algorithms of 2.4 % and a comparable width of the optimal value distribution. In figure 4-8b the optimal values of scenario 2 are shown. Here SCvx converges to solutions with lower cost compared to KOMO with a relative cost difference of less than 1%. The distribution width of the optimal values is in a similar range except for $n_m = 6$, where SCvx converges to the same solution for all feasible trials. Figure 4-9 shows the optimal values for scenario 3. KOMO is able to find solutions with up to ca. 2.3 % less cost compared to SCvx. The maximal cost improvement is achieved for $n_m = 4$ equally for both thrust-to-weight ratios. While SCvx converges in every trial to the same solution, the optimal values for KOMO are, except for $n_m = 4$, more distributed. In general, the statement can be made that KOMO finds slightly better or equally good solutions compared to SCvx.



(a) Scenario 1

(b) Scenario 2

Figure 4-8: Optimal value distribution.

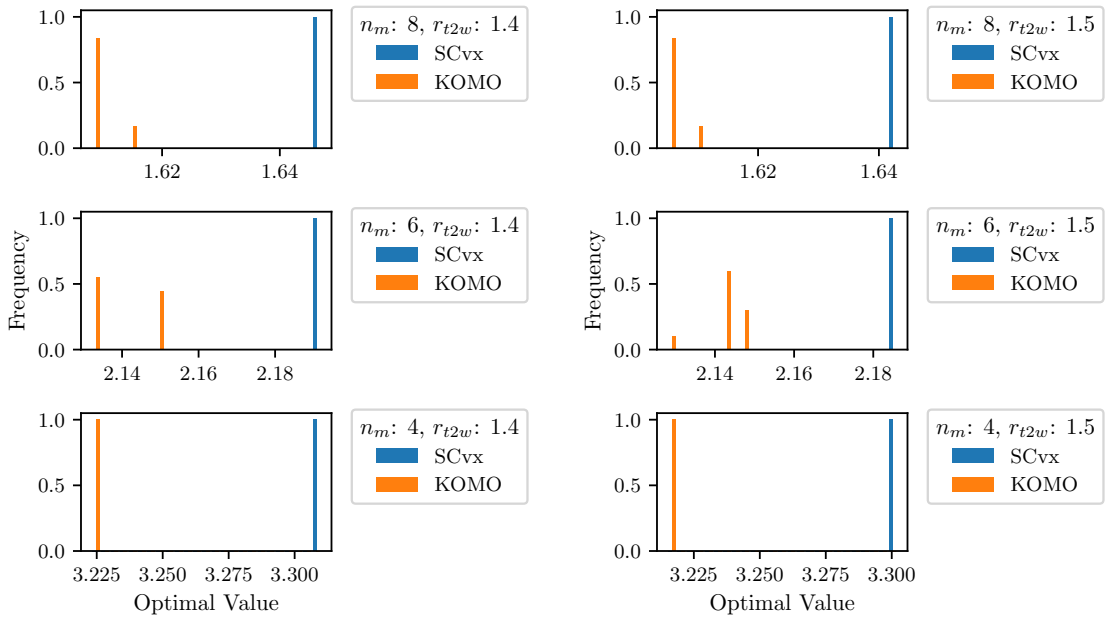


Figure 4-9: Optimal value distribution for scenario 3.

4.2.4 Success Rate

The success rates of the conducted experiments are summarized in table 4.2. A trial is considered feasible if the found solution respects the imposed constraints. For scenario 1, both algorithms had a success rate of 100%. In scenario 2, the success rate of KOMO decreases for $n_m = \{4, 6\}$ from 100% to 87%, while the success rate of SCvx decreases for $n_m = 4$ from 100% only to 93%. The success rate of KOMO for scenario 3 drops considerably for all motor numbers. Especially for tight bounds on the inputs ($r_{t2w} = 1.4$) KOMO struggles to reliably find a solution. This effect becomes less prominent for wider bounds on the inputs ($r_{t2w} = 1.5$). To evaluate how prone the algorithms are to noise on the initial guess, all 3 scenarios (with $r_{t2w} = 1.4$ and $n_m = 4$ for scenario 2 and 3) are reconsidered with noise factors (defined in (3.63)) $\alpha_n = \{0.01, 0.05, 0.1\}$. The resulting success rates are shown in table 4.3. While both algorithms find a feasible solution for scenario 1 in every trial for smaller noise factors, KOMO's success rate drops for $\alpha_n = 0.1$ from 100% to 93%. For scenario 2, the success rate of KOMO exceeds the success rate of SCvx for highly noisy initial guesses. Here SCvx can find a feasible solution in only 60% of the trials. For noise factors $\alpha_n = \{0.05, 0.01\}$, the success rate of SCvx is higher than KOMO's by 6-7 %. In scenario 3 KOMO again has a considerably lower success rate than SCvx. In general, highly noisy initial guesses result in lower success rates for SCvx and KOMO. The only exception is scenario 3 for KOMO, where a higher noise factor benefits the rate of feasible solutions. KOMO finds a feasible solution for scenario 1, for a similar number of trials. For scenario 2, KOMO has in comparison to SCvx a slightly lower success rate, while in scenario 3 KOMO's success rate drops significantly.

Success Rate [%]	**Scenario 1**		**Scenario 2**		**Scenario 3**	
Parameter	KOMO	SCvx	KOMO	SCvx	KOMO	SCvx
$n_m = 0$,$r_{t2w} = 1.4$	**100**	**100**	-	-	-	-
$n_m = 4$,$r_{t2w} = 1.4$	-	-	87	**93**	27	**100**
$n_m = 6$,$r_{t2w} = 1.4$	-	-	87	**100**	60	**100**
$n_m = 8$,$r_{t2w} = 1.4$	-	-	**100**	**100**	40	**100**
$n_m = 4$,$r_{t2w} = 1.5$	-	-	-	-	47	**100**
$n_m = 6$,$r_{t2w} = 1.5$	-	-	-	-	67	**100**
$n_m = 8$,$r_{t2w} = 1.5$	-	-	-	-	80	**100**

Table 4.2: Success rates of KOMO and SCvx for scenario 1, 2 and 3.

Success Rate [%]	**Scenario 1**		**Scenario 2**		**Scenario 3**	
Parameter	KOMO	SCvx	KOMO	SCvx	KOMO	SCvx
$\alpha_n = 0.1$	93	**100**	**87**	60	67	**93**
$\alpha_n = 0.05$	**100**	**100**	87	**93**	67	**100**
$\alpha_n = 0.01$	**100**	**100**	93	**100**	27	**100**

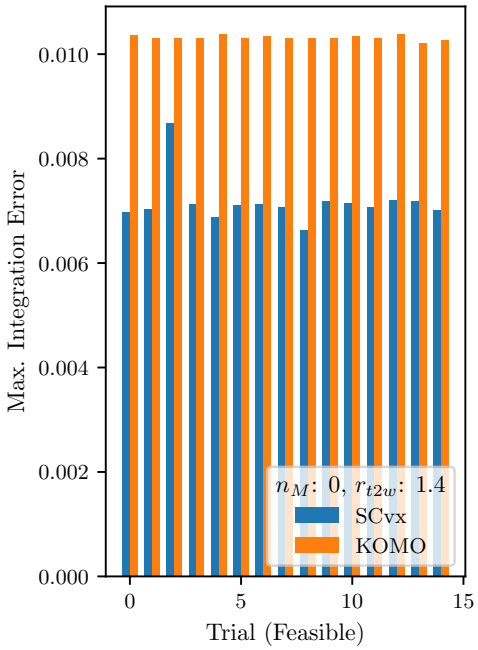
Table 4.3: Success rates of noise tests for scenario 1, 2 and 3.

In summary, KOMO has a similar success rate for scenario 1, a slightly worse success rate for scenario 2 and really struggles with finding solutions for scenario 3.

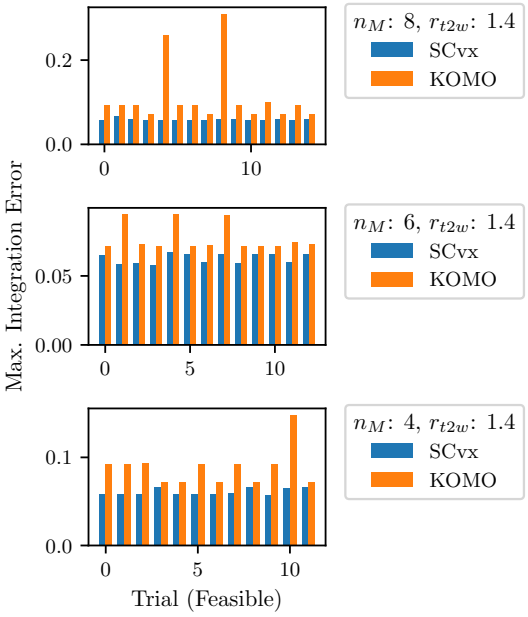
4.2.5 Integration Error

To approximate how well the obtained solutions respect the time-continuous dynamics of the used models, the computed input series is again propagated through the nonlinear dynamics. In contrast to the feasibility check described in section 4.1, the state at x_{k+1} is obtained propagating the constant input u_k with a reduced step size ($0.2\Delta t$) through the dynamics. An additional difference to the feasibility check is that an explicit Euler integration scheme is used for both algorithms. Even though KOMO implicitly enforces an integration scheme, this simplification was made to avoid using implicit methods, which require solving algebraic equations for the unknown new state. For linear dynamics, this can easily be done analytically. In contrast, nonlinear dynamics require numerical methods such as fix-point iterations to obtain the new state [70]. For the feasibility check, this was not an issue since the new state was already known and could be used for calculation. A smaller time grid for evaluation instead requires obtaining states at time points not considered during optimization. Using an integration scheme that was not used during optimization will lead to slightly higher errors for KOMO. Still, information about the feasibility regarding

time-continuous dynamics can be obtained. The maximum over all elements of the difference between the converged solution x^* and the propagated states is shown in the figure 4-10 and 4-11. Figure 4-10a presents the integration error for scenario 1. In scenario 1, the double integrator model is used, where the accelerations of the point mass are defined as the inputs and KOMO enforces the dynamic constraints by construction. Therefore, the observed integration error is purely related to the chosen integration scheme and is not considered in the evaluation. In figure 4-10b the integration error for scenario 2 is shown. In general KOMO and SCvx have a comparable integration error with KOMO's being slightly higher than that of SCvx. Only for scattered trials KOMO shows significantly higher errors. Figure 4-11 shows the integration error for scenario 3, where KOMO has for both thrust-to-weight ratios integration errors at least twice as high as SCvx. The maximal integration error is up to 52 times as high.



(a) Scenario 1



(b) Scenario 2

Figure 4-10: Integration error for scenario 1 and 2.

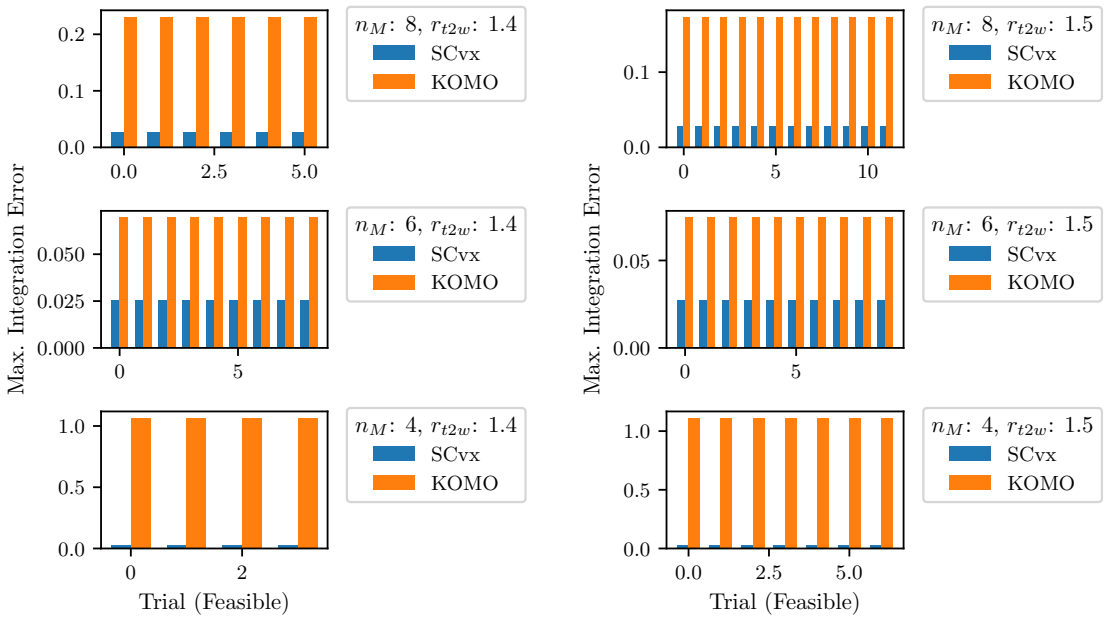


Figure 4-11: Integration error for scenario 3.

4.2.6 Preliminary Results for Multirotors with Three Motors

Tests involving multirotors with motor numbers $n_m = \{2, 3\}$ were not included in the comparison of the two algorithms since these problem settings are notoriously difficult to solve and no scenario was found where KOMO and SCvx were able to solve the same problem. This could have two reasons. Firstly the behavior of the multirotor becomes increasingly difficult to control for small motor numbers, including that not all goal states can be reached from all initial states. This leads to dynamic constraints, which render the problem such that feasible solutions are more difficult to reach and an increasing number of dynamically infeasible local minima. The second reason is related to the initial guess. Since the motion of the multirotor becomes more complex for lower motor numbers, finding a good initial guess is hard. For example, using multirotor models with $n_m = 3$ requires an ongoing rotation around the z-axis due to the torque imbalance caused by the rotor drag of an uneven number of rotors when moving in the x, y, z - directions. This is difficult to predict and a linear interpolation as an initial guess often leads to convergence to infeasible solutions. Nevertheless KOMO, in contrast to SCvx, was able to find solutions to simple problems including multirotor models with $n_m = 3$. One of the solutions is shown in appendix C.

4.3 Discussion

In the following section, the obtained experimental results are discussed. To begin with, it should be mentioned that the algorithmic performance of both SCvx and KOMO is highly dependent on the choice of the user-defined weights in the objective function and the parameters of the algorithms. Therefore, even though both algorithms were extensively tuned for each scenario to achieve peak performance, it cannot be guaranteed that there is no better parameter combination. In general, KOMO needs less extensive weight tuning for scenarios involving more geometric problems, such as scenarios 1 and 2. In contrast, for problems requiring highly dynamic solutions SCvx seems to be favorable regarding tuning effort (a subjective measure).

Since the computation time of KOMO is, in general, higher than the lower bound on the computation time of SCvx but smaller than the total time, no definitive statement can be made whether KOMO is faster than SCvx. An exception to this is scenario 2 with $n_M = 8$, where KOMO's computation time is comparable to the lower bound of SCvx. This indicates that KOMO has potentially lower run times for well-conditioned problems, but it is unclear when these occur. Nevertheless, for scenario 2 with $n_m = \{4, 6\}$ and scenario 3 can safely be assumed that a run time optimized SCvx-version has a shorter computation time. For scenario 1, it is unclear which algorithm would have the time advantage.

While KOMO is able to find in scenario 1 and 3 solutions with up to 2.4 % less cost than SCvx, in scenario 2 it finds solutions with under 1 % more cost. This indicates that KOMO in general finds equally good or better solutions compared to SCvx. One could argue that decreasing the convergence threshold of SCvx would lead to an increased computation time and less costly solutions, closing the gap to KOMO in terms of run time and regarding the obtained cost. This was tested and SCvx tends to converge to the same solutions even for substantially lower thresholds.

Comparing the success rates indicates that KOMO can find feasible solutions at a similar rate to SCvx for problems requiring more geometrical reasoning like scenarios 1 and 2. For problems demanding solutions close to the input constraints and, therefore, highly dynamic problems, KOMO shows a significantly lower success rate than SCvx. Note that deviations in the success rate of 7 % are caused by just one failing trial. This emphasizes that the chosen sample size introduces a coarse grid to evaluate the success rate and conclusions drawn from jumps in the success rate must be treated with caution. An increasing number of rotors renders the problem in a way that it is easier to solve for both algorithms. This correlates with the fact that multirotors with a higher number of rotors are easier to control. The experiments regarding the influence of noise on the initial guess show that both algorithms are similarly prone to noise on the initial guess. Except scenario 2, where SCvx finds significantly fewer solutions with increasing α_n and scenario 3, where KOMO's success rate increases with rising noise values.

In general, the dynamic constraints were violated if a solution was not feasible. How-

ever, both algorithms are effortlessly able to find solutions respecting the other constraints. Therefore the calculated integration error of the obtained solution is, to some degree, a measure of how difficult the problem was to solve for both algorithms. The integration errors of KOMO in scenario 2 are generally only slightly higher than the integration errors of SCvx. Since the integration scheme used to calculate the errors introduces higher errors for KOMO, both algorithms are assumed to find dynamically similarly feasible solutions. However, correlating with the observed low success rate, the integration errors of KOMO for scenario 3 are considerably higher than the errors of SCvx. The magnitude of the integration error for KOMO also indicates that the threshold for the dynamic constraint violations in the initial feasibility check was chosen too high. This supports the claim that KOMO is not well suited for highly dynamic problem settings.

Chapter 5

Conclusion

Multirotors have become increasingly often used in academia and industry. They are successfully applied in various fields, e.g., entertainment such as light shows, cooperative construction, inspection of power lines and off-shore wind parks as well as aerial additive manufacturing. However, since the motion planning problem for multirotors is high dimensional, search and optimization-based methods tend to be slow. On the other hand, optimization-based methods scale better with high dimensional state spaces and can solve motion planning problems fast. k-order Markov optimization is an optimization-based method that has proven its majority in robotic manipulation.

In this work, k-order Markov optimization was compared with successive convexification, an algorithm claimed to be well suited for motion planning problems involving highly dynamic systems with high dimensional state spaces. Therefore, a SCvx motion planning framework was implemented and rigorously validated. The framework can handle different dynamic models and arbitrary obstacles. Three multirotor motion planning scenarios have been formulated using the RAI-framework and the implemented SCvx-framework. KOMO and SCvx were used to solve different test scenarios which were geometrically as well as dynamically challenging. The obtained solutions were analyzed regarding computation time, cost, success rate, and continuous dynamical feasibility. KOMO was able to find solutions with comparable or lower cost than SCvx while having a slightly lower success rate for more geometrical problems and an up to 73% lower success rate for dynamically challenging problems. This leads to the claim that KOMO is well suited for problems requiring more geometrical reasoning. For problems requiring solutions where the control input reaches the control limits, KOMO struggles to find solutions but can solve dynamically complex problems that do not hit the input constraints.

To provide a more precise comparison between both methods, the implementation of SCvx in C++ has to be considered. This would provide clarification for problem types where it is unclear whether SCvx or KOMO has the time advantage. Also, introducing a more refined grid for evaluating the success rate using a higher trial number would reduce imprecision in evaluating the success rate for scenario 2. Introducing the final time as a decision variable would give the optimizers more freedom when

converging to a solution. Even though the optimization problem becomes harder to solve, influencing the step size could lead to higher success rates for highly dynamic problems for which KOMO struggles the most.

Bibliography

[1] Guinness World Records. Most unmanned aerial vehicals (UAVs) airborne simultaneously. 2021. URL https://www.guinnessworldrecords.com/news/commercial/2021/5/3281-drones-break-dazzling-record-for-most-airborne-simultaneously-655062.

[2] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar. Construction of Cubic Structures with Quadrotor Teams. 2011.

[3] Timur Uzakov, Tiago P Nascimento, and Martin Saska. UAV Vision-Based Nonlinear Formation Control Applied to Inspection of Electrical Power Lines. pages 1301–1308, 2020.

[4] Mahmood Shafiee, Zeyu Zhou, Luyao Mei, Fateme Dinmohammadi, Jackson Karama, and David Flynn. Unmanned aerial drones for inspection of offshore wind turbines: A mission-critical failure analysis. *Robotics*, 10(1):1–27, 2021. doi: 10.3390/robotics10010026.

[5] Ketao Zhang, Pisak Chermprayong, Feng Xiao, Dimos Tzoumanikas, Barrie Dams, Sebastian Kay, Basaran Bahadir Kocer, Alec Burns, Lachlan Orr, Christopher Choi, Durgesh Dattatray Darekar, Wenbin Li, Steven Hirschmann, Valentina Soana, Shamsiah Awang Ngah, Sina Sareh, Ashutosh Choubey, Laura Margheri, and Vijay M Pawar. Aerial additive manufacturing with multiple autonomous robots. 609(July 2019), 2022. doi: 10.1038/s41586-022-04988-4.

[6] Quan Quan. *Introduction to multicopter design and control*. Springer US, 2017. ISBN 9789811033827. doi: 10.1007/978-981-10-3382-7.

[7] Shivaji Anagandula. Picture of a fixed-wing aircraft model, 2022. URL https://grabcad.com/library/glider-fixed-wing-fox-3v-1.

[8] Belaid Amine. Picture of a single rotor blade helicopter model, 2018. URL https://grabcad.com/library/helicopter-97.

[9] Waliur Rahman Shikhon. Picture of a quadcopter model, 2017. URL https://grabcad.com/library/simple-quadcopter-frame-1.

[10] Michael Boyle. The Integration of Angular Velocity. *Advances in Applied Clifford Algebras*, 27(3):2345–2374, 2017. doi: 10.1007/s00006-017-0793-z.

[11] Sihao Sun, Angel Romero, Philipp Foehn, Elia Kaufmann, and Davide Scaramuzza. A Comparative Study of Nonlinear MPC and Differential-Flatness-Based Control for Quadrotor Agile Flight. pages 1–16, 2021. URL `http://arxiv.org/abs/2109.01365`.

[12] Danylo Malyuta, Taylor P. Reynolds, Michael Szmuk, Thomas Lew, Riccardo Bonalli, Marco Pavone, and Behcet Acikmese. Convex Optimization for Trajectory Generation. 2021. URL `http://arxiv.org/abs/2106.09125`.

[13] John H Reif. Complexity of the mover's problem and generalizations. *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 421–427, 1979.

[14] Daniel P. Scharf, Behçet Açíkmeşe, Daniel Dueri, Joel Benito, and Jordi Casoliva. Implementation and experimental demonstration of onboard powered-descent guidance. *Journal of Guidance, Control, and Dynamics*, 40(2):213–229, 2017. doi: 10.2514/1.G000399.

[15] Behçet Açikmeşe, Mi Mi Aung, Jordi Casoliva, Swati Mohan, Andrew Johnson, Daniel Scharf, David Masten, Joel Scotkin, Aron Wolf, and Martin W. Regehr. Flight testing of trajectories computed by G-FOLD: Fuel optimal large divert guidance algorithm for planetary landing. *Advances in the Astronautical Sciences*, 148(February 2014):1867–1880, 2013.

[16] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp. *The Mathematics Teacher*, 80(3):172, 2009.

[17] Nathan Ratliff, Matt Zucker, J. Andrew Bagnell, and Siddhartha Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 489–494, 2009. doi: 10.1109/ROBOT.2009.5152817.

[18] Marc Toussaint. *A tutorial on newton methods for constrained trajectory optimization and relations to SLAM, gaussian process smoothing, optimal control, and probabilistic inference*, volume 117. Springer, 2017. ISBN 9783319515472. doi: 10.1007/978-3-319-51547-2_15.

[19] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, volume 2. Society for Industrial and Applied Mathematics (SIAM), 2010. ISBN 9772081415.

[20] Kristoffer Bergman. *Exploiting Direct Optimal Control for Motion Planning in Unstructured Environments*. Number 2133. 2021. ISBN 9789179296773.

[21] Moritz Diehl, Hans Georg Bock, Holger Diedam, Pierre-brice Wieber, Moritz Diehl, Hans Georg Bock, Holger Diedam, Pierre-brice Wieber Fast, and Direct Multiple. Fast Direct Multiple Shooting Algorithms for Optimal Robot Control

To cite this version : Optimal Robot Control. *Fast Motions in Biomechanics and Robotics*, 2005.

[22] Stephen P Boyd. Sequential Convex Programming. *EE364b Lecture Notes Stanford University*, C:1–19, 2008. URL `http://www.stanford.edu/class/ee364b/lectures/seq_slides.pdf`.

[23] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. *Proceedings - IEEE International Conference on Robotics and Automation*, (May 2014):4569–4574, 2011. doi: 10.1109/ICRA.2011.5980280.

[24] Riccardo Bonalli and H Bruno. Optimal Control of Endo-Atmospheric Launch Vehicle Systems : Geometric and Computational Issues arXiv : 1710 . 11501v1 [math . OC] 31 Oct 2017. (November), 2017.

[25] David Mayne. A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems. *International Journal of Control*, 3 (1):85–95, 1966. doi: 10.1080/00207176608921369.

[26] Weiwei Li and Emanuel Todorov. Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems. *International Conference on Informatics in Control, Automation and Robotics Iterative*, pages 222–229, 2011. doi: 10.5220/0001143902220229.

[27] Zhaoming Xie, C. Karen Liu, and Kris Hauser. Differential dynamic programming with nonlinear constraints. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 695–702, 2017. doi: 10.1109/ICRA.2017.7989086.

[28] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1168–1175, 2014. doi: 10.1109/ICRA.2014.6907001.

[29] Taylor A. Howell, Brian E. Jackson, and Zachary Manchester. ALTRO: A Fast Solver for Constrained Trajectory Optimization. *IEEE International Conference on Intelligent Robots and Systems*, pages 7674–7679, 2019. doi: 10.1109/IROS40897.2019.8967788.

[30] Federico Augugliaro, Angela P. Schoellig, and Raffaello D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. *IEEE International Conference on Intelligent Robots and Systems*, pages 1917–1922, 2012. doi: 10.1109/IROS.2012.6385823.

[31] Yaser Alothman, Minhuan Guo, and Dongbing Gu. Using iterative LQR to control two quadrotors transporting a cable-suspended load. *IFAC-PapersOnLine*, 50(1):4324–4329, 2017. doi: 10.1016/j.ifacol.2017.08.861. URL `https://doi.org/10.1016/j.ifacol.2017.08.861`.

[32] Wesam Jasim and Dongbing Gu. Iterative linear quadratic regulator control for quadrotors leader-follower formation flight. *International Journal of Modelling, Identification and Control*, 31(2):152–160, 2019. doi: 10.1504/IJMIC.2019.097995.

[33] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. ACADO toolkit - An open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011. doi: 10.1002/oca.939.

[34] Markus Hehn and Raffaello D'Andrea. *Quadrocopter trajectory generation and control*, volume 44. IFAC, 2011. ISBN 9783902661937. doi: 10.3182/20110828-6-IT-1002.03178. URL `http://dx.doi.org/10.3182/20110828-6-IT-1002.03178`.

[35] Jeff Ferrin, Robert Leishman, Randy Beard, and Tim McLain. Differential flatness based control of a rotorcraft for aggressive maneuvers. *IEEE International Conference on Intelligent Robots and Systems*, pages 2688–2693, 2011. doi: 10.1109/IROS.2011.6048861.

[36] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011. doi: 10.1109/ICRA.2011.5980409.

[37] Richard M. Murray, Muruhan Rathinam, and Willem Sluis. Differential flatness of mechanical control systems: A catalog of prototype systems. *American Society of Mechanical Engineers, Dynamic Systems and Control Division (Publication) DSC*, 57-1:349–357, 1995.

[38] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments. *International Symposium of Robotics Research*, 17(5):459–465, 2013. doi: 10.1007/978-3-319-28872-7_37. URL `https://link.springer.com/chapter/10.1007/978-3-319-28872-7_37`.

[39] Florin Stoical, Vlad Mihai IvĂnuşca, Ionela Prodan, and Dan Popescu. Obstacle avoidance via B-spline parametrizations of flat trajectories. *24th Mediterranean Conference on Control and Automation, MED 2016*, pages 1002–1007, 2016. doi: 10.1109/MED.2016.7536053.

[40] Bahareh Sabetghadam, Rita Cunha, and António Pascoal. Real-time trajectory generation for multiple drones using bézier curves. *IFAC-PapersOnLine*, 53: 9276–9281, 2020. doi: 10.1016/j.ifacol.2020.12.2380.

[41] Klaus Höllig and Jörg Hörner. *Approximationand Modelingwith B-Splines*. Society for Industrial and Applied MathematicsPhiladelphia, 2013. ISBN 978-1-611972-94-8. URL `https://epubs.siam.org/doi/epdf/10.1137/1.9781611972955.ch2`.

[42] Florin Stoican, Ionela Prodan, and Dan Popescu. Flat trajectory generation for way-points relaxations and obstacle avoidance. *2015 23rd Mediterranean Conference on Control and Automation, MED 2015 - Conference Proceedings*, pages 695–700, 2015. doi: 10.1109/MED.2015.7158827.

[43] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. *IEEE International Conference on Intelligent Robots and Systems*, 2017-Septe: 2872–2879, 2017. doi: 10.1109/IROS.2017.8206119.

[44] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-Based Motion Planning for Aggressive Flight in SE(3). *IEEE Robotics and Automation Letters*, 3(3):2439–2446, 2018. doi: 10.1109/LRA.2018.2795654.

[45] Mihail Pivtoraiko and Alonzo Kelly. Generating Near Minimal Spanning Control Sets for Constrained Motion Planning in Discrete State Spaces. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005. doi: 10.1109/IROS.2005.1545046.

[46] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761.

[47] Steven M. LaValle. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998.

[48] Steven M. LaValle. Planning algorithms. *Planning Algorithms*, 9780521862059: 1–826, 2006. doi: 10.1017/CBO9780511546877.

[49] Lydia E. Kavraki, Petr Švestka, Jean Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996. doi: 10.1109/70.508439.

[50] Brendan Burns. Toward Optimal Configuration Space Sampling. *Robotics: Science and Systems*, 2005. doi: 10.15607/RSS.2005.I.015.

[51] David Hsu. The Bridge Test for Sampling Narrow Passages with Probabilistic Roadmap Planners. *IEEE International Conference on Robotics and Automation*, pages 4420–4426, 2003.

[52] Valerie Boor, Mark H. Overmars, and Frank A. Van der Stappen. The Gaussian Sampling Strategy for Probabilistic Roadmap Planners. *International Conference on Robotics and Automation*, 1999.

[53] James J Kuffner and Steven M Lavalle. RRT-Connect : An Efficient Approach to Single-Query Path Planning. *IEEE International Conference on Robotics and Automation*, (April):995–1001, 2000.

[54] Zhiling Tang, Bowei Chen, Rushi Lan, and Simin Li. Vector Field Guided RRT * Based on Motion Primitives for Quadrotor Kinodynamic Planning. *Journal of Intelligent and Robotic Systems*, pages 1325–1339, 2020.

[55] Dustin J Webb. Kinodynamic RRT *: Asymptotically Optimal Motion Planning for Robots with Linear Dynamics. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5054–5061, 2013.

[56] F Palacios-Gomez, L Lasdon, and M Engquist. Nonlinear Optimization by Successive Linear Programming. *Management Science*, 1982. doi: 10.1287/mnsc.28.10.1106.

[57] Paul T Boggs and Jon W Tolle. Sequential Quadratic Programming . *Acta Numerica*, 4:1–51, 1995. doi: 10.1017/S0962492900002518.

[58] Amir Beck and Aharon Ben-tal Luba. A sequential parametric convex approximation method with applications to nonconvex truss topology design problems. pages 29–51, 2010. doi: 10.1007/s10898-009-9456-5.

[59] Wei Wei, Jianhui Wang, Senior Member, Na Li, and Shengwei Mei. Optimal Power Flow of Radial Networks and its Variations : A Sequential Convex Optimization Approach. *IEEE Transactions on Smart Grid*, 8:1–14, 2017. doi: 10.1109/TSG.2017.2684183.

[60] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. cambridge university press, 2004. ISBN 9780521833783.

[61] Yuanqi Mao, Michael Szmuk, and Behcet Acıkmes. Successive Convexification of Non-Convex Optimal Control Problems and Its Convergence Properties. *IEEE 55th Conference on Decision and Control (CDC)*, 2016. doi: 10.1109/CDC.2016.7798816.

[62] Yuanqi Mao, Michael Szmuk, Xiangru Xu, and Behcet Acikmese. Successive Convexification: A Superlinearly Convergent Algorithm for Non-convex Optimal Control Problems. pages 1–35, 2018. URL `http://arxiv.org/abs/1804.06539`.

[63] Daniel Liberzon. *Calculus of Variations and Optimal Control Theory A Concise Introduction*. N.J: Princeton Univer- sity Press, 2012. ISBN 9781400842643.

[64] E.B. Dam, M. Koch, and M. Lillholm. Quaternions, interpolation and animation. 1998. URL `https://books.google.es/books?id=8SIXGwAACAAJ`.

[65] Josep Virgili-Llop, Costantinos Zagaris, Richard Zappulla, Andrew Bradstreet, and Marcello Romano. Convex optimization for proximity maneuvering of a spacecraft with a robotic manipulator. *Advances in the Astronautical Sciences*, 160(September):1059–1078, 2017.

[66] Marc Toussaint. Newton methods for k-order Markov Constrained Motion Problems. 2014. URL `http://arxiv.org/abs/1407.0414`.

[67] Marc Toussaint. KOMO Tutorial, 2020. URL https://marctoussaint.github.io/robotics-course/komoTutorial.html.

[68] Jorge Nocedal, Stephen J Wright, and Stephen M Robinson. *Numerical Optimization*. Springer, 2006. ISBN 9780387303031.

[69] Marc Toussaint. Lecture notes: Optimization Algorithms, 2020.

[70] John C Butcher. *Numerical Methods for Ordinary Differential Equations Second Edition*. 2008. ISBN 9780470723357.

Appendix A

Explicit Features Used in the KOMO-Implementation

The features used to implement the motion planning problem for the full model are listed in table A.2 and the features used for the double integrator implementation are described in A.1. The first column specifies which constraint is implemented, while the second column states which feature is used. The "Order" column states the order of the feature and the time steps on which the constraint is applied are defined in the "Time Step" column. Finally, the constraint's type (eq./ineq.) and the specified targets are shown in the last column. A dash is drawn in the corresponding row to indicate that a constraint is not enforced explicitly.

Constraint	**Feature**	**Order**	**Time Step**	**Target/ Type**
Initial Constraints:				
$p_{k=0} = p_0$	-	-	-	-
$v_{k=0} = v_0$	Position	1	0	$= v_0$
Terminal Constraints:				
$p_{k=N} = p_T$	Position	0	1	$= p_T$
$v_{k=N} = v_T$	Position	1	1	$= v_T$
Dynamic Constraints:				
$x_{k+1} = f_{nl}(x_{k+1}, x_k, u_k)$	Implicitly Enforced by KOMO-Formulation and Box-Constraints	-	-	-
Collision Constraints:				
$S_k \cap O_i = \emptyset,$ $\forall i \in \{1, ..., n_{obs}\}$	Signed Distances	-	[0...1]	≤ 0
State and Input Box-Constraints:				
$p_k \leq p_{max}$	Position	0	[0...1]	$\leq p_{max}$
$p_k \geq p_{min}$	Position	0	[0...1]	$\geq p_{min}$
$v_k \leq v_{max}$	Position	1	[0...1]	$\leq v_{max}$
$v_k \geq v_{min}$	Position	1	[0...1]	$\geq v_{min}$
$u_k \leq u_{max}$	Position	2	[0...1]	$\leq u_{max}$
$u_k \geq u_{min}$	Position	2	[0...1]	$\geq u_{min}$
Objective	**Feature**	**Order**	**Time**	**Target/ Type**
Cost on Inputs:				
$\sum_k u_k^2$	Position	2	[0...1]	sum of squares

Table A.1: Features used to define the motion planning problem with KOMO for the double integrator model.

Constraint	**Feature**	**Order**	**Time Step**	**Target/ Type**
Initial Constraints:				
$p_{k=0} = p_0$	-	-	-	-
$v_{k=0} = v_0$	Position	1	0	$= v_0$
$q_{k=0} = q_0$	-	-	-	-
$\omega_{k=0} = \omega_0$	Orientation	1	0	$= \omega_0$
Terminal Constraints:				
$p_{k=N} = p_T$	Position	0	N	$= p_T$
$v_{k=N} = v_T$	Position	1	N	$= v_T$
$q_{k=N} = q_T$	Orientation	0	N	$= q_T$
$\omega_{k=N} = \omega_T$	Orientation	1	N	$= \omega_T$
Dynamic Constraints:				
$x_{k+1} = f_{nl}(x_{k+1}, x_k, u_k)$ (implicitly)	Violations of Newton-Euler Equations	2	[0...N]	$= 0$
Collision Constraints:				
$S_k \cap O_i = \emptyset$, $\forall i \in \{1, ..., n_{obs}\}$	Signed Distances	2	[0...N]	≤ 0
State and Input Box-Constraints:				
$p_k \leq p_{max}$	Position	0	[0...N]	$\leq p_{max}$
$p_k \geq p_{min}$	Position	0	[0...N]	$\geq p_{min}$
$v_k \leq v_{max}$	Position	1	[0...N]	$\leq v_{max}$
$v_k \geq v_{min}$	Position	1	[0...N]	$\geq v_{min}$
$q_k \leq q_{max}$	Orientation	0	[0...N]	$\leq q_{max}$
$q_k \geq q_{min}$	Orientation	0	[0...N]	$\geq q_{min}$
$\omega_k \leq \omega_{max}$	Orientation	1	[0...N]	$\leq \omega_{max}$
$\omega_k \geq \omega_{min}$	Orientation	1	[0...N]	$\geq \omega_{min}$
$u_k \leq u_{max}$	Forces Acting	-	[0...N]	$\leq u_{max}$
$u_k \geq u_{min}$	Forces Acting	-	[0...N]	$\geq u_{min}$
Objective	**Feature**	**Order**	**Time**	**Target/ Type**
Cost on Inputs:				
$\sum_k u_k^2$	Forces Acting	-	[0...N]	sum of squares

Table A.2: Features used to define the motion planning problem with KOMO for the full model.

Appendix B

Obtained Trajectory for Scenario 2 and 3

In figure B-1 the trajectory obtained with KOMO for scenario 3 is shown and in figure B-2 the trajectory obtained for scenario 2. The converged trajectory is shown at the top left, the initial state is pictured at the top in the middle and the final state is shown at the bottom right.

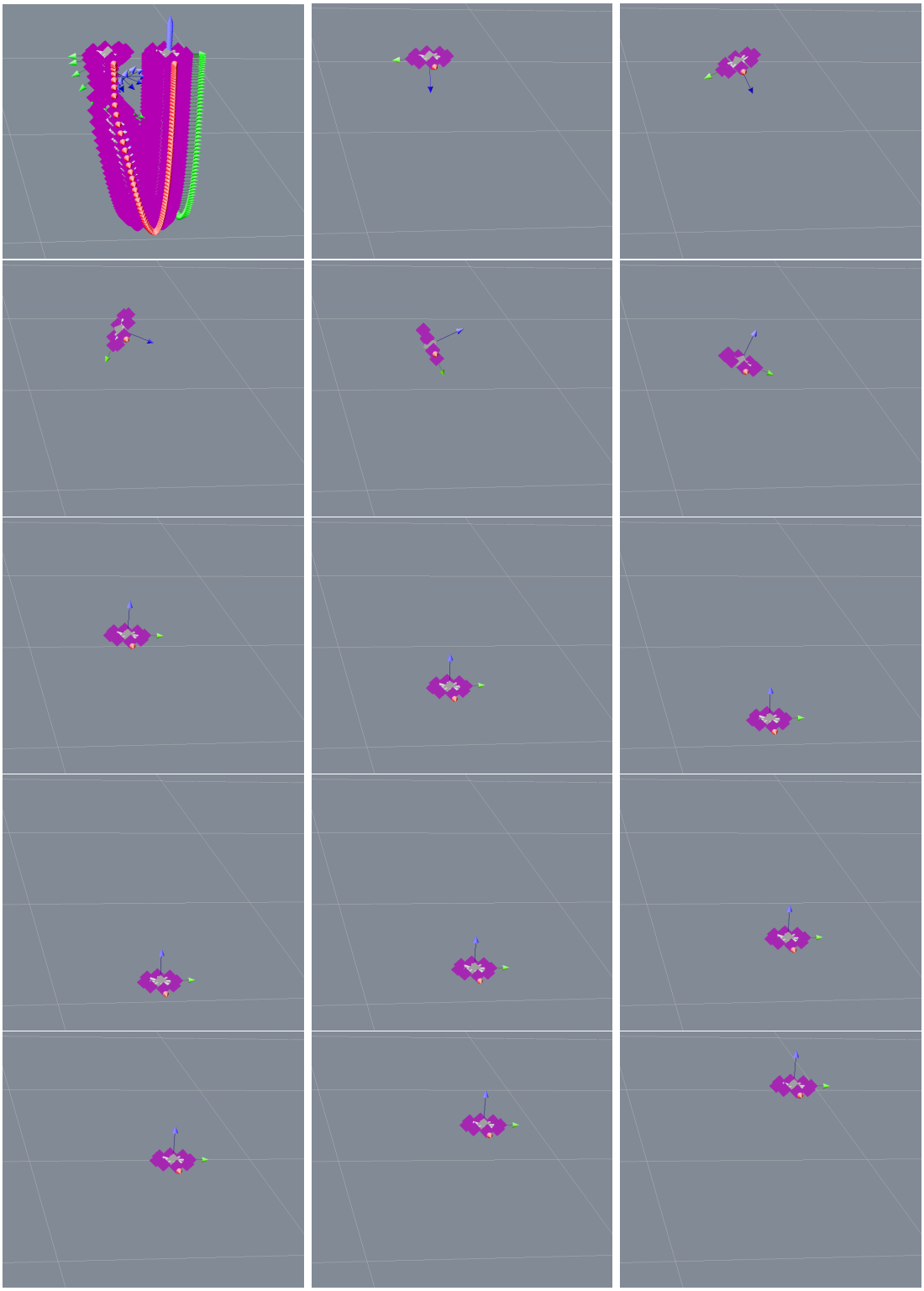


Figure B-1: Trajectory obtained by KOMO for scenario 3 with $n_m = 8, r_{t2w} = 1.4$.

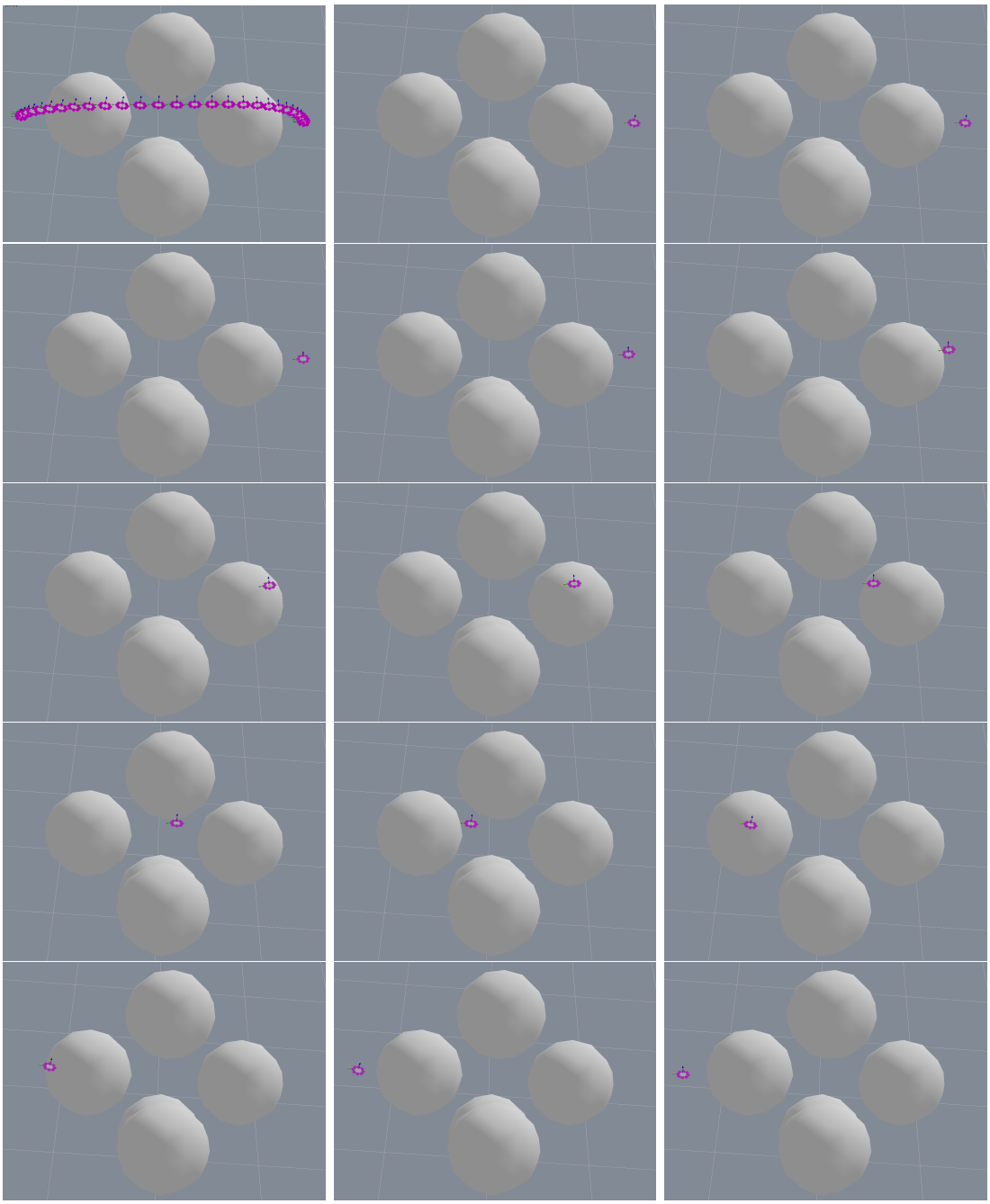


Figure B-2: Trajectory obtained by KOMO for scenario 2 with $n_m = 8, r_{t2w} = 1.4$.

Appendix C

Preliminary Results for Multirotors Using Three Motors

In figure C-1 and figure C-2, the with KOMO obtained forces and states for an example using a multirotor model with only three motors are shown. The scenario parameters are shown in table C.1, note that constraints on the final orientation as well as final and path constraints on the rotational velocity had to be removed because of the rotation around the z-axis necessary to reach the goal position.

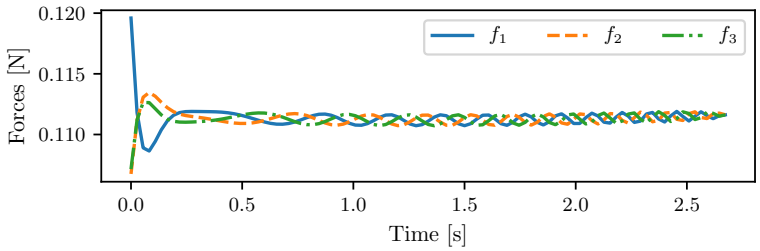


Figure C-1: Obtained inputs for tests including Multirotors with $n_m = 3$.

	Underactuated Multirotors
Obstacles	-
Initial State (x_0)	$p_0 = (0, 0, 1)$ $q_0 = (1, 0, 0, 0)$ $v_0, \omega_0 = (0, 0, 0)$
Final State (x_T)	$p_T = (0, 1, 1)$ $v_T = (0, 0, 0)$ $q_T, \omega_T = -$
Final Time (T) [s]	2.7
#Time Steps (N)	100
Noise Factor (α_n)	0.01
Thrust to Weight Ratio (r_{t2w})	1.4
# Motors (n_m)	3

Table C.1: Scenario parameters for tests including multirotors with $n_m = 3$.

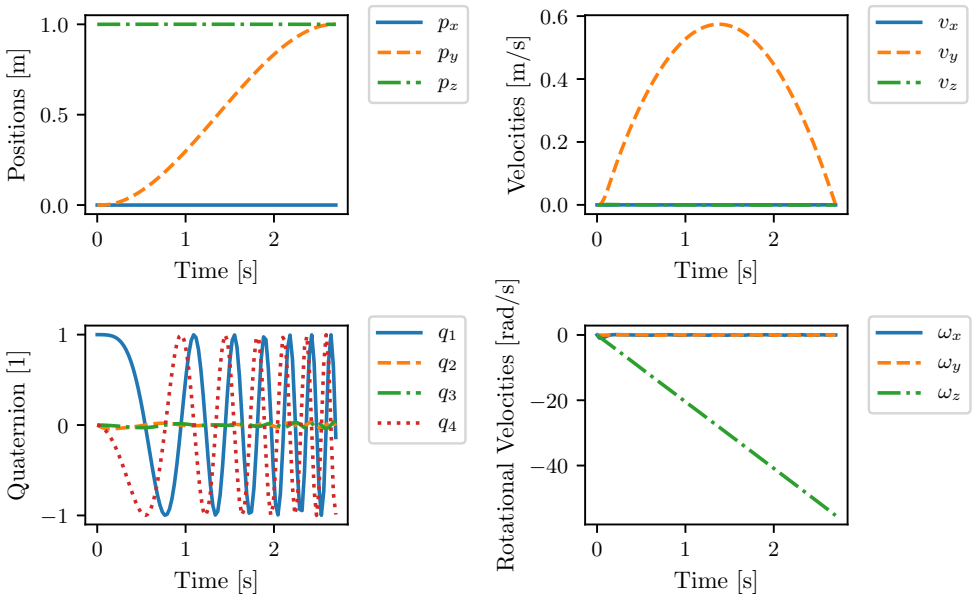


Figure C-2: Converged state trajectory for tests including multirotors with $n_m = 3$.