LULEÅ UNIVERSITY OF TECHNOLOGY

MASTER'S THESIS

On-Board Convex Optimization for Powered Descent Landing of EAGLE

Author:
Andreas WENZEL

Supervisor:
Dr. David SEELBINDER
Dr. Anita ENMARK

A thesis undertaken within:

GNC department
Institute of Space Systems
Deutsches Zentrum für Luft- und Raumfahrt (DLR), Bremen, Germany

Submitted in partial fulfillment of the requirements for the programme of:

Master in Space Science and Technology
Computer Science, Electrical and Space Engineering
Luleå University of Technology

space master

as part of the
Joint European Master in Space Science and Technology (SpaceMaster)

04th September 2017

Abstract

Future space exploration missions require new solutions in Guidance, Navigation and Control for autonomous high precision landing. The German Aerospace Centre, DLR, is currently developing the environment for autonomous GNC Landing experiments, EAGLE, acting as a demonstrator for vertical take-off and landing. The goal of this thesis is to develop a prototype real-time applicable guidance function based on optimal control theory for the powered descent landing, which can be implemented and tested on the on-board computer of EAGLE.

Based on the principle of loss less convexification developed by Açikmese and Ploen [1], the powered descent landing fuel-optimal control problem is converted into a second order cone problem. A discretization and transcription method is designed in order to solve the resulting non-linear program by means of the embedded conic solver ECOS and the developed algorithm is verified by the comparison of simulation results for an example pinpoint landing on Mars from [1].

After the implementation into C-code and a Simulink-Environment, the developed method is adapted by a preceding heuristic estimation for the fuel-optimal flight time given initial and final conditions, which is used as input for the developed trajectory optimization algorithm. This results in a prototype, sub-optimal trajectory optimization and guidance function applicable on on-board systems. The guidance function is tested with EAGLE-specific parameters in two extensive simulation of 41503 sets, respectively, with different initial and final conditions. For all the sets, the resulting trajectory and fuel consumption calculated by the guidance function are compared to the actual fuel-optimal solution, which is obtained via a search algorithm scanning through different flight times in a given range around the estimated flight time.

We find that the developed guidance function overestimates the optimal flight time in all cases, related to a fuel consumption which is 20% higher than the optimal case in 80% of the sets. Furthermore, the typical computation time of the guidance function on the on-board computer of EAGLE can be expected to be less than 0.8 seconds, proving the on-board feasibility of the proposed algorithm. The developed prototype guidance function sets a base for further work on on-board optimization applications and is currently subject to improvement regarding optimality and robustness.

Acknowledgements

This work has been enabled by Dr. David Seelbinder and the GNC-department at the institute of spacesystems of DLR. I want to thank him and Dr. Stephan Theil for giving me the opportunity to work on the exciting topic on convex optimization and on EAGLE. The support I received from Dr. Seelbinder throughout the thesis work, especially his explanations in the frequent meetings throughout my work at DLR, helped me to build up my understanding of optimal control and topics related to on-board computing. Thanks to him, I will continue the work that has started with this thesis. I also want thank Prof. Peter von Ballmoos and Dr. Anita Enmark for examining this thesis work and for the organizational support throughout the project.
Moreover, I want to thank the members of the GNC department for the entertaining lunch breaks, which made the intensive work much more bearable. A special "thank you" belongs to my friends from university, especially the spacemaster students, as well as to my family and friends outside of university, who always supported me.

Contents

1 Introduction

1.1 Motivation

Recent developments in space explorations have resulted in impressive demonstrations of the performance of automative space systems. In particular the successful application of partially reusable launcher systems such as SpaceX's commercial Falcon 9, which has been designed for the launch of satellites into several different orbits and for launching the Dragon spacecraft for cargo transportation to the ISS, has received high public attention. In 2015, a first stage of a Falcon 9 launcher succesfully landed autonomously in Cape Canaveral after it had been launched and separated from the second stage. Another succesfull landing of a Falcon's first stage was demonstrated in 2017 by landing on a platform based on a boat. Those unprecedented landings of a launcher system are expected to enable cheap and frequent launch possibilities, which is especially of interest for commercial space companies. The reusability of spacecrafts and launch systems may also be a key technology for future manned or unmanned missions to other planets like Mars.

In addition, several scientific exploration missions to other bodies of the solar system require precise landing on a planet's surface to meet specific scientific objectives. Missions that include the landing of a vehicle within a few meters of accuracy are very challenging and have to make use of robust high precision guidance, navigation and control (GNC) solutions. Accomplishing such autonomous missions would be impossible without the high performance level of new computer technologies, i.e. on-board computer architectures, which allow the implementation of high fidelity GNC-solutions.
Inherent to both goals, landing reusable launchers on Earth or an exploration lander on another planetary object like Mars, the final phase of approaching the landing site can be divided into several elements: All missions involving a landing space vehicle equipped with one or more engines share an atmospheric re-entry followed by a descent, which may include the use of a parachute for deceleration, and a final powered descent landing, where the engine is ignited to decelerate the vehicle and to maneuver towards a desired final position. This work is focused on the powered descent landing of a spacecraft. Due to the complexity and timescale of the powered descent landing problem, the overall system requires high autonomy and the flight path may have to be recalculated several times during the powered descent phase. In special cases it might even be necessary that the system autonomously selects another suitable landing site in case of any hazard detected at the initially planned landing area during the descent. Therefore, the landing trajectory has to be recalculated several times on the on-board computer of the landing vehicle. While previous missions like Mars exploration rovers required a landing accuracy of around 35 km or even 10 km in the case of the famous Mars Science Laboratory with the rover Curiosity [3], landing a vehicle within a few meters requires new on-board capable trajectory optimization techniques.

Calculating the optimal trajectory for the powered descent landing can be seen as an optimal control problem involving the dynamics of the landing vehicle and several constraints on the control parameters and the actual trajectory, while some relevant parameter is minimized. In the example case of fuel-optimal landing, the amount of consumed fuel has to be minimized. However, the optimal control problem is in general not on-board feasible, since it contains time continuous functions. For the numerical solution of such a problem a discretization and transcription is necessary, which converts the inifinte dimensional optimal control problem into a problem of finite dimension. The solution algorithms for such problems, i.e. non-linear programming, do not in general provide convergence towards the optimal solution and the computation time of corresponding solvers may be too long for real-time applications. In 2007, Açikmese and Ploen provided a mathematical methodology called "lossless convexification" for the powered descent landing problem, transforming it into a convex optimal control problem, i.e. a second order cone problem (SOCP) [1]. This SOCP formulation of the powered descent landing problem has a global solution, and interior point methods for solving convex problems guarantee the convergence towards this solution. Therefore, the resulting SOCP formulation provides an onboard feasible approach for real-time trajectory optimization of the powered descent landing.

1.2 Outline

This thesis focuses on the development of a convex minimum-fuel trajectory optimization algorithm for on-board application on the vertical take-off and landing demonstrator called *Environment for Autonomous GNC Landing Experiments*, EAGLE, of the German Aerospace Centre DLR. The goal is to create a prototype on-board guidance function making use of the lossless convexification method by Açikmese and Ploen and solve the resulting convex optimization problem for the powered descent landing by means of the open source *embedded conic solver* ECOS [9]. Therefore, the convexified optimal control problem is discretized and transcribed making use of full discretization and trapezodial scheme, such that it is numerically solvable. Furthermore, the transcription has to take into account the formalism for optimal control problems expected by ECOS.
The developed optimization method is extended in order to obtain an ob-board applicable guidance function, that can provide optimal, or close to optimal, reference trajectories to the control system of EAGLE.

The structure of this document is as follows: A short introduction to the EAGLE system in section 2.1 and a summary of the mathematical basis required for the transcription of the optimization problem, especially regarding convex and conic optimization, is followed by the derivation of the SOCP formulation of the powered descent landing problem following [1] in chapter 3. Based on this, a transcription method is developed in chapter 4, as well as a complete convex trajectory optimization algorithm applying ECOS. This includes a test and verification of the algorithm with an example of Mars pin-point landing. In chapter 5 the created algorithm is implemented in C and extended to a prototype guidance function, including a quasi-heuristic estimation for the optimal flight time. This guidance function is then tested in chapter 6, followed by an outlook and a final conclusion.

2 Background

2.1 Vertical Take-Off and Landing Demonstrator - EAGLE

Although the main focus of this work lies on the mathematical aspects and generic implementation of an on-board feasible trajectory optimization algorithm, one of the future goals of this work is to implement and test the created algorithm on-board of EAGLE. EAGLE is a vertical take-off and landing vehicle designed as a platform for testing and demonstrating new GNC algorithms related to soft-landing, smooth ascent and hovering. It provides dynamics, sensors and actuators similar to a typical landing vehicle. Hence it can be used to test new GNC solutions, e.g. for space exploration missions, in a realistic environment.
The EAGLE system mainly consists of the main structure housing all relevant sub-systems, like sensors, on-board computer and power supply. There are three legs attached to this housing. The actuation mainly consists of a jet engine generating the main thrust. Its magnitude is controlled by the fuel flow into the engine's burning chamber, while the thrust direction is controlled via two vanes aligned perpendicular below the engine. Those vanes deflect the thrust for maneuvers in pitch- and yaw-direction. For roll-control, a cold-gas reaction control system is applied with actuators placed on the two lever arms of EAGLE. These actuators are electro-magnetic on/off-valves which control the ejection of the pressurized gas, and thus they control the torque acting along the roll direction [10].

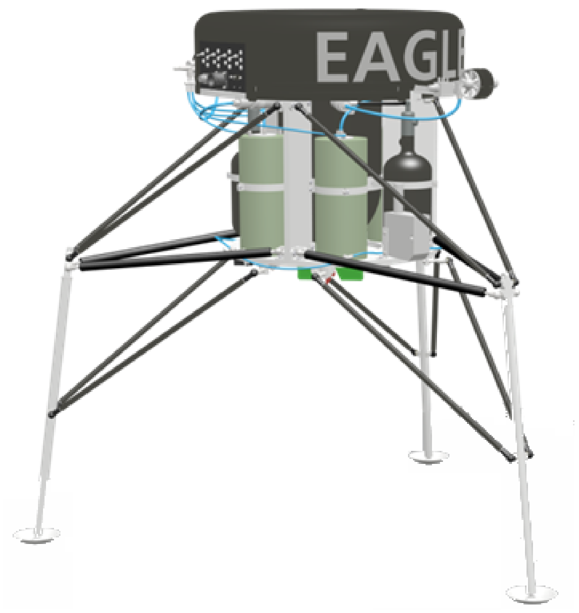

FIGURE 2.1: Design model of EAGLE [10]

The wet mass of EAGLE is about 30 kg including four tanks filled with 1.5 liter of cerzozine as fuel. In figure 2.1 , the symmetrical mounting of the green tanks within the main housing of EAGLE is visible.
For navigation purposes, EAGLE contains an inertial measurement unit (IMU), a GPS receiver, a laser altimeter and a magnetometer. The measurements of these sensors are fused for precise navigation. To overcome problems during tests with EAGLE in the vicinity of the DLR laboratory, i.e. unsufficient GPS-satellite visibility, an optical navigation solution has been developed using camera images for navigation within the EAGLE test environment.

Furthermore, a control system has been developed including two different strategie [10]s. The general system consists of a guidance function providing a reference trajectory. This reference is used as the command for the EAGLE control system and compared to the actual position and velocities estimated by the navigation subsystem. An outer loop calculates a reference quaternion fed to the inner loop controller [10]. As mentioned above, there are two strategies which are implemented into this inner loop controller . First, a simple PD-controller which is hand-tunable was developed to allow for simple testing. The second method is based on sliding-mode control taking into account uncertainties resulting from the ongoing development of EAGLE like uncertainties on mass and moment of intertia [10]. One can easily switch between these two controllers.
The on-board computer runs the operating system *QNX*. A MATLAB /Simulink model including the desired GNC algorithms is created and converted into C-code via the Simulink Coder. This C-code is then cross-compiled and finally executed by *QNX* on-board of EAGLE. Throughout this thesis work, the specific requirements especially regarding on-board computer and control system are taken into account for a follow-up implementation of the prototype guidance function developed in this thesis.

2.2 Mathematical Preliminaries

The work presented in this thesis focuses on real-time trajectory optimization and the implementation of a guidance function for usage in the control system of EAGLE. To understand the underlying principles, this mathematical preparation is an introduction to the fundamental concepts of optimal control and provides the basic definitions required for understanding the applied algorithms. It will cover the definitions towards a general and a convex optimal control problem. Also, the structure of SOCP including the equivalent formulations relevant for the proposed transcription algorithm are introduced.

2.2.1 Optimal Control Problems

The optimal control problem (OCP) considers the optimization of a controlled process. The state of the system at time t is described via the state variable $x(t) \in \mathbb{R}^{n_x}$ while the control variable is given by $u(t) \in \mathbb{R}^{n_u}$, where n_x and n_u are the dimensions of the state and the control variable. The optimal control process in general depends on an independent variable t, which can be associated with time although t can have different meanings in general. One restricts the optimal control process to an interval $t \in [t_0, t_f]$, where the start time t_0 is fixed and the final time t_f can be fixed or free. Systems having a free end time are called autonomous systems. The optimal control problem can be written in its general formulation as:

Problem 1

$$\min_{x,u,t} \quad J = \phi(x(t_0), x(t_f)) + \int_{t_0}^{t_f} f_0(t, x(t), u(t)) \mathrm{d}t \tag{2.1}$$

$$\text{s.t.} \quad \dot{x}(t) = f(t, x(t), u(t)) \tag{2.2}$$

$$\psi_0 \left(x\left(t_0\right)\right) = 0 \tag{2.3}$$

$$\psi_f(x(t_f)) = 0 \tag{2.4}$$

$$c(x(t), u(t), t) \leq 0 \ \forall t \in [t_0, t_f] \tag{2.5}$$

The scalar valued function $J : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times [t_0, t_f] \to \mathbb{R}$ is called cost or objective function measuring the performance of a solution. Equation 2.2 is the dynamics of the problem given by the ordinary differential equation $\dot{x}$, which in our case will be only a linear function of the state and the control vector via:

$$\dot{x}(t) = A^{dyn}x(t) + B^{dyn}u(t), \tag{2.6}$$

where $A^{dyn} \in \mathbb{R}^{n_x} \times \mathbb{R}^{n_x}$ and $B^{dyn} \in \mathbb{R}^{n_u} \times \mathbb{R}^{n_u}$ are called system matrices defining the dynamics. In addition, equations 2.3 and 2.4 define conditions for the state vector at the beginning and end. There are also possible constraints, called path constraints, given by the general formulation in equation 2.5, which includes constraints on the state or on the control, respectively, and mixed constraints on both throughout the defined time interval. It is useful to emphasize the difference between the two vectors for the state and the control variable. The state vector $x(t)$ can be seen as the vector that is differentiated in the dynamics of the system, while the control appears as algebraic values in the algebraic equation [5].

The cost function J consists of two main elements: The first part is the so called Mayer term $\phi(x(t_0), x(t_f))$, which is a function that only depends on the initial and the final state of the system. The second function is the so called Lagrange term, being the integral of a functional f_0. Depending on which of these two parts is not vanishing, the optimization problem is said to have Mayer or Lagrange form. If both are not vanishing, it is a so called Bonza-problem. However, it can be shown that all three formulations are equivalent, meaning that they can be transformed into each other [5].

2.2.2 Non-Linear Programming, Transcription and Discretization

An optimal OCP usually considers continuous state trajectories and piecewise continuous control functions, and as such it is infinite dimensional. However, systems for the numerical solution of such optimization problems require a finite set of variables and constraints. So called direct and indirect approaches are used to overcome this problem and to solve the OCP. The latter formulates the OCP as a boundary value problem and uses Pontryagin's minimum principle for checking the necessary optimal conditions. This approach is also known as "optimize then discretize" [5]. In contrast, direct methods convert the OCP into a non-linear program (NLP) of finite dimension in order to solve it numerically, which is referred to as transcription. One focus of the work presented in this thesis is on the transcription and solution of an OCP, and thus on direct methods which are discussed in further detail.
The first step for transcribing an OCP into an NLP is to construct a stage vector $y \in \mathbb{R}^{n_x+n_u}$ from the state and control variables:

$$y(t) = \begin{pmatrix} x(t) \\ u(t) \end{pmatrix}. \tag{2.7}$$

The additional crucial step is to discretize the trajectory with a grid of nodes. Note that the points do not have to be equidistant, and that depending on the problem and applied numerical method it might be even adventageous to adapt the separation between the grid points. Within the framework presented in this report, the time interval $[t_0, t_f]$ is divided into N equidistant discrete time points $t_i \in [t_0, t_f], i \in [0, N]$ with

$$t_0 = t_0 \leq t_1 \leq ... \leq t_{N-1} \leq t_N = t_f \tag{2.8}$$

and the equal separation

$$\Delta t = t_{i+1} - t_i = \Delta t = \frac{t_f - t_0}{N}. \tag{2.9}$$

From this discretization one obtains a stage variable $y(t_i) = y_i$ for each node i, called the stage variable. The optimization vector $y \in \mathbb{R}^{N\cdot(n_x+n_y)}$ containing the entire trajectory is then composed of the discretized stage variable

$$y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} x_0 \\ u_0 \\ x_1 \\ u_1 \\ \vdots \\ x_N \\ u_N \end{pmatrix}.$$

In this way, the OCP representing an infinite-dimensional optimization problem can be transformed to a finite-dimensional optimization problem. The size of the finally resulting NLP, as well as the computation time and the obtained solution accuracy, strongly depend on the chosen number of nodes of the discretization. The choice of integration scheme for the dynamics $\dot{x}$ has an additional impact on these parameters but also on the complexity of the actual NLP. As an example, the Euler scheme or trapezoidal scheme are simple and quick integration schemes, of which the latter will be used within this work:

$$x(t_{i+1}) = x(t_i) + dt \cdot \dot{x}(t_i) \qquad \text{Euler Scheme} \tag{2.10}$$

$$x(t_{i+1}) = x(t_i) + \frac{dt}{2}\left(\dot{x}(t_i) + \dot{x}(t_{i+1})\right) \qquad \text{Trapezoidal Scheme} \tag{2.11}$$

Applying the trapezoidal integration scheme together with the discretization described above on the general optimal control problem 1 results in a discretized version of the optimal control problem:

Problem 2

$$\min \qquad J = \phi(x(t_0), x(t_N)) + \frac{\Delta t}{2}\sum_{i=1}^{N-1}\left[f_0(x(t_i), u(t_i)) + f_0(x(t_{i+1}), u(t_{i+1}))\right] \tag{2.12}$$

$$\text{s.t.} \qquad x(t_{i+1}) = x(t_i) + \frac{dt}{2}\left(\dot{x}(t_i) + \dot{x}(t_{i+1})\right) \tag{2.13}$$

$$\psi_0\left(x\left(t^0\right)\right) = 0 \tag{2.14}$$

$$\psi_f(x(t_N)) = 0 \tag{2.15}$$

$$c(x(t^i), u(t_i) \leq 0, i \in [0, N] \tag{2.16}$$

The actual transformation of this discretized OCP towards an NLP also depends on the final chosen direct transcription with the three main methods single shooting, multiple shooting and collocation:

Single Shooting

In the single shooting method, the trajectory is calculated by integrating the dynamics starting from an initial state x_0 and using the control u_i at each point as decision variables, while it is also possible to include the initial or final state x_0 and x_N in the set of decision variables in the case of free dimensions in those states. The NLP is solved by iteratively finding the trajectory which follows the path constraints on u_i and which minimizes the so called single shooting defect D [5], defined as the difference between the final state $\tilde{x}_N$ of the resulting integrated trajectory to the desired final state x'_N

$$D = \tilde{x}_N - x'_N = 0. \tag{2.17}$$

Multiple Shooting

In multiple shooting, the control is discretized according to the chosen grid and the single shooting method is applied to each individual segment of the grid, which means that the trajectory is integrated for each segment according to the given dynamics. Therefore, the dynamics is also discretized along the grid. For each segment i, the defect d between the final state of the integrated trajectory $\tilde{x}_{i+1}$ and the final state of the desired trajectory x'_{i+1}, where the latter is the initial state for the integration within the following segment, is minimized in order to achieve continuity of the solution trajectory at the discretization nodes. Therefore, the overall defect D

$$D = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{pmatrix} = \begin{pmatrix} \tilde{x}_1 - x'_1) \\ \tilde{x}_2 - x'_2 \\ \vdots \\ \tilde{x}_N - x'_N \end{pmatrix} = 0 \tag{2.18}$$

is to be minimized [5].
Compared to the single shooting method, the computation time is higher since the dimension of the NLP is increased, but this results in an increased accuracy for the solution.

Collocation Method

In collocation methods, both the state and control variables are discretized, and the discretized dynamics is introduced as equality constraints. Therefore, it is not necessary to actually integrate the trajectory in each segment. The discretization of both control and state results in a large, but sparse NLP [5].

2.2.3 Convex Optimization Problems

Solution algorithms for NLP do not generally provide knowledge about the convergence towards an optimal solution. However, for convex optimization problems, which are a subclass of NLP, solution algorithms provide convergence towards the global optimum within polynomial time [7]. This makes convex optimization attractive for on-board optimization applications.

An OCP is called a convex if the objective function J is a convex function of some variable x which is defined over a convex set $\chi \subset \mathbb{R}^{n_x}$, such that $x \in \chi$. Note that linear functions and sets are also included in convexity. As a subclass of convex optimization, conic optimization problems and SOCP aim to optimize a convex objective function subject to conic constraints. In the following, several basic definitions are presented to fully explain the structure of SOCP, which are solved by ECOS, as well as the connection between several equivalent formulations. This mathematical preparation is a selection of the basic definitions from the lectures of Steven Boyd on convex optimization [7].

Conic Programming and Second Order Cone Problems

A first important step towards a mathematical framework for SOCP formulations is to understand the definition of a convex set and a cone, which is based on several preceding mathematical definitions.

Definition 1 *A **line** passing through two points $x_1 \neq x_2 | x_1, x_2 \in \mathbb{R}^{n_x}$ is given by the points $y \in \mathbb{R}^{n_x}$ with*

$$y = \theta x_1 + (1-\theta)x_2,\ \theta \in \mathbb{R}. \tag{2.19}$$

*The corresponding **closed line segment** are the points defined by $\theta \in [0,1]$.*

In close relation to lines, an affine set can be defined as follows:

Definition 2 *A set $\mathcal{A} \subseteq \mathbb{R}^n$ is an **affine set** if the line through any two distinct points $x_1, x_2 \in \mathcal{A}$ lies in $\mathcal{A}$, i. e. if for $\theta \in \mathbb{R}$*

$$\theta x_1 + (1-\theta)x_2 \in \mathcal{A}. \tag{2.20}$$

This definition can be generalized by introducing **affine combinations** of points $x_1, ..., x_k$, for which

$$\theta_1 x_1 + ... + \theta_k x_k \text{ with } \theta_1 + ... + \theta_k = 1 \tag{2.21}$$

An affine set then contains every affine combination of its points, meaning that if $\mathcal{A}$ is an affine set, $x_1, ...x_k \in \mathcal{A}$ and $\theta_1 + ... + \theta_k = 1$, then the point $\theta_1 x_1 + ... + \theta_k x_k$ also lies in $\mathcal{A}$. In addition, affine sets are always convex according to the following definition:

Definition 3 *A set $\mathcal{C} \in \mathbb{R}^n$ is a **convex set**, if the line segment between any two points $x_1, x_2 \in \mathcal{C}$ lies completely in $\mathcal{C}$, i.e. if for $\theta \in [0,1]$*

$$\theta x_1 + (1-\theta)x_2 \in \mathcal{C}. \tag{2.22}$$

A simplified geometric interpretation is that a set is convex if any point of the set can be connected to another point of the set by a straight line without leaving the set, such that no point of the connection line is outside of the set. Affine sets are always convex, since they always contain the entire line between any two distinct points in the set.
An illustrative example for this geometric interpretation of convex sets is shown in figure 2.2, which will play an important role in the convexification of the powered descent landing problem presented later on. The upper part of the image shows an annulus in grey, which is a non-convex set since connecting two points of the annulus with a straight line may result in a line that partially leaves the annulus and enters the central excluded circle. Therefore, not all connection lines completely lie within the set defined by the annulus. The lower part of the image shows a convex set where any points can be connected without leaving the set.

In analogy to affine combinations, one can define convex combinations as a point of the form

$$\theta_1 x_1 + ... + \theta_k x_k \text{ with } \theta_1 + ... + \theta_k = 1 \text{ and } \theta_i \geq 0, i = 1, ..., k. \tag{2.23}$$

Since all affine sets are convex, affine transformations and especially linear transformations preserve convexity:

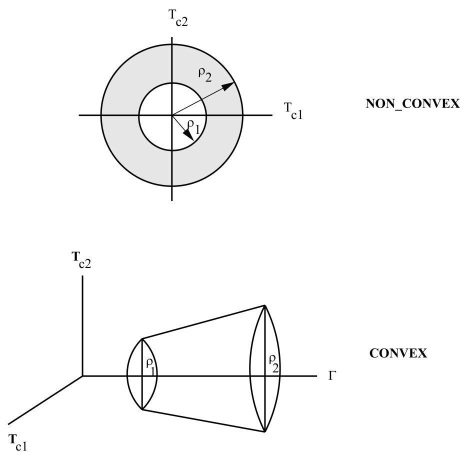

FIGURE 2.2: Example of non-convex (grey annulus in the top image) and convex sets (bottom) [1]

Lemma 1 *Suppose that the set $\mathcal{C} \subseteq \mathbb{R}^n$ is convex and $f : \mathbb{R}^n \to \mathbb{R}^m$ an affine function of form $f = Ax + b$ with $A \in \mathbb{R}^m \times \mathbb{R}^n$ and $b \in \mathbb{R}^m$. The image*

$$f(\mathcal{C}) = \{f(x) | x \in \mathcal{C}\} \tag{2.24}$$

is then convex.

These basic definitions on convex sets are the fundamental base for defining cones and second order cones.

Definition 4 *A set $\mathcal{C}$ is a **cone** if and only if for any $x \in \mathcal{C}$ and $\theta \geq 0$ the point $y = \theta x$ lies in $\mathcal{C}$*

which can be extended to the definition of a convex cone:

Definition 5 *A set $\mathcal{C}$ is a **convex cone** if for any $x_1, x_2 \in \mathcal{C}$ and $\theta_1, \theta_2 \geq 0$, the point*

$$\theta_1 x_1 + \theta_2 x_2 \tag{2.25}$$

lies in $\mathcal{C}$. Thus a set is a convex cone if it is convex and a cone.

Typical and very important examples of convex cones related to this thesis are the norm cone and the second order cone. The norm cone associated with any norm $\|\cdot\|$ on $\mathbb{R}^n$ is given by the set

$$C = \{(x, t) \subseteq \mathbb{R}^{n+1} | \|x\| \leq t\}, \tag{2.26}$$

while the norm cone corresponding to the Eucledian norm $\|\cdot\|_2$ is called **second order cone** with

$$C = \{(x, t) \in \mathbb{R}^{n+1} | \|x\|_2 \leq t\} \tag{2.27}$$

An illustration of this cone, also refered to as Lorentz cone, is given in figure 2.3 . There exist different equivalent formulations of second order cones, of which the most important one for this thesis is

$$\|ax + b\| \leq C^{\mathrm{T}} x + d, \tag{2.28}$$

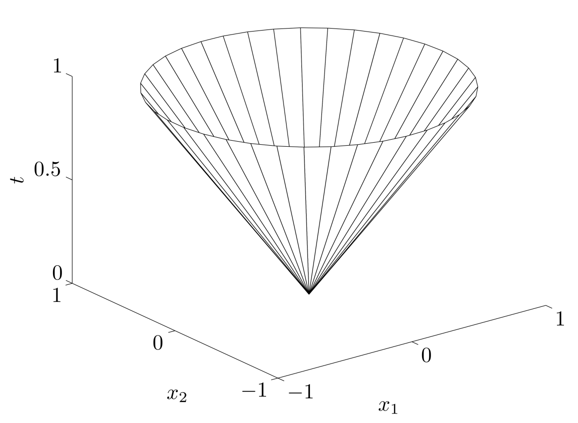

FIGURE 2.3: Example of a second order cone (Lorentzcone) [7]

with $a \in \mathbb{R}^{k \times n}$, b and $d \in \mathbb{R}^k$ and $C \in \mathbb{R}^{n \times k}$. This formulation of a second order cone is a typical way of formulating second order cone constraints in an optimal control problem, and it is equivalent to the general formulation given in equation 2.27, since both sides of equation 2.28 are an affine image of x to variables $u = ax + b$, $u \in \mathbb{R}^k$ and $t = C^{\mathrm{T}}x + d$, $t \in \mathbb{R}$ building up the second order cone $\mathcal{K} = \{(u,t) \in \mathbb{R}^{k+1} \mid \|u\| \leq t\}$. Thus, equation 2.28 implies that the vector $(ax + b, C^{\mathrm{T}}x + d)$ lies within the second order cone set $\mathcal{K}$.
To finally obtain a formulation of second order cone problems, a special type of inequalities has to be introduced, called general inequalities. These inequalities originate from so called proper cones:

Definition 6 *A cone $\mathcal{K} \subseteq \mathbb{R}^n$ is called a **proper cone** if it satisfies the following properties:*

- *$\mathcal{K}$ is convex*
- *$\mathcal{K}$ is closed*
- *$\mathcal{K}$ is solid, meaning that it has non-empty interior*
- *$\mathcal{K}$ is pointed, meaning that if $x \in \mathcal{K}$ and $-x \in \mathcal{K}$, then $x = 0$.*

For further explanations of the above properties the reader is refered to [7]. General inequalities originate from such proper cones as follows:

Definition 7 *Let $\mathcal{K}$ be a proper cone. There is a partial ordering on $\mathbb{R}^n$, called **general inequality**, associated to $\mathcal{K}$ with*

$$x \preceq_{\mathcal{K}} y \Leftrightarrow y - x \in \mathcal{K} \tag{2.29}$$

One example for such a proper cone with a generalized inequality is the non-negative orthant $\mathcal{K} = \mathbb{R}^n_+$. In that case, the general inequality $\preceq_K$ is equivalent to the componentwise ineqality of two vectors, meaning that for $x, y \in \mathbb{R}^n_+$ the general inequality $x \preceq y$ is equivalent to $x_i \leq y_i, i = 1, ..., n$. Another important example is the positive semidefinite cone $\mathcal{K} = \mathcal{S}^n_+$, which is also a proper cone. The corresponding generalized inequality $X \preceq_K Y$ implies that the matrix $Y - X$ is positive semidefinite. In the following, the subscript $\mathcal{K}$ for the generalized inequality is ommitted, since a generalized inequality will always be defined over some conic set.
It will be useful to derive an equivalent version of the second order cone formulation in equaton 2.27 making use of generalized inequalities by applying the definition given in equation 2.29. A second order cone given as $\|ay + b\| \leq C^T y + d$ implies that the vector

$\big(ay + b, C^{\mathrm{T}}y + d\big)$ lies in the cone $\mathcal{K}$. Therefore, one can use definition 7 to set up the following chain:

$$\begin{aligned}
& \big(az + b, C^{\mathrm{T}}y + d\big) \in \mathcal{K} && (2.30)\\
\Leftrightarrow\quad & \big(ay + b, C^{\mathrm{T}}y + d\big) - 0 \in \mathcal{K} && (2.31)\\
\Leftrightarrow\quad & \big(ay + b, C^{\mathrm{T}}y + d\big) \succeq 0 && (2.32)\\
\Leftrightarrow\quad & \big(-ay - b, -C^{\mathrm{T}}y - d\big) \preceq 0. && (2.33)
\end{aligned}$$

A reformulation using matrix notation results in:

$$\begin{aligned}
& \big(-ay - b, -C^{\mathrm{T}}y - d\big) \preceq 0 && (2.34)\\
\Leftrightarrow\quad & \begin{pmatrix} -a \\ -C^T \end{pmatrix} y - \begin{pmatrix} b \\ d \end{pmatrix} \preceq 0 && (2.35)\\
\Leftrightarrow\quad & \begin{pmatrix} -a \\ -C^{\mathrm{T}} \end{pmatrix} y \preceq \begin{pmatrix} b \\ d \end{pmatrix}. && (2.36)
\end{aligned}$$

By defining the matrix $G = \begin{pmatrix} -a \\ -C^{\mathrm{T}} \end{pmatrix}$ and vector $h = \begin{pmatrix} b \\ d \end{pmatrix}$ one obtains the reformulated second order cone as

$$Gy \preceq_K h \tag{2.37}$$

The transformation chain has been presented here since it will be of extraordinary importance for the transcription method described in this thesis. Finally, the SOCP can be written as follows [7]:

Problem 3

$$\begin{aligned}
\min\ & c^{\mathrm{T}}y && (2.38)\\
\text{s.t. } & Ay = B && (2.39)\\
& Gy + s \preceq h && (2.40)
\end{aligned}$$

In this problem, the cost function is a linear function of the optimization variable y, constrained by the equalities $Ay = B$ and the second order cone constraint $Gy \preceq h$. Note that the matrix A and c^T appearing in this problem are different from the matrices a and C^T used in the second order cone equation 2.28. The variable s appearing is introduced as possible slack variables, that might be added to the problem and especially in the formulation of the inequality constraints. Throughout this work, we will not make use of such numerical slack variables and they should not be confused with slack variables introduced later on in the general formulation of the powered descent landing problem. In the following, this formulation of SOCP will be referred to as ECOS-feasible formulation [9], since ECOS has been created to solve optimization problems with the structure given in problem 3.

Linear inequality constraints

Several constraints, appearing especially in the powered descent landing problem, are inequalities which are linear with respect to components of the optimization variable y, and not second order cone constraints. Introducing these constraints in simulations with ECOS is straightforward. If the constraint is linear, the generalized inequality in problem 3 becomes the usual inequality on $\mathbb{R}$, such that $G_l y = h_l$, where the index l denotes that only a one-line element of the matrices G, and h is considered for the linear inequality.

Quadratic Constraints in SOCP

If the optimization problem contains quadratic constraints that can be written in the form of

$$y^T H y + p^T y + q = y^T F^T F y + p^T y + q \leq 0 \tag{2.41}$$

with $y \in \mathbb{R}^n$,$H \in \mathbb{R}^n \times \mathbb{R}^n$, F and $F^T \in \mathbb{R}^n \times \mathbb{R}^n$, $p^T \in 1 \times \mathbb{R}^n$ and $q \in \mathbb{R}$, then this constraint can be rewritten as an equivalent second order cone constraint with [7]:

$$\left\| \begin{matrix} \frac{1+p^T y+q}{2} \\ Fy \end{matrix} \right\|_2 \leq \frac{1 - p^T y - q}{2} \tag{2.42}$$

The connection between the quadratic constraint and the formulation as a second order cone constraint requires that the matrix H can be split such that $H = F^T F$. This decomposition is also called Cholesky decomposition and is only possible for positive semidefinite matrices. Hence, H has to be positive semidefinite if the quadratic constraint should be formulated as a second order cone problem. Note that the decomposed matrix F typically is a lower or upper triangular matrix. A short proof for the connection between equation 2.41 and 2.42 is given in the appendix A.1. The formulation of the quadratic constraint as a second order cone constraint is even more clear and applicable for ECOS if it is rewritten in the form of equation 2.28:

$$\begin{aligned} \left\| \begin{matrix} \frac{1+b^T x+c}{2} \\ Fx \end{matrix} \right\|_2 &\leq \frac{1 - b^T x - c}{2} \\ \left\| \begin{matrix} \frac{1}{2} b^T x + \frac{1+c}{2} \\ Fx + 0 \end{matrix} \right\|_2 &\leq -\frac{1}{2} b^T x + \frac{1-c}{2} \\ \left\| \begin{pmatrix} \frac{1}{2} b^T \\ F \end{pmatrix} x + \begin{pmatrix} \frac{1+c}{2} \\ 0 \end{pmatrix} \right\|_2 &\leq -\frac{1}{2} b^T x + \frac{1-c}{2} \end{aligned} \tag{2.43}$$

Using

$$a = \begin{pmatrix} \frac{1}{2} p^T \\ F \end{pmatrix} \qquad b = \begin{pmatrix} \frac{1+q}{2} \\ 0 \end{pmatrix} \qquad C^T = -\frac{1}{2} p^T \qquad d = \frac{1-q}{2}, \tag{2.44}$$

results in the desired SOCP form of equation 2.28. Therefore this method describes a strategy to formulate quadratic constraints in an ECOS-feasible way.

2.2.4 Interior Point Methods and SOCP with ECOS

ECOS solves conic optimization problems in the general form of problem 3. To do so, it applies so called interior point methods, which are a powerful tool in solving constrained optimization problems. A complete theory of interior point methods and especially numerical issues related to that method is beyond the scope of this project work, since ECOS is mainly used as a black box assuming that it offers enough information to determine whether a solution provided by ECOS is optimal or not. However, a brief overview of interior point methods and the usage of ECOS is provided in the following.
Interior point methods are a widely used approach in order to solve different types of optimization problems. As such, ECOS uses so called primal-dual interior point methods for solving conic problems [9]. The general concept is to replace the constrainted optimization problem 3 by a series of smooth convex unconstrained problems, which can be solved, stepwise e.g. by Newton's method. In order to transform the contrained problem into one which is unconsrainted, a so called barrier function is employed in the cost function. The approach

results in a sequence of solutions

$$\chi^{k+1} = \chi^{k+1} + \alpha^k \Delta\chi^k, k = 0, 1, 2 \ldots \tag{2.45}$$

where $\chi^k \equiv (x^k, y^k, s^k, z^k)$ [9]. the variables x and y are the so called dual variables resulting from the dual problem formulation of problem 3 (see [9] and [7]) and s are possible slack variables from problem 3. The step length α^k is found via line search algorithms and $\Delta\chi^k$ is a search direction found by solving linear systems of equations. It can be shown that the solution χ^k at the iteration steps k follow the so called central path which ends in the final solution set.

3 Powered Descent Landing as a SOCP

This work is based on the so called lossless convexification method for the powered descent landing problem presented by Açikmese and Ploen [1]. In this framework, the OCP describing the powered descent landing of a spacecraft, for example on Mars, is reformulated as a SOCP. The important steps towards a convex formulation of the fuel-optimal landing problem presented in [1] are repeated in the following in order to understand the underlying optimization problem as a whole .

3.1 Problem Formulation

The general minimum-fuel OCP describing a spacecraft landing on a planet in the given time interval $t \in [0, t_f]$ aims to find the time of flight t_f and thrust profile $T(t)$ for which the consumed fuel is minimized or equivalently for which the total mass of the spacecraft at the end of the landing is maximized. It can be formulated as in problem 4 [1].

Problem 4

$$\max_{T_c(t),t_f} \quad m(t_f) = \min_{T(t),t_f} \int_0^{t_f} \|T_c(t)\| \, \mathrm{d}t \tag{3.1}$$

$$\text{s.t.} \quad \ddot{r} = g + \frac{T_c(t)}{m(t)} \qquad \dot{m}(t) = -\alpha \, \|T_c(t)\| \tag{3.2}$$

$$0 < \rho_1 \leq \|T_c(t)\| \leq \rho_2 \qquad r_1(t) \geq 0 \tag{3.3}$$

$$\|a_j x(t) + b_j\| \leq C_j^T + d_j \tag{3.4}$$

$$m(0) = m_{wet} \quad r(0) = r_0 \qquad \dot{r}(0) = \dot{r}_0 \quad r(t_f) = \dot{r}(t_f) = 0 \tag{3.5}$$

The position of the spacecraft at a certain time t is defined by the position vector $r(t)$ within the surface fixed coordinate system illustrated in figure 3.1. The corresponding velocities and accelerations are denoted as $\dot{r}$ and $\ddot{r}$. Together with the mass $m(t)$ of the spacecraft at a

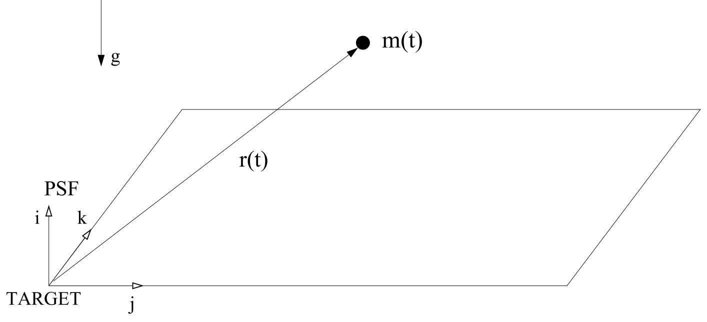

FIGURE 3.1: Surface fixed coodinate frame [1]

given time the aforementioned variables build up the actual state vector $x \in \mathbb{R}^7$ of the OCP with

$$x^T(t) = \begin{bmatrix} r_1(t) & r_2(t) & r_3(t) & \dot{r}_1(t) & \dot{r}_2(t) & \dot{r}_3(t) & m(t) \end{bmatrix}. \tag{3.6}$$

The control variable $u(t) \in \mathbb{R}^3$ of this problem is the net thrust vector $u(t) = \begin{bmatrix} T_{c,1}(t) & T_{c,2}(t) & T_{c,3}(t) \end{bmatrix}$. The net thrust vector is defined via the thrust level T of the n identical thrusters as

$$T_c = (nT\cos(\phi))\, e, \tag{3.7}$$

where e is the corresponding direction of the net thrust as sketched in figure 3.2, and ϕ the cant angle. It is therefore assumed that the n thrusters are arranged symmetrically around the spacecraft with the same cant angle. The dynamics prodiving changes of the state vector

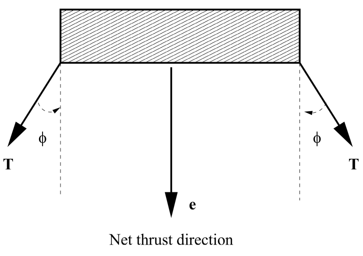

FIGURE 3.2: Geometry of thrusters [1]

x are mainly described through the acceleration of the spacecraft. This acceleration can be written as the sum of the gravitational acceleration g and the acceleration resulting from the net thrust, where the net thrust has to be divided by the spacecraft's mass in order to obtain the corresponding acceleration:

$$\ddot{r} = g + \frac{T_c(t)}{m(t)}. \tag{3.8}$$

The mass loss or fuel consumption related to the thrust is also implemented in the dynamics via

$$\dot{m}(t) = -\alpha \left\| T_c(t) \right\|, \tag{3.9}$$

where α is the fuel consumption rate that is calculated from the specific impulse I_{sp} and the gravitational acceleration on Earth g_e via

$$\alpha = \frac{1}{I_{sp} g_e \cos(\phi)}. \tag{3.10}$$

The additional constraints introduced in the OCP imply that each of the thrusters has an upper and lower thrust boundary, and thus the thrust level $T(t)$ is bounded by the lower and upper boundary ρ_1 and ρ_2

$$0 < T_1 \le T(t) \le T_2.$$

The net thrust is therefore also bounded by the constants $\rho_1 = nT_1\cos(\phi)$ and $\rho_2 = nT_2\cos(\phi)$ resulting in the actual control constraint [1]

$$\rho_1 \le \left\| T_c(t) \right\| \le \rho_2. \tag{3.11}$$

Another constraint in this problem is that the spacecraft is supposed to stay above the surface throughout the whole flight time, and therefore the constraint $r_1 \ge 0$ is introduced. The residual constant parameters in the OCP are the initial and final condition for the state with m_{wet} being the spacecraft's mass at the beginning of the descent including the entire fuel, as well as the initial and final values for the position and velocity with r_0 and $\dot{r}_0$.

The powered descent landing is thus completely formulated as an OCP, that aims to minimize the used fuel or equivalently maximize the total mass of the spacecraft. However, it contains non-convex elements and thus cannot be solved by means of convex optimization. As will be explained in the following section, the non-convex elements are the dynamics, i.e. the formulation of the acceleration, and the control constraint on the thrust vector. These non-convex elements are eliminated step-by step resulting in several adapted OCPs, which finally lead to the SOCP formulation for the powered descent landing.

3.1.1 Problem Formulation 2 - Convexification of Thrust Magnitude

A first step in the convexification of the OCP 4 is to convexify the constraint on the net thrust vector T_c. Figure 2.2 illustrates this constraint with an example 2-D control space in the upper part of the image. Bounding the thrust vector by a lower and an upper boundary results in an annulus for the 2-D case, which is a non-convex set according to the definition 3. In order to convexify the control constraint, a slack variable $\Gamma(t)$ can be introduced to relax this constraint via two new constraints: $\|T_c(t)\| \leq \Gamma(t)$ and $\rho_1 \leq \Gamma(t) \leq \rho_2$. Because of this relaxation the thrust vector is constrained to lie within a cone as illustrated in the lower part of figure 2.2. Implementing these two new constraints and replacing the net thrust in the cost function by the slack variable $\Gamma(t)$ results in the following relaxed OCP [1]:

Problem 5

$$\max_{\Gamma(t),T_c(t),t_f} \quad m(t_f) = \min_{\Gamma(t),T(t),t_f} \int_0^{t_f} \|T_c(t)\|\,\mathrm{d}t \tag{3.12}$$

$$\text{s.t.} \quad \ddot{r} = g + \frac{T_c(t)}{m(t)} \qquad \dot{m}(t) = -\alpha\,\|T_c(t)\| \tag{3.13}$$

$$\|T_c(t)\| \leq \Gamma(t) \tag{3.14}$$

$$0 < \rho_1 \leq \|\Gamma(t)\| \leq \rho_2 \qquad r_1(t) \geq 0 \tag{3.15}$$

$$\|a_j x(t) + b_j\| \leq C_j^T + d_j \tag{3.16}$$

$$m(0) = m_{wet} \quad r(0) = r_0 \qquad \dot{r}(0) = \dot{r}_0 \quad r(t_f) = \dot{r}(t_f) = 0 \tag{3.17}$$

If the optimal solution for the problem 5 is given with $(t_f^*, T_c^*(\cdot), \Gamma^*(\cdot))$, then $(t_f^*, T_c^*(\cdot))$ is also an optimal solution for the original problem 4 as shown in [1]. Another outcome of the related proof is that the optimal solution yields $\|T_c^*(t)\| = \rho_1$ or $\|T_c^*(t)\| = \rho_2$ for all $t \in \left[t, t_f^*\right]$. This implies that for the optimal set $(t_f^*, T_c^*(\cdot), \Gamma^*(\cdot))$ the thrust is expected to be at maximum or minimum level throughout the whole process, also refered to as the "Bang-Bang" solution in optimal control. However, this only holds if the time of flight t_f is an optimization parameter and therefore optimized as well. If t_f is fixed and the problem only aims at finding the thrust profile $T(t)$ that minimizes the used fuel for the given flight time, the optimal solution does not necessarily have such a Bang-Bang structure.
Even though a relaxation and convexification of the thrust-magnitude constraint has been introduced in problem 5, one cannot use means of convex optimization for solving problem 5, since the dynamics is still non-convex. In the calculation of the acceleration $\dot{r}$, the net thrust is divided by the mass, and hence a division by one of the state elements occurs which means a high non-convex relation. To overcome this and finally reach a completely convex formulation of problem 5, variable substitutions have to be introduced as described in the following.

3.1.2 Problem Formulation 3 - Dynamics Linearization

As the second main step in obtaining an entirely convex formulation of the OCP for the powered descent landing, one can eliminate the non-linearities in the dynamics by a sequence of variable substitions. The first set of substitutions are [1]

$$\sigma(t) \equiv \frac{\Gamma(t)}{m(t)} \qquad \text{and} \qquad u(t) \equiv \frac{T_c(t)}{m(t)}, \tag{3.18}$$

introducing the new control variables $\sigma \in \mathbb{R}$ and $u \in \mathbb{R}^3$. Following this substitution, the dynamics $\ddot{r}(t)$ and $\dot{m}(t)$ are transformed into

$$\ddot{r}(t) \equiv u(t) + g, \qquad \text{and} \qquad \frac{\dot{m}(t)}{m(t)} = -\alpha\sigma. \tag{3.19}$$

The resulting dynamics for the mass is a differential equation with the solution

$$m(t) = m_0 \exp\left[-\alpha \int_0^{t_f} \sigma(\tau)\mathrm{d}\tau\right] \tag{3.20}$$

Since the aim of the optimization problem is to maximize the spacecraft's mass at the end of the considered time interval, or equivalently to minimize the total fuel consumption, one can replace the previous cost function given in 3.20 by

$$\min \int_0^{t_f} \sigma(\tau)\mathrm{d}\tau. \tag{3.21}$$

Minimizing this integral maximizes $m(t)$ from equation 3.20, since for $\alpha > 0$ the function 3.20 is a monotonically decreasing function of the integral in 3.21.
Also the constraints have to be rewritten with the new control variables according to the substitution as

$$\|T_c(t)\| \leq \Gamma(t) \Leftrightarrow \|u(t)\| \leq \sigma(t) \tag{3.22}$$
$$\rho_1 \leq \Gamma(t) \leq \rho_2 \Leftrightarrow \frac{\rho_1}{m(t)} \leq \sigma(t) \leq \frac{\rho_2}{m(t)}. \tag{3.23}$$

The second constraint has become bilinear, since $m(t)$ and $\sigma(t)$ are both variables, and multiplying both sides of the new constraint 3.23 with $m(t)$ results in the product $\sigma(t) \cdot m(t)$ to appear within the constraint. In addition, once again a value is divided by the mass $m(t)$, and hence divided by a component of the state variable. Therefore, the applied substitution linearizes the dynamics, but results in non-convex constraints.
Another substitution is required to eliminate the bilinearity and to finally convexify the entire problem. The useful substitution [1]

$$z(t) \equiv \ln\left(m\left(t\right)\right) \tag{3.24}$$

transforms the dynamics of the mass consumption into

$$\frac{\dot{m}(t)}{m(t)} = -\alpha\sigma \Leftrightarrow \dot{z} = -\alpha\sigma(t). \tag{3.25}$$

The new slack varible $\sigma(t)$ therefore can be understood as a mass flow. Also the inequality constraint is changed to

$$\frac{\rho_1}{m(t)} \leq \sigma(t) \leq \frac{\rho_2}{m(t)} \Leftrightarrow \rho_1 e^{-z(t)} \leq \sigma(t) \leq \rho_2 e^{-z(t)}. \tag{3.26}$$

The left part of the new inequality constraint on $\sigma(t)$ is a convex feasible region, while the right one is not. Figures 3.3 and 3.4 illustrate this geometrically. Independent of the explicit structure of $z(t)$, the inequality $\rho_1 e^{-z(t)} \leq \sigma(t)$ describes a convex feasible set for $\sigma(t)$, which is marked in blue in the left plot. Rememebering the geometrically interpretation of convex sets in 2.2.3, one can connect any point of the blue region with any other point within the blue set by a direct line. This is not possible for the blue region in the right figure, showing the non-convex set for $\sigma(t)$ defined by the inequality $\sigma(t) \leq \rho_2 e^{-z(t)}$. Furthermore, even

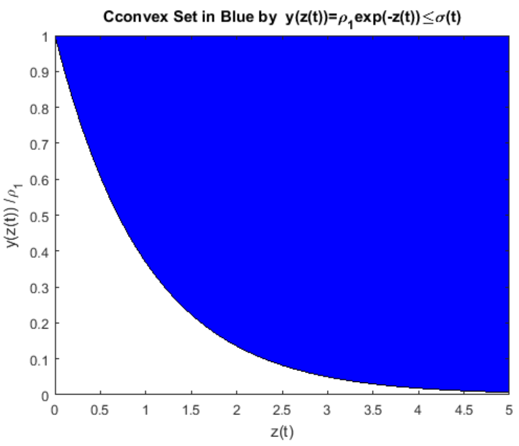

FIGURE 3.3: Convex region defined by left constraint

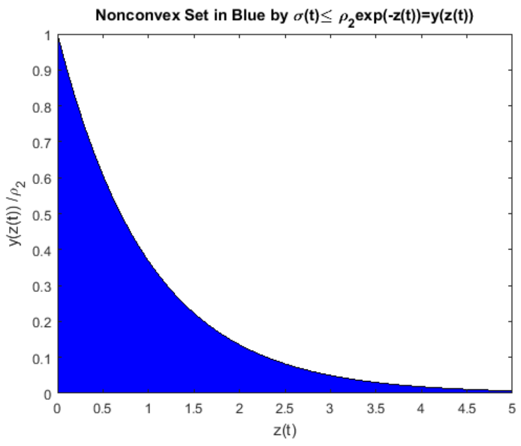

FIGURE 3.4: Nonconvex region defined by left constraint

for the convex part in equation 3.26 the substituion given by equation 3.24 introduces an exponential function with respect to the variable $z(t)$, which is not applicable for ECOS using second order cone or linear constraints. To overcome this, one can use the Taylor series expansion of e^{-z} to write the constraint in an appropriate way [1]. The left part of equation 3.26 can then be written as

$$\rho_1 e^{-z_0} \left[1 - (z - z_0) + \frac{(z - z_0)^2}{2} \right] \leq \sigma, \tag{3.27}$$

using the first three terms of the Taylor expansion and $z_0 = z(t = 0)$. This is a quadratic inequality constraint approximating the exponential inequality constraint, and can potentially be written as an SOCP as explained in section 2.2.3. The right part of the inequality constraint defining the non-convex inequality can be linearized, and thus convexified, by using only the first two terms of the Taylor expansion:

$$\sigma \leq \rho_2 e^{-z_0} \left[1 - (z - z_0) \right]. \tag{3.28}$$

If one defines

$$\mu_1 \equiv \rho_1 e^{-z_0}, \qquad \mu_1 \equiv \rho_2 e^{-z_0} \tag{3.29}$$

and

$$z_0(t) = \ln(m_{wet} - \alpha \rho_2 t), \tag{3.30}$$

such that $z_0(t)$ is a lower bound on $z(t)$ at time t, the inequality constraint $\rho_1 \leq \Gamma(t) \leq \rho_2$ of problem 5 is finally converted into

$$\rho_1 e^{-z_0}\left[1-(z-z_0)+\frac{(z-z_0)^2}{2}\right] \leq \sigma \leq \rho_2 e^{-z_0}\left[1-(z-z_0)\right]. \tag{3.31}$$

In [1] an estimation of the errors introduced by the Taylor series expansion is also given. It is shown that, even for the linear approximation, the errors are below two percent for an example flight time of 70 seconds.
In addition one has to consider so far unmentioned constraints as lower and upper boundaries for the variable $z(t)$, which represents the mass of the spacecraft. The lower boundary is defined by the possibility that the thrusters run with maximum thrust in the interval $[t_0, t]$, and therefore have a maximum of mass consumption. The upper boundary is the opposite case of minimum thrust in the given time interval, such that the variable $z(t)$, which has been introduced to substitute the actual mass $m(t)$, is constrained with

$$\ln\left(m_{wet}-\alpha\rho_2 t\right) \leq z(t) \leq \ln\left(m_{wet}-\alpha\rho_1 t\right). \tag{3.32}$$

Moreover, we introduce a constraint on the pointing of the thrust vector at the final landing time t_f, used in all considered landing problems in the following. This constraint sets the thrust to point straight downwards at the very end of the maneuver, which can be seen as a stabilization of the vehicle. Therefore, the components $u_2(t_f)$ and $u_3(t_f)$ of the control variables have to vanish, while the component pointing towards the surface has to be $u_3(t_f) \leq 0$. In total, one arrives at the entirely convexified version of problem 5 with quadratic or linear constraints as given in problem 6 [1]. One might be interested in adding further constraints on the state or control variables, which is taken into account by the general formulation $\|a_j x(t) + b_j\| \leq C_j^T + d_j$ of second order cone constraints, where the index j denotes the different constraints.

Problem 6

$$\min_{u,\sigma,t_f} \quad \int_0^{t_f} \sigma(t)\mathrm{d}t \tag{3.33}$$

$$\text{s.t.} \quad \ddot{r} = u(t) + g, \quad \dot{z}(t) = -\alpha\sigma(t) \tag{3.34}$$

$$\|u(t)\| \leq \sigma(t) \tag{3.35}$$

$$\mu_1\left[1-(z-z_0)+\frac{(z-z_0)^2}{2}\right] \leq \sigma \leq \mu_2\left[1-(z-z_0)\right] \tag{3.36}$$

$$\ln\left(m_{wet}-\alpha\rho_2 t\right) \leq z(t) \leq \ln\left(m_{wet}-\alpha\rho_1 t\right) \tag{3.37}$$

$$\|a_j x(t) + b_j\| \leq C_j^T + d_j \tag{3.38}$$

$$z(0) = ln(m_{wet}) \quad r(0) = r_0 \quad \dot{r}(0) = \dot{r}_0 \quad r(t_f) = \dot{r}(t_f) = 0 \tag{3.39}$$

$$u_2(t_f) = u_3(t_f) = 0 \quad u_1(t_f) <= 0 \tag{3.40}$$

Having arrived at the final SOCP formulation of the powered descent landing presented by [1], one has to find a way to formulate this into an ECOS-feasible way, i.e. one has to find an applicable discretization and transcription. Before these first contributions of this thesis are presented, some example second order cone constraints are considered that can be introduced into problem 6 via the general formulation of second order cones.

3.1.3 Example Constraints

The most important examples of these constraints are related to the pointing direction of the thrust vector and the glide slope of the spacecraft.

Thrust Pointing Constraint

It might be necessary that the thrust of the spacecraft has to point towards a certain direction during the descent phase and not only at the very end. Also the thrust pointing is almost always limited by the construction and the thruster arrangement itself. To take into account these limitations, one can introduce a thrust pointing constraint into problem 6. The underlying considerations have been performed in [2] as an expansion of the convexified powered descent landing problem from [1]. In general, the thrust pointing constraint results from basic trigonometric considerations to be [2]

$$\frac{\hat{\boldsymbol{n}} \cdot \boldsymbol{T_c}}{\|\boldsymbol{T_c(t)}\|} \geq \cos\left(\theta_{max}\right), \tag{3.41}$$

where the vector $\hat{n}$ is a unit vector of the reference direction, $T_c(t)$ the net thrust vector and θ_{max} the maximum allowed angular separation between the net thrust vector and the reference direction. A visualization of this constraint is given in figure 3.5 in a planar representation. In the left part of the image, the pure bounds on the thrust magnitude are presented without any restriction of the pointing direction. In the right image, the pointing direction is limited within the blue area. The same holds for the middle part of the image, but it illustrates that the thrust pointing constraint is only convex for $\theta_{max} \in [0, \pi]$. Considering

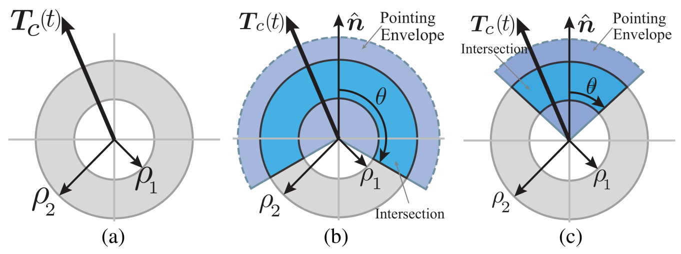

FIGURE 3.5: Graphical illustration of control constraint in planar view [2].

the relaxation of the thrust magnitude via a slack variable as shown in problem 5 and the variable substitutions introduced in 3.18, equation 3.41 can be rewritten as

$$\frac{\hat{n} \cdot u(t)}{\sigma(t)} \leq cos\left(\theta_{max}\right) \tag{3.42}$$

$$\Leftrightarrow 0 \leq= u_1(t) - \cos\left(\theta_{max}\right)\sigma(t) \tag{3.43}$$

Note, that it has been assumed that the thrust points downwards within the defined reference frame and thus $\hat{n} \cdot u(t) = u_1$.

Glide slope constraint

Another typical constraint on the trajectory of the powered descent landing phase is the so called glide slope constraint. The glide slope angle γ shown in figure 3.6 can be defined mathematically by the angle between the projection of the spacecraft's position on the surface and the straight line between the spacecraft's position $r(t) = \begin{bmatrix} r_1(t) & r_2(t) & r_3(t) \end{bmatrix}$ and final position as in equation 3.44 [1].

$$\gamma(t) = \arctan\left(\frac{r_1(t) - r_1(t_f)}{\sqrt{(r_2(t) - r_2(t_f))^2 + (r_3(t) - r_3(t_f))^2}}\right) \tag{3.44}$$

A constraint on this glide slope is that it should not be lower than a certain threshold γ_{max}. Therefore, it can be expressed as a second ordercone constraint through the following for-

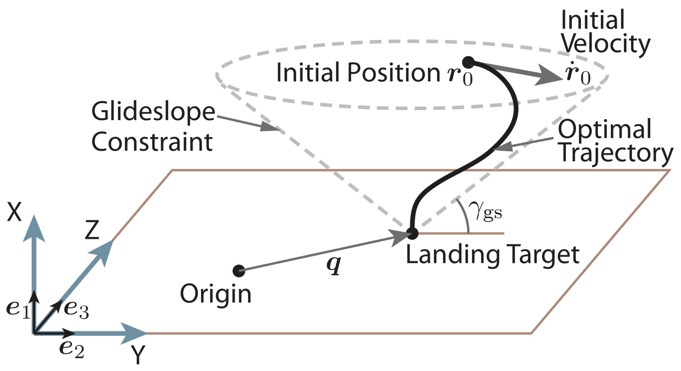

FIGURE 3.6: Definition of glide slope angle constraint on γ [6]

mulations:

$$\gamma(t) \geq \gamma_{max} \tag{3.45}$$

$$\Leftrightarrow \arctan\left(\frac{r_1(t) - r_1(t_f)}{\sqrt{(r_2(t) - r_2(t_f))^2 + \left(r_3^2(t) - r_3(t_f)\right)^2}}\right) \geq \gamma_{max} \tag{3.46}$$

$$\Leftrightarrow \sqrt{(r_2(t) - r_2(t_f))^2 + (r_3(t) - r_3(t_f))^2} \leq \frac{r_1(t) - r_1(t_f)}{\tan(\gamma_{max})} \tag{3.47}$$

$$\Leftrightarrow \begin{pmatrix} r_2(t) - r_2(t_f) \\ r_3(t) - r_3(t_f) \end{pmatrix} \leq \frac{r_1(t) - r_1(t_f)}{\tan(\gamma_{max})} \tag{3.48}$$

4 Transcription of the Convexified Powered Descent Landing Problem

The main contribution of this thesis is the development of a suitable discretization and transcription of problem 6, such that it is transformed into the form of problem 3 and can be passed to ECOS in order to obtain the solution. This chapter presents the developed method before an example simulation of Mars pinpoint landing is set and compared to [1].

4.1 Discretization

The discretization of problem 6 requires a decision on the structure of the optimization variable y. According to the description in chapter 2.2.2, we decide on an optimization variable $y(t)$ at each time t consisting of the state vector $x(t)$ and the control variables $u(t)$ and $\sigma(t)$, where the latter combines two roles as a control variable and a slack variable for the thrust magnitude. Thus, the optimization variable at time t is $y(t) \in \mathbb{R}^{11}$, $y^T(t) = \begin{bmatrix} x(t) & u(t) & \sigma(t) \end{bmatrix}$. Second, the discretization of the problem is chosen as a grid of N equally distributed points t_i, $i \in [0, N]$ in the time intervall $t_i \in [t_0 = 0, t_f]$ separated by $\Delta t = \frac{t_f - t_0}{N-1}$. We assume that we will only handle integer flight times t_f, which on the one hand limits the accuracy and on the other preserves the second order cone structure of the convexified powered descent landing OCP. The optimization variable $y(t)$ is therefore discretized into stage variables $y_i = y(t_i)$ and the complete optimization variable is given by

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}, y \in \mathbb{R}^{n_y = N \cdot 11} \tag{4.1}$$

The subscript i at optimization, state or control variables will always refer to the i-th node in the following if not stated otherwise. Also, individual components of vectors will be placed as an index before the index denoting number of nodes. Due to the discretization, the cost function in problem 6 has to be approximated by the sum

$$\min \int_{t_0}^{t_f} \sigma(t) \mathrm{d}t \approx \sum_{i=0}^{N} w_i \sigma_i, \tag{4.2}$$

where w_i are possible weights for the slack variables at node i. Also the dynamic equations need to be discretized according to the chosen scheme and we assume that the dynamics $\dot{x}$ yields a linear expression. For the specific dynamics of problem 6 and taking into account the structure of the state vector defined in equation 3.6, the discretization allows the dynamics to be replaced by the relation

$$\dot{x}_i = A^{dyn} y_i + B_i = \begin{pmatrix} 0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&-\alpha \end{pmatrix} y_i + \begin{pmatrix} 0 \\ 0 \\ 0 \\ \Delta t \cdot g \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{4.3}$$

It is useful to remember at this point that several elements of the considered dynamics depend on the control variables u and σ, like the change of mass represented by $\dot{z}$. Because of that, the dynamic matrix $A^{dyn} \in \mathbb{R}^{n_x \times (n_x + n_u + n_\sigma)}$ also takes into account the control variables at the nodes i and $i+1$, where n_u and n_σ are the corresponding dimensions of the control variable u_i and σ_i. We therefore arrive at a discretized version of the SOCP for the powered descent landing with

Problem 7

$$\min_{u_i,\sigma_i} \sum_{i=0}^{N} w_i \sigma_i \tag{4.4}$$

$$\text{s.t. } \dot{x}_i = A^{dyn} y_i \tag{4.5}$$

$$\|u_i\| \leq \sigma_i \tag{4.6}$$

$$\mu_{1,i}\left[1 - (z_i - z_{0,i}) + \frac{(z_i - z_{0,i})^2}{2}\right] \leq \sigma_i \leq \mu_{2,i}\left[1 - (z(i) - z_{0,i})\right] \tag{4.7}$$

$$\ln\left(m_{wet} - \alpha \rho_2 t_i\right) \leq z(i) \leq \ln\left(m_{wet} - \alpha \rho_1 t_i\right) \tag{4.8}$$

$$\|a_j y + b_j\| \leq C_j^T + d_j \tag{4.9}$$

$$m_0 = m_{wet}, \qquad r_0 = r_0', \qquad \dot{r}_0 = \dot{r}_0', \qquad r_N = \dot{r}_N' = 0 \tag{4.10}$$

$$u_{2,N} = u_{3,N} = 0 \qquad u_{1,N} \leq 0. \tag{4.11}$$

Note that in this problem the input of the initial and final conditions for the position and velocity are denoted with a $'$, to avoid confusion with the notation in the discretized case. In addition, the variables $z_{0,i}$ and the related $\mu_{1,i}$ and $\mu_{2,i}$ are the discretized versions of the original time dependent variables $z_0(t)$, $\mu_1(t)$ and $\mu_2(t)$ defined in equations 3.30 and 3.29. The discretized $z_{0,i}$ can be computed for each node i via

$$z_{0,i} = \ln\left(m_{wet} - \alpha \rho_2 (t_0 + i \cdot \Delta t)\right). \tag{4.12}$$

4.2 Transcription

Having arrived at a discretized version of the powered descent landing OCP, the task is now to convert problem 7 into the ECOS-feasible form given in problem 3, meaning that the corresponding matrices A,B,c^T,G and h have to be created covering the entire problem 7.

4.2.1 Transforming the Cost-Function

The cost function of problem 7 is implemented via the matrix c^T. Since the cost function in problem 7 is just a sum of weighted components of the optimization variable, i.e. of the slack variable σ_i at each node i, matrix $c^T \in \mathbb{R}^{1 \times n_y}$ can be filled with the weights such that the non-zero elements of c^T select the right elements of y via the product $c^T \cdot y$:

$$\sum_{i=0}^{N} w_i \sigma_i = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_0 & 0 & \cdots & 0 & w_N \end{pmatrix} \cdot y = c^T \cdot y. \tag{4.13}$$

In principle, all weights are equal, because all σ_i appear in the dynamics. However, the first and the last slack variable σ_i and σ_N are associated with the weights of $w_0 = w_N = \frac{1}{2}$, since we will apply the trapezoidal scheme for the integration of the dynamics in which the inital and final point are only incorporated with a prefactor $\frac{1}{2}$.

4.2.2 Transforming the Equality Constraints

Having transformed the cost-function, all equalities appearing in problem 7 including the dynamics have to be formulated as linear equations of the optimization variable via the matrices A and B to yield the equality in problem 3. The matrices $A \in \mathbb{R}^{n_e \times n_y}$ and $B \in \mathbb{R}^{n_e}$, where n_e is the number of equality equations, can be separated into the submatrices A_m and B_m, $m \in [1, 2, 3]$, since there are three different groups of equalities that have to be implemented. These groups of equalities are namely the initial conditions on the states, the dynamics and the final conditions on the states and partly on the control. Each of these groups require a specific procedure in order to be implemented correctly in the matrix equations. The matrices A_m and B_m obtained for each group can then be stacked resulting in the complete matrices A and B.
As an example, the first 7 lines of A and B associated to the submatrices A_1 and B_1 can be used to implement the initial conditions on the states with

$$A_1 y = \begin{pmatrix} 1\ 0\ 0\ 0\ 0\ 0\ 0 \dots 0 \\ 0\ 1\ 0\ 0\ 0\ 0\ 0 \dots 0 \\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \dots 0 \\ 0\ 0\ 0\ 1\ 0\ 0\ 0 \dots 0 \\ 0\ 0\ 0\ 0\ 1\ 0\ 0 \dots 0 \\ 0\ 0\ 0\ 0\ 0\ 1\ 0 \dots 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \dots 0 \end{pmatrix} y = \begin{pmatrix} r'_{1,0} \\ r'_{2,0} \\ r'_{3,0} \\ \dot{r}'_{1,0} \\ \dot{r}'_{2,0} \\ \dot{r}'_{3,0} \\ z_0 = \ln(m_{wet}) \end{pmatrix}, \tag{4.14}$$

illustrating that the matrix A mainly acts as a selecting matrix to choose the required elements of the optimization variable y.
In order to implement the dynamics within this framework, a general formulation for integrating states with the trapezoidal scheme has to be considered. The integration of the state x_i at the discretization node i via the trapezoidal scheme results in the state x_{i+1} of the following node with

$$x_{i+1} = x_i + \frac{\Delta t}{2}(\dot{x}_i + \dot{x}_{i+1}) = x_i + \frac{\Delta t}{2}\left(A^{dyn} \cdot y_i + A^{dyn} \cdot y_{i+1}\right) \tag{4.15}$$

$$\Leftrightarrow x_{i+1} - x_i - \frac{\Delta t}{2} A^{dyn} \cdot y_i - \frac{\Delta t}{2} A^{dyn} \cdot y_{i+1} = 0. \tag{4.16}$$

Considering the k-th component of the state vector x_i and x_{i+1}, equation 4.16 can be written as

$$x_{k,i+1} - x_{k,i} - \frac{\Delta t}{2} \sum_{l=1}^{7} A^{dyn}_{k,l} \cdot y_{l,i} - \frac{\Delta t}{2} \sum_{l=1}^{7} A^{dyn}_{k,l} \cdot y_{l,i+1} = 0. \tag{4.17}$$

Since the state components $x_{k,i}$ are also included in the stage variable y_i and the trapezoidal scheme makes use of the states and the control of the following stage y_{i+1}, we can consider the helpmatrix H

$$H = \begin{bmatrix} -\left(1 + dt \cdot A^{dyn}_{1,1}\right) & -dt \cdot A^{dyn}_{1,2} & -dt \cdot A^{dyn}_{1,3} & \dots & -dt \cdot A^{dyn}_{1,11} & 1 & 0 & 0 & \dots & 0 \\ -dt \cdot A^{dyn}_{2,1} & -\left(1 + dt \cdot A^{dyn}_{2,2}\right) & -dt \cdot A^{dyn}_{2,3} & \dots & -dt \cdot A^{dyn}_{2,11} & 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & & & \vdots & & & \ddots & & 0 \\ -dt \cdot A^{dyn}_{7,1} & \dots & -\left(1 + dt \cdot A^{dyn}_{7,7}\right) & \dots & -dt \cdot A^{dyn}_{7,11} & 0 & & \dots & & 1 \end{bmatrix}, \tag{4.18}$$

which allows equation 4.17 to be represented by the relation

$$H \cdot y_i = B_i. \tag{4.19}$$

The vector B_i had been defined in equation 4.3 and accounts for the influence of gravity in the change of the first velocity component. In order to provide the dynamics at each discretization node, the matrix A_2 can be filled by placing H such that the right elements of the stage variables y_i and y_{i+1} are selected. For the first node, the first 22 columns of A_2 are filled out with elements of H, which results in the selection of the required elements from the stage vector y_0 and y_1. For the second node, matrix H is included in A_2 starting at column 12, since the goal is now to select the elements of the stage variables y_1 and y_2 etc. The resulting shape of the matrix A_2 is sketched in the following equation:

$$A_2 = \left(\begin{array}{lcr} \multicolumn{2}{c}{H} & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots \\ & & 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \ldots \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 & \multicolumn{2}{c}{H \qquad\qquad 0 \ldots} \\ 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 & & 0 \ldots \\ \multicolumn{2}{l}{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0} & \\ \multicolumn{2}{l}{0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0} & \ddots \end{array} \right). \tag{4.20}$$

Note, that the matrix H actually consists of seven rows, which are compressed to two rows in equation 4.20 for illustrative reasons. The matrix B_2 is produced by stacking the vectors B_i of each node on top of each other.
As a third and final element of the equality constraints, the final conditions are incorporated in analogy to the initial conditions as described above, resulting in the submatrices A_3 and B_3. The overall procedure finally results in the equality

$$Ay = \begin{pmatrix} A_1 \\ A_2 \\ A_3 \end{pmatrix} y = \begin{pmatrix} B_1 \\ B_2 \\ B_3 \end{pmatrix} = B, \tag{4.21}$$

covering any relevant equality in problem 7. Since the dimension of each state vector is 7, and problem 7 contains 15 initial and final conditions in total, there are $E = N \cdot 7 + 15$ inequalities to be covered such that $A \in \mathbb{R}^{E \times n_y}$ and $B \in \mathbb{R}^E$.

4.2.3 Transforming the Inequality Constraints

All inequalities appearing in problem 7 have to be represented by the matrix G and vector h from the generalized inequality in problem 3. These inequality constraints can be classified as the linear inequalities and the second order cone inequalities, where the latter includes quadratic constraints in this project as shown in section 2.2.3. ECOS expects that the linear equalities are placed at the beginning of G and h, hence they are incorporated first.
For all linear inequalities that should be implemented into the matrices G and h, one needs to find matrices $C^T \in \mathbb{R}^{1 \times n_y}$ and the scalar b, such that $C^T \cdot y = b$ represents the given constraint. Then C^T and b can be stacked for each of the equality constraints to build up G and h consecutively.
The main linear inequalities in problem 3 are the upper boundary on the mass flow σ_i and the two boundaries for the mass variable z_i at each node respectively.
The upper boundary on σ_i from equation 4.8 can be reformulated as

$$\sigma_i + \mu_{2,i} z_i \leq \mu_{2,i} \left(1 - z_{0,i}\right) \tag{4.22}$$

Therefore, a matrix C_i^T and scalar b_i can be formulated for each node accordingly to represent this inequality. As an example, the bound on the slack variable of the first node σ_0 is introduced via

$$C_0^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \mu_{2,i} & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 \end{pmatrix} \tag{4.23}$$

$$b_0 = \mu_{2,0} \left(1 - z_{0,0}\right) \tag{4.24}$$

Defining C_i^T and b_i for each node by placing the non-zero elements in C_i^T and recalculating $\mu_{2,i}$ according to the considered node results in a set of matrices C_i^T and scalars b_i. Stacking all C_i^T fills up the first rows of the matrix G, while stacking all b_i represents the first components of h.
The other main linear inequalities are the upper and lower boundary on the mass given by

$$\ln\left(m_{wet} - \alpha\rho_2\left(t_0 + i \cdot \Delta t\right)\right) \leq z_i \leq \ln\left(m_{wet} - \alpha\rho_1\left(t_0 + i \cdot \Delta t\right)\right)$$

and can be incorporated in an analogous manner. For the upper boundary, where the task of C_i^T is to select the mass variable z_i for the corresponding node i, b_i contains the corresponding right hand side of the inequality. As an example, the matrices C_0^T and b_0 related to the first node are given by

$$C_0^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix} \tag{4.25}$$

$$C_0 = \ln\left(m_{wet} - \alpha\rho_1\left(t_0 + i \cdot \Delta t\right)\right). \tag{4.26}$$

Again, this set up has to be repeated for each node and the resulting matrices can be stacked and added to the previous constraint, filling up the rows in G and h consecutively. The lower boundary on z_i is implemented in absolute analogy to the upper boundary with the only difference of switched signs in C_i^T and b_i, as well as the changed constant ρ_2 in the calculation of b_i.

Implementing the linear constraints with this method and consecutively building up the matrices G and h also has an important advantage regarding adaptivity of the algorithm. If one wants to add a new linear inequality constraints, the algorithm only requires to create the matrices C^T and b representing the constraint and to stack those below the already implemented lines in G and h.
Such an additional linear inequality considered in our problem is the thrust pointing constraint throughout the entire flight in equation 3.41. It can be implemented, e.g. for the first node, via

$$C_0^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & \cos\left(\theta_{max}\right) & 0 & 0 & \dots & 0 \end{pmatrix} \tag{4.27}$$

$$b_0 = 0 \tag{4.28}$$

and for the other nodes by placing the non-zero elements accordingly, such that always the first component of the control u_i and the mass flow σ_i at node i are selected. An even stricter constraint holds for the final pointing of the thrust, which actually forces the two components of u_N parallel to the surface to be zero, such that the vector u_N strictly points downwards. The corresponding inequality constraint on the third control component, $u_{3,N} \leq 0$, therefore is set up with

$$C_0^T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & \dots & 1 & 0 & 0 & 0 \end{pmatrix}, b_0 = 0. \tag{4.29}$$

This constraint does not violate the previously introduced control pointing constraint holding for the entire trajectory, since the final net thrust vector pointing downwards will be inside the cone defined by the overall pointing constraint.

4.2.4 Transforming Second Order Cone - Inequality Constraints

The non-linear inequalities, i.e. second order cone and quadratic constraints, are prepared afterwards and transformed such that they can be processed by ECOS. For each constraint, this means to:

- Find matrices a_i, C_i^T and vectors or scalar b_i and d_i, such that the constraints are transformed into the form of 2.28. Note that the size of the non-scalar elements depends on the actual dimension of the corresponding cone.
- Stack these elements according to relation 2.36 or 2.44
- Add the resulting stacked elements to the already existing matrix G and vector h.

The first of these constraints is the slack constraint on the control variable $||u_i|| \leq \sigma_i$ at each node. As an example, it can be introduced for the first node by choosing $d_0 = 0$, $b_0^T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ and

$$a_0 = \begin{pmatrix} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0 \ldots 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0 \ldots 0 \\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0 \ldots 0 \end{pmatrix},\ a_0 \in \mathbb{R}^{3\times n_y} \tag{4.30}$$

$$C_0^T = \begin{pmatrix} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0 \ldots 0 \end{pmatrix}^{\mathrm{T}},\ C^T \in \mathbb{R}^{1\times n_y} \tag{4.31}$$

which fulfills the original formulation of the constraint since

$$\|a_0 y + b_0\| = \begin{pmatrix} u_{0,1} \\ u_{0,2} \\ u_{0,3} \end{pmatrix} \leq \sigma_0 = C_0^T y + d_0. \tag{4.32}$$

For each node i, this constraint defines a second order cone of dimension $3+1$, and the non-zero elements of a_i and C_i^T again need to be set according the corresponding node to select the correct elements of the stage variables y_i.
In contrast, imposing the lower boundary on the slack variable σ_i as a second order cone constraint requires a more intensive approach, since it is a quadratic constraint that has to be reformulated accordingly. In order to transform the constraint given by

$$\mu_{1,i}\left[1 - (z_i - z_{0,i}) + \frac{(z_i - z_{0,i})^2}{2}\right] \leq \sigma_i \tag{4.33}$$

into a second order cone constraint, it first has to be formulated as a standard quadratic inequality $y^T F_i^T F_i y + p_i^T y + q_i^T x \leq 0$. Simple re-arrangement of equation 4.33 results in

$$z_i^2 - (2z_{0,i} + 2)\, z_i - \frac{2}{\mu_{1,i}}\sigma_i + 2z_{0,i}^2 + 2z_{0,i} + 2 \leq 0, \tag{4.34}$$

which indeed has the desired shape if the selecting matrices F_i, p_i^T and scalar q_i are chosen properly. The approach in this case is again demonstrated for the first node $i = 0$. Selecting

$$F_0 = \left(0, 0, 0, 0, 0, 0, \frac{\sqrt{2\mu_{1,0}}}{2}, 0, 0, 0, \dots\right) \tag{4.35}$$

$$p_0^T = (0, 0, 0, 0, 0, 0, -(\mu_{1,0} + z_0\mu_{1,0}), 0, 0, 0, -1, 0, 0, \dots) \tag{4.36}$$

$$q_0 = \mu_{1,0}(1 + z_{0,0}) + 0.5\mu_{1,0} \cdot z_{0,0}^2 \tag{4.37}$$

results in the relation

$$y^T F_0^T F_0 y + p_0^T y + q_0 = \frac{\mu_1}{2} z_0^2 - (\mu_{1,0} + \mu_{1,0} z_{0,0}) z_0 - \sigma_0 + (1 + z_{0,0}) + \frac{1}{2}\mu_{1,0} z_{0,0}^2 \leq 0, \tag{4.38}$$

which is an equivalent formulation of the quadratic constraint on σ_0 given by equation 4.33. These three elements can therefore be stacked according to equation 2.44 resulting in a $2 + 1$ dimensional cone for each node i.
As a final example for the implemented second order cone constraints, we consider the glide slope constraint introduced in equation 3.48:

$$\sqrt{\left(r_{2,i} - r'_{2,N}\right)^2 + \left(r_{3,i} - r'_{3,N}\right)^2} \leq \frac{\left(r_{1,i} - r'_{1,N}\right)}{\tan(\gamma_{max})} \tag{4.39}$$

A valid formulation of this constraint as a second order cone can be performed, e.g. for the first node, via the set

$$a_0 = \begin{pmatrix} 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \end{pmatrix}, \qquad a_0 \in \mathbb{R}^{2 \times n_y}, \tag{4.40}$$

$$C_0^T = \begin{pmatrix} \frac{1}{\tan(\tilde{\gamma})} & 0 & 0 & \dots & 0 & 0 \end{pmatrix}, \qquad C_0^T \in \mathbb{R}^{1 \times n_y}, \tag{4.41}$$

$$b_0^T = \begin{pmatrix} -r'_{2,N} & -r'_{3,N} \end{pmatrix}^T, \tag{4.42}$$

$$d_0 = \frac{-r'_{1,N}}{\tan(\tilde{\gamma})}, \tag{4.43}$$

because this set yields the $2 + 1$-dimensional cone

$$\|a_0 y + b_0\| = \sqrt{\left(r_{2,0} - r'_{2,N}\right)^2 + \left(r_{3,0} - r'_{3,N}\right)^2} \leq \frac{r_{1,0} - r'_{1,N}}{\tan(\gamma_{max})} = C_0^T + d_0. \tag{4.44}$$

The non-zero elements in a_i and C_i^T have to be set according to the corresponding node i such that the components $r_{1,i}$, $r_{2,i}$ and $r_{3,i}$ are selected. Stacking the matrices created for each node implements this constraint into the matrices G and h.
Again, the general procedure of formulating the constraints as second order cone constraints and then stacking the created elements to build up the matrices G and h allows for a high adaptivity regarding the implementation of further constraints, which can simply be added and stacked below the already implemented constraints in G and h.
As seen above, the conic constraints can set up cones of different dimensions and thus cover a different amount of rows within G and h. ECOS therefore requires additional inputs for the correct interpretation of the matrices G and h. These inputs are namely the number of linear inequality constraints *diml*, meaning that the first *diml* rows in G and h are related to linear inequality constraints, and an array *dimq* of size m, where m is the number of second

order cone constraints. As an example, for $N = 100$ nodes and given the three second order cone constraints implemented above, we have $m = N \times 3 = 300$ second order cone constraints. For each of these constraints, the array saves the dimension of the related cone. If new constraints are introduced throughout the following course of the thesis, it is assumed that they are transcribed and implemented in the same way as the previously described constraints.
The transcription method results in highly sparse matrices G and A, since there are only a few non-zero elements in each row of these matrices. This sparse character is further increased for more discretization nodes. ECOS therefore expects G and A in the so called column-compressed-storage (CCS) format [4]. While each single element of a matrix is usually defined and represented by their corresponding row and column index, the column-compressed formulation is an alternative representation of matrices and especially useful for sparse matrices. However, handling sparse matrices is not the task of the actual transcription algorithm presented here, since there are powerful libraries and commands available in different programming environments for the handling and conversion between the different matrix representations. The required CCS-format has larger influence in the call of ECOS in the programming language C, and will therefore be investigated in the related section 5.1.1.

4.3 Implementation and Verification

In order to test and verify the discretization and transcription described above, the method is implemented into a Matlab script, making use of ECOS compiled as a Matlab executable. To keep the possibility of testing several landing scenarios, i.e. different spacecraft properties and also different constraints, this script is developed in the most generic way possible. The general approach of the script is depicted in the diagram figure 4.1 with the following steps:

- The simulation parameters together with the initial conditions, final conditions, flight time t_f and the dynamic matrix A^{dyn} are read in
- Application of the algorithm described in the previous section for transcribing the powered descent landing problem into an ECOS-feasible formulation, which is then passed to ECOS
- The solution vector provided by ECOS is processed afterwards in order to extract the computed trajectory and thrust profile.

The arrays G and A passed to ECOS after the transcription are converted into the aforementioned CCS-format via the Matlab command `sparsify` acting on the related arrays in the script.

$$\begin{tikzcd}\boxed{\text{Read-In}} \arrow[r] & \boxed{\text{Transcription}} \arrow[r, "{c^T, A, B, G, h}", "{diml,\ dimq}"'] & \boxed{\text{ECOS}} \arrow[r, "y"] & \boxed{\text{Process Results}}\end{tikzcd}$$

FIGURE 4.1: Flow diagram of the algorithm

The algorithm is tested and verified with an example scenario of a spacecraft landing on Mars, which is also the main example presented in [1]. A comparison of the results obtained by the algorithm developed in this work to the results provided in [1] is therefore used as a verification. The specific parameters for this simulation example are:

$$g = \begin{pmatrix} -3.7114 \\ 0 \\ 0 \end{pmatrix} \frac{\text{m}^2}{\text{s}} \quad r'_0 = \begin{pmatrix} 1500 \\ 0 \\ 2000 \end{pmatrix} \text{m} \quad \dot{r}'_0 = \begin{pmatrix} -75 \\ 0 \\ 100 \end{pmatrix} \frac{\text{m}}{\text{s}} \quad r'_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \text{m} \quad \dot{r}'_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \frac{\text{m}}{\text{s}}$$

$$m_{wet} = 1905 \text{ kg} \quad I_{sp} = 225 \text{ s}$$

$$\bar{T} = 3100 \text{ N} \qquad T_1 = 0.3 \cdot \bar{T} \quad T_2 = 0.8 \cdot \bar{T}$$

$$n = 6 \quad \phi = 27^\circ$$

$$\gamma_{max} = 4^\circ \quad t_f = 72 \text{ s or } 81 \text{ s} \qquad N = 72 \text{ or } 81$$

The first simulation is performed without any restrictions regarding the glide slope angle γ_{max} or the thrust pointing, except for the thrust to point downwards at the final position. The flight time of 72 s was found by [1] to be optimal, which is why it is used here for the comparison. Accordingly, the number of nodes is set to meet the flight time and to yield a step size of $\Delta t = 1$ s.
For this first simulation, the result is displayed in the plots of figure 4.2, showing the actual trajectory, the velocity and accelerations, the net force acting on the spacecraft, the normalized throttle level and the angle θ between the thrust vector and the first axis, i.e. between the pointing of the thrusters and the normal to the surface. The Bang-Bang principle for the optimal solution is clearly visible in the normalized throttle level, yielding an optimal solution. As hinted by the blue line on the top left plot, the vehicle's trajectory is below the surface during the time interval from $t \approx 30$ s until $t \approx 50$ s , which is an unrealistic result in the context of the landing problem. However, this is a consequence from the fact that no constraint is set regarding possible sub-surface flights or the glide slope angle.

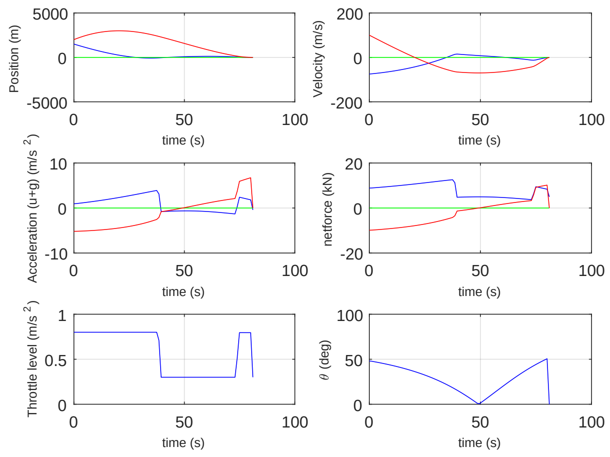

FIGURE 4.2: Simulation results without glideslope constraint. Except for the plots for the throttle level and angle θ, the red curve represents the first axis, blue the third axis and green the second.

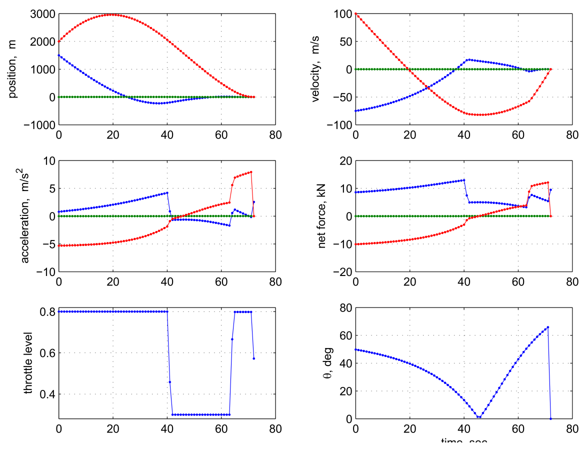

FIGURE 4.3: Results obtained by [1]

The methodology developed in this thesis is verified by the results for an identical simulation setup from [1] shown in figure 4.3, which match our simulations. The only clearly visible difference is the obtained control at the final node, e.g. in the throttle level at the final time t_f. While the final throttle level is at maximum in the method developed in this thesis, it neither stays at maximum nor reaches the minimum for [1], which might be a numerical artifact of the used discretization in [1]. Another invisible difference is the consumed fuel, which in the case of this thesis is calculated by the difference of the mass m_{wet} and the mass at t_f, i.e. by $m_{fuel} = \exp^{z(t_0)} - \exp z(t_f)$. In our simulation, the consumed fuel is 389.39 kg, while the simulations in [1] show a fuel consumption of 387.9 kg. This difference of around 1.5 kg can also be explained by a different discretization and integration technique. For example small time shifts in the computed control history and thus for the throttle level can result in several grams or kilograms of difference in fuel consumption.
To avoid the potential sub-surface flight, the glide slope constraint is imposed on the trajectory in a second simulation run. Again, the simulation results obtained with this specific transcription and ECOS meet the results from [1] as shown in figure 4.4 and 4.5 . The glide slope constraint now avoids that the spacecraft flies sub-surface, requiring a slightly longer fuel-optimal flight time with $t = 81$s [1]. An illustration on how the glideslope constraint is fulfilled is presented in figure 4.6. The straight line defines the limit of the glide slope constraint and only trajectories above that line are allowed. The angle between this line and the surface $r_1 = 0$ corresponds to the minimum glide slope angle $\gamma_{max} = 4°$. Note that r_2 is ignored in this plot since it is set to zero and does not change throughout the entire trajectory. Again, the fuel consumption of 400.48 kg is slighty more than the one found by [1] with 399.5 kg, although the overall computed trajectories and thrust profiles agree for both simulations with the one once found by [1].
The strong agreement between in the trajectories computed within both frameworks, ignoring minor numerical issues, verifies the applied method using ECOS and the corresponding transcription for solving the OCP. Even more, the runtime for the transcription and ECOS computing the optimal solution within this first Matlab based script is in the order of 0.2 s, which suggests a very short computation time also for other landing problems and especially within other programming environments like C. Therefore, the developed transcription together with ECOS can potentially be used on an on-board computer of a landing system for real-time trajectory optimization. The following steps are to transform this simulation into an on-board feasible environment, mainly including the transition from the Matlab-based IDE into C-code and the implementation within a Simulink-Environment.

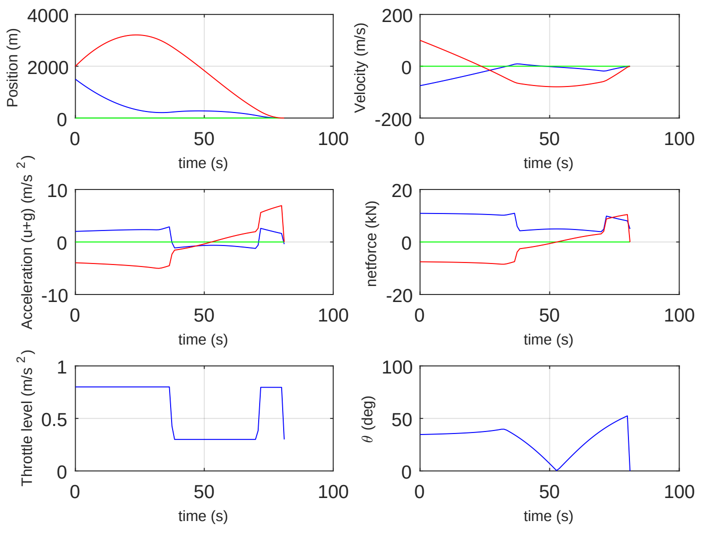

FIGURE 4.4: Simulation results with glideslope constraint. First axis in red, third one in blue.

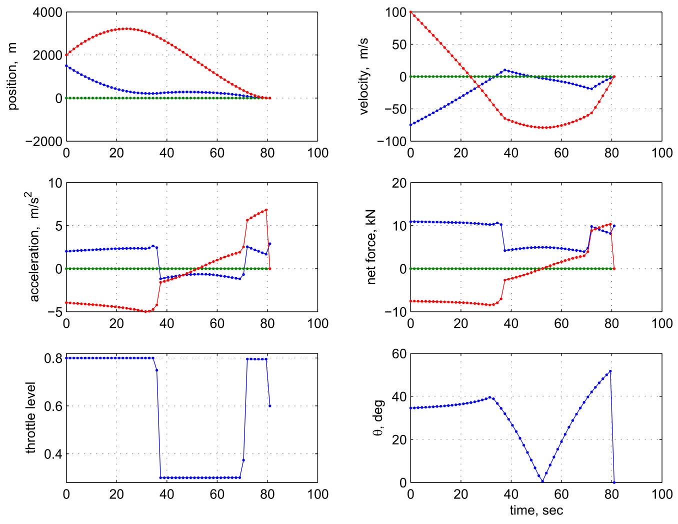

FIGURE 4.5: Results with glideslope constraint by Acikmese [1]

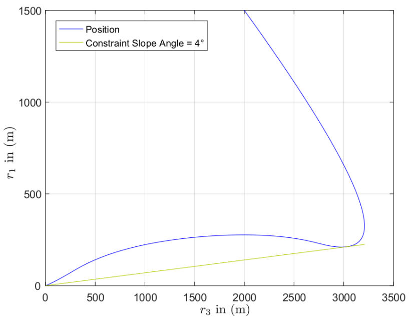

FIGURE 4.6: Illustration of the simulation results regarding the glides lope constraint

5 Development of an On-board Feasible Guidance Function

The second goal of this thesis is to develop an on-board feasible guidance function for EAGLE based on the developed and verified convex trajectory optimization algorithm. The first corresponding task is the transition from the Matlab environment to the programming language C, meaning that the algorithm has to be converted into C-code. In addition, this C-code version should be embedded in a Simulink model, since the *QNX* on-board system of EAGLE executes algorithms which are tested in a Simulink environment and cross-compiled by the Simulink-coder, before it is extended to an actual guidance function.

5.1 Transition to C

The principle algorithm for solving the convexified OCP remains unchanged as in figure 4.1. Although similar, the call of ECOS in C is slightly different than in the MATLAB environment and hence the C-code version of the algorithm requires an adapted approach. The `setup` function for ECOS, which is basically a memory preparation step before the actual solving routine of ECOS, expects the arguments listed in table 6.1 (see [9]).

Data Type	Name	Definition
`idxint`	n	Number of variables, i.e. length of optimization variable y
`idxint`	m	Number of inequality constraints, i.e. length of vector h and first dimension of matrix G
`idxint`	p	Number of equality constraints, i.e. length of vector B and first dimension of matrix A
`idxint`	l	Dimension of positive orthant, i.e. number of linear inequality constraints
`idxint`	ncones	Number of second order cones, i.e. number of second order cone constraints
`idxint*`	q	Array of length $ncones$ with integers defining the dimension of each second order cone
`pfloat*`	Gpr	Array of values of matrix G in column-compressed-storage format
`idxint*`	Gjc	Column index array of matrix G in column-compressed-storage format
`idxint*`	Gir	Row index array of matrix G in column-compressed-storage format
`pfloat*`	Apr	Array of values of matrix A in column-compressed-storage format
`idxint*`	Ajc	Column index array of matrix A in column-compressed-storage format
`idxint*`	Air	Row index array of matrix A in column-compressed-storage format
`pfloat*`	c	Array representing c^T
`pfloat*`	h	Vector h from $Gy \preceq h$
`pfloat*`	b	Vector B from $Ay = B$

TABLE 5.1: Arguments of ECOS-setup call in C

The defined data types `pfloat` and `idxint` are ECOS-specific synonyms for known data types. In our implementation, `idxint` are variables of type long, while `pfloat` are double-variables. The main difference in the call of ECOS in C compared to the Matlab executable is that several arrays related to the matrices G and A have to be passed. These arrays arise from the column-compressed-storage (CCS) format expected by ECOS and a library can be used to handle the matrix setup according to this format in a user friendly way. Within this work, the open-source library *CSparse* by Timothy Davis [8] has been identified as a suitable choice and is introduced in the following.

5.1.1 *CSparse* Library for CCS-Matrix Creation

The main criterions for the selection of a suitable sparse matrix library are

- Open-Source availability: In order to allow follow-up adaptions and detailed understanding of the software, i.e. of the storage handling, open-source availability is key for the implementation on embedded systems.
- Simplicity: The transcription method requires detailed and exact filling of matrix elements. In addition, the generic nature of the developed algorithm allowing for different simulation setups may require a complex generic selection of array elements to fill G and A according to the desired simulation. To keep the overview, simple handling and function calls for filling the elements of the sparse matrices are useful.
- Storage handling: Since the optimization algorithm will be run on an embedded system, extensive dynamic storage allocation has to be avoided.

Especially the simplicity in the use of the chosen *CSparse* library is an advantage. The initialization of a sparse matrix starts with the function `cs_spalloc` including arguments defining the number of rows, number of columns, maximum number of elements, a check whether values are going to be filled in the matrix, and a final check whether the sparse matrix is formulated in triplet form or in the CCS form.
In both formulations, triplet and CCS, the sparse matrix elements are represented by three arrays defining the value, the row and the column. For a sparse triplet matrix, every non-zero element of the matrix is represented by one value in the value array, one value in the row array, and one in the column array, where the values in the latter two arrays define the exact location of the element. The array structure for the triplet matrix already reduces the required storage, since only non-zero elements are considered.
Also in the CCS-framework a matrix is described by three arrays. In contrast to a triplet matrix, the non-zero elements are saved column-wise in the value-array from the top of the matrix to the bottom. The elements of the row-array save the row number of each of these elements, and therefore the row-array has the same length as the value array. The main-difference in the CCS version is the reduced array for the column numbers, which only saves the index numbers of non-zero elements that start a new column. An example for this is presented in the appendix A.2. However, libraries like CSparse are designed to handle CCS-matrices without expecting explicit knowledge about the underlying theory from the user, such that one can rely on the natural representation of matrices with a row and a column number defining the position of the matrix elements.
The triplet form handled by *CSparse* is applied for the preparation and filling of the matrices A and G within our framework, since the corresponding values are easily set via the *CSparse* function `cs_entry`. As an example, for filling the matrix element $A_{1,2} = 3$, the function call is `cs_entry(A,1,2,3)`, which allows the user to fill matrix elements in a natural way without any consideration of sparse matrix theory.

If the filling of a value in the matrix exceeds the maximum number of non-zero matrix elements specified in the matrix initialization, *CSparse* allocates more space for the matrix by doubling the size of the arrays to allow twice the amount of values. This procedure is a tradeoff to allow generic matrix handling while avoiding a large number of allocation processes at the same time. Hence, multiple dynamic allocation are naturally avoided by the library.
Since ECOS requires the arrays in the CCS-format, one can use the function `cs_compress`, which converts the final filled triplet matrix into a CCS form. *CSparse* saves all matrices in a special struct containing the three arrays that define the sparse matrix and several other parameters like the number of elements or the type of the sparse matrix, i.e. triplet or CCS. When calling ECOS after the compression into the CCS format, one can handover the pointers to the value-, column- and row-arrays of these matrix structures. In total, *CSparse* fulfills the requirements stated above and turns out to be a powerful tool for the purposes of this work.

5.1.2 Pre-allocation of Arrays

Besides the new approach of handling sparse matrices in the transcription via *CSparse*, the main difference in the transition from Matlab to C arises from the required pre-allocation of arrays in C. While arrays can be stacked in Matlab without requiring any pre-allocation, the sizes of all arrays need to be fixed in C before the first entries are assigned. Especially the size of the arrays for h and B need to be pre-allocated, while *CSparse* handles the sizes of the arrays A and G dynamically if the initially specified number of non-zero elements is exceeded. However, all matrices and vectors have deterministic dimensions as soon as the problem is set, i.e. as soon as the number of nodes, number of equality constraints, and number of inequality constraints with the corresponding cone dimensions are set. This pre-allocation does not affect the principle algorithm, but needs to be considered and represents the main adaption in the transition from the Matlab environment to a C-based application.

5.2 Convex Simulator in Simulink

The created C-code version of the trajectory optimization algorithm has to be implemented in a Simulink model, since it will be used within a guidance function in the Simulink simulation environment for EAGLE, and compiled via the Simulink-coder for the on-board computer. In addition, another advantage of this implementation is the possibility to create a graphical user interface with the Simulink model, which allows for simple handling of simulation inputs and parameters. The C-code is embedded into a Simulink S-function, having the initial and final conditions, the flight time, as well as the dynamic matrix A^{dyn} as input. The S-function is also masked for tuning several parameters according to the given simulation. An up to date list of these parameters is given in the appendix A.3 together with the corresponding block diagram of this Simulink model, which basically follows the structure of the diagram in figure 4.1 by fusing the transcription and solution process in ECOS into a single trajectory optimization block. The entire solution vector y provided by ECOS is put out from the S-function block besides timing information of the simulation and an additional output with simulation information to the display block. The main purpose of this display block is a proper visualization of the results and the extraction of the actual trajectory and thrust profile from the solution vector.

5.3 Run-time Results and Processor-in-the-Loop-Verification

The optimal trajectory calculated for the example mars landing presented in section 4.3 has been obtained with the developed C-code version within a total computation time of 0.12 s on a regular desktop computer, including one millisecond for the entire transcription. This indicates on the one hand that the actual algorithm can be expected to be on-board feasible thanks to the quick computation, on the other hand the lower boundary on the computation time is actually set by the solution process in ECOS and the problem size.
As a first valuable on-board test, the Simulink model has been included in the general EAGLE-Simulink environment of DLR and it has been cross-compiled for the usage on EAGLE's on-board computer. During a processor-in-the-loop test run on the on-board system, the Mars landing simulation from section 4.3 including the glide slope constraint has been run in parallel, but isolated from the other on-board activities to obtain a rough estimate on the actual on-board performance of the algorithm. In these tests, the computing time for this algorithm have been determined to be around 1 second, meaning a 10-times longer computation time on an the-board architecture compared to the desktop computer.
However, the Mars landing example requires high computation resources while EAGLE-typical maneuvers are less demanding, especially because of the smaller flight time for the maneuvers, which will also reduce the number of nodes within our framework resulting in a shorter computation time. This implies that the expected computation time of the developed software for EAGLE-typical problems on the on-board computer is less than one second and therefore on-board feasible. Hence, it can be implemented as a prototype guidance function for the control system of EAGLE, which will require some further adaptions of the so far developed algorithm as investigated in the following.

5.4 Guidance Function Design

Although the trajectory optimization algorithm converted into C is in principle on-board feasible, it does not provide a closed guidance function, as it expects a fixed flight time as one of the inputs. Therefore, the algorithm only optimizes the trajectory for a fixed flight duration. In fact, finding the fuel optimal trajectory requires to find the thrust profile and flight time which minimize the fuel consumption as discussed in 3.1, meaning that also a corresponding flight time has to be found. A typical approach in optimal control is to transform the OCP such that the flight time becomes an element of the optimization variable y and the OCP is transformed into an autonomous system. However, this is not applicable for the dynamics of the powered descent landing problem in the framework of convex optimization, since e.g. the constraints are represented by linear functions of the optimization variable. The integration of the dynamics of the problem 4 with trapezoidal scheme includes a product $\Delta t \cdot \dot{x}$. If t_f, and thus the node separation Δt, is an element of the optimization variable y, a product of two different elements of y appears within the dynamics, which is impossible to implement in a linear relation via selection matrices as it is done in the developed transcription algorithm in section 4.1. This is a disadvantage of the convex optimization formulation occuring for any integration scheme.
In order to find the optimal flight time, a bi-sectional approach is commonly chosen. As an example, in [1] a lower and an upper limit for the flight time are heuristically derived from the maximum and minimum thrust together with the available fuel, followed by a scan within this interval of flight times. The optimal solution in this method is found with the trajectory and the corresponding flight time that yields a minimum fuel consumption. However, this approach is not completely applicable for a real-time system, since the scan-interval can become relatively large, e.g. about 100 seconds in the calculations from [1]

resulting in a high number of required iteration steps.
In this thesis, a similar, but in principle shorter approach is chosen. The algorithm derived in 4.3 is extended by a preceding flight time estimation based on kinematic considerations. It provides a rough estimation of the optimal flight time, which is assumed to be shorter than the fuel-optimal flight time, and then adds a fixed margin on this optimal flight time to arrive at a value close to the expected fuel-optimal flight time. In addition, one has to consider the case where this method underestimates the fuel-optimal flight time separately, which can result in an infeasible and unsolvable problem formulation for ECOS. To solve these cases, the flight times are increased iteratively in the case of detected infeasibilities, until a feasible problem is created and solved, which results in a robust and reliable on-board applicable algorithm for close to optimal trajectories.

5.4.1 Optimal Flight Time Estimation

In order to estimate the optimal flight time of a vehicle, we consider a simple kinematic model of the spacecraft with a fixed mass throughout the whole flight. We also assume that the spacecraft will have a vanishing velocity when arriving at the target, which is true for the planned maneuvers with EAGLE. In a 1-D example, the vehicle tries to reach the point s_{end} starting from s_0. In the time-optimal case, the thrusters will always run on maximum power, which provides a maximum acceleration for the spacecraft.
Assume a spacecraft is at rest at t_0. At the beginning of the maneuver, it will apply full thrust accelerating the vehicle towards the target. At a certain point t_1, which in this example is right in the middle of the trajectory due to the vanishing velocities at start and end, the vehicle has a velocity of v_2 as a result of the previous acceleration. It will then turn around at this point and provide full thrust in the opposite direction for deceleration while the movement towards the target continues. Because of the basic laws of kinematics, it will have the desired final velocity of $v_{end} = 0$ at the target. This example shows the fundamental concept on which the proposed flight time estimation is based on and is illustrated in image 5.1.

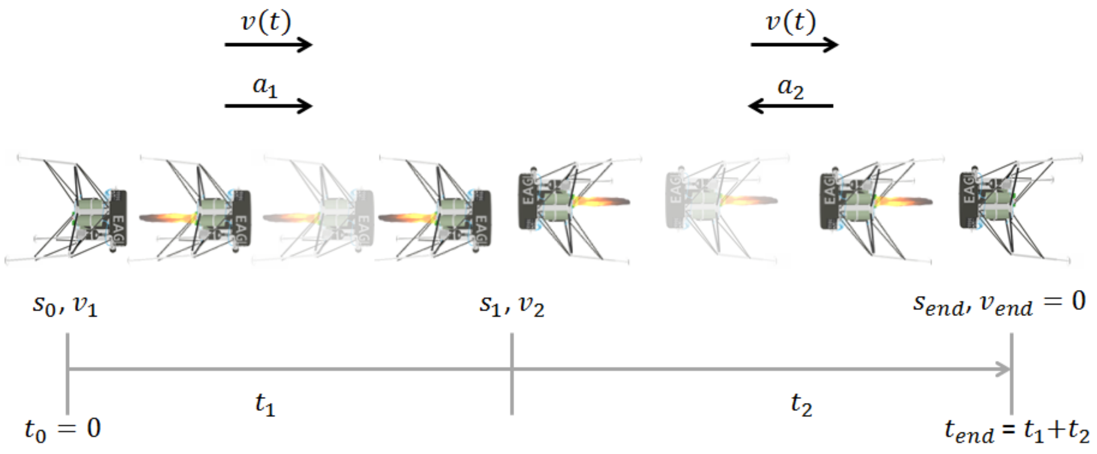

FIGURE 5.1: Sketch of the heuristic defining the optimal time of flight in one dimension

As in the example above, one can divide the movement of the vehicle into two segments of uniform acceleration:

$$s_{end} = s_1 + s_2 + s_0 = \frac{1}{2}a_1 t_1^2 + v_1 t_1 + \frac{1}{2}a_2 t_2^2 + v_2 t_2 + s_0. \tag{5.1}$$

In this formulation, the index 1 always denotes the segment, in which the vehicle is accelerating with a_1 **towards** the target point, while 2 denotes the segment where the acceleration is pointing **away** from the target. If gravity is not acting along the considered dimension, then $a_1 = -a_2$. The velocity v_1 is the initial velocity of the spacecraft $v_1 = v(t_0)$, while v_2 is the velocity at the end of the segment 1. The entire optimal flight time can then be calculated from the flight times for both segments with $t_o = t_1 + t_2$. Hence, the goal is to write equation 5.1 as a function only containing one of the two times for the segments together with only known parameters like initial and final conditions, such that it can be solved and the flight times can be obtained.
Depending on the initial velocity, there exist possible cases where the segment 1 is not existing, since the vehicle will have to break for the entire time and in even further cases it might overshoot over the desired target s_{end}. We therefore start with the calculations related to segment 2, since there will always be a segment involved in which the vehicle will have to break in order to stop at the final point.
The time to reach $v_{end} = 0$ starting with the velocity v_2 is given by

$$t_2 = \frac{v_{end} - v_2}{a_2} = -\frac{v_2}{a_2} \tag{5.2}$$

$$\Leftrightarrow v_2 = v_{end} - t_2 a_2 = -t_2 a_2. \tag{5.3}$$

Note, that the segment 2 has been defined to be the segment where a_2 is pointing away from the target. Therefore, if in the 1-D case $s_e < s_0$, then $a_2 > 0$ and $v_2 < 0$. In contrast, if $s_e > s_0$, then $a_2 < 0$ and $v_2 > 0$, such that we will always obtain a positive time for the braaking segment. Similar, the flight time t_1 for the first segment is the time to accelerate the vehicle from the initial velocity v_1 to the velocity v_2 at the switching point

$$t_1 = \frac{v_2 - v_1}{a_1} \Leftrightarrow t_1 = \frac{v_{end} - t_2 a_2 - v_1}{a_1} = \frac{-t_2 a_2 - v_1}{a_1}, \tag{5.4}$$

where equation 5.3 has been inserted. This flight time also considers the case where there is no first segment, meaning that $t_1 = 0$ if $v_1 = -t_2 a_2 = v_2$. The switching point of starting the braking then falls onto the initial point of the maneuver.
Replacing t_1 in equation 5.1 by equation 5.4 and a reformulation results in a quadratic equation on t_2 that can be solved:

$$s_{end} = \frac{1}{2} a_2 \left(1 - \frac{a_2}{a_1}\right) t_2^2 - \frac{1}{2} \frac{v_1^2}{a_1} + s_0 \tag{5.5}$$

$$\Leftrightarrow t_2 = \sqrt{\frac{s_{end} - s_0 + \frac{1}{2}\frac{v_1^2}{a_1}}{\frac{1}{2} a_2 \left(1 - \frac{a_2}{a_1}\right)}}. \tag{5.6}$$

The corresponding flight time t_1 of the first segment can be obtained by inserting t_2 into equation 5.4, and therefore the estimated optimal flight time is $t_o = t_1 + t_2$. In the following, we will refer to this procedure as flight time algorithm 1. The above equations only result in a reasonable flight time, if the vehicle's initial velocity is small enough such that it does not overshoot the desired target point.
If the vehicle actually overshoots the target point, one can consider the trajectory separated into two parts: first, the spacecraft decelerates during overshooting, reaching a velocity of $v_1' = 0$ at the point s_0'. Afterwards, it accelerates towards the target point in one segment of the trajectory and decelerates again when approaching the target point. Such a maneuver is sketched in figure 5.2. The second part can be solved by the flight time algorithm 1 using s_0' as initial position and the initial velocity $v_1' = 0$. In order to include the overshooting for an

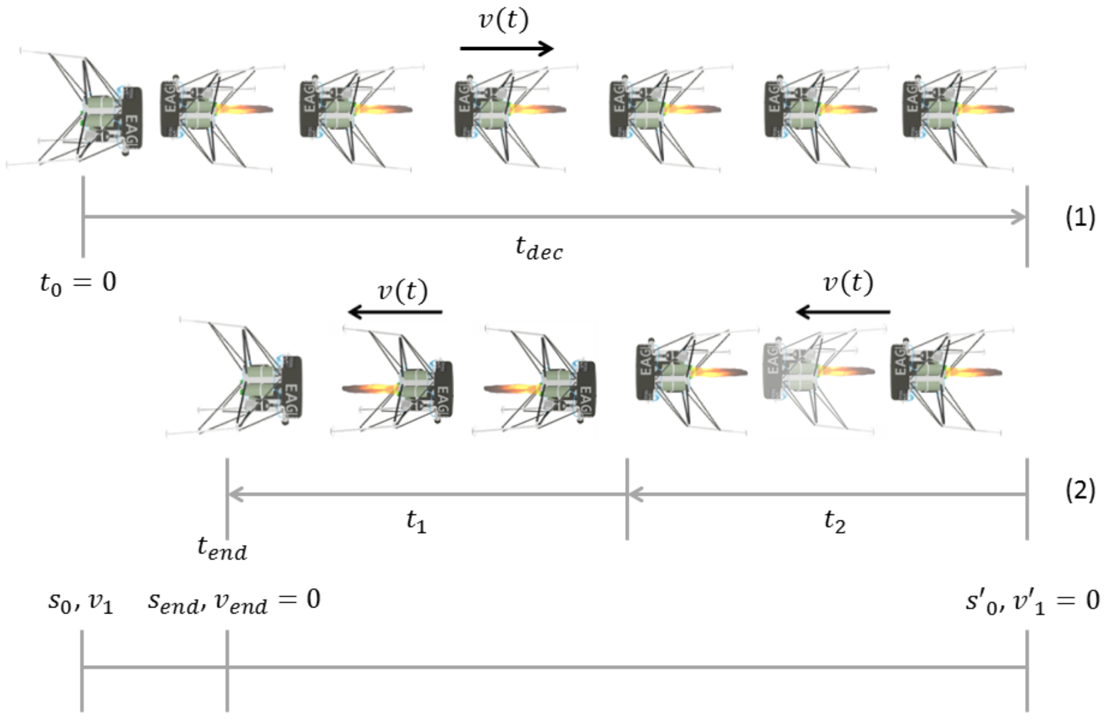

FIGURE 5.2: Sketch for a maneuver with overshooting

arbitrary maneuver, the following procedure can be applied: First, the time t_{dec} to decelerate the vehicle to zero velocity can be calculated as $t_{dec} = \frac{v_1}{a_3}$. The corresponding final point s'_0 is given by

$$s'_0 = \frac{1}{2} a_3 t_{dec}^2 + v_1 t_{dec} + s_0. \tag{5.7}$$

In this case, a_3 is the acceleration pointing against the velocity vector, either being equal to a_1 or to a_2. Then the flight time algorithm 1 can be applied with the inital position s'_0 and velocity v'_0 to calculate t_1 and t_2 for the second part of the trajectory, resulting in the total flight time of $t_o = t_{dec} + t_1 + t_2$.
The complete proposed and adapted method for estimating the optimal flight time t_0 is therefore given by

```
Calculate $t_{dec}$ and $s'_0$
**if** $s'_0$ further away from $s_0$ than $s_{end}$ **then**
    Calculate $t_1$ and $t_2$ with $s'_0$ and $v'_0$ as initial conditions with flight algorithm 1
    $t_o = t_{dec} + t_1 + t_2$
**else**
    Calculate $t_1$ and $t_2$ directly from $s_0$ and $v_0$ with flight algorithm 1
    $t_o = t_1 + t_2$
**end if**
```

5.4.2 Optimal Flight Time Estimation in Two Dimensions

The above developed method results in the optimal flight time for movements in one dimension. However, a flight time estimation for maneuver in more than just one dimension is required.
In the two-dimensional case, a first guess might be to project the two-dimensional movement into one dimension by applying Pythagoras' law. In fact, this only results in a useful approach if the initial velocities are zero or if the initial velocity vector directly points towards the target. As soon as this is not the case, the movement of the vehicle will not be only along a straight line and therefore a different estimation has to be found. Although most of the EAGLE maneuver will start with a vanishing velocity, it will be useful to obtain a more general approach including different initial velocities in order to develop a more complete guidance function.
As such, our approach for two dimensional movements is to split the maneuver into two separate movements and to calculate the optimal flight time for each dimension individually. If the maneuver starts at $x_0 = [x_{1,0}, x_{2,0}]$ and stops at $x_{end} = [x_{1,end}, x_{2,end}]$, then the optimal flight time $t_{o,1}$ is calculated via the flight time algorithm 1 for the first of the two dimensions with a maneuver from $x_{1,0}$ to $x_{1,end}$ and in analogy for the second dimension resulting in the time $t_{o,2}$. Then there are two possible ways of achieving an estimation of the fuel-optimal flight time in the two dimensions. One can use a geometrical approach, via

$$t_{geo} = \sqrt{t_1^2 + t_2^2}. \tag{5.8}$$

This geometrically motivated method is especially useful if the distances that have to be covered in both dimensions and the available thrust in both dimensions are comparable, because then both movements contribute with a considerably large amount of time to the movement.
However, if the movement in one dimension is much longer and especially if the achievable acceleration in one of the dimensions is much smaller than in the other, then this dimension dominates the actual flight time. As a second estimation for the 2-D optimal flight time covering such cases we select the maximum out of the two derived times $t_{0,1}$ and $t_{o,2}$ and then add a margin on that maximum:

$$t_{max} = \max\left[t_{0,1}, t_{0,2}\right] \cdot 1.2 + 2. \tag{5.9}$$

Note that this margin is applied, since $\max\left[t_{0,1}, t_{0,2}\right]$ is an estimate of the optimal time of flight in those two dimensions and will clearly underestimate the fuel-optimal time of flight. The parameters added in equation 5.9 are freely choosable, but throughout this work they have been used as a first guess and already turned out to generate reasonable results. The final estimation for the fuel-optimal time of flight in two dimensions is then selected as the maximum out of the two times as:

$$t_{o,2D} = \max\left[t_{geo}, t_{max}\right] \tag{5.10}$$

This 2-dimensional approach including the margins set in the calculations does not follow a basic underlying theory and is developed to obtain a time estimation as close as possible to the fuel-optimal flight time. It is called flight time algorithm 2 in the following and it sets the base for the time estimation in three dimensions, for typical guidance problems related to EAGLE and in general for the powered descent landing. Those maneuvers occur in three dimensions, where one of the dimensions is affected by gravity.

5.4.3 Flight Estimation for the Powered Movement

For each maneuver in the 3-D case of the powered descent landing or general engine powered movements, the trajectory is split into a two-dimensional maneuver in the $r_2 - r_3$ plane and a one-dimensional vertical maneuver along the gravity affected r_1 axis. The actual difference in both maneuvers is the acting acceleration on the vehicle: Consider a movement along the r_1 axis, i.e. an altitude changing maneuver. In this case, the accelerations a_1 and a_2 defined in equation 5.1 for the considered time intervals of the movement are given by the sum of the maximum achievable acceleration $\frac{\bar{T}}{m}$ derived from the maximum achievable thrust $\bar{T}$ with a constant vehicle mass of m, and the gravitational acceleration g. Depending on the segment of this one-dimensional movement, the thrust is pointing downwards or upwards for acceleration or braking. Applying the flight time algorithm 1 in order to find the optimal time of flight in this dimension results in the time t_{ver}.
For the 2-dimensional movement within the $r_2 - r_3$ plane, the maximum acceleration along each of these axis is limited by the maximum thrust pointing angle. Let $T_{max,2} = T_{max,3}$ be the maximum achievable thrust along the second or third dimension, then for a maximum deviation of θ_{max} from the r_1 axis, these yield

$$T_{max,hor} = T_{max,2} = T_{max,3} = \bar{T} \cdot \sin\left(\theta_{max}\right). \tag{5.11}$$

Therefore, the absolute values a_1 and a_2 are given as $\pm\frac{T_{max,hor}}{m}$. The optimal flight time t_{hor} in the horizontal case is then calculated as in the 2-dimensional case described with the flight time algorithm 2.
The combination of both, t_{hor} and t_{ver}, is performed in analogy to the two-dimensional case and results in the estimated flight time t_f for the 3-D powered movement:

$$t_f = \max\left[\sqrt{t_{hor}^2 + t_{ver}^2}, \max\left[t_{hor}, t_{ver}\right] \cdot 1.2 + 2.0\right] \tag{5.12}$$

The achievement of this flight estimation is that we obtain a rough estimate for the fuel-optimal flight time, which can be used as an input of the trajectory optimization described in section 4.3. Since the flight time estimation also includes different initial velocities and is applicable in three dimensions, the guidance function is not only limited to the powered descent landing, but allows for on-board trajectory calculations related to any maneuver that is driven by the engine. However, the flight time estimation presented here still might underestimate the fuel-optimal flight time, which is considered in the following, final guidance function.

5.4.4 Real-time Applicable Guidance Function

The developed guidance function consists of the trajectory optimization algorithm displayed in figure 4.1 extended by the time of flight estimation as shown in figure 5.3. It is self consistent, meaning that it only requires the initial and final states in order to compute an entire trajectory. This trajectory is then optimal, or close to optimal, depending on the accuracy of the flight time estimation. However, the estimated flight time can be shorter than the actual fuel-optimal flight time, which results in potentially infeasible problems that cannot be solved by ECOS. In such a case, ECOS prints a certain flag allowing for detection of the infeasible problems after one run of the transcription and ECOS, which are summarized as "Optimization" in figure 5.3.
An ad-hoc solution for this problem is to apply the guidance function once for the initial position and velocity, and to increase the flight time iteratively in the case of a detected infeasibility until the problem becomes feasible. The trajectory optimization is run in each

iteration step until an optimal solution is found for a feasible problem. In the proposed guidance function, the increment of the flight time is set to be 3 seconds. On the one hand this is a tradeoff between computation time and accuracy. If the flight time estimation is close, but below the fuel-optimal flight time, the increment should not result in a new flight time that is far above the fuel-optimal solution. In contrast, if the estimation is far below the fuel-optimal flight time, the increment of the flight time should result in a feasible problem as quick as possible in order to reduce the number of optimization processes.

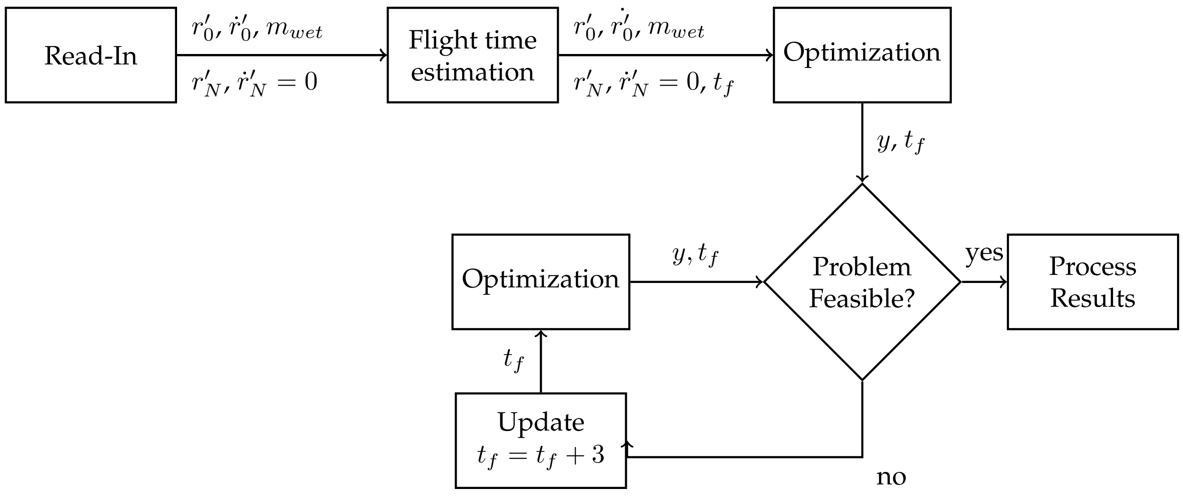

FIGURE 5.3: Flow diagram of the guidance function

The finally achieved guidance function is independent, as it only requires the initial and final states as inputs, and is robust in the sense that it is expected to always provide a solution after a limited number of iteration steps. In addition, these solutions are either optimal or close to optimal, where the latter is true for cases in which the t_f that finally provides a feasible problem is larger than the fuel optimal flight time. The guidance function is subject to in-depth test procedures as described in the following chapter.

6 Simulation and Tests of Guidance Function

6.1 Simulation Setup

To indicate the quality of the solutions provided by the guidance function with respect to robustness, computation time and optimality, it is embedded into two simulation environments, called simulation cases in the following. The purpose of these cases is mainly to test the behaviour of the guidance function for different initial positions on the one hand, and on the other hand for varying final positions. A combination of both in one simulation would require too many computation ressources. In both cases, the guidance function is fed with different sets of initial and final conditions and then the results are analyzed with a reference solution. The entire simulation is set in C-code for quick computation. In order to simulate typical EAGLE maneuvers with the guidance function, the following parameters describing the vehicle's properties are applied in all simulation sets:

Property	Symbol	Definition
Total Mass	m_{wet}	28.85 kg
Number of Thrusters	n	1
Specific Impulse	I_{sp}	2335 s
Cant Angle	ϕ	$0°$
Total Thrust	$\bar{T}$	392 N
Lower Thrust Boundary	T_1	$T_1 = 0.003 \cdot \bar{T}$
Upper Thrust Boundary	T_2	$T_2 = 0.9 \cdot \bar{T}$
Glide Slope Angle	γ_{max}	$4°$
Maximum Thrust Angle	θ_{max}	$5°$

TABLE 6.1: Physical parameters for EAGLE-Maneuver simulations

In addition, the gravity vector is given by $g = \begin{bmatrix}-9.807\ \frac{\mathrm{m}}{\mathrm{s}} & 0 & 0\end{bmatrix}^{\mathrm{T}}$.
The first simulation case considers typical powered descent landing problems. The goal is to start from different initial positions and velocities, and reach the final position $r'_N = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$ with a final velocity of $v'_N = \begin{bmatrix}0 & 0 & 0\end{bmatrix}$. For the initial conditions, the initial position vector is varied in its second and third component, hence all maneuvers start at the same altitude and the corresponding $r_2 - r_3$ plane is covered with a grid of different initial positions. The second component of the initial position $r'_{2,0}$ is varied between ± 50 m with a step size of $\Delta r'_{2,0} = 10$ m. For each of these points, the third component $r'_{3,0}$ is also varied in the same way, resulting in a grid of $11 \cdot 11 = 121$ different initial positions, while the altitude is fixed to $r'_{1,0} = 50$ m. For each of these initial positions, the simulation is run with a scan through different initial velocities, i.e. every component of the initial velocity is scanned between $\pm 7.5\ \frac{\mathrm{m}}{\mathrm{s}}$ in steps of $2.5\ \frac{\mathrm{m}}{\mathrm{s}}$. This results in a total set of $N_{sim} = 121 \cdot 7^3 = 41503$ different trajectories that are considered in this first simulation.

Simulation Case	Fixed	Varied Elements	Step size
1	$r'_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ m	$r'_0 = \begin{pmatrix} 0 \\ [-50, 50] \\ [-50, 50] \end{pmatrix}$ m	10m
	$v'_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \frac{\text{m}}{\text{s}}$	$v'_0 = \begin{pmatrix} [-7.5, 7.5] \\ [-7.5, 7.5] \\ [-7.5, 7.5] \end{pmatrix} \frac{\text{m}}{\text{s}}$	$2.5\frac{\text{m}}{\text{s}}$
2	$r'_0 = \begin{pmatrix} 50 \\ 0 \\ 50 \end{pmatrix}$ m	$r'_N = \begin{pmatrix} [0, 100] \\ 0 \\ [0, 100] \end{pmatrix}$ m	10 m
	$v'_N = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \frac{\text{m}}{\text{s}}$	$v'_0 = \begin{pmatrix} [-7.5, 7.5] \\ [-7.5, 7.5] \\ [-7.5, 7.5] \end{pmatrix} \frac{\text{m}}{\text{s}}$	$2.5\ \frac{\text{m}}{\text{s}}$

TABLE 6.2: Summary of scan intervals in both simulation cases

The second simulation case is built up in a similar way. In contrast to the first simulation case, the initial position is now fixed to be $r'_0 = \begin{bmatrix} 50 \text{ m} & 0 & 50 \text{ m} \end{bmatrix}$, while the final position vector is scanned. The scan covers different final altitudes from $r'_{1,N} = 0$ m to $r'_{1,N} = 100$ m separated by 10 m, such that it includes up and down movements. For each of these altitudes the third component of the final position vector is also varied from $r'_{3,N} = 0$ m to $r'_{3,N} = 100$ m with steps of 10 m, which again results in 121 different target positions to be tested. The initial velocity is varied in the same way as in the first simulation case, such that the second simulation also covers 41503 different sets. In addition, the glide slope constraint is turned off in this second simulation case, since a glide slope is not well defined for upward movements. Due to the chosen scanning intervals, the second simulation might appear to cover only two dimensional movements, since the second component of the final position vector is not varied. However, the second component of the initial velocity is varied and hence 3-D movements are covered in this simulation case, as well.

Both simulation cases are summarized in table 6.2, where the brackets within the vectors in the column for the varied parameters indicate the scan interval of the corresponding component, and the step size column provides the information on how much the individual components are changed in each step.

The objective of these simulations is to study on the on hand the quality of the flight time estimation itself, and on the other hand the related performance of the guidance function regarding fuel optimality. For each of the individual simulation sets the trajectory, corresponding fuel consumption f_g and flight times t_g obtained from the guidance function are saved, in addition to the value of the estimated flight time t_e computed by the flight time estimation block of the guidance function. If the flight time estimation t_e immediately results in a feasible problem, then $t_g = t_e$.

As a reference, a sub-simulation is run after the guidance results have been obtained. This sub-simulation uses the flight duration t_g from the guidance function, sets an interval of test times $t_t \in [1, 1.5 \cdot t_g]$ and applies the trajectory optimization to the simulation set for each of these test times. In this way, the true optimal trajectory and reference optimal flight time t_o is determined as the trajectory, for which the problem is feasible and the fuel consumption f_o is minimal. The results of the guidance function are then compared to those of this sub-simulation for each of the simulation sets and saved for further analysis. Moreover, the number of nodes is fixed to $N = 30$ for all simulations in order to obtain comparable results, especially for the comparison between the guidance function results and the reference simulation.

6.2 Analysis Procdure

The analysis of the data received from these simulations is processed in an identical way for both simulation cases following the objectives of the simulation.
As an indicator of the flight time estimation's quality, the estimated flight times t_e are compared to the optimal flight times t_o via $\Delta t_e = t_e - t_o$. The same calculation is performed for the comparison of the flight time obtained from the guidance function t_g and the optimal flight time, resulting in the deviation Δt_g. While these differences analyse the absolute errors of the flight time estimation and the guidance function, they do not include information on how large this overestimation is relative to the actual optimal flight time. As an example, a certain absolute overestimation has a higher impact for a short optimal flight time than on longer optimal flight times. The relative deviation

$$h_{tg} = \frac{|\Delta t_g|}{t_o} \tag{6.1}$$

is therefore introduced, providing more detailed information on the actual impact of the overestimations. In addition, the ratio between the fuel consumption f_g of the trajectory computed by the guidance function and the optimal fuel consumption f_o is computed resulting in the relative fuel deviation

$$h_f = \frac{f_g}{f_o} \tag{6.2}$$

to illustrate the actual performance of the guidance function regarding optimality. The analysis also provides the distribution of simulation results with respect to several parameters, like run-time of the guidance function and iteration steps.

6.3 Results and Interpretation

To obtain an overview of the typical shape of the computed trajectories, a plot is generated with 30 different trajectories from the simulation case 1 displayed in figure 6.1. The plot only considers trajectories for a vanishing initial velocity, and therefore the individual trajectories lie within a plane. One can observe the symmetrical grid of initial positions at a fixed altitude. All trajectories reach the final positition at the origin of the coordinate system, and the expected conic structure of the set of trajectories towards the end of the maneuver is clearly visible. Almost all of the plotted trajectories show an increasing altitude at the beginning of the maneuver, related to two effects. On the one hand it is due to the fixed flight time, which is put into the optimization algorithm by the guidance function. If this input overestimates the fuel-optimal flight time, then it might be more efficient to have a strong first burn, resulting in an increase in altitude, with a following acceleration of the spacecraft towards the ground and a final braking of the vehicle, in order to reach the vanishing velocity at exactly the desired fixed flight time. As a second reason, the vehicle needs to accelerate towards the origin of the $r_2 - r_3$ if it has a vanishing initial velocity. Due to the limited thrust pointing, this also implies a thrust component accelerating the spacecraft upwards, increasing the altitude.

Figure 6.2 displays the different distributions obtained by the simulation case 1,where
$\tilde{n} = \frac{N_{set}}{N_{sim}}$ is the normalized number of sets N_{set} for which the given parameter has been obtained in the simulation case. The top left plot shows in how many of the simulation sets the flight time estimation underestimates the fuel-optimal flight time marked with red bars. As visible in this distribution, the flight time estimation block tends to overestimate the optimal flight time. In total, an underestimation occurs in 35% of all simulation sets, and therefore

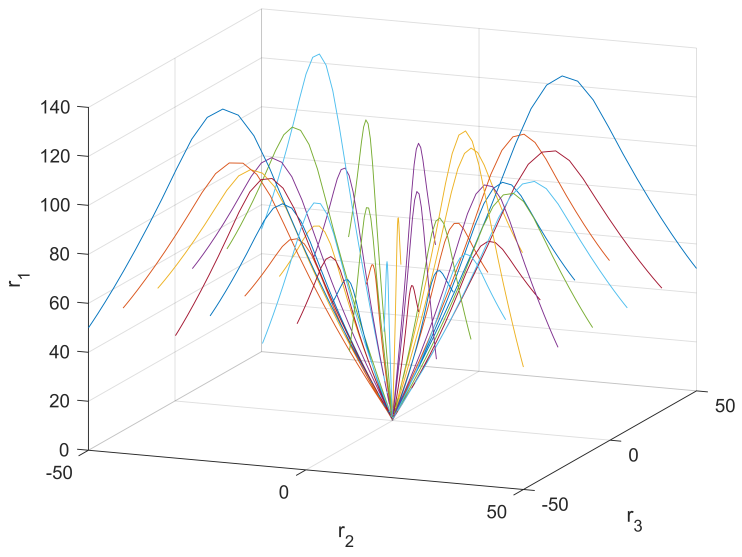

FIGURE 6.1: Plot of 30 extracted trajectories produced by the guidance function for varied initial position

results in the equivalent amount of infeasible problems. This is solved by the iterative guidance function, as proven by the distribution in the top right plot. This distribution implies that $t_g \geq t_o$ for all simulation sets, which was the initial goal of the iterative guidance function. Note, that in contrast to the distribution of Δt_e, the peak at $\Delta t_g = 2$ s dominates now. This is due to the step size of $\Delta t = 3$ s in the iterative guidance function. All sets, for which the flight time is underestimated, result in a feasible problem after a certain amount of time increments. Since the increment is set to 3 seconds, all of these previously infeasible problems will then contribute to $\Delta t_g = 0$ s, 1 s or 2 s as soon as they become feasible, reshaping the distribution of Δt_g towards the smaller over estimations.
Related to this, the bottom right plot provides the distribution of iteration steps, where one stands for the case that only one calculation step has been required, and hence the flight time estimation resulted immediately in a feasible problem. If the estimated flight time had to be increased once, then this implies two computation steps etc. In total, the cases of more than one computation step appear in roughly 35% of all simulation sets, which is consistent with the findings that the flight time estimation underestimated the optimal flight time in 35% of all sets.
The lower left plot is of special interest regarding computing time, revealing that the guidance function provides a solution within 0.08 seconds on a regular Desktop Computer for almost all simulation sets. Taking into account the previously found factor 10 for the computation time on EAGLE's on-board computer, this implies that the guidance function provides a reference trajectory within one second in any case. Moreover, a detailed analysis revealed that cases with computation times more than 0.05 s, and also with a higher number of computation steps, are related to high initial velocities with magnitudes above $5\ \frac{\text{m}}{\text{s}}$, which are not expected for the first EAGLE maneuvers in which this prototype guidance function will be applied. Therefore, the guidance function is indeed on-board feasible, as it provides solutions in any case in much less than a second.

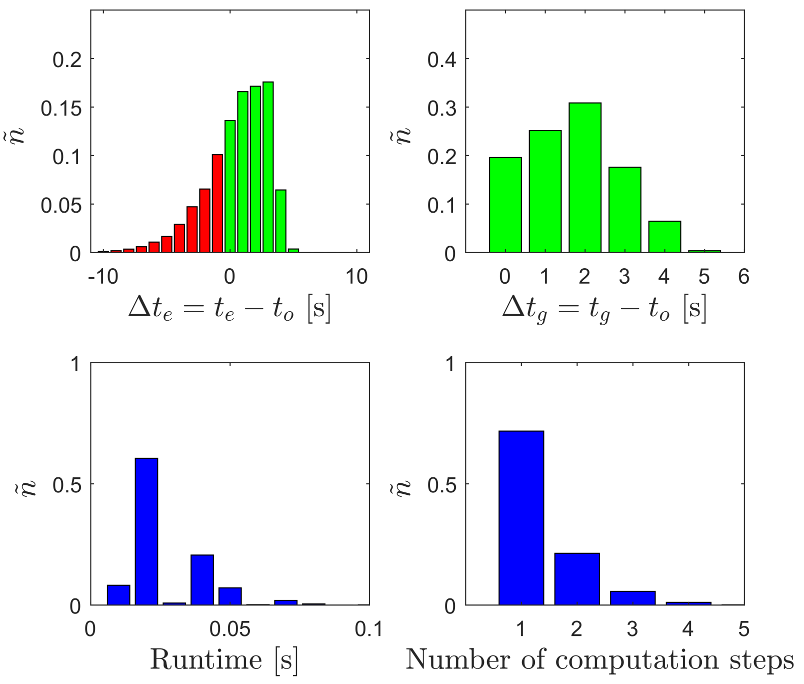

FIGURE 6.2: Distributions for simulation case 1

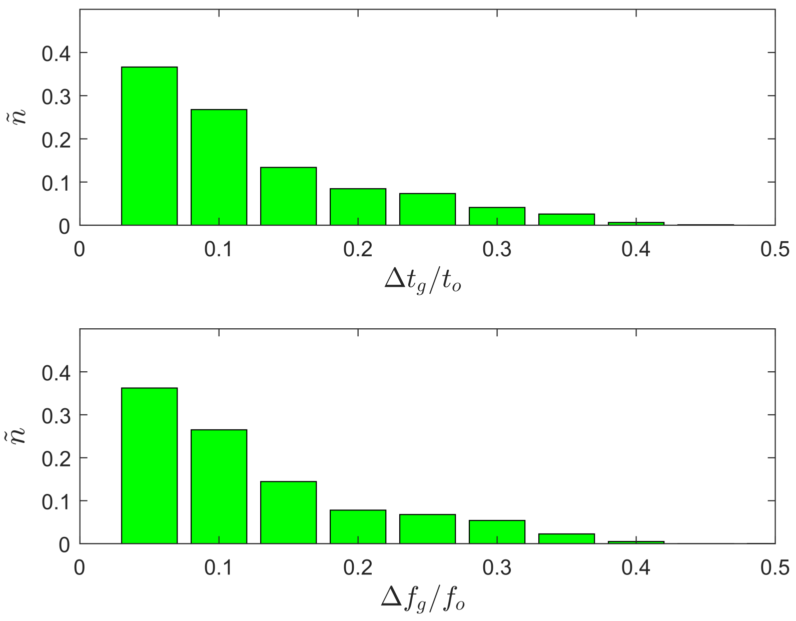

FIGURE 6.3: Distribution of relative deviations for simulation case 1

The top plots in figure 6.2 are distributions for absolute values. However, they do not provide information about the actual overestimation relative to the optimal flight time. An overestimation of $\Delta t_g = 2$ seconds has less effect if it is related to a larger minimum flight time. Therefore, the top plot of figure 6.3 plots the distribution of the relative time deviations for the guidance function $\frac{\Delta t_g}{t_0}$. In 80% of all simulation sets, the trajectories are calculated by the guidance function for flight times that overestimate the optimal flight time by less than 20%. The same holds for the relative fuel consumption, where the 80% of the trajectories computed by the guidance function require less than 20% more fuel than the corresponding optimal trajectories.

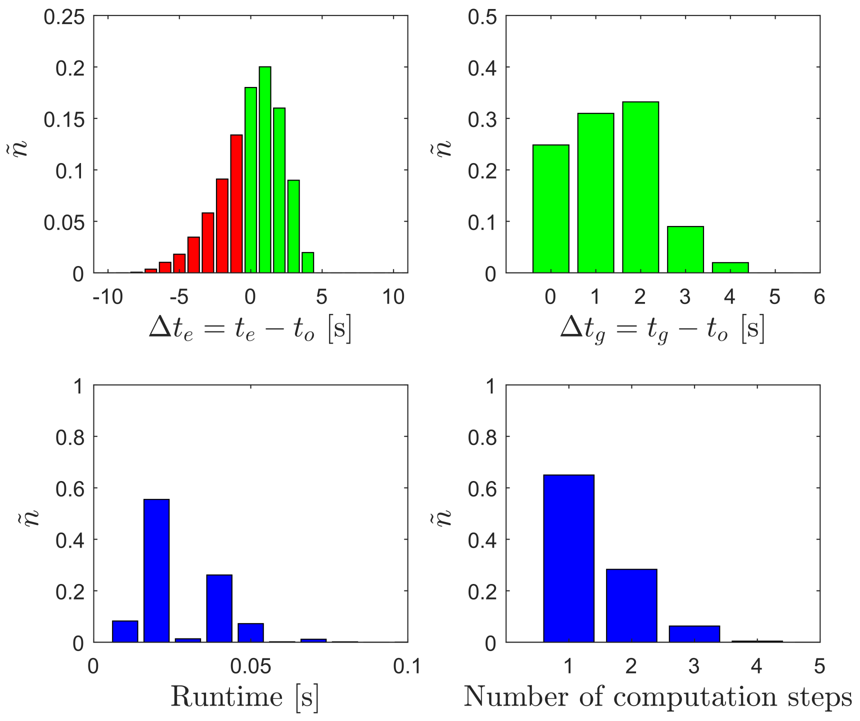

FIGURE 6.4: Distributions for simulation case 2

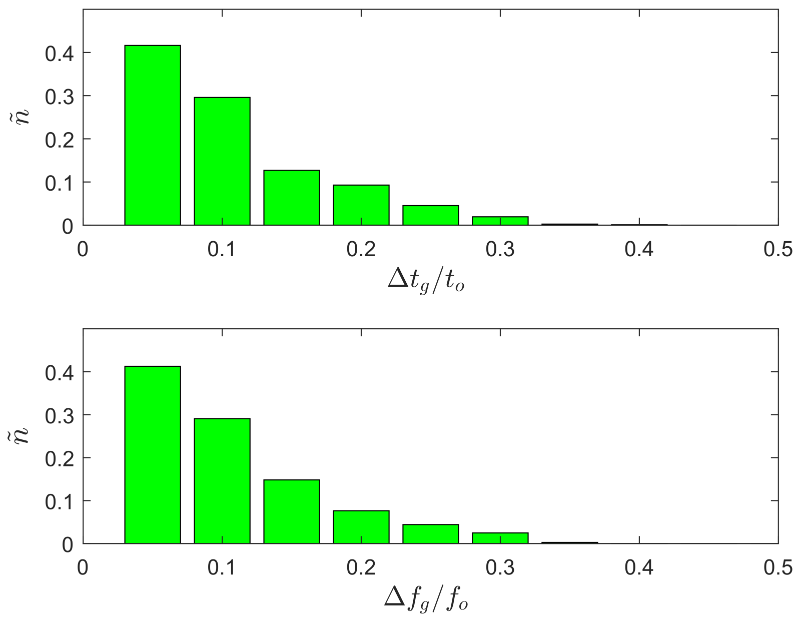

FIGURE 6.5: Distribution of relative deviations for simulation case 2

Compared to this first simulation case, the second simulation case reveals simular results in figure 6.4 and 6.5 . There are two main differences visible between those two cases. On the one hand, the flight time estimation block tends to underestimate the optimal flight time in more simulation sets than in the first simulation case. However, one cannot conclude that this is related only to the up-ward movements included in this simulation case, since the underestimations might also be related to downward movements targeting at a final point close to the initial one. Even in these maneuvers, the guidance function provides a result in less than 0.05 s of runtime in most of the simulation sets.

7 Conclusion and Perspective

In this thesis, an on-board applicable trajectory optimization algorithm for powered descent landing maneuvers has been developed based on convex optimization and the "loss-less" convexification of the powered descent landing method by [1]. The first contribution of this thesis is a computation chain, which transcribes the corresponding convexified optimal control problem into a non-linear program tailored to the needs of the embedded conic solver ECOS, which is used for obtaining the solutions of the problem. The framework developed here has been verified with an example Matlab simulation of Mars pinpoint landing, for which the results are in absolute agreement with the findings of [1].
The second contribution is a guidance function for on-board usage on EAGLE based on this trajectory optimization algorithm, which also has been implemented in C-code and a Simulink environment. In addition, the algorithm had to be extended by a preceding estimation block for estimating the fuel-optimal flight time and an iterative loop for avoiding infeasible problem formulations resulting from this estimation.
This guidance function has been tested with a set of 41503 different powered descent landing maneuvers and reveals that all problems are solved in less than 80 ms on a regular Desk-top computer and that the guidance function provides trajectories which require less than 20% more fuel than the corresponding optimal trajectories. First on-board tests of this guidance function on EAGLE proved that the guidance function is indeed on-board applicable and the corresponding computations require 10 times longer than on a Desktop computer. The guidance function is therefore expected to provide close-to-optimal trajectories within far less than a second on EAGLE, making it a robust, fast and almost optimal tool for real-time trajectory optimization on-board of EAGLE.

The guidance function and trajectory optimization framework developed in this thesis sets a base of on-board trajectory optimization algorithms, especially applied for EAGLE. Thanks to the great adaptivity of the algorithm developed in section 4.3, one can easily add further constraints, assuming that they are available in a convex form. This means that the considered optimal control problem can be improved and become more realistic.
Besides that, the developed guidance function is a first step towards a guidance function in the control system of EAGLE that produces optimal reference trajectories in real-time. Possible improvements of the function itself would consist of an in-depth investigation of possible estimation methods for the optimal flight time, which on the one hand would improve the quality in terms of fuel-usage, but on the other hand also the robustness and computation time. Also, adapted formulations of the considered optimal control problem are currently subject to further investigations and have resulted in promising results besides the work presented in this thesis.
A future major step is the entire implementation of the guidance function within the control system of EAGLE. This has already been partially accomplished, as well. Within the next months, the guidance function will be tested in free flights of EAGLE after an entirely successful implementation.
The developed trajectory optimization algorithm is not only applicable to EAGLE, but provides a general tool for real-time trajectory optimization. As an example, future reusable rockets require fuel-saving and precise trajectory calculations in real-time throughout the whole flight time. The method presented in this thesis can potentially be applied for any vehicle requiring fuel saving trajectories. As shown in this project, a reliable and robust real-time applicable guidance function can be developed with open-source available tools, making the solution cheap and flexible.

A Appendix

A.1 Proof for quadratic constraints as SOCP

The following proof underlines the connection between quadratic constraints given in the form of

$$x^T H x + p^T x + q = x^T F^T F x + p^T x + q \leq 0 \tag{A.1}$$

and the second order cone formulation

$$\left\| \begin{matrix} \frac{1+p^T x+q}{2} \\ Fx \end{matrix} \right\|_2 \leq \frac{1 - p^T x - q}{2} \tag{A.2}$$

which will be used for implementing quadratic constraints as second order cone constraints. A similar, but unreferenced proof has been found on [11] after the following proof had been developed. Substituting $y = p^T x + q$ one obtains:

$$\begin{aligned}
\left\| \begin{matrix} \frac{1+p^T x+q}{2} \\ Fx \end{matrix} \right\|_2 &\leq \frac{1 - p^T x - q}{2} \\
\left\| \begin{matrix} \frac{1+y}{2} \\ Fx \end{matrix} \right\|_2 &\leq \frac{1 - y}{2} \\
\sqrt{\left(\frac{1+y}{2}\right)^2 + (Fx)^2} &\leq \frac{1 - y}{2} \\
\sqrt{\frac{1}{4}(1+y)^2 + x^T F^T F x} &\leq \frac{1 - y}{2} \\
\sqrt{(1+y)^2 + 4x^T F^T F x} &\leq 1 - y \\
(1+y)^2 + 4x^T F^T F x &\leq (1-y)^2 \\
(1+y)^2 - (1-y)^2 + 4x^T F^T F x &\leq 0 \\
4y + 4x^T F^T F x &\leq 0 \\
x^T F^T F x + y &\leq 0 \\
x^T F^T F x + p^T x + q &\leq 0
\end{aligned}$$

A.2 Column-Compressed Storage of Matrices

To illustrate the structure of column-compressed storage matrices, we make use of the following example from [4]: Assume a matrix A is given by

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

Then the column-compressed storage version is represented by the arrays:

$$\text{Value Array: } A_{pr} = \begin{bmatrix} 10 & 3 & 3 & 9 & 7 & 8 & 5 & 8 & 8 & \ldots & 9 & 2 & 3 & 13 & -1 \end{bmatrix}$$

$$\text{Row Array: } A_{ir} = \begin{bmatrix} 1 & 2 & 4 & 2 & 3 & 5 & 6 & 3 & 4 & \ldots & 5 & 6 & 2 & 5 & 6 \end{bmatrix}$$

$$\text{Column Array: } A_{jr} = \begin{bmatrix} 1 & 4 & 8 & 10 & 13 & 17 & 20 \end{bmatrix}$$

Note that the last element of the column array is needed to identify the last non-zero element of the matrix A stating that the number of non-zero elements in matrix A is 19.

A.3 Convex Simulator - Parameters

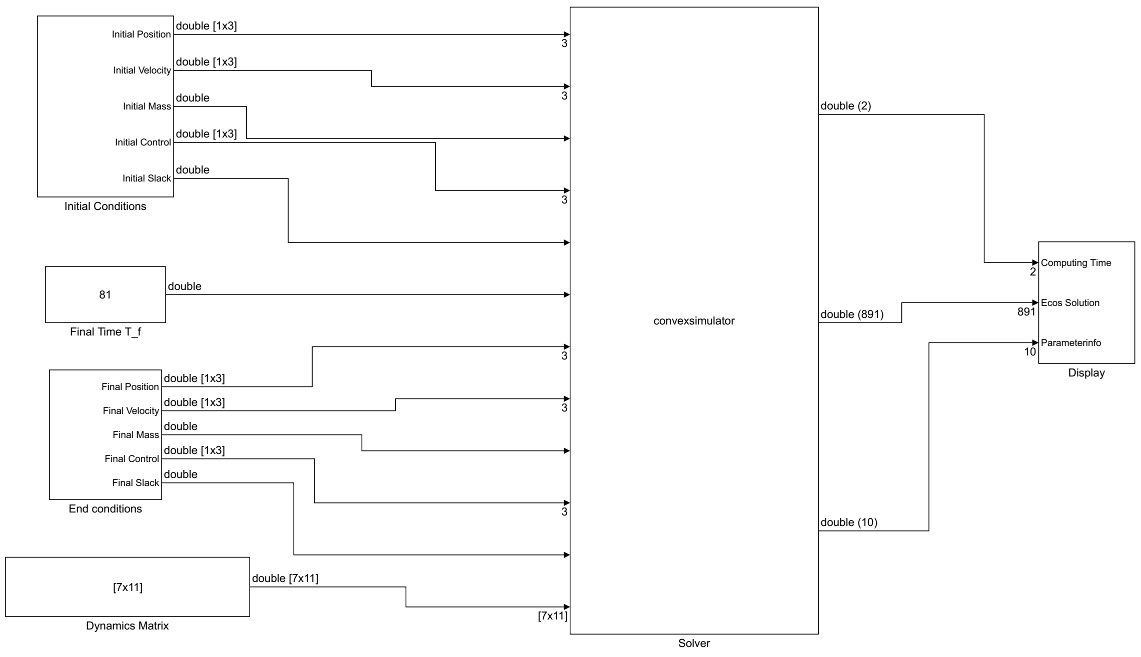

FIGURE A.1: Simulink-Model of the simulation framework

The following provides a list of current parameters tunable via the mask on the S-function:

Parameter Number	Description
1	Number of nodes for discretization
2	Dimension of Position vector x (or in general of state vector)
3	Dimension of Mass (or additional state variables)
4	Dimension of control vector u (or in general of control vector)
5	Dimension of slack variable s (or in general slack vector and additional control variables)
6	Start Time t_0 [s]
7	Number of thrusters n
8	Maximum thrust level of thrusters T [kN]
9	Lower Thrust Boundary (as fraction of T)
10	Upper Thrust Boundary (as fraction of T)
11	Chant angle of thrusters ϕ [DEG]
12	Specific Impulse I_{sp} [s^{-1}]
13	Magnitude of Earth's gravitational acceleration (used for calculating mass flow α from I_{sp})
14	Gravitational acceleration on spacecraft on/off
15	Magnitude of gravitational acceleration vector [$\frac{\mathrm{m}}{\mathrm{s}^2}$] (negative if in negative direction of axis stated in 16)
16	Axis parallel to gravity 1,2, or 3
17	Initial condition for position on/off
18	Initial condition for velocity on/off
19	Initial condition for mass on/off
20	Initial condition for control (u) on/off
21	Initial condition for slack variable on/off
22	Final condition for position on/off
23	Final condition for velocity on/off
24	Final condition for mass on/off
25	Final condition for control (u) on/off
26	Final condition for slack variable on/off
27	Upper constraint on slack variable (σ) on/off
28	Lower constraint on slack variable (σ) on/off
29	Upper constraint on mass variable (z) on/off
30	Lower constraint on mass variable (z) on/off
31	Slack constraint on control ($\|u\| \leq s$) on/off
32	Thrust pointing constraint at end on/off
33	Thrust pointing direction axis at end $\pm 1, \pm 2$ or ± 3
34	Glide slope constraint at end on/off
35	Glide slope angle [DEG]
36	Thrust pointing constraint on/off
37	Maximum deviation of thrust vector to reference axis

TABLE A.1: Parameters for the solver block in the simulation model

Bibliography

[1] B. Açikmese and S. R. Ploen. "Convex programming approach to powered descent guidance for mars landing". In: *Journal of Guidance, Control, and Dynamics* 30.5 (2007), pp. 1353–1366.

[2] Behcet Açikmese et al. "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem". In: *IEEE Transactions on Control Systems Technology* 21.6 (Nov. 2013).

[3] San Martin et. al. "In-flight experience of the Mars Science Laboratory Guidance, Navigation, and Control system for Entry, Descent, and Landing". In: *CEAS Space Journal* (2015).

[4] R. Barrett et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. Philadelphia, PA: SIAM, 1994.

[5] John T. Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics Philadelphia, 2010, pp. 130–131.

[6] L. Blackmore, B. Açikmese, and D.P. Scharf. "Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization". In: *Journal of Guidance, Control and Dynamics* 33.4 (2010).

[7] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004, pp. 36–37.

[8] Timothy Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2006.

[9] A. Domahidi, E. Chu, and S. Boyd. "ECOS: An SOCP solver for embedded systems". In: *European Control Conference (ECC)*. 2013, pp. 3071–3076.

[10] M. Dumke. "GNC 2017 - 10th International ESA Conference on Guidance, Navigation and Control Systems - EAGLE- Environment for Autonomous GNC Landing Experiments". In: 2017.

[11] wyer33. *Converting from Quadratic to Second Order Cone optimization problem*. Mathematics Stack Exchange. Version: 2016-10-04. URL: `https://math.stackexchange.com/q/1939169`.