Guidance Optimization for Mars Pinpoint Landing with Optimal Trigger and Re-optimization

Tiago Medeiros Amaral

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor: Prof. Paulo Jorge Soares Gil

Examination Committee
Chairperson: Prof. Filipe Szolnoky Ramos Pinto Cunha
Supervisor: Prof. Paulo Jorge Soares Gil
Member of the Committee: Eng. Nuno Tiago Salavessa Cardoso Hormigo Vicente

May 2019
Dedicated to my parents.

I don’t say this enough, but I feel incredibly lucky to be your son. You two really are wonderful people.

Thank you for everything.
Acknowledgments

First, I want to thank my thesis supervisor, Professor Paulo Gil, for the support and everything he taught me. It was a pleasure working with him.

Second, I want to thank everyone at Spin.Works for the opportunity and support during this work. Spin.Works provided useful data from a Mars landing mission and an atmosphere model of Mars, for which I am grateful. A special word to Engineer Tiago Hormigo, for being helpful in various situations. I feel that I learned a lot with him.

To my two friends and housemates, Manuel Bonito and João Marques, for helping me in the writing process. Finally, I also want to thank my parents and sisters for the unconditional love and support.
Resumo

Aterragens de precisão serão necessárias em futuras missões a Marte e serão cruciais em missões que aterrem em locais difíceis, como crateras ou vales. Para além disso, em futuras missões tripuladas, é necessário que a nave aterre próximo de carga previamente enviada. Neste trabalho é implementada e testada uma função de guidance para aterragens de precisão em Marte, baseada num método desenvolvido pela NASA. Este método utiliza optimização convexa para criar uma trajectória ótima, i.e, com o menor consumo de combustível possível, que guia a nave de um ponto A até à aterragem num ponto B. Neste trabalho foi desenvolvido um método para determinar qual a melhor posição para iniciar a trajectória propulsiva, um ciclo de re-optimização da trajectória e um método simples para ter em conta o efeito da resistência aerodinâmica. Aterragens em Marte foram simuladas para 1500 diferentes estados iniciais e condições atmosféricas. O algoritmo permitiu aterrar com precisão de meio metro e poupar em média 30% de combustível em comparação com um algoritmo do tipo Apollo. Foi determinado, através de simulações, que os métodos introduzidos são viáveis e robustos. A função desenvolvida mostrou-se capaz de, autonomamente e em tempo real, gera trajectórias ótimas para aterrar com precisão.

Palavras-chave: Aterragem de Precisão, Optimização Convexa, Aterragem Propulsiva, Re-optimização, Iniciação Óptima dos Motores
Abstract

Precise landings will be required for future Mars missions and will be crucial to missions landing on challenging places such as craters or valleys. Also, in future manned missions, it is important to land close to cargo that was previously sent. In this work, a guidance function is implemented and tested for precise landings on Mars, based on a method developed by NASA. In this method, convex optimization is used to create fuel-optimal trajectories that guide the spacecraft from a point A to land on a point B. In this work, it was developed a method for finding the optimal position to trigger the powered descent, a loop for re-optimizing the trajectory and a simple method to take the drag effect into account. A Mars landing scenario was simulated with 1500 different initial states and atmospheric conditions. The algorithm allowed landing with half-meter precision and to save on average 30% of fuel in comparison with an enhanced Apollo-like guidance function. It was determined with simulations that the introduced methods are viable and robust. The developed function has shown to be able of, autonomously and in real-time, generate fuel-optimal trajectories to land on target.

Keywords: Pinpoint Landing, Powered Descent Guidance, Optimal Trigger Point, Convex Optimization, Re-optimization
# Contents

Acknowledgments ...................................................... v
Resumo .................................................................. vii
Abstract ................................................................ ix
Nomenclature ........................................................... xv
Glossary ................................................................. xvii

1 Introduction .............................................................. 1
  1.1 Precise Landing on Mars ............................................. 1
  1.2 Methods for Landing on Target ................................. 2
  1.3 Guidance Algorithm for Mars Pinpoint Landing .......... 3

2 Background ............................................................... 5
  2.1 Convex Optimization ............................................... 5
    2.1.1 Advantages of Convex Optimization ...................... 6
    2.1.2 Solving a Problem With Convex Optimization ......... 7
  2.2 Second-Order Cone Programming .............................. 7

3 Pinpoint Landing Problem ............................................. 9
  3.1 Problem Formulation .............................................. 9
  3.2 Convexification Process .......................................... 11
    3.2.1 Convexification of Thrust Magnitude Constraint ...... 11
    3.2.2 Linearization of The Dynamics ............................ 12
  3.3 Discretization ...................................................... 15
  3.4 Minimum-Landing-Error Case ................................. 16
  3.5 Additional Constraints .......................................... 16
  3.6 Optimal Flight Time ............................................ 19

4 Solving the Pinpoint Landing Problem .......................... 21
  4.1 Solver selection .................................................. 21
  4.2 Adapting the Problem for ECOS ............................... 22
  4.3 Test and Validation .............................................. 23
  4.4 Time-Step Selection ............................................ 28
List of Figures

2.1 One convex set (left) and two non-convex sets [14]. ......................... 6
2.2 Example of a convex function. The line segment that connect any two points \((x, f(x))\) and \((y, f(y))\) lies above the graph of the function [14]. ......................... 6

3.1 Spacecraft model [13]. .................................................. 10
3.2 Surface fixed coordinate frame [13]. ........................................ 10
3.3 Visual illustration of thrust magnitude constraint convexification [13]. .... 12
3.4 Visual illustration of constraint (3.18). The white area represents the left part of the inequality and defines a convex set. The grey area represents the right part of the inequality and does not define a convex set. ......................... 13
3.5 Visual illustration of glideslope constraint [18]. ............................. 17
3.6 Visual illustration of thrust pointing constraint [18]. ......................... 18

4.1 Landing case without additional constraints. ............................... 25
4.2 Landing case without sub-surface flight constraint. ....................... 25
4.3 Landing case with glideslope constraint. .................................. 26
4.4 Landing case with vertical flight. ......................................... 26
4.5 Landing case with thrust-pointing constraint. .............................. 27
4.6 Minimum-landing-error case. ............................................. 27
4.7 Long flight scenario. \(r_0 = [1500, 0, 2000]^T\), \(r_0 = [-75, 0, 100]^T\), \(t = 78\). .................................................. 29
4.8 Scenario with medium time of flight. \(r_0 = [1500, -3500, 2000]^T\), \(r_0 = [-65, -30, -20]^T\), \(t = 66\). .................................................. 29
4.9 Vertical and short flight scenario. \(r_0 = [1500, 0, 0]^T\), \(r_0 = [-65, 0, 0]^T\), \(t = 31\). .................................................. 29

5.1 Flow diagram of the improved guidance algorithm. ....................... 33
5.2 Fuel consumption required to land starting the maneuver at different times of the unpowered descent flight. Time represents the duration of unpowered flight. Here, \(t = 0\) marks the beginning of the study, in a position known to be far from the optimal trigger position. The red section represent infeasible solutions. ........................................ 34
5.3 Search for optimal trigger time sequence. ................................ 35
5.4 Mean execution times of the optimization problem for different times-of-flight. ........................................ 36
5.5 Mars pinpoint landing example with \( r_0 = [-5540, 570, 6420]^T \) m, \( v_0 = [207, -25, -103]^T \) m/s, \( m = 346 \text{ kg} \) and \( t_f = 36 \text{ s} \). The fuel consumption is 34.6 kg.

5.6 Three-dimensional trajectory produced by the guidance function.

5.7 Thrust pointing constraint with and without correction.

5.8 Thrust profile with and without re-optimization.

6.1 Two major blocks of simulink model.

6.2 GNC block.

6.3 Dynamics block.

6.4 S-function block for trajectory calculations.

6.5 Plot of the 1537 trajectories.

6.6 Throttle level corrections in the re-optimization process.

6.7 Pitch and Yaw corrections in the re-optimization process.

6.8 Comparison of fuel consumption between the improved convex guidance algorithm, on the left, and an Apollo-like guidance algorithm on the right. Both guidance functions were capable of pinpoint landings.

6.9 Fuel savings as a result of the optimal trigger search.

6.10 Landing ellipse of the improved guidance function.

6.11 Final vertical position error.

6.12 Final lateral position errors.

6.13 Final vertical velocity error.

6.14 Execution times.

6.15 Optimally of the guidance function.
Nomenclature

Greek symbols

$\alpha$ Fuel consumption rate.

$\Gamma$ Slack variable.

$\gamma_{gs}$ Glideslope angle.

$\phi$ Cant angle.

$\theta$ Pointing angle.

$\rho_1$ Net thrust lower bound.

$\rho_2$ Net thrust upper bound.

Roman symbols

$e$ Net thrust direction.

$g$ Gravitational acceleration vector of Mars.

$g_e$ Gravitational acceleration of Earth.

$I_{sp}$ Specific impulse.

$m$ Spacecraft mass.

$m_{dry}$ Spacecraft dry mass.

$m_{wet}$ Spacecraft wet mass.

$n$ Number of thrusters.

$N$ Number of temporal nodes.

$r$ Position vector.

$T$ Thrust level.

$t$ Time.

$T_1$ Thrust level lower limit.
\( T_2 \)  Thrust level upper limit.

\( T_c \)  Total net thrust vector.

\( t_f \)  Time of flight.

\( z \)  Spacecraft mass logarithm.

**Subscripts**

0  Initial.

max  Maximum.

min  Minimum.

**Superscripts**

T  Transpose.
Glossary

CCS  Column Compressed Storage
CQP  Convex Quadratic Programming
ECOS Embedded Conic Solver
EDL  Entry, Descent and Landing
ESA  European Space Agency
GNC  Guidance, Navigation and Control
IPM  Interior-Point Methods
LP   Linear Programming
NASA National Aeronautics and Space Administration.
PDG  Powered Descent Guidance
SDP  Semidefinite Programming
SOCP Second-Order Cone Programming
Chapter 1

Introduction

1.1 Precise Landing on Mars

The next years will certainly be exciting for anyone interested in Mars exploration. With recent advances in rocket technology, such as the re-usability of launch vehicles and the cost reduction associated, not only Mars manned missions are closer than ever but also new ambitious missions are surging [1]. One important requirement for these future missions will be the ability to perform a pinpoint landing, which is defined as autonomously guiding a spacecraft to land in a chosen target with an accuracy smaller than 100 meters. This is crucial for missions that require landing in challenging places, such as craters or valleys [2]. Also, in future manned missions, it is important to land close to cargo that was previously sent [2, 3].

In a typical mission to the Mars surface, the spacecraft is first injected into a transfer orbit from Earth to Mars. When arriving at Mars, the process of Entry, Descent and Landing (EDL) starts, in order to decelerate the spacecraft from hypersonic velocities to near zero and soft land. The entry phase of EDL starts typically at an altitude of 125 km and ends at the deployment of the parachute [4]. In this phase, the spacecraft enters the Martian atmosphere and aerobrakes to supersonic speeds using a heat-shield or an aeroshell [5]. To initiate the descent phase, a single or a combination of parachutes are deployed, typically 10 km or less above Mars’ surface [4]. Due to the thin Martian atmosphere, the terminal velocity achieved with a parachute is significantly higher than in Earth’s atmosphere, typically around 50 m/s [4]. Therefore, Mars EDL usually finishes with a powered descent phase where rocket thrusters are used to slow down the spacecraft to the desired velocity at touchdown.

Due to the high unpredictability of Mars’ atmosphere, the spacecraft can accumulate large position and velocity errors during the atmospheric entry phase. Additionally, because of the winds in the parachute descent phase, the state of the spacecraft (position and velocity) gets even more dispersed and it can not be accurately estimated [5]. These position dispersions can be up to 8-10 km with respect to a prescribed target at the start of the powered descent phase [5, 6]. Thus, previous missions to the red planet didn’t have good landing precision. Mars exploration rovers had a landing accuracy of 35 km and later the Mars Science Laboratory, with rover Curiosity, had an accuracy of 10 km [4]. However, high
precision landing was not a requirement in these past missions.

Therefore, to achieve pinpoint landing in future missions, new onboard techniques must be used to correct the errors accumulated during entry and descent [7]. This calls for a powered descent guidance (PDG) algorithm capable of calculating a fuel-optimal trajectory that guides a spacecraft from an original state to the target. This trajectory must be calculated in real-time, because the location of the spacecraft at the start of the powered descent phase cannot be correctly predicted.

1.2 Methods for Landing on Target

Several approaches to solve the pinpoint landing problem have been proposed over the years. During the Apollo missions, an algorithm based on a quartic polynomial in time was used, that assumed linear variation of vertical acceleration, with quadratic variations in other axis but no optimal conditions [8]. In another technique, an energy-optimal algorithm was created that assumes flight over a flat planet and neglects aerodynamic forces [9]. These assumptions reduce the complexity of the problem and allow analytical solutions to be found. However, the formulation does not account for constraints on neither the minimum or maximum thrust magnitude of the engines. Without these constraints, a physically impossible solution could be produced [10].

Other approach proposed to solve the problem is by an approximation to a related optimal control problem that does not use a minimum fuel function as cost [11]. Legendre pseudospectral methods can also be used to find a numerical solution for the powered landing scenario [12]. There are also other methods that convert the infinite-dimension problem into a finite-dimensional optimization problem and then solve them with nonlinear programming methods, but this is not recommended for real-time applications because convergence is not always guaranteed [13].

Convex Optimization

For a pinpoint landing on Mars, the guidance function must be created in real-time. Therefore, it is essential to use an algorithm that is fast and with guaranteed convergence to the global optimum [13]. This leads to the interest of using convex optimization, more specifically second-order cone programming (SOCP), which is a subclass of convex optimization.

Convex optimization is a subclass of optimal control. It allows finding a solution that cannot be analytically solved, using reliable and efficient algorithms. It consists of minimizing a convex cost function, which is the optimization objective of the problem, subjected to convex constraints. Convex optimization guarantees that is possible to find the global-optimal solution, if there is one, and with guaranteed convergence [14]. SOCP problems can be solved in polynomial-time using available open-source interior-point methods algorithms, with any given stopping condition and level of accuracy. Additionally, current research with customized methods has improved computation time by orders of magnitude [15–17]. However, using convex optimization is not straightforward. The majority of problems in their original form are not convex and must be reformulated into the convex-form. This reformulation can be really challenging and is generally done by experts [14].
Convex optimization has proven to be a practical way to solve the pinpoint landing problem [2]. NASA developed an algorithm for pinpoint landing on Mars using convex optimization [13]. It was proven that the non-convex pinpoint landing problem can be reformulated as a convex optimization problem, more specifically as a second-order cone problem. Additionally, and more important, it was shown that this problem can be losslessly convexified, i.e., the problem can be converted into a convex optimization problem whose optimal solution is the same as the original nonconvex problem [18]. In this convex programming approach, a fuel-optimal trajectory and thrust profile are created to autonomously guide the spacecraft from an initial state (position and velocity) to the target and perform a soft landing.

This approach was first developed in 2005 and various enhancements have been made in the last decade [7, 18–22]. In 2008, the rotation of the planet was introduced in the dynamics and no subsurface flight between time samples was guaranteed [21]. A major improvement of the algorithm was the capability to handle the case when no feasible solution exists to reach the target. When that happens, the algorithm finds the closest reachable location to the target and then obtains the fuel-optimal trajectory to land on the new location [22]. In most Mars missions, it is important to limit the orientation of the spacecraft such that crucial sensors can point to the surface. Therefore, thrust pointing constraints were included and the required convexification of the enhancement problem was demonstrated [20].

It is worth mentioning that convex optimization is a key enabling factor of SpaceX Falcon 9 landings. SpaceX generates real-time flight code using convex optimization [2]. Lars Blackmore, one of the co-inventors of the convex programming approach to pinpoint landing, is now the principal rocket landing engineer at SpaceX.

1.3 Guidance Algorithm for Mars Pinpoint Landing

The objective of this work is to develop and implement a guidance algorithm for precise landing on Mars, based on NASA’s approach to pinpoint landing [7, 13, 18–24]. In this framework, some features were introduced to improve robustness and overall feasibility of the original guidance function. First, a search method is introduced to determine the optimal position to trigger the powered descent. Second, a periodic loop for re-optimizing the trajectory is included. This improves robustness since it is possible to efficiently correct dispersions that may occur because of drag or unexpected winds. Third, a procedure to take the drag effect into consideration is introduced.

The purpose of this algorithm is not to guide the spacecraft through the entire process of Mars entry, descent and landing. The algorithm is responsible for the last but critical phase, the powered landing. This guidance function is supposed to be called at an altitude of about 8 km. It will be responsible for deciding when to trigger the powered descent and generate trajectory references and thrust profiles for control systems.

The guidance algorithm was validated with simulations of pinpoint landings on Mars. The landing is tested for 1500 initial states, with dispersions of 5 km and different atmospheric conditions, based on possible initial states from an ESA mission [25]. The quality and feasibility of the improved guidance function was tested, based on optimality, robustness and computation time.
Chapter 2

Background

In this chapter, the theoretical background is presented to introduce convex optimization and explain its adaptation to the pinpoint landing problem. Convex optimization was selected because it guarantees convergence to the global optimum, which is essential for a real-time onboard computation of an optimal trajectory [13]. Additionally, it has been shown that the set of feasible initial states, i.e., the initial states for which it is possible to land on target, is more than twice larger when convex optimization is used than other traditional approaches [26].

Convex optimization is a subclass of mathematical optimization, or simply optimization. Mathematical optimization theory consists in solving a given problem with a certain optimal criteria. We formulate an optimization problem by presenting an objective cost function, which we want to minimize, and a set of constraints that the problem is subjected to. The problem can be written in its general formulation as

\[
\begin{align*}
\text{minimize} & \quad f_0(x) \\
\text{subject to} & \quad f_i(x) \leq 0 \quad i = 1, \ldots, n \\
& \quad f_j(x) = 0 \quad j = 1, \ldots, m.
\end{align*}
\]  

(2.1)

The vector \( x = (x_1, \ldots, x_n) \) is the optimization variable and the function \( f_0 \) is the cost function of the problem. The functions \( f_i : i = 1, \ldots, n \) and \( f_j : j = 1, \ldots, m \) represent the inequality and equality constraints functions of the problem, respectively. A solution (vector \( x \)) is called feasible if it satisfies all of the problem’s constraints. A solution is called optimal if it has the smallest cost value of the set of feasible solutions [14].

2.1 Convex Optimization

Only an overview of the convex optimization method is presented here since there is extensive information available [14, 27]. In a convex optimization problem, the cost function must be convex, all equality constraints must be linear and the inequality constraints have to define a convex set. A convex optimization problem can be represented in the same standard form as in Equation 2.1, where the vector \( x \) represents the optimization variables that must satisfy all constraints.
A convex set can be described as a region that, for every two points inside the region, the straight line that connects the pair of points is also within the set. Assuming a convex set $S$ and two points $x_1$ and $x_2$ inside the set, it is true that:

$$\theta x_1 + (1 - \theta)x_2 \in S, \quad \text{for } 0 \leq \theta \leq 1, \quad x_1, x_2 \in S. \quad (2.2)$$

Figure 2.1 illustrates some examples of convex and nonconvex sets. On the left, the hexagon, which includes its boundary, is convex. In the middle, the set is nonconvex because you can have two points of the set where the line that connects them is not contained in the set. On the right, the square does not include part of its boundary, therefore it is not a convex set.

A function $f$ is called convex, if its domain ($\text{dom} f$) defines a convex set and if for all $x, y \in \text{dom} f$ we have

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y), \quad \text{for } 0 \leq \theta \leq 1.$$ 

In other words, the line segment that connects any two points of a function lies above the function. A convex function is a function with positive curvature, which means that there is always a local minimum. In layman terms, convex optimization can be summarized as finding the optimal value of a function in a domain without holes or indentations [28].

2.1.1 Advantages of Convex Optimization

Convex problems are easier to solve because they do not require a good initial guess and because they ensure finding the global-optimal, if there is one, with guaranteed convergence [14, 28]. In convex optimization, when searching for an optimal point in a constrained problem, it is easier to ensure not generating infeasible solutions in a convex feasible region. This means that if there are two feasible
solutions, any solution in the line segment connecting them is also feasible, which helps local searches. A convex objective function ensures that a local optimum is also the global optimal. Without convexity, a local search algorithm can converge or get stuck at a sub-optimal solution. It is important to emphasize the importance of ensured convergence. Since Mars pinpoint landing requires a real-time onboard calculation of the optimal trajectory, it is essential to use an algorithm with guaranteed convergence to the global optimum.

2.1.2 Solving a Problem With Convex Optimization

Solving an optimization problem using convex optimization can be divided into two parts. First, if the original formulation of the problem is not convex, the problem has to be reformulated into the convex form. Second, a solver must be used to find the optimal solution. Solvers usually require that the problem is presented in a specific way. Therefore, the convex problem still has to be adapted to meet the requirements of the solver.

A vast number of nonconvex problems can be reformulated as convex problems via relaxations. Relaxation is the process of transforming constraints or even the structure of the problem in a way that all non-convexities are eliminated. Recognizing convex optimization problems, or those that can be transformed into convex optimization is challenging. In addition, the reformulation to the convex-form, if often not straightforward. Any constraint that may not form a convex set or any nonlinearity in the dynamics of the problem must be reformulated. To do that, several techniques can be used to transform nonconvex problems into convex optimization problems. The most common are changing variables and the introduction of slack variables to reformulate a non-convex constraint[14].

The most difficult part of solving a problem using convex optimization is to reformulate it to the convex form. This means that, if it is possible to formulate the problem as a convex optimization problem, then usually it is possible to solve it [14]. Usually, there is no analytical way to solve convex optimization problems, but there are effective numerical methods for solving them.

One approach to solve a convex problem is to manually convert the problem into a standard form of convex optimization subclass (LP, CQP, SOCP, or SDP) and then use an appropriate solver for the subclass chosen [27]. This approach allows the possibility to use existing solver software. There is also the possibility to use a modeling system that allows writing the problem in high-level language. The modeling language does the work of transforming the problem to the standard form of convex optimization and uses an internal solver to find a solution. This method is more user-friendly since there is no need to reformulate the problem. However, solving the problem with a modeling system is usually slower than using a solver directly [14].

2.2 Second-Order Cone Programming

Convex optimization has several sub-fields (quadratically constrained quadratic programming, semidefinite programming and others). In this work, the problem is formulated as a Second-Order Cone Pro-
gramming (SOCP) problem. This subclass is briefly described here, but more information can be found in [27].

Second-Order Cone Programming problems are convex optimization problems in which a linear function is minimized over the intersection of an affine linear manifold with the Cartesian product of second-order cones [27]. Linear programs, convex quadratic programs and quadratically constrained convex quadratic programs can all be formulated as SOCP problems, as can many other problems that do not fall into these three categories.

Second-order Cone problems are formulated as

\[
\begin{align*}
\text{minimize} & \quad c_0^T x \\
\text{subject to} & \quad A_0 x = b_0 \\
& \quad \| A_i x + b_i \|_2 \leq c_i^T x + d_i, \quad i = 1, \ldots, p,
\end{align*}
\]

where the vector \( x \) is the optimization variable and \( c \) is the vector that represents the linear cost function. Matrices \( A_0 \) and \( b_0 \) describe the \( m \) linear equations that the solution has to satisfy. The terms \( A_i, b_i, c_i \) and \( d_i \), with \( i = 1, \ldots, p \), represent the different conic constraints of the problem.

Second-order Cone Programming problems can be solved very efficiently using Interior-Point Methods (IPM), which is one type of algorithms for solving convex optimization problems [14]. Theoretical and experimental studies have shown that interior-point methods require a relatively small number of iterations (typically less than 50) to solve a problem. With recent advances in processor architecture, it is now possible to solve an SOCP problem in roughly the same time as a linear programming problem of equivalent size [27]. It has been shown that interior-point methods can solve SOCP problems in polynomial time, that is, the time required to solve a problem increases by a polynomial function of the size of the input [14]. Polynomial-time algorithms are considered fast, which makes SOCP relevant for real-time applications [14]. SOCP gives us the “best of both worlds” — increased modeling power and flexibility combined with the speed, reliability, and scalability of linear programming [27].
Chapter 3

Pinpoint Landing Problem

In this chapter, the fuel-optimal pinpoint landing problem is presented and formulated as a convex optimization problem, following the approach for Mars pinpoint landing, developed by NASA [7, 13, 18–22]. It was proven that the pinpoint landing can be reformulated as a convex optimization problem, more specifically as a second-order cone problem. Additionally, it was shown that the problem is lossless, i.e., that it can be converted into a convex optimization problem whose optimal solution is also the optimal solution of the original nonconvex problem [18]. To provide a better perception of this guidance technique, the principal steps required to solve the pinpoint landing as an SOCP problem are outlined here [13].

3.1 Problem Formulation

The first step to formulate the pinpoint landing problem is to define the equations of motion of the spacecraft. The spacecraft was modeled as a point mass and the translation dynamics are decoupled from the attitude dynamics. This commonly used assumption reduces the complexity of the problem. This is acceptable because any attitude maneuver required to point the thruster in the correct direction for translation control can be performed very quickly, such that the interaction between the attitude and translation dynamics are minimal [13]. Although attitude control is not an objective of this guidance function, it is helpful to understand that, since the dynamics are only translational, the attitude control can be seen as the process required to point the spacecraft to the same thrust pointing directions calculated by this algorithm.

The model of the spacecraft (Figure 3.1), assumes that there are \( n \) identical thrusters and that they have equal thrust levels at any time. It is possible to model any spacecraft by varying the number of thrusters and the cant angle \( \phi \), i.e., the angle between the thrusters and the central axis of the spacecraft.

The total net thrust provided is

\[
T_c = (nT \cos \phi)e, \tag{3.1}
\]

where \( n \) is the number of thrusters, \( \phi \) the cant angle and \( e \in \mathbb{R}^3 \) is the unit vector describing the thrust direction (see Figure 3.1). A typical engine has a minimum thrust level below which it can not operate
reliably. Thus, an upper and lower thrust limit $T_1$ and $T_2$ are defined for each thruster such that the thrust level $T(t)$ of each thruster is

$$0 \leq T_1 \leq T(t) \leq T_2.$$  \hfill (3.2)$$

The net thrust of the spacecraft will therefore also be limited by an lower and an upper boundaries $\rho_1 = nT_1 \cos(\phi)$ and $\rho_2 = nT_2 \cos(\phi)$, resulting in the thrust control constraint

$$0 \leq \rho_1 \leq \|T_c(t)\| \leq \rho_2.$$  \hfill (3.3)$$

In a surface fixed frame of reference (Figure 3.2), the translation dynamics can be expressed as

$$\ddot{r}(t) = g + \frac{T_c(t)}{m(t)}$$  \hfill (3.4)$$

$$\dot{m}(t) = -\alpha \|T_c(t)\|,$$  \hfill (3.5)$$

where $r \in \mathbb{R}^3$ is the position vector relative to the target, $g \in \mathbb{R}^3$ is the acceleration of gravity of Mars, $T_c \in \mathbb{R}^3$ is the net thrust vector, $m$ the mass of the spacecraft and $\alpha$ the fuel consumption rate. Note that the origin of the frame of reference was set to be the location of the target, to simplify the notation of the problem. The fuel consumption rate, which can be calculated from the specific impulse $I_{sp}$ of the thrusters and the gravitational acceleration of Earth $g_e$, is

$$\alpha = \frac{1}{I_{sp} g_e \cos \phi}.$$  \hfill (3.6)$$
With this notation, the fuel-optimal pinpoint landing is formulated as a non-convex problem.

**Problem 1 - Non-Convex Problem**

\[
\begin{align*}
\text{max} & \quad m(t_f) - \min_{t_f, T_c(\cdot)} \int_0^{t_f} \|T_c(t)\| \, dt \\
\text{subject to} & \quad \ddot{r}(t) = g + T_c(t)/m(t), \quad \dot{m}(t) = -\alpha \|T_c(t)\|, \\
& \quad 0 < \rho_1 \leq \|T_c(t)\| \leq \rho_2, \quad r_1(t) \geq 0, \\
& \quad r(0) = r_0, \quad \dot{r}(0) = \dot{r}_0, \\
& \quad r(t_f) = \dot{r}(t_f) = 0, \\
& \quad \|A_i x + b_i\|_2 \leq c_i^T x + d_i, \quad i = 1, ..., p.
\end{align*}
\]  

(3.7a)

Eq. (3.7a) is the cost of the problem, which is to maximize the final mass of the spacecraft. Equations from (3.7b) to (3.7e) represent the constraints of the problem. The constraint \(r_1(t) \geq 0\) was added to avoid sub-surface flight. Equations (3.7d) and (3.7e) represent the initial and final states, where \(m_{\text{wet}}\) is the initial mass of the spacecraft.

It’s important to note that the thrust boundary in Eq. (3.7c) is not convex due to the non-zero minimum boundary. To take into account constraints that may be added to the problem later, Eq. (3.7f) was introduced, which is the formulation of second-order cone constraints.

### 3.2 Convexification Process

#### 3.2.1 Convexification of Thrust Magnitude Constraint

Due to the minimum thrust level imposed, the region of possible thrust magnitude values defines a non-convex set. Figure 4.9(a) illustrates a two-dimensional scenario of the possible values of the thrust vector. In this 2D example, we can have two points of the set, in which the line that connects them is not always inside the set itself, which by definition, represents a non-convex set.

To overcome this non-convexity, a slack variable \(\Gamma(t)\) is introduced to substitute \(\|T_c(t)\|\) in the cost function and on the constraint (3.7c). This slack variable has no physical meaning and it is only used for the calculations. The non-convex set is transformed into a convex set, by adding a new convex constraint \(\|T_c(t)\| \leq \Gamma(t)\). A visual illustration of this relaxation is represented in Figure 4.9(b). Notice how the 2D graph of Figure 4.9(a) is stretched along the \(\Gamma\) direction, making the set of possible solutions a sliced cone, which represents a convex set.

With this relaxation, the thrust constraints became convex, but feasible solutions of the relaxed problem are not necessarily feasible solutions of the original problem. As an example, the pair \((\|T(t)\|, \Gamma(t)) = (0, \rho_2)\) is feasible in the relaxed set, but \(\|T(t)\| = 0\) is not feasible in the original set. However, it can be shown that, for an optimal solution, \(\|T(t)\| = \Gamma(t)\), for any \(t \in [t_0, t_f]\), thus, the optimal solutions of the relaxed problem are in the surface of the cone, which are feasible solutions of the original set. Addi-
tionally, it can also be shown that, in optimal solutions, \( \|T_c(t)\| = \rho_1 \) or \( \|T_c(t)\| = \rho_2 \), for any \( t \in [t_o, t_f] \), therefore, the net thrust is expected to be at maximum or minimum levels during the whole flight. See [13] for proofs.

### 3.2.2 Linearization of The Dynamics

Another source of non-convexity in the original formulation is in the equations of motion of the problem. In the calculation of the acceleration \( \ddot{r}(t) \), the net thrust is divided by the mass of the spacecraft, and a division by one of the state elements occurs, which is a high non-convex relation. To overcome this situation, a change of variables is made. First, two new control variables, \( \sigma \in \mathbb{R} \) and \( u \in \mathbb{R}^3 \) were introduced:

\[
\sigma(t) \equiv \frac{\Gamma(t)}{m(t)} \tag{3.8}
\]

\[
u(t) \equiv \frac{T_c(t)}{m(t)}. \tag{3.9}
\]

With this substitution, the equations of motion can be rewritten as:

\[
\ddot{r}(t) = u(t) + g, \tag{3.10}
\]

\[
\frac{\dot{m}(t)}{m(t)} = -\alpha \sigma(t). \tag{3.11}
\]

The resulting mass dynamics becomes a differential equation, with the solution:

\[
m(t) = m_0 \exp \left[ -\alpha \int_0^t \sigma(\tau) \, d\tau \right]. \tag{3.12}
\]

Having in mind that the objective of the algorithm is to minimize fuel consumption (or maximize the final mass \( m(t_f) \)), it is possible to represent the same objective as minimizing

\[
\int_0^{t_f} \sigma(t) \, dt. \tag{3.13}
\]
The thrust constraints can be rewritten in terms of the new variables $u$ and $\sigma$ as

$$\|T_c(t)\| \leq \Gamma(t) \iff \|u(t)\| \leq \sigma(t), \quad (3.14)$$

$$\rho_1 \leq \Gamma(t) \leq \rho_2 \iff \frac{\rho_1}{m(t)} \leq \sigma(t) \leq \frac{\rho_2}{m(t)}. \quad (3.15)$$

The inequalities in Eq. (3.15) define a convex set for the control input $u$ [13]. However, because $m$ is a variable of the problem, these inequalities become bilinear, which defines a non-convex constraint. Therefore, an additional variable change is introduced to convexify the inequalities in Eq. (3.15):

$$z \equiv \ln m \iff m \equiv e^z. \quad (3.16)$$

With this substitution, the mass consumption equation (3.11) can be rewritten as

$$\frac{\dot{m}(t)}{m(t)} = -\alpha \sigma(t) \iff \dot{z}(t) = -\alpha \sigma(t) \quad (3.17)$$

and the inequalities in Eq. (3.15) become

$$\rho_1 e^{-z(t)} \leq \sigma(t) \leq \rho_2 e^{-z(t)}, \quad \forall \ t \in [0, t_f]. \quad (3.18)$$

Note here that, for any composition of $z(t)$, the left part of the inequality (3.18) describes a convex feasible set, while the right part does not. This is easier to understand graphically. In Figure 3.4, we can connect any two points of the white region, and the line that connects both points lies within the white region as well. That is not possible for all the points in the grey area. Thus, the inequality $\sigma(t) \leq \rho_2 e^{-z(t)}$ is not convex.

Further, the inequality (3.18) does not fit within the formulation of a second-order cone problem
constraint. So, to rewrite the constraint in a proper way, a Taylor series expansion of $e^{-z}$ is performed,

$$e^{-z} = e^{-z_0} \left[ 1 - (z - z_0) + \frac{(z - z_0)^2}{2} \right] + \ldots$$  \hspace{1cm} (3.19)

The left part of Eq. (3.18) is approximated by a second-order cone, using the first three terms of the Taylor expansion,

$$\rho_1 e^{-z_0} \left[ 1 - (z - z_0) + \frac{(z - z_0)^2}{2} \right] \leq \sigma.$$  \hspace{1cm} (3.20)

For the right part of the inequality in Eq. (3.18), a linearization makes it convexified, by using the first two terms of the Taylor series of $e^{-z}$,

$$\sigma \leq \rho_2 e^{-z_0} [1 - (z - z_0)].$$  \hspace{1cm} (3.21)

The second-order cone and linear approximations of the inequalities in (3.18) are therefore obtained:

$$\rho_1 e^{-z_0} \left[ 1 - [z(t) - z_0(t)] + \frac{[z(t) - z_0(t)]^2}{2} \right] \leq \sigma(t) \leq \rho_2 e^{-z_0} [1 - [z(t) - z_0(t)]], \; \forall \; t \in [0, t_f],$$  \hspace{1cm} (3.22)

where $z_0(t) = \ln (m_{\text{wet}} - \alpha \rho_2 t)$, such that $z_0(t)$ is a lower bound on $z(t)$ at time $t$. The approximations using Taylor series expansion is acceptable because it can be shown that the errors are below two percent for an example flight that lasts 70 seconds [13].

Finally, two additional constraints are introduced to impose the lower and upper boundaries for the variable $z(t)$, at any given time. Since the thrusters have a maximum and minimum thrust levels, the mass consumption will also be limited. Therefore, the variable $z(t)$, which has been introduced to substitute the mass of the spacecraft $m(t)$ is constrained as

$$\ln (m_{\text{wet}} - \alpha \rho_2 t) \leq z(t) \leq \ln (m_{\text{wet}} - \alpha \rho_1 t).$$  \hspace{1cm} (3.23)

The lower boundary represents the possibility that the thrusters run at maximum thrust in the time interval $[t_0, t_f]$, which leads to a maximum mass consumption. The upper boundary corresponds to the opposite case, when the thrusters run at minimum thrust. It is possible now to formulate the pinpoint landing as a convex optimization problem.

**Problem 2 - Convex Problem**

$$\min_{t_f, u(t), \sigma(t)} \int_0^{t_f} \sigma(t) dt$$  \hspace{1cm} (3.24a)

subject to

$$\ddot{r}(t) = g + u(t), \quad \dot{z}(t) = -\alpha \sigma(t),$$  \hspace{1cm} (3.24b)

$$\|u(t)\| \leq \sigma(t),$$  \hspace{1cm} (3.24c)

$$\rho_1 e^{-z_0} \left[ 1 - [z(t) - z_0(t)] + \frac{[z(t) - z_0(t)]^2}{2} \right] \leq \sigma(t) \leq \rho_2 e^{-z_0} [1 - [z(t) - z_0(t)]].$$  \hspace{1cm} (3.24d)
\[
\ln(m_{\text{wet}} - \alpha r) - \alpha t \leq z(t) \leq \ln(m_{\text{wet}} - \alpha r) + \alpha t, \quad (3.25a) \\
z(0) = \ln(m_{\text{wet}}), \quad r(0) = r_0, \quad \dot{r}(0) = \dot{r}_0, \quad (3.25b) \\
r(t_f) = \dot{r}(t_f) = 0, \quad (3.25c) \\
\|A_i x + b_i\|_2 \leq c_i x + d_i, \quad i = 1, \ldots, p. \quad (3.25d)
\]

Additional constraints may be added to the problem. This is taken into consideration with the constraint (3.25d), which represents the formulation of a second-order cone constraint, where \(i\) represents the number of constraints. It is worth mentioning that this problem is not the same as Problem 1, due to the approximations made. However, it has been shown that Problem 2 is convex and its optimal solutions are feasible solutions of the original problem [13].

### 3.3 Discretization

The discretization of Problem 2 is done by splitting the time domain into equal time intervals and applying the constraints at the temporal nodes of each time interval [21]. For any time interval \([0, t_f]\) and time increment \(\Delta t\), the temporal nodes are described as

\[
t_k = k\Delta t, \quad k = 0, \ldots, N \quad (3.26)
\]

where \(N\Delta t = t_f\). The control inputs are discretized as:

\[
u(t) = u_k + (u_{k+1} - u_k)\tau \quad (3.27)
\]

\[
\sigma(t) = \sigma_k + (\sigma_{k+1} - \sigma_k)\tau, \quad \tau = \frac{t - t_k}{\Delta t}, \quad \text{for} \quad t \in [t_k, t_{k+1}], \quad k = 0, \ldots, N - 1 \quad (3.28)
\]

With this discretization and the imposition of the control and state constraints at the time nodes \(t_k : k = 0, \ldots, N\), it’s possible to define a discretized version of Problem 2. Problem 3 represents a finite dimension SOCP problem and solves the pinpoint landing for a given \(t_f\) [21].

**Problem 3 - Convex and Discretized Pinpoint Landing Problem**

\[
\begin{align*}
\min_{t_f, u(\cdot), \sigma(\cdot)} & \quad -z_N \\
\text{subject to} & \quad r_{k+1} = r_k + \frac{\Delta t}{2} (\dot{r}_k + \dot{r}_{k+1}) + \frac{\Delta t^2}{12} (u_{k+1} - u_k), \quad (3.29b) \\
& \quad \dot{r}_{k+1} = \dot{r}_k + \frac{\Delta t}{2} (u_k + u_{k+1}) + g\Delta t, \quad (3.29c) \\
& \quad z_{k+1} = z_k - \frac{\alpha\Delta t}{2} (\sigma_k + \sigma_{k+1}), \quad (3.29d) \\
& \quad \|u_k\| \leq \sigma_k, \quad (3.29e)
\end{align*}
\]
$$
\rho_1 e^{-z_0 k} \left[ 1 - [z_k - z_{0k}] + \frac{(z_k - z_{0k})^2}{2} \right] \leq \sigma_k \leq \rho_2 e^{-z_0 k} \left[ 1 - [z_k - z_{0k}] \right], \quad (3.30a)
$$

$$
\ln(m_{\text{wet}} - \alpha \rho_2 k \Delta t) \leq z_k \leq \ln(m_{\text{wet}} - \alpha \rho_1 k \Delta t), \quad (3.30b)
$$

$$
z(0) = \ln(m_{\text{wet}}), \quad r(0) = r_0, \quad \dot{r}(0) = \dot{r}_0, \quad (3.30c)
$$

$$
r(N) = \dot{r}(N) = 0, \quad (3.30d)
$$

$$
\|A_i x + b_i\|_2 \leq c_i^T x + d_i, \quad i = 1, ..., p. \quad (3.30e)
$$

### 3.4 Minimum-Landing-Error Case

In the case where there isn’t enough fuel to land on target, the algorithm should determine which is the minimum distance to the target possible to achieve, and then calculate the fuel-optimal trajectory to this new target [22]. The formulation of this optimization problem is slightly different. Although the dynamics and most of the constraints stay the same, the cost of the problem is different in the minimum-landing-error case is different. In this case, instead of optimizing the fuel usage, the objective is to minimize the final distance to the original target. Also, in the minimum-fuel case, the final position of the spacecraft is imposed by the constraints. That cannot happen in the minimum-landing-error case since the final position needs to be free to search [22].

**Problem 4 - Discretized Minimum-Error-Landing Problem**

$$
\min_{t_f, u(\cdot), \sigma(\cdot)} \|r(N)\|^2 \quad (3.31a)
$$

subject to

$$
r_{k+1} = r_k + \frac{\Delta t}{2} (\dot{r}_k + \dot{r}_{k+1}) + \frac{\Delta t^2}{12} (u_{k+1} - u_k), \quad (3.31b)
$$

$$
\dot{r}_{k+1} = \dot{r}_k + \frac{\Delta t}{2} (u_k + u_{k+1}) + g \Delta t, \quad (3.31c)
$$

$$
z_{k+1} = z_k - \frac{\alpha \Delta t}{2} (\sigma_k + \sigma_{k+1}), \quad (3.31d)
$$

$$
\|u_k\| \leq \sigma_k, \quad (3.31e)
$$

$$
\rho_1 e^{-z_0 k} \left[ 1 - [z_k - z_{0k}] + \frac{(z_k - z_{0k})^2}{2} \right] \leq \sigma_k \leq \rho_2 e^{-z_0 k} \left[ 1 - [z_k - z_{0k}] \right], \quad (3.31f)
$$

$$
\ln(m_{\text{wet}} - \alpha \rho_2 k \Delta t) \leq z_k \leq \ln(m_{\text{wet}} - \alpha \rho_1 k \Delta t), \quad (3.31g)
$$

$$
z(0) = \ln(m_{\text{wet}}), \quad r(0) = r_0, \quad \dot{r}(0) = \dot{r}_0, \quad (3.31h)
$$

$$
r_1(N) = 0, \quad \dot{r}(N) = 0, \quad m(t_f) \geq m_{\text{dry}}, \quad (3.31i)
$$

$$
\|A_i x + b_i\|_2 \leq c_i^T x + d_i, \quad i = 1, ..., p. \quad (3.31j)
$$

### 3.5 Additional Constraints

Additional constraints were introduced to ensure that specific mission characteristics are accomplished, such as having the sensors pointing to the soil and having a safe navigation throughout the flight [13, 18].
Glide slope constraint

The glideslope constraint consists in defining a set of allowed positions, shaped like a cone, where the probe can be located during the flight. The shape of the cone is defined by the minimum slope angle $\gamma_{gs}$ and by the vertex of the cone, which is located at the landing position. This minimum glideslope angle, which is the angle that the cone does with the surface, can be changed accordingly with landing specifications and surroundings. The glideslope constraint was introduced to maintain the lander trajectory above a certain angle with respect to the final point. This is necessary both to avoid terrain and "subsurface flight", which was seen to be optimal in some cases [18].

Figure 3.5: Visual illustration of glideslope constraint [18].

The position of the spacecraft is given by $r(t) = [r_1(t), r_2(t), r_3(t)]^T$ and the lateral and vertical positions of the spacecraft relative to the target can be described by an angle $\gamma$,

$$\gamma(t) = \arctan\left(\frac{\sqrt{(r_2(t) - r_2(t_f))^2 + (r_3(t) - r_3(t_f))^2}}{r_1(t) - r_1(t_f)}\right).$$  \hspace{1cm} (3.32)

To assure that the spacecraft stays inside the cone area, we must have $\gamma \geq \gamma_{gs}$, during the entire maneuver. This constraint is convex, so it can be introduced easily into the problem. To impose $\gamma \geq \gamma_{gs}$, using the general formulation of Eq. (3.25d) we have:

$$\sqrt{(r_2(t) - r_2(t_f))^2 + (r_3(t) - r_3(t_f))^2} \leq \frac{r_1(t) - r_1(t_f)}{\tan(\gamma_{max})}.$$  \hspace{1cm} (3.33)

Thrust Pointing Constraint

In some missions, it might be necessary to have onboard sensors pointing towards the surface. Thus, the orientation of the spacecraft (attitude) has to be limited. The thrust pointing constraint is introduced to take this limitation into account [18]. The objective is to constrain the maximum tilting angle of the spacecraft relative to the vertical direction.

Since the spacecraft is modeled as a point mass with a thrust vector applied, it is possible to directly restrict the attitude of the spacecraft by simply constraining the directions that the thrust vector can point
This constraint can be trigonometrically expressed as

$$\hat{n} \cdot \frac{T_c(t)}{\|T_c(t)\|} \geq \cos(\theta_{max}),$$

(3.34)

where $\hat{n}$ is a unit vector representing the vertical direction, $T_c(t)$ the net thrust vector and $\theta_{max}$ the maximum allowed angle between the net thrust vector and vertical direction. A visual illustration of this constraint is given in Figure 3.6. In the left image, the bounds on the thrust magnitude are represented without any pointing restriction. In Figures 3.6(b) and 3.6(c), the pointing direction is limited within a pointing envelope. These cases were used to illustrate that the thrust pointing constraint is only convex for $\theta_{max} \in [0, \pi/2]$. Considering the variable substitutions of the control variables as shown in section 3.2.2, Eq. (3.34) can be rewritten as

$$\hat{n} \cdot T_c(t) \geq \Gamma(t) \cos(\theta_{max}) \quad \Leftrightarrow \quad \hat{n} \cdot u(t) \geq \sigma(t) \cos(\theta_{max}).$$

(3.35)

**Maximum Throttle Change Constraint**

In a fuel-optimal trajectory, the thrust profile tends to be at the minimum or maximum magnitude. Consequently, the thrust magnitude tends to change from the lower bound to the upper bound or vice versa in just one time-step. Since this behaviour can not be safely replicated by a rocket engine, a maximum throttle change constraint was included.

The maximum throttle change constraint can be expressed as:

$$\|T(t_{k+1})\| - \|T(t_k)\| \leq \Delta T_{max},$$

(3.36)

where $\Delta T_{max}$ represents the maximum change of thrust magnitude allowed in one time-step. It has been shown in [7] that this constraint can be conservatively approximated by:

$$\Delta T_{min}/m_0 \leq \sigma(t_{k+1}) - \sigma(t_k) \leq \Delta T_{max}/m_0.$$  

(3.37)

Since this constraint represent a linear inequality, it can be easily added to the problem.
3.6 Optimal Flight Time

The discretized problem of pinpoint landing is time-dependent. This means that the time of flight is not one of the optimization variables and must be introduced as an input. However, it's not possible to know what is the optimal $t_f$ that will correspond to the fuel-optimal trajectory. This is both true for the minimum-fuel and minimum-landing-error case. Therefore, to determine the optimal $t_f$ and its associated trajectory, Problem 3 and 4 must be solved multiple times, with different times-of-flight.

To determine the optimal $t_f$, a line-search must be performed, which is an iterative method to find the local minimum of a function. The line-search is complicated, since the set of feasible times-of-flight for which a feasible trajectory exist is unknown beforehand, and can be disconnected [29]. Also, the shape of the function that relates flight time with fuel consumption can vary with boundary conditions. However, it has been shown empirically that the fuel consumption function of $t_f$ is a unimodal function, i.e, a function $f(x)$ with only one minimum at $x = m$, where for $x \leq m$ it monotonically decreases and for $x \geq m$ it monotonically increases [21]. Therefore, if the minimum and maximum values for possible times of flight are known, the optimal $t_f$ lies somewhere between those values.

The golden search method was selected to perform the line-search because it is an efficient iterative technique for finding the minimum in strictly unimodal functions. More information about the golden search method and how to implement it can be found in [29–31]. It was seen that, when using the golden search in a Mars landing scenario, finding the optimal $t_f$ with sufficient accuracy requires calculating up to ten trajectories [29].
Chapter 4

Solving the Pinpoint Landing Problem

There are various solvers available adequate for the pinpoint landing problem, each with its advantages and disadvantages. The main characteristics required in a solver are accuracy, robustness, reliability and overall good performance.

The solvers available can be roughly divided into two groups: classic or direct solvers and parser solvers. Parser solvers distinguish from the others because they allow the user to write the problem in high-level language. Usually, a solver requires the user to input the problem in a particular formulation. Parser solvers, on the other hand, act like a modeling system, allowing the problem to be written in different forms. The parser solver then processes the information to an internal direct solver that finds the solution of the problem.

Parser-solvers are great for early stages of algorithm designing. They allow easy prototyping and development of an algorithm. In this stage, the speed of the solver is most often unimportant. What is relevant is the possibility of making smalls changes to the problem with minimum effort. However, this practicality comes with a cost, since parser solvers are usually slower than direct solvers. When the algorithm is complete, it may be required that the final code runs much faster to meet application needs. This is especially true in real-time applications such as the pinpoint landing scenario, therefore, a direct solver must be used, which runs much faster [14].

The direct solvers are faster, but require more work. To guarantee efficiency, they require specific standard-forms of inputting the problem. Consequently, users must reformulate the problem to the desired form of the solver [14]. This reformulation, as it will be shown, can sometimes be demanding.

4.1 Solver selection

There are many modeling systems available for convex optimization. CVX is a MATLAB modeling language for convex optimization problems that allows constraints and objectives to be specified using standard MATLAB expression syntax [32, 33]. CVXPY works in analogy with CVX but designed for Python [34]. CVXGEN is also a modeling system but a more powerful tool than CVX and CVXPY [35]. Besides allowing the problem to be written in high-level language, it also generates customized C code.
created to run faster and more efficient than in any other modeling system. CVX_GEN is at least 20 times faster than CVX [35]. However, CVX_GEN can not be used in this work, since it is designed for convex, QP-representable problems only, which does not include SOCP problems. Therefore, CVX was the chosen modeling system to prototype and make small adjustments in the code.

Regarding direct solvers, there is also many solvers capable of solving second-order cone programming problems, such as: AMLP [36], CPLEX [37], FICO Xpress [38], Gurobi [39], MOSEK [40] and ECOS [17]. All these solvers are commercial, except ECOS (Embedded Conic Solver), which is an open-source solver designed for SOCP problems. ECOS is written in low footprint, library free ANSI-C code, and it was designed specifically for embedded applications.

For small and medium-size problems, which is the case of the pinpoint landing, ECOS is faster than most existing SOCP solvers [17]. Furthermore, the authors of the convex programming method for pinpoint landing developed a custom solver for their algorithm and compared it with ECOS [41]. The conclusion was that their solver performed similarly to ECOS for small-to-medium problems and there was no need for solver customization [41]. Thus, ECOS was selected to be used in the final code, since it is fast, open-source and meets all the requirements for an onboard implementation such as the pinpoint landing.

4.2 Adapting the Problem for ECOS

In order to use ECOS solver, the problem must be reformulated into the specific formulation required by ECOS:

\[
\begin{align*}
\text{minimize} & \quad \omega^T y, \\
\text{subject to} & \quad Ay = B, \\
& \quad Gy \leq h.
\end{align*}
\]

The vector \(y\) contains the optimization variables and the vector \(\omega^T\) represents the cost of the problem. Matrices \(A\) and \(B\) represent the equality constraints and the matrices pair \(G\) and \(h\) contains the inequality constraints. The problem has to be rewritten in a way that all optimization variables must be contained in the column vector \(y\) and all constraints must be represented by the matrices \(A, B, G\) and \(h\).

The vector \(y \in \mathbb{R}^{11N}\) is constructed by stacking \(N\) smaller column vectors \(y_k \in \mathbb{R}^{11}\), as

\[
y = \left[ y_1 \quad y_2 \quad \cdots \quad y_k \quad \cdots \quad y_N \right]^T, \\
y_k = \left[ r_k \quad \dot{r}_k \quad z_k \quad u_k \quad \sigma_k \right]^T,
\]

for \(k = 1, ..., N\), where \(N\) is the number of time nodes. Each smaller vector \(y_k\) contains the problem parameters for each time node of the problem. The parameters are: the position vector \(r_k \in \mathbb{R}^3\), the velocity vector \(\dot{r}_k \in \mathbb{R}^3\), mass logarithm \(z_k \in \mathbb{R}\), the thrust acceleration vector \(u_k \in \mathbb{R}^3\) and the required
slack variable $\sigma_k \in \mathbb{R}$.

All constraints of the problem must be provided in matrix form. Each row of the matrices $A$ and $G$ has the same length of the vector $y$, so each constraint of the problem is represented by a single row of these matrices. The matrix pair $A$ and $B$ are constructed by stacking all equality constraints on top of each other and the same goes for inequality constraints on $G$ and $h$ matrices. The transcription of these constraints to the matrix form is explained in Appendix A.

The inequality constraints are represented in the matrix pair $G$ and $h$, but they are composed of linear and second-order cone constraints. Therefore, ECOS expects to receive the structure $\text{dims}$ that informs how the inequality constraints are divided. This structure has a variable $\text{dims}.l$ that holds the number of existing linear inequality constraints. The other component of $\text{dims}$ is a vector $\text{dims}.q$, representing the number of existing second-order cone constraints and the dimension of each constraint. The construction of $\text{dims}$ is also explained in Appendix A. Finally, ECOS expects that linear inequality constraints are stacked in the first rows of $G$ and $h$, with the second-order cone constraints following next.

To improve efficiency, ECOS requires the matrices $A$ and $G$ to be in the column-compressed-storage (CCS) form. CCS is a matrix representation where only non-zero elements are stored. In the pinpoint landing problem, matrices $A$ and $G$ in their original form are really sparse, with less than 1% of non-zero elements. With this technique, all zero entries are ignored, offering memory savings and execution efficiency.

Matrices $A$ and $G$ are constructed as explained in Appendix A and are converted to the CCS format afterwards. In the C code, the open-source library CSparse was used for this purpose [42]. This library was chosen for its performance and storage handling. As will be discussed later, the execution of this library is expeditious and has no effect on the overall execution time of the algorithm.

### Minimum-Landing-Error Case

Some adjustments had to be made for the minimum-landing-error case. The cost of the problem in the minimum-landing-error case (minimize $\|r(N)\|^2$) is no longer linear, which is not allowed in an SOCP problem. To overcome this, an auxiliary variable was used to relax this situation. To do so, the new cost of the problem is to simply minimize the auxiliary variable and a new constraint was introduced to declare that the auxiliary variable must be equal or lower than the previous cost expression ($\text{aux} \leq \|r(N)\|^2$).

The matrices $A$, $B$, $G$, $h$ and $\omega$ are constructed in analogy with the minimum-fuel case. The only difference in matrices $G$ and $h$ is that, in this case, there is one more row, representing the new constraint added ($\text{aux} \leq \|r(N)\|^2$). In matrices $A$ and $B$, there are fewer rows since there are less final conditions. The major difference is in the matrix $\omega$, because the cost is different. More information on the construction of these matrices is also presented in Appendix A.

### 4.3 Test and Validation

In order to validate the code, the results of the algorithm were compared to examples presented in the literature [13, 22]. Five case studies were tested: a flight with no additional constraints applied...
Table 4.1: Fuel consumption difference between results and literature [13].

<table>
<thead>
<tr>
<th>Case</th>
<th>Fuel consumption [kg]</th>
<th>Difference [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>No additional constraints</td>
<td>389.5</td>
<td>387.9</td>
</tr>
<tr>
<td>No sub-surface</td>
<td>391.8</td>
<td>390.4</td>
</tr>
<tr>
<td>Glideslope</td>
<td>400.5</td>
<td>399.5</td>
</tr>
<tr>
<td>Vertical flight</td>
<td>295.6</td>
<td>293.6</td>
</tr>
</tbody>
</table>

to the problem; one flight with no sub-surface flight constraint; a flight with glideslope constraint; a flight with thrust-pointing constraint and finally the minimum-landing-error case, when there isn’t enough fuel to reach the target. The spacecraft parameters of these simulations are:

\[
g = \begin{bmatrix} -3.711 & 0 & 0 \end{bmatrix} \text{ m/s}^2, \quad m_{dry} = 1505 \text{ kg},
\]

\[
m_{wet} = 1905 \text{ kg}, \quad I_{sp} = 225 \text{ s}, \quad \bar{T} = 3.1 \text{ kN},
\]

\[
T_1 = 0.3\bar{T}, \quad T_2 = 0.8\bar{T}, \quad n = 6, \quad \phi = 27 \text{ deg},
\]

where \(\bar{T}\) is the maximum thrust of each thruster.

Figures 4.1 to 4.4 were constructed to mirror the examples presented in [13]. All graphs have the same scale as the original examples for a simple comparison. The results of all scenarios are in agreement with the original examples, with only a small difference in fuel consumption. Table 4.1 summarizes the differences of the first four cases.

It was not possible to make a comparison with the case of the thrust-pointing constrained flight, because there is an error in [13]. The initial position and velocity of the spacecraft do not correspond to the graphs presented. Therefore, instead of a comparison, an example is shown in Figure 4.5 to show that the thrust-pointing constraint is well implemented. The initial state is \(r_0 = [2000, 1500, 2500]^T, \quad v_0 = [-75, 80, 20]^T\) and a constraint that limits the thrust-pointing angle from vertical at 45° was introduced. A fuel-optimal trajectory without this constraint would use angles above 45°, but with the addition of this constraint, the angle is limited in the first and final seconds of the flight. This example is interesting, since all constraints are activated, demonstrating that all constraints are properly implemented and that the limitations of the spacecraft are taken into account.

To validate the minimum-landing-error case, a comparison was made with an example from [22]. Note that there is also an error in this article. The spacecraft parameters used for the simulations were not the ones presented in [22] but the ones on the first article [13]. Otherwise, the results from [22] would not make sense. The shortest distance to target possible to achieve in this work was 419.6 m, instead of 404 m in [22], which corresponds to a difference of 3.9%.

It is worth mentioning that, in the original literature, there is no reference to the time-step used in the discretization. The size of the time-step influences not only the numerical accuracy of the problem but the fuel consumption itself. In this work, the time-step is one second, which showed to combine good accuracy with fast execution. The time-step selection will be discussed in section 4.4, but it was found that the fuel consumption decreases linearly with time-step reduction. Therefore, the differences to the
Figure 4.1: Landing case without additional constraints.

Figure 4.2: Landing case without sub-surface flight constraint.
Figure 4.3: Landing case with glideslope constraint.

Figure 4.4: Landing case with vertical flight.
Figure 4.5: Landing case with thrust-pointing constraint.

Figure 4.6: Minimum-landing-error case.
results from the literature can be explained by a different time-step. Another possible reason for these differences is that different solvers have slightly different results due to different optimization parameters. The trajectories in Figures 4.1 to 4.6 were computed with ECOS where the original examples were obtained using SEDUMI [43]. Nonetheless, the differences are smaller than 1\%, showing that the method to solve the pinpoint landing using ECOS is working properly.

4.4 Time-Step Selection

With the discretization of the problem, the time of flight is divided into equal time segments and the constraints of the problem are applied to the time nodes. In the original literature of the convex programming for pinpoint landing, there is no reference nor analysis on this time-step. Since the time-step impacts the accuracy of the results and the performance of the solver, we studied this issue to determine which time-step is most suitable for this application.

Small time-steps will lead to more time nodes, which is good for numerical accuracy, but more time nodes will increase the overall size of the problem, which affects the execution time of the solver. The goal is to find the best balance between numerical accuracy and run-time of the algorithm.

To analyze this issue, three different flight scenarios were considered: a long flight (around 75 seconds) that uses almost all fuel available; a medium time flight (around 65 seconds) that consumes 85\% of the fuel available and a vertical and short flight (around 30 seconds) that consumes less than 50\% of the fuel available. Besides the time-step, all the other problem parameters stayed the same in all scenarios, represented in Figures 4.7, 4.8 and 4.9.

It was observed that, in all scenarios, the fuel consumption increases linearly with the time-step increase, as expected. Regarding the execution time, the run-time of a trajectory calculation decreases exponentially with the time-step increase.

Since the algorithm is supposed to run in real-time, faster execution times are more important than perfect numerical accuracy. However, larger time-steps can generate infeasible solutions, due to numerical errors in the solver. This usually happened for time-steps larger than 3s, but in a few rare cases it also happened for time-steps in the range of 1.5s to 2s. To avoid numerical problems, the time-step should be considerably less than 1.5s. Thus, a time-step of 1.0s was selected, which offers sufficient numerical accuracy and fast execution times.
Figure 4.7: Long flight scenario. $r_0 = [1500, 0, 2000]^T$, $r_0 = [-75, 0, 100]^T$, $t = 78$.

Figure 4.8: Scenario with medium time of flight. $r_0 = [1500, -3500, 2000]^T$, $r_0 = [-65, -30, -20]^T$, $t = 66$.

Figure 4.9: Vertical and short flight scenario. $r_0 = [1500, 0, 0]^T$, $r_0 = [-65, 0, 0]^T$, $t = 31$. 

Chapter 5

Approach to Pinpoint Landing

In this chapter, the approach to pinpoint landing and the design of the guidance function will be discussed. The guidance function will be based on the convex optimization method for pinpoint landing, but some improvements will be made to address some shortcomings and enhance the overall performance and feasibility of the method. The guidance function of this work will be referred to as the improved guidance algorithm, while the NASA algorithm will be referred to as the original guidance algorithm.

5.1 Improved Guidance Algorithm

The original guidance algorithm uses convex optimization to calculate a fuel-optimal trajectory that guides a spacecraft from a point A to land in a point B. However, the original guidance algorithm is not capable of deciding where to trigger the engines, i.e., the point A is where the powered trajectory begins and it has to be provided as input. One contribution of this work is to introduce a method to autonomously determine the optimal position to trigger the powered descent.

Additionally, in the original guidance algorithm, the spacecraft follows only one thrust profile, i.e., after the powered descent begins, there are no more trajectory calculations. In the improved guidance function, however, multiple trajectories are calculated during the powered descent, to re-optimize and correct errors efficiently. Another contribution is a method to take the drag effect into consideration, which is not taken in the original guidance algorithm.

The improved guidance algorithm design can be split into three phases. The first phase is the optimal trigger search, that finds the optimal location to trigger the powered descent and the trajectory data associated. The first step of the optimal trigger search is to perform a line-search to find the optimal $t_f$ that it would take if the powered descent started at the current state of the spacecraft. When there is already a good estimate of the optimal $t_f$, a successive comparison of the future and the current state starts until a trigger decision is made.

The second phase is the re-optimization, where new trajectories are computed to correct eventual errors. After the engines are triggered, the spacecraft starts following the thrust profile selected in the trigger decision. As soon as the spacecraft starts following a thrust profile, the onboard computer
starts computing the trajectory data of the next state. The re-optimization is set to stop calculating new trajectories about 10 s before the end of the flight so that the procedure is executed safely. Finally, when the spacecraft reaches a predefined location above target, a vertical and slow descent is performed for a safe touchdown. Figure 5.1 illustrates a flow diagram summary of the improved guidance function tasks.

With this design, the improved guidance function is able to autonomously guide a spacecraft from the end of the atmospheric braking phase through the landing. Every input required by the guidance function can be read during the flight, which makes this guidance function completely autonomous. The required inputs are the position, velocity, acceleration and mass of the spacecraft. The guidance function outputs are the thrust commands for the propulsion systems of the spacecraft and also trajectory and velocity profiles for the control systems. The improved guidance algorithm is designed to be robust, reliable and able to always generate a feasible trajectory.

5.2 Optimal Trigger Location

In the original method for pinpoint landing [13], what the algorithm does is to find the fuel-optimal trajectory that gets the spacecraft from a given initial position to the target. However, there is no procedure to determine the optimal position to trigger the powered descent, i.e., the initial position from which to start the trajectory. This initial position from which the trajectory is calculated is crucial. If the trajectory starts too late, the spacecraft may not slow down enough and crash. If the trajectory starts too soon, it will not be a fuel-optimal trajectory because the spacecraft will spend more time fighting gravity.

To address the problem of the optimal trigger point, a method using an interpolation look-up table was developed by the authors of the original algorithm [29]. This interpolation table is used to store the fuel consumption that it would take to land from a grid of thousands of possible initial states. By estimating the futures states of the spacecraft and reading its corresponding fuel consumption from the table, it is possible to know on what position is more efficient to start the maneuver. The interpolation look-up table is a well-tought possible solution, however, it is memory expensive, which is a major constraint in flight computers. Additionally, creating the table requires computing hundreds of thousands of trajectories in the designing phase (200,000 in the case of [29]). To test the method, every entry of the table must be calculated, and if an error is found or if any of the mission's parameters is changed, the table has to be completely re-created.

In this work, an effective alternative is developed. After heuristically analyzing the triggering of the powered descent on different points of the descending flight, it was observed that it is not efficient to start the powered trajectory at the beginning of the descent phase. In unpowered flight, the required fuel to reach target decreases over time and has a minimum near the point in flight where it is not possible anymore to slow down and land the spacecraft (see the profile in Figure 5.2). This behaviour was observed in all of the 1500 simulations.

With this consistent pattern, the approach to find the optimal triggering point can be to make in-flight periodic comparisons of the fuel consumption that it would take if the trigger happened in the current position or in a predicted future state. If the consumption of the future state is lower than the current
First trajectory calculation. Line search to find the first optimal tf.

Calculate future state data.

Compare future state with current state.

Trigger decision

Unpowered flight continues

Ignition of motors.

Follow the thrust profile.

Last step of re-optimization?

Re-optimization

Calculate future re-optimization point data.

Final vertical descent and touchdown.

Figure 5.1: Flow diagram of the improved guidance algorithm.
Figure 5.2: Fuel consumption required to land starting the maneuver at different times of the unpowered descent flight. Time represents the duration of unpowered flight. Here, \( t = 0 \) marks the beginning of the study, in a position known to be far from the optimal trigger position. The red section represents infeasible solutions.

state, the unpowered flight continues and so does the comparison. Whenever the fuel consumption of the future state is higher than the current state or if it is infeasible to start the maneuver in the future state, then it is optimal to trigger the powered descent in the current position.

In this approach, some factors have to be taken into consideration. First, the interval of time in which a state is propagated ahead has to be larger than the time required to compute the trajectory starting at that state. Second, the error between the propagated and actual states (position and velocity) must be small enough. Large errors in propagation can lead to bad trigger decisions or saturation of the spacecraft’s control systems, which can lead to failure.

Figure 5.3 represents the time sequence of the developed method. In a point known to be far from the optimal trigger point, marked as point 0, the search for the optimal trigger begins. Before reaching point 1, the data of points 1 and 2 are already calculated. In point 1 starts the computation of point 3 data. Point 1* marks the end of the computation of point 3. Here, a comparison is made between states 2 and 3 to decide if the trigger is done on point 2 or if the unpowered flight continues. If there is no trigger in point 2, then the computation of point 4 data starts and the procedure is repeated until trigger.

Since there is no knowledge of the time-of-flight of the powered trajectory, the first trajectory calculations need to perform a line-search to find the optimal time-of-flight and its associated trajectory. After that, since there is already a good estimate of the optimal \( t_f \), it is possible to predict the optimal \( t_f \) of the next state based on the \( t_f \) of the previous state. For example, if the next point is five seconds ahead, its optimal \( t_f \) will be around the previous \( t_f \) minus five seconds. This was seen to be true and therefore only a few trajectories calculations are enough to find the optimal \( t_f \) and the optimal trajectory of the next state.
To select the period of the optimal trigger search, it is important to analyze the number of trajectories that must be computed on each search loop and also the time required to compute each trajectory.

5.2.1 Real-time Trajectory Computation Time

In a Mars mission, the landing guidance algorithm is supposed to run on a radiation-hardened flight processor. Thus, it is important to characterize the execution time of the algorithm in a flight processor. Since flight processors are way more complex than desktop processors (different instruction sets and memory access speeds), the run-time on a desktop computer cannot simply be scaled to a flight processor by the ratio of clock speeds [41].

It was not possible in this work to use a radiation-hardened processor and analyze the real execution time of the algorithm. However, the authors of the convex programming approach to pinpoint landing implemented their code in a NASA flight processor and their results can be used as a reference point for the real-time execution times [41]. As discussed in Section 4.1, the customized solver has similar performance to ECOS and there is no need for a solver customization [41]. Since the execution time of the algorithm of this work-frame is affected only by the speed of ECOS (the code prior to the call of ECOS runs instantaneously), the performance of both algorithms will be similar. In addition, the trajectory example used in [41] is similar to the ones of this work. Thus, it is considered acceptable to use the results from [41] as a reference.

An extensive analysis was done in [41] on the execution time of a trajectory calculation on a radiation-hardened processor and it was concluded that an optimal trajectory can be calculated on average in 0.7 s. The tests were performed in a BAE RAD750 PowerPC, which is part of a flight software testbed at NASA Jet Propulsion Laboratory. The execution time varies if the solution is feasible or not. Figure 5.4 shows the mean run-times of the optimization problem as function of the time of flight. The red square in this figure separates infeasible times-of-flight (on the left) from feasible ones (on the right). For feasible times of flight, the execution time does not oscillate much and the worst-case execution time is 0.8 s. For infeasible times-of-flight, the execution time fluctuates between 0.4 s and 1.3 s. Thus, the worst-case
execution time that will be used is 1.3 s for infeasible trajectories and 0.8 s for feasible trajectories.

This is just a reference point and the execution times of this work can be slightly different. Testing the real execution times of the improved guidance function in a flight processor must be addressed in future work. Nonetheless, these execution times are the worst-case scenario and can be used as a reference in the demonstration of the optimal trigger search method.

5.2.2 Trigger Search Time-step

To choose a time-step for the optimal trigger search, it is important to know how large is the window of times-of-flight where the optimal $t_f$ of the next state can be. It was seen that the optimal $t_f$ of the next state lies around the optimal $t_f$ of the previous loop minus the time-step of search. This $t_f$ will be called the predicted time-of-flight. Occasionally the optimal $t_f$ of the next step is smaller or larger than the predicted $t_f$. The difference between the larger and smaller possible times-of-flight will be called the window of times-of-flight where the optimal $t_f$ of the next step can be. The size of this window will determine how many trajectories have to be computed for each trigger search loop and, consequently, the time-step of the search. If the size of the window is, for example, nine seconds, not all nine corresponding trajectories must be calculated. An analysis was made to measure the consumption of trajectories with a $t_f$ larger than the optimal. It was seen that if the $t_f$ of a trajectory is 1 s or even 2 s larger than the optimal $t_f$, the additional fuel consumption is lower than 2%. Therefore, in a window of nine seconds, only four times-of-flight could be tested with acceptable results.

The size of such windows of times-of-flight was analyzed for different trigger search time-steps.
Table 5.1: Range of possible times-of-flight in the optimal trigger search and worst-case execution time associated.

<table>
<thead>
<tr>
<th>Search time-step [s]</th>
<th>Minimum difference from predicted $t_f$ [s]</th>
<th>Maximum difference from predicted $t_f$ [s]</th>
<th># of Trajectory Calculations</th>
<th>Worst-Case Execution Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>-1</td>
<td>2</td>
<td>3</td>
<td>3.4</td>
</tr>
<tr>
<td>3</td>
<td>-2</td>
<td>2</td>
<td>3</td>
<td>3.4</td>
</tr>
<tr>
<td>4</td>
<td>-2</td>
<td>2</td>
<td>3</td>
<td>3.4</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
<td>2</td>
<td>3</td>
<td>3.4</td>
</tr>
<tr>
<td>6</td>
<td>-2</td>
<td>3</td>
<td>3</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Table 5.1 lists the window size for different time-steps used in the optimal trigger search. Each trigger search time-step was tested in 1500 descent trajectories, starting from different initial states. It was seen that, for every time-step tested, only three times-of-flight must be tested.

In the optimal trigger search loop, when computing the trajectories of the different times-of-flight, the worst-case scenario would be to have just one feasible $t_f$ and the remaining times-of-flight infeasible, that can take longer to compute. So, to compute three trajectories, corresponding to the three $t_f$s of Table 5.1, the worst-case execution time would be $1.3 + 1.3 + 0.8 = 3.4$ s. Therefore, it was decided to use the trigger search time-step of 4 s, since it is the minimum time-step of Table 5.1 larger than 3.4 s.

5.3 Re-optimization in Real-Time

Another development to the pinpoint landing algorithm is a periodic loop for re-optimizing the trajectory during powered flight. Re-optimizing the trajectory improves the robustness of the guidance and allows an efficient correction of errors. This means that, whenever a dispersion in flight occurs, instead of having a trajectory control constantly correcting that error, a new fuel-optimal trajectory can be produced from the dispersed position.

The time sequence for the re-optimization case is simple. Whenever the spacecraft starts following the last thrust profile generated, there is enough time to compute the future state and its associated trajectory. This can be quickly computed with few trajectory calculations since the window of the possible times-of-flight is small.

The ideal condition would be to re-optimize the trajectory as frequent as possible to correct possible errors sooner in the powered trajectory, leading to smaller “jumps” in the thrust magnitude and direction. However, the frequency of the re-optimization has to take into account the computations execution of the next loop. As in the optimal trigger search, the size of the times-of-flight window of the next re-optimization loop will determine the re-optimization time-step. Table 5.2 lists the window of possible times-of-flight for different re-optimization time-steps and the worst-case execution time associated. Again, the worst-case scenario is when all times-of-flight generate infeasible solutions, with exception of one $t_f$, which will correspond to the optimal solution. Therefore, the time-step selected for the re-optimization was 5 s since it is the smallest time-step that is larger than the worst-case execution time.

In the context of the re-optimization, a final vertical descent phase was included. Instead of pointing
Table 5.2: Range of possible times-of-flight in the re-optimization loop and worst-case execution time associated.

<table>
<thead>
<tr>
<th>Re-optimization time-step [s]</th>
<th>Minimum difference from predicted $t_f$ [s]</th>
<th>Maximum difference from predicted $t_f$ [s]</th>
<th># of Trajectory Calculations</th>
<th>Worst-Case Execution Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td>3.4</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>4.7</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>4.7</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>9</td>
<td>4</td>
<td>4.7</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>8</td>
<td>4</td>
<td>4.7</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>9</td>
<td>4</td>
<td>4.7</td>
</tr>
</tbody>
</table>

directly to the target, the algorithm produces trajectories to reach a specific point above the target with a specific downward velocity and then a final thrust profile is produced to vertically land on the target. This was included to assure a vertical and smooth descent in the last seconds of the flight, which improves the safety of the landing. Other approaches could be made to achieve the same effect, such as reducing the maximum tilt angle and maximum thrust in the last seconds of flight. However, having a separate final vertical descent is a simple and robust solution, that integrates well in the re-optimization scenario.

It is worth mentioning that the addition of the final vertical descent is not a fuel-optimal addition. The optimal scenario would be to let the algorithm find the fuel-optimal trajectory from start to finish. However, mission safety is more important than fuel optimality. The touchdown is the most critical part of the flight and it is important that the spacecraft lands in a safe and predefined manner. In this way, it is guaranteed that the spacecraft has a consistent behaviour at touchdown, for all the possible initial states. The maximum thrust on this final trajectory can also be set to a value slightly higher than the hovering thrust magnitude, which will generate a smooth vertical deceleration. It was seen that the addition of this vertical and smooth descent greatly increased the safety of the landing.

### 5.4 Drag Effect

Due to the quadratic formulation, the drag force could not be included in the original pinpoint landing algorithm. If drag is not taken into account, especially when acceleration due to drag is a significant part of the acceleration profile, the flight path will be completely different from what was calculated and the trajectory has to be constantly corrected. To overcome this situation, we tried to linearly approximate the drag force so that it could be included in the problem formulation. However, this cannot be done because its inclusion would invalidate the change of variables made and consequently the overall formulation of the problem.

There is, however, an alternative. Since the algorithm does not take drag into account, the thrust profile produced can be interpreted as the optimal force that should act on the spacecraft. Since the drag effect is helping to slow down the spacecraft, it can be subtracted from the calculated thrust profile, if we measure the drag force using the spacecraft’s accelerometers.

However, subtracting the drag force has side effects that cannot be ignored. Since the beginning of
the powered descent starts at relatively high speeds, the drag acceleration is about 40% of the maximum thrust. This value is even higher than the minimum thrust provided by the engine. Thus, subtracting the drag from the optimal force can drastically change the direction and magnitude of thrust. This could mean that the spacecraft had to turn upside down at the beginning of the maneuver and have thrust magnitude lower than the minimum possible.

To address this problem, a simple function was included to check and correct if the resulting thrust profile violates any of the constraints. If the resulting thrust direction violates the maximum pointing angle, the direction of thrust is set at the original angle calculated by the algorithm and the thrust magnitude remains the same. In the case that the resulting thrust magnitude is lower than the engine minimum, the thrust magnitude is changed to the minimum value possible, while the thrust direction stays the same. Both violations can happen at the same time. In that case, the direction of thrust is corrected first, and then the magnitude. Tests showed that this method performs well because it incorporates smoothly in the re-optimization technique. Whenever the thrust vector has to be corrected, the trajectory is slightly different from what was calculated but the differences are small and are corrected in the next re-optimization loop. The thrust command corrections always increase the vertical component of thrust by decreasing the thrust-pointing angle or increasing the thrust magnitude. Therefore, the spacecraft will always have enough vertical deceleration and there is no risk of creating infeasible solutions due to these adjustments.

Instead of subtracting the drag force, one could neglect it and let the re-optimization method correct its effect. However, having accelerations higher than expected by the algorithm leads to huge jumps in thrust magnitude when the trajectory is re-optimized. That happens because since the spacecraft decelerates faster than it was optimally calculated, the next re-optimization loop will require less acceleration and it will start with a minimum thrust magnitude. This behaviour occurred consistently and it is not desired. Nonetheless, testing this approach actually gave a good sense of the robustness of the re-optimization technique. Even when flight trajectories had different accelerations than what was expected, the re-optimization algorithm was always able to generate a feasible solution.

### 5.5 Pinpoint Landing Example

We developed an example to illustrate a pinpoint landing on Mars. The simulations were developed in Simulink and the landing parameters are listed in Table 5.3. The target is located at the origin of the reference frame and the initial conditions are

\[
\begin{align*}
\mathbf{r}_0 &= \begin{bmatrix} -5540 & 570 & 6420 \end{bmatrix}^T \text{ m}, \\
\mathbf{v}_0 &= \begin{bmatrix} 207 & -25 & -103 \end{bmatrix}^T \text{ m/s}, \\
m_0 &= 346 \text{ kg}.
\end{align*}
\]

All trajectories are calculated to end at an altitude of 20 m above target, with downward velocity of 5 m/s. Afterwards, a final vertical and smooth deceleration is performed to land at target with zero velocity. In
Table 5.3: Landing parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mars Gravitational Acceleration</td>
<td>$g$</td>
<td>$0.03725258^T$ m/s$^2$</td>
</tr>
<tr>
<td>Spacecraft Total Mass</td>
<td>$m_{\text{wet}}$</td>
<td>346 kg</td>
</tr>
<tr>
<td>Specific Impulse</td>
<td>$I_{sp}$</td>
<td>311 s</td>
</tr>
<tr>
<td>Number of Motors</td>
<td>$n$</td>
<td>1</td>
</tr>
<tr>
<td>Engines Cant Angle</td>
<td>$\phi$</td>
<td>0 deg</td>
</tr>
<tr>
<td>Engine Maximum Thrust</td>
<td>$T$</td>
<td>3600 N</td>
</tr>
<tr>
<td>Maximum Throttle Level</td>
<td>$T_{\text{max}}$</td>
<td>100%</td>
</tr>
<tr>
<td>Minimum Throttle Level</td>
<td>$T_{\text{min}}$</td>
<td>30%</td>
</tr>
<tr>
<td>Maximum Pointing Angle</td>
<td>$\theta_{\text{max}}$</td>
<td>60 deg</td>
</tr>
<tr>
<td>Glideslope Angle</td>
<td>$\gamma_{gs}$</td>
<td>5 deg</td>
</tr>
<tr>
<td>Optimal Trigger Time-Step</td>
<td>$\text{try}_{\Delta t}$</td>
<td>4 s</td>
</tr>
<tr>
<td>Re-optimization Time-Step</td>
<td>$\text{reopt}_{\Delta t}$</td>
<td>5 s</td>
</tr>
</tbody>
</table>

this flight, the decision to trigger the engines was made at the state:

$$r_0 = \begin{bmatrix} -832 \\ 1 \\ 2884 \end{bmatrix}^T \text{m}, \quad (5.2a)$$

$$v_0 = \begin{bmatrix} 130 \\ -17 \\ -147 \end{bmatrix}^T \text{m/s}, \quad (5.2b)$$

Figure 5.5 illustrates the time evolution of position, velocity, acceleration, net thrust force, throttle level and the angle between the thrust vector direction and the vertical direction. During the powered trajectory, three re-optimizations were made and the total time-of-flight was 36 s. The fuel consumption was 34.6 kg. Figure 5.7 compares the thrust-pointing angle of this trajectory with or without a thrust command correction. It can be seen that the thrust commands would violate the maximum-pointing angle constraint and therefore they are corrected in the first ten seconds of the powered flight. Consequently, the flight path is slightly different from what was expected but everything is corrected with the re-optimization.

Figure 5.8 illustrates how the re-optimization method affects the thrust profile. The solid lines represent the re-optimization thrust profile and the dashed ones the original thrust profile where there is no re-optimization, no final vertical burn and no thrust command correction. The last five seconds of the re-optimization thrust profile correspond to the final vertical burn.

On both thrust profiles the drag force is subtracted from the output of the convex optimization algorithm and there is a violation of the maximum pointing angle. The re-optimization profile is corrected while the original profile is not. Note that in the first five seconds of powered flight, if there was no thrust command correction, the profiles would be coincident. To correct the thrust-pointing angle, the vertical component of the thrust vector was increased and the lateral component decreased. Due to this correction, the re-optimization of the next step slightly changes the thrust profile of the remaining flight.

This trajectory was selected as an example because it has a thrust command correction but also because it slightly overshoots the target (see Figure 5.6). If the trigger decision was made earlier, the trajectory would have a more direct path to the target, but it wouldn’t be the fuel-optimal solution, which is interesting.
Figure 5.5: Mars pinpoint landing example with \( \mathbf{r}_0 = [-5540, 570, 6420]^T \) m, \( \mathbf{v}_0 = [207, -25, -103]^T \) m/s, \( m = 346 \) kg and \( t_f = 36 \) s. The fuel consumption is 34.6 kg.
Figure 5.6: Three-dimensional trajectory produced by the guidance function.

Figure 5.7: Thrust pointing constraint with and without correction.
Figure 5.8: Thrust profile with and without re-optimization.
Chapter 6

Simulations of Mars Pinpoint Landing

The quality of the improved guidance algorithm is demonstrated in simulations of pinpoint landings on Mars. The parameters of the mission, spacecraft characteristics and Mars' atmosphere model were provided from the context of an ESA mission [25, 44–46]. The improved guidance function developed in this work is responsible for the last and critical phase of the mission, the powered descent and pinpoint landing.

Regarding the EDL of the mission, when the spacecraft reaches the atmosphere of Mars, it aero-brakes to supersonic speeds using a heat-shield. At this point, a supersonic retro-propulsion burn slows down the spacecraft to subsonic speeds [25]. Due to atmospheric uncertainties, the spacecraft can accumulate considerable position and velocity errors during these entry and braking phases.

For the setup of the pinpoint landing simulations, 1537 possible atmospheric condition were considered, and for each of them, the spacecraft has a slightly different acceleration pattern in the supersonic retro-propulsion burn phase. This originates 1537 possible initial states for the powered descent, deemed representative of the possible real initial conditions [44, 46]. Each flight will be continued in the same atmospheric conditions that lead to the initial state of the powered descent.

The range of initial state vectors for the powered descent are listed in Table 6.1 and the parameters of the landings are listed in Table 5.3. The initial altitude has a dispersion of 3 km and the horizontal position has a dispersion of almost 5 km. All trajectories are calculated to end at an altitude of 20 m above the target, with a downward velocity of 5 m/s. Afterwards, a final vertical and smooth deceleration is performed to touchdown at target with zero velocity. The main goal of these simulations is to test if it is possible to pinpoint land in all of the 1537 possible states.

6.1 Simulations Setup

A Simulink model was prepared to simulate the pinpoint landing on Mars. The model can be split into two major blocks, the dynamics and GNC blocks (Figure 6.1). The GNC block is responsible for every task related to generating thrust commands. Figure 6.2 illustrates how the GNC block is structured. The line search sub-block only runs once, at the beginning of the simulation, and is responsible for
Table 6.1: Range of initial state vector components.

<table>
<thead>
<tr>
<th>Range</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Position [m]</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>$[-3605; -8037]$</td>
</tr>
<tr>
<td>$y$</td>
<td>$[430; 1086]$</td>
</tr>
<tr>
<td>$z$</td>
<td>$[5248; 8137]$</td>
</tr>
<tr>
<td>Velocity [m/s]</td>
<td></td>
</tr>
<tr>
<td>$V_x$</td>
<td>$[190; 216]$</td>
</tr>
<tr>
<td>$V_y$</td>
<td>$[-30; -20]$</td>
</tr>
<tr>
<td>$V_z$</td>
<td>$[-129; -84]$</td>
</tr>
</tbody>
</table>

estimating the first time-of-flight. The optimal trigger search sub-block runs periodically, with a sample time equal to the trigger search time-step, until trigger location is found. Finally, a trigger flag starts the thrust computations sub-block, which is responsible for calculating the trajectories and correcting the thrust commands when necessary.

Trajectory calculations are performed inside an s-function block, as can be seen in Figure 6.4. An s-function is a Simulink block that can execute C code in the Simulink environment. This feature was used since it is possible to use the graphical interface of Simulink and MATLAB for simulations and still test the performance of the algorithm in C language.

In each loop of the optimal trigger search, three trajectories are calculated. The three times-of-flight tested are the ones with a difference of $-2, 0$ and $2$ s from the predicted $t_f$ (time-of-flight of the previous loop minus four seconds, which is the period of the trigger search). Similarly, in every re-optimization loop, four trajectories are calculated. The four times-of-flight tested are the ones with a difference of $0, 3, 6$ and $9$ s from the predicted $t_f$ (time-of-flight of the previous loop minus five seconds, which is the period of the re-optimization.)

Finally, the dynamics block deals with everything related to flight dynamics. It receives the thrust command as input and computes the change in position, velocity and mass of the spacecraft, based on the acting forces (thrust vector, gravity acceleration and drag). The drag forces are computed with an atmosphere model of Mars [46].

### 6.2 Results and Discussion

To obtain the typical shape of the flight path, a plot was made with the 1537 simulated trajectories. Figure 6.5 illustrates the vertical profile of these trajectories, where the marked dots represent the trigger location. It was observed that the optimal altitude to trigger tends to be in the range of $2$ km to $3$ km and that in some trajectories, the optimal behaviour is to overshoot the target.

In every case studied, the improved guidance function was able to generate a feasible trajectory to land on the target. Although during each flight the guidance function calculates trajectories to start from dozens of locations, during the trigger search and the re-optimization phases, not a single case of in-feasibility was found, demonstrating two aspects of the improved guidance function. First, that calculating three trajectories in the trigger search and four trajectories in the re-optimization phase is
Figure 6.1: Two major blocks of Simulink model.

Figure 6.2: GNC block.

Figure 6.3: Dynamics block.
enough to find a feasible and optimal solution. Second, it also shows that the subtraction of the drag force and the thrust commands corrections associated do not produce infeasible trajectories and do not impact the feasibility of subsequent trajectory calculations.

It was seen that with the re-optimization technique, the throttle level can fluctuate between minimum and maximum a few times during the flight. Figure 6.6 illustrates a typical thrust profile with this behaviour. An example where this behaviour does not occur can be seen in Figure 5.5 of Section 5.5. The thrust profiles of all trajectories tend to be similar to one of these examples. On the other hand, the orientation corrections of the re-optimization tend to be insignificant for all trajectories. An example of these corrections can be seen in Figure 6.7.

The average fuel consumption of the landings is $32.6 \text{ kg}$, with the worst case being $45.77 \text{ kg}$. This worst case consumption happened only once in the 1537 trajectories and the second worst case is $40.8 \text{ kg}$.

To study the fuel-optimality of the improved guidance function, its consumption was compared to an enhanced Apollo-like PD algorithm, capable of pinpoint landing but not fuel-optimal. This Apollo-like guidance function was used in end-to-end simulations of pinpoint landings on Mars [25]. The conditions of both simulations were set to be the same. The difference is that more than 1537 initial states were studied with the Apollo-like guidance. Figure 6.8 illustrates the fuel consumption of both cases. The convex optimization guidance function, being fuel-optimal, is capable of a lower consumption. On average, it is possible to save 30% of fuel when using the improved convex optimization guidance function.

It is also worth analyzing the fuel consumption savings due to the optimal trigger. For that purpose, the consumption of each landing was compared to a case without trigger search, that is, if the trigger
Figure 6.5: Plot of the 1537 trajectories.

Figure 6.6: Throttle level corrections in the re-optimization process.
Figure 6.7: Pitch and Yaw corrections in the re-optimization process.

Figure 6.8: Comparison of fuel consumption between the improved convex guidance algorithm, on the left, and an Apollo-like guidance algorithm on the right. Both guidance functions were capable of pinpoint landings.
happened right at the beginning of the simulation. Figure 6.9 illustrates this comparison. On average it was possible to save $23.6\%$ of fuel by deciding to trigger the powered descent later on the flight.

Regarding landing accuracy, it was possible to land with better than half-meter precision in every case studied. Figure 6.10 illustrates the landing ellipse of the improved guidance function, where the worst lateral error is 25 cm. We also analyze the final position and velocity errors. Figures 6.12 and 6.11 show the lateral and vertical errors of the final position, respectively. Figure 6.13 represents the vertical error of the final velocity. Note that the final position and velocity on all calculated trajectories are zero. Not a single computed trajectory was incapable of landing on target. Thus, it is during the flight that the spacecraft deviates from what was supposed.

As can be seen in Figure 6.12, the horizontal errors in the final position are minimal and do not affect the mission safety. Landing with half-meter horizontal precision is considered to be excellent. However, a vertical error was observed in the final position, which cannot be ignored. This final vertical error ranges from $-0.53$ m to 0.64 m. Having final positions above the ground would mean that when the engines shut down, the lander would fall, which is not desirable. In the case where the final position is below ground level, the spacecraft can impact the surface still with downward velocity. Nonetheless, these errors are considered to be small and can be easily corrected with trajectory control systems. Note that all the trajectories profile calculated by the convex solver land perfectly on target with zero velocity. Thus, the position and velocity profiles can be used as reference for the trajectory control. The final position errors are probably explained by the approximations in the original formulation of the optimization problem [13].

The final velocity errors are minimal and can be ignored. The final velocity in the horizontal directions
ranges from $-0.006 \text{ m/s}$ to $0.007 \text{ m/s}$. The final vertical velocity error is illustrated in Figure 6.13 and it ranges from $-0.04 \text{ m/s}$ to $0.08 \text{ m/s}$, which is also considered not large enough to raise concerns.

![Landing ellipse of the improved guidance function.](image1)

**Figure 6.10:** Landing ellipse of the improved guidance function.

![Final vertical position error.](image2)

**Figure 6.11:** Final vertical position error.
Figure 6.12: Final lateral position errors.

Figure 6.13: Final vertical velocity error.
Table 6.2: Minimum, average and maximum executions times of guidance function tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution times [ms]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Minimum</td>
<td>Average</td>
<td>Maximum</td>
</tr>
<tr>
<td>Line-Search</td>
<td>60.3</td>
<td>78.3</td>
<td>155.2</td>
</tr>
<tr>
<td>Optimal Trigger Search</td>
<td>32.2</td>
<td>52.5</td>
<td>76.6</td>
</tr>
<tr>
<td>Re-optimization</td>
<td>24.6</td>
<td>89.9</td>
<td>151.7</td>
</tr>
</tbody>
</table>

### 6.2.1 Execution Times

The final code is supposed to run on a flight computer but it is worth evaluating its execution times on a regular computer. These simulations were run on a laptop, with Windows 10 and 16 GB of RAM. The processor is a six-core eight-generation i7 (model 8750H) with a base clock speed of 2.2 GHz. During simulations, the processor was working at an average frequency of 3.9 GHz and MATLAB and Simulink were the only tasks running.

The execution times of the different tasks of the improved guidance function are represented in Figure 6.14. These tasks are the first line-search, the optimal trigger search and the re-optimization computations. What is being analyzed is the total execution time of each task. In other words, the total execution time of the line search, the three trajectory computations in the trigger search and the four trajectory calculations in the re-optimization phase. Table 6.2 lists the minimum, average and worst-case execution time of these three tasks.

The results imply that the flight processor can be up to 33 times slower than processor used, in order to compute the re-optimization trajectories under 5 seconds. For the optimal trigger search, the processor can be 50 times slower. In single trajectory calculations, it was on average possible to find a solution under 19 ms. The code prior to the actual call of the solver, which is the construction of data and matrices to provide as input to the solver, is running in less than 1 microsecond. Therefore, the run-time of a trajectory calculation is being affected only by the call of the solver.

### 6.2.2 Optimality

Since the optimal search is performed with a period of four seconds, there will be cases where the spacecraft will trigger its engines before the exact optimal location. Additionally, when calculating the trajectory of the next loop on the optimal trigger search and in the re-optimization loop, not every $t_f$ is tested. In the trigger search, only three out of five possible times-of-flight are tested and in the re-optimization loop, only four out of nine times-of-flight are tested. Therefore, in some cases, the optimal $t_f$ will not be selected. These deviations from the optimal scenario can accumulate, therefore, it is important to test the optimality of the landings. To test this, we performed additional 1537 simulations where the period of the optimal trigger was set at 1 s. Also, in the trigger search and re-optimization loops, every $t_f$ of the window of times-of-flight was tested. This is still an approximation since the real optimal trigger time could be between the trigger times tested, but having a time-step of one second was considered to be a good approximation.

In Figure 6.15, the fuel consumption of the optimal scenario is compared with the actual landing
Figure 6.14: Execution times.
consumption to measure the overall optimality of the improved guidance function. It was seen that 96% of the flight trajectories are at least 98% fuel-optimal and that the worst case of optimality was 89%. This shows that even if the optimal trigger search is performed with a period of four seconds, and that not every $t_f$ is tested in trajectory calculations, it is possible to find a near-optimal trajectory. It is worth mentioning that in the two worst fuel consumption cases, the optimality of the solution was 97% and 99%. Thus, they were not caused by the improved guidance function design.
Chapter 7

Conclusions

The main goal of this thesis was to develop a fuel-optimal guidance function for pinpoint landings on Mars, based on a convex optimization method developed by NASA. Some developments were made to improve robustness and readiness of the original method.

The original formulation of the problem was successfully implemented in C source code. Considerable changes were made to transform the problem into matrix format in order to use the ECOS solver. It was confirmed that the numerical results were in agreement with the original literature of the convex programming approach to pinpoint landing, which validated the transcription. The C code was optimized to both memory use and execution time. The ECOS solver was designed for embedded applications and so it is possible to implement the source code of this work on any spacecraft. Additionally, the code was developed with open-source available tools, making it inexpensive and flexible.

The improved guidance function was tested for more than 1500 cases of Mars landing and it was always possible to land with better than half-meter precision. The performance of this guidance function was compared to an enhanced Apollo-like guidance algorithm, capable of pinpoint landings but not fuel-optimal. Using the convex optimization guidance function, it was possible to save 30% in fuel consumption.

The methods for finding the optimal trigger and re-optimizing the trajectory were tested and have shown to be reliable and robust. Not a single case of infeasibility was found. The optimality achieved with these two methods was seen to be at least 98% for 96% of the cases studied. Thus, it was concluded that the guidance function is capable of generating nearly optimal trajectories and to decide when is the optimal time to trigger the powered descent. Additionally, in a preliminary design phase, the execution times of the guidance function appear to be viable for real-time calculations.

7.1 Future Work

Since there was a successful validation of the guidance function design, the next step is to implement the guidance function with control systems in a more realistic simulation. The major aspects to test are if the spacecraft is capable of following the thrust profile calculated and correct the final position error.
Other important issue to study is the extra fuel required to correct eventual deviations and the adequate risk level of the guidance function.

The primary goal of this work was to study pinpoint landings on Mars. Since there was always fuel available to land on target, the minimum-landing-error case, where the goal is to land as close to target possible, was not simulated. Future work can be to investigate a scenario with limited fuel, where it is not possible to land on target.

Finally, the source code should be implemented in a radiation hardened processor, to test the real execution times of the improved guidance function.
Bibliography


Appendix A

Construction of input matrices for ECOS solver

A.1 Cost matrix

The cost of the problem is

\[
\text{minimize} \quad \int_0^{t_f} \| T_c(t) \| \, dt \quad \Leftrightarrow \quad \int_0^{t_f} \sigma(t) \, dt. \quad \text{(A.1)}
\]

To represent this in the matrix form \((\omega^T \cdot y)\), \(\omega\) is build to select each component \(\sigma_k\) of the vector \(y\).

\[
\omega = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T
\]

A.2 Transcription of equality constraints

All linear equality constraints of the problem must be described in the matrix form as \(A \cdot y = B\). Matrices \(A\) and \(B\) are constructed by stacking all linear equality constraint of the problem and they can be divided in 3 groups as

\[
A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \quad B = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \end{bmatrix},
\]

where the pair \((A_1, B_1)\) represents the initial state conditions, the pair \((A_3, B_3)\) the final state conditions and \((A_2, B_2)\) the equations of motion.

The initial conditions \((r(0) = r_0, \dot{r}(0) = \dot{r}_0, \text{ and } z(0) = \ln(m_{\text{wet}}))\) are converted to the form \(A_1 \cdot y = B_1\)
with:

\[
A_1 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0
\end{bmatrix},
B_1 = \begin{bmatrix}
r_{10} \\
r_{20} \\
r_{30} \\
r_{1f} \\
r_{2f} \\
r_{3f} \\
\ln m_{\text{wet}}
\end{bmatrix}.
\]

In a similar way, the final conditions \( r(t_f) = \dot{r}(t_f) = 0, u_1(t_f) = \sigma(t_f), \) and \( u_2(t_f) = u_3(t_f) = 0 \) are converted to the form \( A_3 \cdot y = B_3 \) with:

\[
A_3 = \begin{bmatrix}
0 & \cdots & 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 0 & 0 & 1 & 0 & \cdots & 0 \\
0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0
\end{bmatrix},
B_3 = \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}.
\]

The equations of motion, presented in [21], are:

\[
r_{k+1} = r_k + \frac{\Delta t}{2} (\dot{r}_k + \dot{r}_{k+1}) + \frac{\Delta t^2}{2} (u_{k+1} - u_k), \quad (A.2)
\]

\[
\dot{r}_{k+1} = \dot{r}_k + \frac{\Delta t}{2} (u_k + u_{k+1}) + g \Delta t, \quad (A.3)
\]

\[
z_{k+1} = z_k - \alpha \Delta t^2 (\sigma_k + \sigma_{k+1}), \quad (A.4)
\]

which can be reformulated as:

\[
\begin{align*}
\begin{cases}
    r_{1,k+1} - r_{1,k} = \frac{\Delta t}{2} (\dot{r}_{1,k} + \dot{r}_{1,k+1}) - \frac{\Delta t^2}{2} (u_{1,k+1} - u_{1,k}) = 0 \\
    r_{2,k+1} - r_{2,k} = \frac{\Delta t}{2} (\dot{r}_{2,k} + \dot{r}_{2,k+1}) - \frac{\Delta t^2}{2} (u_{2,k+1} - u_{2,k}) = 0 \\
    r_{3,k+1} - r_{3,k} = \frac{\Delta t}{2} (\dot{r}_{3,k} + \dot{r}_{3,k+1}) - \frac{\Delta t^2}{2} (u_{3,k+1} - u_{3,k}) = 0
\end{cases} \\
\end{align*}
\]

\[
\begin{align*}
\begin{cases}
    \dot{r}_{1,k+1} - \dot{r}_{1,k} = \frac{\Delta t}{2} (u_{1,k} + u_{1,k+1}) = g_1 \Delta t \\
    \dot{r}_{2,k+1} - \dot{r}_{2,k} = \frac{\Delta t}{2} (u_{2,k} + u_{2,k+1}) = g_2 \Delta t \\
    \dot{r}_{3,k+1} - \dot{r}_{3,k} = \frac{\Delta t}{2} (u_{3,k} + u_{3,k+1}) = g_3 \Delta t
\end{cases} \\
\end{align*}
\]

\[
z_{k+1} = z_k + \alpha \Delta t^2 (\sigma_k + \sigma_{k+1}) = 0. \quad (A.7)
\]
It is possible to rewrite this in matrix form as $A_{\text{dyn}} \cdot \begin{bmatrix} y_k \\ y_{k+1} \end{bmatrix} = B_{\text{dyn}}$, with $(A_{\text{dyn}}, B_{\text{dyn}}) \in (\mathbb{R}^{7 \times 22}, \mathbb{R}^{7 \times 1})$, where $A_{\text{dyn}}$ is represent by:

$$
\begin{pmatrix}
-1 & 0 & 0 & -\frac{\Delta t}{2} & 0 & 0 & 0 & 0 \frac{\Delta t^2}{2} & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 \frac{\Delta t}{2} & 0 & 0 & 0 & 0 \frac{\Delta t^2}{2} & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 \frac{\Delta t}{2} & 0 & 0 & 0 \frac{\Delta t^2}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 \frac{\Delta t}{2} & 0 & 0 & 0 \frac{\Delta t^2}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \frac{\Delta t}{2} & 0 & 0 & 0 \frac{\Delta t^2}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \frac{\Delta t}{2} & 0 & 0 & 0 \frac{\Delta t^2}{2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \frac{\Delta t}{2} & 0 & 0 & 0 \frac{\Delta t^2}{2} & 0 & 0 & 0 \\
\end{pmatrix}
$$

With this reformulation, it is possible to build $A_2$ by arranging $A_{\text{dyn}}$ along $A_2$ in order to select the right pair $y_k$ and $y_{k+1}$ of $y$ for each time node with

$$
A_2 = \begin{bmatrix}
[A_{\text{dyn}}] & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & [A_{\text{dyn}}] & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & [A_{\text{dyn}}] & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & [A_{\text{dyn}}] & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & [A_{\text{dyn}}] & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & [A_{\text{dyn}}] & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & [A_{\text{dyn}}] & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & [A_{\text{dyn}}] & 0 & 0 \\
\end{bmatrix}
$$

and

$$
B_2 = \begin{bmatrix}
B_{\text{dyn}} \\
B_{\text{dyn}} \\
B_{\text{dyn}} \\
B_{\text{dyn}} \\
B_{\text{dyn}} \\
B_{\text{dyn}} \\
B_{\text{dyn}} \\
B_{\text{dyn}} \\
\end{bmatrix}
$$

Note that each zero represents a zero matrix $[7 \times 11]$. There are $(N-1)$ sets of equations, so the size of $A_2$ is $[7(N-1) \times 11N]$ and $B_2$ is $[7(N-1) \times 1]$. 

### A.3 Transcription of inequality constraints

All inequality constraints must be introduced in ECOS in the form $G \cdot y \leq h$. It is also expected that all linear inequality constraints are placed at the top of the matrix $G$ and $h$ and that the non-linear constraints follow next. Matrices $G$ and $h$ are build by stacking all inequality constraints of the problem and they can...
be divided in different groups as:

\[
G = \begin{bmatrix}
G_{l1} \\
G_{l2} \\
G_{l3} \\
G_{l4} \\
G_{l5} \\
G_{s1} \\
G_{s2} \\
G_{s3}
\end{bmatrix}, \quad h = \begin{bmatrix}
h_{l1} \\
h_{l2} \\
h_{l3} \\
h_{l4} \\
h_{l5} \\
h_{s1} \\
h_{s2} \\
h_{s3}
\end{bmatrix}.
\]

Each division of \( G \) and \( h \) represents a type of constraints.

**Linear inequality constraints**

The constraint

\[
\ln m_{wet} - \alpha \rho_2 k \Delta t \leq z_k \iff -z_k \leq -\ln m_{wet} - \alpha \rho_2 k \Delta t
\]

can be formulated in the matrix form \( G_{l1_k} \cdot y \leq h_{l1_k} \) with

\[
G_{l1_k} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & \cdots & 0
\end{bmatrix},
\]

and \( h_{l1_k} = \begin{bmatrix}
-\ln m_{wet} - \alpha \rho_2 k \Delta t
\end{bmatrix},\)

where each pair \((G_{l1_k}, h_{l1_k})\) is built to select the correct \( y_k \) in the vector \( y \). The matrix \( G_l \) and \( h_l \) is constructed by stacking all the matrices \( G_{l1_k} \) and \( h_{l1_k} \) of each time node.

In the same way, the constraints of the type:

\[
z_k \leq \ln m_{wet} - \alpha \rho_1 k \Delta t,
\]

are transformed to \( G_{l2_k} \cdot y = h_{l2_k} \) by stacking the corresponding matrix pair \((G_{l2_k}, h_{l2_k})\) of each constraint:

\[
G_{l2_k} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0
\end{bmatrix},
\]

and \( h_{l2_k} = \begin{bmatrix}
\ln m_{wet} - \alpha \rho_1 k \Delta t
\end{bmatrix},\)

The upper bound on \( \sigma_k \) constraint:

\[
\sigma_k \leq \mu_2 (1 - (z_k - z_{0_k})) \iff \sigma_k + \mu_2 z_k \leq \mu_2 (1 + z_{0_k})
\]
is converted to $Gl_{3k} \cdot y_k \leq hl_{3k}$ with:

$$Gl_{3k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \ \mu_{2k} & 0 & 0 & 0 & 1 & \cdots & 0 \ \end{bmatrix},$$

$$hl_{3k} = \begin{bmatrix} \mu_{2k} (1 + z_{0k}) \end{bmatrix}.$$

Matrices $Gl_3$ and $hl_3$ are constructed by stacking all the matrices $Gl_{3k}$ and $hl_{3k}$ of each time node. The thrust-pointing constraint:

$$u_{1k} \geq \cos \theta_{\text{max}} \sigma_k \iff -u_{1k} + \cos \theta_{\text{max}} \sigma_k \leq 0$$

is reformulated to $Gl_{4k} \cdot y_k \leq hl_{4k}$ with:

$$Gl_{4k} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \ \mu_{2k} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ \end{bmatrix},$$

$$hl_{4k} = \begin{bmatrix} 0 \ 0 \ \end{bmatrix}.$$

The upper bound of the maximum throttle change constraint:

$$\sigma(t_{k+1}) - \sigma(t_k) \leq \Delta T_{\text{max}} / m_0,$$

is rewritten in $Gl_{5k} \cdot y_k \leq hl_{5k}$ with:

$$Gl_{5k} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & 0 \ \end{bmatrix},$$

$$hl_{5k} = \begin{bmatrix} \Delta T_{\text{max}} / m_0 \end{bmatrix}.$$

In analogy, the lower bound of the maximum throttle change ($\Delta T_{\text{min}} / m_0 \leq \sigma(t_{k+1}) - \sigma(t_k)$) is reformulated to $Gl_{6k} \cdot y_k \leq hl_{6k}$ with:

$$Gl_{6k} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & \cdots & 0 \ \end{bmatrix},$$

$$hl_{6k} = \begin{bmatrix} \Delta T_{\text{min}} / m_0 \end{bmatrix}.$$

**Second-order Cone constraints**

Second-order cone constraints are represented as

$$\| A_k \cdot y + c_k \| \leq b_k \cdot y + d_k$$
and ECOS expects to receive these constraints in the following manner:

\[
G_{sk} = \begin{bmatrix} -b_k \\ -A_k \end{bmatrix}, \quad h_{sk} = \begin{bmatrix} d_k \\ c_k \end{bmatrix}.
\]

The constraint

\[
\|u_k\| \leq \sigma_k \iff \begin{bmatrix} u_{1k} \\ u_{2k} \\ u_{3k} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \leq \sigma_k + 0
\]

is rewritten in the correct form with

\[
A_{1k} = \begin{bmatrix} 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \ldots & 0 & 0 \end{bmatrix}, \quad c_{1k} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},
\]

\[
b_{1k} = \begin{bmatrix} 0 & \ldots & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \ldots & 0 \end{bmatrix}, \quad d_{1k} = \begin{bmatrix} 0 \end{bmatrix},
\]

where each matrix pair \(A_{1k}\) and \(b_{1k}\) is constructed to select the right elements of vector \(y\). This is a 4 dimensional second-order cone constraint. Matrices \(G_{si}\) and \(h_{si}\) is constructed by stacking all the matrices \((A_{1k}, b_{1k}, c_{1k}\) and \(d_{1k}\)) of each time node. In analogy, the glideslope constraint, which is represented by

\[
\sqrt{(r_{2k} - r_{2N})^2 + (r_{3k} - r_{3N})^2} \leq \frac{r_{1k} - r_{1N}}{\tan(\gamma_{max})} \iff \begin{bmatrix} r_{2k} \\ r_{3k} \end{bmatrix} + \begin{bmatrix} -r_{2N} \\ -r_{3N} \end{bmatrix} \leq \frac{r_{1k}}{\tan(\gamma_{max})} - \frac{r_{1N}}{\tan(\gamma_{max})},
\]

is transformed into a 3 dimensional second-order cone constraint with

\[
A_{2k} = \begin{bmatrix} 0 & \ldots & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 \end{bmatrix}, \quad c_{2k} = \begin{bmatrix} -r_{2N} \\ -r_{3N} \end{bmatrix},
\]

\[
b_{2k} = \begin{bmatrix} 0 & \ldots & 1 \tan(\gamma_{max}) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 \end{bmatrix}, \quad d_{2k} = \begin{bmatrix} -\frac{r_{1N}}{\tan(\gamma_{max})} \end{bmatrix}.
\]

The last inequality constraint is the lower bound on \(\sigma_k\) and it is described by:

\[
\mu_{1k} \left[ 1 - (z_k - z_{0k}) + \frac{(z_k - z_{0k})^2}{2} \right] \leq \sigma_k \iff z_k^2 - (2z_{0k} + 2)z_k - \frac{2}{\mu_{1k}} \sigma_k + z_{0k}^2 + 2z_{0k} + 2 \leq 0 \quad (A.8)
\]

Since this constraints contains quadratic components, a different approach has to be taken to convert it to the second-order cone form. Constraints that can be written in the form

\[
y^TF^TFy + p^Ty + c \leq 0,
\]
which is the case of constraint (A.8), can be reformulated as a second-order cone constraint as:

$$\left\| \frac{(1 + p^T y + c)/2}{Fy} \right\| \leq \frac{(1 - p^T y - c)}{2} \quad \Leftrightarrow \quad \left\| \begin{bmatrix} \frac{p^T}{2} \\ F \end{bmatrix} \cdot y + \begin{bmatrix} \frac{c + 1}{2} \\ 0 \end{bmatrix} \right\| \leq -\frac{p^T}{2} \cdot y + \frac{1 - c}{2}.$$

Therefore, for each constraint, matrices $F$, $p^T$ and $c$ have to be build to select the right elements of vector $y$, such as:

$$F_k = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$p_k^T = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 0 & -(2z_{0k} + 2) & 0 & 0 & 0 & -\frac{2}{\mu_1} & \cdots & 0 \end{bmatrix}$$

$$c_k = \begin{bmatrix} z_{0k}^2 + 2z_{0k} + 2 \end{bmatrix}.$$

With this transformation, it is finally possible to represent this constraint as a 3 dimensional second-order cone constraint, with

$$A_{3k} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & -(z_{0k} + 1) & 0 & 0 & 0 & -\frac{1}{\mu_1} & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix},$$

$$b_{3k} = \begin{bmatrix} 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 & (z_{0k} + 1) & 0 & 0 & 0 & \frac{1}{\mu_1} & \cdots & 0 \end{bmatrix},$$

$$c_{3k} = \begin{bmatrix} \frac{(z_{0k}^2 + 2z_{0k} + 3)/2}{2} \\ 0 \end{bmatrix}, \quad d_{3k} = \begin{bmatrix} -(z_{0k} + 2z_{0k} + 2) + 1 \end{bmatrix}.$$

Matrices $G_{s3}$ and $h_{s3}$ is constructed by stacking all the matrices ($A_{3k}$, $b_{3k}$, $c_{3k}$ and $d_{3k}$) of each time node. Lastly, a structure $\text{dims}$ has to be made to inform ECOS how matrices $G$ and $h$ are structured. This structure has a variable $\text{dims.l}$ that holds the number of linear constraints and a vector $\text{dims.q}$, which represents the number of second-order cone constraints and the dimension of each one.

$$\text{dims.l} = 4N + 2(N - 1), \quad \text{dims.q} = \begin{bmatrix} 4 & 4 & \cdots & 4 & 3 & 3 & \cdots & 3 & 3 & 3 & \cdots & 3 \end{bmatrix}.$$