Politehnica University of Bucharest
Faculty of Automatic Control and Computers
Department of Automatic Control and Systems Engineering

BACHELOR THESIS

Hands-Off Control for an Aerospace Application

Graduate
Robert-Antonio Stăvărache

Advisor
Assoc. Prof. Bogdan D. Ciubodaru

Coadvisor
Assist. Prof. Andrei Sperilă

Bucharest, 2024

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

The work presented in this thesis lies at the intersection of four theoretical domains. The main subject, upon which this thesis is based, is that of control theory, with an emphasis on a particular branch known as optimal control theory. In the context of this framework, numerical methods and optimization techniques represent the means through which the theoretical results discussed in this thesis are treated, from a practical point of view. These two disciplines allow for the translation of abstract, continuous-time problems into numerical problems, for which tractable and efficient solutions are available. The final domain which plays a crucial role in this thesis is that of hardware design and the implementation of numerical algorithms. With the aid of hardware concepts and design tools, numerical algorithms, which are initially tested in a high-level software environment, are adapted towards a hardware-centric implementation for a specific type of platform which, in the case of the application treated in this thesis, is an FPGA (Field Programmable Gate Array).

My passion for control theory has led me to choose the subject treated in this thesis. I have always been intrigued by the principles of control theory, while also being fascinated by the strong foundation of mathematical concepts upon which this field has been built. Notably, recent history has also shown the true potential of this domain. Control theory has been met with a significant increase in popularity among scientists and engineers at the beginning of the 20th century, thanks to it providing efficient automation solutions for all kinds of technical applications. As control theory came into its own, as a standalone technical discipline, its focus was gradually channeled towards applications in the aeronautical and aerospace domains. Nowadays, significant innovations in the field of hardware equipment allow high performance processing units to run sophisticated control algorithms, which enable the deployment of autonomous systems capable of conducting complex tasks.

The work presented in [10] represents the starting point of this thesis. The aforementioned paper presents a remarkable result regarding systems which are optimal in regards to fuel consumption. The associated solution is obtained via a captivating mathematical approach and, moreover, the challenge of understanding the distinct perspective of the authors, my affinity for practical implementations of theoretical results and my particular interest for aerospace applications have further motivated me in pursuing the subject of this thesis.

The main objectives of this thesis are as follows. I will present a brief outline of optimal control theory, which is intended as a set of preliminary theoretical notions for the main results obtained in [10]. Afterwards, I will elaborate upon the numerical implementation for the solution proposed in [10] using MATLAB as a high level programming language. Lastly, I aim to develop a hierarchy of HDL (hardware description language) modules, which implement the solution in the form of a control algorithm, to be mounted on an FGPA platform.

All of the aforementioned objectives are structured in this thesis according to the following chapter breakdown. In the second chapter, I will analyse general theoretical notions related to dynamical systems and optimal control theory. This will be followed by typical problem formulations from these fields, such as the “Minimum Time Problem” and the “Minimum Fuel Problem” (L^1 optimal control), followed by the L^0 Optimal Control problem (along with the “Hands-off Control” algorithm), which will be discussed based on the solution of the previous two problems. Lastly, conditions for a crucial property called ”normality” will be analyzed, along with the stability guarantees of the provided algorithm.

The third chapter will be dedicated to numerical adaptations of the optimal control problems. The main purpose of this chapter is to prove that the relevant optimal control problems (Minimum time control and L^1 optimal control) have an analog representation, through the form of numerical optimization problems. Simulations will be performed, in order to exemplify the applicability of the alternative numerical forms of the optimal control problems. Firstly, a set of simulations will be presented, for which inbuilt Matlab solvers are used for the optimization problems. Afterwards, a particular solver will be studied and implemented, namely the ADMM solver. This solver will allow for a fine performance tuning of the numerical algorithm, a very important aspect when it comes to hardware implementations. Throughout this chapter, all the simulations are focused on a system modeling the translation dynamics of a satellite.

The fourth chapter starts by presenting the benefits of HDL implementations, along with possible problems that one may encounter when dealing with them. In addition to this, a special tool is presented which greatly aids with hardware implementations, by allowing for the conversion of these implementations from a high level programming language to HDL. Moving on, the process of designing the algorithm is discussed, along with the encountered problems and the solutions meant to address them. Lastly, the performance of the resulting HDL modules are analyzed, followed by a discussion of the FPGA implementation for the algorithm.

Finally, the fifth chapter will present the conclusions drawn after a thorough analysis of the algorithm, from both a theoretical point of view and a practical one. Comments will be provided on the reliability of the hardware implementation, by taking system performance into consideration.

2 Optimal Control Theory

Optimal control theory is a branch of control theory which uses the distinct approach of defining the control problem and objectives through the use of a cost function which is further optimized. Any physical restriction imposed to or by the system can be treated as constraints for the optimization problem. The solution to this optimization problem is called the *optimal control law*. Applying this control law assures stability, along with a set of performances modeled by the cost function.

Before dipping into optimal control theory concepts and fundamentals, some theoretical aspects regarding signals and general control theory have to be covered.

2.1 Signals and Norms

A signal u is a function defined on a time interval taking complex or real values:

$$u\colon \mathcal{T} \mapsto \mathbb{R} \quad (or \quad \mathbb{C}) \tag{2.1}$$

Signals are used to map the evolution of physical variables, such as speeds, positions, temperatures, forces, etc. In the control theory formalism, a system is a entity which receives a set input signals and creates a mapping to a set of output signals. Based on the nature of this mapping, the system can be described using a set of properties such as **linearity**, **time invariance**, **stability**, **controllablity**, **obesrvability** and many other properties. For the sake of this thesis, only the last 3 mentioned properties will be analyzed in the following sections.

The most common tool in functional analysis, used to quantify the amplitude (magnitude), energy, action, and other functions' quantitative aspects, are norms. The most common set of norms used is the set of Lebesgue norms:

$$||u||_p = \left(\int_a^b |u(t)|^p \mathrm{d}t\right)^{\dfrac{1}{p}} \tag{2.2}$$

Based on the choice of p, the Lebesgue L^p norm of a signal can have some significance. For example, for $p = 2$, the energy of the signal is obtained, for $p = 1$, the so-called action, and for $p = \infty$, the maximum value of the function, namely the amplitude of the function. Of great interest to this thesis is the case where $p = 1$. If signal $u(t)$ is used to command the effort of an actuating component for a physical system, then the L^1 norm of $u(t)$ can be correlated to the amount of energy used by that actuator, in order to produce the effort. If the reader is interested in finding more details on this matter, reference [2], chapter 2, section 4 offers an more detailed view regarding scalar and vector signals norms.

As already mentioned, the main focus in control theory is oriented on the study of systems, their nature and, most importantly, algorithms and procedures which can influence the behavior of the system in favourable way, according to the control problem.

2.2 Dynamical Systems

A dynamical system is a set of differential equations describing the physical evolution of a particular system. The physical complexity of the phenomena taking place dictates the mathematical complexity of the set of equations. As a means of quantifying the system's parameters throughout its evolution, the following notations are used:

- *x(t)* - Represents the state of the system. It is a $n \times 1$ vector with real entries. The state vector internally describes the system, in a way relevant to the proposed problem, at a given time t.
- *u(t)* - Represents the input to the system. It is a $m \times 1$ vector with real entries. The input consists of information/signals that are introduced into the system, at a given time t.
- *y(t)* - Represents the output of the system. It is a $p \times 1$ vector with real entries. This vector represents the actual quantities that are available for measurements, based on which the control signals are synthesized.

Using the introduced variables, the following set of equations are generally used to fully describe the dynamics of a system:

$$\dot{x}(t) = f[x(t), u(t), t] \tag{2.3}$$
$$y(t) = g[x(t), u(t), t] \tag{2.4}$$

Where:

- $f[x, u, t]$ - Is a function that reproduces the internal dynamics of the system, modeling how the state x evolves, under some input u, at a given time t.
- $g[x, u, t]$ - Is a function that represents the relationship between the output of the system and the state x, input u, at a given time t.

The solution of equation (2.3) describes the evolution of state over time and is defined as

$$x(t) = \phi[t, u_{(t_0,t]}, x(t_0)] \tag{2.5}$$

This is also called *transition function*, and can be further substituted in equation (2.4) to obtain the output of the system as a function of time. These two results represent the cornerstone of any control method. However, the theoretical results regarding solutions to different control problems vary, based on the complexity and non-linearity of f and g.

Based on the form of f and g, two concepts can be introduced:

- **Controllability**. This property denotes the ability to drive the system, from the origin of the state space, to an arbitrary point in the state space, in finite time, using finite control signals. For some special cases (the next section, for example), this principle also applies for steering any initial state to the origin, in finite time.
- **Observability**. This is usually associated with the possibility of determining the initial state of the system, $x(t_0)$, based on the evolution on a finite time interval, knowing the input signals. By doing so, this initial state can be then substituted in equation (2.5), and calculated at any time instant.

Generally, the analysis of these properties is a complex mathematical procedure, especially for highly non-linear system. However, a more specific class of systems, for which many solutions have been obtained over time, are Linear Time-invariant systems (LTIs). Moreover, for this class of systems, the analysis of those properties is reduced to determining whether a matrix has a full rank or not.

2.3 Linear Time-Invariant Systems

The equations which model this class of systems only consist of linear terms (also called state-space systems / state-space representation):

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{2.6}$$
$$y(t) = Cx(t) + Du(t) \tag{2.7}$$

Expressing fundamental properties such as *controllability* or *observability* also becomes easier.

According to [8], for LTI systems, controllability and observability can be expressed with the help of two matrices:

Theorem 2.1 (Controllability)**.** *A linear time-invariant system is said to be controllable if the matrix*

$$R = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix} \tag{2.8}$$

is of rank n, where n is the size of A.

And:

Theorem 2.2 (Observability)**.** *A linear time-invariant system is said to be observable if the matrix*

$$Q = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{2.9}$$

is of rank n, where n is the size of A.

Observability is an essential property when the state is not available for direct measurements. For such observable systems, an auxiliary system can be designed, called “state observer”, which takes the original system’s output and input signals, and provides an estimation of the original system’s internal state. The study of these state observers forms, by itself, another branch of control theory and it is beyond the scope of the work presented in this manuscript. In the sequel, we assume that such an observer has been designed, or that the true state is available for measurement ($y(t) \equiv x(t)$, or, $C = I_n$).

Also, for this class of systems, the state transition equation (2.5) can be expressed as:

$$x(t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)}Bu(\tau)\mathrm{d}\tau \tag{2.10}$$

2.4 The Optimal Control Problem

The central part of an optimal control problem is the performance index. This index is a function used to quantify the performance of the system based on its evolution, over a finite or infinite time horizon. Particularly, the performance indexes invoked in this thesis will be positive definite functions, taking lower values the closer the system’s evolution is to the ideal/desired one.

For a time horizon $(t_0, t_f]$ (t_f can also be infinite), the performance index is usually defined as:

$$J(x_0, t_0, u) = K(x_f, t_f) + \int_{t_0}^{t_f} L[x(\tau), u(\tau), \tau]\mathrm{d}\tau \tag{2.11}$$

with the help of two auxiliary functions. The term $K(x, t)$ is called "terminal cost" and it inflicts a measure of quality for the final state of the system. The integral quantifies the performance of the system during the transition between the initial state and the final state. The term being integrated, $L(x, u, t)$, is called "running cost". Moreover, it is also worth noticing that L also depends on $u(t)$. This way, the running cost can be used to mix state transition performance with control signals optimization. According to section 2.1, signal norms can be used to describe the magnitude of a variable and the nature of its evolution. As control signals are directly associated with the actuators of a control system, the energy, for example, of a control signal is strongly linked to the energy consumed/produced by the actuators, and hence can be associated with overheating of the actuator. By including a Lebesgue norm in the running cost and, consequently, in the cost function, its corresponding real effect will be diminished.

Using the concepts discussed above, the general optimal control problem can be stated as:

$$\begin{aligned} \min_{u \in \mathcal{U}} \quad & J(x_0, t_0, u) \\ \text{s.t.} \quad & \dot{x}(t) = f[x(t), u(t), t] \\ & u \in \Omega \end{aligned} \tag{2.12}$$

Here, Ω is a set defining the possible values for the control signals. In the majority of optimal control problems, the control signal is bounded by a minimum and maximum value, corresponding to physical limitation on the actuators. For the rest of this manuscript, the input signals on every channel are considered to be absolutely bounded by a maximum value: $|u_i(t)| \leq u_{max}$. This maximum value can further be used to scale matrix B in equation (2.6), therefore resulting in a unitary module bounded control signal. The term $\mathcal{U}$ denotes the set of control laws stabilizing the system.

A solution to problem (2.12) is a signal $u^*(t)$, known as "optimal control law". By applying this control law to the system and substituting the resulting state trajectory in the running and terminal cost, the resulting value of the performance index, J^*, will have the lowest value among all other control laws from $\mathcal{U}$, satisfying constraints Ω.

Usual approaches to determining the optimal control are based on the Hamiltonian function. The Hamiltonian is a generalization of the widely used Lagrangian formalism and considered to be a very powerful tool when it comes to functional optimization. In contrast to usual optimization problems, where the objective function is defined on subsets of $\mathbb{R}^n$, the performance index of an optimal control problem is an operator applied to functions of time (such as the state and input vectors). The Hamiltonian function, inspired from the mechanical Hamiltonian formalism, was first introduced in control theory by Lev Pontryagin in 1956 as:

$$\begin{aligned} H(x, p, u) &= L(x, u) + \langle p, f(x, u) \rangle \\ &= L(x, u) + p^\intercal f(x, u) \end{aligned} \tag{2.13}$$

where L is the running cost from (2.11), f is the function describing the system dynamics from (2.3) and $p(t)$ is an auxiliary variable named *costate* or *adjoint state*, as presented in [1]. In order to understand the utility of costates, one must recall the Lagrangian formalism of solving numerical optimization problems. For a constrained optimization problems, the Lagrangian function is introduced as a function depending on the objective variable and on a set of auxiliary variables called Lagrange multipliers. These multipliers act as weighing factors for the terms describing the constraints of the problem. This new function sets the basis of an unconstrained optimization problem and, according to the theory of duality, it can be handled to recover optimal points for the initial problem. Further details on this matter, and any other convex numerical optimization concerns can be found at [3].

As the set of equations describing the system (which are nothing less but a manner of tying the system states to the input signals) is itself a constraint, it is added to the Hamiltonian function, "weighed" by the costate vector. This allows for the definition of an alternative optimization problem using the Hamiltonian. Additional to this, a set of equations is often invoked as being part of a set of necessary conditions for the optimality of a control law, the *canonical set of equations*, or the *Hamiltonian set of equations*:

Definition 2.1. *The following system of differential equations is called the canonical, or Hamiltonian, system:*

$$\begin{aligned} \dot{x} &= \frac{\partial H}{\partial p}(x,p,u) = f(x,u) \\ \dot{p} &= -\frac{\partial H}{\partial x}(x,p,u) = -\frac{\partial L}{\partial x}(x,u) - (\frac{\partial f}{\partial x}(x,u))^{\intercal} p \end{aligned} \tag{2.14}$$

Using the above mentioned concepts, Lev Pontryagin has stated a set of necessary conditions which can be used to test if a control law is optimal or not, for an optimal control problem, such as the one in (2.12). This set of conditions can be extended to sufficiency conditions if the Hamiltonian, along with the system equations and the performance index admit second order variations (similar to second order derivatives for real functions).

Before stating the principle, another important aspect of optimal control problems requires an introduction, that of a terminal set. This is a geometrical notion used to describe the set of points in the state space ($\mathbb{R}^n$) where the final state of the system is allowed to lie in. When target sets are used, the final time (terminal time) for the optimal control problem is defined as the first time instant when $x(t)$ intersects the target set. Target sets can be used to express various control problems such as tracking a target, hitting a point in the space, avoiding certain regions, etc.

Article [12] presents a historical timeline of the most popular methods of dealing with optimality in various theoretical domains, starting with the Brachystochrone and ending with Pontryagin's Maximum Principle.

2.5 Pontryagin's Minimum Principle

This principle extends results from numerical optimization of functions defined on subsets of $\mathbb{R}^n$ to optimal control problems. The main result revolves around optimizing $H(x,p,u)$ as a functional of $u(t)$. As presented in [1], chapter 5, section 13, the principle state that, for a control signal $u^*(t)$ to be an optimum point for the control problem, it is necessary that the following affirmations are true:

1. There exist a solution $(x^*(t), p^*(t))$ to the Hamiltonian set of equations, (2.14), with $x^*(t)$ satisfying initial and final state boundary conditions.

2. Considering the Hamiltonian is now only a functional of $u(t)$, the following equality takes place:

   $$\min_{u\in\Omega} H(x^*(t), p^*(t), u) = H(x^*(t), p^*(t), u^*(t)), \quad \forall t \in [t_0, t_f] \tag{2.15}$$

   This means that, for every time instant t' in the specified interval, if $H(x^*(t'), p^*(t'), u)$ is considered to be a real valued function (of u), it achieves its minimum value for $u = u^*(t')$.

3. The Hamiltonian $H(x^*(t), p^*(t), u^*(t))$ is zero for all t in $[t_0, t_1]$.

$$H(x^*(t), p^*(t), u^*(t)) = 0, \quad \forall t \in [t_0, t_1]$$

In addition to these conditions, it is possible to derive a number of necessary conditions which are based on the geometry of the target set. These additional requirements are linked to the final value of the costate vector and, therefore, provide more boundary conditions for the solution of the Hamiltonian system of equations.

Before the Hands-off Control algorithm will be presented, two specific optimal control problems, along with their solution, will be introduced. Their solution to these two control problems is presented in [1] as a direct application of the Pontryagin minimum principle.

2.6 Minimum Time Control

One of the first optimal control problems to be posed was the *Minimum Time Control* problem. The speed at which a system's output can reach a specific set of values is a natural indicator of its performance. This problem is of interest when control signals are restricted in magnitude, which is, indeed, the case with almost all physical control systems. Otherwise, using an unconstrained input signal, the controllable system could reach any state in an arbitrarily small time.

Thus, the main objective is to make the error between the desired output and the actual output as small as possible, in the shortest time possible. The final error between the actual and output and the reference trajectory should be zero, but some cases also tolerate an ϵ approximation to 0.

Formulated as an optimization problem, the minimum time problem can be defined as:

$$\begin{aligned} \min_{u \in \mathcal{U}} \quad & \int_{t_0}^{T} \mathrm{d}t \\ \text{s.t.} \quad & \dot{x}(t) = f[x(t), u(t), t] \\ & |u_i(t)| \leq 1, \quad i \in \overline{1, m} \\ & (x(T), T) \in \mathcal{S} \end{aligned} \tag{2.16}$$

where $\mathcal{S}$ is the target set. The performance index only contains the integral, which is equal to the horizon length of the control problem. Thus, the minimization is directly applied to the length of the time interval, required to hit the target set. The analysis done in [1] is oriented to the following class of systems:

$$\dot{x}(t) = f[x(t), t] + B[x(t), t]u(t) \tag{2.17}$$

Equation (2.17) differs from (2.3) as the control influence is separated from the free evolution of the system. The state equation is linear in u, making is easier to differentiate and deduce optimal conditions. To be noted is that the differentiation procedure is not the usual one, applied for 1-variable functions. It is a technique implying concepts of variational calculus and functional analysis. If the reader is interested in finding more about the concept, [1], chapter 5, section 6 offers some introductory details, while [9] covers the entire field of functional analysis.

The terms L and K, from (2.11) can be identified as $L[x(t), u(t), t] = 1$ and $K(x(t), t) \equiv 0$. The Hamiltonian of this control problem is defined as:

$$\begin{aligned} H(x(t), p(t), u(t), t] &= L[x(t), u(t), t] + \langle p(t), \dot{x}(t) \rangle \\ &= 1 + \langle p(t), f[x(t), t] + B[x(t), t]u(t) \rangle \\ &= 1 + \langle p(t), f[x(t), t] \rangle + \langle p(t), B[x(t), t]u(t) \rangle \end{aligned} \tag{2.18}$$

where the angled brackets denote the inner product.

As stated earlier, for a control signal to be an optimum point for problem (2.16), there has to exist a solution for the Hamiltonian set of equations, denoted $(p^*(t), x^*(t))$. Moreover, the input signal $u^*(t)$ has to minimize the Hamiltonian defined above, for every time instant. This means that:

$$\min_{u\in\Omega} H(x^*(t), p^*(t), u, t) = H(x^*(t), p^*(t), u^*(t), t), \quad \forall t \in [t_0, T] \tag{2.19}$$

Explicitly, this means that:

$$1 + \langle p^*(t), f[x^*(t), t]\rangle + \langle p^*(t), B[x^*(t), t]u^*(t)\rangle \leq 1 + \langle p^*(t), f[x^*(t), t]\rangle + \langle p^*(t), B[x^*(t), t]u(t)\rangle \tag{2.20}$$

for any $u(t)$ in the admissible controls set. The first two terms on every side of the inequality are the same, so the relation can be re-written as:

$$\langle p^*(t), B[x^*(t), t]u^*(t)\rangle \leq \langle p^*(t), B[x^*(t), t]u(t)\rangle \tag{2.21}$$

By expanding the inner product and separating the terms (operation possible as there are not any constraints coupling input signals between input channels), the following problem is obtained:

$$\min_{|u_k(t)|\leq 1} u_k(t) f_k(t) \tag{2.22}$$

Where $f_k(t)$ is composed out of rows from matrix $B(x(t), t)$ and the costate. The minimum is achieved when $u_k(t)$ has maximum magnitude and the opposite sign of $f(t)$, and the minimum value will be:

$$\min_{|u_k(t)|\leq 1} u_k(t) f_k(t) = -|f_k(t)| \tag{2.23}$$

obtained when:

$$u_k^*(t) = -\text{sgn}\left[f_k(t)\right] \quad = -\text{sgn}\left[\sum_{j=1}^{n} b_{j,k}[x^*(t), t] p_j^*(t)\right] \tag{2.24}$$

Therefore, $u_k^*(t)$ will be a function, taking the only two values, ± 1, based on the sign of $f_k(t)$, also being called "Bang-Bang" control. This can be physically interpreted as turning the actuators on, at full power, in the most favourable way which facilitates driving the system from the initial state to the origin. This behavior is also rather intuitive, as a short response time (usually) implies a significant control effort. This definition, however, raises some problems when $f_k(t)$ is zero. If this occurs only for time intervals of null Lebesgue measure, $u_k^*(t)$ will only switch its polarity during these instants. However, if $f_k(t)$ is identically 0 for a time interval of positive Lebesgue measure, the problem given in (2.22) does not have a unique solution, and $u_k^*(t)$ cannot be deduced based on this result. Such a problem is also referred to as **singular**. Otherwise, the problem is called **normal**.

Normality and **singularity** are properties which depend on the dynamics of the system. The more complex the state equations are, the harder it is to determine if the minimum time problem is normal or not. For LTI systems, however, it has been proved that a time optimal problem is normal if and only if the system is controllable.

Based on the obtained results above, for a specific target set, the control law can be deduced by exploiting the state equations. An LTI system with two states, measured as outputs, and one input will be used in the hardware implementation chapter and, for this system, a complete description of the minimum time control can be obtained by using the explicit form of the state trajectory, as a function of $u(t)$, and by taking into consideration the expression of the optimal control solution, which is know *a-priori*.

2.7 The L^1 Optimal Control

This optimal control problem finds its origins in quantifying fuel consumption using the Lebesgue L^1 norm of the input signals. This is especially the case when $u(t)$ can be correlated to the rate at which resources used by the actuators are expelled from the system. Thus, a control law of high absolute values over large time intervals can lead to a complete depletion of resources (or fuel). The L^1 optimal control problem searches for the control law with the lowest L^1 norm of the control signal, which assures the intersection between the state trajectory and a specified target set.

For a system with m inputs, the performance index to be minimized is:

$$J(u) = \sum_{i=1}^{m} \lambda_i \int_{t_0}^{T} |u_i(t)| \,\mathrm{d}t \quad = \int_{t_0}^{T} \sum_{i=1}^{m} \lambda_i \, |u_i(t)| \,\mathrm{d}t \tag{2.25}$$

Here, T can either be a predefined time value (if the control problem is fixed-time), or a variable time moment. The terms λ are weighing factors, signalling which input channels may be more relevant to the fuel optimization problem.

However, as it will be discussed in the following section, only the fixed-time problem is of interest. Therefore, the general L^1 optimal control Problem can be stated as:

$$\begin{aligned} \min_{u\in\mathcal{U}} \quad & \int_{t_0}^{T} \sum_{i=1}^{m} \lambda_i \, |u_i(t)| \,\mathrm{d}t \\ \text{s.t.} \quad & \dot{x}(t) = f[x(t),t] + B[x(t),t]u(t) \\ & |u_i(t)| \leq 1, \quad i \in \overline{1,m} \end{aligned} \tag{2.26}$$

The analysis of this problem is very similar to the one that has been done for the Minimum Time Problem. Firstly, the Hamiltonian for the optimal control problem can be defined as:

$$\begin{aligned} H(x(t),p(t),u(t),t] &= L[x(t),u(t),t] + \langle p(t), \dot{x}(t) \rangle \\ &= \sum_{i=1}^{m} \lambda_i |u_i(t)| + \langle p(t), f[x(t),t] + B[x(t),t]u(t) \rangle \\ &= \sum_{i=1}^{m} \lambda_i |u_i(t)| + \langle p(t), f[x(t),t] \rangle + \langle p(t), B[x(t),t]u(t) \rangle \end{aligned} \tag{2.27}$$

Using this expression, the Pontryagin minimum principle states that for a control signal $u^*(t)$ to be an optimum point for (2.26), it is necessary for the canonical set of equations (2.14) to have a solution, denoted $(x^*(t), p^*(t))$, which satisfies the boundary conditions $x^*(t_0) = x_0$, $(x^*(T), T) \in \mathcal{S}$, along with any other conditions that $\mathcal{S}$ may introduce on $p^*(T)$. It is also necessary that the Hamiltonian, evaluated as a real function of u:

$$H(x^*(t), p^*(t), u, t) = \sum_{i=1}^{m} \lambda_i |u_i(t)| + \langle p^*(t), f[x(t),t] \rangle + \langle p^*(t), B[x^*(t),t]u(t) \rangle \tag{2.28}$$

attains its minimum value, at any time instant t, for $u(t) = u^*(t)$.

$$\min_{u\in\Omega} H(x^*(t), p^*(t), u, t) = H(x^*(t), p^*(t), u^*(t), t), \quad \forall t \in [t_0, T] \tag{2.29}$$

Similar to the minimum time case, the previous equation can be expanded using the definition of H. Terms which do not depend on $u(t)$ cancel out, leading to:

$$\sum_{i=1}^{m} \lambda_i |u_i^*(t)| + \langle p^*(t), B[x^*(t), t] u^*(t) \rangle \leq \sum_{i=1}^{m} \lambda_i |u_i(t)| + \langle p^*(t), B[x^*(t), t] u(t) \rangle \tag{2.30}$$

for any time instant. Using the definition of the inner product, the latter inequality can also be reformulated as:

$$\sum_{i=1}^{m} \lambda_i |u_i^*(t)| + \sum_{k=1}^{m} u_k^*(t) \sum_{j=1}^{n} b_{j,k}[x^*(t), t] p_j^*(t) \leq \sum_{i=1}^{m} \lambda_i |u_i(t)| + \sum_{k=1}^{m} u_k(t) \sum_{j=1}^{n} b_{j,k}[x^*(t), t] p_j^*(t) \tag{2.31}$$

And by grouping the terms under the same summation operator, we obtain:

$$\sum_{i=1}^{m} \left(\lambda_i |u_i^*(t)| + u_i^*(t) \sum_{j=1}^{n} b_{j,i}[x^*(t), t] p_j^*(t) \right) \leq \sum_{i=1}^{m} \left(\lambda_i |u_i(t)| + u_i(t) \sum_{j=1}^{n} b_{j,i}[x^*(t), t] p_j^*(t) \right) \tag{2.32}$$

Using the same notation as in the case of the Minimum Time Problem, this inequality is equivalent to:

$$\min_{u \in \Omega} \sum_{i=1}^{m} \lambda_i |u_i(t)| + u_i(t) f_i(t) = \sum_{i=1}^{m} \lambda_i |u_i^*(t)| + u_i^*(t) f_i(t) \tag{2.33}$$

As, once again, there is no form of correlations between the control signals for any time instant, the minimum operator can be interchanged with the sum, giving:

$$\begin{aligned} \min_{u \in \Omega} \sum_{i=1}^{m} \lambda_i |u_i(t)| + u_i(t) f_i(t) &= \sum_{i=1}^{m} \min_{u \in \Omega} \left[\lambda_i |u_i(t)| + u_i(t) f_i(t) \right] \\ &= \sum_{i=1}^{m} \min_{u \in \Omega} \left[\lambda_i \left(|u_i(t)| + \frac{u_i(t) f_i(t)}{\lambda_i} \right) \right] \\ &= \sum_{i=1}^{m} \lambda_i \min_{u \in \Omega} \left[|u_i(t)| + \frac{u_i(t) f_i(t)}{\lambda_i} \right] \end{aligned} \tag{2.34}$$

This identity holds for every time moment. Therefore, this is equivalent to finding the minimum value of the real-valued function $g(x) = |x| + ax$ at every time instant, where a is a real constant. Recalling the input constraints, Ω is defined by the hyperbox $|x| \leq 1$. The function, therefore, admits the following branched definition:

$$g(x) = \begin{cases} (1+a)x & x \in [0, 1] \\ (-1+a)x & x \in [-1, 0) \end{cases}$$

In order to find the minimum of this function, a has to be taken into consideration. Depending on a, 4 cases can be distinguished:

- $a > 1$ - $g(x)$ will be strictly increasing for all $x \in [-1, 1]$, therefore having $x = -1$ as its absolute minimum
- $a < -1$ - $g(x)$ will be strictly decreasing for all $x \in [-1, 1]$, therefore having $x = 1$ as its absolute minimum
- $a \in (-1, 1)$ - $g(x)$ will have opposite monotony on the two halves of $[-1, 1]$ (decreasing on $[-1, 0)$, increasing on $[0, 1]$), therefore attaining a minimum value for $x = 0$
- $a = \pm 1$ - $g(x)$ will be identically 0 on one half of the $[-1, 1]$ interval. This is the case where it is impossible to decide the minimum argument of $g(x)$

Having analyzed this problem, it is easy now to find the minimum argument of (2.34). This can be done with the help of an operator called the "Dead-zone function":

$$\mathrm{D}_a(x) = \begin{cases} -1 & x \in (-\infty, -a) \\ 0 & x \in (-a, a) \\ 1 & x \in (a, \infty) \\ \in [-1, 0] & x = -a \\ \in [0, 1] & x = a \end{cases} \tag{2.35}$$

where a is a positive number. The L^1 Optimal Control therefore becomes:

$$u_i^*(t) = \mathrm{D}_1\left(\frac{f_i(t)}{\lambda_i}\right) = \mathrm{D}_{\lambda_i}\left(\sum_{j=1}^{n} b_{j,k}[x^*(t), t]p_j^*(t)\right) \tag{2.36}$$

resulting in a three-state control law called the "Bang-off-Bang" control. Nevertheless, this definition also encounters difficulties whenever $\sum_{j=1}^{n} b_{j,k}[x^*(t), t]p_j^*(t)$ is zero over a time interval of non-zero length (in such cases, the problem is called **singular**; otherwise, it is deemed **normal**). For LTI systems, a fixed-time L^1 Control problem, with $\mathcal{S} = (0, T)$ as target set, has a solution (under the assumption that it is normal), if:

- The system is controllable
- The fixed terminal time T is greater than the minimum time T^* required to drive the initial state x_0 to the origin.

2.8 The L^0 Optimal Control

The work presented in [10] proposes a different approach to the fuel optimization problem, by modifying the performance index. Instead of using the L^1 norm of the control signal, it intends to minimize a different indicator of the fuel consumption, that of the time support of the input signals. By minimizing the total length of the time intervals during which the actuators are turned on, the sparsest control law is obtained. In order to accomplish this objective, the authors proposed a new performance index which extends the definition of the Lebesgue L^p norm to the limit case where $p = 0$. Thus, the following pseudo-norm is obtained:

$$||u||_0 = \int_{supp(u)} \mathrm{d}t \tag{2.37}$$

When testing this new operator for the norm axioms, it is easy to see that it does not verify the absolute homogeneity axiom. For any non-zero complex constant, α, $\alpha u(t)$ has the same time support as $u(t)$, therefore $||\alpha u||_0 = ||u||_0$. The L_0 pseudo-norm can also be defined as the Lebesgue measure of the set:

$$\mathcal{T} = \{t \geq 0 \mid u(t) \neq 0\} \tag{2.38}$$

The performance index describing the optimal control problem becomes:

$$J(u) = \sum_{i=1}^{m} \lambda_i ||u_i||_0 = \sum_{i=1}^{m} \lambda_i \int_{supp(u_i)} \mathrm{d}t \tag{2.39}$$

It is easy to notice that the usual technique of solving the optimal control problem using the Hamiltonian and the minimum principle of Pontryagin becomes a very complex procedure, as the exact expression of $L(x, u, t)$ is very difficult to deduce. Instead, the authors makes use of the L^1 Optimal control properties and prove the equivalence between the two optimal control problems (L^0 and L^1) under normality assumptions.

For a fixed time horizon optimal control problem, with the target set $\mathcal{S} = (0, T)$, the L^1 problem has a unique solution if the initial problem is normal (which, for LTI systems, is equivalent to controllability) and T is greater than the the minimum time corresponding to the initial state x_0 of the system. The set of solutions to the L^1 problem can be denoted as $\mathcal{U}_1^*(T, x_0)$, and is not empty. The performance index from (2.25) will be denoted as $J_1(u)$, and the one from (2.39) will be denoted as $J_0(u)$. Moreover, the set of all admissible control laws (which satisfy the constraints and steer the system from an initial state x_0 to the origin in time T) is denoted by $\mathcal{U}(T, x_0)$. Then, for every $u \in \mathcal{U}(T, x_0)$:

$$\begin{aligned} J_1(u) &= \sum_{i=1}^{m} \lambda_i \int_{t_0}^{T} |u_i(t)| \, \mathrm{d}t = \sum_{i=1}^{m} \lambda_i \int_{supp(u_i)} |u_i(t)| \, \mathrm{d}t \\ &\leq \sum_{i=1}^{m} \lambda_i \int_{supp(u_i)} \mathrm{d}t \\ &= J_0(u) \end{aligned} \tag{2.40}$$

The inequality is caused by the time intervals for which $|u(t)| < 1$. This means that for every admissible control u, the system will perform better in the L^1 control case, rather than in the L0 control one.

$$J_1(u) \leq J_0(u), \quad \forall u \in \mathcal{U}(T, x_0) \tag{2.41}$$

Now, for any u_1^* in $\mathcal{U}_1^*(T, x_0)$, knowing from the previous section that this command signal is a three-state control law: $u_1^*(t) \in \{-1, 0, 1\}$, its time support, $supp(u^*)$, will only include the intervals where $u^\star(t) \neq 0$.

Therefore:

$$J_1(u_1^*) = \sum_{i=1}^{m} \lambda_i \int_{t_0}^{T} |u_{1i}^*(t)| \, \mathrm{d}t = \sum_{i=1}^{m} \lambda_i \int_{supp(u_{1i}^*)} |u_{1i}^*(t)| \, \mathrm{d}t = J_0(u_1^*) \tag{2.42}$$

Equation (2.41) suggests that as a function of u, J_0 is always lower bounded by J_1. Equation (2.42) points out that the only arguments for which those two functions intersect are the elements of $\mathcal{U}_1^*(T, x_0)$, which is included in $\mathcal{U}(T, x_0)$. This means that the elements of the set $\mathcal{U}_1^*(T, x_0)$ minimize J_0, over the entire set $\mathcal{U}(T, x_0)$.

Thus, the set of optimal solutions of the L0 control problems, $\mathcal{U}_0^*(T, x_0)$, includes $\mathcal{U}_1^*(T, x_0)$:

$$\mathcal{U}_1^*(T, x_0) \subseteq \mathcal{U}_0^*(T, x_0) \tag{2.43}$$

In order to prove the reversed inclusion, the equality deduced earlier, (2.42) is used. Additionally, it is known that u_1^* minimizes J_1, therefore:

$$J_1(u_1^*) \leq J_1(u_0^*) \tag{2.44}$$

Equation (2.41), applied for elements in $\mathcal{U}_0^*(T, x_0)$, holds that:

$$J_1(u_0^*) \leq J_0(u_0^*) \tag{2.45}$$

But u_0^* is a minimizer of J_0, hence:

$$J_0(u_0^*) \leq J_0(u_1^*) \tag{2.46}$$

Then:

$$J_1(u_0^*) \leq J_0(u_1^*) \tag{2.47}$$

So, $J_1(u_1^*) \leq J_1(u_0^*)$, and $J_1(u_0^*) \leq J_0(u_1^*)$. But, $J_0(u_1^*) = J_1(u_1^*)$, thus $J_1(u_1^*) \leq J_1(u_0^*) \leq J_1(u_1^*)$ which results in:

$$J_1(u_1^*) = J_1(u_0^*) \tag{2.48}$$

Therefore, u_0^* also minimizes J_1 and $u_0^* \in \mathcal{U}_1^*(T, x_0)$.

$$\mathcal{U}_0^*(T, x_0) \subseteq \mathcal{U}_1^*(T, x_0) \tag{2.49}$$

From (2.43) and (2.49):

$$\mathcal{U}_1^*(T, x_0) = \mathcal{U}_0^*(T, x_0) \tag{2.50}$$

The last result proves the equivalence between the two problems. Hence, when computing the sparsest control law, the L^1 norm can be used. This remarkable result, as it will be presented in the following section, has been used in the same article, [10], to design a control algorithm which ensures stability, under some assumptions regarding the disturbances acting on the system, at the cost of a control law with an arbitrarily small sparsity rate.

This itself is a remarkable mathematical result, which can be used to ensure stability of a control system with an arbitrarily sparse input signal.

2.9 Self-Triggering Hands-Off Control

In order to quantify the performance of this algorithm, the notion of "Sparsity rate" needs to be introduced. Taking the L^0 pseudo-norm of a signal on a fixed time interval and dividing it by the length of the time interval, the sparsity rate of the signal is obtained. It measures the percentage of time during which the signal was non-identically zero.

$$\mathcal{R}_T(u) = \frac{||u||_0}{T} \tag{2.51}$$

If the signal $u(t)$ is interpreted as the actuator effort, then the previously introduced operator indicates the percentage of time for which the actuators were turned on.

The main idea of this algorithm is to exploit the characteristics of an L^1 optimal control law. More specifically, the property of it being also the sparsest admissible control law is employed. Nonetheless, the conditions under which this problem has a solution are essential to the development of this algorithm. The normality property can be ensured by picking an appropriate LTI model, for which the pair (A, B) is controllable. The other condition is linked to the length of the time horizon for the optimal control problem. As previously stated, this horizon has to be larger than the minimum one. It is clear now why both problems have been analyzed in this chapter, as both contribute to the design of this algorithm. As presented in [10], the proposed control scheme is:

Algorithm 2.1: Self-triggering Hands-off Control

```
Input : System dynamics $(f[x(t), u(t), t], B[x(t), t])$ and the desired sparsity rate r
1 $k = 0$
2 while 1 do
3     $x_k = measure\_current\_state()$ // Measure initial state
4     $T_k = Min\_Time(x_k)$
       // Compute the minimum time corresponding to the current state
5     $T_k = max(T_k/r, T_{safe})$ // Compute the horizon for L1 problem
6     $u_k = L1(x_k, T_k)$ // Solve L1 problem
7     $Send\_control\_to\_actuators(u_k)$ // Apply control law to the system
8     $k = k + 1$
9 end
```

The above-mentioned workflow ensures that, during the algorithm's runtime, the actuators are being powered for a maximum of $r\%$ of the total functioning time. Line 4 computes the minimum time required to drive the system to the final state, from the current one. Line 5 expands this time horizon by dividing by r. The algorithm also includes a safety mechanism, which prevents the algorithm from trying to compute L^1 problems on time intervals of a few sample periods length. For such short time intervals, the problem can become ill posed, as the natural switching times for the L^1 problem will more likely be situated between samples. Numerically, however, this is not possible, and the control will only switch at fixed sampling instants. In order to prevent such scenarios, the result of the minimum time solver is compared with a fixed minimum length of the control problem time horizon. The maximum between those two is further provided as an input to a fixed-time horizon L^1 problem solver. The L^1 solution, as it has been shown earlier, is equivalent to the L^0 solution, thus having the smallest time support (L^0 pseudo-norm). An interesting fact is that, between two iterations of the algorithm, the system behaves as if it were in open-loop configuration, with a scheduled input signal.

Based on a result which will be stated shortly, the sparsity rate of one iteration is a key factor for computing the sparsity rate for the entire execution time of the algorithm. Using the output of line 4, line 5 will compute the time horizon for the iteration in one of the two following cases:

- $T_k = T_{min}(x_k)/r$ - in this scenario, an admissible control is $u^*(t)$, equal to the minimum time control for $t \in [0, T_{min}(x_k))$, and 0 for $[T_{min}(x_k), T_k]$. Hence, $||u^*||_0 = T_{min}(x_k)$. Obviously, the L^1 optimal control, u_k, will have the lowest L^0 pseudo-norm, among all other controls, including $u^*(t)$. Consequently:

$$R_{T_k}(u_k) = \frac{||u_k||_0}{T_k} \leq \frac{T_{min}(x_k)}{T_{min}(x_k)/r} = r \tag{2.52}$$

- $T_k = T_{safe}$ - similarly, an admissible control is $u^*(t)$, defined on $[0, T_{safe})$, where $u^*(t) \neq 0$ only for $t \leq T_{min}(x_k)$. Therefore, $||u^*||_0 = T_{min}(x_k) \geq ||u_k||_0$. Knowing that $T_{safe} \geq T_{min}(x_k)/r$:

$$\begin{aligned} R_{T_k}(u_k) &= \frac{||u_k||_0}{T_k} = \frac{||u_k||_0}{T_{safe}} \\ &\leq \frac{||u_k||_0}{T_{min}(x_k)/r} \\ &\leq \frac{T_{min}(x_k)}{T_{min}(x_k)/r} \\ &\leq r \end{aligned} \tag{2.53}$$

Concluding, each iteration will generate a control law with a sparsity rate lower than r. Definition (2.51) can be extended for infinite time horizons. Supposing the algorithm runs for an indefinitely long period of time, the following lemma, introduced in [10], provides the result which proves the utility of this control algorithm.

Lemma 2.1. *Let* $\boldsymbol{u}$ *be a measurable signal defined on* $[0, \infty)$. *If there is a time sequence* $t_{k+1} = t_k + T_k$, *where* $t_0 = 0$ *and* $T_k > 0$, *for which* $R_{T_k}(u_k) \leq r$, *then:*

$$R_\infty(u) \leq r \tag{2.54}$$

Hence, as long as the algorithm is running, the actuators will be powered for a percentage of time of at most r.

2.10 Stability Guarantees

In the absence of external or internal disturbance or model uncertainty, the use of a L^1 optimal control law guarantees the convergence of $x(t)$ to the origin of the state space, thus ensuring stability. However, in the presence of any *persistent* disturbance, a steady state error equal to 0 is impossible to achieve. The authors' focus is oriented on a slightly different concept called practical stability. This property implies that the state will be located inside a ball centered at the origin of the state space, of radius h, for a bounded disturbance, $||d(t)||_\infty \leq \delta$, at any time instant t.

The constant h, bounding the state norm at each time instant, varies with respect to various parameters of the system, and the algorithm. For example, consider the open loop system is a stable LTI system with a degree of stability described by the following equation:

$$||x(t, x_0)|| \leq \beta(x_0) e^{-\alpha t} ||x_0|| \tag{2.55}$$

The closed loop system state, under the presence of persistent and stochastic disturbances, will be driven very close to zero. However, due to the presence of $d(t)$, the steady state error, between the state and the origin, will not converge to 0. Practical stability, however, guarantees that the state trajectory will lie in a ball of radius h, centered in the origin. This radius decreases as h increases.

Another important factor which influences the degree of practical stability (h) is the maximum sparsity rate r. An imposed maximum sparsity rate close to 0 would imply very little external influence to the system. This would lead to increased savings of energy (fuel) but would drastically degrade the performances, as the scaling factor, r^{-1} would become very large in magnitude. This would cause very long time intervals during which the system evolves in an open-loop configuration, and could cause large deviations from the origin of the state space. The safety factor, T_{min}, also impacts the practical stability achieved by the algorithm. This impact of this constant can be observed when the state is close to the origin, as the minimum time corresponding to its state would be very small, even scaled with the inverse of r. Thus, the safety mechanism would assign T_k the value T_{min}, again leading to an open loop-behavior for a comparatively long time.

3 Numerical Implementations

This chapter presents how the previously introduced notions have been used in order to develop numerical algorithms. These procedures can be implemented (for example, in MATLAB) in order to obtain an environment capable of solving a Hands-off Control problem.

As presented in [10], the Hands-off Control solution relies on solving two optimal control problems: the Minimum Time Control subproblem and L^1 Optimal Control subproblem. Those two, along with their proposed solutions, are discussed in the following two sections.

3.1 Minimum Time Problem

An usual approach to the minimum time problem is based on null-controllable regions. A state is said to be null-controllable in time T if there exist a control law $u(t)$, in the set of admissible commands $\mathcal{A}_u$, which steers the state to the origin. The null-controllable region at time T is the set containing all the null-controllable state in time T. As stated in [6], the null-controllable region in time T can be proved to be given by:

$$\mathcal{NC}(T) = \left\{ \int_0^T e^{-A\tau} Bu(\tau)d\tau \;\middle|\; u(t) \in \mathcal{A}_u \right\}, \tag{3.1}$$

where $\mathcal{A}_u$ is a symmetric set of admissible control signal values.

Naturally, as T increases, $\mathcal{NC}(T)$ will also increase in size, giving the following inclusion:

$$\mathcal{NC}(T_1) \subseteq \mathcal{NC}(T_2), \quad \forall T_1, T_2 \quad T_1 \leq T_2 \tag{3.2}$$

This can be interpreted as: “if x_0 can be driven to 0 in time T, it can also be driven in any time greater than T”. Therefore, the minimum time problem can be solved by finding the minimum T for which $x_0 \in \mathcal{C}(T)$, and $x_0 \notin \mathcal{C}(\widetilde{T})$ for any other $\widetilde{T} < T$.

In what follows, the previous concepts will be applied to a discrete-time representation of the state-space system from (2.6). This approximation of the initial continuous-time system can be obtained through the process of discretization, using an appropriately chosen sampling period T_s. The inclusion of x_0 in $\mathcal{C}(T)$ is tested using the discrete time state equation:

$$x_{k+1} = A_d x_k + B_d u_k \tag{3.3}$$

By subsequent substitutions of x_k in the equation above with k ranging from 1 to N-1, the state at sample time N can be written as:

$$x_N = A_d^N x_0 + \begin{bmatrix} A_d^{N-1}B_d & A_d^{N-2}B_d & \dots & A_dB_d & Bd \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \tag{3.4}$$

For simplicity, the block matrices can be denotes as:

$$E = \begin{bmatrix} A_d^{N-1}B_d & A_d^{N-2}B_d & \dots & A_dB_d & B_d \end{bmatrix} \tag{3.5}$$

and

$$\mathcal{U} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} \tag{3.6}$$

Using this definition of x_N, the following optimization problem has been defined:

$$\begin{aligned} \min_{\mathcal{U}\in\mathbb{R}^{Nm}} \quad & ||x_N - x_f||_2^2 = ||A_d^N x_0 + E\mathcal{U} - x_f||_2^2 \\ \text{s.t.} \quad & |\mathcal{U}_{im+j}| \leq 1, i = \overline{0, N-1}, \\ & \qquad\qquad\quad j = \overline{1, m} \end{aligned} \tag{3.7}$$

Where x_f is the desired final state. The norm in (3.7) can be further written as:

$$\begin{aligned} ||x_N - x_f||_2^2 &= ||A_d^N x_0 + E\mathcal{U} - x_f||_2^2 \qquad (F = A_d^N - x_f) \\ &= (F + EU)^\top (F + E\mathcal{U}) \\ &= \frac{1}{2} U^\top Q \mathcal{U} + q^\top U + c \end{aligned} \tag{3.8}$$

where

$$Q = 2E^\top E \tag{3.9}$$
$$q = EF^\top \tag{3.10}$$
$$c = F^\top F \tag{3.11}$$

The resulting optimization problem is:

$$\begin{aligned} \min_{\mathcal{U}\in\mathbb{R}^{Nm}} \quad & \frac{1}{2}\mathcal{U}^\top Q \mathcal{U} + q^\top \mathcal{U} \\ \text{s.t.} \quad & |\mathcal{U}_{im+j}| \leq 1, i = \overline{0, N-1}, \\ & \qquad\qquad\quad j = \overline{1, m} \end{aligned} \tag{3.12}$$

This is a standard Quadratic Programming (QP) problem, with the objective variable being subject to box constraints. After finding the minimum point $\mathcal{U}^*$ of (3.12), it can be substituted in (3.8) in order to find the minimum value of the norm. This value is used to decide if the final state was achieved with the optimal control law. If the value is very close to 0 (can be compared with an ϵ), then the system can be driven to state x_f successfully.

Having this numerical procedure defined, given a state x_0 and a time horizon length T, using the reasoning introduced above, the following two cases can be deduced:

1. x_0 can be driven to the origin in time T using an admissible control sequence
2. x_0 cannot be driven to the origin in time T using an admissible control sequence

The first case leads to the conclusion that the minimum time is not greater than T, while the second case points out that T is smaller than the minimum time. By further increasing or decreasing T, a binary-search algorithm results, capable of computing the minimum time corresponding to a initial state x_0.

3.2 L^1 Problem

Recall the fixed-time L^1 Optimal Control problem for continuous LTI systems, with terminal state constraints:

$$\begin{aligned} \min_{u \in \mathcal{U}} \quad & \int_{t_0}^{T_f} \sum_{i=1}^{m} \lambda_i \left| u_i(t) \right| \mathrm{d}t \\ \text{s.t.} \quad & \dot{x}(t) = Ax(t) + Bu(t) \\ & |u_i(t)| \leq 1, \quad i \in \overline{1, m} \quad t \in [t_0, T_f] \\ & x(T_f) = x_f \end{aligned} \tag{3.13}$$

Using the same discrete representation from (3.3) and the definitions of the final state and matrices E and F, the optimization problem can be rewritten as:

$$\begin{aligned} \min_{\mathcal{U} \in \mathbb{R}^{Nm}} \quad & \sum_{i=0}^{N-1} \sum_{j=1}^{m} \lambda_j \left| \mathcal{U}_{im+j} \right| \\ \text{s.t.} \quad & E\mathcal{U} + F = 0 \\ & |\mathcal{U}_{im+j}| \leq 1, \quad i = \overline{0, N-1}, \\ & \qquad\qquad\qquad j = \overline{1, m} \end{aligned} \tag{3.14}$$

In order to obtain a linear programming (LP) formulation, an additional set of variables is introduced, called slack variables. Those variables allow the reformulation of the cost function and the constraints as follows:

$$\begin{aligned} \min_{\begin{bmatrix} \mathcal{U} \\ \mathcal{T} \end{bmatrix} \in \mathbb{R}^{2Nm}} \quad & \begin{bmatrix} \mathcal{U}^\top & \mathcal{T}^\top \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{f} \end{bmatrix} \\ \text{s.t.} \quad & E\mathcal{U} + F = 0 \\ & -\mathcal{T} \leq \mathcal{U} \leq \mathcal{T} \\ & 0 \leq \mathcal{T} \leq 1 \end{aligned} \tag{3.15}$$

Vector $\mathbf{f}$ contains the weighting factors λ_i as:

$$\mathbf{f} = \begin{bmatrix} \mathbf{f}_\lambda \\ \mathbf{f}_\lambda \\ \vdots \\ \mathbf{f}_\lambda \end{bmatrix}, \quad \mathbf{f} \in \mathbb{R}^{N*m} \tag{3.16}$$

$$\mathbf{f}_\lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_m \end{bmatrix}, \quad \mathbf{f}_\lambda \in \mathbb{R}^{m} \tag{3.17}$$

The inequality constraints can be written as a matrix inequality:

$$A_{ineq}\begin{bmatrix}\mathcal{U}\\ \mathcal{T}\end{bmatrix} \leq b_{ineq}$$
$$A_{ineq} = \begin{bmatrix}-\mathcal{I}_{Nm} & \mathcal{I}_{Nm}\\ -\mathcal{I}_{Nm} & -\mathcal{I}_{Nm}\\ 0_{Nm} & \mathcal{I}_{Nm}\\ 0_{Nm} & -\mathcal{I}_{Nm}\end{bmatrix} \tag{3.18}$$
$$b_{ineq} = \begin{bmatrix}0_{Nm\times 1}\\ 0_{Nm\times 1}\\ 1_{Nm\times 1}\\ 0_{Nm\times 1}\end{bmatrix}$$

Resulting in:

$$\begin{aligned}\min_{\substack{\mathcal{U}\in\mathbb{R}^{N*m}\\ \mathcal{T}\in\mathbb{R}^{N*m}}} \quad & \begin{bmatrix}\mathcal{U}^\top & \mathcal{T}^\top\end{bmatrix}\begin{bmatrix}\mathbf{0}\\ \mathbf{f}\end{bmatrix}\\ \text{s.t.} \quad & \begin{bmatrix}E & 0^\top\end{bmatrix}\begin{bmatrix}\mathcal{U}\\ \mathcal{T}\end{bmatrix} = -F\\ & -A_{ineq}\begin{bmatrix}\mathcal{U}\\ \mathcal{T}\end{bmatrix} \leq b_{ineq}\end{aligned} \tag{3.19}$$

The latter form is a standard Linear Programming (LP) problem, with the objective variable being subject to box and equality constraints.

The two resulting optimization problems (QP and LP) represent the two main subproblems in every iteration of the Hands-off Control algorithm. For testing purposes, the two procedures have been implemented in MATLAB, initially, using in-built optimization problem solvers.

3.3 The Satellite Model

The model for which the hands-off control algorithm, along with the two subproblems (minimum time and L^1 control), have been implemented is a simplified LTI system modeling the translation dynamics of an orbiting satellite. This system is based on a model of an one unit (1U) CubeSat (a satellite that is roughly cubical in shape), equipped with a set of thrusters which are capable of moving the satellite on all 3 axes. The state equations describing the translational dynamics are:

$$\dot{x} = Ax + Bu$$
$$x = \begin{bmatrix}\dot{x_1}\\ \dot{x_2}\\ \dot{x_3}\\ x_1\\ x_2\\ x_3\end{bmatrix} \tag{3.20}$$
$$A = \begin{bmatrix}0_3 & 0_3\\ I_3 & 0_3\end{bmatrix}$$
$$B = \begin{bmatrix}\frac{F_{max}}{M}I_3\\ 0_3\end{bmatrix}$$

Here, x_i is the distance relative to a reference system situated on the Earth's orbit, with a predefined fixed orientation, and $\dot{x_i}$ is the speed, with the subscript indicating that these variables are associated with the i^{th} axis. The term F_{max} represents the maximum force that a pair of thrusters can exert on one axis, and M is the mass of the entire system. The discrete-time counterpart for system (3.20) is:

$$\begin{aligned} x_{k+1} &= A_d x_k + B_d u_k \\ A_d &= \begin{bmatrix} I_3 & 0_3 \\ k_a I_3 & 0_3 \end{bmatrix} \\ B_d &= \begin{bmatrix} k_{b_1} I_3 \\ k_{b_2} I_3 \end{bmatrix} \end{aligned} \tag{3.21}$$

This discrete-time model has been obtained by first considering that the inputs of the continuous-time system have Zero-Order Hold (ZOH) extrapolators attached to them, and then using the eponymous ZOH discretization procedure. Thus, assuming the same initial conditions and the same sets of inputs, the state variables of the discretized system will match those of the continuous-time system with ZOH-endowed inputs, allowing us to state that

$$x_k = x(kT_s) \tag{3.22}$$

The constants k_a, k_{b_1} and k_{b_2}, appearing in the matrices A_d and B_d will depend on the sampling period. For example, with a sampling period of $T_s = 6.2832$ and considering that the satellite has a total mass of 1Kg, while being equipped with a pair of thrusters which are capable of generating a maximum force of $1.4mN$ on each axis, the model's parameters are equal to:

$$\begin{cases} k_a = 6.2832 \\ k_{b_1} = 0.0088 \\ k_{b_2} = 0.0276 \end{cases} \tag{3.23}$$

The size and structure of matrices A_d and B_d allow the computation of matrix E from (3.5) in a much simple manner. The exact expression of A_d^N can be deduced through inductive reasoning, which yields:

$$A_d^N = \begin{bmatrix} I_3 & 0_3 \\ Nk_a & I_3 \end{bmatrix} \tag{3.24}$$

along with the matrix product:

$$A_d^N B_d = \begin{bmatrix} k_{b_1} I_3 \\ (Nk_a k_{b_1} + k_{b_2}) I_3 \end{bmatrix} \tag{3.25}$$

In this way, computations which include the matrix E can use these identities in order to calculate results directly, without the need to store the pre-computed value of E, thus saving up memory. We conclude this section by pointing out that, for the system described in (3.21), each pair of states consisting of $(x_i, \dot{x}_i)$ is decoupled from the dynamics of the other two pairs. Therefore, each axis can be treated separately, thorough its own control law. In the sequel, we will work only with the discrete-time dynamics from (3.21) that have been split into three separate and independent subsystems. Thus, the state vector of each such subsystem will consists of the pair $(x_i, \dot{x}_i)$, which is a 2 states (outputs) x 1 input system.

3.4 Numerical Examples using MATLAB Solvers

In order to exemplify the functionality of the proposed solutions, the system introduced above is used. After analyzing the singular values diagram of the system, a sampling period of $T_s = 6.28$ seconds has been chosen. This assures that the the discrete counterpart of the continuous-time system will have a similar behaviour, as the highet singular value has a magnitude of under 10^{-2}. The continuous-time state space model is:

$$\dot{x}(t) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0.0014 \\ 0 \end{bmatrix} u(t) \tag{3.26}$$

with the discrete counterpart:

$$x_{k+1} = \begin{bmatrix} 1 & 0 \\ 6.2832 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0.0088 \\ 0.0276 \end{bmatrix} u_k \tag{3.27}$$

As presented in section 3.3, the state is a vector of two elements, consisting of the speed on the axis (measured n m/s) and the position (displacement) on the axis (measured in m), related to a reference point on the orbit. The initial state of the system is considered to be:

$$x_o = \begin{bmatrix} 0.5[m/s] \\ 3.75[m] \end{bmatrix} \tag{3.28}$$

and the reference final state is the origin of the axis (the null vector).

3.4.1 Minimum Time

The minimum time algorithm relies on solving QP problems. For this purpose, the MATLAB *quadprog* routine is used. The arguments of this routine are: *H, f, A, b, Aeq, beq, LB, UB*:

- H, f - define the quadratic cost to be minimized
- A, b - define the inequality constraints
- Aeq, beq - define the equality constraints
- LB, UB - define the box constraints for the objective variable x

The matrices H and f are defined as in (3.9), whereas the magnitude constraints for input signal u are treated as box constraints, thus LB is equal to 1 and UB to -1 (as vectors, sized corresponding to $\mathcal{U}$).

For this initial setup, the resulting minimum time is equal to **136** samples, or, equivalently, **854.5132** seconds. The control signal which drives the system to the origin of the state space in the shortest time possible, along with the state trajectory are illustrated in Figure 3.1, located below, and Figure 3.2, located at the top of the next page.

The high overshoot, seen in Figure 3.2 is caused by the initial speed of the system, combined with the low maximum force that the thrusters can exert. A very important aspect is the form of the resulting control signal. Recalling Chapter 2, Section 2.6, the command signal obeys the two-state rule (*bang-bang* control), being either -1 and $+1$. However, a deviation from this form can be seen at time instant 609, where the control signal takes a value of -49% of the maximum thrust force. This is a consequence of approximating the continuous-time system, along with its minimum time optimal control, using a discrete representation. Thus, the solution to the discrete-time optimal control problem is, itself, an approximation of the continuous-time one. As T_s decreases, the resulting control law will become more similar to the theoretical form (bang-bang). A proposed solution for this computational error is to force the result to be a bang-bang control law, by applying a "sgn" function over it. Moreover, due to the choice of T_s, the switching time most likely does not overlap with a sampling moment. The total computation time of the algorithm was **0.08169**5 seconds, including the evaluation of the intermediary

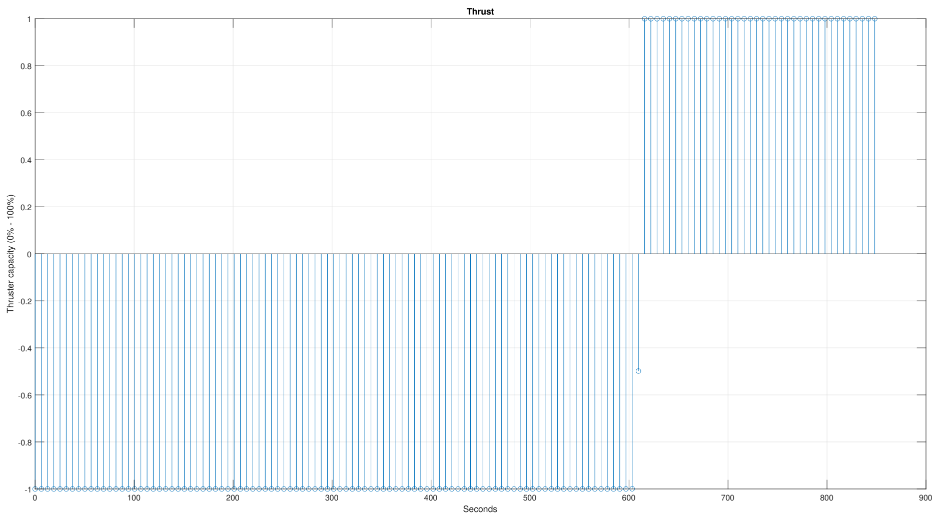

Figure 3.1: Minimum time optimal control

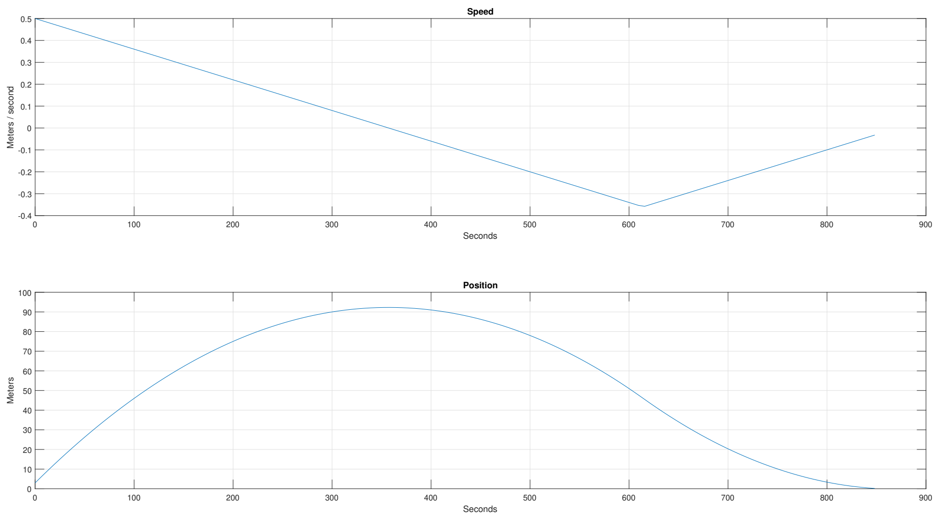

Figure 3.2: Minimum time state trajectory

final state error, required for the bisection algorithm. *quadprog* solver was called for **14** times, summing a total of **144** iterations done in order to find the optimal control law. The resulting final state is:

$$x_f = \begin{bmatrix} -0.0335[m/s] \\ 0.01831[m] \end{bmatrix} \tag{3.29}$$

3.4.2 L^1 Control

For the L^1 control problem, an LP solver is required. The solver offered by MATLAB is the *linprog* routine, whose input arguments are similar to the ones of *quadprog*, excepting for the cost function, which is now defined through a vector **f**.

For this numerical example, the same configuration as previously discussed is used (the same system and initial state). The time horizon over which the problem is solved is equal to **170** samples or, equivalent to $\mathbf{1.0681 \cdot 10^3}$ seconds (must be greater than the minimum time corresponding to the initial state, which is 136 samples). The optimal control returned by *linprog* along with the resulting state trajectory are represented in Figure 3.3 and Figure 3.4, bellow. The elapsed time during *linprog* execution was **0.134998** seconds, leading to a final state of:

$$x_f = \begin{bmatrix} -0.0088[m/s] \\ 0.0276[m] \end{bmatrix} \tag{3.30}$$

Moreover, the obtained control signal is of the type deduced theoretically in Section 2.7, Chapter 2 (a three-state control signal - *bang-off-bang*). There are two samples of this signal which do not comply to the theoretical result, as in the minimum time case. The discussion on these two samples is identical to the previous one, regarding the error in the minimum time optimal control. A proposed solution here would be to approximate each sample to the closest integer, using the "round" function.

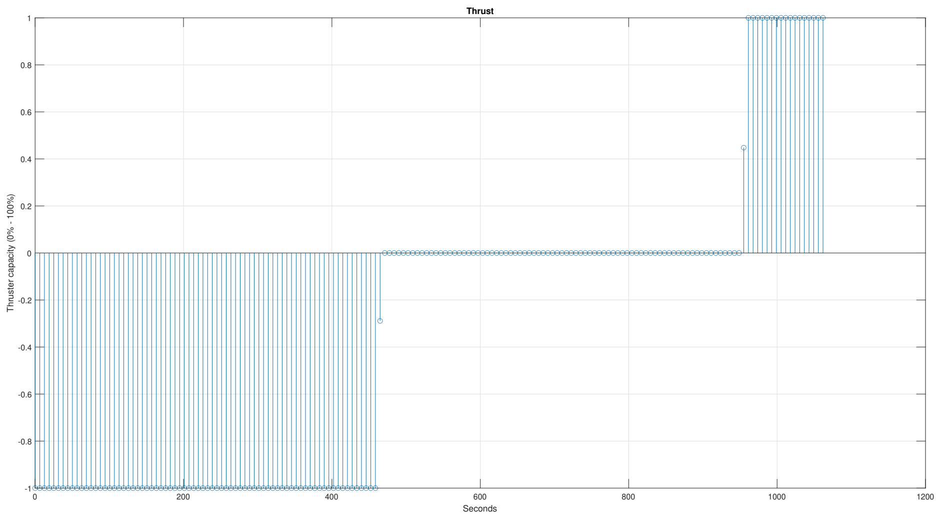

Figure 3.3: L1 optimal control

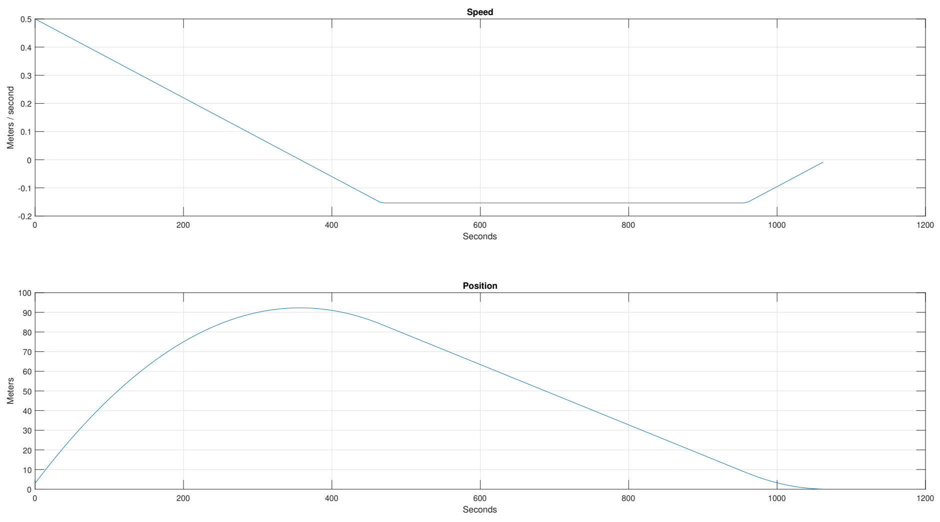

Figure 3.4: L1 control resulting state trajectory

3.4.3 Hands-Off Control Algorithm

By combining the two solvers exemplified previously in the manner proposed in algorithm 2.1, the Hands-off Control Algorithm can be obtained.

For this simulation, the same system and initial condition are used. Additionally, a stochastic disturbance is introduced to the system, bounded in magnitude by 0.01. This perturbation is meant to model phenomena that interfere with the satellite's ideal trajectory, such as atmospheric drag, solar pressure, gravitational anomalies, relativistic effects, etc. The imposed maximum sparsity rate was set to **0.65**, and the safety factor is set to $5 \cdot T_s$.

The behavior of the closed loop system is represented in Figure 3.6, on the next page, while the control signal produced by the algorithm is illustrated in Figure 3.5, below. The obtained sparsity rate, which indicates the real percentage of on-time for the thruster, is **0.35591**, which is close to half of the imposed maximum. The effect of the perturbation can still be seen in the speed of the body, even after the position reaches a so-called steady-state, at the origin of the axis.

The control effort is higher at the beginning of the simulation, as the system is initially situated far from the origin and has a non-zero initial speed. Afterwards, the thrusters are only turned on only to bring corrections to the position of the satellite, if required.

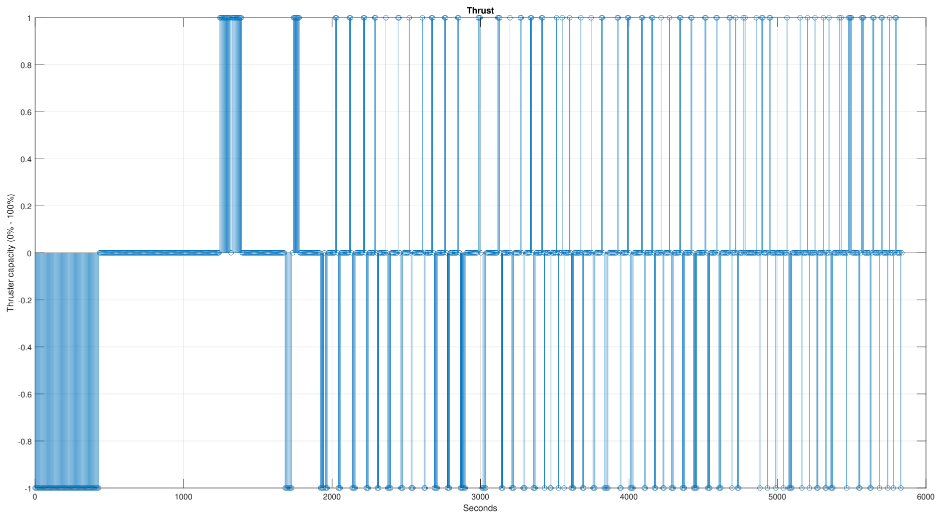

Figure 3.5: Hands-off Control

Finally, for the hardware implementation stage, the two solvers require a conversion from MATLAB code to HDL (Hardware Description Language) logic. The problem in using the above-mentioned solver is that MATLAB HDL coder only supports floating-point code conversion, whereas the desired target FPGA only supports fixed-point logic. Moreover, manually implementing a solver would allow a higher degree of flexibility in terms of resource usage along with more code optimization, based upon the problem's formulation.

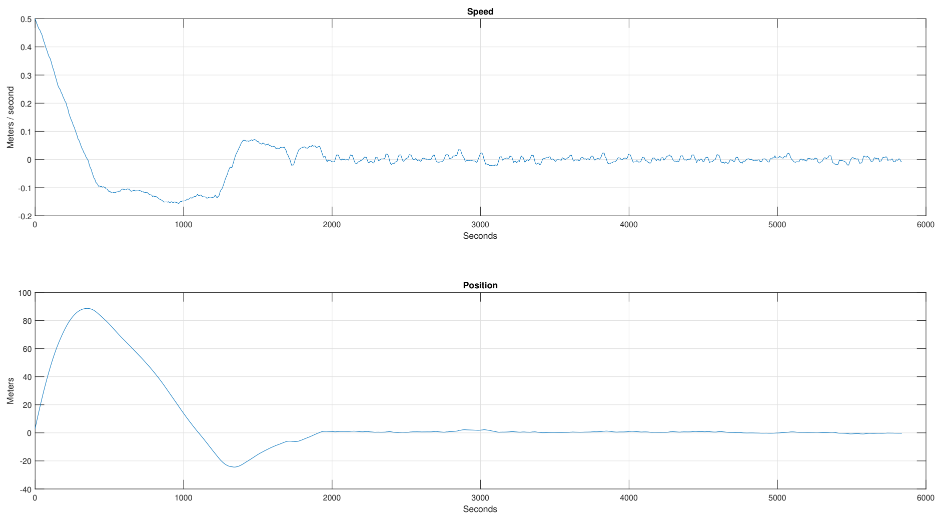

Figure 3.6: Hands-off Control state trajectory

3.5 ADMM Solver

For the QP problem, usual second order methods would require computing the inverse of the Hessian matrix Q. This, however, is not possible, as it is singular for any time horizon greater than the dimension of the state space. Gradient methods, on the other hand, would require projection on the feasible domain. Thus, for the LP problem, an explicit projection would be required on the intersection between the equality constraint (the system hits the desired final state) and the inequality constraint (the control signal are unitarily bounded in absolut value). Finding this intersection is, in itself, another optimization problem. Therefore, a particular solver is required, that treats these constraints in a more efficient manner and does not pose any numerical problem, such as ill-posed matrix inversion.

The work presented in [7] shows an example of an FPGA implementation of the MPC control algorithm over long time horizons. The MPC controller consists of a QP problem, similar to the one formulated for the minimum time problem. The results of this article, along with conclusions drawn after comparing several solvers, argue in favor of an Alternating Direction Method of Multipliers (ADMM) solver for the final implementation.

Essentially, the ADMM algorithm is based on a dual-primal optimization approach. Each iteration updates both the primal and dual variables. What makes this algorithm different from the classical *dual decomposition* method is the introduction of the augmented Lagrangian, and the use of decomposed objective variables. Thus, instead of jointly updating all the primal variables, the variables are updated in an alternating manner, after a proper decomposition has been performed. More details on the theoretical aspects of this algorithm can be found in the reference [4].

The approach presented in [7] uses an adapted form of the algorithm, published in [11]. This algorithm is also available as an open source package for various programming languages, under the name of OSQP (which stands for “Operator Splitting Quadratic Programming”). As stated in the previously mentioned work, this algorithm is capable of solving problems of the following form:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & \frac{1}{2}x^\top Qx + qx \\ \text{s.t.} \quad & lb \le Ax \le ub \end{aligned} \tag{3.31}$$

The inequality constraints can also be used to represent equality constraints. A constraint such as

$$a^\top x = b$$

can be encoded in the vectors lb, ub and matrix A in the following way:

- Constant b is included in vectors lb and ub in the same position, with opposite signs ("-" for lb and "+" for ub).
- Vector $a^\top$ is included as a line in matrix A in the same position as b in the two vectors lb, ub.

Another great advantage originating from using this algorithm is that it also provides solutions for LPs, by substituting matrix Q with the zero matrix.

The core of the algorithm consists of a linear system of equations, called the "KKT System". The solution of this system is used for the primal variables update, where the aforementioned system is of the following form:

$$\begin{bmatrix} Q+\sigma I & A^\top \\ A & -\rho^{-1} I \end{bmatrix} \begin{bmatrix} \widetilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1} y^k \end{bmatrix} \tag{3.32}$$

An important remark is that the matrix of this system consists only of the quadratic cost matrix, Q, the inequality constraint matrix A, and two constants σ and ρ, with the latter two acting as step sizes for the algorithm. Therefore, each iteration solves a system of equations with the same matrix. By pre-computing a factorization for this matrix, before the initiation of the ADMM algorithm, computational effort may easily be reduced when solving the KKT system. More details about the chosen factorization will be discussed in chapter 4. The auxiliary variable $\widetilde{x}_{k+1}$ is the one causing the *split* between the optimization variables. It defines an intermediary optimization problem for which ν_{k+1} serves as a Lagrange multiplier. The optimality conditions for this newly defined problem lead to the formation of the KKT System. The term z is introduced in order to transform the inequality constraint for x into a box constraint, which is easier to project on, and y is a Lagrange multiplier, associated with the inequality (box) constraint. Following this step, the rest of the ADMM iteration updates variables x, z, y and ν.

In what follows, the effects of replacing *quadprog* with ADMM are analyzed, by doing a performance comparison between the two algorithms. Before doing so, it is important to mention that constants α, σ and ρ have a great impact on the output of the ADMM solver. Thus, a manual tuning of those parameters was done before comparing the performances. For all of the simulations, the same system along with the same initial conditions are employed.

3.5.1 Minimum Time

When solving QPs, the following set of parameters was found to be suitable for the algorithm:

$$\begin{cases} \sigma = 0.01 \\ \rho = 0.01 \\ \alpha = 1.9 \end{cases} \tag{3.33}$$

The solution to the minimum time problem offered by the algorithm based on the ADMM solver is represented in Figure 3.7, along with the state trajectory represented in Figure 3.8, on the next page. The solution to the minimum time problem offered by this algorithm is **136** samples, identical to the one offered by the *quadprog*-based algorithm. There is a slight difference between the control signals offered by the two solvers, in the sense that the one offered by the ADMM has two samples which do not fit to the "bang-bang" rule. This can be caused by numerical errors introduced by solving the KKT system at each iteration. Nonetheless, the result in this scenario is identical to the previous one, as only the minimum time is of interest, not the optimal control signal.

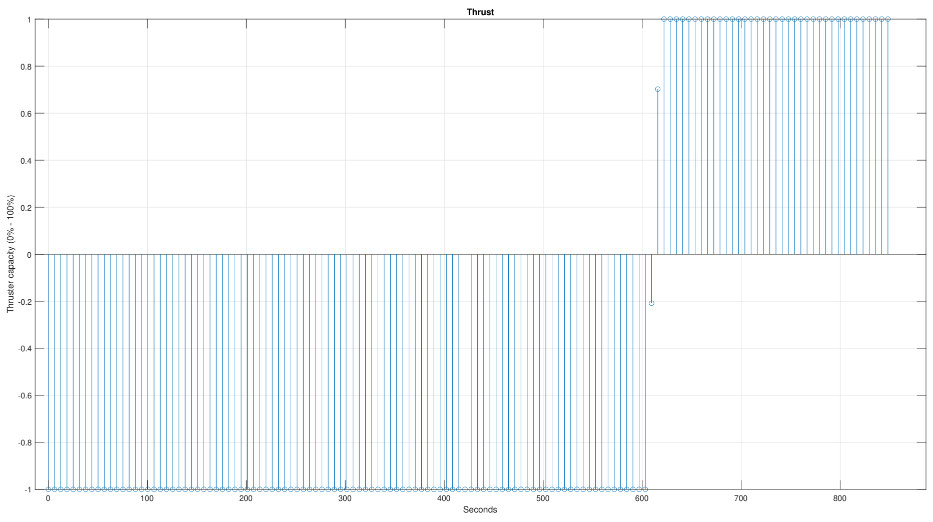

Figure 3.7: ADMM - Minimum time optimal control

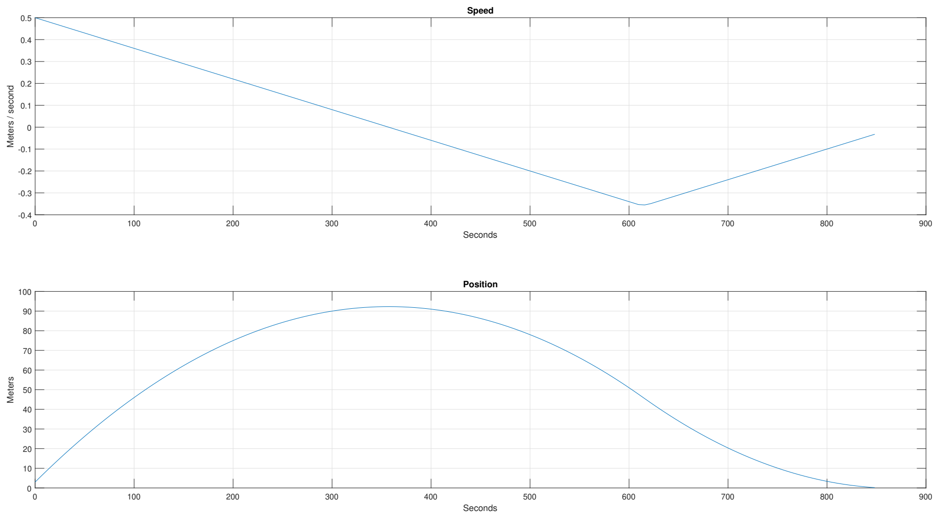

Figure 3.8: ADMM - Minimum time control state trajectory

The solution was calculated in **0.096561** seconds, with a total of **7000** iterations executed by the algorithm. The final state of the system, after applying the optimal control, is:

$$x_f = \begin{bmatrix} -0.0322[m/s] \\ 0.0175[m] \end{bmatrix} \tag{3.34}$$

3.5.2 L^1 Control

The set of parameters suitable for solving LPs has been experimentally determined as $\sigma = 0.01$, $\rho = 1.8$ and $\alpha = 1.9$. The output for the L^1 optimal control problem using the ADMM is illustrated in Figure 3.9, with the state trajectory being illustrated in Figure 3.10. The computation lasted only **0.014078** seconds, which is 10 times faster than the one associated with the *linprog* routine, and produces a result very similar to the one offered by *linprog*.

The resulting control signal is a three-state signal, with the exception of 3 samples situated between the theoretically prescribed states of the signal. This, again, may be the result of numerical errors accumulating during execution time. Nevertheless, the solution is viable as it drives the system close enough to the origin, to a final state of:

$$x_f = \begin{bmatrix} -0.0159[m/s] \\ 0.0725[m] \end{bmatrix} \tag{3.35}$$

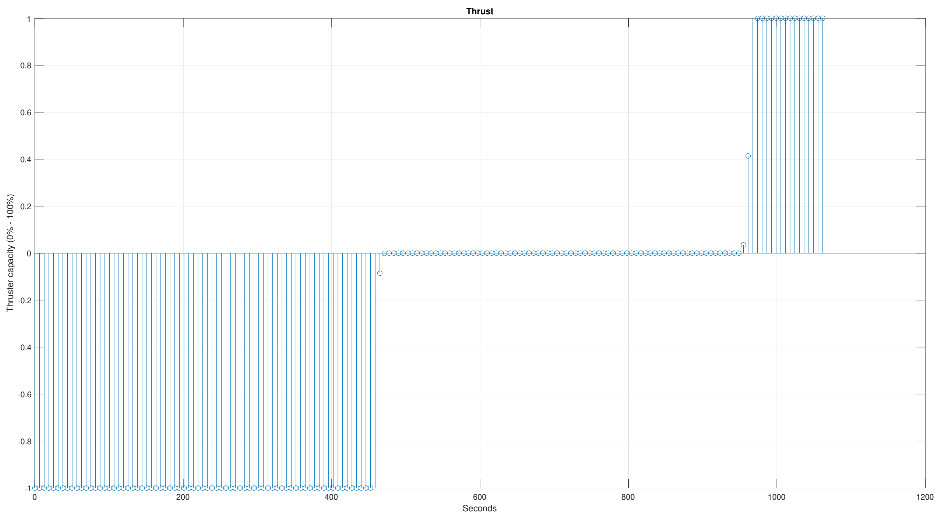

Figure 3.9: ADMM - L1 optimal control

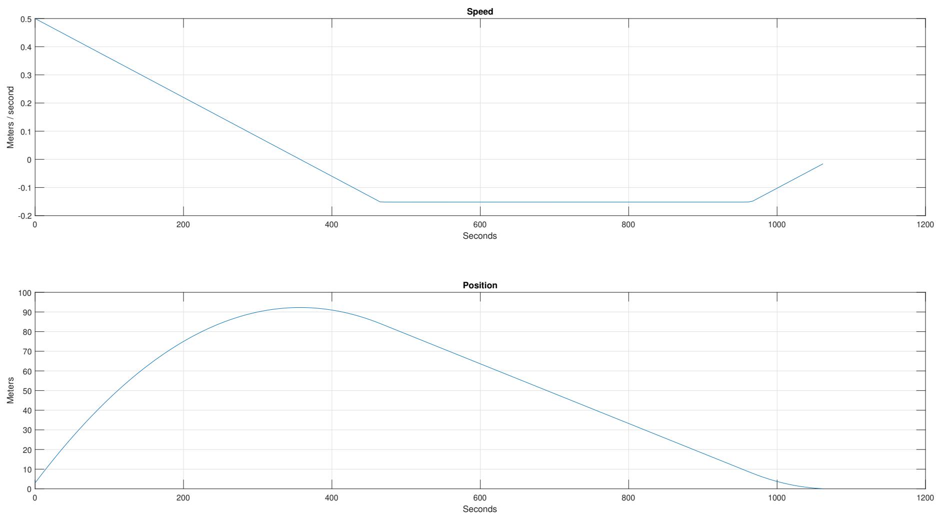

Figure 3.10: ADMM - L1 control state trajectory

3.5.3 Hands-off Control Algorithm

By substituting both routines provided by MATLAB with the newly developed ones, a hardware implementation-oriented form of the Hands-off control algorithm is obtained. Again, the system is considered to be affected by a stochastic disturbance for the purpose of simulating a real orbital behavior. The imposed maximum sparsity rate is equal to **0.65**, with the safety factor being configured as $5 \times T_s$. The closed-loop system evolves according to figure 3.12, while the controller provides the control signals from figure 3.11.

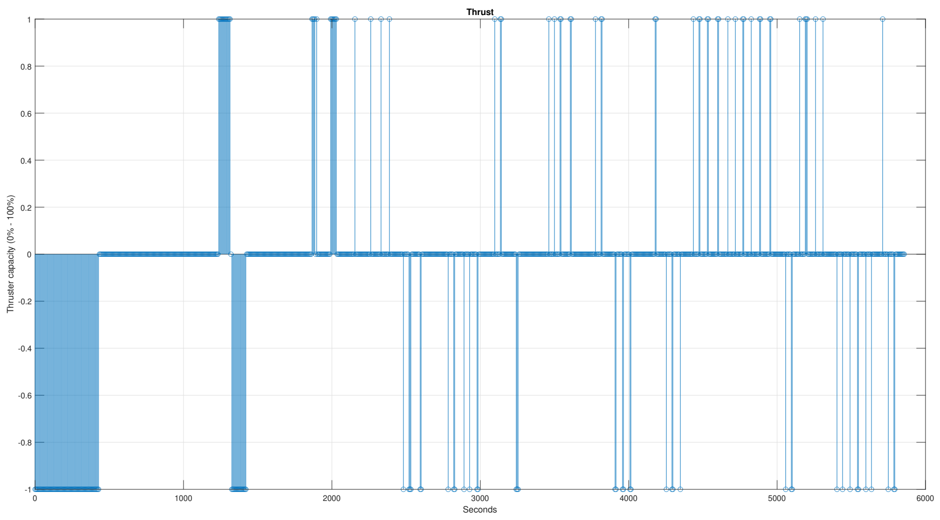

Figure 3.11: ADMM - Hands-off Control

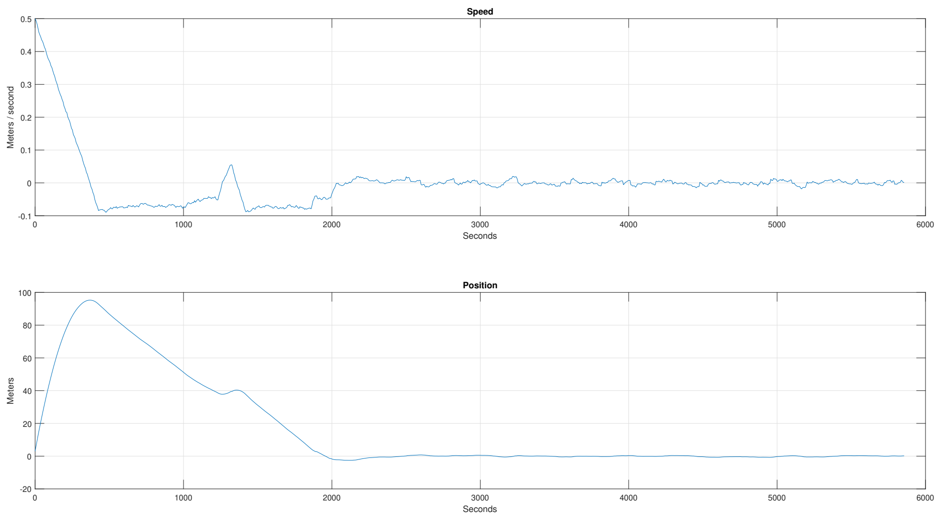

Figure 3.12: ADMM - Hands-off Control state trajectory

The control algorithm generates a control signal with a sparsity rate of **0.20364**. It can be deduced from the two plots that the system is control algorithm achieves the objective of driving the system to the origin of the state space and prevents the disturbance from steering the system away from this state.

4 Hardware Implementation

This chapter presents how the Hands-off Control algorithm was adapted to a hardware implementation, meant to be implemented on an FPGA. This type of hardware platform has become a very profitable choice for many applications, especially for processing large amounts of data at high rates. FPGAs are equipped with significant quantities of computational units such as Look-up Tables (LUT), Flip-Flops (FF) and Digital Signal Processing slices (DSP) which can be assigned in any manner preferred by the designer. This flexibility allows for the fine tuning of both computation time and resource usage. However, it does come with a trade-off, namely that the main drawback in FPGA development is the reduced speed at which solutions can be implemented.

Programming languages available for FPGA development, such as VHDL or Verilog, are low-level languages. Functions, which are the most common blocks of the usual programming languages, are treated as modules in the low-level formalism. Those modules have a continuous connection to their input and output signals. Therefore, triggering and synchronising mechanisms are required, which are widely considered to be building blocks of an FPGA implementation. By interconnecting multiple modules and further integrating them into larger architectures, a complex programming logic can be obtained. Developing an efficient module in HDL is a time-consuming task, as it is very similar to designing a digital circuit (register layout, wiring, input and output ports, etc), capable of producing the desired logical diagram.

As industrial companies started to show more interest in the potential of FPGA platforms, a partial solution emerged for the problem described in the previous paragraph. High-Level Synthesis (HLS) tools are specialized pieces of software which convert code from a high-level programming language (such as C, C++, Python, MATLAB) to hardware description languages (VHDL, Verilog). This approach significantly accelerates the process of developing FPGA-based products. Nonetheless, raw code conversion can often produce inefficient module designs and can, occasionally, introduce bugs in the resulting programs. The key to those shortcomings is the use of HLS directives, which influence the conversion process, redistribute hardware resource consumption and can create or modify communication interfaces (protocols and ports).

The tool used for this thesis is Vitis (formerly known as Vivado) HLS, created by AMD. This software allows for C++ code conversion to Verilog/VHDL instructions. The documentation for this software can be found at [13].

4.1 Vitis HLS and Workflow

Vitis HLS is a software suite which accelerates FPGA development by synthesizing HDL modules from C++ code. Modules are generated from C++ functions, which are also called “hardware functions”. Before HDL code generation, a process of development and testing is required. In order to test the functionality of a C++ “module”, a test bench file is necessary. Test benches are files containing other functions and logical instructions which are meant to test the module in development, by stimulating it and then collecting its output. A good practice is to create a “software function”, which embeds the desired resulting logical instructions. This can serve as a reference model, by helping to apply corrections and modifications to the hardware function.

The design workflow of the chosen application is the following:

1. Create the software function (or the reference model);
2. Debug the software function and ensure its complete and correct functionality;
3. Create the hardware function;
4. Create the test bench using the software function;
5. Test, debug and optimize the hardware function;
6. Convert C++ code to HDL.

Since the numerical solutions presented in the previous chapter have been tested and prototyped in MATLAB, steps 1 and 2 were done based on the developed MATLAB model. Thus, a manual code conversion from MATLAB to C++ was required. In order to complete step 3, an introduction to writing efficient hardware-oriented code was necessary. Also, during this step, some Vitis HLS libraries have been consulted for possible functions which could included as part of the final design. Notably, steps 4 and 5 required most of the effort invested in the implementation of the control algorithms. This was caused by the continuous changes that have been made to the hardware function, in the process of achieving a satisfactory trade-off between a reasonably small latency of the resulting HLD module and moderate hardware resources requirements.

4.2 The Minimum Time Problem

The continuous-time versions of the subsystems introduced at the conclusion of section 3.3 are akin to a double integrator system. For such a system, [1], Chapter 7, Section 2 describes the solution for the time-optimal problem by applying the Pontryagin minimum principle in an explicit form. The result yields an analytic expression of the minimum time, as a function of the two initial states (position and speed):

$$T^*(x,\dot{x}) = \begin{cases} \dot{x} + \sqrt{4x + 2\dot{x}^2} & x > -\frac{1}{2}\dot{x}|\dot{x}| \\ -\dot{x} + \sqrt{-4x + 2\dot{x}^2} & x < -\frac{1}{2}\dot{x}|\dot{x}| \\ |\dot{x}| & x = -\frac{1}{2}\dot{x}|\dot{x}| \end{cases} \tag{4.1}$$

This function can be adapted to the above-mentioned system by scaling the states according to the system dynamics.

4.3 The "Hardware Function"

According to the algorithm proposed in [10], each iteration requires two optimal control problems to be solved. Thus, the C++ hardware functionality is separated in two main blocks. The first block solves the minimum time problem, using the expression from (4.1). The amount of resources allocated to this task is relatively small, as the only operations which are required are addition, subtraction, comparison and the extraction of the square root. The only important factor which can influence the accuracy of this block's output is the precision of the fixed-point representation. This aspect will be discussed at the end of this chapter.

The second sub-module is assigned to the L^1 control problem. As discussed in chapter (3), this problem can be turned into a numerical optimization problem, more specifically to an LP, with the problem being discussed in this module being the one presented in (3.19). To this end, the ADMM solver discussed in the aforementioned chapter is used. For the solver to be applied, the constraints from (3.19) must first be brought to a form which is similar to those from (3.31). The initial set of constraints is:

$$\begin{cases} |\mathcal{U}_i| \leq \mathcal{T}_i & \forall i = \overline{1:Nm} \\ 0 \leq \mathcal{T}_i \leq 1 & \forall i = \overline{1:Nm} \\ E\,\mathcal{U} = F \end{cases} \tag{4.2}$$

which is equivalent to:

$$\begin{cases} \mathcal{U}_i - \mathcal{T}_i \leq 0 & \forall i = \overline{1:Nm} \\ \mathcal{U}_i + \mathcal{T}_i \geq 0 & \forall i = \overline{1:Nm} \\ 0 \leq \mathcal{T}_i \leq 1 & \forall i = \overline{1:Nm} \\ E\,\mathcal{U} = F \end{cases} \tag{4.3}$$

This set of inequalities, along with the last equality, can be rewritten in matrix form as:

$$\begin{bmatrix} -2 * \mathbf{1}_{Nm} \\ \mathbf{0}_{Nm} \\ \mathbf{0}_{Nm} \\ -F \end{bmatrix} \leq \begin{bmatrix} I_{Nm} & -I_{Nm} \\ I_{Nm} & I_{Nm} \\ 0_{Nm} & I_{Nm} \\ E & -0_{n\times(Nm)} \end{bmatrix} \begin{bmatrix} \mathcal{U} \\ \mathcal{T} \end{bmatrix} \leq \begin{bmatrix} \mathbf{0}_{Nm} \\ 2 * \mathbf{1}_{Nm} \\ \mathbf{1}_{Nm} \\ -F \end{bmatrix} \tag{4.4}$$

The upper and lower limits for $\mathcal{U} - \mathcal{T}$ and $\mathcal{U} + \mathcal{T}$, respectively, have been introduced as an additional measure of numerical stability, and the last n lines force the equality $E\,\mathcal{U} = F$. Therefore, the inequality matrix, A, is:

$$A = \begin{bmatrix} I_{Nm} & -I_{Nm} \\ I_{Nm} & I_{Nm} \\ 0_{Nm} & I_{Nm} \\ E & -\mathbf{0}_{n\times(Nm)} \end{bmatrix} \tag{4.5}$$

It is worth mentioning that throughout the functioning time of a hardware module, its layout of resource allocation remains unchanged. Thus, reallocation of memory or computational resources is not possible, in spite of the varying length of the L^1 control problem horizon. The procedural approach would be to allocate memory in accordance with the horizon length, for every ADMM call. The proposed HDL solution, for this matter, is to create an architecture capable of solving an L^1 problem for a maximum horizon length N_{max}. The inequalities which constraint the input vector $\mathcal{U}$ remain unchanged, while the equality constraint is partitioned such that only the necessary input variables are taken into consideration when evaluating the constraint. For a horizon length of n_{L1}, the matrix E becomes:

$$E = \begin{bmatrix} A_d^{n_{L1}-1}B_d & A_d^{n_{L1}-2}B_d & \ldots & A_dB_d & B_d & \mathbf{0}_{n\times m} \ldots & \mathbf{0}_{n\times m} \end{bmatrix} \tag{4.6}$$

The zero blocks at the end of the matrix are introduced in order to match the size of $\mathcal{U}$. In a similar way, the vector q from the objective function is also extended with zeros, in order to match the same size. We point out, however, that after the ADMM-based solver goes through its iterations, only the first n_{L1} variables are further evaluated and returned to the control algorithm.

Compared to the minimum time block, the module treating the L^1 problem achieves a much higher resource consumption. More memory is required to store the objective variables (primal variables), the Lagrange multipliers (dual variables) and, most importantly, for solving the KKT system. The objective variable, x_k, takes up a block of $\mathcal{O}(2N_{max})$ memory slots, as it stores both the control signal samples and the slack variables. The variables y_k and z_k occupy a total of $\mathcal{O}(3N_{max})$ each. The resource consumption for this block is mostly composed out of units (of memory, LUT and DSP) designated for solving the KKT system.

For communication purposes, the module is equipped with a set of interfaces, one for receiving data regarding the current state of the satellite, and one for sending the control signals to the actuators. After analyzing a library of communication interfaces offered by Vitis, the AXI Stream package was chosen due to it being a reliable interface for this application. Generically, it consists of a bus, *TDATA*, of arbitrary width, and three control signals *TREADY*, *TVALID* and *TLAST*. *TVALID* is used by the transmitter to signal the start of a transaction. Once *TREADY* is received, the transaction starts until the last data package is sent, during which *TLAST* is communicated. This protocol allows the transmission of large amounts of data in high-rate bursts, which is essential for transferring vectors or matrices.

The proposed architecture for the resulting Hands-off Control module is the following:

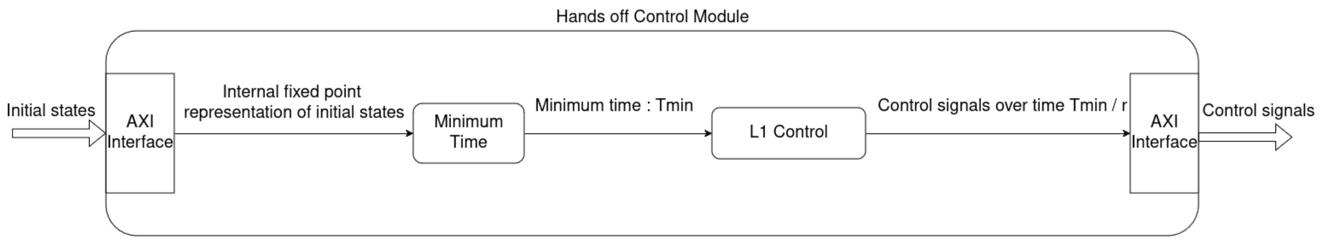

Figure 4.1: Hardware function design

The L^1 control block is further composed out of sub-blocks, responsible for solving the KKT system.

4.4 Solving the KKT System

The true challenge of the hardware implementation lies with choosing an efficient manner of solving the KKT system. Before discussing aspects related to caching and factorization of the KKT matrix, another important optimization is brought to attention. Recalling equation (3.32), the KKT matrix is composed of a positive semi-definite matrix Q and the constraints matrix A. The structure of this matrix is presented in (4.5). Such a structure leaves space for optimization, if the equations forming the linear system are written in an explicit form. In order to benefit from this, the vectors composing the system have to be partitioned conformally with the blocks that make up the KKT matrix.

Knowing that $\tilde{\mathbf{x}}_{k+1}$ contains the control inputs and the slack variables, it can be partitioned as follows:

$$\tilde{\mathbf{x}}_k = \begin{bmatrix} \mathbf{u} \\ \mathbf{t} \end{bmatrix} \tag{4.7}$$

Moreover, by using the fact that z_k and y_k are of size $3 * N_{max} + 2$ (2 comes from the equality constraint), the "b" vector of the system can be partitioned as:

$$\mathbf{b} = \begin{bmatrix} \sigma \mathbf{x}_k - \mathbf{q} \\ \mathbf{z}_k - \rho^{-1}\mathbf{y}_k \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{b}_4 \\ \mathbf{b}_5 \\ \mathbf{b}_6 \end{bmatrix} \tag{4.8}$$

where every $\mathbf{b}_i$ is of length N_{max}, excepting the last one, which is of length 2 (the number of states). In a similar way, ν is partitioned as:

$$\nu = \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \\ \nu_4 \end{bmatrix} \tag{4.9}$$

Thus, the system can be written in an expanded form as:

$$\begin{bmatrix} \sigma I_{N_{max}} & \mathbf{0}_{N_{max}} & I_{N_{max}} & I_{N_{max}} & \mathbf{0}_{N_{max}} & E^{\intercal} \\ \mathbf{0}_{N_{max}} & \sigma I_{N_{max}} & -I_{N_{max}} & I_{N_{max}} & I_{N_{max}} & \mathbf{0}_{N_{max}\times 2} \\ I_{N_{max}} & -I_{N_{max}} & -\rho^{-1} I_{N_{max}} & \mathbf{0}_{N_{max}} & \mathbf{0}_{N_{max}} & \mathbf{0}_{N_{max}\times 2} \\ I_{N_{max}} & I_{N_{max}} & \mathbf{0}_{N_{max}} & -\rho^{-1} I_{N_{max}} & \mathbf{0}_{N_{max}} & \mathbf{0}_{N_{max}\times 2} \\ \mathbf{0}_{N_{max}} & I_{N_{max}} & \mathbf{0}_{N_{max}} & \mathbf{0}_{N_{max}} & -\rho^{-1} I_{N_{max}} & \mathbf{0}_{N_{max}\times 2} \\ E & \mathbf{0}_{2\times N_{max}} & \mathbf{0}_{2\times N_{max}} & \mathbf{0}_{2\times N_{max}} & \mathbf{0}_{2\times N_{max}} & -\rho^{-1} I_2 \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{t} \\ \nu_1 \\ \nu_2 \\ \nu_3 \\ \nu_4 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \\ \mathbf{b}_4 \\ \mathbf{b}_5 \\ \mathbf{b}_6 \end{bmatrix} \tag{4.10}$$

Multiplying the matrix with the vector of unknown variables gives:

$$\begin{cases} \sigma u + \nu_1 + \nu_2 + E^{\intercal} \nu_4 = b_1 \\ \sigma t - \nu_2 + \nu_2 + \nu_3 = b_2 \\ u - t - \rho^{-1} \nu_1 = b_3 \\ u + t - \rho^{-1} \nu_1 = b_3 \\ t - \rho^{-1} \nu_3 = b_5 \\ Eu - \rho^{-1} \nu_4 = b_6 \end{cases} \tag{4.11}$$

and solving for $\mathbf{u}$ and $\mathbf{t}$ results in:

$$\begin{cases} \mathbf{u} = (E^{\intercal} E + \frac{\sigma}{\rho} I + 2I)^{-1} (\rho^{-1} + b_3 + b_4 + E^{\intercal} b_6) \\ \mathbf{t} = (b_2 + \rho b_4 + \rho b_3 - \rho b_4) \cdot \dfrac{1}{\sigma + 3\rho} \end{cases} \tag{4.12}$$

which can be further substituted in (4.11) to obtain

$$\begin{cases} \nu_1 = \rho(u - t - b_3) \\ \nu_2 = \rho(u + t - b_4) \\ \nu_3 = \rho(t - b_5) \\ \nu_4 = \rho(Eu - b_6) \end{cases} \tag{4.13}$$

It follows by direct inspection that the initial equations system of size $5N_{max} + 2$ has been reduced to one of size N_{max}. If an elimination method is used, then the complexity drops from $\mathcal{O}(25N_{max}^2)$ to $\mathcal{O}(N_{max}^2)$. If, instead, matrix inversion is applied, then the complexity of the operation drops from $\mathcal{O}(125N_{max}^3)$ to $\mathcal{O}(N_{max}^3)$, which is a notable improvement.

In spite of all these computational improvements, it is important to point out that the caching of the KKT matrix has a critical impact on the obtained runtime. Before adopting a caching method, a Vitis Library function was chosen as a solver for the equation system, namely *polinearsolver*, which is a solver for positive definite systems of equations. However, this lead to a very long ADMM iteration time, due to high complexity, and regular tests displayed a duration of upwards to 15 or even 20 minutes, thus creating an unfavorable environment for development.

4.5 Caching the KKT Matrix

As previously mentioned, the matrix involved in the linear system of equations is constant during the execution time of the algorithm. Therefore, this matrix can be factorized in a way which would facilitate the solving of the system, with the result being stored before initializing the solver. Such a procedure is termed "caching" a factorized form of the matrix. Two important properties of the KKT matrix are its symmetry and positive definiteness. This allows for a *Cholesky* decomposition of the matrix.

Theorem 4.1. *Let A be a symmetric positive definite matrix in $\mathbb{R}^{n\times n}$. Then, there is a unique lower triangular matrix L in $\mathbb{R}^{n\times n}$ with positive elements on its diagonal, such that:*

$$A = LL^\intercal \tag{4.14}$$

where L is called a Cholesky factor.

The matrix which is subject to the factorization is:

$$K = E^\intercal E + \frac{\sigma}{\rho} + 2I \tag{4.15}$$

The Cholesky factorization complexity is of $\mathcal{O}(N_{max}^3/3)$. By doing this at the beginning of the algorithm, the equation system can be solved using two LTRIS (linear triangular solver) procedures, each one having a complexity of $\mathcal{O}(N_{max}^2)$.

For the Cholesky factorization, a function was implemented, based on the algorithm from [5], Chapter 2, Section 11. As stated in the section describing the satellite model, any operation involving matrix E can be done without the explicit need of storing E, as its elements can be computed on the spot. Therefore, this function does not require any input matrix or vector, only the L^1 optimal control problem horizon length, which is used to compute the powers of A_d. The output of this function is the Cholesky factor, L, corresponding to matrix K. An additional optimization can be done when storing the output of the factoring function. The result, as defined in theorem 4.1, is a lower triangular matrix. Thus, storing it in a two dimensional array of sizes $N_{max} \times N_{max}$ would lead to half of the memory occupied by the matrix to be wasted. As a solution for this, matrix L is stored in a vector, as a concatenation of its lines. Furthermore, this implies a different way of accessing elements in this matrix. As a general rule:

$$L[i][j] = \mathbf{L}\left[\left\lfloor \frac{i(i+1)}{2} \right\rfloor + j\right] \tag{4.16}$$

where L is the Cholesky factor and $\mathbf{L}$ is the one dimensional array storing it.

The *LTRIS* and *UTRIS* routines, used to solve the lower and upper triangular systems of equations, resulting from the factorization of the KKT matrix, were implemented according to the algorithms in [5], Chapter 1, Section 9.

4.6 Hardware Optimizations

Since one of the objectives of the work presented in this chapter is to obtain a fast response time from the control algorithm, but also to comply with the available set of resources available to the hardware platform, a series of modifications have been made to the C++ hardware function. For optimization purposes, Vitis provides two highly effective directives, also called "Pragmas". Those directives can be attached to a loop, in order to influence the performance of the hardware modules executing the logical intructions contained by the aforementioned loop.

4.6.1 Unrolling

The first pragma discussed is “Unroll”. This directive, along with an *unrolling factor*, is used to parallelize the execution of a loop. The unrolling factor is used to specify the level of parallelization. For example, an unrolling factor of 1 leads to a full parallelization of the loop instructions. This can also be seen as replacing the loop with a parallel implementation of the loop instructions over the entire domain of the loop. An unrolling factor of 2 causes the loop to be executed in two “iterations”. The hardware logic will handle half of the loop’s computational effort in one clock cycle, and the other half in the next clock cycle. A direct consequence of applying this directive is a significant decrease in the propagation time of the module, but also an increase in resource consumption, especially in terms of LUTs and FFs. However, no loop will be unrolled explicitly, as this pragma is implicitly used by the *pipeline* directive.

4.6.2 Pipelining

This directive is used predominantly in loop optimizations. By attaching this primer to a loop, the compiler forces the hardware instructions, obtained from the loop conversion, to be able to execute the equivalent of one loop iteration in a fixed number of clock cycles, which is given as an argument to the PIPELINE pragma. This, however, can only be achieved if all the instruction inside the loop are able to be carried out in parallel. Most of the time, this is possible for usual loops. For nested loops, however, adding the pragma option to one of the middle or outer loops has a strong impact on the resulting hardware module. By trying to pipeline an outer loop, the compiler will attempt to fully unroll the inner loops, leading to a massive increase in resource consumption (LUTs and FFs) in order to achieve better runtime performance. This is only possible if the inner loops have a fixed number of iterations, which does not vary from one hardware function call to another. Another necessary condition, allowing the outer loops to the pipelined, is that the arrays being processed in the inner loops must be able to handle multiple I/O operations at once. This can be achieved by declaring the arrays in a segmented manner, which leads to multiple independent memory banks storing the array.

The first condition of inner loops fixed number of iterations was achieved in the following manner. Suppose the initial loop structure is:

Algorithm 4.1: Standard nested loops structure

```
1 for $i = 0;\ i < n;\ i = i + 1$ do
2     /* Outer loop instructions involving a variable $n\_temp$*/
3     for $i = 0;\ i < n\_temp;\ i = i + 1$ do
4         /* Inner loop instructions */
5     end
6 end
```

Pipelining the outer loop is not possible as it would require unrolling a loop with an unknown number of iterations (the inner loop). Nevertheless, changing the logic of the inner loop in the following way:

Algorithm 4.2: Modified nested loops structure

```
1 for $i = 0;\ i < n;\ i = i + 1$ do
2     /* Outer loop instructions involving a variable $n\_temp$*/
3     for $i = 0;\ i < n\_max;\ i = i + 1$ do
4         if $i \leq n\_temp$ then
5             /* Inner loop instructions */
6         end
7     end
8 end
```

This allows for the loop to be unrolled, as it has a fixed number of iterations, namely n_max, thus leading to a decreased runtime. This optimization, however, can lead to wasting additional hardware resources if n_temp rarely reaches values close to n_max, as the hardware blocks allocated for the higher values of the loop iterator i will never get to be used.

The second condition, which entails the possibility of parallelizing an array's I/O operations, can be satisfied by storing temporary copies of the array which may be accessed in parallel to the main array. This can also be done by storing results, meant to be uploaded into the main arrays, in temporary data structures. In order to obtain a balance between memory consumption and time performance, transfers between auxiliary arrays and the main arrays have to be done at the level of the outermost loops, relative to the instructions where the auxiliary arrays replace the main arrays. This method has been heavily employed in the hardware functions which compute the Cholesky decomposition, since there are multiple internal nested loops in the algorithm which operate upon the same row/column.

4.6.3 Precomputing Divisions

Divisions are expensive operations, which also take more clock cycles to complete. This is why, whenever possible, the inverse of the divisor is stored and further divisions are treated as multiplications between the dividend and the pre-computed inverse of the divisor. In order to compute a certain column of the matrix L, the Cholesky decomposition routine has to perform a series of divisions, with luckily have the same divisor. Thus, computing this quantity and storing its inverse will save up resources, especially DSPs, and also computing time. This method is also applied to the minimum time computation routine and to any divisions involving the ADMM algorithm parameters. Moreover, any division where the divisor is a constant value can be replaced by multiplications with inverse values of divisor.

4.6.4 Accessing the Cholesky Factor

The last adjustment which was made to the code is one related to the means of accessing the matrix L. According to (4.16), a multiplication and an addition need to be done in order to access an element. For a full pipeline to be done, the instructions for one loop iteration have to be executed in one clock cycle. Reading both registers (i and j), multiplying, dividing the results and adding it with another variable is simply infeasible in one clock cycle. Thus, we take the vector index from equation (4.16), and denote it by vec_index. Incrementing i by 1 yields:

$$\frac{(i+1)(i+2)}{2} + j = \frac{i(i+1) + 2(i+1)}{2} + j = vec\_index + i + 1 \tag{4.17}$$

where the *floor* operator is considered to be implicitly applied, along with the division, since C++ returns the "floor value" of the division between two integer variables. Equation (4.17) shows how, instead of explicitly computing the index at each iteration, this index can be initialized according to the starting point in the vector and then incremented during each iteration, which solves the clock cycle problem. A similar technique can be applied when j is incremented, leading to a simpler result:

$$\frac{i(i+1)}{2} + j + 1 = vec\_index + 1 \tag{4.18}$$

4.7 The Target FPGA

Before discussing the hardware requirements of the resulting optimized module, we present here the target platform along with its specifications. The PYNQ-Z1 development board is a specialized piece of hardware equipped with an ARM A9 CPU, which runs a Linux distribution. This allows the user to exploit the capabilities of the ZYNQ FPGA by means of a Python framework and a web server, without having to design user interfaces for the hardware modules. The HDL modules are uploaded to the board's memory, where they are instantiated in a Python script which runs on an ARM chip-based Jupyter Notebook server. The FPGA mounted on the board is a Zynq 7000 SoC platform, and its key specifications are the following:

Logic slices	13300
6-input Look-up Tables	53200
Flip-flops	106400
Block RAM	630KB
DSP Slices	22

Table 4.1: Zynq 7000 SoC Hardware Resources

4.8 Performance and Resource Consumption

The first aspect which is relevant to the hardware adaptation of the control algorithm is the precision of the fixed point representation. This is an alternative form of representing non-integer numbers. While the floating point representation can achieve very high precision over a large range of values, it requires a specialized architecture for basic operations, such as addition, multiplication and division. Fixed point numbers have a fixed precision and cover a much lower range of values, but can be used on any type of hardware supporting basic integer operations, and do not require any specific hardware components. The format of a fixed point number is represented using a word length, W and an integer length, I. An example of a fixed-point number is:

$$\underbrace{001...0110}_{\text{Integer part = I}} \quad . \quad \underbrace{01100..110}_{\text{Fraction part = W - I}} = \sum_{j=1}^{I} b_j 2^{j-1} + \sum_{j=1}^{W-I} b_j 2^{-(j-1)} \tag{4.19}$$

Increasing the length of the word size and that of the integer size can lead to a more precise representation, but also causes all of the employed arrays to occupy larger portions of the available memory. For more complex computations requiring finer representation, such as the minimum time computation, the following set of sizes was used:

$$\begin{cases} W = 55 \\ I = 27 \end{cases} \tag{4.20}$$

while for representing and storing the data involved in solving the L^1 optimal control problem, the set of sizes:

$$\begin{cases} W = 40 \\ I = 20 \end{cases} \tag{4.21}$$

offered satisfactory precision, compared to the floating-point version of the algorithm. The average error between the two control signal vectors, namely the one computed by the floating-point version of the algorithm and the one produced by the fixed-point implementation, had a norm of under 5×10^{-2}. This choice of sizes also led to 93% of the FPGA memory being consumed, for a maximum size $N_{max} = 300$.

The resulting LUT consumption is at 90% of capacity, with all of the loops being modified according to 8, in order to allow a full pipeline. The DSP consumption is also close to maximum, at around 97% of capacity. DSP depletion is mostly caused by the Cholesky factorization routine, the LTRIS and UTRIS routines, and the minimum time computation.

The total latency of the HDL module hovers at around $1.43e8$ clock cycles. For a clock frequency of 100MHz, this leads to a total propagation time of 1.46 seconds, required to compute the control signals. This is a reasonable response time, as the chosen sampling period of the system is equal to 6.28 seconds.

4.9 HDL Module Simulation

A testbench was designed in Vitis, meant to replicate the MATLAB simulations shown in the previous chapter. Consequently, the employed system is the same as the one used for previous simulations, and we consider it to be affected by stochastic disturbances having the same characteristics as those in Chapter 3. Using the Vitis simulator, the testbench emulated the functioning of the algorithm, collecting its output data and feeding it to the system, in a manner similar to Algorithm 2.1. The resulting state trajectory of the system is illustrated in the figure bellow:

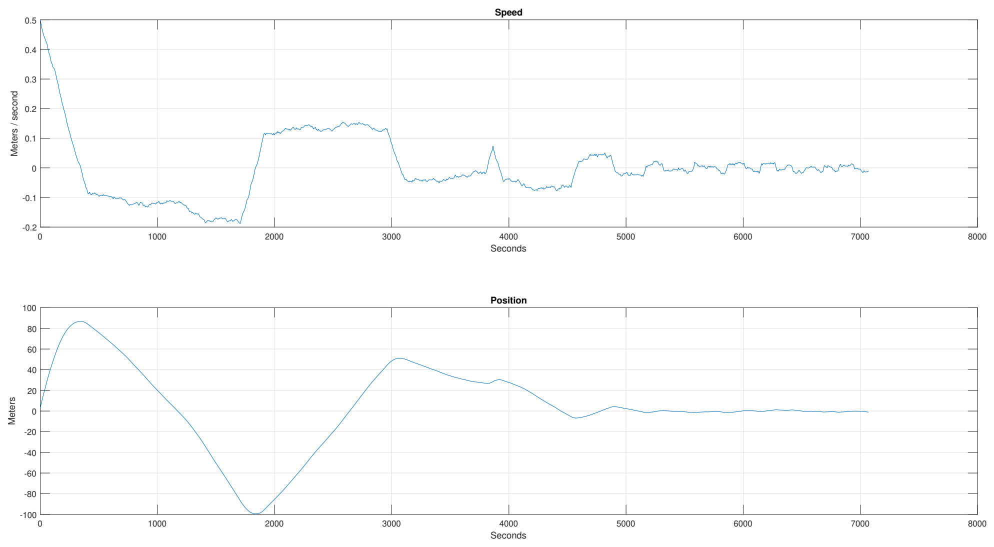

Figure 4.2: HDL - Hands-off Control state trajectory

and the control signals synthesized by the controller are figured at the top of the next page. The imposed sparsity rate was, as earlier, equal to **0.5**, and the resulting sparsity rate is **0.16696**.

A larger deviation from the origin of the state space can be observed at the middle part of the simulation, which is most likely caused by a series of samples for which the disturbance signal was larger in magnitude. This can be interpreted as a stronger external influence on the satellite, such as an impact with another body, which deviates it from the specified trajectory. Nonetheless, the position of the system eventually reaches a steady-state centered on 0, while the effect of the disturbance can still be seen in the evolution of the velocity. As can be seen in Figure 4.3, the control algorithm will periodically send impulses to correct the resulting deviations.

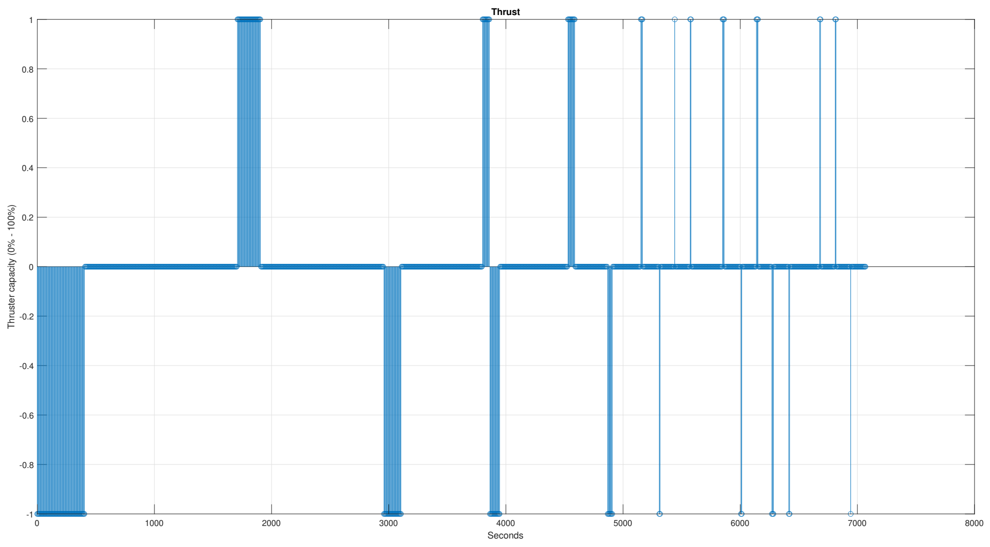

Figure 4.3: HDL - Hands-off Control

5 Conclusion

Throughout this thesis, a complex optimal control problem has been discussed. Before analyzing the problem itself, along with its proposed solution, a theoretical background has been established. This has been done with the help of notions originating from various domains such as: dynamical systems, classical control theory, optimal control theory, functional analysis and optimization theory. Afterwards, based on the aforementioned notions, the L^0 optimal control problem has been stated, together with a solution proposed in article [10]. Therein, a control algorithm was designed by the authors, namely the "'Hands-Off Control" algorithm, which aims to generate a control signal with an pre-imposed sparsity rate. The theoretical chapter ends with some remarks regarding the stability guarantees of the control algorithm. The numerical implementation chapter reiterated through the theoretical foundations of the Hands-off Control problem, this time from a practical point of view. The discussion was focused on the development of numerical procedures, capable of finding solutions to the optimal control problems. After reformulating the problems from the optimal control framework to the domain of numerical optimization, the main issue became that of choosing the right solver for each problem. Firstly, the performance obtained using inbuilt MATLAB solvers was analyzed. Afterwards, a particular solver was presented and adapted for the particular problems treated in this thesis. Finally, the closed-loop system was simulated based on the aforementioned solver and its performance was also analyzed . In the hardware implementation chapter, the employed software tools and their functionalities were presented, with each of them playing a very important part in the implementation process. Before discussing the process of hardware conversion, the dynamical system which serves as the chosen application of this thesis was introduced. This system models the dynamics of an orbiting satellite, equipped with thrusters which are able to influence the satellite's motion along linear axes. Moving on, the proposed architecture for the hardware module was analyzed, with a particular emphasis on the sub-module which handles a linear system of equations. Solving this system required specialized notions pertaining to efficient hardware-oriented programming, along with dedicated numerical methods. Finally, the hardware requirements of the designed algorithm are presented, along with a demonstration of the module's functionality. Simulations were undertaken using the Vitis HDL simulator, which are meant to replicate the behavior of the algorithm, as if it were mounted on an FPGA platform.

After thorough analysis of the closed-loop system's performance, achieved using the Hands-Off Control algorithm, I have come to the following conclusion. For a system with slow dynamics, the control algorithm accomplishes the objective of steering the system to a given state, even in the presence of persistent external disturbance. The actuators are turned on for very short time intervals, which are separated by long periods of complete inactivity. Although this results in a very low fuel consumption, the use of this control algorithm impacts performance indicators, such as overshoot, undershoot, transient time and steady-state error. If such indicators are, indeed, vital for the system which is being controlled, then the use of this control algorithm may not be the best choice. However, if the aforementioned indicators are not of great interest, and the only goal is to bring and maintain the system to a set of coordinates (within its state space) at a meager cost in energy consumption, then the control algorithm presented in this thesis is a promising candidate. Such instances of design requirements are often met in aerospace applications, where a certain position/orientation needs to be maintained under the influence of external disturbance (gravitational pull, sun pressure, atmospheric drag), all while having a limited amount of energy available for steering the space vehicle.

Regarding the actual utility of a dedicated hardware implementation for the control algorithm, the verdict is highly dependent on the environment in which the closed-loop system evolves, and boils down to whether the implementation achieves its purposes or not. The main objective of the hardware-oriented implementation is to reduce the computation time of the algorithm. As stated in the previous chapter, a response time of 1.49 seconds has been achieved, thanks to the chosen implementation. This value, however, can be further improved by additional optimizations, or even by sacrificing either performance or numerical precision. Nonetheless, for the present scenario in which the system's state is measured once every 6.28 seconds, a response time of 1.49 is appropriate and we consider the overall objective of the implementation to be accomplished. However, if the system's state is driven to within a small vicinity of the desired reference value, then the control algorithm will have to deal with control problems which span over short time horizons. Despite the briefness of these intervals, the output of the algorithm will still be provided after a fixed computation time of 1.49 seconds. This is an example of a situation where using the hardware implementation of the control algorithm would lead only to wasting most of the available hardware resources.

5.1 Future Work

The work presented in this thesis may act as a springboard for many avenues of investigation regarding the theoretically-oriented design of the presented algorithm, the efficiency of the proposed numerical procedures, or even the chosen hardware implementation scheme.

From a theoretical point of view, the algorithm can be adapted into a hybrid scheme, composed of multiple control laws, which would require the design and testing of an additional control law besides the L^0 optimal one. This auxiliary controller would be specialized in dealing with violent and quickly evolving disturbance-based scenarios, or it would alternatively be responsible for steering the system towards the reference trajectory, should a large deviation occur. In doing so, the L^0 optimal controller would be relegated to keeping the system in a desired state, while rejecting moderate disturbance and keeping fuel consumption at acceptably low levels, after the more complex control objectives have been accomplished by the additional controller.

From a numerical point of view, another method of making the computational procedures more efficient would be to investigate an analytical way of solving the KKT system, while leveraging knowledge about the structure of the KKT matrix. Based on the theory of matrix analysis, various computational shortcuts could be exploited which would lead to a faster response time of the algorithm discussed in this thesis.

Finally, there are also a series of modifications which can be applied to the hardware module, in order to ensure that it will be able to cover a wider range of scenarios. Firstly, the sparsity rate being imposed by the algorithm should be modifiable, without resynthesizing the procedure. Indeed, having a separate interface on which a new sparsity rate can be provided to the algorithm would be useful for scenarios in which, due to unforeseen circumstances, fuel levels would drop bellow critical levels and energy consumption would need to be drastically curtailed. In addition to this, another possible improvement concerns the hardware platform itself. A more powerful development board would allow for a more complex state-space model to be used for the computation of the control signals, thus taking into consideration a more precise description of the physical system.

6 Personal Contributions

The following table describes my personal involvement on the topics presented in this thesis:

Hands-Off Control for an Aerospace Application Robert-Antonio Stăvărache Assoc. Prof. Bogdan D. Ciubotaru Assist. Prof. Andrei Sperilă		
Nr	Activity	Time [days]
1	Research on Optimal Control Theory concepts	30
2	Research on Optimization techniques	15
3	Develop Matlab solver for Minimum Time problem using *quadprog*	10
4	Develop Matlab solver for L^1 optimal control problem using *linprog*	15
5	Develop Matlab self-triggering Hands-off Control algorithm	7
6	Document on Vitis HLS software	30
7	Research on ADMM as a general purpose optimization problem solver	5
8	Adapt ADMM solver for the Minimum Time problem	5
9	Adapt ADMM solver for the L^1 optimal control problem	5
10	Develop Matlab self-triggering Hands-off Control algorithm based on the new solver (ADMM)	3
11	Develop the C++ testbench file based on the Matlab routines	5
12	Develop the C++ module for HDL conversion	10
13	Debug, optimize and synthesize the module	20
14	Simulate the resulting HDL module	5
15	Write the documentation	20

Table 6.1: Personal contributions

7 Special Acknowledgements

I would like to express my sincere gratefulness to the Amiq Consulting team for all the support they offered me. I extend my appreciation to my mentors, Cătălina, Ovidiu and Vlad, for their continuous assistance in the preparation of the hardware implementation section of this thesis. Thanks to their vast experience in the digital electronics field, I was provided with much insightful advice regarding the design procedures. Also, thanks to them and the rest of the Amiq team, I have also received valuable suggestions and comments regarding the contents of this manuscript.

I also express my deep gratitude to my thesis coordinators, Dr. Ciubotaru and Dr. Sperilă, for the numerous recommendations regarding the complex theoretical domains into which I ventured, during the creation of this thesis. Thanks to them, I have been offered a strong basis of bibliographic references to serve as inspiration, and many ingenious ideas pertaining to the numerical implementation presented in this thesis. Most importantly, they offered me the emotional support required to pursue such a complex thesis project.

Bibliography

[1] Michael Athans and Peter L. Falb. *Optimal Control. An Introduction to the Theory and its Applications*. Dover Publications, Inc., 1966.

[2] Stephen Boyd and Craig Barrat. *Linear Controller Design: Limits of Performance*. Prentice-Hall, 1991.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[4] Stephen Boyd et al. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Now Foundations and Trends, 2011.

[5] Bogdan Dumitrescu, Boris Jora, and Corneliu Popeea. *Metode de calcul numeric matriceal. Algoritmi fundamentali*.

[6] Tingshu Hu, Zongli Lin, and Li Qiu. "An explicit description of null controllable regions of linear systems with saturating actuators". In: *Systems & Control Letters* 47 (2002), pp. 65–78.

[7] Min Jeong, Manuel Schoen, and Jürgen Biela. "When FPGAs Meet ADMM with High-level Synthesis (HLS): A Real-time Implementation of Long-horizon MPC for Power Electronic Systems". In: *11th International Conference on Power Electronics and ECCE Asia* (2023), pp. 1704–1711.

[8] Thomas Kailath. *Linear Systems*. Prentice-Hall.

[9] L. V. Kantorovich and G. P. Akilov. *Functional Analysis*. Pergamon Press, 1982.

[10] Masaaki Nagahara, Daniel E. Quevedo, and Dragan Nesic. "Maximum Hands-Off Control: A Paradigm of Control Effort Minimization". In: *IEEE Transactions on Automatic Control* 61.3 (2016), pp. 735–747.

[11] B. Stellato et al. "OSQP: an operator splitting solver for quadratic programs". In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672. DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2.

[12] Hector J. Sussmann and Jan C. Willems. "300 Years of Optimal Control: From The Brachystochrone to the Maximum Principle". In: *IEEE Control Systems* (1997).

[13] *Vitis High-Level Synthesis User Guide*. URL: https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction.