UNIVERSIDAD DE CASTILLA - LA MANCHA

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS INDUSTRIALES

CIUDAD REAL

TRABAJO FIN DE GRADO EN
INGENIERÍA MECÁNICA

N.º 19-1-225435

OPTIMAL ATTITUDE CONTROL OF SATELLITES

Autor:
VICTORIO ÚBEDA SOSA

Director:
PABLO PEDREGAL TERCERO

JULIO 2019

ACKNOWLEDGEMENTS

I am grateful to my advisor, Pablo Pedregal, for his constant support during the course of this project and for opening my mind to optimal control and variational problems. I am also very thankful that he agreed to supervise this project, which originated from an idea unrelated to his area of research. He has made a great effort to direct me towards the solution of the problem, using his own original techniques.

I would also like to thank José Carlos Bellido, for his tireless mathematical and academic advice during the four years of my degree.

I do not want to forget Francisco Ramos de la Flor and Oscar Juan Durá. I consider them to be excellent professors – I have very fond memories of their lectures on programming and physics, respectively. I am also very grateful for the opportunities that they have given me to work as a teaching assistant in their departments.

I am deep in debt to the Dynamics and Control Group at University College Dublin: Willi, Daire, Joe, David, and Paul, who gave me the wonderful opportunity to work with them on the EIRSAT-1 project [1, 2] during the summer of 2017. It was at the end of that placement when the idea for this final year project came to my mind as a way of consolidating what I had learned about attitude control, with my love of variational problems and optimal control, which I developed during my Bachelor's degree.

I would like to acknowledge the rest of the EIRSAT-1 team: Lána, Conor, David, Joe, Lorraine, Sheila, Jessika, and Ronan, for all the skills they helped me acquire as regards project management and working with teams, deadlines and milestones. They will be of incredible value to my career in the future.

I dedicate this work to my family: My mother Almudena and father Javier; to Bárbara, Javi, Ángela, and all the others that I am not able to mention owing to space limitations. Their support during these years has been relentless and has encouraged me to continue pushing forward when not all conditions were favourable.

I would like to dedicate a few lines to Ángela, who appeared in my life in October 2018, and gave me the required support and energy to keep pushing towards the completion of this first degree.

I am grateful to the Spanish education system, whose public university and scholarship programmes make it possible for people like me to receive an outstanding higher education at no cost.

An honourable mention goes to my friend Rodrigo, who passed away in September 2018. Your saxophone will never stop resonating in my heart.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Resumen de contenidos

El presente trabajo, denominado *Optimal attitude control of satellites* (Control óptimo de la orientación de satelites), pretende desarrollar algoritmos que, desde el punto de vista del Control Óptimo, sean capaces de calcular señales de control que, una vez aplicadas a los actuadores de cierto satélite, lo conduzcan desde su estado inicial a un estado final deseado.

En ingeniería aeroespacial, es a menudo importante asegurar que vehículos atmosféricos o espaciales mantengan ciertos requisitos relacionados con su trayectoria, posicionamiento u orientación. La disciplina que se encarga de asegurar estos requisitos es conocida como control de azimuth o control de orientación. Con este fin se han empleado numerosas técnicas, estudiadas por la teoría clásica de control, aunque recientemente se han aplicado otros métodos más modernos, los cuales recurren a la lógica difusa o redes neuronales.

El control óptimo es una disciplina de la matemática aplicada que trata de minimizar ciertos costes o funcionales definidos como integrales, en el que se ven involucradas variables a las cuales, por otro lado, se les obliga a ser soluciones de un sistema de ecuaciones diferenciales y quizás a cumplir con restricciones en forma de igualdad o desigualdad. Algunas de las variables involucradas en el sistema de ecuaciones están a nuestra disposición para ser elegidas según nuestro criterio. Estas son las variables del problema de optimización y son denominadas señales de control. La señal óptima de control será aquella que, siendo siendo capaz de conducir el sistema desde un estado inicial hasta uno final en un tiempo deseado, minimice dicha cantidad o coste.

En el caso de estudio, el sistema de ecuaciones diferenciales es un sistema dinámico (en el que la variable independiente es el tiempo), que describe la dinámica y cinemática rotacional de un satélite, que es tratado como un cuerpo rígido, rotando en el espacio, actuado por ciertos torques o pares, los cuales son considerados como la variable de control. Se han seguido dos estrategias distintas: en la primera, a partir de una linealización de las ecuaciones del movimiento, se ha implementado un Regulador Lineal

Cuadrático (LQR), el cual considera un coste en forma de integral con un integrando cuadrático. En el segundo caso, el cual cual constituye la mayor contribución de este trabajo, se ha desarrollado un controlador no lineal, basado en la simulación numérica de cierto problema variacional equivalente al problema de control óptimo pertinente.

El Capítulo 1 constituye una introducción, en la que se derivan las ecuaciones cinemáticas y dinámicas del sistema, y se presentan los dos problemas de control óptimo que serán el caso de estudio. En el Capítulo 2 se muestran resultados de dichos problemas de control óptimo para configuraciones con cierto interés práctico. Los Capítulos 3 y 4 exhiben, respectivamente, las herramientas matématicas necesarias para la comprensión de los algoritmos; dichas herramientas son el cálculo de variaciones y el control óptimo. En estos capítulos se exponen también los algoritmos en forma de pseudocódigo. Se ha decidido incluir el capítulo de resultados con anterioridad a la base matemática, pues son estos los que mayor interés presentan desde el punto de vista práctico. El lector interesado en los entresijos matemáticos puede dirigirse directamente a los Capítulos 3 y 4. El Capítulo 5 habla de las conclusiones del trabajo y propone posibles continuaciones a este proyecto. Por último, en el Apéndice A se expone el código fuente de los algoritmos en lenguaje *Fortran 95*.

1 2 Introduction to attitude control

Attitude control is a branch of aeronautical and aerospace engineering that is in charge of controlling the orientation of a satellite, or any kind of spacecraft with respect to a predefined reference frame, celestial entity or nearby objects and fields.

A common requirement for a spacecraft is that its orientation to be maintained, with a certain amount of accuracy, fixed with respect to a particular reference frame. This frame can be inertial, centred at the centre of the celestial body around which the satellite is orbiting, fixed to the sun, or, alternatively, any other conveniently defined reference.

The design of attitude determination and control systems (ADCS) is normally left to control engineers, who have to deal with the design of the control algorithm, the selection or manufacturing of hardware, and the implementation of the actuation method that is in charge of providing the necessary torque that actuates the satellite in accordance with the desired attitude changes calculated by the algorithm, or instructed from Earth's ground.

The aim of the present work is to propose alternative control algorithms, which differ in their essence from the solutions adopted in previous literature. Previously adopted techniques range from those provided by classical control theory: regulators of type PI (Proportional-integral), PD (Proportional-derivative) or PID (Proportional-integral-derivative) [3, 4, 5, 6, 7] to more advanced techniques, including Wave-Based-Control [8, 1, 2], LQR (Linear-quadratic regulator) [9], fuzzy logic [3, 10] or even artificial intelligence and neural networks [11].

In [12], Crouch uses geometric control theory to control a satellite equipped with gas actuators. In his work [13], Krishnan also considers the problem of attitude control with gas actuators, but in the case of under-actuated satellites. In [3], Wisniewski studies the attitude control of satellites under magnetic actuation using both linear and non-linear control theory. Moreover Wisniewski in [14], also studies means of controlling a magnetically actuated spacecraft, this time from the viewpoint of optimal control. The same goal is achieved by Lovera, De Marchi and Bittanti in [15] and by Lovera and Silani in [16], this time from a slightly different perspective: optimal periodic control. In [9], Lovera and Varga developed several methods for discrete-time optimal attitude control.

In this project, two different algorithms are proposed. The first, which was used as a first approach to convince the author that the problem could be successfully tackled, relied on an initial linearisation of the problem about a stationary point, thus avoiding the non-linearity present in the state equation. A quadratic cost on state and control was then considered, which led to a typical Linear-Quadratic Regulator (LQR) problem with initial and final state conditions. The problem was then approximated using numerical methods.

The second algorithm proposed, which is the main focus of the present work, is based on an optimization method that relies on numerically solving a variational formulation equivalent to an optimal control problem with constraints in the form of inequalities. This method was proposed by Pedregal in [17, 18, 19] and has been proved to work in numerous situations and problems in science and engineering, particularly the optimal control problem studied in the present work.

The object of this chapter is to produce a dynamical system that will replicate the dynamic behaviour of the spacecraft that we shall consider. This dynamical system will, in essence, be worked out from the conservation of angular momentum and a certain kinematic representation in terms of quaternions. These will lead to a system of ordinary differential equations describing the dynamical system that we intend to stabilize.

The equations of motion will correspond to a notional rigid satellite orbiting around a larger celestial object and equipped with actuators, which provide full control over the torques that are exerted on its body.

As might be expected, a number of technical aspects at which must be studied if a thorough understanding of a typical engineering design of an ADCS is to be pursued. These include, but are not limited to: orbit determination, actuator design, and the study of disturbances such as aerodynamic drag or gravity gradient. Nonetheless, we will not have the opportunity to discuss these in the present work. We refwe the interested reader to specialized books and doctoral theses on the subject, such as [3, 20, 21, 22, 4, 23, 7], along with some journal and conference papers, such as [24, 25, 26, 27, 28, 29, 30].

Chapter 1 is an introduction, in which the kinematic and dynamic equations of motion are derived, and the two optimal control problems that are object of study are presented. In Chapter 2, a number of simulations, based on practical manoeuvres are showcased. Chapters 3 and 4 are devoted to the mathematical background necessary to understand the proposed control algorithms. Likewise, they present the algorithms in pseudocode form. The reader interested only in the mathematical background can directly refer to Chapters 3 and 4. In chapter 5 conclusions are presented and potential continuations of this work are proposed. Finally, in Chapter A, the algorithms in *Fortran 95* language are exposed.

1.3 Reference frames

In order to endow the problem with an adequate theoretical framework, it is necessary to define a set of reference frames in which the equations of motion will be written. The satellite is considered to be a rigid body rotating in space; the set of reference frames is then conveniently defined, following the trends of previous work in literature [23, 4, 7]:

- **Earth Centred Inertial (ECI) reference frame.** The origin of this reference frame is placed at the centre of the Earth. Its Z axis is normal to the equatorial plane and parallel to the Earth's rotational axis, pointing towards the Earth's North Pole. The X axis points towards the vernal equinox. Finally, the Y axis completes the right-handed orthonormal frame.

- **Earth-Centred/Earth-Fixed (ECEF) reference frame.** Its origin is also placed at the centre of mass of the Earth, with the Z axis pointing towards Earth's North Pole. Its X axis is constantly pointing in the direction of the intersection between the Greenwich meridian and the Earth's equator, and it consequently rotates with the Earth, sharing its angular rate. The Y axis completes the right-handed orthonormal frame.

- **Local-Vertical/Local-Horizontal (LVLH) reference frame - Orbital Frame.** Its origin is now located at the centre of mass of the spacecraft. The Z axis points towards the centre of mass of the Earth, aligned with the nadir direction, while the Y axis is parallel to the orbit angular velocity (normal to the equatorial plane). The X axis completes the right-handed orthonormal triad.

- **Body-Fixed Frame.** This frame is similarly placed at the centre of mass of the satellite. However, it is chosen to be aligned with the principal axes of inertia of the spacecraft. The orientation of this frame with respect to the inertial (ECI) frame is described by means of the quaternion parametrization discussed below.

Our main concern will be the relative orientation between the ECI and Body Fixed Frame, as it essentially describes the relative orientation of the satellite with respect to a frame that moves at speeds which are sufficiently low to be considered as inertial. The angular velocities and orientation parameters (quaternions, Euler angles, etc) used from now on in this work describe the orientation of the Body Fixed Frame with respect to the inertial (ECI) frame. However, all four classically defined frames are presented for reference.

1.4 Dynamic equations of motion

As stated previously, the spacecraft is considered to be a rigid body rotating in space. If we choose to make our calculations in the ECI (inertial) frame, the law of conservation of angular momentum states that the variation in angular momentum over time equals the sum of all external torques exerted on the satellite.

$$\frac{d\mathbf{L}}{dt} = \mathbf{M} \tag{1.1}$$

We introduce the expresion $\mathbf{L} = \boldsymbol{J}\boldsymbol{\omega}$, where $\boldsymbol{\omega}$ is the angular velocity of the satellite and $\boldsymbol{J}$ is the tensor of inertia of the spacecraft, and continue to develop equation (1.1), differentiating with respect to the inertial frame, yet writing the components of each vector in the body-fixed frame, which yields the introduction of the term $\boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega}$:

$$\boldsymbol{J}\frac{d\boldsymbol{\omega}}{dt} + \boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} = \mathbf{M} \tag{1.2}$$

Furthermore, we assume that, as is the case of most real spacecraft designs, the body fixed frame is placed in such a way that the tensor of inertia is diagonal, and its elements are the three principal moments of inertia:

$$\boldsymbol{J} = \begin{pmatrix} J_1 & 0 & 0 \\ 0 & J_2 & 0 \\ 0 & 0 & J_3 \end{pmatrix} \tag{1.3}$$

To clarify, let us at this point recall that equation (1.2), which we have derived from conservation of angular momentum, is essentially a first order system of ordinary differential equations. Considering that $\boldsymbol{J}$ is a tensor of size 3×3, and writing $\boldsymbol{\omega}$ and $\mathbf{M}$ as 3×1 column vectors, the system (1.2) implies three ordinary differential equations.

As we shall see, the torque $\boldsymbol{M}$ exerted on the satellite will indeed be the control variable, that is to say, a quantity or signal that we can set to our convenience with the aim of modifying the behaviour of the dynamics of the satellite, controlling its attitude and angular velocity.

1.5 Attitude representation parameters

The evolution of the angular velocity has been presented by means of the dynamical system (1.2). Note, however, that no equation involving the dynamics of the actual attitude or orientation has been stated and its evolution law is, therefore, still unknown. We now aim to find an additional set of ordinary differential equations that will determines this dynamics. These equations will, in fact, relate to a set of parameters determining this orientation, their derivatives, and the angular velocity.

At this point, a decision as to which kinematic representation to choose must be made. The most common attitude parameters employed throughout literature are

- Direction cosine matrix (DCM).
- Euler angles.
- Euler axis and principal angle.
- Unit quaternions or Euler symmetric parameters.
- (Modified) Rodrigez parameters.
- Cailey-Klein matrices.

Each of the above has its own advantages and drawbacks. However, when it comes to simulation and computation, the unit quaternion is a clear winner, as it is the most widely used option throughout literature

1.5.1 Euler angles

The concept of Euler angles is a mathematical construction used to represent the orientation of a moving body with respect to a given reference frame. There is an extensive theory behind this concept; however, we are only interested in Euler angles for representation purposes, as they provide a very intuitive idea of how a body is oriented in space.

As mentioned earlier, two reference frames will be considered, both of which have the same origin: a reference frame xyz aligned with the principal axes of the satellite, which can rotate freely with respect to a non-moving, inertial frame XYZ. Both xyz and XYZ are positively oriented ortonormal triads.

In this situation, a sequence of three rotations can be used to describe the orientation of xyz with respect to XYZ. There are, in fact, numerous conventions for the definition of Euler Angles. We choose the *roll-pitch-yaw* convention, as it is one of those most frequently used in engineering applications. According to this convention, a rotation represented by the ordered triad $\Phi = (\phi, \theta, \psi) \in [-\pi, \pi] \times [-\pi, \pi] \times [-\pi, \pi]$ is described as follows:

We imagine the xyz perfectly aligned with XYZ. As XYZ stays fixed, xyz then overcomes a sequence of rotations:

- The first, which is denominated as *yaw* is a rotation of angle ϕ about the z axis.
- The second, which is denominated as *pitch* is a rotation of angle θ about the y axis.
- The third, which is denominated as *roll* is a rotation of angle ψ about the x axis.

Positive angles are taken as follows: a rotation of $\pi/2$ about x maps the y axis onto the z axis, a rotation of $\pi/2$ about y maps the z axis onto the x axis, and a rotation of $\pi/2$ about z maps the x axis onto the y axis.

1.5.2 Unit quaternions

Unit quaternions, also known as Euler symmetric parameters, are 4-tuples of mutually dependent parameters $\boldsymbol{q} = (q_1, q_2, q_3, q_4) \in \mathbb{R}^4$ extensively used to represent rotations in areas such as control engineering and computer graphics. A quaternion is generally written in the form $\boldsymbol{q} = \eta + \epsilon_1 i + \epsilon_2 j + \epsilon_3 k$, and in some respects *expands* the set of complex numbers $w = \alpha + \beta i$, which, if multiplying another complex number, can be viewed as an operator that rotates and stretches a vector represented as a complex number. For instance, a quaternion can describe rotations and stretches when pre-multiplying other quaternions. A quaternion representing a rotation of an angle α around an axis defined by the unit vector $\boldsymbol{v}$ is given by [23]:

$$\begin{cases} q_1 = \epsilon_1 = v_1 sin(\alpha/2) \\ q_2 = \epsilon_2 = v_2 sin(\alpha/2) \\ q_3 = \epsilon_3 = v_3 sin(\alpha/2) \\ q_4 = \eta = cos(\alpha/2) \end{cases} \tag{1.4}$$

The quantity $\eta = q_4$ is called the *scalar part* of the quaternion, whereas $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \epsilon_3)$ is its vector part.

It is clear from the previous equations that a quaternion for attitude representation needs to have unit norm; that is $\|\boldsymbol{q}\|^2 = q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$. Intuitively, this restriction corresponds to the fact that in attitude kinematics, we shall be dealing with pure rotations with no stretches [23, 31, 32].

Some benefits of using quaternions among all possible attitude representation include: [6, 33, 34]

- Quaternion kinematics are not singular, unlike other representations which imply division by zero for certain orientations.
- In contrast to, say, Euler Angles or the Euler axis-principal angle representation, the kinematics equations of motion in terms of quaternions do not require trigonometric calculations, which are generally computationally expensive.

1.6 Conversion between attitude representations

In this section, we present the equations that will be employed to convert between the two attitude representation parameters used in this work. Please recall that we are working with the convention $\boldsymbol{q} = (\boldsymbol{\epsilon}, \eta)$ for quaternions, and the *roll, pitch, yaw* convention for Euler Angles. Both equations are taken from [34].

1.6.1 Euler angles to quaternion

Given a set of three Euler angles $\Phi = (\phi, \theta, \psi)$ representing a certain orientation according to the *roll, pitch, yaw* convention, its equivalent quaternion is given by

$$\boldsymbol{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{pmatrix} = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \eta \end{pmatrix} = \\ = \begin{pmatrix} \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) + \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \end{pmatrix} \quad (1.5)$$

1.6.2 Quaternion to Euler angles

Conversely, if a quaternion $\boldsymbol{q} = (q_1, q_2, q_3, q_4) = (\epsilon_1, \epsilon_2, \epsilon_3, \eta)$ is given, its equivalent triad of Euler angles is calculated as follows

$$\Phi = \begin{pmatrix} \psi \\ \theta \\ \psi \end{pmatrix} = \begin{pmatrix} Arg\,[2q_4q_3 + 2q_1q_2 + i(1 - 2(q_2^2 + q_3^2))] \\ asin\,(2q_4q_2 - 2q_3q_1)) \\ Arg\,[2q_4q_1 + 2q_2q_3 + i(1 - 2(q_1^2 + q_2^2))] \end{pmatrix} \quad (1.6)$$

where $Arg(z)$, the principal argument of the complex number z, is taken such that $-\pi < Arg(z) \leq \pi$. Analogously, it is considered that $-\pi < asin(x) \leq \pi$ for every real number x.

1.7 Kinematic equations of motion in terms of quaternions

If unit quaternions are used for attitude representation, the dynamics of that attitude are given by [23, 6]:

$$\frac{d\boldsymbol{q}}{dt} = \frac{1}{2}\Omega(\boldsymbol{\omega})\boldsymbol{q} \quad (1.7)$$

where $\boldsymbol{q}$ is a column vector of size 4×1 and Ω is the following skew-symmetric matrix:

$$\Omega(u_1, u_2, u_3) = \begin{pmatrix} 0 & u_3 & -u_2 & u_1 \\ -u_3 & 0 & u_1 & u_2 \\ u_2 & -u_1 & 0 & u_3 \\ -u_1 & -u_2 & -u_3 & 0 \end{pmatrix} \quad (1.8)$$

Note that if we write $\boldsymbol{q} = \begin{pmatrix} \boldsymbol{\epsilon} \\ \eta \end{pmatrix}$, we have

$$\begin{cases} \dfrac{d\boldsymbol{\epsilon}}{dt} = -\dfrac{1}{2}\boldsymbol{\omega} \times \boldsymbol{\epsilon} + \dfrac{1}{2}\eta\boldsymbol{\omega} \\ \dfrac{d\eta}{dt} = -\dfrac{1}{2}\boldsymbol{\omega}^t\boldsymbol{\epsilon} \end{cases} \tag{1.9}$$

1.8 Non-linear equations of motion

If $\boldsymbol{\omega}$ denotes the angular velocity of the spacecraft with respect to the inertial reference frame, in terms of the body fixed frame, and $\boldsymbol{u}$ represents the torques exerted on the satellite, the conservation of angular momentum yields [1]

$$J\boldsymbol{\omega}' = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{u} \tag{1.10}$$

Moreover, the kinematic equations in terms of quaternions take the form

$$\boldsymbol{q}' = \frac{1}{2}\Omega(\boldsymbol{\omega})\boldsymbol{q} \tag{1.11}$$

where Ω is given by (1.8).

In summary, the evolution of dynamics of the satellite is determined by the following system of non-linear differential equations

$$\begin{cases} J\boldsymbol{\omega}' = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{u} \\ \boldsymbol{\epsilon}' = -\dfrac{1}{2}\boldsymbol{\omega} \times \boldsymbol{\epsilon} + \dfrac{1}{2}\eta\boldsymbol{\omega} \\ \eta' = -\dfrac{1}{2}\boldsymbol{\omega}^t\boldsymbol{\epsilon} \end{cases} \tag{1.12}$$

where $\boldsymbol{q} = (\boldsymbol{\epsilon}, \eta)$ is a quaternion that represents the orientation of the body with respect to the inertial frame, $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \epsilon_3)$ is the vector part of the quaternion, and η its scalar part.

1.9 Linearised equations of motion

In this section, we aim to derive a linearisation of the equations of motion (1.12). The linearisation will involve only the first three components of the quaternion (its vector part $\boldsymbol{\epsilon}$). This will decrease the number of variables under consideration, which simplifies subsequent computations. Once the vector part $\boldsymbol{\epsilon}$ of the quaternion has been determined, it is possible to recover the scalar part η by simply imposing $||\boldsymbol{q}||^2 = 1$, which gives

$$||\boldsymbol{q}|| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2} = \sqrt{\epsilon_1^2 + \epsilon_2^2 + \epsilon_3^2 + \eta^2} \tag{1.13}$$

[1]In what follows, prime super-indices indicate derivatives with respect to time.

That is

$$\eta = \sqrt{1 - ||\boldsymbol{\epsilon}||^2} = \sqrt{1 - \epsilon_1^2 - \epsilon_2^2 - \epsilon_3^2} \tag{1.14}$$

As derived earlier, and using the notation previously discussed, the non-linear equations of motion are given by

$$\begin{cases} J\boldsymbol{\omega}' = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{u} \\ \boldsymbol{\epsilon}' = -\dfrac{1}{2}\boldsymbol{\omega} \times \boldsymbol{\epsilon} + \dfrac{1}{2}\eta\boldsymbol{\omega} \\ \eta' = -\dfrac{1}{2}\boldsymbol{\omega}^t\boldsymbol{\epsilon} \end{cases} \tag{1.15}$$

From now on, we shall ignore the existence of η, as it is uniquely determined by $\eta = \sqrt{1 - ||\boldsymbol{\epsilon}||^2}$.

The main assumption is that of considering J, the inertia tensor of the spacecraft, as a diagonal matrix

$$J = \begin{pmatrix} J_1 & 0 & 0 \\ 0 & J_2 & 0 \\ 0 & 0 & J_3 \end{pmatrix} \tag{1.16}$$

This assumption is not at all restrictive, as it suffices to considered the body fixed reference frame aligned with the principal axes of inertia.

In Lemma 1 of [35], Yang proved that there is a one-to-one relation between $\boldsymbol{\epsilon}'$ and $\boldsymbol{\omega}$, which is given by

$$\boldsymbol{\epsilon}' = \frac{1}{2}G(\boldsymbol{\epsilon})\boldsymbol{\omega} \tag{1.17}$$

where

$$G(\boldsymbol{\epsilon}) = \begin{pmatrix} \sqrt{1 - ||\boldsymbol{\epsilon}||^2} & -\epsilon_3 & \epsilon_2 \\ \epsilon_3 & \sqrt{1 - ||\boldsymbol{\epsilon}||^2} & -\epsilon_1 \\ -\epsilon_2 & \epsilon_1 & \sqrt{1 - ||\boldsymbol{\epsilon}||^2} \end{pmatrix} \tag{1.18}$$

The previous equations can be written in compact form as

$$\begin{cases} J\boldsymbol{\omega}' = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{u} \\ \boldsymbol{\epsilon}' = \dfrac{1}{2}G(\boldsymbol{\epsilon})\boldsymbol{\omega} \end{cases} \tag{1.19}$$

It is to these equations that a first order Taylor expansion centred at a stationary point ($\boldsymbol{\omega} = \mathbf{0}$, $\boldsymbol{\epsilon} = \mathbf{0}$) is applied, which yields the linearised system sought [2] [3]

$$\begin{pmatrix} \boldsymbol{\omega}' \\ \boldsymbol{\epsilon}' \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 0_{3\times3} & 0_{3\times3} \\ I_3 & 0_{3\times3} \end{pmatrix}\begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{\epsilon} \end{pmatrix} + \begin{pmatrix} J^{-1} \\ 0_{3\times3} \end{pmatrix}\boldsymbol{u} \tag{1.20}$$

Note that J is trivially invertible, as it represents a tensor of inertia, and is, therefore, a positive definite matrix.

[2] I_n denotes the n-th order identity matrix , while $0_{n\times m}$ represents the null matrix of size $n \times m$.
[3] This is equation (7) in [33].

In [35], [33] and [6], Yang proved that this linear model is fully controllable. This fact will be corroborated later in simulation.

1.10 Non-linear optimal control problem

The main problem that this works aims to solve is that of finding an optimal choice of $\boldsymbol{u} : [0, T] \to \mathbb{R}^3$ in the sense that a functional

$$\boldsymbol{u} \longmapsto \int_0^T F(t, \boldsymbol{\omega}, \boldsymbol{q}, \boldsymbol{u})\, dt \tag{1.21}$$

is minimized amongst all possible choices of $\boldsymbol{u}$, in addition to complying with

$$\begin{cases} J\boldsymbol{\omega}' = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{u} & \text{in } [0, T] \\ \boldsymbol{\epsilon}' = -\dfrac{1}{2}\boldsymbol{\omega} \times \boldsymbol{\epsilon} + \dfrac{1}{2}\eta\boldsymbol{\omega} & \text{in } [0, T] \\ \eta' = -\dfrac{1}{2}\boldsymbol{\omega}^t \boldsymbol{\epsilon} & \text{in } [0, T] \end{cases} \tag{1.22}$$

and

$$\begin{cases} \boldsymbol{\omega}(0) = \boldsymbol{\omega}_0, \quad \boldsymbol{q}(0) = \boldsymbol{q}_0 \\ \boldsymbol{\omega}(T) = \boldsymbol{\omega}_T, \quad \boldsymbol{q}(T) = \boldsymbol{q}_0 \end{cases} \tag{1.23}$$

where the time interval $[0, T]$, a positive definite diagonal matrix J of order 3, and initial and final conditions $(\boldsymbol{\omega}_0, \boldsymbol{q}_0)$, $(\boldsymbol{\omega}_T, \boldsymbol{q}_T)$ are given.

More precisely, if we denote

$$\begin{aligned} \boldsymbol{x} &= (\boldsymbol{\omega}, \boldsymbol{q}) = (\boldsymbol{\omega}, \boldsymbol{\epsilon}, \eta) \\ \boldsymbol{x}_0 &= (\boldsymbol{\omega}_0, \boldsymbol{q}_0) \\ \boldsymbol{x}_T &= (\boldsymbol{\omega}_T, \boldsymbol{q}_T) \end{aligned} \tag{1.24}$$

The optimal control problem can be stated as

$$\text{Minimize in } \boldsymbol{u}(t) : \quad I(\boldsymbol{u}) = \int_0^T F(t, \boldsymbol{x}(t), \boldsymbol{u}(t))\, dt \tag{1.25}$$

subject to

$$\begin{cases} x_1' = \omega_1' = x_2 x_3 (J_2 - J_3)/J_1 + u_1/J_1 \\ x_2' = \omega_2' = x_1 x_3 (J_3 - J_1)/J_2 + u_2/J_2 \\ x_3' = \omega_3' = x_1 x_2 (J_1 - J_2)/J_3 + u_3/J_3 \\ x_4' = \epsilon_1' = (x_1 x_7)/2 - (x_2 x_6)/2 + (x_3 x_5)/2 \\ x_5' = \epsilon_2' = (x_1 x_6)/2 - (x_3 x_4)/2 + (x_2 x_7)/2 \\ x_6' = \epsilon_3' = (x_2 x_4)/2 - (x_1 x_5)/2 + (x_3 x_7)/2 \\ x_7' = \eta' = -(x_1 x_4)/2 - (x_2 x_5)/2 - (x_3 x_6)/2 \end{cases} \tag{1.26}$$

and end-point conditions

$$\begin{cases} \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{x}(T) = \boldsymbol{x}_T \end{cases} \tag{1.27}$$

1 11 Linearised optimal control problem

One of the approaches shown in this work is that of using the linearised model to find approximate solutions to the optimal control problem. We shall work with the linearised model derived earlier:

$$\begin{pmatrix} \boldsymbol{\omega}' \\ \boldsymbol{\epsilon}' \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0_{3\times3} & 0_{3\times3} \\ I_3 & 0_{3\times3} \end{pmatrix} \begin{pmatrix} \boldsymbol{\omega} \\ \boldsymbol{\epsilon} \end{pmatrix} + \begin{pmatrix} J^{-1} \\ 0_{3\times3} \end{pmatrix} \boldsymbol{u} \tag{1.28}$$

Similarly, as in the non-linear case, we shall search for paths $\boldsymbol{u}[0,T] \to \mathbb{R}^3$, which minimize a functional

$$\boldsymbol{u} \longmapsto \int_0^T F(t, \boldsymbol{\omega}, \boldsymbol{\epsilon}, \boldsymbol{u})\ dt \tag{1.29}$$

subject to (1.28).

Analogous to what was done in the previous section, we denote

$$\boldsymbol{x} = (\boldsymbol{\omega}, \boldsymbol{\epsilon}) \tag{1.30}$$

In this case, we shall restrict ourselves further to a very particular integrand: we will use

$$F(t, \boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2} \left(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u} \right) \tag{1.31}$$

The matrices Q and R are, respectively, positive semidefinite and positive definite matrices.

In summary, the linearised optimal control problem reads as

$$\text{Minimize in } \boldsymbol{u} : \quad I(\boldsymbol{u}) = \frac{1}{2} \int_0^T \left(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u} \right)\ dt \tag{1.32}$$

subject to

$$\begin{cases} \boldsymbol{x}' = A\boldsymbol{x} + B\boldsymbol{u} \ \text{ in } [0,T] \\ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{x}(T) = \boldsymbol{x}_T \end{cases} \tag{1.33}$$

where the initial and final conditions $\boldsymbol{x}_0$, $\boldsymbol{x}_T$ are given, and

$$A = \begin{pmatrix} 0_{3\times3} & 0_{3\times3} \\ \frac{1}{2} I_3 & 0_{3\times3} \end{pmatrix} \tag{1.34}$$

$$B = \begin{pmatrix} J^{-1} \\ 0_{3\times3} \end{pmatrix} \tag{1.35}$$

The first equation in (1.33) can be written in explicit notation as:

$$\begin{cases} x_1' = \omega_1' = u_1/J_1 \\ x_2' = \omega_2' = u_2/J_2 \\ x_3' = \omega_3' = u_3/J_3 \\ x_4' = \epsilon_1' = x_1/2 = \omega_1/2 \\ x_5' = \epsilon_2' = x_2/2 = \omega_2/2 \\ x_6' = \epsilon_3' = x_3/2 = \omega_3/2 \end{cases} \tag{1.36}$$

This particular optimal control problem, in which we intend to stabilize a linear dynamical sytem, and seek to minimize a functional which is quadratic in state and control, is called a Linear Quadratic Regulator, or LQR for short. There are numerous techniques with which to solve an LQR problem. In [33, 6], Yang even proposes explicit feedback controls for the linearised equations of motion. However, we opt for a numerical strategy, as this approach can be extended to more general situations.

There is a detail that is worth mentioning: Once the LQR has been solved (or, for that matter, numerically aproximated), we attain a suitable control $\boldsymbol{u}$, along with the associated state $\boldsymbol{x}$. However, this state is worked out from the linearised model (1.33), which does not replicate the exact dynamics of the system as the state drifts from the stationary point $\boldsymbol{x} = \mathbf{0}$. It is for this reason that it is important to simulate the dynamics of the system using the non-linear model (1.12), when the calculated control $\boldsymbol{u}$ is applied.

CHAPTER 2

RESULTS

This chapter is devoted to the numerical approximation of solutions to two suitable optimal control problems in which the state equations are imposed as constraints. In the first approach an LQR problem is constructed with a quadratic functional on state and control and a state equation obtained from the linearised equations of motion. This first problem is tackled using classical LQR techniques. In the latter, however, we deal with the fully non-linear state law, and a more general class of functionals. The strategy is to use iterative algorithms to solve the resulting non-linear variational problem.

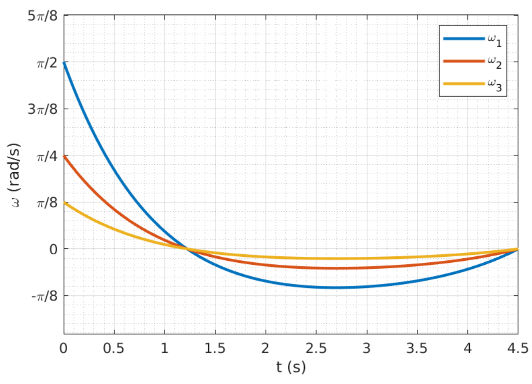
2.1 Numerical simulations

The algorithms developed are able to successfully steer the system from a prescribed state to the desired orientation and angular velocities. In this section we show a number of numerical simulations of particular manoeuvrers. Unless otherwise stated, the functional that we attempt to minimize is always (in both linear and non-linear models) a quadratic functional of the type

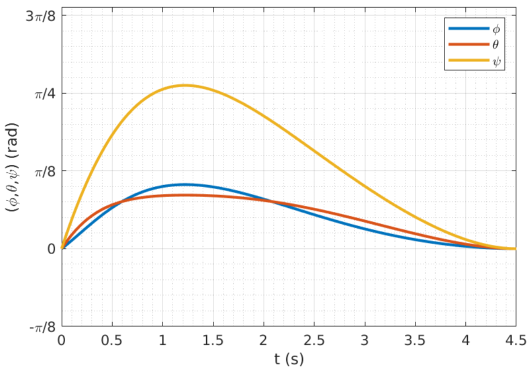
$$I(\boldsymbol{u}) = \frac{1}{2}\int_0^T \left(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u}\right) \tag{2.1}$$

where Q and R are positive semidefinite and definitive positive matrices, respectively. We chose both Q and R to be diagonal matrices. Their units are such that the cost $I(\boldsymbol{u})$ is non-dimensional; that is, the first three elements of the diagonal of Q have units of $rad^{-2}s$, while the rest are non-dimensional. R has units of $N^{-2}m^{-2}s^{-1}$.

This is merely for comparison purposes: even though the non-linear model admits a more general class of integrands, the LQR can, by definition, deal only with functionals of the type shown above.



(a) Angular velocities.



(b) Euler angles

Figure 2.1: Detumbling to a fixed orientation – Results obtained from the LQR.

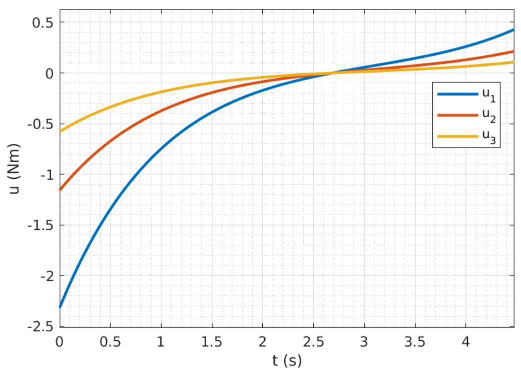


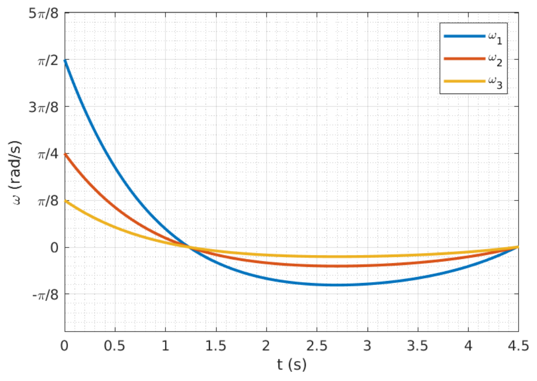
Figure 2.2: Detumbling to a fixed orientation – Control calculated with the LQR.

2.1.1 Detumbling to a predefined orientation using the linearised model. Tensor of inertia proportional to the identity

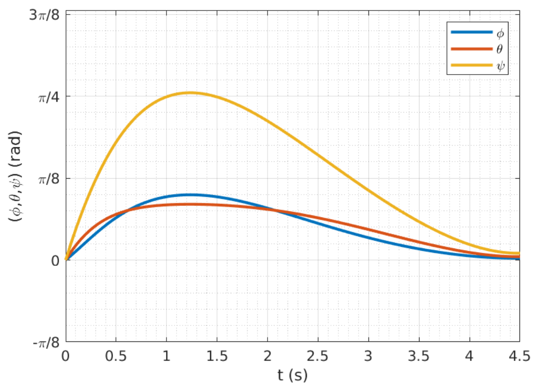
In this case, the satellite starts with a *tumbling* state of motion, that is, with angular velocity around all three axes. The algorithm attempts to take it to rest and to a predefined orientation. The control is calculated by solving the corresponding LQR problem, imposing end-point conditions. However, once the control signal has been computed, we simulate having that control as an input to the fully non-linear system. The matrix of inertia J is proportional to the identity matrix I_3. This causes the term $-\boldsymbol{\omega} \times J\boldsymbol{\omega}$ to vanish from the evolution law (1.12). The first three equations of the linearised dynamics (1.20) therefore identically replicate the non-linear dynamics. This makes the linearised model much more accurate in these situations.

Figure 2.1 shows the results of a simulation in which the satellite starts with non-zero angular velocity around the three axes at time $t = 0$ and is commanded to go to rest ($\boldsymbol{\omega} = \mathbf{0}\ rad/s$) at time $t = T = 4.5s$ with a predefined final orientation $\Phi = (\phi, \theta, \psi) = (0, 0, 0)\ rad$. The control calculated from the LQR problem is depicted in Figure 2.2.

The calculated control (Figure 2.2) is then applied in simulation to the fully non-linear system. Figure 2.3 shows the behaviour of the non-linear system when the control calculated using the LQR is applied.



(a) Angular velocities.



(b) Euler angles.

Figure 2.3: Detumbling to a fixed orientation – Results obtained from the LQR applied to the non-linear model.

Name	**Symbol**	**Value**	**Unit**
Final time	T	4.5	s
Tensor of inertia	J	I_3	$kg\ m^2$
Integrand $F = \frac{1}{2}(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u})$	Q	I_6	
	R	I_3	
Initial angular velocity	$\boldsymbol{\omega}_0$	$(\pi/2, \pi/4, \pi/8)$	rad/s
Final angular velocity	$\boldsymbol{\omega}_T$	$(0,0,0)$	rad/s
Initial orientation (Euler angles)	Φ_0	$(0,0,0)$	rad
Final orientation (Euler angles)	Φ_T	$(0,0,0)$	rad
Number of subintervals in the discretization		200	

Table 2.1: Detumbling to a fixed orientation using the LQR – Simulation parameters.

The parameters used to configure this simulation are presented in Table 2.1. Note that the units of Q and R are such that the cost $I(\boldsymbol{u})$ is non-dimensional; that is, the first three elements of the diagonal of Q have units of $rad^{-2}s$, while the rest are non-dimensional. R has units of $N^{-2}m^{-2}s^{-1}$, while I_n denotes the n-th order identity matrix.

Moreover, the simulation results are showcased in Table 2.2.

Name	Symbol	Value	Unit
Achieved final angular velocity	$\boldsymbol{\omega}(T)$	$(4.22, 2.11, 2.14)10^{-3}$	rad/s
Error in final angular velocity	$\lvert\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T\rvert$	$8.39 \cdot 10^{-3}$	rad/s
Achieved final orientation	$\Phi(T)$	$(8.48, 1.63, 3.29)10^{-3}$	rad
Error in final orientation	$\lvert\Phi(T) - \Phi_T\rvert$	$3.77 \cdot 10^{-2}$	rad
Cost	$I(\boldsymbol{u})$	4.95	
CPU time		$1.68 \cdot 10^{-3}$	s

Table 2.2: Detumbling to a fixed orientation using the LQR – Simulation results.

2.1.2 Detumbling to a predefined orientation using the linearised model. Tensor of inertia not proportional to the identity.

This situation is completely identical to the previous one: the satellite starts with angular velocity around all three axes. The algorithm then tries to steer to a predetermined orientation with zero angular velocity. The tensor of inertia J is not, however, proportional to the identity matrix, and the nonlinear effects of the term $-\boldsymbol{\omega} \times J\boldsymbol{\omega}$ are, therefore, not negligible.

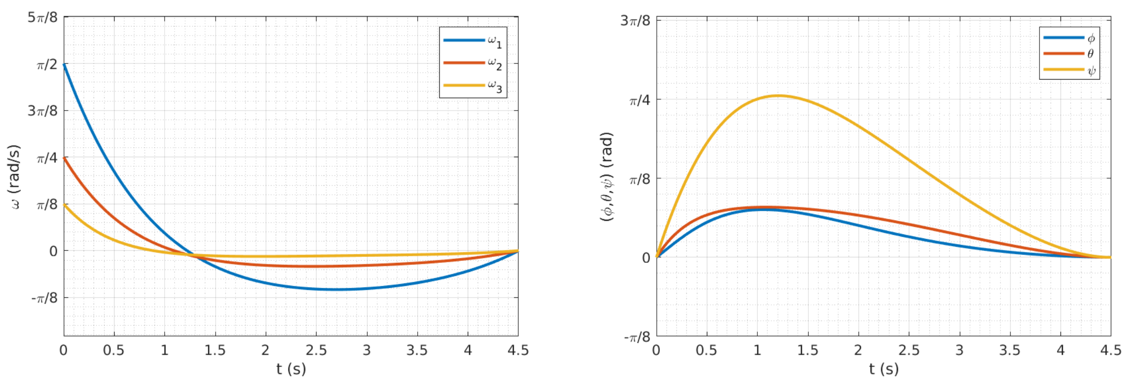
The control is calculated with the linear model. Once the control signal has been computed, we simulate having that control as an input to the fully non-linear system.

Figure 2.4 shows the results of a simulation in which the satellite starts with non-zero angular velocity around the three axes at time $t = 0$ and is commanded to go to rest ($\boldsymbol{\omega} = \mathbf{0}\ rad/s$) at time $t = T = 4.5s$ with a predefined final orientation of $\Phi = (\phi, \theta, \psi) = (0, 0, 0)\ rad$. The control calculated from the LQR problem is depicted in Figure 2.5.

The calculated control (Figure 2.5) is then applied in simulation to the fully non-linear system. Figure 2.6 shows the behaviour of the non-linear system when the control calculated using the LQR is applied.

The parameters used to configure this simulation are presented in Table 2.3.

The simulation results are showcased in Table 2.4.



(a) Angular velocities. (b) Euler angles.

Figure 2.4: Detumbling to a fixed orientation – Results obtained from the LQR.

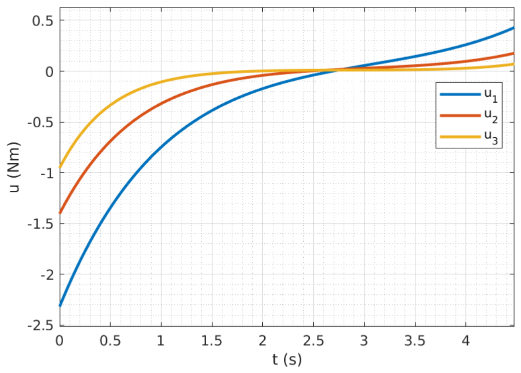
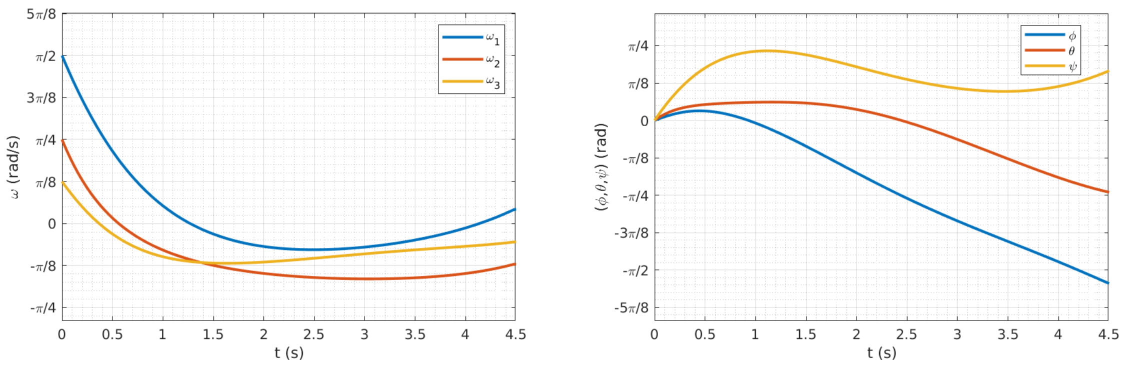


Figure 2.5: Detumbling to a fixed orientation – Control calculated with the LQR.



(a) Angular velocities. (b) Euler angles.

Figure 2.6: Detumbling to a fixed orientation – Results obtained from the LQR applied to the non-linear model.

Name	**Symbol**	**Value**	**Unit**
Final time	T	4.5	s
Tensor of inertia	J	$diag(1, 0.75, 0.5)$	$kg\ m^2$
Integrand $F = \frac{1}{2}(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u})$	Q	I_6	
	R	I_3	
Initial angular velocity	$\boldsymbol{\omega}_0$	$(\pi/2, \pi/4, \pi/8)$	rad/s
Final angular velocity	$\boldsymbol{\omega}_T$	$(0,0,0)$	rad/s
Initial orientation (Euler angles)	Φ_0	$(0,0,0)$	rad
Final orientation (Euler angles)	Φ_T	$(0,0,0)$	rad
Number of subintervals in the discretization		200	

Table 2.3: Detumbling to a fixed orientation using the LQR – Simulation parameters.

Name	Symbol	Value	Unit
Achieved final angular velocity	$\boldsymbol{\omega}(T)$	$(1.33, -3.81, -1.71)10^{-1}$	rad/s
Error in final angular velocity	$\lvert\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T\rvert$	$4.38 \cdot 10^{-1}$	rad/s
Achieved final orientation	$\Phi(T)$	$(-1.71, -0.75, 0.52)$	rad
Error in final orientation	$\lvert\Phi(T) - \Phi_T\rvert$	1.94	rad
Cost	$I(\boldsymbol{u})$	4.52	
CPU time		$2.56 \cdot 10^{-3}$	s

Table 2.4: Detumbling to a fixed orientation using the LQR – Simulation results.

It is highly evident that the linearised model does not behave well when the non linear term $-\boldsymbol{\omega} \times J\boldsymbol{\omega}$ is present – that is, when J is not proportional to the identity. While it dissipates rotational energy sufficiently well ($\boldsymbol{\omega}(T)$ is of the order of magnitude of $10^{-1}rad/s$), the final orientation is nowhere near the desired value ($|\Phi(T) - \Phi_T| = 1.94rad = 111.15°$).

2.1.3 Detumbling to a predefined orientation using the non-linear model. Tensor of inertia proportional to the identity

Analogously, the satellite begins with non-zero angular velocity around its three principal axes. The algorithm attempts to take it to a predefined orientation with no angular velocity. The tensor of inertia is set proportional to the identity. Note that the configuration of the simulation replicates that of Section 2.1.1. Nevertheless, the control is now computed using the non-linear model.

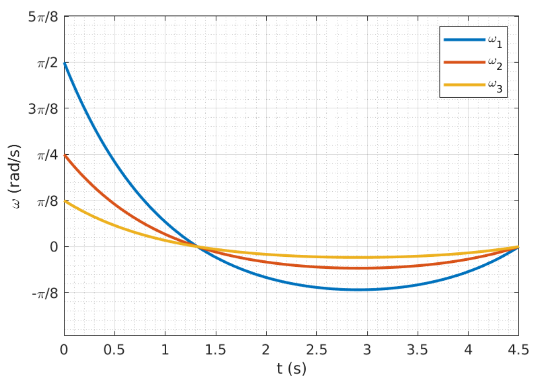
Figure 2.7 shows the results of a simulation in which the satellite starts with non-zero angular velocity around the three axes at time $t = 0s$ and is commanded to go to rest ($\boldsymbol{\omega} = \mathbf{0}\ rad/s$) at time $t = T = 4.5s$ with a predefined final orientation $\Phi = (\phi, \theta, \psi) = (0, 0, 0)\ rad$. The calculated control is depicted in figure 2.8.

In the non-linear case, constraints are enforced in a fundamentally different manner to that which occurs in the LQR. It is for this reason that it is a good idea to keep an eye on parameters that indicate that all constraints are being enforced correctly. We consequently keep track of how the norm of the quaternion evolves (as stated previously, any quaternion used to represent orientation needs to have unit norm). Figure 2.9 shows how the norm of the quaternion remains sufficiently close to 1 at all times. Keeping track of this quantity is not necessary in the linear model, as $||\boldsymbol{q}|| = 1$ is precisely the condition enforced to work out η.

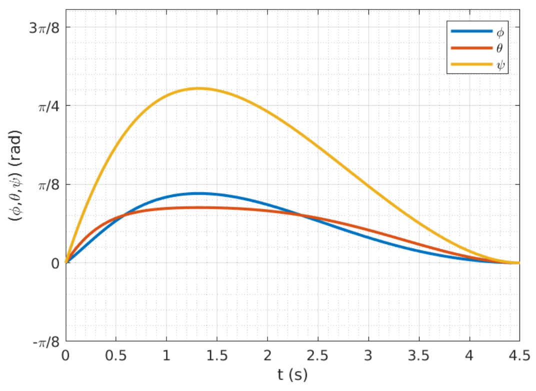
Figure 2.10 shows how the convergence parameter evolves throughout the iterative process.

The parameters used in the simutation are presented in Table 2.5. The parameter *Precision* is the threshold below which absolute values are considered to be zero. This parameter controls when the algorithm will decide that, at a certain iteration, $\boldsymbol{u}$ is sufficiently close to be an optimal solution of the control problem.

The simulation results are showcased in Table 2.6.



(a) Angular velocities.



(b) Euler angles.

Figure 2.7: Detumbling to a fixed orientation – Results obtained from the non-linear model.

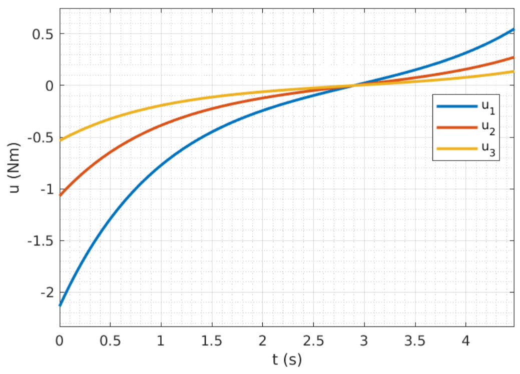


Figure 2.8: Detumbling to a fixed orientation – Control calculated with the non-linear model.

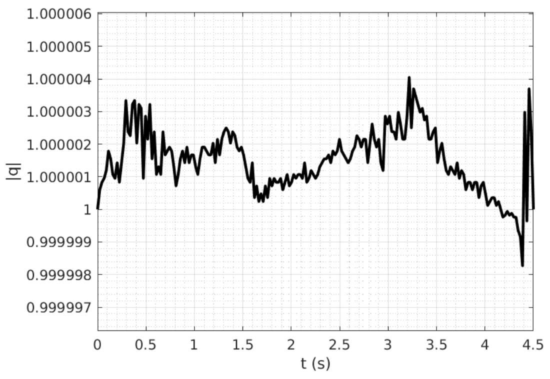


Figure 2.9: Detumbling to a fixed orientation – Norm of the quaternion.

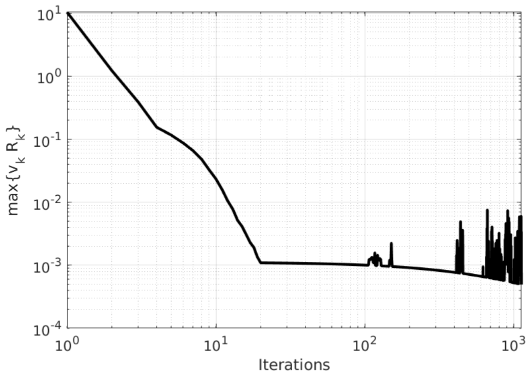


Figure 2.10: Detumbling to a fixed orientation – Evolution of the convergence parameter.

Name	**Symbol**	**Value**	**Unit**
Final time	T	4.5	s
Tensor of inertia	J	I_3	$kg\ m^2$
Integrand $F = \frac{1}{2}(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u})$	Q	I_7	
	R	I_3	
Initial angular velocity	$\boldsymbol{\omega}_0$	$(\pi/2, \pi/4, \pi/8)$	rad/s
Final angular velocity	$\boldsymbol{\omega}_T$	$(0, 0, 0)$	rad/s
Initial orientation (Euler angles)	Φ_0	$(0, 0, 0)$	rad
Final orientation (Euler angles)	Φ_T	$(0, 0, 0)$	rad
Number of subintervals in the discretization		200	
Precision		$5 \cdot 10^{-4}$	

Table 2.5: Detumbling to a fixed orientation using the non-linear model – Simulation parameters.

Name	**Symbol**	**Value**	**Unit**
Achieved final angular velocity	$\boldsymbol{\omega}(T)$	$(-1.24, -6.18, -3.09)10^{-3}$	rad/s
Error in final angular velocity	$\lvert\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T\rvert$	$7.0 \cdot 10^{-3}$	rad/s
Achieved final orientation	$\Phi(T)$	$(0.35, 0.69, 1.39)10^{-4}$	rad
Error in final orientation	$\lvert\Phi(T) - \Phi_T\rvert$	$1.59 \cdot 10^{-4}$	rad
Cost	$I(\boldsymbol{u})$	4.47	
CPU time		4.96	s

Table 2.6: Detumbling to a fixed orientation using the non-linear model – Simulation results.

2.1.4 Detumbling to a predefined orientation using the non-linear model. Tensor of inertia not proportional to the identity

The configuration of the simulation is identical to that shown in Section 2.1.1. Nevertheless, the control is now computed using the non-linear model.

Figure 2.11 represents the control calculated using the non-linear model, while Figure 2.12 shows the evolution of the state.

Figure 2.13 shows that the norm of the quaternion remains sufficiently close to 1 for all $t \in [0, T]$. Figure 2.14 shows how the convergence parameter evolves throughout the iterative process.

The parameters used in the simutation are presented in Table 2.7, while the simulation results are showcased in Table 2.8.

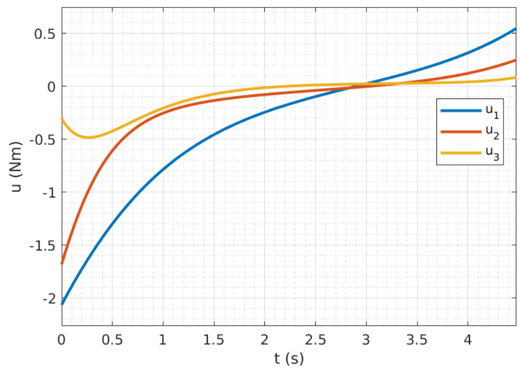
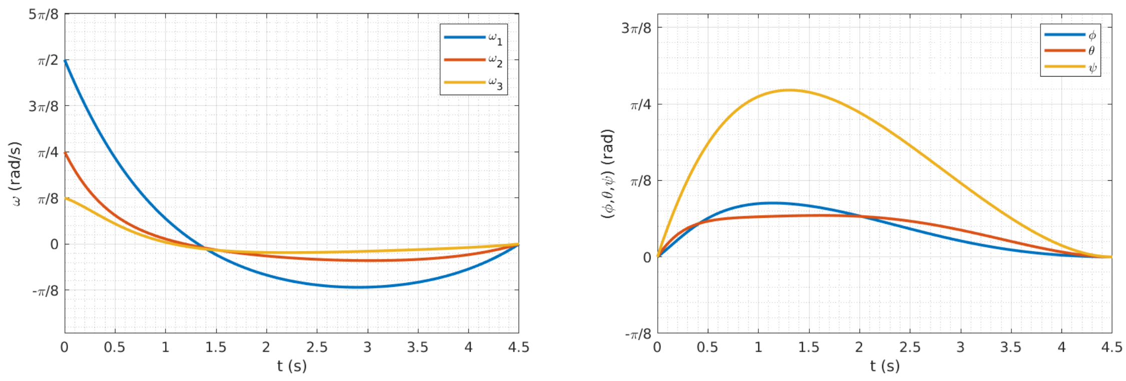


Figure 2.11: Detumbling to a fixed orientation – Control calculated with the non-linear model.



(a) Angular velocities. (b) Euler angles.

Figure 2.12: Detumbling to a fixed orientation – Results obtained from the non-linear model.

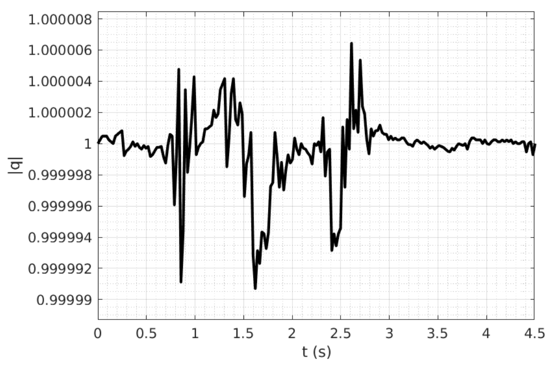


Figure 2.13: Detumbling to a fixed orientation – Norm of the quaternion.

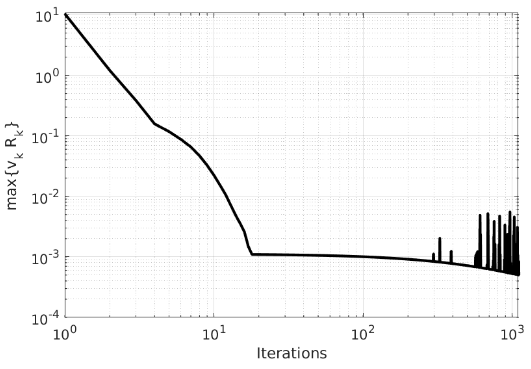


Figure 2.14: Detumbling to a fixed orientation – Evolution of the convergence parameter.

Name	**Symbol**	**Value**	**Unit**
Final time	T	4.5	s
Tensor of inertia	J	$diag(1, 0.75, 0.5)$	$kg\ m^2$
Integrand $F = \frac{1}{2}(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u})$	Q	I_7	
	R	I_3	
Initial angular velocity	$\boldsymbol{\omega}_0$	$(\pi/2, \pi/4, \pi/8)$	rad/s
Final angular velocity	$\boldsymbol{\omega}_T$	$(0, 0, 0)$	rad/s
Initial orientation (Euler angles)	Φ_0	$(0, 0, 0)$	rad
Final orientation (Euler angles)	Φ_T	$(0, 0, 0)$	rad
Number of subintervals in the discretization		200	
Precision		$5 \cdot 10^{-4}$	

Table 2.7: Detumbling to a fixed orientation using the non-linear model – Simulation parameters.

Name	**Symbol**	**Value**	**Unit**
Achieved final angular velocity	$\boldsymbol{\omega}(T)$	$(-1.24, -5.65, -1.94)10^{-3}$	rad/s
Error in final angular velocity	$\lvert\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T\rvert$	$6.1 \cdot 10^{-3}$	rad/s
Achieved final orientation	$\Phi(T)$	$(2.18, 6.36, 0.14)10^{-5}$	rad
Error in final orientation	$\lvert\Phi(T) - \Phi_T\rvert$	$1.55 \cdot 10^{-4}$	rad
Cost	$I(\boldsymbol{u})$	4.27	
CPU time		4.98	s

Table 2.8: Detumbling to a fixed orientation using the non-linear model – Simulation results.

2.1.5 Detumbling to an arbitrary orientation using the non-linear model

One of the tools that the non-linear model provides is that of not having enforce a specific final state condition. This allows us to find controls that, for example, dissipate all the rotational energy from a tumbling satellite without the need for enforcing a final orientation.

In this simulation, we use the non-linear model to steer the satellite from a tumbling state of motion (all three angular velocities are non-zero) to rest ($\boldsymbol{\omega} = \mathbf{0}\ rad/s$).

Figure 2.16 shows the results of a simulation in which the satellite starts with non-zero angular velocity around the three axes at time $t = 0$ and is commanded to go to rest ($\boldsymbol{\omega} = \mathbf{0}\ rad/s$) at time $t = T = 4.5s$. This time, no predefined final orientation is imposed. We also take advantage of the ability of the non-linear model to impose constraints in the form of inequatilities in order to request that $-1Nm \leq u_i \leq 0Nm,\ i = 1, 2, 3$.

Figure 2.15 shows the control computed using the non-linear model. Figure 2.16 displays the evolution of the state.

As occurred earlier, Figures 2.17 and 2.18 show, respectively, the norm of the quaternion in the interval $[0, T]$, and the evolution of the convergence parameter throughout the iterative process.

The parameters used in the simulation were set as in Table 2.9. The simulation results are presented in Table 2.10.

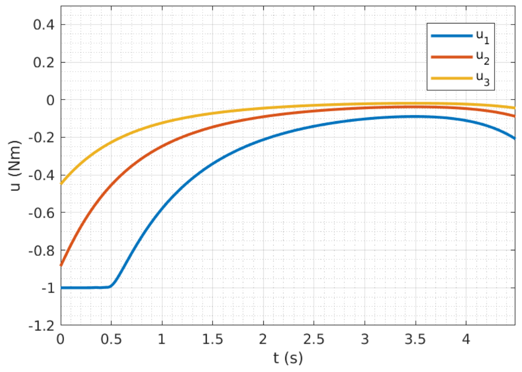
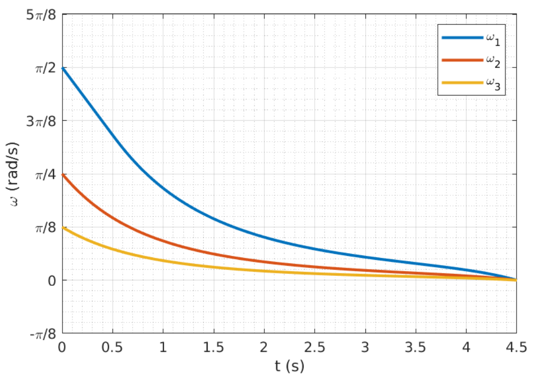
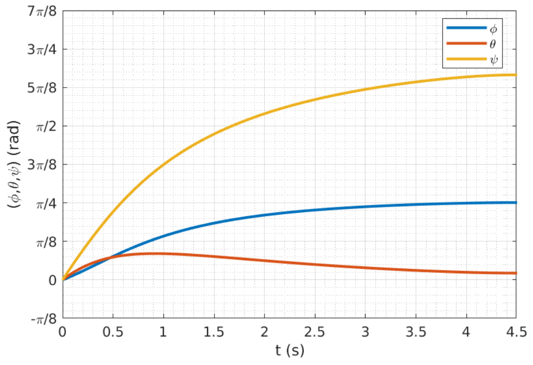


Figure 2.15: Detumbling to an arbitrary orientation – Control calculated with the non-linear model.



(a) Angular velocities.



(b) Euler angles.

Figure 2.16: Detumbling to an arbitrary orientation – Results obtained from the non-linear model.

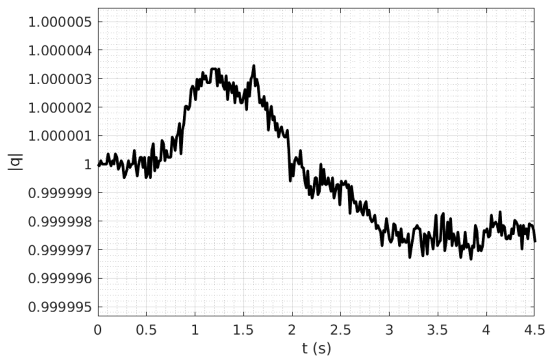


Figure 2.17: Detumbling to an arbitrary orientation – Norm of the quaternion.

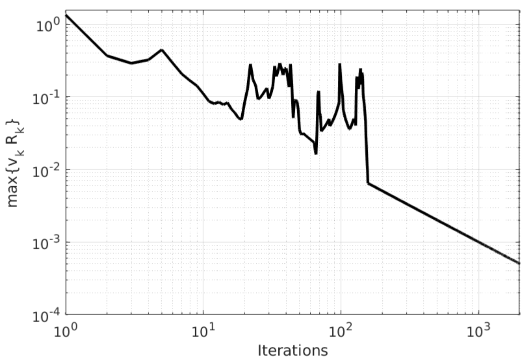


Figure 2.18: Detumbling to an arbitrary orientation – Evolution of the convergence parameter.

Name	Symbol	Value	Unit
Final time	T	4.5	s
Tensor of inertia	J	I_3	$kg\ m^2$
Integrand $F = \frac{1}{2}(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u})$	Q	I_7	
	R	I_3	
Initial angular velocity	$\boldsymbol{\omega}_0$	$(\pi/2, \pi/4, \pi/8)$	rad/s
Final angular velocity	$\boldsymbol{\omega}_T$	$(0, 0, 0)$	rad/s
Initial orientation (Euler angles)	Φ_0	$(0, 0, 0)$	rad
Number of subintervals in the discretization		300	
Precision		$5 \cdot 10^{-4}$	

Table 2.9: Detumbling to a fixed orientation using the non-linear model – Simulation parameters.

Name	Symbol	Value	Unit
Achieved final angular velocity	$\boldsymbol{\omega}(T)$	$(3.13, 1.33, 0.66)10^{-3}$	rad/s
Error in final angular velocity	$\lvert\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T\rvert$	$3.5 \cdot 10^{-3}$	rad/s
Cost	$I(\boldsymbol{u})$	1.005	
CPU time		27.32	s

Table 2.10: Detumbling to a fixed orientation using the non-linear model – Simulation results.

2.1.6 Single axis rest to rest using the linearised model

The subsequent sections demonstrate another very common situation: a single axis, rest-to-rest manoeuvre. In this case, the satellite has already been de-tumbled, signifying means that its initial angular velocity is zero. The spacecraft is then commanded to rotate around a single axis and to rest again.

Figure 2.20 shows the results of a simulation in which the satellite starts with zero angular velocity and is then commanded to rotate an angle of $\frac{\pi}{2}$ around its z axis. The control calculated using the linearised system is shown in Figure 2.19.

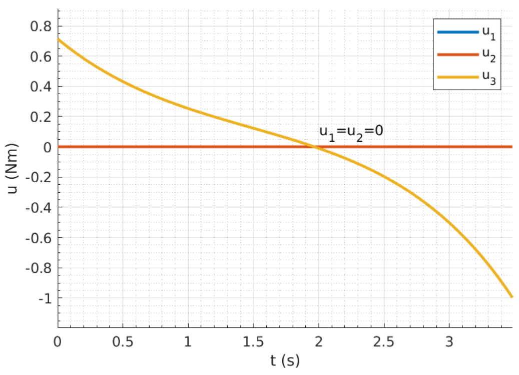
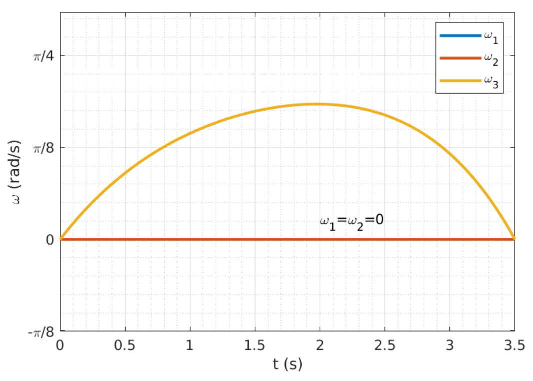
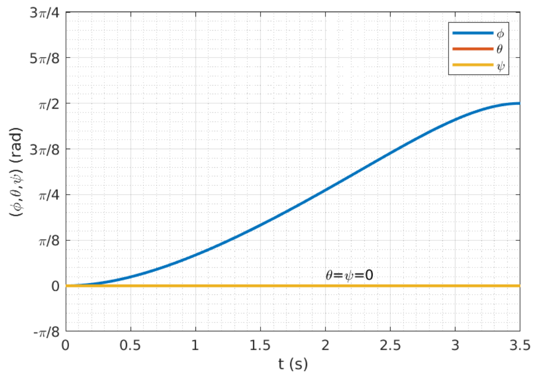


Figure 2.19: Single axis, rest to rest – Control calculated with the LQR.

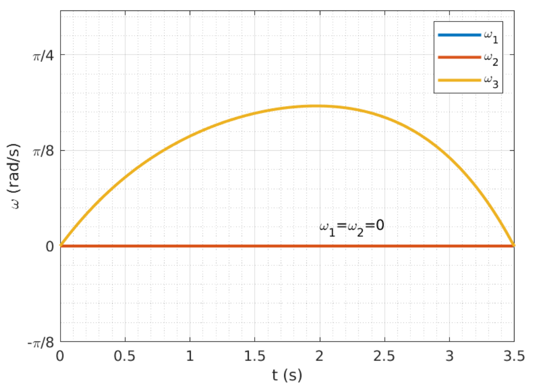


(a) Angular velocities.

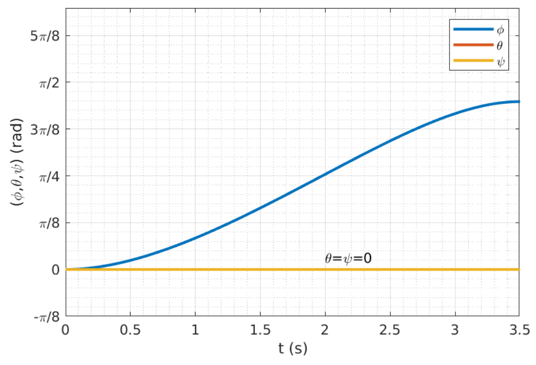


(b) Euler angles.

Figure 2.20: Single axis, rest to rest – Results obtained from the LQR.



(a) Angular velocities.



(b) Euler angles.

Figure 2.21: Single axis, rest to rest – Results obtained from the LQR applied to the non-linear model.

The calculated control is then applied in simulation to the fully non-linear system. Figure 2.21 shows the behaviour of the non-linear system when the control calculated from the LQR is applied in simulation to the fully non-linear system.

The parameters used to configure the simulation are presented in Table 2.11, while Table 2.12 shows some simulation results.

In this case, the error in the final state (after applying the control to the fully non-linear system) is

$$\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T = (0, 2.78, -1.26)10^{-3}\ rad/s$$

$$\Phi(T) - \Phi_T = (0.16, 0, 0)rad$$

, that is

$$|\Phi(T) - \Phi_T| \approx 9^\circ$$

which demonstrates that the linearised model does not work perfectly well when the initial and final states differ by a considerable amount. This is a great advantage of the non-linear model: in contrast to the LQR model, its accuracy is near perfect, and does not depend on the initial and final conditions.

Name	Symbol	Value	Unit
Final time	T	3.5	s
Tensor of inertia	J	$diag(0.5, 0.75, 1)$	$kg\ m^2$
Integrand $F = \frac{1}{2}(\boldsymbol{x}^t Q\boldsymbol{x} + \boldsymbol{u}^t R\boldsymbol{u})$	Q	I_6	
	R	I_3	
Initial angular velocity	$\boldsymbol{\omega}_0$	$(0, 0, 0)$	rad/s
Final angular velocity	$\boldsymbol{\omega}_T$	$(0, 0, 0)$	rad/s
Initial orientation (Euler angles)	Φ_0	$(0, 0, 0)$	rad
Final orientation (Euler angles)	Φ_T	$(\pi/2, 0, 0)$	rad
Number of subintervals in the discretization		200	

Table 2.11: Detumbling to a fixed orientation using the LQR – Simulation parameters.

Name	Symbol	Value	Unit
Achieved final angular velocity	$\boldsymbol{\omega}(T)$	$(0, 2.78, -1.26)10^{-3}$	rad/s
Error in final angular velocity	$\lvert\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T\rvert$	$3 \cdot 10^{-3}$	rad/s
Achieved final orientation	$\Phi(T)$	$(1.41, 0,)$	rad
Error in final orientation	$\lvert\Phi(T) - \Phi_T\rvert$	0.16	rad
Cost	$I(\boldsymbol{u})$	1.89	
CPU time		$2.56 \cdot 10^{-3}$	s

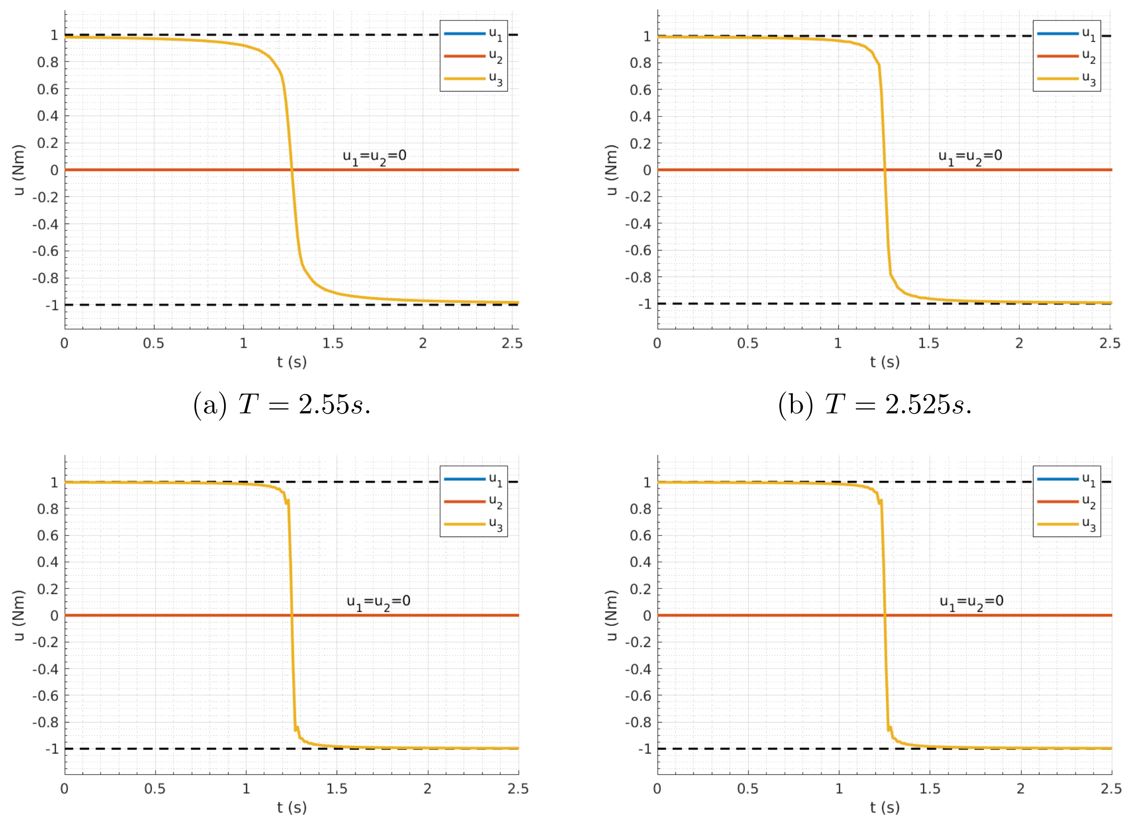
Table 2.12: Detumbling to a fixed orientation using the LQR – Simulation results.

2.1.7 Single axis rest to rest using the non-linear model

We now find ourselves in the same situation as before: starting at rest, the satellite is instructed to rotate an angle of $\frac{\pi}{2}$ around its z axis; this time we use the non-linear model, while imposing $|u_i| \leq 1Nm,\ i = 1, 2, 3$. It fact, we use this situation to showcase a profound result of optimal control: as the time interval $[0, T]$ is shortened, the control approaches that of the *bang-bang* type. That is, it abruptly switches between the two extreme values $u_{max} = 1Nm$, $u_{min} = -1Nm$.

Figure 2.22 shows the control as computed by the non-linear model for $T = 2.55s$, $T = 2.525s$, $T = 2.52s$, $T = 2.515s$. The way in which the control approaches a bang-bang strategy as the time interval is reduced is very evident.

Table 2.13 shows the parameters used in the four simulations. In Table 2.14, the worst result obtained in each of the four simulations is presented.



(a) $T = 2.55s$.

(b) $T = 2.525s$.

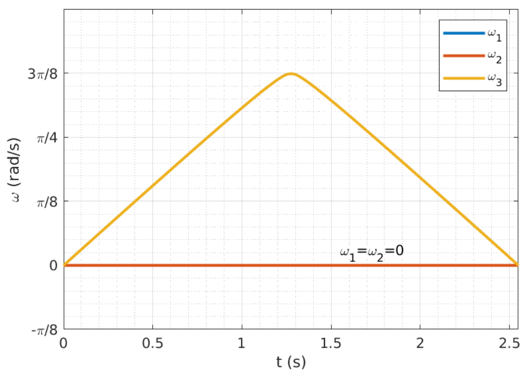
(c) $T = 2.52s$.

(d) $T = 2.515s$.

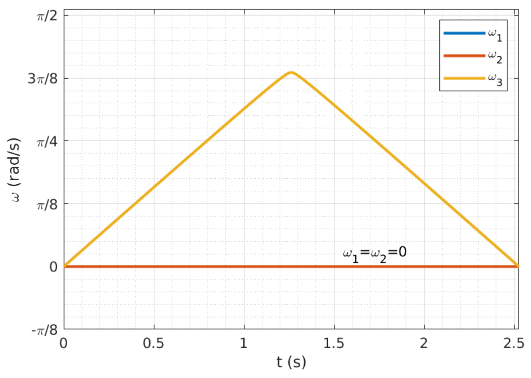
Figure 2.22: Single axis, rest to rest – Control calculated with the non-linear model.

Name	**Symbol**	**Value**	**Unit**
Final time	T	$2.55, 2.525, 2.52, 2.515$	s
Tensor of inertia	J	$diag(0.5, 0.75, 1)$	$kg\ m^2$
Integrand $F = \frac{1}{2}(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u})$	Q	$0_{7\times 7}$	
	R	$0_{3\times 3}$	
Initial angular velocity	$\boldsymbol{\omega}_0$	$(0, 0, 0)$	rad/s
Final angular velocity	$\boldsymbol{\omega}_T$	$(0, 0, 0)$	rad/s
Initial orientation (Euler angles)	Φ_0	$(0, 0, 0)$	rad
Final orientation (Euler angles)	Φ_T	$(\pi/2, 0, 0)$	rad
Number of subintervals in the discretization		200	
Precision		$5 \cdot 10^{-4}$.	

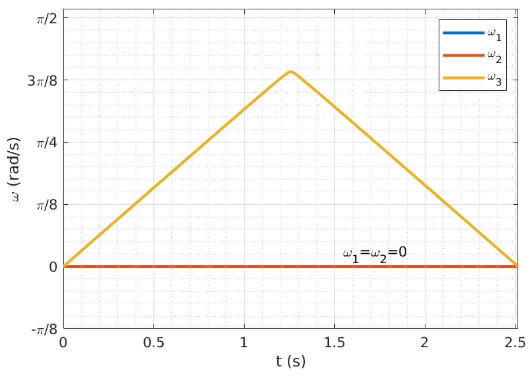
Table 2.13: Detumbling to a fixed orientation using the LQR – Simulation parameters.



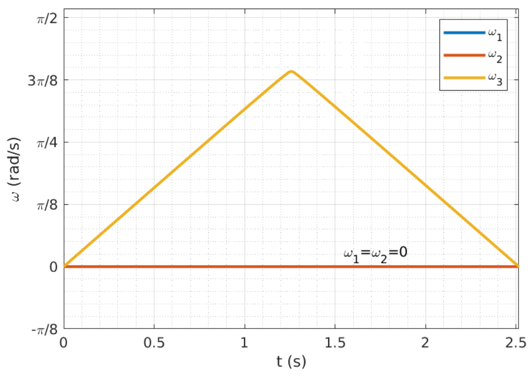
(a) $T = 2.55s$.



(b) $T = 2.525s$.



(c) $T = 2.52s$.

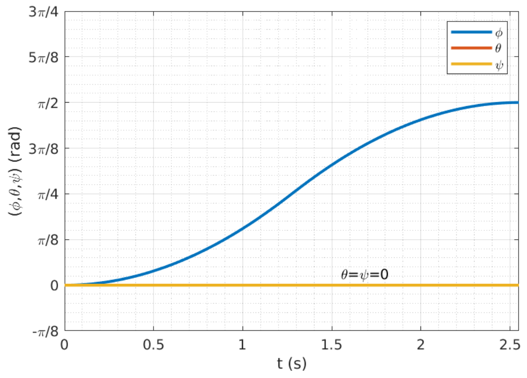


(d) $T = 2.515s$.

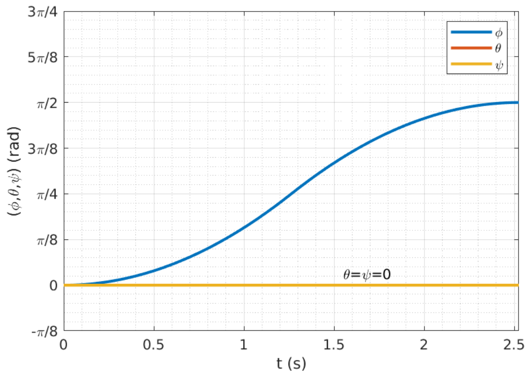
Figure 2.23: Single axis, rest to rest – Angular velocities obtained from the non-linear model.

Name	**Symbol**	**Value**	**Unit**
Achieved final angular velocity	$\boldsymbol{\omega}(T)$	$(0, 2.78, -0.56)10^{-3}$	rad/s
Error in final angular velocity	$\lvert\boldsymbol{\omega}(T) - \boldsymbol{\omega}_T\rvert$	$2.84 \cdot 10^{-3}$	rad/s
Achieved final orientation	$\Phi(T)$	$(1.568, 0,)$	rad
Error in final orientation	$\lvert\Phi(T) - \Phi_T\rvert$	$3.8 \cdot 10^{-3}$	rad
Cost	$I(\boldsymbol{u})$	$7.49 \cdot 10^{-3}$	
CPU time		23.45	s

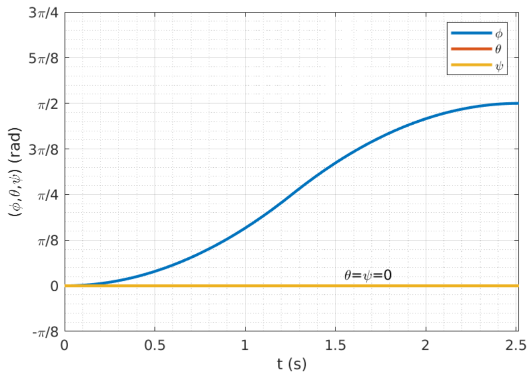
Table 2.14: Detumbling to a fixed orientation using the LQR – Simulation results.



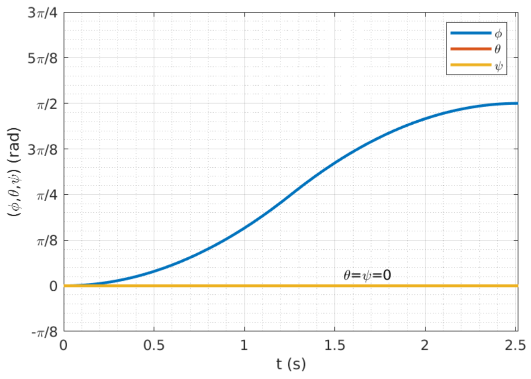
(a) $T = 2.55s$.



(b) $T = 2.525s$.

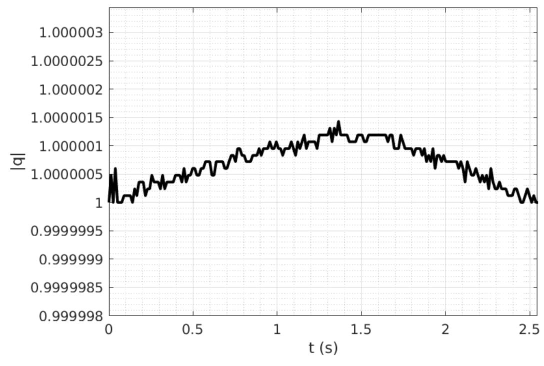


(c) $T = 2.52s$.

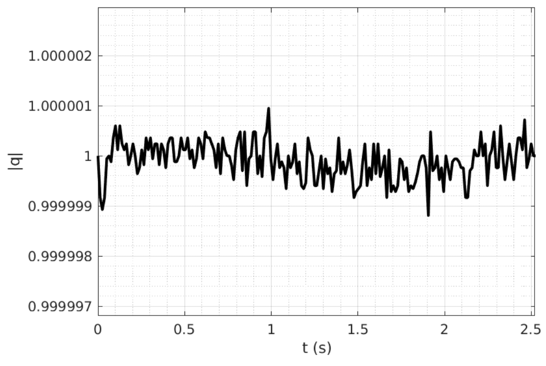


(d) $T = 2.515s$.

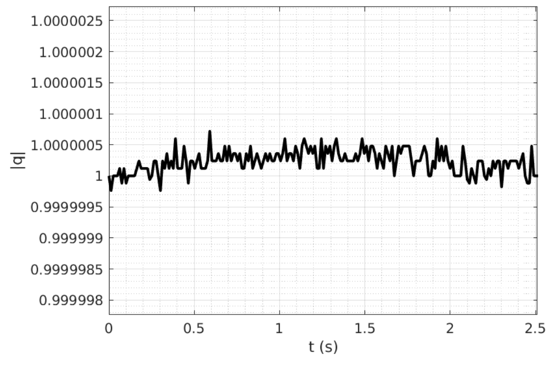
Figure 2.24: Single axis, rest to rest – Euler angles obtained from the non-linear model.



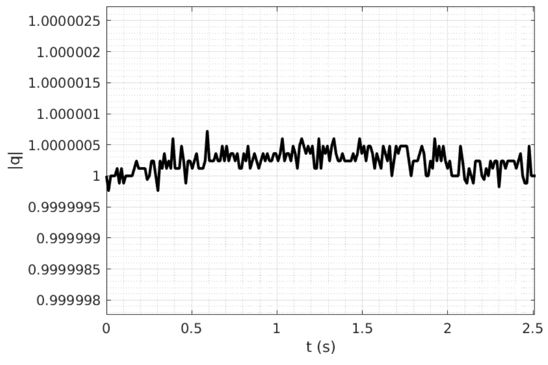
(a) $T = 2.55s$.



(b) $T = 2.525s$.

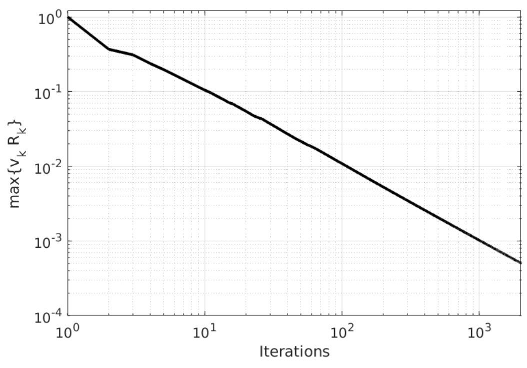


(c) $T = 2.52s$.

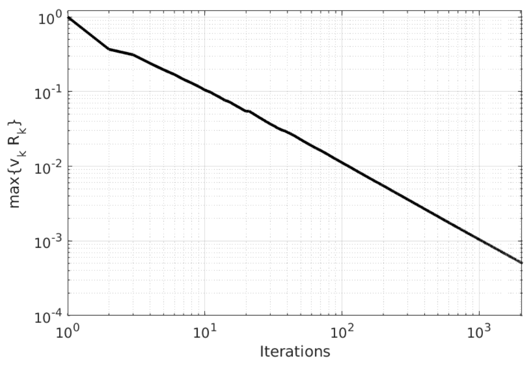


(d) $T = 2.515s$.

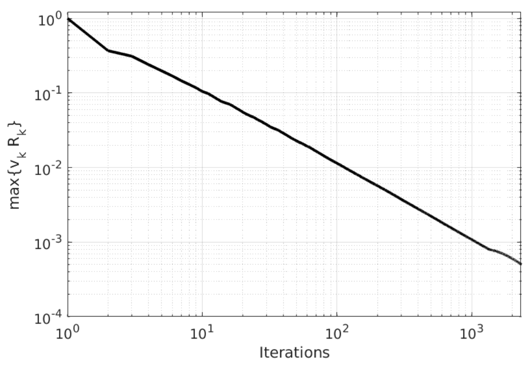
Figure 2.25: Single axis, rest to rest – Norm of quaternion obtained from the non-linear model.



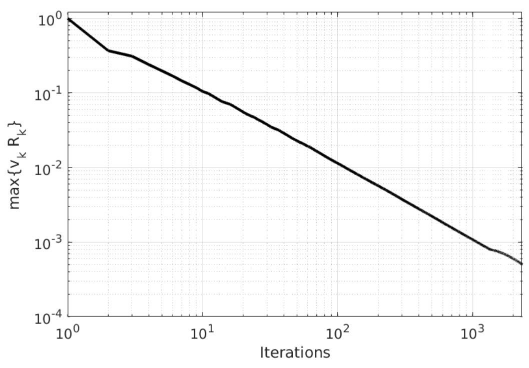
(a) $T = 2.55s$.



(b) $T = 2.525s$.



(c) $T = 2.52s$.



(d) $T = 2.515s$.

Figure 2.26: Single axis, rest to rest – Convergence of the algorithm.

2.2 Comparison between the non-linear and linearised models

Two different approaches have been employed to tackle the proposed optimal control problem. On the one hand, a linearisation was made about a stationary point, which led to a linear evolution law of the type

$$\boldsymbol{x}' = A\boldsymbol{x} + B\boldsymbol{u} \tag{2.2}$$

which, together with a functional of the type

$$\boldsymbol{u} \longmapsto \frac{1}{2}\int_0^T \left(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u}\right) \tag{2.3}$$

resulted in a typical LQR that was solved using the numerical methods.

On the other hand, we proposed a fully non-linear optimal control with an arbitrary integrand and the non-linear evolution law

$$\begin{cases} J\boldsymbol{\omega}' = -\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{u} \\ \boldsymbol{\epsilon}' = -\frac{1}{2}\boldsymbol{\omega} \times \boldsymbol{\epsilon} + \frac{1}{2}\eta\boldsymbol{\omega} \\ \eta' = -\frac{1}{2}\boldsymbol{\omega}^T\boldsymbol{\epsilon} \end{cases} \tag{2.4}$$

which was tackled by means of its variational reformulation.

2 2 1 Advantages of the linear approach

Both perspectives have their own advantages and disadvantages. We start by formulating a few advantages of the LQR as opposed the non-linear approach:

- The major advantage of the linear approach is its computation time: Solving an LQR problem is considerably faster that finding solutions to a general, non-linear optimal control problem. We have been able to obtain computation times of the order of magnitude of milliseconds to solve the LQR problem, in contrast to the tenths of seconds required to approximate the non-linear optimal control problem on the same computer [1]. Computation times for the LQR could be reduced even further if explicit feedback control techniques were employed.
- Solving the LQR does not require the user to keep track of convergence parameters, as it is not an iterative method, unlike the non-linear model.

This makes the LQR a more suitable algorithm for implementation in a system that requires near to real-time response, but that is willing to sacrifice accuracy for the sake of quick response times.

2 2 2 Advantages of the non-linear approach

In the author's opinion, the non-linear model is superior to the LQR for a number of reasons:

- The non-linear model allows a much broader class of integrands, in contrast to the LQR, which is restricted to a quadratic integrand on time and control.
- It is arbitrarily accurate for any choice of initial and final state. Its accuracy can be improved by simply lowering the convergence threshold of the algorithm, which essentially dictates when the algorithm decides that the solution it has found is sufficiently close to be optimal and to comply with all constraints. In contrast, the LQR can be used exclusively for very fine manoeuvres, and its accuracy is dependent on initial and final conditions.
- The non-linear model makes it possible to enforce a wide variety of constraints: from pointwise equality and inequality constraints to natural boundary conditions, through global integral constraints and fixed boundary conditions.
- The non-linear algorithm enables the user to perform complicated manoeuvres that could not be carried out with almost any other conventional method: for example, detumbling and steering the satellite to an arbitrary orientation can be achieved with a single manoeuvre. This is impossible for most other control methods, in which two manoeuvres would have to be employed separately.

[1]For comparison purposes, all Fortran 95 subroutines were compiled with gfortran. The simulations were run on a Linux machine with an Intel Core i7 4510U processor and 16Gb of RAM.

CHAPTER 3

VARIATIONAL PROBLEMS

3.1 Introduction to variational problems

In this chapter, we are concerned with problems of the type

$$\text{Minimize in } \boldsymbol{u}(t): \ I(\boldsymbol{u}) = \int_0^T F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))dt \tag{3.1}$$

for a certain functional $I : \mathcal{C} \longrightarrow \mathbb{R}$ built upon an integrand $F(t, \boldsymbol{u}, \boldsymbol{p}) : [0, T] \times \mathbb{R}^N \times \mathbb{R}^N \longrightarrow \mathbb{R}$. The class of functions $\mathcal{C}$ is of paramount importance in variational problems, although we shall disregard this fact throughout this work. One would typically consider $\mathcal{C}$ to be some type of function space, such as the Sobolev Spaces $W^{1,p}([0, T], \mathbb{R}^N)$, $1 \leq p \leq \infty$.

In addition to being a minimizer of the functional (3.1), $\boldsymbol{u}$ is typically requested to fulfil a series of constraints. These might take several forms, such as *End-point conditions*

$$\boldsymbol{u}(0) = \boldsymbol{u}_0, \ \boldsymbol{u}(T) = \boldsymbol{u}_T \tag{3.2}$$

Natural boundary conditions

$$\frac{\partial F}{\partial \boldsymbol{p}}(0, \boldsymbol{u}(0), \boldsymbol{u}'(0)) = \boldsymbol{0}, \ \frac{\partial F}{\partial \boldsymbol{p}}(T, \boldsymbol{u}(T), \boldsymbol{u}'(T)) = \boldsymbol{0} \tag{3.3}$$

Global constraints

$$\begin{cases} \displaystyle\int_0^T \boldsymbol{G}(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))dt = \boldsymbol{g} \\ \boldsymbol{G}(t, \boldsymbol{u}, \boldsymbol{p}') : [0, T] \times \mathbb{R}^N \times \mathbb{R}^N \longrightarrow \mathbb{R}^m \\ \boldsymbol{g} \in \mathbb{R}^m \end{cases} \tag{3.4}$$

Poinwise constraints

$$\begin{aligned} \boldsymbol{G}(t,\boldsymbol{u}(t),\boldsymbol{u}'(t)) \leq 0 \quad \forall t \in [0,T] \\ \boldsymbol{G}(t,\boldsymbol{u},\boldsymbol{u}') : [0,T] \times \mathbb{R}^N\mathbb{R}^N \longrightarrow \mathbb{R} \end{aligned} \tag{3.5}$$

or any plausible combination of all of the above. Even different components of $\boldsymbol{u}$ might have to comply with different types of constraints.

A function $\boldsymbol{u}$ that complies with all constraints is denominated as a **feasible solution**.

3 2 An existence result

The purpose of this section is to study the necessary conditions under which a certain variational problem is ensured to have at least one optimal solution. If the reader is, to some extent, familiar with optimization problems, they will not be surprised to find that the three essential properties of a variational problem that ensure the existence of optimal solutions are related to the smoothness, coercivity and convexity of the integrand F with respect to some or all of its variables.

We now present, without its proof, one of the classic theorems of the Calculus of Variations, particularized to the situation in which the competing curves are regarded as curves in N-dimensional space, that is, funcions $\boldsymbol{u} : [0,T] \longrightarrow \mathbb{R}^N$.

Theorem 1 *Consider the problem*

$$\textit{Minimize in } \boldsymbol{u}(t) : \ I(\boldsymbol{u}) = \int_0^T F(t,\boldsymbol{u}(t),\boldsymbol{u}'(t))\ dt$$

for an integrand $F(t,\boldsymbol{u},\boldsymbol{p}) : [0,T] \times \mathbb{R}^N \times \mathbb{R}^N \longrightarrow \mathbb{R}$, subject to end-point constaints

$$\boldsymbol{u}(0) = \boldsymbol{u}_0, \ \boldsymbol{u}(T) = \boldsymbol{u}_T.$$

Suppose that the integrand F satisfies conditions (i) to (iii):

(i) ***Smoothness****. F is smooth with respect to all of its variables.*

(ii) ***Coercivity****. There exist $C > 0$, $r > 1$ such that $C(|\boldsymbol{p}|^r - 1) \leq F(t,\boldsymbol{u},\boldsymbol{p})$, for all $(t,\boldsymbol{u},\boldsymbol{p}) \in [0,T] \times \mathbb{R}^N \times \mathbb{R}^N$.*

(iii) ***Convexity****. The function $F(t,\boldsymbol{u},\cdot) : \mathbb{R}^N \longrightarrow \mathbb{R}$ is convex for all $(t,\boldsymbol{u}) \in [0,T] \times \mathbb{R}^N$.*

Then, the variational problem admits an optimal solution $\boldsymbol{u}$.

If condition (iii) is substituted by a more demanding condition (iii')

(iii') ***Joint convexity*** *The function $F(t,\cdot,\cdot) : \mathbb{R}^{2N} \longrightarrow \mathbb{R}$ is convex for all $t \in [0,T]$.*

Then, every optimal solution $\boldsymbol{u}$ of the problem is a global minimum.

In addition, if strict convexity takes place rather than regular convexity in (iii'), then optimal solutions are unique.

What is behind this existence result is a rather complex property of the functional I denominated *weak lower semicontinuity.* In fact, this property is ensured by the convexity and smoothness of the integrand F regarded as a function of $\boldsymbol{p}$. However, as occurs with many aspects of variational problems and optimal control that have appeared in this project, this is a very technical subject that we do not aim to cover.

3.3 Optimality conditions – The Euler-Lagrange equations

Optimality conditions for variational problems take the form of a set of differential equations that are the analogous for infinite-dimensional optimization problems to what the Karush–Kuhn–Tucker equations are for finite-dimensional optimization. This set of differential equations is called the Euler-Lagrange equations. Fulfilment of the Euler-Lagrange equations is a necessary condition for optimality. This means that if a feasible solution $\boldsymbol{u}$ is optimal, then optimality conditions are satisfied. However, there can be feasible solutions for which the Euler-Lagrange equations are fulfilled, yet the solution is not optimal; these are analogous to *saddle points* in finite dimension.

We address optimality conditions under prescribed end-point conditions, natural end-point conditions and integral constraints.

3.3.1 End-point conditions

Here, we present the Euler-Lagrange equations for variational problems under end-point conditions. We avoid some technical aspects that would complicate the discussion far beyond the scope of this work. The previous idea is stated formally in the following theorem.

Theorem 2 ***Optimality conditions. The Euler-Lagrange equations.***

Let $F(t,\boldsymbol{u},\boldsymbol{p}) : [0,T] \times \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ be a smooth integrand, and let $\boldsymbol{u}_0, \boldsymbol{u}_T \in \mathbb{R}^N$. If the feasible solution $\bar{\boldsymbol{u}}$ is optimal for the following variational problem

$$\textit{Minimize in } \boldsymbol{u}(t): \quad I(\boldsymbol{u}) = \int_0^T F(t,\boldsymbol{u}(t),\boldsymbol{u}'(t))\ dt \tag{3.6}$$

subject to $\boldsymbol{u}(0) = \boldsymbol{u}_0$, $\boldsymbol{u}(T) = \boldsymbol{u}_T$.

Then

$$\begin{cases} -\dfrac{\partial}{\partial t}\dfrac{\partial F}{\partial \boldsymbol{p}}(t,\bar{\boldsymbol{u}}(t),\bar{\boldsymbol{u}}'(t)) + \dfrac{\partial F}{\partial \boldsymbol{u}}(t,\bar{\boldsymbol{u}}(t),\bar{\boldsymbol{u}}'(t)) = 0 \quad \forall t \in (0,T) \\ \boldsymbol{u}(0) = \boldsymbol{u}_0 \\ \boldsymbol{u}(T) = \boldsymbol{u}_T \end{cases} \tag{3.7}$$

This is a set of N ordinary differential equations in $u_1, ..., u_n$. The main and strongest assumption is the smoothness of the integrand F.

3 3 2 Natural boundary conditions

The objective of this section is to address the issue of more general constraints. In particular, we would like to extend the results of Theorem 2 in order to incorporate the occurrence of more general types of constraints. For instance, we shall deal with natural boundary conditions (no constraint is enforced at the end points), global constraints in the form of integrals, and some combinations of the above. For example, an end-point condition at $t = 0$ and a natural boundary condition at $t = T$, while a global constraint is enforced on the entire interval $[0, T]$.

A natural boundary condition, also called a transversality condition, is essentially the name we give to the absence of a constraint at a specific end-point; either $t = 0$, $t = T$, or perhaps both. This type of constraint can be combined with an end-point constraint at the other end, or, as we will see, with global integral constraints.

Theorem 3 ***Optimality conditions under natural end-point constraints***

Let $F(t, \boldsymbol{u}, \boldsymbol{p}) : [0, T] \times \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ be a smooth integrand, and let $\boldsymbol{u}_0, \boldsymbol{u}_T \in \mathbb{R}^N$. If the feasible solution $\bar{\boldsymbol{u}}$ is optimal for the following variational problem

$$\textit{Minimize in } \boldsymbol{u}(t): \quad I(\boldsymbol{u}) = \int_0^T F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))\, dt \tag{3.8}$$

subject to $\boldsymbol{u}(0) = \boldsymbol{u}_0$

Then

$$\frac{\partial}{\partial t}\frac{\partial F}{\partial \boldsymbol{p}}(t, \bar{\boldsymbol{u}}(t), \bar{\boldsymbol{u}}'(t)) - \frac{\partial F}{\partial \boldsymbol{u}}(t, \bar{\boldsymbol{u}}(t), \bar{\boldsymbol{u}}'(t)) = 0 \quad \forall t \in (0, T) \tag{3.9}$$

together with

$$\begin{cases} \boldsymbol{u}(0) = \boldsymbol{u}_0 \\ \dfrac{\partial F}{\partial \boldsymbol{p}}(T, \bar{\boldsymbol{u}}(T), \bar{\boldsymbol{u}}'(T)) = 0 \end{cases} \tag{3.10}$$

If, in contrast, we enforce $\boldsymbol{u}(T) = \boldsymbol{u}_T$ and let the end at $t = 0$ with no constraint, then we would have

$$\begin{cases} \boldsymbol{u}(T) = \boldsymbol{u}_T \\ \dfrac{\partial F}{\partial \boldsymbol{p}}(0, \bar{\boldsymbol{u}}(0), \bar{\boldsymbol{u}}'(0)) = 0 \end{cases} \tag{3.11}$$

rather than (3.10).

Finally, if no constraint is enforced at the end-points, and both ends are released, it would be necessary to request

$$\begin{cases} \dfrac{\partial F}{\partial \boldsymbol{p}}(0, \bar{\boldsymbol{u}}(0), \bar{\boldsymbol{u}}'(0)) = 0 \\ \dfrac{\partial F}{\partial \boldsymbol{p}}(T, \bar{\boldsymbol{u}}(T), \bar{\boldsymbol{u}}'(T)) = 0 \end{cases} \tag{3.12}$$

The conditions $\frac{\partial F}{\partial \boldsymbol{p}}(0, \bar{\boldsymbol{u}}(0), \bar{\boldsymbol{u}}'(0)) = 0$, $\frac{\partial F}{\partial \boldsymbol{p}}(T, \bar{\boldsymbol{u}}(T), \bar{\boldsymbol{u}}'(T)) = 0$ are called natural boundary conditions, or transversality conditions at $t = 0$ and $t = T$, respectively.

3.3.3 Integral contraints

In applications, it is common that a certain integral quantity needs to be equal to a predefined value. This amounts to requesting that

$$\int_0^T H_i(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))\ dt = h_i, \quad i = 1, 2, ..., l \tag{3.13}$$

for $h_i \in \mathbb{R}$ fixed, and integrable maps $H_i : [0, T] \times \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$. For example, enforcing these types of constraints is essential to ensure that physical quantities such as length or volume remain fixed.

In order to deal with this type of constraints, we consider a vector of multipliers $\boldsymbol{z} \in \mathbb{R}^l$ and the modified functional

$$\hat{I}(\boldsymbol{u}) = \int_0^T \left(F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t)) + H_i(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))z_i\right)\ dt \tag{3.14}$$

We then would apply either optimality conditions or a suitable approximation algorithm to the modified minimization problem.

3.4 Approximation of smooth variational problems under end-point conditions

As stated previously, variational problems are, more often than not, impossible to solve by hand, in an analytical fashion. Rather, it is far more common to rely on numerical methods for the approximation of such solutions. In this section, different algorithms for the numerical approximation of variational problems with different types of constrains are derived. These algorithms are based on those presented in [17, 18, 19].

We shall attempt to approximate solutions of the typical variational problem

$$\text{Minimize in } \boldsymbol{u}(t) : \ I(\boldsymbol{u}) = \int_0^T F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))\ dt \tag{3.15}$$

subject to end-point conditions:

$$\begin{cases} \boldsymbol{u}(0) = \boldsymbol{u}_0 \\ \boldsymbol{u}(T) = \boldsymbol{u}_T \end{cases} \tag{3.16}$$

We rely on a classic steepest descent method: In each step, the algorithm will find a step direction and a step size in such a way that, if we have a feasible solution $\boldsymbol{u}^k$, we can move proportionally to the step size in the step direction to find a new feasible solution $\boldsymbol{u}^{k+1}$ such that $I(\boldsymbol{u}^{k+1}) < I(\boldsymbol{u}^k)$. The algorithm will then proceed iteratively until convergence.

3.4.1 The descent direction

Let us suppose that, at a certain step of the algorithm, we have found a feasible solution $\boldsymbol{u}$ (complying with $\boldsymbol{u}(0) = \boldsymbol{u}_0$, $\boldsymbol{u}(T) = \boldsymbol{u}_T$). We now consider a perturbation $\boldsymbol{U}$ that has the same nature as $\boldsymbol{u}$ ($\boldsymbol{U}$ is a function $\boldsymbol{U} : [0,T] \to \mathbb{R}^N$), but that vanishes at the boundary:

$$\boldsymbol{U}(0) = \boldsymbol{U}(T) = 0 \tag{3.17}$$

Note that every perturbed curve $\boldsymbol{u} + \tau\boldsymbol{U}$, for $\tau \in \mathbb{R}$, complies with the end-point conditions, just as $\boldsymbol{u}$ does.

Let us now consider the function $\Phi : \mathbb{R} \to \mathbb{R}$, defined by

$$\phi(\epsilon) = I(\boldsymbol{u} + \epsilon\boldsymbol{U}) = \int_0^T F(t, \boldsymbol{u} + \epsilon\boldsymbol{U}, \boldsymbol{u}' + \epsilon\boldsymbol{U}') \tag{3.18}$$

for a fixed path $\boldsymbol{u}$ and perturbation $\boldsymbol{U}$ that vanishes at its boundary.

We now compute the derivative of Φ and evaluate at 0.

$$\begin{aligned}\Phi(\epsilon) &= \frac{d}{d\epsilon}\int_0^T F(t, \boldsymbol{u} + \epsilon\boldsymbol{U}, \boldsymbol{u}' + \epsilon\boldsymbol{U}') = \int_0^T \frac{d}{d\epsilon} F(t, \boldsymbol{u} + \epsilon\boldsymbol{U}, \boldsymbol{u}' + \epsilon\boldsymbol{U}') = \\ &= \int_0^T \left[\frac{\partial F}{\partial \boldsymbol{u}}(t, \boldsymbol{u} + \epsilon\boldsymbol{U}, \boldsymbol{u}' + \epsilon\boldsymbol{U}')\boldsymbol{U} + \frac{\partial F}{\partial \boldsymbol{p}}(t, \boldsymbol{u} + \epsilon\boldsymbol{U}, \boldsymbol{u}' + \epsilon\boldsymbol{U}')\boldsymbol{U}'\right]\end{aligned} \tag{3.19}$$

$$\Phi(0) = \int_0^T \left[\frac{\partial F}{\partial \boldsymbol{u}}(t, \boldsymbol{u}, \boldsymbol{u}')\boldsymbol{U} + \frac{\partial F}{\partial \boldsymbol{p}}(t, \boldsymbol{u}, \boldsymbol{u}')\boldsymbol{U}'\right] \tag{3.20}$$

where dependence on t has been dropped for simplicity.

For a fixed $\boldsymbol{u}$, we now focus on solving the following variational problem in $\boldsymbol{U}$

$$\text{Minimize in } \boldsymbol{U} : \int_0^T \left(\frac{1}{2}|\boldsymbol{U}'|^2 + \frac{\partial F}{\partial \boldsymbol{u}}(t, \boldsymbol{u}, \boldsymbol{u}')\boldsymbol{U} + \frac{\partial F}{\partial \boldsymbol{p}}(t, \boldsymbol{u}, \boldsymbol{u}')\boldsymbol{U}'\right) \tag{3.21}$$

subject to

$$\boldsymbol{U}(0) = \boldsymbol{U}(T) = 0 \tag{3.22}$$

This variational problem is crafted in such a way that its solution (which is, in fact, unique) can be expressed very nicely in closed form. This is Lemma 6.1 in [18], book to which we refer the reader for a more complete analysis on the subject.

The solution to this variational problem in $\boldsymbol{U}$ is

$$\begin{aligned}\boldsymbol{U}(t) = \frac{t-T}{T}\int_0^T \left(\bar{F}_p(s) - (T-s)\bar{F}_u(s)\right)\, ds &+ \\ + (t-T)\int_0^t \bar{F}_u(s)\, ds &+ \\ + \int_t^T \left(\bar{F}_p(s) - (T-s)\bar{F}_u(s)\right)\, ds&\end{aligned} \tag{3.23}$$

where

$$\begin{aligned}\bar{F}_u(t) &= \frac{\partial F}{\partial \boldsymbol{u}}(t, \boldsymbol{u}(t), \boldsymbol{U}'(t)) \\ \bar{F}_p(t) &= \frac{\partial F}{\partial \boldsymbol{p}}(t, \boldsymbol{u}(t), \boldsymbol{U}'(t))\end{aligned} \tag{3.24}$$

Its derivative can be easily calculated:

$$\begin{aligned}\boldsymbol{U}'(t) = \frac{1}{T}\int_0^T \left(\bar{F}_p(s) - (T-s)\bar{F}_u(s)\right)\, ds\, + \\ - T\int_0^t \bar{F}_u(s)\, ds\, - \bar{F}_p(t)\end{aligned} \tag{3.25}$$

Note, however, that the integrals above will probably need to be evaluated through the use of quadrature methods.

We shall check tat the quantity

$$||\boldsymbol{U}||^2 = \int_0^T |\boldsymbol{U}'(t)|^2\, dt \tag{3.26}$$

decreases with every iteration. That is

$$||\boldsymbol{U}^{k+1}||^2 < ||\boldsymbol{U}^k||^2,\ k = 1, 2, ... \tag{3.27}$$

until it eventually vanishes. In practice, this translates into it becoming sufficiently small – smaller than a predefined $\epsilon > 0$. The moment that this occurs, we shall consider that the corresponding $\boldsymbol{u}$ is a solution of the variational problem under end-point constraints.

3 4 2 The step size

When a decision has been made as regards the direction $\boldsymbol{U}$ in which the algorithm will move in order to find a better approximation $\boldsymbol{u} + \tau\boldsymbol{U}$ of the solution such that $I(\boldsymbol{u} + \tau\boldsymbol{U}) < I(\boldsymbol{U})$, it is necessary to decide on the step size τ.

Ideally, we would like to choose τ such that the derivative of $\Phi(\epsilon)$ vanishes at $\epsilon = \tau$. However, the equation

$$\Phi'(\tau) = \int_0^T F(s, \boldsymbol{u}(s) + \tau\boldsymbol{U}(s), \boldsymbol{u}'(s) + \tau\boldsymbol{U}'(s)\, ds = 0 \tag{3.28}$$

might be impossible or completely impractical to solve for τ. It is more reasonable to approximate τ in such a way that we simply ensure that $\boldsymbol{u} + \tau\boldsymbol{U}$ is a better solution than $\boldsymbol{u}$. The resulting τ might not be optimal, in the sense that there could be a different choice that makes $I(\boldsymbol{u} + \tau\boldsymbol{U})$ even smaller. However, we are satisfied to find a good τ in a process that is considered sufficiently computationally effective.

There are different methods for the selection of the step size. Here, we choose that proposed in [18], which is based on approximating Φ by a parabola and finding its vertex. The method works as follows:

1. Firstly, we start with two predefined constants $\tau_0, c > 0$, which are parameters of the algorithm.
2. Then, the quantity $\Phi(\tau_0) - \Phi(0) - c\tau_0\Phi'(0)$ is computed.
3. If this quantity is less than or equal to 0, we then update $\boldsymbol{u}$ to $\boldsymbol{u} + \tau_0\boldsymbol{U}$.
4. Otherwise (the quantity is strictly positive), we then compute

   $$\tau = \frac{-\tau_0^2\phi'(0)}{2(\Phi(\tau_0) - \Phi(0) - \tau_0\Phi'(0))}$$

   and update $\boldsymbol{u}$ to $\boldsymbol{u} + \tau\boldsymbol{U}$.

3 5 Approximation of smooth variational problems under more restrictive constraints

In this section, various algorithms for the numerical approximation of variational problems with different types of constrains are derived. These algorithms are based on those presented in [17, 18, 19].

3 5 1 Pointwise constraints in the form of inequalities

This section addresses the approximation of variational problems under pointwise constraints in the form of inequalities. In order to place things in context, we shall attempt to approximate solutions to the variational problem

$$\text{Minimize in } \boldsymbol{u}(t): \; I(\boldsymbol{u}) = \int_0^T F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))\, dt \tag{3.29}$$

subject to:

- End-point conditions $\boldsymbol{u}(0) = \boldsymbol{u}_0$, $\boldsymbol{u}(T) = \boldsymbol{u}_T$ and
- Pointwise constraints $G(\boldsymbol{u}(t), \boldsymbol{u}'(t)) \leq 0 \; \forall t \in [0, T]$.

The integrand $F(t, \boldsymbol{u}, \boldsymbol{p}) : [0, T] \times \mathbb{R}^N \times \mathbb{R}^N \longrightarrow \mathbb{R}$ and the map $G : \mathbb{R}^N \times \mathbb{R}^N \longrightarrow \mathbb{R}^m$ are both considered to be smooth with respect to all of their variables. These assumptions, along with the convexity of the integrand $F(t, \boldsymbol{u}, \boldsymbol{p})$ with respect to $\boldsymbol{p}$, are key aspects as regards the following derivations.

Our strategy will be to find solutions to pointwise constrained variational problems by iteratively approximating the solutions of unconstrained (except for end-point conditions) variational problems, which are carefully crafted using the ingredients of the original, pointwise-constrained problem: The functional is augmented by adding a term to the integrand which takes into account how far the solution is from complying with all the constraints: A non-negative term which grows exponentially as $\boldsymbol{u}$ moves away from complying with all constraints and decreases as all constraints are close to be met.

At each iteration, this term is updated until convergence takes place, that is, until this new functional is minimized. This implies that all the constraints are satisfied.

As will be shown, this augmented integrand relies on the idea of multipliers, which are the main tool employed to impose constraints.

We introduce the auxiliary map $\boldsymbol{v} : [0,T] \longrightarrow \mathbb{R}^m$. Each component of $\boldsymbol{v}$

$$v_i(t), \ i = 1, 2, ..., m$$

is called a **multiplier**, which, as we shall see, is in charge of enforcing the constraint

$$G_i(\boldsymbol{u}(t), \boldsymbol{u}'(t)) \leq 0 \ \forall t \in [0,T], \ i = 1, 2, ..., m$$

Once multipliers are introduced, we construct the functional $J(\boldsymbol{u}, \boldsymbol{v})$, which depends on the function $\boldsymbol{u}$ (that which we intend to minimize), and the vector of multipliers $\boldsymbol{v}$:

$$J(\boldsymbol{u}, \boldsymbol{v}) := \int_0^T \left(F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t)) + \sum_{k=1}^{m} \left(e^{v_k(t) G_k(\boldsymbol{u}(t), \boldsymbol{u}'(t))} \right) - m \right) dt \tag{3.30}$$

The algorithm will then proceed in the following manner: Upon initialization of $\boldsymbol{u}$ to an arbitrary, but feasible solution (an affine function complying with $\boldsymbol{u}(0) = \boldsymbol{u}_0$, $\boldsymbol{u}(T) = \boldsymbol{u}_T$, to give an example), it will also initialize the multipliers, in this case to a positive constant, (for example $\boldsymbol{v} \equiv \mathbf{1}$). It will then proceed with an iterative scheme consisting of two steps:

- First, it will (approximately) solve the variational problem costructed upon the functional $J(\boldsymbol{u}, \boldsymbol{v})$, with the current set of multipliers, and the same constraint as our original problem.
- Second, if the solution $\boldsymbol{u}$ of this problem is sufficiently close to the solution of our original problem (we shall see how this condition materializes later), the algorithm will stop; otherwise, it will update the multipliers in a suitable manner, and then carry out the minimization of $J(\boldsymbol{u}, \boldsymbol{v})$ again, with this new set of multipliers.

In order to put this into more precise words, we shall define the following notation: We use super-indices, as in $\boldsymbol{u}^j(t)$ or $\boldsymbol{v}^j(t)$ to refer to the j-th iteration of the algorithm for the corresponding quantity. Sub-indices are, however, used in a different way for $\boldsymbol{u}$ and $\boldsymbol{v}$. For instance, $v_k(t), u_k \ k = 1, 2, ...$ represent the k-th component of the vector $\boldsymbol{v}$ or $\boldsymbol{u}$, while $\boldsymbol{u}_0, \boldsymbol{u}_T \in \mathbb{R}^N$ are the N-dimensional real vectors used in end-point conditions.

Note that if all the constraints are met, each of the exponentials of the term $\sum_{k=1}^{m} e^{v_k(t) G_k(\boldsymbol{u}(t), \boldsymbol{u}'(t))}$ in 3.30 is equal to 1, and therefore the sum of all of them equals m (the total number of inequality constraints); the last two summands of the integrand will then cancel each other out.

Having introduced this notation, we can now present the algorithm in a pseudocode manner.

1. **Initialization.**

   (a) Set $\boldsymbol{u}^0(t)$ to an arbitrary but feasible solution. For example:

$$\boldsymbol{u}^0(t) = \frac{T-t}{T} \boldsymbol{u}_0 + \frac{t}{T} \boldsymbol{u}_T$$

(b) Set the vector of multipliers $\boldsymbol{v}(t)$ arbitrarily, with positive components. We will normally use

$$v_k^0(t) \equiv 1, \;\; k = 1, 2, ..., m$$

2. **Approximation.** Minimize the functional

$$J(\boldsymbol{u}^j, \boldsymbol{v}^j) := \int_0^T \left(F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t)) + \sum_{k=1}^m \left(e^{v_k^j(t) G_k(\boldsymbol{u}(t), \boldsymbol{u}'(t))} \right) - m \right) \, dt$$

approximately, using the techniques presented in the previous section. Then set $\boldsymbol{u}^{j+1}$ as the solution to this problem.

3. **Updating of multipliers.** Evaluate

$$M = \max \left\{ v_k^j(t) G_k(\boldsymbol{u}^{j+1}(t), (\boldsymbol{u}^{j+1})'(t)) : \;\; t \in [0, T], k = 1, 2, ..., m \right\}$$

(a) If this quantity M (which plays the role of an indicator of how close the algorithm is to the desired solution that meets all constraints) is smaller than a predefined small constant δ, then $\boldsymbol{u}^{j+1}$ is sufficiently close to the solution of the variational problem with pointwise constraints.

(b) Otherwise, if $M > \delta$, update the multipliers according to the following rule.

$$v_k^{j+1}(t) = e^{v_k^j(t) G_k(\boldsymbol{u}^{j+1}(t), (\boldsymbol{u}^{j+1})'(t))} v_k^j(t) \;\; \forall t \in [0, T], \;\; k = 1, 2, ..., m$$

Then, set

$$\boldsymbol{v}^j \leftarrow \boldsymbol{v}^{j+1}, \;\; \boldsymbol{u}^j \leftarrow \boldsymbol{u}^{j+1}$$

and proceed from Step 2.

A thorough explanation of why this algorithm makes sense, and why we can be sure that, if convergence takes place, the solution is in fact sufficiently close to the solution of the constrained variational problem, can be found in [17, 19].

3.5.2 Pointwise constraints in the form of equalities

We have now learned a method to enforce inequality constraints of the form $G(\boldsymbol{u}(t), \boldsymbol{u}'(t)) \leq 0 \; \forall t \in [0, T]$. However, in certain situations one might also (or alternatively) wish to enforce p equality constraints in the form $g(\boldsymbol{u}(t), \boldsymbol{u}'(t)) = 0 \; \forall t \in [0, T]$, for a smooth map $g : \mathbb{R}^N \times \mathbb{R}^N \longrightarrow \mathbb{R}^p$. If this is the case, the previous ideas can be modified in such a way that the algorithm will find solutions to the variational problem subject to these equality constraints. This can be done in two different ways:

1. Consider the quantity

$$\frac{1}{2} \left| g(\boldsymbol{u}, \boldsymbol{p}) \right|^2$$

and solve the variational problem subject to the p additional inequality constraints

$$\frac{1}{2} \left| g(\boldsymbol{u}(t), \boldsymbol{u}'(t)) \right|^2 \leq 0 \; \forall t \in [0, T]$$

2. Alternatively, solve the variational problem which complies with the additional $2p$ inequality constraints

$$g(\boldsymbol{u}(t), \boldsymbol{u}'(t)) \leq 0, \;\; -g(\boldsymbol{u}(t), \boldsymbol{u}'(t)) \leq 0 \;\;\; \forall t \in [0, T]$$

3.5.3 Integral constraints

As stated in section 3.3.3, imposing global constraints in the form of integrals is a crucial tool if it is necessary to model certain problems in physics and engineering. In this situation, we must deal with the following problem:

$$\text{Minimize in } \boldsymbol{u}(t): \; I(\boldsymbol{u}) = \int_0^T F(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))\, dt \tag{3.31}$$

subject to

$$\begin{cases} \displaystyle\int_0^T H(t, \boldsymbol{u}(t), \boldsymbol{u}'(t))\, dt \leq \mathbf{0} \\ \boldsymbol{u}(0) = \boldsymbol{u}_0 \\ \boldsymbol{u}(T) = \boldsymbol{u}_T \end{cases} \tag{3.32}$$

for a map $H(t, \boldsymbol{u}, \boldsymbol{p}) \to \mathbb{R}^l$.

The method used to address this type of constraints consists of translating integral constraints into pointwise constraints by expanding the vector $\boldsymbol{u}$ with the vector $\hat{\boldsymbol{u}} : [0, T] \to \mathbb{R}^l$ and imposing the following constraints

$$\begin{cases} H(t, \boldsymbol{u}(t), \boldsymbol{u}'(t)) - \hat{\boldsymbol{u}}(t) \leq \mathbf{0} \;\; \forall t \in [0, T] \\ \hat{\boldsymbol{u}}(0) = \hat{\boldsymbol{u}}(T) = \mathbf{0} \end{cases} \tag{3.33}$$

This is the method shown in Section 6.3 of [18].

It is common for integral constraints to appear in the form of equalities rather than inequalities. If this is the case, we can easily combine this with the method explained in Section 3.5.2 in order to impose global, integral constraints in the form of equalities.

CHAPTER 4

OPTIMAL CONTROL

4.1 Introduction to optimal control

We are interested in problems of the form

$$\text{Minimize in } \boldsymbol{u}(t) : \quad \int_0^T F(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \; dt \; + \phi(\boldsymbol{x}(T)) \tag{4.1}$$

subject to

$$\begin{cases} \boldsymbol{x}'(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \; \forall t \in [0,T] \\ \boldsymbol{x}(t) \in \Omega \subset \mathbb{R}^N \quad \forall t \in [0,T] \\ \boldsymbol{u}(t) \in \boldsymbol{K} \subset \mathbb{R}^m \quad \forall t \in [0,T] \end{cases} \tag{4.2}$$

and end-point conditions

$$\begin{cases} \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{x}(T) = \boldsymbol{x}_T \end{cases} \tag{4.3}$$

This is a general class of optimal control problems that are discussed in more detail than in the present work in books such as [36, 37, 38, 39, 40]. The main components of these problems are as follows:

- **The time horizon** $[0, T] \subset [0, +\infty)$ is the time interval with which we are concerned. Without loss of generality, the dynamical system of interest will evolve from $t = 0$ to $t = T$.
- **The state variables vector** $\boldsymbol{x}(t) : [0, T] \longrightarrow \mathbb{R}^N, N \geq 1$. The N variables $x_1(t), x_2(t), \ldots, x_N(t)$ describe the evolution of the dynamical system. In typical examples from science and engineering, these would represent variables of interest such as positions, angles, velocities or accelerations of components of a mechanical system, current passing through a certain electric component, voltage drop between two points, etc.

- **The control variables vector $\boldsymbol{u}(t) : [0,T] \longrightarrow \mathbb{R}^m, 1 \leq m \leq N$.** The m variables $u_1(t), u_2(t), \ldots, u_m(t)$ are at our disposal to be chosen in order to affect the behavior of the dynamical system to fulfill the end-point conditions (4.3) while minimizing the cost functional (4.1).

- **The state law or state equations $\boldsymbol{x}'(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$.** It is an ordinary differential equation if $N = 1$, or a set of ordinary differential equations otherwise. It describes the behaviour of the system of concern. In problems derived from physics and engineering, it may typically correspond to some kind of physical law, such as equilibrium of forces, conservation of energy, charge, or angular momentum.

- **Restrictions on state and control**. The values that $\boldsymbol{x}(t)$ and $\boldsymbol{u}(t)$ can take in the interval $[0,T]$ are restricted to the sets Ω and $\boldsymbol{K}$. Further restrictions are imposed on the extremes of the interval through the use of the initial and final states $\boldsymbol{x}_0$ and $\boldsymbol{x}_T$, respectively, by the boundary conditions (4.3).

- **Cost functional**

$$I(\boldsymbol{u}) = \int_0^T F(t, \boldsymbol{x}(t), \boldsymbol{u}(t))\; dt + \phi(\boldsymbol{x}(T))$$

  This is the quantity that we aim to minimize. It depends on both the choice of the control $\boldsymbol{u}$ and the evolution of the state $\boldsymbol{x}$ which is determined by means of the state law and initial conditions. Commonly, though not always, the density F has some type of quadratic dependence on state and control, such as $F(t, \boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u}$. The term $\phi(\boldsymbol{x}(T))$ penalizes the final state; it is optional and is, in fact, omitted in a number of situations. Moreover, the integrand F can be easily modified in such a way that it takes into account the term $\phi(\boldsymbol{x}(T))$. It is for this reason that we shall consider $\phi \equiv 0$.

4 1 1 Controllability

In control problems (optimal or otherwise) there is typically the important issue of controllability: when final state constraints are to be enforced with the system starting from a predetermined position. The space of feasible controls is reduced to a subset of the initial function space that we were initially searching, with the objective of discovering the optimal control function $\boldsymbol{u}$. There can in fact, be choices of initial and final states $\boldsymbol{x}_0$, $\boldsymbol{x}_T$ for which no function $\boldsymbol{u}$ is able to ensure that the system meets the requirements (4.3). However, controllability is an important and difficult field in itself; a question far from trivial and that we shall not attempt to answer in any of the cases discussed in this work. For instance, we shall rely on the fact that if an optimal solution is found for a particular situation, then that configuration is controllable. However, the converse is not true [41] – a variety of reasons can cause the algorithm not to find a feasible solution. This may be caused by numerical issues, problems with the implementation of the algorithm, or because that the configuration is, in fact, not controllable.

4.2 An existence result

In this section, we shall provide a result regarding the existence of solutions to optimal control problems under certain assumptions on different components of the problem. In particular, we provide a sufficient condition for the existence of solutions to optimal control problems. In short, we can state:

Theorem 4 *If the following assumptions (1 to 5) hold simultaneously, the optimal control problem (4.1)-(4.3) admits optimal solutions.*

1. *The set Ω is closed.*
2. *The set $\boldsymbol{K}$ is convex.*
3. *The density $F(\boldsymbol{x}, \boldsymbol{u})$ is convex in $\boldsymbol{u}$.*
4. *The density $F(\boldsymbol{x}, \boldsymbol{u})$ is coercive in $\boldsymbol{u}$.*
5. *The state law is linear in $\boldsymbol{u}$. That is to say, $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \boldsymbol{f}_0(\boldsymbol{x}) + \boldsymbol{f}_1(\boldsymbol{x})\boldsymbol{u}$ with $\boldsymbol{f}_0$ and $\boldsymbol{f}_1$ continuous.*

4.3 A variational method for optimal control

Our main strategy employed to solve optimal control problems under pointwise constraints will be to derive an equivalent variational problem in the sense that solutions to the optimal control problem can be recovered from solutions to the variational problem.

Let us suppose that the following optimal control problem is given

$$\text{Minimize in } \boldsymbol{u}(t) : \quad \int_0^T F(t, \boldsymbol{x}(t), \boldsymbol{u}(t))\, dt \tag{4.4}$$

subject to

$$\begin{cases} \boldsymbol{x}'(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \ \forall t \in [0, T] \\ \boldsymbol{x}(t) \in \Omega \subset \mathbb{R}^N \quad \forall t \in [0, T] \\ \boldsymbol{u}(t) \in \boldsymbol{K} \subset \mathbb{R}^m \quad \forall t \in [0, T] \\ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{x}(T) = \boldsymbol{x}_T \end{cases} \tag{4.5}$$

and let us suppose that the state law $\boldsymbol{x}' \mapsto \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{u})$ is such that $\boldsymbol{u}$ can be worked out in terms of $\boldsymbol{x}$ and $\boldsymbol{x}'$, that is, there exists a suitable function $\boldsymbol{\psi} : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ such that

$$\boldsymbol{u}(t) = \boldsymbol{\psi}\left(\boldsymbol{x}(t), \boldsymbol{x}'(t)\right) \quad \forall t \in [0, T] \tag{4.6}$$

If this hypothesis on $\boldsymbol{f}$ holds (which is the case for most practical situations), then the optimal control problem is equivalent to the following variational problem:

$$\text{Minimize in } \boldsymbol{x}(t): \quad \int_0^T F\left(t, \boldsymbol{x}(t), \boldsymbol{\psi}(\boldsymbol{x}(t), \boldsymbol{x}'(t))\right)\, dt \tag{4.7}$$

subject to

$$\begin{cases} \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{x}(T) = \boldsymbol{x}_T \end{cases} \tag{4.8}$$

Once an optimal solution $\boldsymbol{x}$ of the variational problem has been found (or approximated using the pertinent numerical method), the control can be easily recovered using

$$\boldsymbol{u}(t) = \boldsymbol{\psi}\left(\boldsymbol{x}(t), \boldsymbol{x}'(t)\right)$$

Note that, if there is the need to enforce pointwise constraints of the form $G(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{u}) \leq 0$ in $[0, T]$, it is sufficient to translate this condition to the pointwise constraint $G\left(\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{\psi}(\boldsymbol{x}, \boldsymbol{x}')\right) \leq 0$ in $[0, T]$ and enforce it on its variational counterpart using, for example, the techniques explained in section 3.5.

4.4 The linear quadratic regulator

The Linear Quadratic Regulator, or *LQR* in what follows, is a particular optimal control problem, for which Pontryaguin's maximum principle takes the form of a system of linear ordinary differential equations, which is relatively easy to solve either analytically or by means of numerical methods.

The problem is once again stated by attempting to find the control $\boldsymbol{u} : [0, T] \longrightarrow \mathbb{R}^m$, which minimizes the cost functional

$$\text{Minimize in } \boldsymbol{u}(t) \int_0^T F(t, \boldsymbol{x}(t), \boldsymbol{u}(t))\, dt \; + \; \phi(\boldsymbol{x}(T)) \tag{4.9}$$

subject to

$$\begin{cases} \boldsymbol{x}'(t) = \boldsymbol{f}(t, \boldsymbol{x}(t), \boldsymbol{u}(t)) \ \forall t \in [0, T] \\ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{x}(T) = \boldsymbol{x}_T \\ \boldsymbol{x}(t) \in \Omega \subset \mathbb{R}^N \ \forall t \in [0, T] \\ \boldsymbol{u}(t) \in \boldsymbol{K} \subset \mathbb{R}^m \ \forall t \in [0, T] \end{cases} \tag{4.10}$$

Particularly, in LQR problems, the integrand F in (4.9) involves quadratic forms on state and control.

$$I(\boldsymbol{u}) = \frac{1}{2} \int_0^T \left(\boldsymbol{x}(t)^t Q \boldsymbol{x}(t) + \boldsymbol{u}(t)^t R \boldsymbol{u}(t)\right)\, dt \; + \; \phi(\boldsymbol{x}(T)) \tag{4.11}$$

that is

$$F(t, \boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2} \left(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u}\right) \tag{4.12}$$

where Q and R are, respectively, positive semidefinite and positive definite matrices.

We shall introduce Pontryaguin's maximum principle for a general class of optimal control problems and then particularize the results to the very specific case of a Linear-Quadratic Regulator. First, let us expand the cost functional by introducing the costate

$$\boldsymbol{p}(t) : [0,T] \longrightarrow \mathbb{R}^N$$

Let us then add the term $\boldsymbol{p}(t)(\boldsymbol{f}(t) - \boldsymbol{x}'(t))$ to the integrand in (4.11) in order to obtain the so-called extended cost functional

$$\bar{I}(\boldsymbol{u}) = \int_0^T \big(F + \boldsymbol{p}^t \cdot (\boldsymbol{f} - \boldsymbol{x}')\big)\, dt + \phi(\boldsymbol{x}(T)) \qquad (4.13)$$

where dependencies on time and other variables have been dropped for clarity, as will be done without warning throughout what remains of this section.

Note that, according to the state law, the new term $\boldsymbol{p}(t)(\boldsymbol{f}(t) - \boldsymbol{x}'(t))$ should vanish $\forall t \in [0,T]$, and for optimal trajectories, the costate $\boldsymbol{p}$ can, therefore, be arbitrarily chosen without altering the value of the integral in (4.13).

Let us now derive a variation in $\bar{I}(\boldsymbol{u})$

$$\delta\bar{I}(\boldsymbol{u}) = \frac{\partial\phi}{\partial\boldsymbol{x}}\delta\boldsymbol{x}\Big|_{t=T} + \int_0^T \left(\frac{\partial F}{\partial\boldsymbol{x}}\delta\boldsymbol{x} + \frac{\partial F}{\partial\boldsymbol{u}}\delta\boldsymbol{u} + \boldsymbol{p}^t\left(\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}\delta\boldsymbol{x} + \frac{\partial\boldsymbol{f}}{\partial\boldsymbol{u}}\delta\boldsymbol{u}\right)\right) dt - \int_0^T \boldsymbol{p}^t\delta\dot{\boldsymbol{x}}dt \qquad (4.14)$$

By using integration by parts in the last integral on the right-hand side:

$$\delta\bar{I}(\boldsymbol{u}) = \frac{\partial\phi}{\partial\boldsymbol{x}}\delta\boldsymbol{x}\Big|_{t=T} + \int_0^T \left(\frac{\partial F}{\partial\boldsymbol{x}}\delta\boldsymbol{x} + \frac{\partial F}{\partial\boldsymbol{u}}\delta\boldsymbol{u} + \boldsymbol{p}^t\left(\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}}\delta\boldsymbol{x} + \frac{\partial\boldsymbol{f}}{\partial\boldsymbol{u}}\delta\boldsymbol{u}\right) + (\boldsymbol{p}')^t\delta\boldsymbol{x}\right) dt + \\ - \boldsymbol{p}^t(T)\delta\boldsymbol{x}(T) + \boldsymbol{p}^t(0)\delta\boldsymbol{x}(0) \qquad (4.15)$$

Upon rearranging and grouping together the terms involving δx and δu, we obtain

$$\delta\bar{I}(\boldsymbol{u}) = \frac{\partial\phi}{\partial\boldsymbol{x}}\delta\boldsymbol{x}\Big|_{t=T} + \int_0^T \left(\frac{\partial F}{\partial\boldsymbol{x}} + \boldsymbol{p}^t\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}} + (\boldsymbol{p}')^t\right)\boldsymbol{x}\ dt + \int_0^T \left(\frac{\partial F}{\partial\boldsymbol{u}} + \boldsymbol{p}^t\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{u}}\right)\delta\boldsymbol{u}\ dt + \\ - \boldsymbol{p}^t(T)\delta\boldsymbol{x}(T) + \boldsymbol{p}^t(0)\delta\boldsymbol{x}(0) \qquad (4.16)$$

The state at time $t = 0$ is fixed and is given by $\boldsymbol{x}(0) = \boldsymbol{x}_0$, and the quantity $\delta\boldsymbol{x}(0)$, therefore, vanishes.

$$\delta\bar{I}(\boldsymbol{u}) = \frac{\partial\phi}{\partial\boldsymbol{x}}\delta\boldsymbol{x}\Big|_{t=T} + \int_0^T \left(\frac{\partial F}{\partial\boldsymbol{x}} + \boldsymbol{p}^t\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}} + (\boldsymbol{p}')^t\right)\boldsymbol{x}\ dt + \int_0^T \left(\frac{\partial F}{\partial\boldsymbol{u}} + \boldsymbol{p}^t\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{u}}\right)\delta\boldsymbol{u}\ dt + \\ - \boldsymbol{p}^t(T)\delta\boldsymbol{x}(T) \qquad (4.17)$$

For any optimal trajectory, this variation should be zero, as choosing $\bar{F}(t, \boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{u} + \delta\boldsymbol{u}) = \bar{F} + \delta\bar{F}$, would lead to a change in trajectory that would increase the value of the functional. Moreover, each of the following terms should vanish separately:

$$\begin{cases} \dfrac{\partial\phi}{\partial\boldsymbol{x}}(\boldsymbol{x}(0)) - \boldsymbol{p}^t(T) = 0 \\ \dfrac{\partial F}{\partial\boldsymbol{x}} + \boldsymbol{p}^t\dfrac{\partial\boldsymbol{f}}{\partial\boldsymbol{x}} + (\boldsymbol{p}')^t = 0 \\ \dfrac{\partial F}{\partial\boldsymbol{u}} + \boldsymbol{p}^t\dfrac{\partial\boldsymbol{f}}{\partial\boldsymbol{u}} = 0 \end{cases} \qquad (4.18)$$

In the particular case of an LQR problem, the cost functional, which we now attempt to minimize, takes the following form

$$I(\boldsymbol{u}) = \frac{1}{2}\int_0^T \left(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u}\right) dt \tag{4.19}$$

that is to say

$$F(t, \boldsymbol{x}, \boldsymbol{u}) = \frac{1}{2}\left(\boldsymbol{x}^t Q \boldsymbol{x} + \boldsymbol{u}^t R \boldsymbol{u}\right) \tag{4.20}$$

Furthermore, the state equation is linear in both state and control

$$\boldsymbol{x}' = A\boldsymbol{x} + B\boldsymbol{u} \tag{4.21}$$

Optimality conditions (4.18) then become

$$\begin{cases} \boldsymbol{p}(T) = 0 \\ Q\boldsymbol{x} + A^t\boldsymbol{q} + \boldsymbol{p}' = 0 \text{ in } [0,T] \\ R\boldsymbol{u} + B^t\boldsymbol{p} = 0 \text{ in } [0,T] \end{cases} \tag{4.22}$$

In order to discover the control $\boldsymbol{u}$ that minimizes (4.19) One can build the following system of ordinary differential equations

$$\begin{cases} \boldsymbol{x}' = A\boldsymbol{x} + B\boldsymbol{u} \\ \boldsymbol{p}' = -Q\boldsymbol{x} - A^t\boldsymbol{p} \\ \ \,0 = R\boldsymbol{u} + B^t\boldsymbol{p} \end{cases} \tag{4.23}$$

with boundary conditions

$$\begin{cases} \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{p}(T) = 0 \end{cases} \tag{4.24}$$

From the third equation in (4.23), we have

$$\boldsymbol{u}(t) = -R^{-1}B^t\boldsymbol{p}(t) \text{ in } [0,T] \tag{4.25}$$

and thus

$$\begin{cases} \boldsymbol{x}' = A\boldsymbol{x} - (BR^{-1}B^t)\,\boldsymbol{p} \\ \boldsymbol{p}' = -Q\boldsymbol{x} - A^t\boldsymbol{p} \\ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{p}(T) = 0 \end{cases} \tag{4.26}$$

or, in compact form,

$$\begin{cases} \boldsymbol{X}'(t) = L\boldsymbol{X}(t) \\ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{p}(T) = 0 \end{cases} \tag{4.27}$$

where

$$\boldsymbol{X}(t) = \begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{p}(t) \end{pmatrix} \tag{4.28}$$

$$L = \begin{pmatrix} A & -BR^{-1}B^t \\ -Q & -A^t \end{pmatrix} \tag{4.29}$$

To date, we have derived the necessary conditions for optimality of a control $\boldsymbol{u}(t)$. Equations (4.18) are essentially the well-known *Pontryaguin's maximum principle*, whereas in equations (4.22), they are particularized for an LQR problem.

In summary, in order to find the control $\boldsymbol{u}(t)$ that minimizes (4.19), it is possible to solve the system of ordinary differential equations (4.27) for the state $\boldsymbol{u}(t)$ and the costate $\boldsymbol{p}(t)$ and then compute $\boldsymbol{u}(t)$ by means of equation (4.25).

One possible approach would be, after proving the existence of solutions to the optimal control problem, that of writing optimality conditions throughout Pontryaguin's maximum principle. Nonetheless, we shall follow a completely different path, specifically designed to produce a problem that can be simulated without the requirement of much expertise in the field of numerical methods.

One difficulty that might be encountered when constraints in the form $\boldsymbol{x}(T) = \boldsymbol{x}_T$ are to be respected is that the state in (4.27) propagates forward in time, while the costate propagates backwards. This is addressed in section 4.4.1, in which a method that can be employed to enforce such constraints is analysed.

As a final note in this section, we would like to recall that Q and R are often selected to be diagonal in engineering applications [33, 4, 7]. This assumption can simplify a great amount of the work when these problems are tackled numerically

4 4 1 Enforcement of final state constraints on the LQR

Let us now consider the following system of linear differential equations of size $2N$, with initial and final constraints on the state $\boldsymbol{x}(t)$

$$\begin{cases} \boldsymbol{X}'(t) = L\boldsymbol{X}(t) \text{ in } [0,T] \\ \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ \boldsymbol{x}(T) = \boldsymbol{x}_T \end{cases} \tag{4.30}$$

where $\boldsymbol{X}(t) = \begin{pmatrix} \boldsymbol{x}(t) \\ \boldsymbol{p}(t) \end{pmatrix}$, $\boldsymbol{x}(t) \in \mathbb{R}^N$ is the state and $\boldsymbol{p}(t) \in \mathbb{R}^N$ is the costate. $L(t) \in \mathbb{M}_{2N\times 2N}(\mathbb{R})$ originates from Pontryaguin's maximum principle (4.29).

The way in which the final state condition $\boldsymbol{x}(T) = \boldsymbol{x}_T$ can be imposed is not trivial. Here, the strategy is to find an initial condition for the costate at time $t = 0$, $\boldsymbol{p}(0)$ such that the state arrives at $\boldsymbol{x}_T$ at time $t = T$.

It is necessary to solve $N + 1$ cauchy problems

$$\boldsymbol{X}_i'(t) = L(t)\boldsymbol{X}_i(t) \quad i = 0, 1, 2, .., N \tag{4.31}$$

with initial conditions

$$\begin{aligned} \boldsymbol{X}_0(0) &= \begin{pmatrix} \boldsymbol{x}_0 \\ 0 \end{pmatrix} \\ \boldsymbol{X}_i(0) &= \begin{pmatrix} 0 \\ \boldsymbol{e}_i \end{pmatrix} \quad i = 1, 2, ..., N \end{aligned} \tag{4.32}$$

where $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \dots, \boldsymbol{e}_N\}$ is the canonical basis for $\mathbb{R}^N$

We then keep track of their solutions $\boldsymbol{x}_i(T),\ i = 0, 1, .., N$ at time T in order to build the matrix

$$A = (\boldsymbol{x}_1(T), \boldsymbol{x}_2(T), ..., \boldsymbol{x}_N(T)) \tag{4.33}$$

The main result obtained in this section is outlined in the following theorem.

Theorem 5 *The system (4.30) is equivalent to*

$$\begin{cases} \boldsymbol{X}'(t) = L\boldsymbol{X}(t) \ \textit{in} \ [0, T] \\ \boldsymbol{X}(0) = \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{p}(0) \end{pmatrix} \end{cases} \tag{4.34}$$

where $\boldsymbol{p}(0)$ *is the solution to*

$$A\boldsymbol{p}(0) + \boldsymbol{x}_0(T) = \boldsymbol{x}_T \tag{4.35}$$

In other words, if it is possible to compute $\boldsymbol{p}(0) = A^{-1}(\boldsymbol{x}_T - \boldsymbol{x}_0(T))$, the solution $\boldsymbol{X}(t)$ to (4.34), with no final constraint on the state or the costate, where the initial condition has been built upon the desired initial state $\boldsymbol{x}_0$, and the calculated initial costate $\boldsymbol{p}(0)$ will satisfy $\boldsymbol{x}(T) = \boldsymbol{x}_f$ and (trivially) $\boldsymbol{x}(0) = \boldsymbol{x}_0$.

Proof. Let us write

$$\boldsymbol{p}(0) = \sum_{i=1}^{N} \boldsymbol{p}_i \boldsymbol{e}_i$$

In virtue of the superposition principle, if $\boldsymbol{X}_0, \boldsymbol{X}_1, ..., \boldsymbol{X}_N$ are solutions for (4.31), then

$$\boldsymbol{X} = \boldsymbol{X}_0 + \sum_{i=1}^{N} \boldsymbol{p}_i \boldsymbol{X}_i$$

must be a solution for (4.34). Upon evaluating at final time $t = T$,

$$\boldsymbol{X}(T) = \boldsymbol{X}_0(T) + \sum_{i=1}^{N} \boldsymbol{p}_i \boldsymbol{X}_i(T)$$

Thus

$$\boldsymbol{x}(T) = \boldsymbol{x}_0(T) + \sum_{i=1}^{N} \boldsymbol{p}_i \boldsymbol{x}_i(T) = \boldsymbol{x}_0(T) + A\boldsymbol{p}(0) \tag{4.36}$$

Then introduce (4.35) to attain

$$\boldsymbol{x}(T) = \boldsymbol{x}_T \tag{4.37}$$

□

4.5 Outline of the numerical approximation algorithms

The following flowcharts (Figures 4.1 to 4.3) represent, in a very schematic manner, the numerical processes on which the presented algorithms are based. Note, however, that if a thorough understanding of these processes is sought, the sections in which the pertinent steps of the algorithms are explained would need to be consulted.

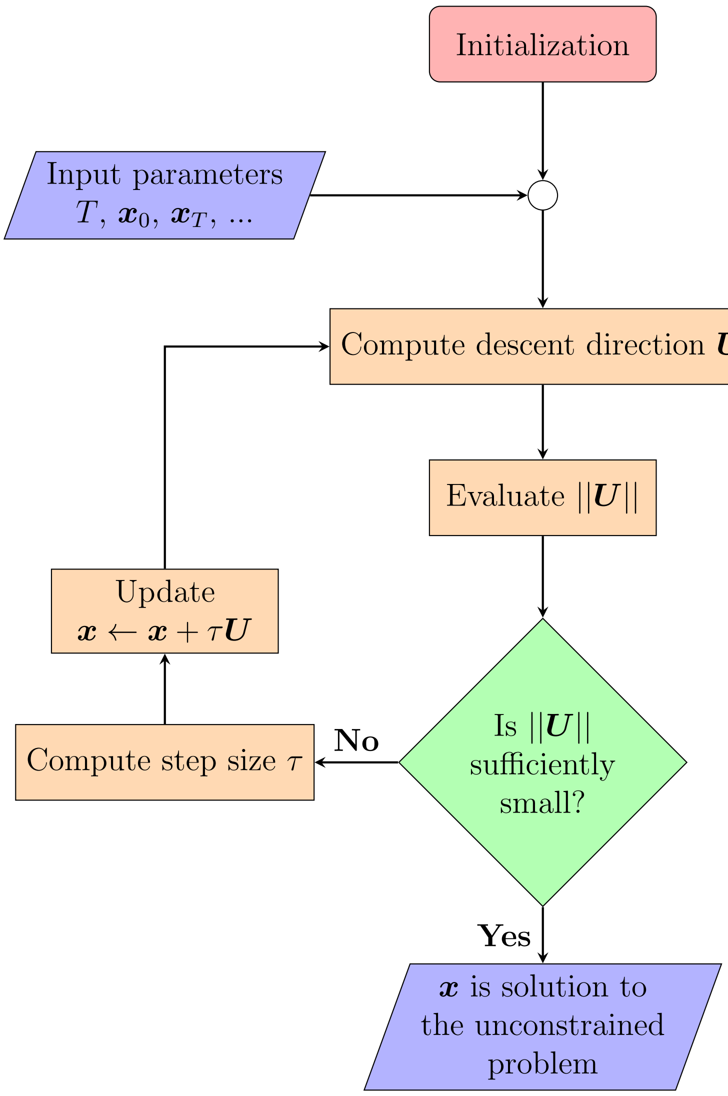

Initialization
Input parameters
T, $\boldsymbol{x}_0$, $\boldsymbol{x}_T$, ...
Compute descent direction $\boldsymbol{U}$
Evaluate $||\boldsymbol{U}||$
Update
$\boldsymbol{x} \leftarrow \boldsymbol{x} + \tau\boldsymbol{U}$
Compute step size τ
No
Is $||\boldsymbol{U}||$ sufficiently small?
Yes
$\boldsymbol{x}$ is solution to the unconstrained problem

Figure 4.1: Flowchart of the unconstrained minimization algorithm.

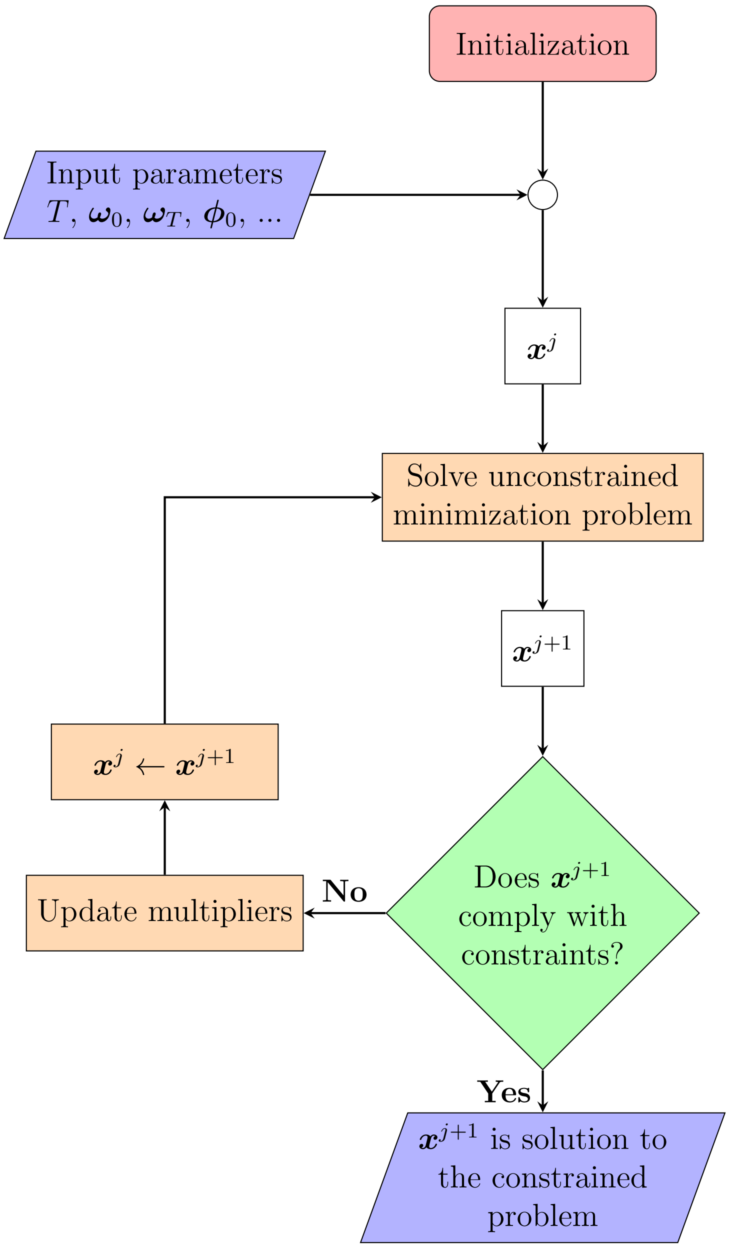

Initialization
Input parameters
T, $\boldsymbol{\omega}_0$, $\boldsymbol{\omega}_T$, $\boldsymbol{\phi}_0$, ...
$\boldsymbol{x}^j$
Solve unconstrained minimization problem
$\boldsymbol{x}^{j+1}$
$\boldsymbol{x}^j \leftarrow \boldsymbol{x}^{j+1}$
Update multipliers
No
Does $\boldsymbol{x}^{j+1}$ comply with constraints?
Yes
$\boldsymbol{x}^{j+1}$ is solution to the constrained problem

Figure 4.2: Flowchart of the constrained minimization algorithm.

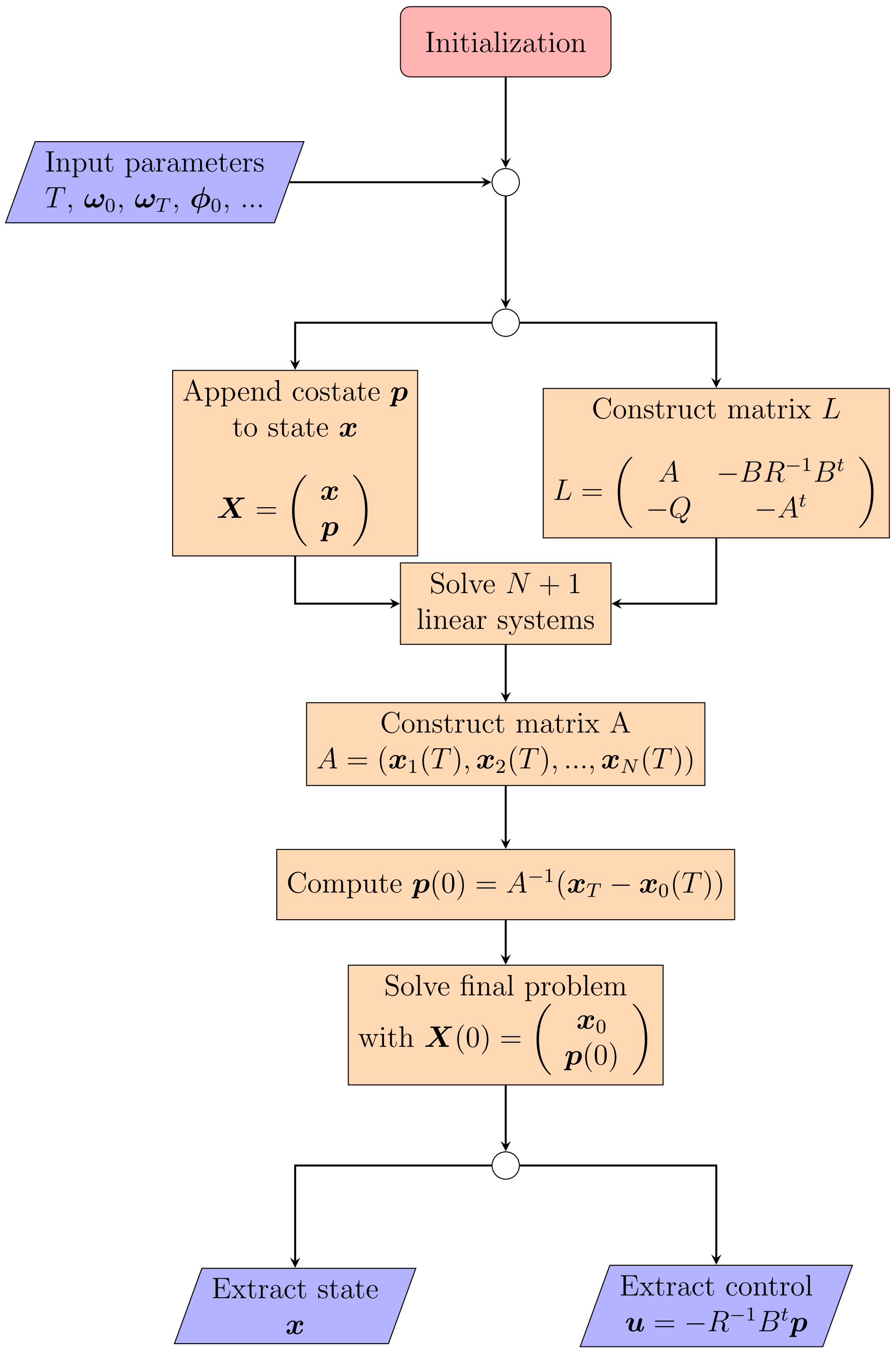

Initialization
Input parameters T, $\boldsymbol{\omega}_0$, $\boldsymbol{\omega}_T$, $\boldsymbol{\phi}_0$, ...
Append costate $\boldsymbol{p}$ to state $\boldsymbol{x}$
$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{p} \end{pmatrix}$
Construct matrix L
$L = \begin{pmatrix} A & -BR^{-1}B^t \\ -Q & -A^t \end{pmatrix}$
Solve $N+1$ linear systems
Construct matrix A
$A = (\boldsymbol{x}_1(T), \boldsymbol{x}_2(T), ..., \boldsymbol{x}_N(T))$
Compute $\boldsymbol{p}(0) = A^{-1}(\boldsymbol{x}_T - \boldsymbol{x}_0(T))$
Solve final problem with $\boldsymbol{X}(0) = \begin{pmatrix} \boldsymbol{x}_0 \\ \boldsymbol{p}(0) \end{pmatrix}$
Extract state $\boldsymbol{x}$
Extract control $\boldsymbol{u} = -R^{-1}B^t\boldsymbol{p}$

Figure 4.3: Flowchart of the algorithm to solve the LQR problem with end-point constraints.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions and future work

This work has addressed the problem of developing two algorithms for the attitude control of satellites from the viewpoint of Optimal Control. Two different approaches have been employed: in the first place, an LQR controller was implemented for the linearised equation of motion in terms of quaternions; secondly, the fully non-linear system was controlled using the variational reformulation of the corresponding optimal control. Both strategies have proved to be successful. There is, however, a large number of topics that could be explored further.

We have attempted to emphasize, whenever the topic arose, that controllability is an issue that must be addressed in this type of situations, but that we consider it to be far out of the scope of this project. Studying ontrollability in this type of situations is, therefore, proposed as one of the possible topics for future work.

It is clear that this work has addressed a purely academic situation, which is somewhat far from reality in the sense that it does not consider each and every aspect of a complete attitude control design. There are some concerns that future works may look at if a thorough implementation of these methods is sought. For example, future reports might incorporate the influence of orbital mechanics, magnetic actuation, gravitational anomalies, or the effect of disturbances such as gravity gradient and atmospheric drag (for low orbit satellites). It would also be interesting to study the in which the actuators can exert only a torque around one or two axes of the satellite – this situation is interesting because that is what happens with under actuated-satellites or when a fully actuated satellite loses control over some of its actuators. It has been proved that it is possible to control the attitude of under-actuated satellites [25, 42, 43, 44] even with magnetic actuation [27]. We also believe that studying the effect of uncertainty or incomplete measurements may be an interesting project to be carried out in the future [45].

From a more practical perspective, we would highly encourage the implementation the two exposed methods, or variations of them on an actual, physical test rig, and the evaluation of their performance and viability to be extended to real world applications in industry or research.

Our implementation of the algorithm leaves some room for improvement: even though both implementations work reasonably well, we are aware that an expert in the field of numerical methods would be able to improve the performance of both as regards accuracy and computational time.

As has been stated and showcased in Section 2.1.7, optimal controls for problems with inequatlity constraints on control have bang-bang behaviour as the time interval is reduced [46, 47, 48, 49, 50]. This may lead to future research: a future work is proposed in which *switching rules* (rules of criteria that allow the controller to decide when to switch from one of the extreme values to the other) are derived and appliend, either in experiment or in simulation. This type of analysis has already taken place in works suchs as [51, 52, 53].

5 2 Conclusiones y trabajos futuros

Este trabajo se ha centrado en desarrollar dos algoritmos para el control de orientación de satélites desde el punto de vista del Control Óptimo. Cada uno de los dos algoritmos ha seguido una estrategia distinta: en primer lugar, se implementó un Regulador Linear Cuadrático (LQR) basado en una linealización de las ecuaciones dinámicas en términos de cuaterniones; en segundo lugar, se implementó un controlador no lineal a través de cierto problema de control óptimo, el cual se resolvió numéricamente por medio de su reformulación variacional. Ambas estrategias han resultado exitosas. Sin embargo, existe un gran número de detalles que podrían ser explorados con más profundidad.

En todo momento se ha intentado enfatizar el hecho de que la controlabilidad es un asunto que debe tenerse en cuenta en este tipo de situaciones, sin embargo, esto se ha considerado fuera del alcance del trabajo. Estudiar la controlabilidad en este tipo de sistemas es, por tanto, uno de los problemas propuestos para futuros trabajos.

El presente trabajo ha estudiado un problema de naturaleza académica, el cual no contempla todos y cada uno de los aspectos necesarios en la implementación de un problema de control. Posibles trabajos futuros podrían, por ejemplo, centrarse en la incorporación de actuación magnética, o el efecto de perturbaciones, como el gradiente gravitacional o la fricción aerodinámica (cuyo efecto es más notable en satélites de órbita baja). Sería también interesante estudiar el problema análogo a este trabajo, pero donde los actuadores no sean capaces de ejercer un par en los tres ejes del satélite; el interés de esta situación reside en su aplicación a satélites que han perdido control sobre uno de sus actuadores. Se ha probado en distintos trabajos que dicha configuración puede llegar a ser controlable [25, 42, 43, 44], incluso cuando se emplea actuación magnética [27]. Estudiar el efecto de la incertidumbre en las medidas, o medidas incompletas podría constituir también la base para un interesante trabajo en el futuro [45].

Desde un punto de vista más práctico, nos gustaría animar a futuros estudiantes a realizar la implementación de los métodos de control expuestos, o variaciones de los mismos, en un entorno de laboratorio, así como la evaluación de su rendimiento y su viabilidad para ser empleados en la industria.

La implementación numérica de los algoritmos presentados en este trabajo deja, sin duda, margen de mejora: aunque es cierto que ambos algoritmos funcionan razonablemente bien, es probable que un experto en el campo de la simulación numérica sea capaz de mejorar el rendimiento de los mismos, así como de aumentar su precisión y reducir sus tiempos de cálculo. Se propone, por tanto, esta tarea como un posible trabajo para el futuro.

Se ha comentado brevemente en la Sección 2.1.7, que controles óptimos en problemas con restricciones en forma de desigualdad en el control presentan un comportamiento de tipo *bang-bang*, esto es, saltan abruptamente entre los dos extremos [46, 47, 48, 49, 50]. Esto puede dar lugar a futuros trabajos de investigación en los que, por ejemplo, se deriven métodos para predecir dichos saltos abruptos. Este tipo de análisis ya ha tenido lugar en trabajos como [51, 52, 53].

Appendices

APPENDIX A

COMPUTER CODE

In this chapter, the subroutines used to carry out all numerical simulations of this work are presented. They are all written in Fortran 95 language.

A 1

Linearised model

```
! ============================================================
! satellites_linear.f95
!
! This program calculates the control and state of a
! fully actuated satellite using optimal control with
! quadratic cost
!       I(u) = 1/2 * integ( x*Qx + u*Ru ) dt
!
! It uses a linearised model of the form
!       x' = Ax + Bu
! Presented by Yang in [Yang, Y. Analytic LQR design for
! spacecraft control system based on quaternion model.
! Journal of Aerospace Engineering, 25(3):448-453, jul 2012]
!
! INPUT:
! - "input_file": This file is created by the user.
! It contains the simulation parameters as follows:
!  T_0: Initial time
!  T_1: Final time
!  j1 j2 j3: Moments of inertia
!  q1 q2 q3 q4 q5 q6: Elemets of diagonal matrix q
!  r1 r2 r3: Elements of diagonal matrix R
!  omega_0(1) omega_0(2), omega_0(3): Initial ang. vel.
!  omega_1(1) omega_1(2), omega_1(3): Final ang. vel.
!  eul_0(1) eul_0(2) eul_0(3): Initial Euler angles
!  eul_1(1) eul_1(2) eul_1(3): Final Euler angles
!  m: Number of intervals in the discretization
!
! OUTPUT:
! - "results_control": Control
! - "results_angveloc": Angular velocities
! - "results_eulerang": Euler angles
! - "results_quaternion": Quaternion
! - "results_time": Time vector
! - "results_misc": Miscelaneous results (Cost, CPU time...)
!
! The convention for quaternion and
! Euler angles is as follows:
! q = (epsilon,eta)
! eul = (phi,theta,rho) = (yaw,pitch,roll)
!
! This program depends on LAPACK library: It must be
! compiled with -llapack and -lblas.
! ==========================================================

! ======== INITIALIZATION ========
! Variables declared here are passed to all subroutines
module initialization
implicit none
real:: q1, q2, q3, q4, q5, q6
real:: r1, r2, r3
real:: j1, j2, j3
end module initialization

! ======== MAIN PROGRAM ========
program main
use initialization
implicit none
interface
        subroutine func(n,x,t,y)
        integer, intent(in):: n
        real, intent(in):: x(n), t
        real, intent(out):: y(n)
        end subroutine func
end interface

integer, allocatable:: ipiv(:)
real, allocatable:: state_0(:), state_1(:)
real, allocatable:: state_aux_0(:), aux_solution_0_T(:)
real, allocatable:: aux_solutions(:,:), costate_0(:)
real, allocatable:: lu_fact(:,:)
real, allocatable:: state(:,:), state_costate(:,:), time(:)
integer:: n, nm, m, i, j, k
```

```
integer:: info
real:: omega_0(3), omega_1(3), eul_0(3), eul_1(3)
real:: quat_0(4), quat_1(4), eulang(3)
real:: t_0, t_1, pi, cpu_clock_ini, cpu_clock_fin
real:: cost, h
n = 6 ! Dimension of the problem
nm = 3 ! Dimension of control
pi = 4.*atan(1.)

! Open input file
open (11,file='input_file',status='old')
! Load input parameters from file
read(11,*) T_0
read(11,*) T_1
read(11,*) j1, j2, j3
read(11,*) q1, q2, q3, q4, q5, q6
read(11,*) r1, r2, r3
read(11,*) omega_0(1), omega_0(2), omega_0(3)
read(11,*) omega_1(1), omega_1(2), omega_1(3)
read(11,*) eul_0(1), eul_0(2), eul_0(3)
read(11,*) eul_1(1), eul_1(2), eul_1(3)
read(11,*) m
close(11)

allocate (state_0(n), state_1(n))
allocate (lu_fact(n,n), ipiv(n))
allocate (state_aux_0(2*n), aux_solution_0_T(n))
allocate (aux_solutions(n,n), costate_0(n))
allocate (state(m+1,n), state_costate(m+1,2*n), time(m+1))

! Here the state is (w,epsilon).
! Eta is calculated afterwards imposing |q|=1
call eul2quat(eul_0,quat_0)
call eul2quat(eul_1,quat_1)
state_0 = (/omega_0,quat_0(1:3)/)
state_1 = (/omega_1,quat_1(1:3)/)

call CPU_TIME(cpu_clock_ini)

! Solve n+1 auxiliary problems
! Problem i=0
state_aux_0 = 0. ! Construct initial condition
do k = 1,n
        state_aux_0(k) = state_0(k)
end do

! Call the solver
call rk4(func,2*n,state_aux_0,t_0,t_1,m,state_costate,time)
```

```fortran
do k = 1,n ! Save solution at final time
        aux_solution_0_T(k) = state_costate(m+1,k)
end do

! Problems i=1,...,N
aux_solutions = 0.
do i = 1,n
        ! Construct init. condition (Canonical basis of Rn)
        state_aux_0 = 0.
        state_aux_0(n+i) = 1.
        ! Call the solver
        call rk4(func,2*n,state_aux_0,t_0,t_1, &
        m,state_costate,time)
        do k = 1,n ! Save solution at final time
                aux_solutions(k,i) = state_costate(m+1,k)
        end do
end do

! Solve p_0 = A^-1 (x_T-x_0(T))
! costate_0 = inv(aux_solutions)*(state_1-aux_solution_0_T)
! LU factorization
lu_fact = aux_solutions
call sgetrf(n,n,lu_fact,n,ipiv,info)
costate_0 =  state_1-aux_solution_0_T
! Solve linear problem
call sgetrs('n',n,1,lu_fact,n,ipiv,costate_0,n,info)

! Final problem
do k = 1,n
        state_aux_0(k) = state_0(k)
        state_aux_0(k+n) = costate_0(k)
end do

call rk4(func,2*n,state_aux_0,t_0,t_1,m,state_costate,time)

call CPU_TIME(cpu_clock_fin)

cost = 0. ! Calculate cost
h = (t_1-t_0)/m
do i = 1,m+1
        cost = cost + h*(q1*state_costate(i,1)**2 &
        + q2*state_costate(i,2)**2 &
        + q3*state_costate(i,3)**2 &
        + q4*state_costate(i,4)**2 &
        + q5*state_costate(i,5)**2 &
        + q6*state_costate(i,6)**2 &
        + r1*(-state_costate(i,7)/(j1*r1))**2 &
        + r2*(-state_costate(i,8)/(j2*r2))**2 &
        + r3*(-state_costate(i,9)/(j3*r3))**2)
end do

! Save results
open(11,file='results_misc', &
status='replace',action='write')
write(11,*) 'LINEAR␣MODEL;'
write(11,*) 'Initial␣time␣T_0;', t_0
write(11,*) 'Final␣time␣T_1;', t_1
write(11,*) 'Moments␣of␣inertia;', j1, j2, j3
write(11,*) 'Q␣(diagonal␣matix);', q1, q2, q3, q4, q5, q6
write(11,*) 'R␣(diagonal␣matix);', r1, r2, r3
write(11,*) 'Initial␣angular␣velocity␣w_0;', omega_0
write(11,*) 'Initial␣angular␣velocity␣w_1;', omega_1
write(11,*) 'Initial␣orientation␣(yaw,pitch,roll);', eul_0
write(11,*) 'Final␣orientation␣(yaw,pitch,roll);', eul_1
write(11,*) '#␣Intervals␣discretization;', m
write(11,*) 'Cost␣I(u)␣=␣', cost
write(11,*) 'CPU␣time␣(s);', cpu_clock_fin-cpu_clock_ini
close(11)

open(11,file='results_control_LQR', &
status='replace',action='write')
do i = 1,m+1
        write(11,*) -state_costate(i,7)/(j1*r1), &
                    -state_costate(i,8)/(j2*r2), &
                    -state_costate(i,9)/(j3*r3)
end do
close(11)

open(11,file='results_control', &
status='replace',action='write')
do i = 1,m
        write(11,*) &
        (state_costate(i+1,1) - &
        state_costate(i,1))*m/(T_1-T_0), &
        (state_costate(i+1,2) - &
        state_costate(i,2))*m/(T_1-T_0), &
        (state_costate(i+1,3) - &
        state_costate(i,3))*m/(T_1-T_0)
end do
close(11)

open(11,file='results_quaternion', &
status='replace',action='write')
do i = 1,m+1
        write(11,*) &
        state_costate(i,4:6), &
```

```fortran
        (1-state_costate(i,4)**2 &
        -state_costate(i,5)**2 &
        -state_costate(i,6)**2)**.5
end do
close(11)

open(11,file='results_eulerang', &
status='replace',action='write')
do i = 1,m+1
        call quat2eul((/state_costate(i,4:6), &
        (1-state_costate(i,4)**2 &
        -state_costate(i,5)**2 &
        -state_costate(i,6)**2)**.5/) &
        ,eulang)
        write (11,*) eulang
end do
close(11)

open(11,file='results_angveloc', &
status='replace',action='write')
do i = 1,m+1
        write (11,*) state_costate(i,1:3)
end do
close(11)

! Here it does not make a lot of sense to take |q| as
! an output, as we precisely imposed |q|=1 to get q_4

open(11, file='results_time', &
status='replace',action='write')
do i = 1, m+1
        write (11,*) T_0+(T_1-T_0)*(i-1)/m
end do
close (11)

deallocate (ipiv)
deallocate (state_0, state_1)
deallocate (state_aux_0, aux_solution_0_T)
deallocate (aux_solutions, costate_0)
deallocate (lu_fact)
deallocate (state, state_costate, time)

end program main

! ======== 4TH ORDER RUNGE-KUTTA METHOD ========
subroutine rk4(func,n,x0,t0,t1,m,x,time)
implicit none
!interface
```

```fortran
!        subroutine func(n,x,t,y)
!        integer, intent(in):: n
!        real, intent(in):: x(n), t
!        real, intent(out):: y(n)
!        end subroutine func
!end interface
integer, intent(in):: n, m
real, intent(in):: x0(n), t0, t1
real, intent(out):: x(m+1,n), time(m+1)
real:: h, a1(N), a2(N), a3(N), a4(N)
integer:: i, k
h = (t1-t0)/m

do i = 1,n
        x(1,i) = x0(i)
end do

do k = 1,m
        time(k) = t0 + (k-1)*h
        call func(n,x(k,:),time(k),a1)
        a1 = h*a1
        call func(n,x(k,:)+.5*a1,time(k)+h/2.,a2)
        a2 = h*a2
        call func(n,x(k,:)+.5*a2,time(k)+h/2.,a3)
        a3 = h*a3
        call func(n,x(k,:)+a3,time(k)+h,a4)
        a4 = h*a4
        do i = 1,n
                x(k+1,i) = x(k,i) &
                + (a1(i) + 2*a2(i)  +2*a3(i) + a4(i))/6.
        end do
end do
time(m+1) = t1
end subroutine rk4

! ======== DYNAMICS ========
! y = dx = Lx, where L is the matrix related
! to the LQR problem. That is:
! L =  ( A , -B R^-1 B* )
!      (-Q ,     -A*    )
subroutine func(n,x,t,y)
use initialization
implicit none
integer, intent(in):: n
real, intent(in):: x(n), t
real, intent(out):: y(n)
y(1) =  -x(7)/(r1*j1**2)
y(2) =  -x(8)/(r2*j2**2)
```

```
y(3) =  -x(9)/(r3*j3**2)
y(4) =   x(1)*.5
y(5) =   x(2)*.5
y(6) =   x(3)*.5
y(7) =  -x(10)*.5 - q1*x(1)
y(8) =  -x(11)*.5 - q2*x(2)
y(9) =  -x(12)*.5 - q3*x(3)
y(10) = -q4*x(4)
y(11) = -q5*x(5)
y(12) = -q6*x(6)
end subroutine func

! ======== EULER ANGLES TO QUATERNION ========
subroutine eul2quat(eul,q)
real, intent(in):: eul(3)
real, intent(out):: q(4)
real:: cy, sy, cp, sp, cr, sr
cy = cos(eul(1)*.5);
sy = sin(eul(1)*.5);
cp = cos(eul(2)*.5);
sp = sin(eul(2)*.5);
cr = cos(eul(3)*.5);
sr = sin(eul(3)*.5);
q(1) = cy * cp * sr - sy * sp * cr
q(2) = sy * cp * sr + cy * sp * cr
q(3) = sy * cp * cr - cy * sp * sr
q(4) = cy * cp * cr + sy * sp * sr
end subroutine eul2quat

! ======== QUATERNION TO EULER ANGLES ========
subroutine quat2eul(q,eul)
real, intent(in):: q(4)
real, intent(out):: eul(3)
eul(1) = atan2(2.*(q(4)*q(3)+q(1)*q(2)), &
1.-2.*(q(2)**2.+q(3)**2.))
eul(2) = asin(2.*(q(4)*q(2)-q(3)*q(1)))
eul(3) = atan2(2.*(q(4)*q(1)+q(2)*q(3)), &
1.-2.*(q(1)**2.+q(2)**2.))
end subroutine quat2eul
```

A 2 Nonlinear model

```
! ==============================================================
! satellites_nonlinear.f95
!
```

```
! This program calculates the control and state of a
! fully actuated satellite using optimal control with
! quadratic cost
!       I(u) = 1/2 * integ( x*Qx + u*Ru ) dt
!
! And constraints on control
!       ui_min <= u_i <= ui_max ,  i=1,2,3.
!
! It uses the fully non-linear model
!       Jw' = -w x Jw + u
!       q' = 1/2 * Omega(w) * q
!
! INPUT:
! - "input_file": This file is created by the user.
! It contains the simulation parameters as follows:
!  T_0: Initial time
!  T_1: Final time
!  j1 j2 j3: Moments of inertia
!  q1 q2 q3 q4 q5 q6 q7: Elemets of diagonal matrix q
!  r1 r2 r3: Elements of diagonal matrix R
!  omega_0(1) omega_0(2), omega_0(3): Initial ang. vel.
!  omega_1(1) omega_1(2), omega_1(3): Final ang. vel.
!  eul_0(1) eul_0(2) eul_0(3): Initial Euler angles
!  eul_1(1) eul_1(2) eul_1(3): Final Euler angles
!  m: Number of intervals in the discretization
!  u1min u2min u3min u1max u2max u3max: Control constraints
!  mask(1) ... mask(7): Natural/Essential bound. constraints
!  prec: Precission of the algorithm
!
! OUTPUT:
! - "results_control": Control
! - "results_angveloc": Angular velocities
! - "results_eulerang": Euler angles
! - "results_quaternion": Quaternion
! - "results_quatnorm": Norm of quaternion
! - "results_time": Time vector
! - "results_converg": Convergence
! - "results_misc": Miscelaneous results (Cost, CPU time...)
!
! The convention for quaternion and
! Euler angles is as follows:
! q = (epsilon,eta)
! eul = (phi,theta,rho) = (yaw,pitch,roll)
! ==========================================================

! ======== INITIALIZATION ========
! Variables declares here are passed to all subroutines
module initialization
```

```fortran
implicit none
integer:: N, m, nn
real:: T_0, T_1
real:: j1, j2, j3
real:: q1, q2, q3, q4, q5, q6, q7
real:: r1, r2, r3
real:: u1min, u1max, u2min, u2max, u3min, u3max
end module initialization

! ======== MAIN ========
program main
use initialization
implicit none

integer, allocatable:: mask(:)
real, allocatable:: state_0(:), state_1(:), state(:, :)
integer:: i
real:: cpu_clock_ini, cpu_clock_fin
real:: prec, cost, eulang(3), pi
real:: omega_0(3), omega_1(3)
real:: eul_0(3), eul_1(3), quat_0(4), quat_1(4)
N=7; ! Dimension of state
nn=14;  ! Number of constraints
pi = 4*atan(1.)
allocate (state_0(N), state_1(N), mask(N))

! Open input file
open (11,file='input_file',status='old')
! Load input parameters from file
read(11,*) T_0
read(11,*) T_1
read(11,*) j1, j2, j3
read(11,*) q1, q2, q3, q4, q5, q6, q7
read(11,*) r1, r2, r3
read(11,*) omega_0(1), omega_0(2), omega_0(3)
read(11,*) omega_1(1), omega_1(2), omega_1(3)
read(11,*) eul_0(1), eul_0(2), eul_0(3)
read(11,*) eul_1(1), eul_1(2), eul_1(3)
read(11,*) m
read(11,*) u1min, u2min, u3min, u1max, u2max, u3max
read(11,*) mask(1), mask(2), mask(3), mask(4),&
           mask(5), mask(6), mask(7)
read(11,*) prec
close(11)

allocate (state(m+1,N))

call eul2quat(eul_0,quat_0)
call eul2quat(eul_1,quat_1)
state_0 = (/omega_0,quat_0/)
state_1 = (/omega_1,quat_1/)

do i=1, m+1
        state(i, :)=((m+1.-i)/m)*state_0+((i-1.)/m)*state_1
end do

call CPU_TIME(cpu_clock_ini)
call optimalcontrol(prec, mask, state, cost)
call CPU_TIME(cpu_clock_fin)

! Save results
open(11,file='results_misc', &
status='replace',action='write')
write(11,*) 'NONLINEAR␣MODEL;'
write(11,*) 'Initial␣time␣T_0;', t_0
write(11,*) 'Final␣time␣T_1;', t_1
write(11,*) 'Moments␣of␣inertia;', j1, j2, j3
write(11,*) 'Q␣(diagonal␣matix);', q1, q2, q3, q4, q5, q6
write(11,*) 'R␣(diagonal␣matix);', r1, r2, r3
write(11,*) 'Initial␣angular␣velocity␣w_0;', omega_0
write(11,*) 'Initial␣angular␣velocity␣w_1;', omega_1
write(11,*) 'Initial␣orientation␣(yaw,pitch,roll);', eul_0
write(11,*) 'Final␣orientation␣(yaw,pitch,roll);', eul_1
write(11,*) '#␣Intervals␣discretization;', m
write(11,*) 'Precision;', prec
write(11,*) 'Mask␣(1=bound.cond./0=natural);', mask
write(11,*) 'Cost␣I(u);', cost
write(11,*) 'CPU␣time␣(s);', cpu_clock_fin-cpu_clock_ini
close(11)

open(11,file='results_control', &
status='replace',action='write')
do i = 1,m
        write(11,*) &
        (state(i+1,1) - state(i,1))*m/(T_1-T_0), &
        (state(i+1,2) - state(i,2))*m/(T_1-T_0), &
        (state(i+1,3) - state(i,3))*m/(T_1-T_0)
end do
close(11)

open(11,file='results_quaternion', &
status='replace',action='write')
do i = 1,m+1
        write(11,*) state(i,4:7)
end do
close(11)
```

```fortran
open(11,file='results_eulerang', &
status='replace',action='write')

do i = 1,m+1
        call quat2eul(state(i,4:7),eulang)
        write (11,*) eulang
end do
close(11)

open(11,file='results_angveloc', &
status='replace',action='write')
do i = 1,m+1
        write (11,*) state(i,1:3)
end do
close(11)

open(11,file='results_quatnorm', &
status='replace',action='write')
do i = 1, m+1
        write (11,*) &
        state(i,4)**2+state(i,5)**2+ &
        state(i,6)**2+state(i,7)**2
end do
close (11)

open(11,file='results_time', &
status='replace',action='write')
do i = 1, m+1
        write (11,*) T_0+(T_1-T_0)*(i-1)/m
end do
close (11)

deallocate (state_0, state_1, state, mask)
end program main

! ======== CONSTRAINTS ========
subroutine constraints(s, x, p, R)
use initialization
implicit none
real:: s, x(N), p(N), R(nn)
R (1)  =  j1*p(1) + (j3-j2)*(x(2)*x(3)) - u1max
R (2)  = -j1*p(1) - (j3-j2)*(x(2)*x(3)) + u1min
R (3)  =  j2*p(2) + (j1-j3)*(x(1)*x(3)) - u2max
R (4)  = -j2*p(2) - (j1-j3)*(x(1)*x(3)) + u2min
R (5)  =  j3*p(3) + (j2-j1)*(x(1)*x(2)) - u3max
R (6)  = -j3*p(3) - (j2-j1)*(x(1)*x(2)) + u3min
R (7)  =  p(4) - .5*(+ x(1)*x(7) - x(2)*x(6) + x(3)*x(5))
R (8)  = -p(4) + .5*(+ x(1)*x(7) - x(2)*x(6) + x(3)*x(5))
```

```fortran
R (9)  =  p(5) - .5*(+ x(1)*x(6) - x(3)*x(4) + x(2)*x(7))
R (10) = -p(5) + .5*(+ x(1)*x(6) - x(3)*x(4) + x(2)*x(7))
R (11) =  p(6) - .5*(+ x(2)*x(4) - x(1)*x(5) + x(3)*x(7))
R (12) = -p(6) + .5*(+ x(2)*x(4) - x(1)*x(5) + x(3)*x(7))
R (13) =  p(7) - .5*(- x(1)*x(4) - x(2)*x(5) - x(3)*x(6))
R (14) = -p(7) + .5*(- x(1)*x(4) - x(2)*x(5) - x(3)*x(6))
end subroutine constraints

! ======== INTEGRAND ========
subroutine integrand(s, x, p, y, F)
use initialization
implicit none
real, intent(in):: s, x(N), p(N), y(m, nn)
real, intent(out):: F
real:: l(nn), R(nn)
integer:: i
call constraints(s, x, p, R)
call indexfunction(s, y, l)
! 0.5 * u^tRu
F =     (r1*(j1*p(1) - j2*x(2)*x(3) + j3*x(2)*x(3))**2)/2
F = F + (r2*(j2*p(2) + j1*x(1)*x(3) - j3*x(1)*x(3))**2)/2
F = F + (r3*(j3*p(3) - j1*x(1)*x(2) + j2*x(1)*x(2))**2)/2
! + 0.5 * x^tQx
F = F + (q1*x(1)**2)/2
F = F + (q2*x(2)**2)/2
F = F + (q3*x(3)**2)/2
F = F + (q4*x(4)**2)/2
F = F + (q5*x(5)**2)/2
F = F + (q6*x(6)**2)/2
F = F + (q7*x(7)**2)/2
F = F - nn
do i = 1,nn
        F = F + exp(l(i)*R(i))
end do
end subroutine integrand

! ======== AUX INTEGRAND 1 ========
subroutine auxintegrand1(s, x, p, y, H)
use initialization
implicit none
real, intent(in)::  s, x(N), p(N), y(m, nn)
real, intent(out):: H(N)
real:: l(nn)!, R(nn)
!call constraints(s, x, p, R)
call indexfunction(s, y, l)
H(1)=q1*x(1)-(l(7)*x(7)*exp(l(7)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(1)=H(1)+(l(8)*x(7)*exp(-l(8)*(p(4)-&
                (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(1)=H(1)-(l(9)*x(6)*exp(l(9)*(p(5)-&
                (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(1)=H(1)+(l(10)*x(6)*exp(-l(10)*(p(5)-&
                (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(1)=H(1)+(l(11)*x(5)*exp(l(11)*(p(6)+&
                (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(1)=H(1)-(l(12)*x(5)*exp(-l(12)*(p(6)+&
                (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(1)=H(1)+(l(13)*x(4)*exp(l(13)*(p(7)+&
                (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(1)=H(1)-(l(14)*x(4)*exp(-l(14)*(p(7)+&
                (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(1)=H(1)-r3*(j1*x(2)-j2*x(2))*&
                (j3*p(3)-j1*x(1)*x(2)+j2*x(1)*x(2))
H(1)=H(1)+r2*(j1*x(3)-j3*x(3))*&
                (j2*p(2)+j1*x(1)*x(3)-j3*x(1)*x(3))
H(1)=H(1)+l(3)*x(3)*&
                exp(l(3)*(j2*p(2)+x(1)*x(3)*(j1-j3)-1))*(j1-j3)
H(1)=H(1)-l(4)*x(3)*&
                exp(-l(4)*(j2*p(2)+x(1)*x(3)*(j1-j3)+1))*(j1-j3)
H(1)=H(1)-l(5)*x(2)*&
                exp(-l(5)*(x(1)*x(2)*(j1-j2)-j3*p(3)+1))*(j1-j2)
H(1)=H(1)+l(6)*x(2)*&
                exp(-l(6)*(j3*p(3)-x(1)*x(2)*(j1-j2)+1))*(j1-j2)

H(2)=q2*x(2)+(l(7)*x(6)*exp(l(7)*(p(4)-&
                (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(2)=H(2)-(l(8)*x(6)*exp(-l(8)*(p(4)-&
                (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(2)=H(2)-(l(9)*x(7)*exp(l(9)*(p(5)-&
                (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(2)=H(2)-(l(11)*x(4)*exp(l(11)*(p(6)+&
                (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(2)=H(2)+(l(10)*x(7)*exp(-l(10)*(p(5)-&
                (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(2)=H(2)+(l(12)*x(4)*exp(-l(12)*(p(6)+&
                (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(2)=H(2)+(l(13)*x(5)*exp(l(13)*(p(7)+&
                (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(2)=H(2)-(l(14)*x(5)*exp(-l(14)*(p(7)+&
                (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(2)=H(2)-r3*(j1*x(1)-j2*x(1))*&
                (j3*p(3)-j1*x(1)*x(2)+j2*x(1)*x(2))
H(2)=H(2)-r1*(j2*x(3)-j3*x(3))*&
                (j1*p(1)-j2*x(2)*x(3)+j3*x(2)*x(3))
H(2)=H(2)-l(1)*x(3)*&
                exp(-l(1)*(x(2)*x(3)*(j2-j3)-j1*p(1)+1))*(j2-j3)
```

```
H(2)=H(2)+l(2)*x(3)*&
        exp(-l(2)*(j1*p(1)-x(2)*x(3)*(j2-j3)+1))*(j2-j3)
H(2)=H(2)-l(5)*x(1)*&
        exp(-l(5)*(x(1)*x(2)*(j1-j2)-j3*p(3)+1))*(j1-j2)
H(2)=H(2)+l(6)*x(1)*&
        exp(-l(6)*(j3*p(3)-x(1)*x(2)*(j1-j2)+1))*(j1-j2)

H(3)=q3*x(3)-(l(7)*x(5)*exp(l(7)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(3)=H(3)+(l(8)*x(5)*exp(-l(8)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(3)=H(3)+(l(9)*x(4)*exp(l(9)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(3)=H(3)-(l(10)*x(4)*exp(-l(10)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(3)=H(3)-(l(11)*x(7)*exp(l(11)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(3)=H(3)+(l(12)*x(7)*exp(-l(12)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(3)=H(3)+(l(13)*x(6)*exp(l(13)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(3)=H(3)-(l(14)*x(6)*exp(-l(14)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(3)=H(3)+r2*(j1*x(1)-j3*x(1))*&
        (j2*p(2)+j1*x(1)*x(3)-j3*x(1)*x(3))
H(3)=H(3)-r1*(j2*x(2)-j3*x(2))*&
        (j1*p(1)-j2*x(2)*x(3)+j3*x(2)*x(3))
H(3)=H(3)-l(1)*x(2)*&
        exp(-l(1)*(x(2)*x(3)*(j2-j3)-j1*p(1)+1))*(j2-j3)
H(3)=H(3)+l(3)*x(1)*&
        exp(l(3)*(j2*p(2)+x(1)*x(3)*(j1-j3)-1))*(j1-j3)
H(3)=H(3)+l(2)*x(2)*&
        exp(-l(2)*(j1*p(1)-x(2)*x(3)*(j2-j3)+1))*(j2-j3)
H(3)=H(3)-l(4)*x(1)*&
        exp(-l(4)*(j2*p(2)+x(1)*x(3)*(j1-j3)+1))*(j1-j3)

H(4)=q4*x(4)+(l(9)*x(3)*exp(l(9)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(4)=H(4)-(l(10)*x(3)*exp(-l(10)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(4)=H(4)-(l(11)*x(2)*exp(l(11)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(4)=H(4)+(l(12)*x(2)*exp(-l(12)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(4)=H(4)+(l(13)*x(1)*exp(l(13)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(4)=H(4)-(l(14)*x(1)*exp(-l(14)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
```

```fortran

H(5)=q5*x(5)-(l(7)*x(3)*exp(l(7)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(5)=H(5)+(l(8)*x(3)*exp(-l(8)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(5)=H(5)+(l(11)*x(1)*exp(l(11)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(5)=H(5)-(l(12)*x(1)*exp(-l(12)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(5)=H(5)+(l(13)*x(2)*exp(l(13)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(5)=H(5)-(l(14)*x(2)*exp(-l(14)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2

H(6)=q6*x(6)+(l(7)*x(2)*exp(l(7)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(6)=H(6)-(l(8)*x(2)*exp(-l(8)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(6)=H(6)-(l(9)*x(1)*exp(l(9)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(6)=H(6)+(l(10)*x(1)*exp(-l(10)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(6)=H(6)+(l(13)*x(3)*exp(l(13)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2
H(6)=H(6)-(l(14)*x(3)*exp(-l(14)*(p(7)+&
        (x(1)*x(4))/2+(x(2)*x(5))/2+(x(3)*x(6))/2)))/2

H(7)=q7*x(7)-(l(7)*x(1)*exp(l(7)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(7)=H(7)+(l(8)*x(1)*exp(-l(8)*(p(4)-&
        (x(1)*x(7))/2+(x(2)*x(6))/2-(x(3)*x(5))/2)))/2
H(7)=H(7)-(l(9)*x(2)*exp(l(9)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(7)=H(7)+(l(10)*x(2)*exp(-l(10)*(p(5)-&
        (x(1)*x(6))/2+(x(3)*x(4))/2-(x(2)*x(7))/2)))/2
H(7)=H(7)-(l(11)*x(3)*exp(l(11)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
H(7)=H(7)+(l(12)*x(3)*exp(-l(12)*(p(6)+&
        (x(1)*x(5))/2-(x(2)*x(4))/2-(x(3)*x(7))/2)))/2
end subroutine auxintegrand1

! ======== AUX INTEGRAND 2 ========
subroutine auxintegrand2(s, x, p, y, T, G)
use initialization
implicit none
real, intent(in):: s, x(N), p(N), y(m, nn), T
real, intent(out):: G(N)
integer:: i
```

```
real:: l(nn), H(N)!, R(nn)
!call constraints(s, x, p, R)
call indexfunction(s, y, l)
call auxintegrand1(s, x, p, y, H)
G(1)=j1*r1*(j1*p(1)-j2*x(2)*x(3)+j3*x(2)*x(3))
G(1)=G(1)+j1*l(1)*exp(-l(1)*(x(2)*x(3)*(j2-j3)-j1*p(1)+1))
G(1)=G(1)-j1*l(2)*exp(-l(2)*(j1*p(1)-x(2)*x(3)*(j2-j3)+1))
G(1)=G(1)-(T-s)*H(1)

G(2)=j2*r2*(j2*p(2)+j1*x(1)*x(3)-j3*x(1)*x(3))
G(2)=G(2)+j2*l(3)*exp(l(3)*(j2*p(2)+x(1)*x(3)*(j1-j3)-1))
G(2)=G(2)-j2*l(4)*exp(-l(4)*(j2*p(2)+x(1)*x(3)*(j1-j3)+1))
G(2)=G(2)-(T-s)*H(2)

G(3)=j3*r3*(j3*p(3)-j1*x(1)*x(2)+j2*x(1)*x(2))
G(3)=G(3)+j3*l(5)*exp(-l(5)*(x(1)*x(2)*(j1-j2)-j3*p(3)+1))
G(3)=G(3)-j3*l(6)*exp(-l(6)*(j3*p(3)-x(1)*x(2)*(j1-j2)+1))
G(3)=G(3)-(T-s)*H(3)

G(4)=l(7)*exp(l(7)*(p(4)-(x(1)*x(7))/2+&
        (x(2)*x(6))/2-(x(3)*x(5))/2))
G(4)=G(4)-l(8)*exp(-l(8)*(p(4)-(x(1)*x(7))/2+&
        (x(2)*x(6))/2-(x(3)*x(5))/2))
G(4)=G(4)-(T-s)*H(4)

G(5)=l(9)*exp(l(9)*(p(5)-(x(1)*x(6))/2+&
        (x(3)*x(4))/2-(x(2)*x(7))/2))
G(5)=G(5)-l(10)*exp(-l(10)*(p(5)-(x(1)*x(6))/2+&
        (x(3)*x(4))/2-(x(2)*x(7))/2))
G(5)=G(5)-(T-s)*H(5)

G(6)=l(11)*exp(l(11)*(p(6)+(x(1)*x(5))/2-&
        (x(2)*x(4))/2-(x(3)*x(7))/2))
G(6)=G(6)-l(12)*exp(-l(12)*(p(6)+(x(1)*x(5))/2-&
        (x(2)*x(4))/2-(x(3)*x(7))/2))
G(6)=G(6)-(T-s)*H(6)

G(7)=l(13)*exp(l(13)*(p(7)+(x(1)*x(4))/2+&
        (x(2)*x(5))/2+(x(3)*x(6))/2))
G(7)=G(7)-l(14)*exp(-l(14)*(p(7)+(x(1)*x(4))/2+&
        (x(2)*x(5))/2+(x(3)*x(6))/2))
G(7)=G(7)-(T-s)*H(7)
end subroutine auxintegrand2

! ======== EULER ANGLES TO QUATERNION ========
subroutine eul2quat(eul,q)
real, intent(in):: eul(3)
real, intent(out):: q(4)
```

```fortran
real:: cy, sy, cp, sp, cr, sr
cy = cos(eul(1)*.5);
sy = sin(eul(1)*.5);
cp = cos(eul(2)*.5);
sp = sin(eul(2)*.5);
cr = cos(eul(3)*.5);
sr = sin(eul(3)*.5);
q(1) = cy * cp * sr - sy * sp * cr
q(2) = sy * cp * sr + cy * sp * cr
q(3) = sy * cp * cr - cy * sp * sr
q(4) = cy * cp * cr + sy * sp * sr
end subroutine eul2quat

! ======== QUATERNION TO EULER ANGLES ========
subroutine quat2eul(q,eul)
real, intent(in):: q(4)
real, intent(out):: eul(3)
eul(1) = atan2(2.*(q(4)*q(3)+q(1)*q(2)), &
        1.-2.*(q(2)**2.+q(3)**2.))
eul(2) = asin(2.*(q(4)*q(2)-q(3)*q(1)))
eul(3) = atan2(2.*(q(4)*q(1)+q(2)*q(3)), &
        1.-2.*(q(1)**2.+q(2)**2.))
end subroutine quat2eul

! ======== INDEX FUNCTION ========
subroutine indexfunction(s, z, L)
use initialization
implicit none
real:: s, z(m, nn), L(nn)
integer:: k
k=int((s-T_0)*m/(T_1-T_0))+1
L=z(k, :)
end subroutine indexfunction

! ======== OPTIMAL CONTROL ========
subroutine optimalcontrol(prec, mask, state, cost)
use initialization
implicit none
external integrand
integer:: i, j, k, mask(N)
real:: deriv(N), midpoint(N), r(nn), y(m, nn)
real:: convergence, prec, state(m+1, N), hh
real:: cost_prev, cost
real:: H(m, N), y_0, G_aux(m, N), G_auxx(m, N)
real:: G(m, N), integ(N),norm, aux(m, N), tam, eta, F

open(11,file='results_converg', &
status='replace',action='write')
```

```

hh = (T_1-T_0)/m
y_0 = 1.
y = 1.
eta = 1.e-01;

! // Optimal control algorithm ========
do
! Convergence criteria
convergence=0.
do i=1, m
        deriv=(state(i+1, :)-state(i, :))/hh
        midpoint=0.5*(state(i, :)+state(i+1, :))
        call &
        constraints(T_0+hh*(i-0.5), midpoint, deriv, r)
        do k=1, nn
                if (convergence<abs(r(k)*y(i, k))) then
                        convergence=abs(r(k)*y(i, k))
                endif
        end do
end do
print*, 'convergence␣', convergence
write(11,*) convergence
if (convergence<prec) exit
! Initialization
cost_prev=1.e+10
! Iterative process
do
        cost=0.
        integ=0.
        do i=1, m
                midpoint=0.5*(state(i,:)+state(i+1,:))
                deriv=(state(i+1,:)-state(i,:))/hh
                call auxintegrand1(T_0+hh*(i-0.5), &
                midpoint,deriv,y,H(i,:))
                call integrand(T_0+hh*(i-0.5), &
                midpoint,deriv,y,F)
                cost=cost+hh*F
                call auxintegrand2(T_0+hh*(i-0.5), &
                midpoint,deriv,y,T_1,G_aux(i,:))
                call auxintegrand2(T_0+hh*(i-0.5), &
                midpoint,deriv,y,T_0,G_auxx(i,:))
                do j=1, N
                        if (mask(j).eq.1) then
                                G(i,j)=G_aux(i,j)
                                integ(j)=integ(j)+hh*G(i,j)
                        else
                                G(i,j)=G_auxx(i,j)
                        end if
                end do
        end do
        cost_prev=cost
        norm=0.
        do i=1, m
                aux(i,:)=0.
                do k=1, i
                        do j=1, N
                                if (mask(j).eq.1) then
                                        aux(i,j)=&
                                        aux(i,j)+H(k,j)
                                else
                                        aux(i,:)=&
                                        aux(i,:)-hh*G(k,:)
                                end if
                        end do
                end do
                do j=1, N
                        if (mask(j).eq.1) then
                                aux(i,j)=&
                                aux(i,j)*hh*(hh*i-(T_1-T_0))
                        end if
                end do
                do k=i+1, m
                        do j=1, N
                                if (mask(j).eq.1) then
                                        aux(i,j)=&
                                        aux(i,j)+hh*G(k,j)
                                else
                                        aux(i,:)=aux(i,:)-&
                                        hh**2*i*H(k,:)
                                end if
                        end do
                end do
                aux(i,:)=aux(i,:)+&
                (hh*i-(T_1-T_0))*integ/(T_1-T_0)
                norm=norm+dot_product(aux(i,:), aux(i,:))
        end do
        tam=eta
        do
                if (abs(tam)<1.e-05) exit
                state(2:m+1,:)=state(2:m+1,:)+tam*aux
                cost=0.
                do i=1, m
                        midpoint=&
                        0.5*(state(i,:)+state(i+1,:))
                        deriv=(state(i+1,:)-state(i,:))/hh
                        call integrand(T_0+hh*(i-0.5), &
                        midpoint,deriv,y,F)
                        cost=cost+hh*F
                end do
                if (cost-cost_prev<0.) exit
                state(2:m+1,:)=state(2:m+1,:)-tam*aux
                tam=(0.5*norm*tam**2)/&
                (cost-cost_prev+norm*tam)
        end do
        if (abs(tam)<prec) exit
end do
! Actualizacion de multiplicadores
do i=1, m
        deriv=(state(i+1,:)-state(i,:))/hh
        midpoint=0.5*(state(i,:)+state(i+1,:))
        call constraints(T_0+hh*(i-0.5),midpoint,deriv,r)
        do k=1, nn
                y(i,k)=y(i,k)*exp(y(i, k)*r(k))
        end do
end do
if (y_0/maxval(y)<1.e-05) exit
end do
! ======== Optimal control algorithm //
close(12)
end subroutine optimalcontrol
```

A 3 Verification on the dynamic system

```
! ============================================================
! test_dynamics.f95
!
! This program simulates a rotating satellite actuated
! by a given control.
!
! It integrates the non-linear system
!       Jw' = - w x Jw + u
!       q' = 1/2 * Omega(w) * q
! Using numerical methods (Runge-Kutta 4 for ODEs)
!
! INPUT
! - "input_file": input file used by "satellites_nonlinear"
!       or "satellites_linear"
! It also takes the output from "satellites_nonlinear"
!       or "satellites_linear" as an input:
! - "results_control"
! - "results_angveloc"
! - "results_eulerang"
! - "results_quaternion"
! - "results_time"
! - "results_misc"
!
! OUTPUT
! - "real_angveloc": Angular velocity as simulated
! - "real_eulerang": Euler angles as simulated
! - "real_quaternion": Quaternion as simulated
!
! The convention for quaternion and
! Euler angles is as follows:
! q = (epsilon,eta)
! eul = (phi,theta,rho) = (yaw,pitch,roll)
! ============================================================

! ======== INITIALIZATION ========
module initialization
implicit none
integer:: m
real:: j1, j2, j3, t_0, t_1
real, allocatable:: control(:,:)
end module initialization

! ======== MAIN PROGRAM ========
program main
use initialization
implicit none
interface
        subroutine dynamics(n,x,t,dx)
        integer, intent(in):: n
        real, intent(in):: x(n), t
        real, intent(out):: dx(n)
        integer:: k
        end subroutine dynamics
end interface
integer:: i
real:: state_0(7), eulang(3)
real, allocatable:: time(:)
real, allocatable:: state_calcul(:,:), state_real(:,:)

! Open input file
open (11,file='input_file',status='old')
! Load input parameters from file
read(11,*)
read(11,*)
read(11,*) j1, j2, j3
read(11,*)
read(11,*)
read(11,*)
read(11,*)
read(11,*)
read(11,*)
read(11,*) m
close(11)

allocate (control(m,3), time(m+1))
allocate (state_calcul(m+1,7), state_real(m+1,7))

open (11,file='results_control',status = 'old')
do i = 1,m
        read(11,*) control(i,1), control(i,2), control(i,3)
end do
close(11)

open (11, file = 'results_time', status = 'old')
do i = 1,m+1
        read(11,*) time(i)
end do
close(11)

open (11, file = 'results_angveloc', status = 'old')
do i = 1,m
        read(11,*) &
        state_calcul(i,1), &
        state_calcul(i,2), &
        state_calcul(i,3)
end do
close(11)

open (11, file = 'results_quaternion', status = 'old')
do i = 1,m
        read(11,*) &
        state_calcul(i,4), &
        state_calcul(i,5), &
        state_calcul(i,6), &
        state_calcul(i,7)
end do
close(11)

t_0 = time(1)
t_1 = time(m+1)
state_0 = state_calcul(1,:)

call rk4(dynamics,7,state_0,t_0,t_1,m,state_real,time)
```

```

open(11, file='real_angveloc', &
status='replace',action='write')
do i = 1,m+1
        write(11,*) state_real(i,1:3)
end do
close(11)

open(11, file='real_eulerang', &
status='replace',action='write')
do i = 1,m+1
        call quat2eul(state_real(i,4:7),eulang)
        write(11,*) eulang
end do
close(11)

open(11, file='real_quaternion', &
status='replace',action='write')
do i = 1,m+1
        write(11,*) state_real(i,4:7)
end do
close(11)

end program main

! ======== QUATERNION TO EULER ANGLES ========
subroutine quat2eul(q,eul)
real, intent(in):: q(4)
real, intent(out):: eul(3)
eul(1) = atan2(2.*(q(4)*q(3)+q(1)*q(2)), &
        1.-2.*(q(2)**2.+q(3)**2.))
eul(2) = asin(2.*(q(4)*q(2)-q(3)*q(1)))
eul(3) = atan2(2.*(q(4)*q(1)+q(2)*q(3)), &
        1.-2.*(q(1)**2.+q(2)**2.))
end subroutine quat2eul

! ======== DYNAMICS ========
subroutine dynamics(n,x,t,dx)
use initialization
implicit none
integer, intent(in):: n
real, intent(in):: x(n), t
real, intent(out):: dx(n)
integer:: k

k = int((t-t_0)*m/(t_1-t_0))+1

dx(1) = x(2)*x(3)*(j2-j3)/j1 + control(k,1)/j1
```

```fortran
dx(2) = x(1)*x(3)*(j3-j1)/j2 + control(k,2)/j2
dx(3) = x(1)*x(2)*(j1-j2)/j3 + control(k,3)/j3
dx(4) = .5*(+ x(1)*x(7) - x(2)*x(6) + x(3)*x(5))
dx(5) = .5*(+ x(1)*x(6) - x(3)*x(4) + x(2)*x(7))
dx(6) = .5*(+ x(2)*x(4) - x(1)*x(5) + x(3)*x(7))
dx(7) = .5*(- x(1)*x(4) - x(2)*x(5) - x(3)*x(6))
end subroutine dynamics

! ======== 4TH ORDER RUNGE-KUTTA METHOD ========
subroutine rk4(func,n,x0,t0,t1,m,x,time)
implicit none
integer, intent(in):: n, m
real, intent(in):: x0(n), t0, t1
real, intent(out):: x(m+1,n), time(m+1)
real:: h, a1(N), a2(N), a3(N), a4(N)
integer:: i, k
h = (t1-t0)/m

do i = 1,n
        x(1,i) = x0(i)
end do

do k = 1,m
        time(k) = t0 + (k-1)*h
        call func(n,x(k,:),time(k),a1)
        a1 = h*a1
        call func(n,x(k,:)+.5*a1,time(k)+h/2.,a2)
        a2 = h*a2
        call func(n,x(k,:)+.5*a2,time(k)+h/2.,a3)
        a3 = h*a3
        call func(n,x(k,:)+a3,time(k)+h,a4)
        a4 = h*a4
        do i = 1,n
                x(k+1,i) = x(k,i) + &
        (a1(i) + 2*a2(i)  +2*a3(i) + a4(i))/6.
        end do
end do
time(m+1) = t1
end subroutine rk4
```

BIBLIOGRAPHY

[1] Victorio Úbeda, Daire Sherwin, Joseph Thompson, David Mckeown, and William O'Connor. Modelling EIRSAT-1 dynamics and actuation to validate and test a novel attitude control system. In *2nd Symposium on Space Educational Activities*, SSEA–2018–70, 2018.

[2] Daire Sherwin, Joseph Thompson, David Mckeown, William O'Connor, and Victorio Úbeda. Wave-based attitude control of EIRSAT-1, 2U cubesat. In *2nd Symposium on Space Educational Activities*, SSEA–2018–93, 2018.

[3] Rafal Wiśniewski. *Satellite Attitude Control Using Only Electromagnetic Actuation.* PhD thesis, Aalborg University, 1996.

[4] Zdenko Tudor. *Design and Implementation of Attitude Control for 3-axes Magnetic Coil Stabilization of a Spacecraft.* PhD thesis, Norges teknisk-naturvitenskaplige universitet, 2011.

[5] Pooya Sekhavat, Hui Yan, Andrew Fleming, I Michael Ross, and Kyle T Alfriend. Closed-Loop Time-Optimal Attitude Maneuvering of Magnetically Actuated Spacecraft. *The Journal of the Astronautical Sciences*, 58(1):81–97, 2011.

[6] Yaguang Yang. Spacecraft attitude determination and control: Quaternion based method. *Annual Reviews in Control*, 36(2):198–219, 2012.

[7] Gaute Bråthen. *Design of Attitude Control System of a Double CubeSat.* PhD thesis, Norges teknisk-naturvitenskaplige universitet, 2013.

[8] Andrea Bellanca, Vincenzo Pesce, Paolo Lunghi, and Michèle Lavagna. Wave based control analysis and experimental validation for reliable space transportation applications. In *10th International ESA Conference on Guidance, Navigation and Control Systems*, pages 1–16, 2017.

[9] M. Lovera and A. Varga. Optimal discrete-time magnetic attitude control of satellites. In *16th Trienial World Congress*, volume 38, pages 157–162. IFAC, 2005.

[10] Alex Walker, Phil Putman, and Kelly Cohen. Fuzzy Logic Attitude Control of a Magnetically Actuated CubeSat. In *AIAA Infotech@Aerospace (I@A) Conference*, pages 1–8, Reston, Virginia, aug 2013. American Institute of Aeronautics and Astronautics.

[11] Manuela Battipede, Piero Gili, and Luca Massotti. Neural and LQR Optimal Attitude Control of a Flexible Micro-Satellite. In *AIAA Guidance, Navigation, and*

Control Conference and Exhibit, Reston, Virigina, aug 2003. American Institute of Aeronautics and Astronautics.

[12] P. Crouch. Spacecraft attitude control and stabilization: Applications of geometric control theory to rigid body models. *IEEE Transactions on Automatic Control*, 29(4):321–331, apr 1984.

[13] Hariharan Krishnan, N. H. McClamroch, and Mahmut Reyhanoglu. Attitude stabilization of a rigid spacecraft using two momentum wheel actuators. *Journal of Guidance, Control, and Dynamics*, 18(2):256–263, mar 1995.

[14] Rafal Wisniewski and FL Markley. Optimal magnetic attitude control. In *14th World Congress of IFAC*, pages 7991–7996, 1999.

[15] Marco Lovera, Eliana De Marchi, and Sergio Bittanti. Periodic attitude control techniques for small satellites with magnetic actuators. *IEEE Transactions on Control Systems Technology*, 10(1):90–95, 2002.

[16] Enrico Silani and Marco Lovera. Magnetic spacecraft attitude control: a survey and some new results. *Control Engineering Practice*, 13(3):357–371, mar 2005.

[17] Pablo Pedregal. Direct Numerical Algorithm for Constrained Variational Problems. *Numerical Functional Analysis and Optimization*, 38(4):486–506, apr 2017.

[18] Pablo Pedregal. *Optimization and Approximation*, volume 108 of *UNITEXT*. Springer International Publishing, Cham, 2017.

[19] Pablo Pedregal. A direct algorithm for constrained variational problems in several dimensions. *Computers & Mathematics with Applications*, 75(1):105–121, jan 2018.

[20] Marcel J. Sidi. *Spacecraft Dynamics and Control: A Practical Engineering Approach.* Cambridge University Press, 2000.

[21] Frederik Bajersvej, Dk-Aalborg Øst, and Rafael Wisniewski. *Attitude control system for AAU cubesat.* PhD thesis, Aalborg University, 2002.

[22] Matthew Nehrenz and Matt Sorgenfrei. On the Development of Spacecraft Operating Modes for a Deep Space CubeSat. In *AIAA SPACE 2015 Conference and Exposition*, Reston, Virginia, aug 2015. American Institute of Aeronautics and Astronautics.

[23] Ashish Tewari. *Automatic Control of Atmospheric and Space Flight Vehicles*, volume 20. Birkhäuser Boston, Boston, MA, nov 2011.

[24] A. Craig Stickler and K.T. Alfriend. Elementary Magnetic Attitude Control System. *Journal of Spacecraft and Rockets*, 13(5):282–287, may 1976.

[25] Victoria Coverstone-Carroll. Detumbling and reorienting underactuated rigid spacecraft. *Journal of Guidance, Control, and Dynamics*, 19(3):708–710, may 1996.

[26] Thomas Bak, Rafal Wisniewski, and Morgens Blanke. Autonomous attitude determination and control system for the Orsted satellite. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 2, pages 173–186. IEEE, 1996.

[27] S.P. Bhat and A.S. Dham. Controllability of spacecraft attitude under magnetic actuation. In *42nd IEEE International Conference on Decision and Control*, volume 3, pages 2383–2388. IEEE, 2003.

[28] Cynthia Duda. CubeSat Technical Aspects. In *55th International Astronautical Congress of the International Astronautical Federation, the International Academy of Astronautics, and the International Institute of Space Law*, Reston, Virigina, oct 2004. American Institute of Aeronautics and Astronautics.

[29] Ali Aydinlioglu and Marco Hammer. COMPASS-1 pico satellite: magnetic coils for attitude control. In *Proceedings of 2nd International Conference on Recent Advances in Space Technologies, 2005. RAST 2005.*, volume 2005, pages 90–93. IEEE, 2005.

[30] Junquan Li, Mark Post, Thomas Wright, and Regina Lee. Design of Attitude Control Systems for CubeSat-Class Nanosatellite. *Journal of Control Science and Engineering*, 2013:1–15, 2013.

[31] J.R. Wertz. *Spacecraft Attitude Determination and Control.* Springer Science & Business Media, Dordrecht, 1978.

[32] E. T. Whittaker and Sir William McCrea. *A Treatise on the Analytical Dynamics of Particles and Rigid Bodies.* Cambridge University Press, Cambridge, 1988.

[33] Yaguang Yang. Analytic LQR Design for Spacecraft Control System Based on Quaternion Model. *Journal of Aerospace Engineering*, 25(3):448–453, jul 2012.

[34] Malcolm Shuster. A Survey of Attitude Representations. *The Journal of the Astronautical Sciences*, 41:436–517, 1993.

[35] Yaguang Yang. Quaternion based model for momentum biased nadir pointing spacecraft. *Aerospace Science and Technology*, 14(3):199–202, 2010.

[36] Lawrence Evans. *An introduction to mathematical optimal control theory.* University of California, Berkeley, 1983.

[37] Dirk Aeyels. Stabilization of a class of nonlinear systems by a smooth feedback control. *Systems & Control Letters*, 5(5):289–294, apr 1985.

[38] Eduardo D. Sontag and Héctor J. Sussmann. Further comments on the stabilizability of the angular velocity of a rigid body. *Systems & Control Letters*, 12(3):213–217, apr 1989.

[39] Eduardo D. Sontag. *Mathematical control theory.* Springer Science & Business Media, 1990.

[40] Abraham Ralph and Marsden Jerrold E. *Foundation of Mechanics.* AMS Chelsea Publishing, 2008.

[41] Pablo Pedregal. *A variational approach to optimal control for ODEs.* Graduate Studies in Mathematics, AMS, (accepted).

[42] Giulio Avanzini and Fabrizio Giulietti. Magnetic Detumbling of a Rigid Spacecraft. *Journal of Guidance, Control, and Dynamics*, 35(4):1326–1334, jul 2012.

[43] Alberto Calloni, Andrea Corti, Andrea Zanchettin, and Marco Lovera. Robust attitude control of spacecraft with magnetic actuators. *Proc. of the American Control Conf.*, (June):750–755, 2012.

[44] James Biggs. Under-actuated attitude control of Cubesats. In *5th nterplanetary Cubesat Workshop*. Politecnico di Milano, 2016.

[45] James Clary and Kwang Lee. Stochastic theory of minimal realization. In *1976 IEEE Conference on Decision and Control including the 15th Symposium on Adaptive Processes*, pages 1268–1275. IEEE, dec 1976.

[46] Zvi Artstein. Discrete and Continuous Bang-Bang and Facial Spaces Or: Look for the Extreme Points. *SIAM Review*, 22(2):172–185, apr 1980.

[47] Irmgard Flugge-Lotz. *Discontinuous Automatic Control.* Princeton University Press, 1953.

[48] Henry Hermes and Joseph P LaSalle. *Functional analysis and time optimal control. Mathematics in Science and Engineering.* New York-London: Academic Press, 1969.

[49] Igor Kluvánek and Greg Knowles. *Vector measures and control systems. North-Holland Mathematics Studies.* New York: North-Holland Publishing Co, 1976.

[50] L. M. Sonneborn and F. S. Van Vleck. The Bang-Bang Principle for Linear Control Systems. *Journal of the Society for Industrial and Applied Mathematics Series A Control*, 2(2):151–159, jan 1964.

[51] M. de la Sen and A. Ibeas. Stability Results for Switched Linear Systems with Constant Discrete Delays. *Mathematical Problems in Engineering*, 2008(January):1–28, 2008.

[52] Edoardo Serpelloni, Manfredi Maggiore, and Christopher Damaren. Bang bang hybrid stabilization of perturbed double integrators. In *53rd IEEE Conference on Decision and Control*, volume 2015-Febru, pages 771–776. IEEE, dec 2014.

[53] Robin Temporelli, Philippe Micheau, and Maxime Boisvert. Control of an electromechanical clutch actuator by a parallel Adaptive Feedforward and Bang-Bang controller: Simulation and Experimental results. *IFAC-PapersOnLine*, 50(1):4787–4793, jul 2017.