Politecnico di Torino

Politecnico di Torino

Master Degree course in Aerospace Engineering

A.Y. 2025/2026

Graduation Session March/April 2026

Evaluation of Model Predictive Control Strategies for Satellite Flying Formations in Generic Circular Orbits

Supervisors:

Mauro Mancini

Fabio Faliero

Candidate:

Marco Francesco Sgura

Abstract

This thesis investigates the challenge of autonomously guiding several satellites into a specific Generic Circular Orbit (GCO) configuration. As the demand for global connectivity and Earth observation scales, traditional ground-based station-keeping becomes computationally prohibitive and operationally inefficient. For this reason, a distributed controller architecture is considered and analysed. The research focuses on the transition from a generic injection point to a specific, stable circular orbit. The dynamics are modelled using the linearised Clohessy-Wiltshire (CW) relative motion equations to represent the state of each satellite relative to a virtual leader or a specific orbital slot. The proposed MPC framework addresses the guidance problem by solving a constrained optimisation problem at each time step. Key features of the controller include intrinsic fuel optimisation consumption through norm objective functions, and the inclusion of actuator limits, to account for the maximum thrust capabilities of typical small-satellite propulsion systems. Simulations demonstrate that the MPC approach effectively navigates the local orbital environment while ensuring all agents reach their assigned phases. A series of comparisons with standard passivity-based and boundary layer controllers is performed, as well as a simplified test under disturbances. Drawbacks and issues with the implementation are also discussed, while several recommendations for further research on the topic are given.

Table of Contents

List of Figures

Glossary

ADMM	Alternating-Direction Method of Multipliers
CARE	Continuous Algebraic Riccati Equations
CRO	Circular Relative Orbit
CW	Clohessy-Wiltshire
DARE	Discrete Algebraic Riccati Equations
ECI	Earth-Centred Inertial
EO-1	Earth Observing-1
FF	Flying Formation
GCO	Generic Circular Orbit
GPS	Global Positioning System
GRACE	Gravity Recovery and Climate Experiment
GRACE-C	Gravity Recovery and Climate Experiment - Continuity
GRACE-FO	Gravity Recovery and Climate Experiment - Follow On
IPM	Interior-point method
LISA	Laser Interferometer Space Antenna
LQR	Linear Quadratic Regulator
LVLH	Local-Vertical Local-Horizontal
MIMO	multiple-input multiple-output

MPC	Model Predictive Controller
MPC	Model Predictive Control
NLP	Non-Linear Programming
OSQP	Operator-Splitting Quadratic Program
PH	Port-Hamiltonian
PID	Proportional-Integral-Derivative
QP	Quadratic Program
QP	Quadratic Programming
RK	Runge-Kutta
SFF	Satellite Flying Formation
SISO	single-input single-output
TH	Tschauner-Hempel

Chapter 1

Introduction

In the aerospace industry, a Satellite Flying Formation (SFF), often shortened to Flying Formation (FF), refers to a group of spacecraft whose motion is collectively governed by a shared control strategy [1], generally flying in close proximity to each other. While a universally agreed-upon definition does not exist, what's most important is to distinguish FFs from spacecraft constellations, which are instead large-scale satellite formations that may cover the entire globe, and in which any of the agents may use only their own current position and velocity states, like in GPS satellites[1]. Usually, but not always, FFs involve at least one spacecraft whose control law tracks another member's position as a desired state [1]. The type of formation varies depending on the geometrical arrangement of the single units. There are two main types of formations [2]:

- **trailing formations** — all spacecraft orbit the Earth on the same path, covering the same area on the ground at different time intervals. Notable examples are the Gravity Recovery and Climate Experiment (GRACE) (launched in 2002), GRACE-FO (launched in 2018) and GRACE-C (planned for 2028 [3]) and Landsat 7 with Earth Observing-1 (EO-1).
- **cluster formations** — the spacecraft move in a cluster, possibly following a geometrical shape arrangement while doing so. The most notable example is the planned Laser Interferometer Space Antenna (LISA) mission for the detection of gravitational waves [4][5].

Since the early 2000s, Satellite Flying Formations have gained increased interest due to the attractive ability to perform the typical tasks of single, bigger, heavier satellites with the following added benefits:

[1] In GPS satellites the relative position information is kept, however only the absolute position and velocity vectors of the satellite are used to follow the correct flight path [1].

- lower design, production and launch costs;
- reduced total mission failure risk by running several rocket launches;
- higher flexibility and adaptability;
- redundancy through numbers increases mission reliability;
- ease of satellite replacement when obsolete.

Despite this, there are numerous challenges when designing a FF mission:

- use of robust control laws
- extremely high risk of collision
- partial or total control loss should be avoided at all costs
- disturbances may make extremely precise satellite collocation difficult to accomplish

1.1 Scenario

Let's consider a fictitious satellite orbiting Earth, called `chief`. We call it fictitious because it effectively does not exist, as it will be used merely as a point of reference for all our other considerations. The `chief`'s movement around the Earth agrees with the following assumptions:

1. perfectly circular orbit;
2. no disturbances.

Let's now consider some real satellites. Their number is irrelevant for now, so let's just pick one for the ease of writing, called `chaser_1`. Both the `chief` and `chaser_1` are initially on the same orbit, thus moving at the same speed (given the circular orbit) and occupying the same location in space. At time $t = t_0$, `chaser_1` is tasked with reaching a very specific orbit, one that from the point of view of the `chief` forms a perfect circle of radius R, is centred on the `chief` itself and is tilted by 30 degrees with respect to the north-south tangent to Earth's surface. This orbit is called a Generic Circular Orbit (GCO) or Circular Relative Orbit (CRO). As we will show later, this kind of local orbit is very special, because once `chaser_1` manages to reach it, it doesn't require any further thruster action to stay on it[2].

[2]Disturbances like drag, solar pressure or the oblateness of the Earth can still deviate the natural path of `chaser_1` from the CRO.

If we then add more real satellites, named `chaser_2`, `chaser_3` and so on, their task is also that of reaching the CRO (one at a time), but they are also tasked to distance themselves as much as possible from the satellites already in the formation. This means that all satellites in the formation will be at the vertex of an N_{sat}-sided regular polygon, with N_{sat} being the number of spacecraft in the CRO. Figure 1.1 shows an example of CRO with three satellites on it.

Number of spacecraft	Shape
1	-
2	line
3	triangle
4	square
⋮	⋮

Table 1.1: Geometric shape of the FFs depending on the amount of satellites.

To sum it all up, the scope of the analysed mission is to design a control law (essentially, an algorithm) which will guide any of the N_{sat} `chaser` spacecraft into a CRO while also keeping an N_{sat}-sided FF.

1.2 Proposed solution

To accomplish the aforementioned goals, a control law must be designed and adopted in each of the N_{sat} satellites. For this purpose, Model Predictive Control (MPC) was chosen, mainly because of its flexibility. An MPC controller can solve our guidance problem and find an optimal way of doing so at the same time. Furthermore, we can add some constraints to make sure that the controller can see the limitations coming from the thrusters, in the form of saturations. To facilitate this task, a gradually-inflating reference for the CRO is used, featuring a simple linearly expanding radius.

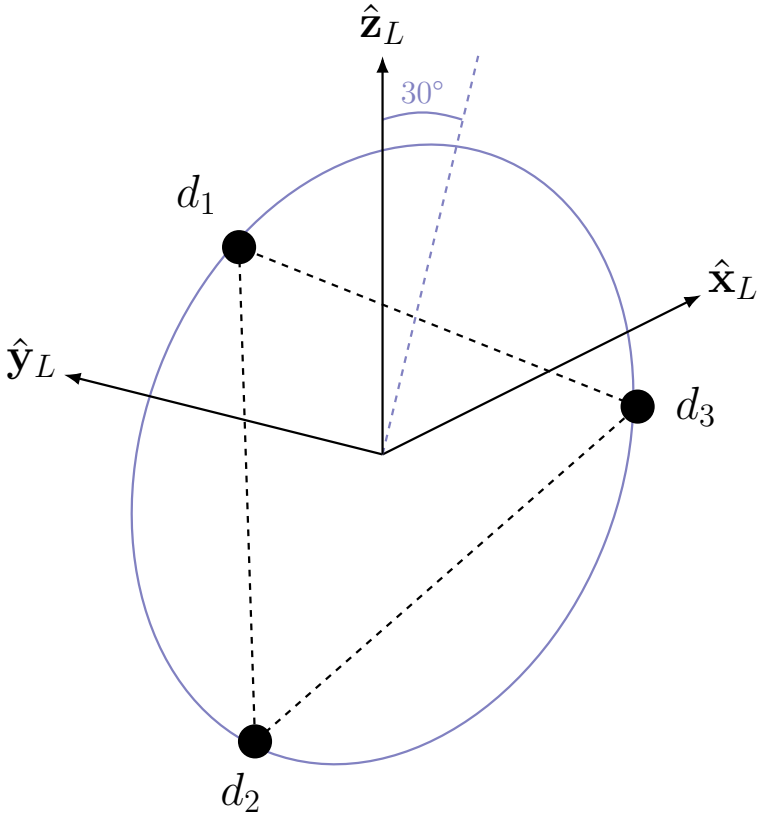


Figure 1.1: A Circular Relative Orbit formation of $N_{\text{sat}} = 3$ satellites.

1.3 Methodology and implementation

Tackling the problem at hand involves the usage of some assumptions and several theoretical foundations — models and formulas used together to reach the desired tasks.

- the motion of all satellites through space is modelled using the Clohessy-Wiltshire (CW) equations. These are a linearised, relative version of the standard equations of orbital motion, and prove to be very handy for this kind of scenario since we don't need to integrate the full state vector of the `chief` but only that of each `chaser`. All info about the integrators can be found in Chapter 4.

- it is assumed that all satellites move on roughly circular orbits at all times. The reference orbit is also assumed perfectly circular and equatorial. This

eases the analysis which may be extended to more intricate cases by simply swapping out the CW model with a different one[3].

- the controller of each `chaser` is an MPC, reading the states of all other satellites and formulating the optimal path to reach the destination. The main drawback of any MPC is the relatively long time it takes to generate an optimal solution. It is thus mandatory to make sure that the calculation time of each solution stays within the assigned window of operation. For example, if the MPC is tasked with regenerating a new solution every 0.01 seconds, we need to limit the calculation time to $\ll 0.01$ seconds.

1.4 Document organisation

The document is split into several parts, each representing a phase in the study as a whole. Chapter 1, the current one, provides basic information about the scenario and how the problem associated with it was tackled. Chapter 2 provides details for the mathematical formulation of the problem, describes the various reference frames that have been used, and shows derivations for the fundamental Clohessy-Wiltshire relations that describe relative orbital motion as well as the equations describing the reference orbit the satellite aims for. Chapter 3 describes the design of the proposed optimal controller and how the control objectives have been achieved by summarizing them into an appropriate objective function and constraints. It also shows how the presented optimal control has been adapted to a more conventional structure which aligns with the Quadratic Programming (QP) problem to be solved. Chapter 4 goes over the results of the simulations, with thorough comments and comparison with similar academic results. Lastly, Chapter 5 contains a summary of the work presented in this thesis, as well as suggestions and potential improvements for any future work continuing from here onwards. As a bonus, some more technical information about optimisation problems and their implementation is given in Appendices A and B respectively.

Note — all important mathematical formulas are framed in a box.

[3]For instance, the Tschauner-Hempel (TH) equations of relative motion may be used when analysing elliptical orbits.

Chapter 2

Dynamical model

2.1 Reference frames

Usually, when treating the topic of orbital mechanics, several reference frames may be encountered depending on the type of problem being analysed. In this work, two main sets are mentioned.

2.1.1 Earth-Centred Inertial

The **Earth-Centred Inertial (ECI)** frame is non-moving with respect to the stars, while the Earth rotates with respect to it. It is defined by the following axes:

- **x** pointing towards the constellation of Aries;
- **y** obtained as the third vector the right-handed triad;
- **z** perpendicular to the equatorial plane.

This triad can be considered inertial if one neglects the precession of the equinoxes. It is typically used for large-scale manoeuvres, generally performed in open-loop.

While useful for initial derivations, the ECI frame is not very practical for a close-range manoeuvre, such as a rendezvous mission or a formation-keeping scenario. For this reason, another reference frame may be defined to describe the motion of a chaser spacecraft relative to a reference point moving in orbit around the Earth. This is the Local-Vertical Local-Horizontal (LVLH) frame, which is described below.

2.1.2 Local-Vertical Local-Horizontal

The **Local-Vertical Local-Horizontal (LVLH)** frame is centered on a predetermined reference point moving along a circular orbit around the Earth, and is defined by the following axes:

- **x** pointing radially outwards from the origin with respect to Earth's centre;
- **y** obtained as the third vector of the right-handed triad;
- **z** aligned with the angular momentum of the satellite's orbit, i.e. normal to the orbital plane.

The scheme of the ECI and LVLH frames is shown in Figure 2.1. It is important to note that the **y** axis is not always aligned with the velocity vector of the reference point, since its orbit may be elliptical. The previous statement is instead true if the orbit is perfectly circular, a condition that is never truly reached in real scenarios. Nonetheless, the approximation may be deemed reasonable enough when the eccentricity of the parent orbit is very small.

Lastly, from a mathematical point of view, the origin of the LVLH frame can be treated either as a target spacecraft or as just an imaginary point moving along a specified orbit.

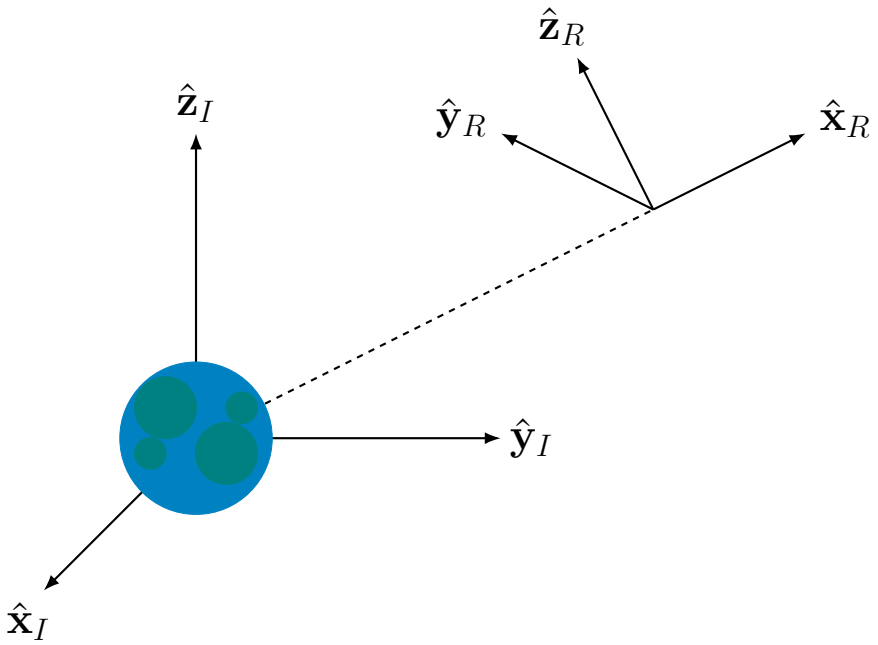


Figure 2.1: Earth-Centred Inertial (ECI) and Local-Vertical Local-Horizontal (LVLH) reference frames.

2.2 Clohessy-Wiltshire equations

The Clohessy-Wiltshire (CW) equations are a set of linear, time-invariant differential equations which describe the relative orbital motion of a chaser spacecraft with respect to a target spacecraft in LVLH coordinates. The equations are derived from the Newtonian equations for orbital motion, and they are particularly straightforward to implement in any orbital simulator. The first proper formulation was obtained by W. H. Clohessy and R. S. Wiltshire in 1960 for a hypothetical orbital rendezvous mission [6], effectively consolidating G. W. Hill's work on the orbital motion of the Moon around the Earth, dating back to 1878 [7]. Clohessy-Wiltshire's model uses many simplifications, but it is accurate enough for preliminary design of the relative motion control strategy [8].

The extensive derivation of the relative dynamics equations can be found in [8], while an outline of the derivation is provided below. The two-body problem is depicted as:

$$\ddot{\mathbf{r}} = -\mu \frac{\mathbf{r}}{r^3} + \frac{\mathbf{F}}{m} \tag{2.1}$$

These equations are expressed in the ECI frame. The target sees no external accelerations, giving

$$\ddot{\mathbf{r}}_t = -\mu \frac{\mathbf{r}_t}{r_t^3} \tag{2.2}$$

On the other hand, the chaser spacecraft is subject to actuator forces, resulting in

$$\ddot{\mathbf{r}}_c = -\mu \frac{\mathbf{r}_c}{r_c^3} + \frac{\mathbf{F}_c}{m_c} \tag{2.3}$$

Following the basics of relative motion, it is known that

$$\rho = \mathbf{r}_c - \mathbf{r}_t \quad \longrightarrow \quad \ddot{\rho} = \ddot{\mathbf{r}}_c - \ddot{\mathbf{r}}_t \tag{2.4}$$

By subtracting (2.2) from (2.3) and inserting (2.4) we obtain

$$\ddot{\rho} = -\mu \frac{\mathbf{r}_c}{r_c^3} + \mu \frac{\mathbf{r}_t}{r_t^3} + \frac{\mathbf{F}_c}{m_c} \tag{2.5}$$

We can rewrite (2.5) using a more compact notation as

$$\ddot{\rho} = \mathbf{f}_g(\mathbf{r}_c) - \mathbf{f}_g(\mathbf{r}_t) + \frac{\mathbf{F}_c}{m_c} \tag{2.6}$$

Next, it is possible to linearise by Taylor expansion around the target position, such that

$$\mathbf{f}_g(\mathbf{r}_c) - \mathbf{f}_g(\mathbf{r}_t) = \left. \frac{\mathrm{d}\mathbf{f}_g(\mathbf{r}_c)}{\mathrm{d}\mathbf{r}} \right|_{\mathbf{r}=\mathbf{r}_t} (\mathbf{r}_c - \mathbf{r}_t) \tag{2.7}$$

where

$$\frac{\mathrm{d}\mathbf{f}_g(\mathbf{r}_c)}{\mathrm{d}\mathbf{r}} = \begin{bmatrix} \dfrac{\partial \mathbf{f}_{g,1}(\mathbf{r}_c)}{\partial r_1} & \cdots & \dfrac{\partial \mathbf{f}_{g,1}(\mathbf{r}_c)}{\partial r_3} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial \mathbf{f}_{g,3}(\mathbf{r}_c)}{\partial r_1} & \cdots & \dfrac{\partial \mathbf{f}_{g,3}(\mathbf{r}_c)}{\partial r_3} \end{bmatrix} \tag{2.8}$$

It can be shown that the diagonal terms are

$$\left.\frac{\partial f_{g,i}(\mathbf{r}_c)}{\partial r_j}\right|_{i=j} = -\frac{\mu}{r^3}\left[1 - 3\frac{r_i^2}{r^2}\right] \tag{2.9}$$

while the remaining components are

$$\left.\frac{\partial f_{g,i}(\mathbf{r}_c)}{\partial r_j}\right|_{i\neq j} = -3\frac{\mu}{r^3}\frac{r_i r_j}{r^2} \tag{2.10}$$

It follows that (2.7) can be expressed as:

$$\mathbf{f}_g(\mathbf{r}_c) - \mathbf{f}_g(\mathbf{r}_t) = -\frac{\mu}{r_t^3}\begin{bmatrix} 1 - 3\dfrac{r_x^2}{r_t^2} & 3\dfrac{r_x r_y}{r_t^2} & 3\dfrac{r_x r_z}{r_t^2} \\ 3\dfrac{r_y r_x}{r_t^2} & 1 - 3\dfrac{r_y^2}{r_t^2} & 3\dfrac{r_y r_z}{r_t^2} \\ 3\dfrac{r_z r_x}{r_t^2} & 3\dfrac{r_z r_y}{r_t^2} & 1 - 3\dfrac{r_z^2}{r_t^2} \end{bmatrix}\boldsymbol{\rho} = -\frac{\mu}{r_t^3}\mathbf{M}\boldsymbol{\rho} \tag{2.11}$$

We can thus rewrite

$$\ddot{\boldsymbol{\rho}} = -\frac{\mu}{r_t^3}\mathbf{M}\boldsymbol{\rho} + \frac{\mathbf{F}_c}{m_c}. \tag{2.12}$$

It must be noted that (2.12) is still formulated in ECI coordinates.

To calculate the derivatives in LVLH frame, which is rotating with respect to the ECI frame, it is necessary to use the transport theorem. Specifically, the LVLH frame rotates with a rate of ω, i.e. the orbital angular frequency (in $^{\mathrm{rad}}/_{\mathrm{s}}$) of the target. The transport theorem is expressed as

$$\frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2} = \frac{\mathrm{d}^{*2}\mathbf{x}^*}{\mathrm{d}t^2} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \mathbf{x}^*) + 2\boldsymbol{\omega} \times \frac{\mathrm{d}\mathbf{x}^*}{\mathrm{d}t} + \frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} \times \mathbf{x}^* \tag{2.13}$$

where d^* denotes a derivative calculated in the moving reference frame, while $\mathbf{x}^*$ is any vector expressed in the same frame.

When expanding $\ddot{\boldsymbol{\rho}}$ using (2.13) we can rewrite (2.12) as

$$\frac{\mathrm{d}^{*2}\boldsymbol{\rho}^*}{\mathrm{d}t^2} + \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{\rho}^*) + 2\boldsymbol{\omega} \times \frac{\mathrm{d}\boldsymbol{\rho}^*}{\mathrm{d}t} + \frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} \times \boldsymbol{\rho}^* = -\frac{\mu}{r_t^3}\mathbf{M}\boldsymbol{\rho} + \frac{\mathbf{F}_c}{m_c} \tag{2.14}$$

Eq. (2.14) can be written in component-wise formulation by exploiting the following relations:

$$\mathbf{r}_t = \begin{bmatrix} r \\ 0 \\ 0 \end{bmatrix}, \quad ||\mathbf{r}_t|| = r, \quad \boldsymbol{\omega} = \begin{bmatrix} 0 \\ 0 \\ \omega \end{bmatrix} \tag{2.15}$$

$$\boldsymbol{\rho}^* = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\mathcal{F}_{LVLH}} \tag{2.16}$$

$$\boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{\rho}^*) = \boldsymbol{\omega} \times \begin{bmatrix} -\omega y \\ \omega x \\ 0 \end{bmatrix} = \begin{bmatrix} -\omega^2 x \\ -\omega^2 y \\ 0 \end{bmatrix} \tag{2.17}$$

$$2\boldsymbol{\omega} \times \frac{\mathrm{d}^*\boldsymbol{\rho}^*}{\mathrm{d}t} = \begin{bmatrix} -2\omega\dot{y} \\ 2\omega\dot{x} \\ 0 \end{bmatrix} \tag{2.18}$$

$$\frac{\mathrm{d}\boldsymbol{\omega}}{\mathrm{d}t} \times \boldsymbol{\rho}^* = \begin{bmatrix} -\dot{\omega} y \\ \dot{\omega} x \\ 0 \end{bmatrix} \tag{2.19}$$

$$-\frac{\mu}{r_t^3}\mathbf{M}\boldsymbol{\rho}^* = -\frac{\mu}{r^3}\begin{bmatrix} -2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\boldsymbol{\rho}^* = -\frac{\mu}{r^3}\begin{bmatrix} -2x \\ y \\ z \end{bmatrix} \tag{2.20}$$

Lastly, we note that ω is constant for circular orbits and returns

$$\omega = \sqrt{\frac{\mu}{r^2}} \quad \longrightarrow \quad \begin{array}{c} \omega^2 = \dfrac{\mu}{r^2} \\ \dot{\omega} = 0 \end{array} \tag{2.21}$$

which turns (2.19) nil and simplifies (2.20) as follows

$$-\frac{\mu}{r_t^3}\begin{bmatrix} -2x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2\omega^2 x \\ -\omega^2 y \\ -\omega^2 z \end{bmatrix} \tag{2.22}$$

The Clohessy-Wiltshire equations are therefore obtained by substituting (2.17), (2.18), (2.19) and (2.20) inside of (2.14) and unwrapping it, giving us

$$\boxed{\begin{aligned} \ddot{x} - 3\omega^2 x - 2\omega\dot{y} &= u_x \qquad (2.23a) \\ \ddot{y} + 2\omega\dot{x} &= u_y \qquad (2.23b) \\ \ddot{z} + \omega^2 z &= u_z \qquad (2.23c) \end{aligned}}$$

where $\mathbf{u}$ is the acceleration seen by the chaser spacecraft given the application of the external force $\mathbf{F}$.

2.2.1 State-space formulation

By defining the state and input vectors respectively as

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\rho} \\ \dot{\boldsymbol{\rho}} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \tag{2.24}$$

one can rewrite (2.23) using the state-space formulation, i.e.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{2.25}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} \tag{2.26}$$

where

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3\omega^2 & 0 & 0 & 0 & 2\omega & 0 \\ 0 & 0 & 0 & -2\omega & 0 & 0 \\ 0 & 0 & -\omega^2 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{0}_{3\times3} \\ \mathbf{I}_{3\times3} \end{bmatrix}, \quad \mathbf{C} = \mathbf{I}_{3\times3} \tag{2.27}$$

To use steady-state systems in numerical applications one also needs to discretize them such that

$$\mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k \tag{2.28}$$

$$\mathbf{y}_k = \mathbf{C}_d\mathbf{x}_k \tag{2.29}$$

where k is the current time step, while the d subscript indicates the discrete-time version of the associated continuous-time matrix. From this point onwards only the discrete-time model will be used; we will therefore omit the d subscript on all $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ matrices.

The discretisation of a steady-state system can be quickly performed using the `ss` and `c2d` commands from Matlab's *Control System Toolbox* suite:

```
1    lti_cont = ss(A, B, C, []);
2    DT = T/N; % time sample
3    lti_disc = c2d(lti_cont, DT);
4    A_d = lti_disc.A;
5    B_d = lti_disc.B;
6    C_d = lti_disc.C;
```

2.3 Motion in the LVLH frame

The relative motion of the chaser spacecraft with respect to the origin of the LVLH frame may be either uncontrolled or actively controlled. However, only a subset of trajectories can be maintained without external actuation. In this work, the trajectory of interest is a Circular Relative Orbit of the chaser about the origin of the LVLH frame. Under specific conditions, this motion represents a natural solution of the Clohessy-Wiltshire equations [9]. The equations describing this motion are reported below and briefly derived for completeness.

First, let us assume that the initial conditions of (2.23) satisfy

$$\dot{y}_0 + 2\omega x_0 = 0. \tag{2.30}$$

which ensures that the motion about the origin is bounded. The wanted trajectory is given by three oscillatory relations [9], which are

$$x = A_0 \cos(nt + \alpha) \tag{2.31a}$$
$$y = -2A_0 \sin(nt + \alpha) \tag{2.31b}$$
$$z = B_0 \cos(nt + \beta) \tag{2.31c}$$

If we compute the derivatives of (2.31) we obtain

$$\dot{x} = -A_0 n \sin(nt + \alpha) \tag{2.32a}$$
$$\dot{y} = -2A_0 n \cos(nt + \alpha) \tag{2.32b}$$
$$\dot{z} = -B_0 n \sin(nt + \beta) \tag{2.32c}$$

and

$$\ddot{x} = -A_0 n^2 \cos(nt + \alpha) = -n^2 x \tag{2.33a}$$
$$\ddot{y} = 2A_0 n^2 \sin(nt + \alpha) = -n^2 y \tag{2.33b}$$
$$\ddot{z} = -B_0 n^2 \cos(nt + \beta) = -n^2 z \tag{2.33c}$$

Inserting them inside (2.23) yields

$$-A_0 n^2 \cos(nt + \alpha) - 3A_0 n^2 \cos(nt + \alpha) + 4A_0 \omega n \cos(nt + \alpha) = u_x \tag{2.34a}$$
$$-2A_0 n^2 \sin(nt + \alpha) + n^2 y_{off} - 2A_0 \omega n \sin(nt + \alpha) = u_y \tag{2.34b}$$
$$-B_0 n^2 \cos(nt + \beta) + B_0 \omega^2 \cos(nt + \beta) = u_z \tag{2.34c}$$

Imposing

$$\boxed{n = \omega, \quad y_{\text{off}} = 0} \tag{2.35}$$

makes the left-hand side of (2.34) identically zero. Consequently, the Clohessy-Wiltshire equations are satisfied with $\mathbf{u} = 0$, showing that the motion is a natural

(unforced) solution of the dynamics. Finally, in order to set the size of the relative orbit, the following condition must be imposed

$$\boxed{2A_0 = R} \tag{2.36}$$

The parameter B_0 must also be determined. In fact, the only control-free relative orbits correspond to trajectories lying on a plane rotated by 30° about the $y-$axis. Since the maximum oscillation along the $x-$direction is already fixed, the geometric constraint yields

$$A_0 = R\sin\phi \doteq \frac{R}{2} \quad \longrightarrow \quad \phi = \frac{\pi}{6} \tag{2.37}$$

which results in

$$\boxed{B_0 = R\cos\phi = \frac{\sqrt{3}}{2}R} \tag{2.38}$$

For any other inclination angle, the resulting relative trajectories become elliptical.

The requirements for a control-free circular relative orbit with radius R can thus be summarised in the following list:

1. the angular rate of the reference and local orbits must be the same;
2. the offset along $\mathbf{y}$ must be zero;
3. the maximum amplitude of the oscillations along $\mathbf{y}$ must be R;
4. the maximum amplitude of the oscillations along $\mathbf{x}$ must be $R/2$;
5. the maximum amplitude of the oscillations along $\mathbf{z}$ must be $\sqrt{3}/2\ R$.

Although this trajectory represents a natural solution of the Clohessy-Wiltshire equations, it is inherently unstable in the presence of external disturbances. As a result, the chaser spacecraft tends to drift away from the desired motion over time. For this reason, the next chapter introduces a control strategy aimed at stabilizing and maintaining the relative orbit.

Chapter 3

MPC design

An Model Predictive Controller (MPC) entails a feedback control strategy where control inputs are computed by minimizing an objective function for a dynamical system subject to constraints, across a fixed-length window that shifts forward in time [10]. MPCs come in several variants depending on the plant dynamics, wanted performance and robustness, as well as type of reference to be used. In this thesis, a linear, discrete-time MPC will be designed and implemented.

An MPC may be considered a receding-horizon variant of the Linear Quadratic Regulator (LQR)[1], with the added benefit of constraints enforcement. The reasons behind the choice of an MPC controller over other types of algorithms are multiple and not trivial at all:

- **constraints handling** — the optimal input sequence takes into consideration the constraints imposed by the user, which can be used to avoid unwanted states. This is essential in the field of aerospace engineering to avoid dangerous or nonsensical flying conditions for both aircraft and satellites, as well as to include actuator saturations. In fields like robotic engineering, for example, the range of motion of a robot arm must be limited to not damage their surroundings, to limit the turning speed and to prevent human injury.
- **MIMO structure** — the controller has an intrinsic multiple-input multiple-output (MIMO) architecture, meaning it can handle more than one state and control input at a time. This is one of the main drawbacks of the PID controller, which is the most used type of controller for industrial applications,

[1]To be specific, an LQR derives the optimal static feedback gain K by solving the Continuous Algebraic Riccati Equations (CARE) (in continuous time) or Discrete Algebraic Riccati Equations (DARE) (in discrete time), which yield the solution to the infinite-horizon optimization problem offline. The LQR can be thus seen as a special case of the linear MPC in which the horizon is extended to infinity and no constraints are imposed on the states or inputs.

but which can be used directly only for SISO systems and is difficult to scale to MIMO configurations;

- **future predictions** — the controller returns an optimised input value depending on the current state of the system and the future references. This makes it a *proactive* algorithm, since it adapts itself to the instantaneous conditions of the problem;
- **penalisations** — the user controls the behaviour of the MPC by tuning the weights of the cost function inside the algorithm. This way, they can decide which states get prioritised when generating the control action, how quickly the system should converge to the reference and how intensely the actuators should be used.

The main drawback of using MPCs is the higher computational burden that the controller puts on the hardware performing all the calculations. Given the high number of variables involved in the solution process of the optimisation problem and the wanted frequency of said solutions, the MPC may fail to keep up with the required performance, especially if the problem is very big in terms of number of states and equations, if the control horizon is too long or if the discretisation of time into samples is too dense. Moreover, if the plant has been modelled using non-linear dynamics inside the MPC, the algorithm runtime increases, as solving a non-linear optimisation problem is intrinsically harder than a linear or quadratic one. For this reason, and given the nature of the Clohessy-Wiltshire (CW) equations, a linear MPC will be used, running on the Operator-Splitting Quadratic Program (OSQP), a fast open-source solver based on the Alternating-Direction Method of Multipliers (ADMM) algorithm.

3.1 Minimisation problem

The problem is structured[2] [11] as follows

$$\min_{\mathbf{x},\mathbf{u}} \sum_{k=1}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k, \mathbf{x}_r, \mathbf{u}_r) + V_f(\mathbf{x}_N, \mathbf{x}_r), \tag{3.1a}$$

$$\text{subject to } \mathbf{x}_0 = \mathbf{x}(t), \tag{3.1b}$$

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k, \quad k \in \mathbb{N}_{[0,N-1]}, \tag{3.1c}$$

$$(\mathbf{x}_k, \mathbf{u}_k) \in \mathcal{Z}, \quad k \in \mathbb{N}_{[0,N-1]} \tag{3.1d}$$

[2] In general one should use $\mathbf{y} = \mathbf{C}\mathbf{x}$ instead of $\mathbf{x}$, since not all states may be observable. In our case this is redundant, since $\mathbf{C} = \mathbf{I}_{n_x}$, effectively making $\mathbf{y} = \mathbf{x}$.

where N is the prediction horizon,

$$\mathbf{x} = [\mathbf{x}_0, \dots, \mathbf{x}_N], \quad \mathbf{x}_k \in \mathbb{R}^{n_x} \tag{3.2a}$$
$$\mathbf{u} = [\mathbf{u}_0, \dots, \mathbf{u}_N], \quad \mathbf{u}_k \in \mathbb{R}^{n_u} \tag{3.2b}$$

are respectively the vectors of states and inputs at each time step, $\mathbf{x}_r \in \mathbb{R}^{n_x}$ and $\mathbf{u}_r \in \mathbb{R}^{n_u}$ are their reference counterparts, while $\mathcal{Z} \in \mathbb{R}^{n_x+n_u}$ is a convex polyhedron containing the origin in its interior [11], representing the space defined by the collection of inequalities through which the constraints are imposed. Last but not least,

$$J = \sum_{k=1}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k, \mathbf{x}_r, \mathbf{u}_r) + V_f(\mathbf{x}_N, \mathbf{x}_r) \tag{3.3}$$

is the cost function (also called "performance index"), where ℓ is the stage cost and V_f is the terminal cost.

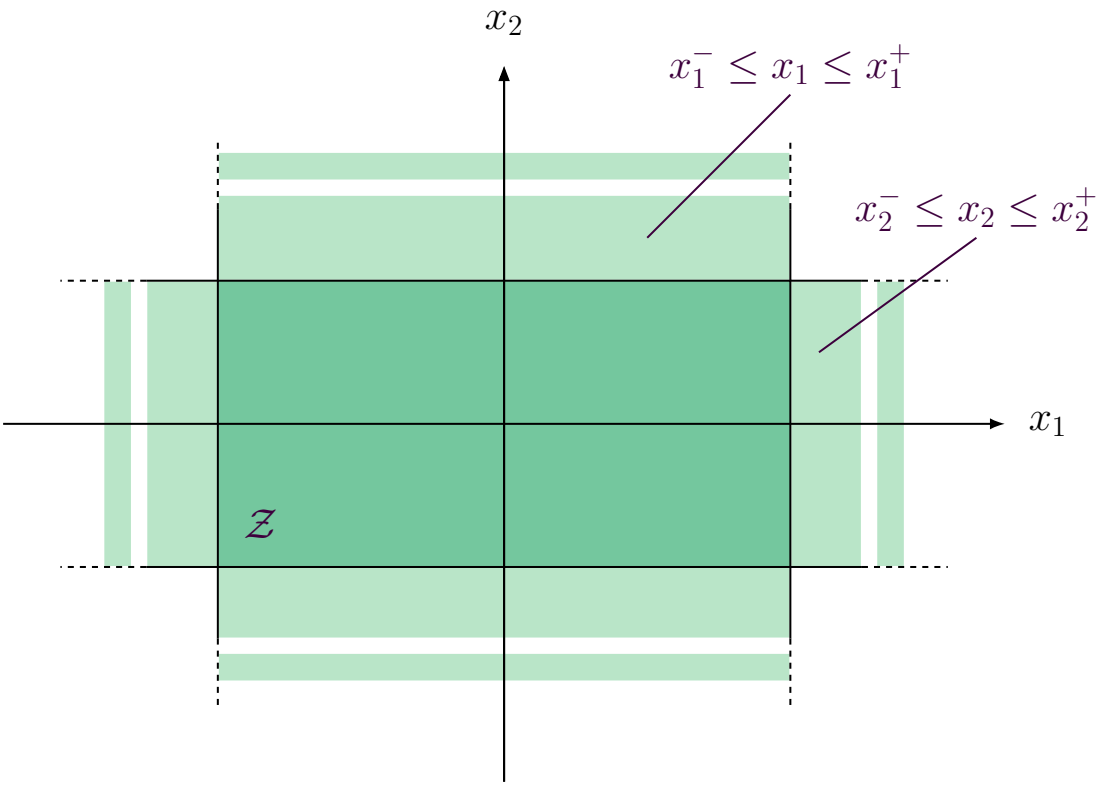


Figure 3.1: A trivial example of two inequalities defining a $\mathcal{Z}$ set in a bidimensional state-space.

At each time step the MPC controller identifies the current state of the plant (3.1b) and solves a constrained optimization problem (3.1d) to determine an ideal

sequence of control actions. This calculation uses an internal plant model (3.1c) to minimize a cost function (3.1a) over a defined horizon. Only the first action in this sequence is applied; the remaining steps are discarded, and the entire optimization cycle repeats at the next interval.

3.2 Incremental input formulation

In its standard formulation, the controller optimizes over absolute control inputs, requiring the system state to be driven directly to a desired reference. While effective, this approach can only optimise the control input as-is, and limits management of the actuator effort. For instance, the system may be able to track the reference but overshoot it while doing so.

To address this limitation, the incremental (or velocity-form) MPC formulation recasts the problem in terms of control input increments rather than absolute inputs. By augmenting the system model with an integrator on the input channel, integral action is embedded directly into the controller structure, ensuring tracking without the need for a separate integrator in the loop. This also naturally enforces input rate constraints, which are often physically motivated — for instance, actuator slew rate limits — and are otherwise cumbersome to handle in the standard formulation.

The incremental formulation thus offers a principled and elegant extension of standard MPC, retaining all the advantages of predictive control while endowing it with robustness to low-frequency disturbances and seamless handling of rate constraints. To take this aspect into consideration we can expand

$$\mathbf{u}_k = \mathbf{u}_{k-1} + \Delta\mathbf{u}_k \tag{3.4}$$

By substituting (3.4) inside (2.25) we can write

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_{k-1} + \mathbf{B}\Delta\mathbf{u}_k \tag{3.5}$$

By expressing the input as

$$\mathbf{x}_{u,k+1} = \mathbf{u}_k \tag{3.6}$$

we can unite (3.4) and (3.5) into a single, augmented system of equations

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{x}_{u,k} + \mathbf{B}\Delta\mathbf{u}_k \tag{3.7}$$

$$\mathbf{x}_{u,k+1} = \mathbf{x}_{u,k} + \Delta\mathbf{u}_k \tag{3.8}$$

which can be rewritten as

$$\begin{bmatrix}\mathbf{x}\\ \mathbf{x}_u\end{bmatrix}_{k+1} = \begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{x}_u\end{bmatrix}_k + \begin{bmatrix}\mathbf{B}\\ \mathbf{I}\end{bmatrix}\Delta\mathbf{u}_k \tag{3.9}$$

We therefore define the new state and (delta) input vectors as

$$\boldsymbol{\xi} = \begin{bmatrix} \mathbf{x} \\ \mathbf{x}_u \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ u_x \\ u_y \\ u_z \end{bmatrix}, \quad \Delta\mathbf{u} = \begin{bmatrix} \Delta u_x \\ \Delta u_y \\ \Delta u_z \end{bmatrix} \tag{3.10}$$

and the new state matrices as

$$\boldsymbol{\mathcal{A}} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{bmatrix}, \quad \boldsymbol{\mathcal{B}} = \begin{bmatrix} \mathbf{B} \\ \mathbf{I} \end{bmatrix}, \quad \boldsymbol{\mathcal{C}} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix} \tag{3.11}$$

giving us

$$\boxed{\begin{aligned} \boldsymbol{\xi}_{k+1} &= \boldsymbol{\mathcal{A}}\boldsymbol{\xi}_k + \boldsymbol{\mathcal{B}}\Delta\mathbf{u}_k \\ \mathbf{y}_k &= \boldsymbol{\mathcal{C}}\boldsymbol{\xi}_k \end{aligned}} \tag{3.12a}$$
$$\tag{3.12b}$$

3.3 Constraints

The presence of discontinuities in the form of actuator saturations requires the input to be bounded within those limits. This mathematically anomalous behaviour is encountered frequently in the real world, and the flexible nature of the MPC allows for inclusion of such information in a very straightforward way. Despite this simplicity, a control designer should always try to preserve the convexity of the $\mathcal{Z}$ set, a concept which is explained in more detail in Appendix A.

Since our case involves merely constant upper- and lower bounds limitations, convexity will not be affected. We can therefore take (3.1a) and impose three additional inequalities (one per thruster direction) associated with the saturations on $\mathbf{u}$ to the problem as follows

$$\min_{\boldsymbol{\xi},\mathbf{u}} \sum_{k=1}^{N-1} \ell(\boldsymbol{\xi}_k, \Delta\mathbf{u}_k, \boldsymbol{\xi}_r, \Delta\mathbf{u}_r) + V_f(\boldsymbol{\xi}_N, \boldsymbol{\xi}_r), \tag{3.13a}$$

$$\text{subject to } \boldsymbol{\xi}_0 = \boldsymbol{\xi}(t), \tag{3.13b}$$

$$\boldsymbol{\xi}_{k+1} = \boldsymbol{\mathcal{A}}\boldsymbol{\xi}_k + \boldsymbol{\mathcal{B}}\Delta\mathbf{u}_k, \quad k \in \mathbb{N}_{[0,N-1]}, \tag{3.13c}$$

$$\mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}, \quad k \in \mathbb{N}_{[0,N-1]} \tag{3.13d}$$

where $\mathbf{u}_k = \boldsymbol{\xi}_{k+1}\{7,8,9\}$ as stated in (3.6).

3.4 Input horizon

We now need to add a constraint to the input deltas. We introduce the input horizon N_u which is the number of future time steps over which the controller actively optimizes and computes control inputs. After the input horizon, the incremental control input is set to zero for the remainder of the prediction horizon, effectively constraining $\mathbf{u}$ to stay constant. This provides several benefits: it allows us to reduce the complexity of the system, adds another tuning parameter to better fine-tune the final response and forces the MPC to ignore inputs far into the future. To add a constraint to $\Delta\mathbf{u}$ we simply expand the previous problem to

$$\begin{aligned}
&\min_{\boldsymbol{\xi},\mathbf{u}} \sum_{k=1}^{N-1} \ell(\boldsymbol{\xi}_k, \Delta\mathbf{u}_k, \boldsymbol{\xi}_r, \Delta\mathbf{u}_r) + V_f(\boldsymbol{\xi}_N, \boldsymbol{\xi}_r), && \text{(3.14a)}\\
&\text{subject to } \boldsymbol{\xi}_0 = \boldsymbol{\xi}(t), && \text{(3.14b)}\\
&\qquad \boldsymbol{\xi}_{k+1} = \boldsymbol{\mathcal{A}}\boldsymbol{\xi}_k + \boldsymbol{\mathcal{B}}\mathbf{u}_k, \quad k \in \mathbb{N}_{[0,N-1]}, && \text{(3.14c)}\\
&\qquad \mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}, \quad k \in \mathbb{N}_{[0,N-1]} && \text{(3.14d)}\\
&\qquad \Delta\mathbf{u}_{\min} \leq \Delta\mathbf{u}_k \leq \Delta\mathbf{u}_{\max}, \quad k \in \mathbb{N}_{[0,N_u-1]} && \text{(3.14e)}\\
&\qquad \Delta\mathbf{u}_k = \mathbf{0}, \quad k \in \mathbb{N}_{[0,N_u-1]} && \text{(3.14f)}
\end{aligned}$$

The $\mathcal{Z}$ set is thus defined by the following equations: (3.14d) for the absolute inputs along the whole horizon, (3.14e) for the relative inputs along the input horizon and (3.14f) for the relative inputs from there onwards.

3.5 Quadratic Programming

Since the form of the cost function may be chosen arbitrarily, one may decide to use an even-power polynomial for this purpose, since it's naturally convex. Starting from the standard problem formulation, Quadratic Programming (QP) involves shaping the cost function [11] as

$$J = \sum_{k=1}^{N-1} \left[(\mathbf{x}_k - \mathbf{x}_r)^\mathsf{T} \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_r) + \mathbf{u}_k^\mathsf{T} \mathbf{R} \mathbf{u}_k \right] + (\mathbf{x}_N - \mathbf{x}_r)^\mathsf{T} \mathbf{P} (\mathbf{x}_N - \mathbf{x}_r) \tag{3.15}$$

where

$$\begin{aligned}
&\mathbf{Q} \in \mathcal{S}_{\succeq}^{n_x} \subset \mathbb{R}^{n_x \times n_x} && \text{(3.16a)}\\
&\mathbf{R} \in \mathcal{S}_{\succ}^{n_u} \subset \mathbb{R}^{n_u \times n_u} && \text{(3.16b)}\\
&\mathbf{P} \in \mathcal{S}_{\succeq}^{n_x} \subset \mathbb{R}^{n_x \times n_x} && \text{(3.16c)}
\end{aligned}$$

are respectively the **state**, **input** and **terminal cost matrices**, with $\mathcal{S}^n_{\succeq}$ and $\mathcal{S}^n_{\succ}$ the subsets of all positive semi-definite[3] and positive definite matrices in $\mathbb{R}^{n\times n}$, respectively. These matrices define the trade-off between performance and control effort in optimal control problems like MPCs and LQRs, with a slightly different effect in both cases. In fact, much like in an LQR, their role is as follows:

- **state cost matrix** — the $\mathbf{Q}$ matrix penalizes the deviation of the all-but-last system states from the desired reference. Diagonal entries correspond to each state variable, with larger values translating to a stronger effort to keep that state close to its wanted value;
- **input cost matrix** — the $\mathbf{R}$ matrix penalizes the magnitude of control inputs. Diagonal entries correspond to each input variable. As stated in (3.16b), the input cost matrix must be strictly positive, as we always pay some cost for actuation. Larger value are indicative of a more conservative controller, i.e. one that uses less aggressive inputs;
- **terminal cost matrix** — the $\mathbf{P}$ matrix penalizes the deviation of the last system states from the desired reference. Behaviour is essentially the same as the state cost matrix, except that it's usually set to much higher values. In this way, the algorithm returns a state and input sequence such that their last value is as close as possible to the reference, while retaining some flexibility for the previous states.

The effects of the matrix parameters are summarised in Table 3.1.

Another aspect to take into account is the type of reference. If the desired state is not the same for all the time steps of the optimisation problem, one needs to explicitly indicate

$$J = \sum_{k=1}^{N-1} \left[(\mathbf{x}_k - \mathbf{x}_{r,k})^\mathsf{T} \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_{r,k}) + \mathbf{u}_k^\mathsf{T} \mathbf{R} \mathbf{u}_k \right] + (\mathbf{x}_N - \mathbf{x}_{r,N})^\mathsf{T} \mathbf{P} (\mathbf{x}_N - \mathbf{x}_{r,N}) \quad (3.17)$$

where $\mathbf{x}_r = [\mathbf{x}_{r,1}, \dots, \mathbf{x}_{r,N}]$ is the full reference vector.

Finally, if we consider the state expansion presented in (3.10) we may rewrite (3.17) as

$$\boxed{J = \sum_{k=1}^{N-1} \left[(\boldsymbol{\xi}_k - \boldsymbol{\xi}_{r,k})^\mathsf{T} \mathbf{Q}^* (\boldsymbol{\xi}_k - \boldsymbol{\xi}_{r,k}) + \Delta\mathbf{u}_k^\mathsf{T} \mathbf{R}^* \Delta\mathbf{u}_k \right] + (\boldsymbol{\xi}_N - \boldsymbol{\xi}_{r,N})^\mathsf{T} \mathbf{P}^* (\boldsymbol{\xi}_N - \boldsymbol{\xi}_{r,N})} \quad (3.18a)$$

[3] As a reminder, a matrix $\mathbf{A}$ is positive semidefinite if $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} \geq 0$. This is essentially the extension of the concept of $ax^2 \geq 0$ to higher-dimensional spaces.

where

$$\mathbf{Q}^* \in \mathcal{S}_{\succeq}^{n_\xi} \subset \mathbb{R}^{n_\xi \times n_\xi} \tag{3.19a}$$
$$\mathbf{R}^* \in \mathcal{S}_{\succeq}^{n_u} \subset \mathbb{R}^{n_u \times n_u} \tag{3.19b}$$
$$\mathbf{P}^* \in \mathcal{S}_{\succeq}^{n_\xi} \subset \mathbb{R}^{n_\xi \times n_\xi} \tag{3.19c}$$

are the expanded versions of the $\mathbf{Q}$, $\mathbf{R}$ and $\mathbf{P}$ matrices, respectively. In our specific case, as seen in (3.10), we have $n_\xi = n_x + n_u = 9$, while $n_u = 3$ stays the same. The easiest choice is to pick $\mathbf{Q}^*$, $\mathbf{R}^*$ and $\mathbf{P}^*$ such that they are diagonal. In this way we can simplify the tuning process by defining

$$\mathbf{Q}^* = \begin{bmatrix} w_p\mathbf{I}_3 & & \\ & w_v\mathbf{I}_3 & \\ & & w_u\mathbf{I}_3 \end{bmatrix} \tag{3.20a}$$
$$\mathbf{R}^* = w_{\Delta u}\mathbf{I}_3 \tag{3.20b}$$
$$\mathbf{P}^* = w_t\mathbf{Q}^* \tag{3.20c}$$

with only five independent parameters.

Matrices	
$\mathbf{Q}, \mathbf{Q}^*$	penalises state deviation
$\mathbf{R}, \mathbf{R}^*$	penalises control effort
$\mathbf{P}, \mathbf{P}^*$	penalises terminal state deviation
Weights	
w_p	position deviation cost
w_v	velocity deviation cost
w_u	absolute inputs cost
$w_{\Delta u}$	incremental inputs cost
w_t	terminal scaling factor

Table 3.1: Effect of each weight in the cost function.

3.6 Reference trajectory

The reference trajectory used in the simulations is generated for the six original state variables (position and velocity) at every time step. We start by calculating the final separation angle. In general, this is given geometrically by

$$\varphi_i = \theta + \omega t + (i-1)\frac{2\pi}{N_{\text{sat}}}, \quad i \in \mathbb{N}_{[1,N_{\text{sat}}]} \tag{3.21}$$

where θ is the global phase shift, N_{sat} is the total number of satellites and i the identifier of each spacecraft. The phase angle φ_i varies depending on the chosen formation. In our case, a triangular formation was analysed, and by assuming a null phase shift we get that at the beginning of the simulation ($t = 0$ s) the angles are simply

$$\varphi_1(0) = 0^\circ \tag{3.22}$$
$$\varphi_2(0) = 120^\circ \tag{3.23}$$
$$\varphi_3(0) = 240^\circ \tag{3.24}$$

The values change at each time step of the simulation, given the ωt component, and are used to define the reference states vector in accordance with (2.31) and (2.32), with their coefficients imposed by (2.35), (2.36) and (2.38). The aforementioned relations have all been derived in Chapter 2.

The radius of the orbit is gradually increased, by including a simple, linearly increasing variable external to the MPC. The radius is thus calculated as

$$\Delta r = 0.001(R - ||\mathbf{x}||_2) \tag{3.25}$$
$$r_{k+1} = r_k + \Delta r \tag{3.26}$$

Since Δr becomes increasingly small, r_{k+1} is clamped to R once their difference is less than $1 \cdot 10^{-5}$ m.

The final trajectory equations thus become

$$\boxed{\begin{aligned} x_{r,k} &= \frac{r_k}{2}\cos\left(\omega t + (i-1)\frac{2\pi}{N_{\text{sat}}}\right) && \text{(3.27a)} \\ y_{r,k} &= -r_k \sin\left(\omega t + (i-1)\frac{2\pi}{N_{\text{sat}}}\right) && \text{(3.27b)} \\ z_{r,k} &= \frac{\sqrt{3}}{2} r_k \cos\left(\omega t + (i-1)\frac{2\pi}{N_{\text{sat}}}\right) && \text{(3.27c)} \end{aligned}}$$

followed by

$$\boxed{\begin{aligned}
\dot{x}_{r,k} &= -\frac{r_k}{2}\omega \sin\left(\omega t + (i-1)\frac{2\pi}{N_{\text{sat}}}\right) && \text{(3.28a)}\\
\dot{y}_{r,k} &= -r_k\omega \cos\left(\omega t + (i-1)\frac{2\pi}{N_{\text{sat}}}\right) && \text{(3.28b)}\\
\dot{z}_{r,k} &= -\frac{\sqrt{3}}{2} r_k\omega \sin\left(\omega t + (i-1)\frac{2\pi}{N_{\text{sat}}}\right) && \text{(3.28c)}
\end{aligned}}$$

This ensures that the target trajectory is always within the saturation capabilities of the spacecraft[4].

The final result is therefore a reference CRO that is propagated for a number of steps equal to that of the MPC, and which is gradually inflated to the final radius.

[4]The process of finding the right speed of increase is entirely empirical, as it's based on trial-and-error.

Chapter 4

Experimental/numerical evaluation

4.1 Overview

All simulations have been set up in a Matlab-Simulink environment, with the OSQP solver assembled and run in a Matlab System block inside Simulink[1]. The simulations have all been executed on an Intel Core 7 processor. The simulation parameters are listed in Table 4.1; the various controller parameters are shown in Table 4.3; lastly, the initial conditions for both spacecraft are shown in Table 4.2.

Several analyses have been performed: a **noise-free** analysis, one with **low** and **high noise**[2] on the actuators, and one with a **thruster misfire** event.

Integrator details	
solver	`ode4` (Runge-Kutta 4)
dt_{sim}	0.01 s
T_{sim}	12.5 $T_{\text{orb}} \approx 9.5 \cdot 10^4$ s

Table 4.1: Simulation parameters.

[1]For additional details and information about how to implement the mathematical formulas in a Matlab-Simulink code, please refer to Appendix B.

[2]At least two orders and one order of magnitude lower than $\mathbf{u}_{\max}$, respectively.

Initial conditions						
Satellite	x_0	y_0	z_0	$\dot{x}_0$	$\dot{y}_0$	$\dot{z}_0$
`chaser_1`	$R/2$	0	$\sqrt{3}/2\, R$	0	$-2\omega x_0$	0
`chaser_2`	0	0.1	0	0	0	0

Table 4.2: Initial conditions for `chaser_1` and `chaser_2`.

MPC parameters	
T_{MPC}	$0.1\ T_{\text{orb}} \approx$ 760 s
dt_{MPC}	10 s
N_{MPC}	76
MPC weights	
w_p	0.0001
w_v	0.0001
w_u	5
$w_{\Delta u}$	100
w_t	1000
OSQP solver settings	
`warm_start`	`true`
`polish`	`true`
`check_termination`	1
`eps_abs`	$1 \cdot 10^{-5}$
`eps_rel`	$1 \cdot 10^{-5}$
`max_iter`	$1 \cdot 10^{4}$

Table 4.3: Model Predictive Control and OSQP parameters

4.2 Noise-free simulation

The first simulation performed is noise-free: `chaser_1` is already positioned on the final CRO and is assumed to be holding a perfect Flying Formation position[3], while `chaser_2` is injected near the origin of the LVLH frame and is ordered to reach its assigned position.

The goal is to show that the controller can guide the chaser spacecraft, let it follow the reference states and ultimately have it reach the desired target separation between `chaser_1` and `chaser_2`, all in a disturbance-free environment.

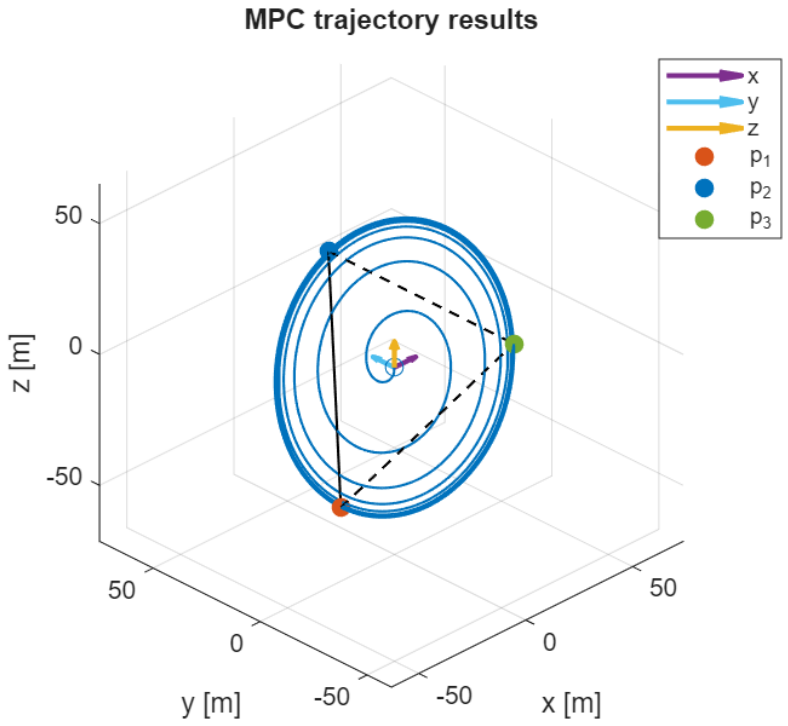


Figure 4.1: Spiral motion of `chaser_2` around the origin.

As a result, Figure 4.1 shows the final trajectory in the LVLH frame. The green dot represents the hypothetical final position of `chaser_3`, while the black lines show the sides of the formation (the dashed lines indicate a missing member). Figure 4.2 shows the position and velocity states of `chaser_2`, as well as the accelerations supplied by the actuators, which are shown as clamped by the modelled saturations. Figure 4.3 shows the input signal from the controller, which is not clamped yet. Lastly, Figure 4.4 shows the distance from `chaser_1`.

[3]All spacecraft injected on a CRO must abide to the initial conditions expressed in (2.30).

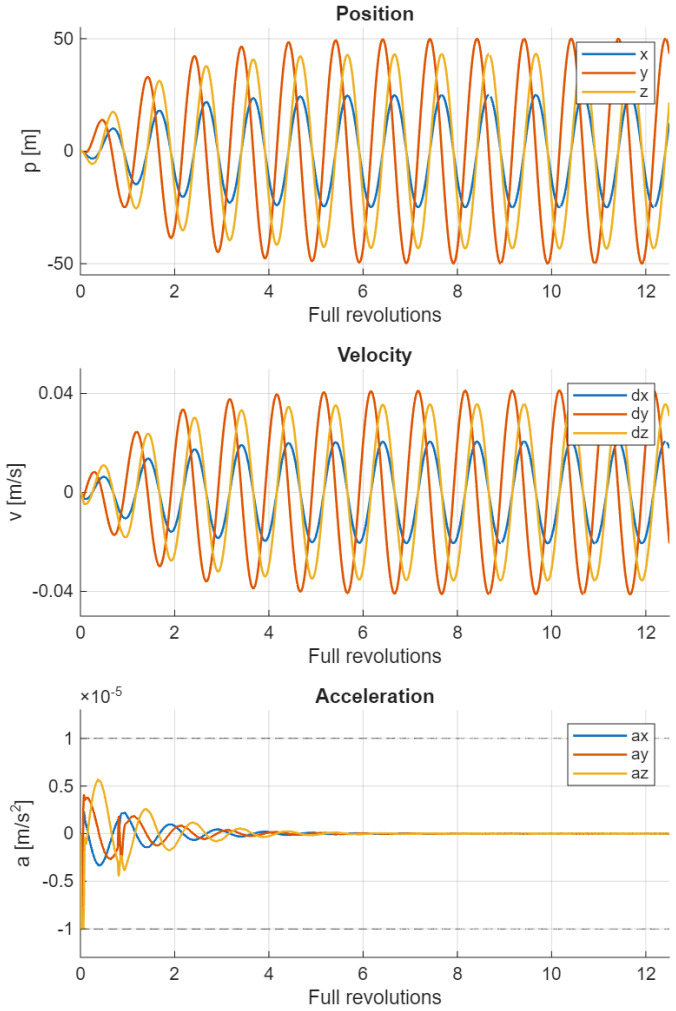


Figure 4.2: Position and velocity state variables and accelerations of `chaser_2` in the noise-free simulation.

An extremely important detail is shown in Figure 4.3. One may see how OSQP fails at enforcing the input constraints — we will expand on this in Chapter 5. Despite this, the saturation violation is very short and happens only at the very beginning of the simulation as the satellite accelerates to the initial velocity imposed by the reference. This is deemed acceptable for the purposes of this work.

As visible in Figure 4.4, the final position error in the noise-free simulation is approximately 0.00025 m. This hints at the fact that, given a sufficiently long simulation time T_{sim}, the satellite will eventually find itself at or below the maximum wanted position error along the CRO.

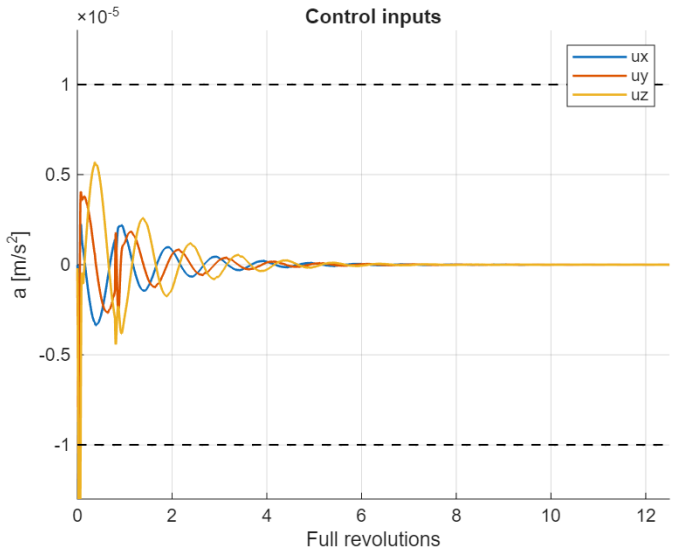


Figure 4.3: Control effort expressed as acceleration of `chaser_2` in the noise-free simulation. Note the violation at $t \approx 0$ seconds.

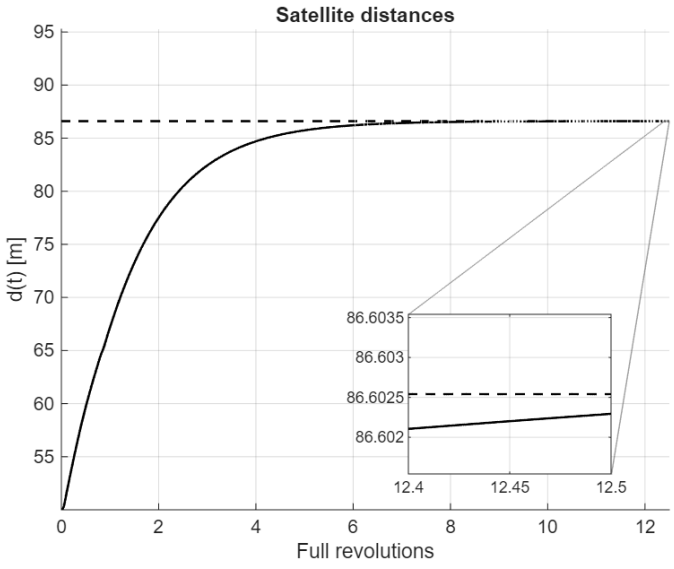


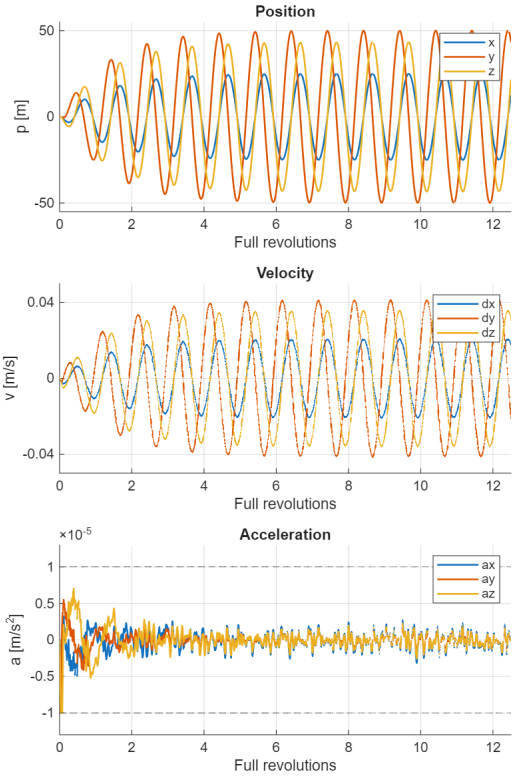
Figure 4.4: Distance between `chaser_1` and `chaser_2` positions in the noise-free simulation.

4.3 Noise on inputs

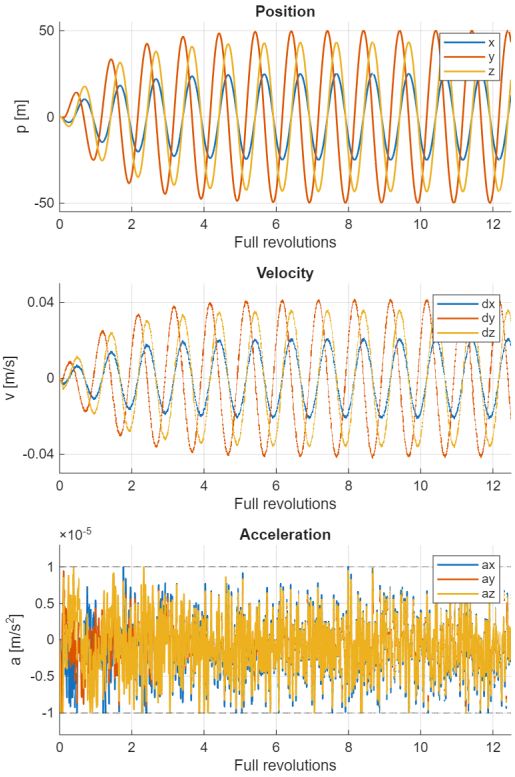
The second and third simulations show the system behaviour when a non-zero mean Gaussian white noise is added. The noise is simply expressed as

$$w_k \sim \mathcal{N}(\mu, \sigma^2) \tag{4.1}$$

where μ is the mean value and σ^2 the variance, with the noise injected into the input channel. Results are shown in Figures 4.5a, 4.6a and 4.7a for the low-noise case, and in Figures 4.5b, 4.6b and 4.7b for the high-noise case.



(a) Low noise — $\mu = 10^{-7}$ m/s^2, $\sigma^2 = 10^{-8}$ m/s^2

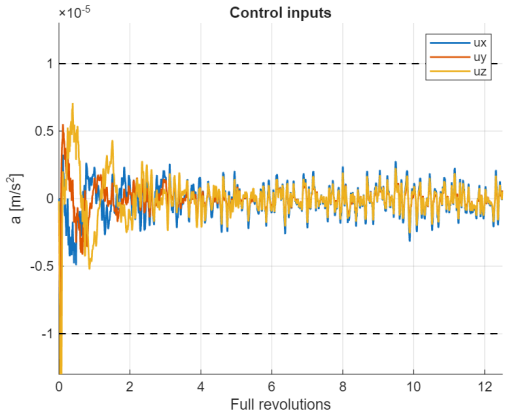


(b) High noise — $\mu = 10^{-6}$ m/s^2, $\sigma^2 = 10^{-7}$ m/s^2

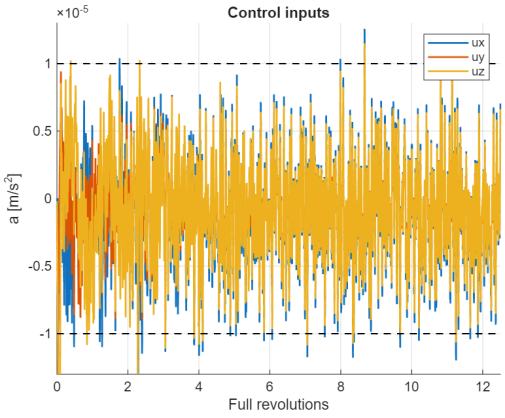
Figure 4.5: Position and velocity state variables and accelerations of `chaser_2` in the noisy simulations.

Overall, the behaviour of the satellite remains controllable, with the distance converging to the final value. This means that the MPC is still able to fight the non-zero mean component of the added noise. In both cases one can deduce that the MPC cannot logically reach a zero-input situation, resulting in the behaviour shown in Figures 4.6a and 4.6b.

The main difference between all three simulations is the final accuracy, which is severely lowered, at approximately 0.05 m for the low-noise case and at $0.5 \sim 1$ m for the high-noise case. This is because the random component of the noise cannot be predicted by the MPC. The issue found in the first simulation, i.e. the constraints violation, appears again, this time more frequent and not just limited to $t = 0$ seconds.

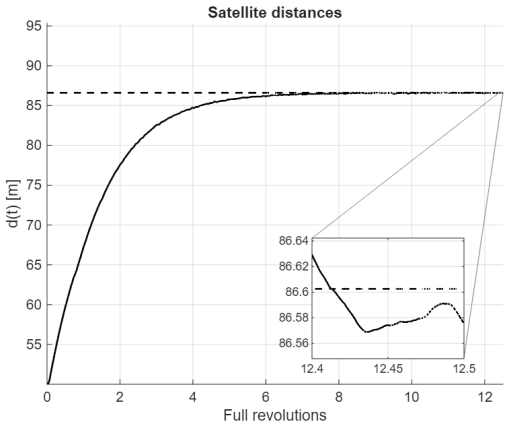


(a) Low noise — $\mu = 10^{-7}$ m/s^2, $\sigma^2 = 10^{-8}$ m/s^2

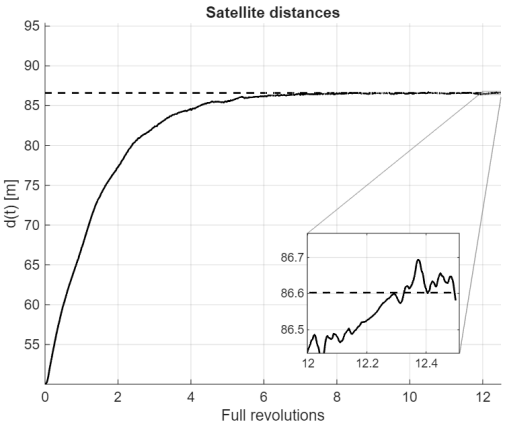


(b) High noise — $\mu = 10^{-6}$ m/s^2, $\sigma^2 = 10^{-7}$ m/s^2

Figure 4.6: Control effort expressed as acceleration of `chaser_2` in the noisy simulations.



(a) Low noise — $\mu = 10^{-7}$ m/s^2, $\sigma^2 = 10^{-8}$ m/s^2



(b) High noise — $\mu = 10^{-6}$ m/s^2, $\sigma^2 = 10^{-7}$ m/s^2

Figure 4.7: Distance between `chaser_1` and `chaser_2` positions in the noisy simulations.

4.4 Thruster misfire

In the fourth and final simulation, a thruster misfire is analysed. The failure is implemented as a simple constant value added to the input channel via a double step function. The amplitude of the misfire is 50% the value of $U_{\max}$, for a duration of 100 seconds. Results are shown in Figures 4.8, 4.9 and 4.10.

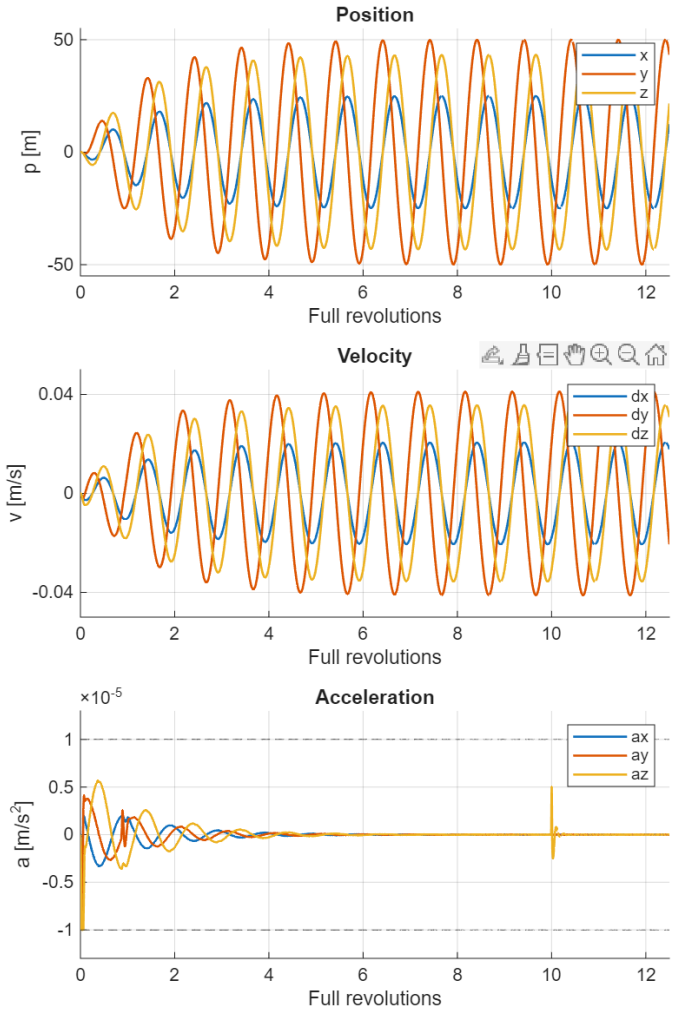


Figure 4.8: Position and velocity state variables and accelerations of `chaser_2` in the thruster misfire simulations.

The controller immediately reacts to the displacement induced by the misfire. The maximum overshoot with respect to the previous distance value is measured to be about 0.06 m, with a settling time of about 0.3 revolutions.

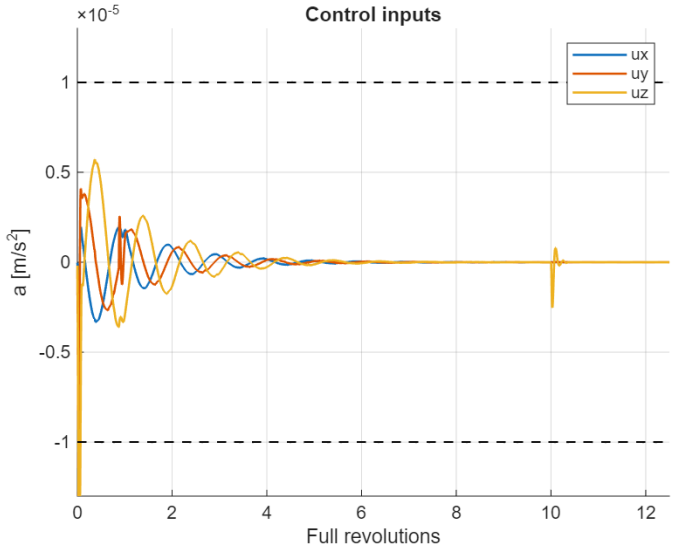


Figure 4.9: Control effort expressed as acceleration of `chaser_2` in the thruster misfire simulations. The wrongful thruster action is compensated by the MPC during the late transient of the misfire simulation.

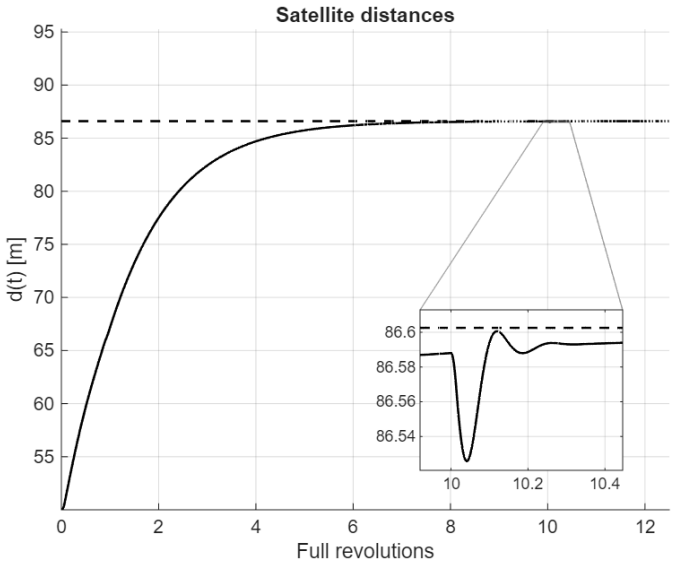


Figure 4.10: Distance between `chaser_1` and `chaser_2` positions in the thruster misfire simulations. Note how the system restabilizes after the misfire.

Chapter 5

Conclusion

5.1 Summary

The main objective of this thesis was to design and implement an Model Predictive Control logic to guide a satellite to a specified final CRO along which a Flying Formation has to be maintained. The controller is tasked with both reaching the final CRO and keeping the FF. This is usually performed in two different phases and possibly with separate controllers; we attempted to find a single-architecture to perform both actions without interruptions while still achieving a solid accuracy once steady-state is reached.

A series of tests have also been run to analyse performance under disturbances, namely:

- **noisy input signal** — white Gaussian noise was artificially induced in the control input signal line to check for sensitivity to small, unpredictable input variations;
- **thruster misfire** — thruster failure was simulated by increasing the input signal value to 50% of the maximum value for 100 seconds once the transient is mostly complete.

5.2 Complications

The present scenario involved several hurdles that needed to be overcome:

- **extremely low saturation inputs** — these severely penalise the performance of the satellite and limit the effectiveness of the controller;
- **dynamics with a high degree of instability** — reaching a high speed in one of the three directions puts the stability of the system at risk. Firing

the thrusters for too long in one of the in-plane directions leads to a state divergence due to the coupled nature of said variables;

- **badly scaled matrices and vectors** — given the extremely low saturations, the orders of magnitude of the position and velocity states are several times higher than those of the inputs, a behaviour which is also reflected in the constraints of the MPC. The assembly of all matrices by the ADMM algorithm makes the resulting $\mathbf{H}$ matrix very badly scaled (check Chapter B for more insight on the problem building process);
- **numerical noise** — operating with numbers that go as low as 10^{-8} causes problems with the tolerances of the solver, since some values may be lower than said limit and could be counted as noise by the controller.

The immediate consequence of all these factors is a simulation that does indeed show satisfactory results, but which is also extremely sensitive to aspects such as change of initial conditions, injection of noise and disturbances, influencing the ability to follow a trajectory and/or to reach references far from the initial conditions without some additional aid. The parameters that let the system work as well as it did in the aforementioned simulations are the result of a very tight tug of war between enforcing constraints and fighting low orders of magnitude, during the tuning phase of the controller parameters. For instance, a solution tolerance too low can make the solution infeasible as the ADMM solver reads the values of the upper and lower constraints affected by the numerical noise and finds that they are illogical[1], while keeping it too high can lead to the same ADMM solver ignoring completely the values associated with e.g. the incremental inputs[2].

5.3 Comparison with similar works

While the results show a great accuracy under the no-disturbances assumption, different implementations of the problem by other authors show much better performance even under disturbances. For instance, in [12] results show an accuracy of about 10^{-3} mm using a passivity-based controller augmented with an atmospheric drag estimator, running in a non-linear dynamics system known as the Port-Hamiltonian formulation. In another example [13], authors show that while their

[1] If the $\mathbf{H}$ matrix is badly scaled, numerical noise is added on top of the very small values constituting the constraints. This can make the lower boundary for said constraint higher than the upper boundary (and vice-versa), making both nonsensical and leading to a so-called "primal infeasibility".

[2] The order of magnitude of $\Delta\mathbf{u}$ is about 10^{-8}. This means that a constraint on $\Delta\mathbf{u}$ may be completely ignored when the tolerance on the solution is around 10^{-5}.

optimal controller based on a bang-bang law cannot effectively reduce the error past a limit, a sliding mode strategy proves that even in the presence of disturbances the error is eliminated, without having to use a controller as complex as the MPC. It must be noted however that these works consider a mere FF-keeping control problem, not a full CRO-reaching-and-FF-keeping one. This means that the satellite starts near the final CRO and doesn't have to go through the approach phase, effectively ignoring that part. For this reason, we feel that the combination of the two tasks into a single control design problem, without any controller changes in between the two phases, returns very solid results even if the overall accuracy is inferior to other analyses when simply compared 1:1.

5.4 Future work

Finally, we will go over several suggestions and recommendations for any future work and potential continuations or extensions of the present tests, with the aim of guiding subsequent research in a more informed direction.

5.4.1 Solver alternatives

The first thing to take into consideration is that while OSQP is surely one of, if not *the* fastest solver which could be used inside the MPC, other solvers may be deemed more useful for a better constraint enforcement action, since OSQP is not perfectly strict and may fail under certain circumstances, as visible in the results in Chapter 4. Moreover, the speed of OSQP is not really needed given the chosen time step of integration of the solver, and a slower solver may still be deemed sufficient. Algorithm alternatives prioritizing strictness and precision of the solution may be identified in:

- **PROXQP** — often used in robotics applications, it's an efficient solver that features a global convergence guarantee for convex QPs, which is mandatory for safe closed-loop control. As all open-source software, it is freely obtainable online [14];

- **MOSEK** — arguably the highest precision specialist for conic optimization. It uses a primal-dual homogeneous IPM and handles infeasible/unbounded cases appropriately. Has an academic license available [15];

- **FORCESPRO** — a commercial solver that is widely considered the industry state of the art. Requires a purchase to use [16].

5.4.2 Flexible trajectory planning

Right now the satellites use a reference based on the analytical solution of the Clohessy-Wiltshire (CW) equations, applied at each time step of the MPC horizon. A flexible alternative would be to generate a reference for the trajectory that depends on the current state of other agents, measured by the individual satellite. In such cases, following techniques should be evaluated:

- **conversion to a non-linear solver for norm-based optimisation** — the act of calculating the Euclidean distance between two points requires a non-linear process which is mathematically incompatible with the QP solvers shown earlier, and which must be external to the MPC code. To calculate the constraints and to optimise the distances between satellites directly inside the cost function, one should switch to Non-Linear Programming (NLP);
- **conversion to non-linear dynamics in spherical coordinates** — by switching to a non-linear formulation one may express the current position and velocity of the satellite in the spherical coordinates around the origin, removing the need for a rotation from the Cartesian to the (ρ, θ, ϕ) formulation;
- **conversion to difference-based formulation** — by rewriting the dynamics in such a way that the states become the difference between agents positions and velocities, the optimisation of such variables should become much more straightforward.

5.4.3 Controlled radius variation

If using a predetermined trajectory that depends on a variable radius, one of the following alternatives may be deemed more robust and/or flexible for obtaining said variable:

- **adaptive radius** — as shown in [17], using an adaptive law to increase the radius can ensure stability for high radii. The law governing this behaviour is

$$\dot{R} = \text{Proj}_{[R_i, R_s]} \left[-Gh \|\mathbf{x} - \mathbf{x}_r\|_2 - \gamma(R - R_f) \right] \tag{5.1}$$

  where the radius is increased when the following condition is respected

$$\dot{R} > 0 \quad \text{if} \quad \max\left(\|\mathbf{x} - \mathbf{x}_r\|\right) < \eta\bar{\delta} \tag{5.2}$$

  with $\bar{\delta}$ being a limit on the state error in (5.2) shown to ensure convergence;
- **artificial references** — adding artificial reference variables that are constrained to identify strictly reachable states [11], i.e.

$$\mathbf{x} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{5.3}$$

and controlling how quickly they converge to the final radius reference may, in some cases, prove to be a reliable solution to the problem;

- **radius state expansion** — involves adding an additional state variable ρ governed by the integrator equation

$$\rho_k = \rho_{k-1} + \Delta\rho \tag{5.4}$$

and subsequently including both a constraint [17] on $\Delta\rho$ as

$$0 < \Delta\rho < \frac{U_{\max}}{4\omega} \cdot \Delta t_{MPC} \tag{5.5}$$

and a cost function component for ρ such as

$$J_\rho = w_\rho(\rho - R)^2 \tag{5.6}$$

defined in a similar fashion to (3.15) where w_ρ is a 1D weight acting on the ρ quadratic state error. In this way the MPC itself can manage how quickly the radius is changed. Mathematically speaking, this is the simplest of the three solutions; it requires however some tinkering with the standard problem assembly shown in Appendix B.

5.4.4 Normalisation of the state-space system

Given the high discrepancy between the orders of magnitude of the state variables in the expanded formulation, a total normalisation of the state-space model is advisable. The normalisation shall be performed by scaling all variables by their (hypothesized) maximum value, making them oscillate only between 0 and 1. This improves the conditioning of the $\mathbf{H}$ matrix inside the solver and leads to more precise solutions. We suppose that the constraint enforcement action by OSQP would greatly benefit by the normalisation, since the tests with higher saturations ($> 1 \cdot 10^{-5}$ m/s^2) show no issue in this regard.

5.4.5 Robust MPC

The usage of a robust MPC can improve the performance of the controller when states are noisy, generally due to lower-quality sensors. To tackle this problem, a robust Model Predictive Controller can be designed, as shown in [11].

5.5 Final comments

While the results of the simulations show some signs of success, all the complications encountered severely slowed down the development of a more robust and flexible

configuration. For this reason, the information gathered during the period of writing of this thesis heavily suggests further refinement of the methodology, changing some of the techniques and improving the flexibility and adaptability of the controller. Nevertheless, for the specific initial conditions and model parameters chosen, the results show a very good performance in the noise-free and thruster misfire cases, despite the usage of fairly elementary strategies, such as the radius management. Furthermore, the simulations are very quick at completing, making the fine-tuning phase much less tedious than expected.

Appendix A

Convexity

One of the main hurdles when dealing with MPCs is the convexity of the $\mathcal{Z}$ set, of the cost function and, as a result, of the entire optimisation problem [18].

Definition 1 (Convex set) *The set $\mathcal{Z}$ is convex if it contains every line segment between any two points in the set.*

Examples of convex sets in 2-dimensional spaces are circles, triangles, parallelograms, etc. while examples of convex sets in 3-dimensional spaces are spheres, cylinders, parallelepipeds, etc.

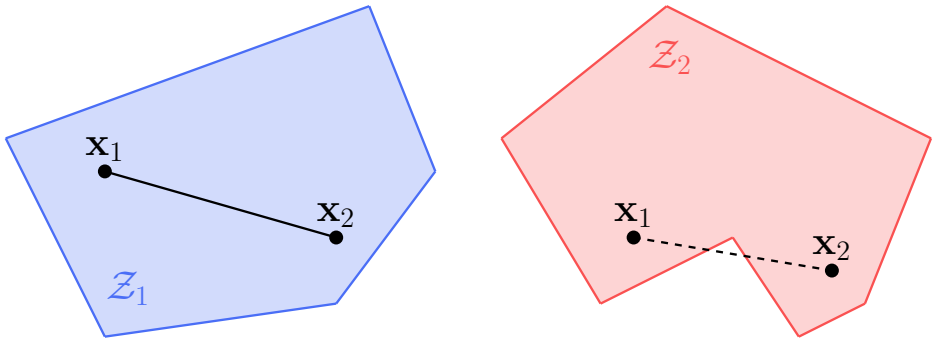


Figure A.1: A convex set, on the left, and a non-convex set, on the right.

Definition 2 (Convex function) *A function $f : \mathcal{X} \to \mathbb{R}$ is convex if it never intersects the line segment between any two points of its graph.*

In the case of simple, low-dimensional spaces (1- or 2D), one can imagine a convex function as shaped like a cup, or a bowl. This example loses meaning in higher-dimensional spaces.

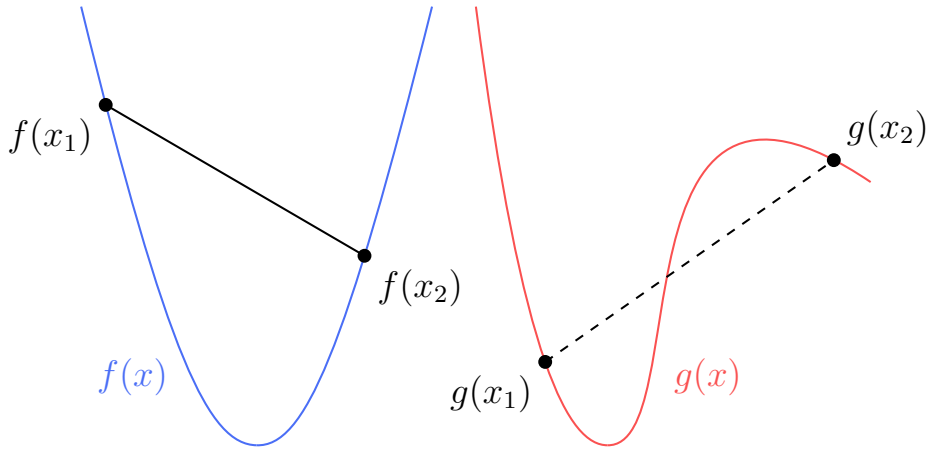


Figure A.2: A convex function, on the left, and a non-convex one, on the right.

Assembling a Quadratic Program (QP) means using a quadratic function, which is naturally convex.

Definition 3 (Convex optimisation problem) *The optimisation problem*

$$\min\ f(x) \tag{A.1}$$
$$\text{subject to } x \in \mathcal{Z} \tag{A.2}$$

is a convex optimisation problem if $\mathcal{Z}$ is a convex set and if $f : \mathcal{X} \to \mathbb{R}$ is a convex function.

A minimisation problem being convex has many advantages, the main ones being that it can be solved much more efficiently by dedicated algorithms, and that any local solution to a convex problem is automatically a global solution [18].

Adding constraints to an unconstrained problem will determine the shape of $\mathcal{Z}$, making the volume smaller than infinity in one or more of the $n_x + n_u$ directions, potentially resulting in a non-convex set (e.g. when holes are formed in the dominion). Should the set $\mathcal{Z}$ be non-convex, many *convexification* approaches are available.

Appendix B

Casting to OSQP

In order to solve the optimisation problem using the Operator-Splitting Quadratic Program (OSQP) we must convert the mathematical formulation used in Chapter 3 to a slightly different algebraic structure. The main hurdle is the need to unfold the sequential time step problem into a single, full-length system of equations which OSQP can solve using the Alternating-Direction Method of Multipliers (ADMM) algorithm. The problem structure [19] adopted by OSQP is the following

$$\min_{\mathbf{z}} \frac{1}{2}\mathbf{z}^{\mathsf{T}}\mathbf{H}\mathbf{z} + \mathbf{q}^{\mathsf{T}}\mathbf{z} \tag{B.1a}$$

$$\text{subject to } \mathbf{l} \leq \mathbf{G}\mathbf{z} \leq \mathbf{u} \tag{B.1b}$$

where $\mathbf{z} \in \mathbb{R}^n$ is the optimisation variable vector (also called the "decision variable vector"), $\mathbf{H} \in \mathcal{S}^n_{\succeq}$ is the objective function matrix, $\mathbf{q} \in \mathbb{R}^n$ is the objective function vector, $\mathbf{G} \in \mathbb{R}^{m \times n}$ is the linear constraints matrix, while $\mathbf{l}$ and $\mathbf{u}$ are the vectorised lower and upper bounds for the linear constraints such that

$$l_i \in \mathbb{R} \cup \{-\infty\}, \quad i \in \mathbb{N}_{[1,m]} \tag{B.2}$$

$$u_i \in \mathbb{R} \cup \{+\infty\}, \quad i \in \mathbb{N}_{[1,m]} \tag{B.3}$$

Let us start by defining the optimisation variables vector. Using the incremental formulation shown in (3.10) we can append the states at each time step one after the other, followed by the incremental inputs in the same way. We obtain

$$\mathbf{z} = \begin{bmatrix} \boldsymbol{\xi}_0 \\ \vdots \\ \boldsymbol{\xi}_N \\ \Delta\mathbf{u}_0 \\ \vdots \\ \Delta\mathbf{u}_{N-1} \end{bmatrix} \tag{B.4}$$

As an immediate consequence, we can see that, in our case, we have

$$n = n_\xi \cdot (N+1) + n_u \cdot N \tag{B.5}$$

We continue by assembling the $\mathbf{H}$ matrix accordingly. Given the adoption of Quadratic Programming with its form shown in (3.18a), the components which include a reference inside the cost function can be expanded using the binomial theorem as follows

$$J_{\xi,k} = (\boldsymbol{\xi}_k - \boldsymbol{\xi}_{k,r})^\mathsf{T} \mathbf{Q}^* (\boldsymbol{\xi}_k - \boldsymbol{\xi}_{k,r}) \tag{B.6}$$

$$= \underbrace{\boldsymbol{\xi}_k^\mathsf{T} \mathbf{Q}^* \boldsymbol{\xi}_k}_{\text{quadratic}} + \underbrace{\boldsymbol{\xi}_{r,k}^\mathsf{T} \mathbf{Q}^* \boldsymbol{\xi}_{r,k}}_{\text{constant}} - \underbrace{2\boldsymbol{\xi}_{r,k}^\mathsf{T} \mathbf{Q}^* \boldsymbol{\xi}_k}_{\text{linear}} \tag{B.7}$$

$$J_{\xi,N} = (\boldsymbol{\xi}_N - \boldsymbol{\xi}_{N,r})^\mathsf{T} \mathbf{P}^* (\boldsymbol{\xi}_N - \boldsymbol{\xi}_{N,r}) \tag{B.8}$$

$$= \underbrace{\boldsymbol{\xi}_N^\mathsf{T} \mathbf{P}^* \boldsymbol{\xi}_N}_{\text{quadratic}} + \underbrace{\boldsymbol{\xi}_{r,N}^\mathsf{T} \mathbf{P}^* \boldsymbol{\xi}_{r,N}}_{\text{constant}} - \underbrace{2\boldsymbol{\xi}_{r,N}^\mathsf{T} \mathbf{P}^* \boldsymbol{\xi}_N}_{\text{linear}} \tag{B.9}$$

where the constant terms do not change the location of the minimum in the cost function and can be safely ignored during the search for the solution. Note that $J_{\Delta\mathbf{u},k} = \Delta\mathbf{u}_k^\mathsf{T} \mathbf{R}^* \Delta\mathbf{u}_k$ stays unchanged since it doesn't include a reference for $\Delta\mathbf{u}_k$.

By splitting the cost function into two parts, i.e. a purely quadratic contribution, written as

$$J_{\text{quad}} = \sum_{k=1}^{N-1} (\boldsymbol{\xi}_k^\mathsf{T} \mathbf{Q}^* \boldsymbol{\xi}_k + \Delta\mathbf{u}_k^\mathsf{T} \mathbf{R}^* \Delta\mathbf{u}_k) + \boldsymbol{\xi}_N^\mathsf{T} \mathbf{P}^* \boldsymbol{\xi}_N \tag{B.10}$$

and a linear contribution, written as

$$J_{\text{lin}} = \sum_{k=1}^{N-1} (-2\boldsymbol{\xi}_{r,k}^\mathsf{T} \mathbf{Q}^* \boldsymbol{\xi}_k) - 2\boldsymbol{\xi}_{r,N}^\mathsf{T} \mathbf{P}^* \boldsymbol{\xi}_N \tag{B.11}$$

we have expressed our cost function in the form shown in (B.1). Let us now unwrap and assemble the quadratic component coefficients in a single bigger matrix, $\mathbf{H}$, and the linear component coefficients in a single vector, $\mathbf{q}$, in accordance with (B.4). The boxes on the right side of vectors and matrices help understand how the matrices are assembled. We can write

$$\mathbf{H} = \begin{bmatrix} \mathbf{Q}^* & & & & & & \\ & \ddots & & & & & \\ & & \mathbf{Q}^* & & & & \\ & & & \mathbf{P}^* & & & \\ & & & & \mathbf{R}^* & & \\ & & & & & \ddots & \\ & & & & & & \mathbf{R}^* \end{bmatrix} \begin{array}{|l} \boldsymbol{\xi}_0 \\ \\ \boldsymbol{\xi}_{N-1} \\ \boldsymbol{\xi}_N \\ \Delta\mathbf{u}_0 \\ \\ \Delta\mathbf{u}_{N-1} \end{array} \tag{B.12}$$

where the subscript indicates the k-th state or input vector inside $\mathbf{z}$. Using code, one may write

```
% quadratic objective
H = blkdiag(kron(speye(N), Q), P, kron(speye(N), R) );
```

We continue by defining the $\mathbf{q}$ vector as

$$\mathbf{q} = \begin{bmatrix} -2\boldsymbol{\xi}_{r,0}^{\mathsf{T}}\mathbf{Q}^* \\ \vdots \\ -2\boldsymbol{\xi}_{r,N-1}^{\mathsf{T}}\mathbf{Q}^* \\ -2\boldsymbol{\xi}_{r,N}^{\mathsf{T}}\mathbf{P}^* \\ 0 \\ \vdots \\ 0 \end{bmatrix} \begin{matrix} \boldsymbol{\xi}_0 \\ \\ \boldsymbol{\xi}_{N-1} \\ \boldsymbol{\xi}_N \\ \Delta\mathbf{u}_0 \\ \\ \Delta\mathbf{u}_{N-1} \end{matrix} \tag{B.13}$$

where we use zeroes in the incremental input slots for the same reasons already stated earlier. This vector can be assembled in code as follows

```
% linear objective (from offset reference)
q = [repmat(-Q*xr, N, 1); -P*xr; zeros(N*nu, 1)]
```

If using a reference that changes for each time step, `xr` must also be vectorised, resulting in

```
xr   = traj(:); % vectorised trajectory
bigQ = kron(speye(N), Q); % block diagonal of N matrices of Q
q    = [-bigQ*xr(1:N*nx); -P*xr(N*nx+1:end); zeros(N*nu, 1)];
```

We can now impose the dynamics of the system. This is done inside the $\mathbf{G}$ matrix. Said matrix should also contain all the other constraints, so it is better to define one component of $\mathbf{G}$ at a time. The dynamics are expressed by (2.25), which needs to be unrolled into a time-independent formulation. We obtain

$$\mathbf{z}_0 = \boldsymbol{\xi}_0 \tag{B.14}$$
$$\mathbf{z}_1 = \boldsymbol{\mathcal{A}}\boldsymbol{\xi}_0 + \boldsymbol{\mathcal{B}}\Delta\mathbf{u}_0 \tag{B.15}$$
$$\vdots$$
$$\mathbf{z}_N = \boldsymbol{\mathcal{A}}\boldsymbol{\xi}_{N-1} + \boldsymbol{\mathcal{B}}\Delta\mathbf{u}_{N-1} \tag{B.16}$$

or, equivalently

$$-\boldsymbol{\xi}_0 \leq -\mathbf{z}_0 \leq -\boldsymbol{\xi}_0 \tag{B.17}$$
$$\mathbf{0} \leq -\mathbf{z}_1 + \boldsymbol{\mathcal{A}}\boldsymbol{\xi}_0 + \boldsymbol{\mathcal{B}}\Delta\mathbf{u}_0 \leq \mathbf{0} \tag{B.18}$$
$$\vdots$$
$$\mathbf{0} \leq -\mathbf{z}_N + \boldsymbol{\mathcal{A}}\boldsymbol{\xi}_{N-1} + \boldsymbol{\mathcal{B}}\Delta\mathbf{u}_{N-1} \leq \mathbf{0} \tag{B.19}$$

which are essentially pure equalities and can be expressed in matrix form as

$$\mathbf{G}_{\text{eq}} = \begin{bmatrix} -\mathbf{I}_{n_\xi} & & & & \mathbf{0} & \mathbf{0} & & \\ \boldsymbol{\mathcal{A}} & -\mathbf{I}_{n_\xi} & & & & \boldsymbol{\mathcal{B}} & \mathbf{0} & \\ & \ddots & \ddots & & & & \ddots & \ddots \\ & & \boldsymbol{\mathcal{A}} & -\mathbf{I}_{n_\xi} & & & & \boldsymbol{\mathcal{B}} \quad \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_0 \\ \boldsymbol{\xi}_1 \\ \\ \boldsymbol{\xi}_N \end{bmatrix} \tag{B.20a}$$
$$= \begin{bmatrix} \boldsymbol{\mathcal{A}}_{\text{full}} & \boldsymbol{\mathcal{B}}_{\text{full}} \end{bmatrix} \tag{B.20b}$$

The upper and lower bounds[1] are thus

$$\mathbf{l}_{\text{eq}} = \mathbf{u}_{\text{eq}} = \begin{bmatrix} -\boldsymbol{\xi}_0 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_0 \\ \boldsymbol{\xi}_1 \\ \\ \boldsymbol{\xi}_N \end{bmatrix} \tag{B.21}$$

The constraints in (B.20) and the bounds in (B.21) can be translated to code as

```
% linear dynamics
Ax = kron( speye(N+1), -speye(nx)) + ...
        kron(sparse(diag(ones(N, 1), -1)), A);
Bu = kron([sparse(1, N); speye(N)], B);

Geq = [Ax, Bu];
leq = [-x0; zeros(N*nx, 1)];
ueq = leq;
```

We now need to enforce our custom inequality constraints. The procedure is identical. The constraint matrix becomes trivial

$$\mathbf{G}_{\text{ineq}} = \mathbf{I}_n \tag{B.22}$$

[1] It is worth noting that, in order to follow the dynamics of the system, we get $m \geq n_\xi \cdot (N+1)$, given the number of constraints associated with the state-space model. Any further constraints aren't strictly required for the MPC to function.

while the bounds are slightly more complicated, as we need to discern them based on the individual time step temporal collocation with respect to N_u, the input horizon. We thus define

$$\mathbf{l}_{\text{ineq}} = \begin{bmatrix} \boldsymbol{\xi}_{\min} \\ \vdots \\ \boldsymbol{\xi}_{\min} \\ \Delta\mathbf{u}_{\min} \\ \vdots \\ \Delta\mathbf{u}_{\min} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_0 \\ \\ \boldsymbol{\xi}_N \\ \Delta\mathbf{u}_0 \\ \\ \Delta\mathbf{u}_{N_u-1} \\ \Delta\mathbf{u}_{N_u} \\ \\ \Delta\mathbf{u}_{N-1} \end{bmatrix} \qquad \mathbf{u}_{\text{ineq}} = \begin{bmatrix} \boldsymbol{\xi}_{\max} \\ \vdots \\ \boldsymbol{\xi}_{\max} \\ \Delta\mathbf{u}_{\max} \\ \vdots \\ \Delta\mathbf{u}_{\max} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_0 \\ \\ \boldsymbol{\xi}_N \\ \Delta\mathbf{u}_0 \\ \\ \Delta\mathbf{u}_{N_u-1} \\ \Delta\mathbf{u}_{N_u} \\ \\ \Delta\mathbf{u}_{N-1} \end{bmatrix} \tag{B.23}$$

The constraints in (B.22) and the bounds in (B.23) can thus be implemented via code as

```
% input and state constraints
Gineq = speye((N+1)*nx + N*nu);
lineq = [repmat(xmin, N+1, 1); repmat(umin, Nu+1, 1); repmat(
zeros(nu,1), N-(Nu+1), 1)];
uineq = [repmat(xmax, N+1, 1); repmat(umax, Nu+1, 1); repmat(
zeros(nu,1), N-(Nu+1), 1)];
```

All constraints and bounds need to be packaged in a single component of each as follows

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{\text{eq}} \\ \mathbf{G}_{\text{ineq}} \end{bmatrix} \qquad \mathbf{l} = \begin{bmatrix} \mathbf{l}_{\text{eq}} \\ \mathbf{l}_{\text{ineq}} \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} \mathbf{u}_{\text{eq}} \\ \mathbf{u}_{\text{ineq}} \end{bmatrix} \tag{B.24}$$

which easily translates to

```
% all OSQP constraints
G = [Aeq; Aineq];
l = [leq; lineq];
u = [ueq; uineq];
```

Last but not least, we can finally build the entire problem by running

```
% create an OSQP object ---
prob = osqp;
```

All the problem matrices and vectors $(\mathbf{H}, \mathbf{q}, \mathbf{G}, \mathbf{l}, \mathbf{u})$, as well as any settings can be included via

```
% Setup workspace                          % defaults:
prob.setup(H, q, G, l, u,              ...
    'warm_start',              true,   ... % true
    'polish',                  true,   ... % false
    'check_termination',       1,      ... % 25
    'adaptive_rho',            true,   ... % true
    'eps_abs',                 1e-5,   ... % 1e-3
    'eps_rel',                 1e-5,   ... % 1e-3
    'max_iter',                10000);     % 4000
```

It is highly advised to build the problem just once (setup phase) and only then solve it (step phase) at each time step. The solution can be obtained by calling

```
% call for solution
prob.solve()
```

At runtime, the problem shall be updated with the current initial conditions as

```
% update initial conditions
l(1:nx) = -[x; u_old];
u(1:nx) = -[x; u_old];
```

where `x` is the satellite's current position. To update the problem, we simply run

```
% update problem with new I/C
prob.update('l', l, 'u', u);
```

If, like in our case, the reference also changes at each time step, it is possible to update it in the same way by calling

```
% update problem with new I/C
prob.update('l', l, 'u', u, 'q', q);
```

Bibliography

[1] Daniel Scharf, Fred Hadaegh, and Scott Ploen. «A Survey of Spacecraft Formation Flying Guidance and Control (Part I): Guidance». In: vol. 2. July 2003, pp. 1733–1739. ISBN: 0-7803-7896-2. DOI: 10.1109/ACC.2003.1239845 (cit. on p. 1).

[2] M K Hobish. *Remote Sensing Tutorial*. Feb. 2001. URL: https://web.archive.org/web/20070810231952/http://rst.gsfc.nasa.gov/Sect16/Sect16_11.html (cit. on p. 1).

[3] Jane J. Lee and Andrew Wang. Mar. 2024. URL: https://www.jpl.nasa.gov/news/us-germany-partnering-on-mission-to-track-earths-water-movement/ (cit. on p. 1).

[4] W M Folkner, F Hechler, T H Sweetser, M A Vincent, and P L Bender. «LISA orbit selection and stability». In: *Classical and Quantum Gravity* 14.6 (June 1997), p. 1405. DOI: 10.1088/0264-9381/14/6/003. URL: https://doi.org/10.1088/0264-9381/14/6/003 (cit. on p. 1).

[5] Karsten Danzmann. *eLISA: The Mission - Gravity is the dominant force in the universe.* URL: https://web.archive.org/web/20131205104316/https://www.elisascience.org/articles/elisa-mission/elisa-mission-gravitational-universe (cit. on p. 1).

[6] W. H. Clohessy and R. S. Wiltshire. «Terminal Guidance System for Satellite Rendezvous». In: *Journal of the Aerospace Sciences* 27.9 (1960), pp. 653–658. DOI: 10.2514/8.8704. eprint: https://doi.org/10.2514/8.8704. URL: https://doi.org/10.2514/8.8704 (cit. on p. 8).

[7] G. W. Hill. «Researches in the Lunar Theory». In: *American Journal of Mathematics* 1.1 (1878), pp. 5–26. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2369430 (visited on 01/27/2026) (cit. on p. 8).

[8] Wigbert Fehse. *Automated Rendezvous and Docking of Spacecraft*. Cambridge Aerospace Series. Cambridge University Press, 2003 (cit. on p. 8).

[9] Hanspeter Schaub. «Spacecraft relative orbit geometry description through orbit element differences». In: *14th U.S. National Congress of Theoretical and Applied Mechanics* (Jan. 2002) (cit. on p. 12).

[10] Matlab documentation. *What is Model Predictive Control?* URL: https://www.mathworks.com/help/mpc/gs/what-is-mpc.html (cit. on p. 14).

[11] Pablo Krupa, Johannes Köhler, Antonio Ferramosca, Ignacio Alvarado, Melanie N. Zeilinger, Teodoro Alamo, and Daniel Limon. *Model predictive control for tracking using artificial references: Fundamentals, recent results and practical implementation*. 2024. arXiv: 2406.06157 [eess.SY]. URL: https://arxiv.org/abs/2406.06157 (cit. on pp. 15, 16, 19, 36, 37).

[12] Satoshi Satoh and Yuki Hamanaka. «Nonlinear formation tracking control based on generalized canonical transformations with adaptive mechanism for atmospheric drag». In: *Advances in Space Research* 77 (2026), pp. 671–685. DOI: 10.1016/j.asr.2025.11.031 (cit. on p. 34).

[13] A. Imani, M. Bahrami, and B. Ebrahimi. «Optimal Sliding Mode Control for Spacecraft Formation Flying». In: *2nd International Conference on Control, Instrumentation and Automation (ICCIA)*. IEEE, 2011, pp. 38–43 (cit. on p. 34).

[14] Antoine Bambade, Fabian Schramm, Sarah El Kazdadi, Stéphane Caron, Adrien Taylor, and Justin Carpentier. «PROXQP: an Efficient and Versatile Quadratic Programming Solver for Real-Time Robotics Applications and Beyond». working paper or preprint. Sept. 2023. URL: https://inria.hal.science/hal-04198663 (cit. on p. 35).

[15] MOSEK ApS. *MOSEK*. URL: https://www.mosek.com/products/mosek/ (cit. on p. 35).

[16] embotech. *FORCESPRO — The reliable solver*. URL: https://www.embotech.com/forcespro (cit. on p. 35).

[17] Dario Ruggiero, Takahiro Ito, Elisa Capello, and Yuichi Tsuda. «Adaptive guidance for low-thrust formation flight mission in Circular Relative Orbit». In: *Acta Astronautica* 234 (2025), pp. 13–25. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2025.04.026. URL: https://www.sciencedirect.com/science/article/pii/S0094576525002267 (cit. on pp. 36, 37).

[18] Alberto Bemporad. *Model predictive control*. 2025. URL: http://cse.lab.imtlucca.it/~bemporad/mpc_course.html (cit. on pp. 39, 40).

[19] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. «OSQP: an operator splitting solver for quadratic programs». In: *Mathematical Programming Computation* 12.4 (2020), pp. 637–672. DOI: 10.1007/s12532-020-00179-2. URL: https://doi.org/10.1007/s12532-020-00179-2 (cit. on p. 41).