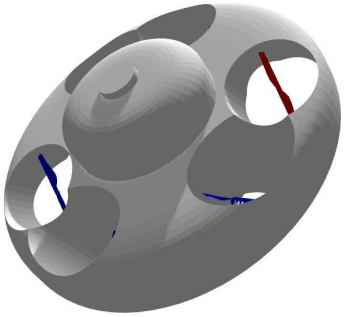
TÉCNICO LISBOA



A Nonlinear Model Predictive Control for payload transportation and formation flying by a free-flyer robot

Rui Filipe Freitas Correia

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor: Prof. Rodrigo Martins de Matos Ventura

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. Rita Maria Mendes de Almeida Correia da Cunha

October 2020

Dedicated to my parents

Acknowledgments

This thesis marks the end of a long and challenging academic journey. This was accomplished with the support and contribution of several people to whom I am grateful and dedicate this section.

I thank my supervisor Prof. Rodrigo Ventura for providing me the opportunity to work in this project. His insights and expertise during this project were key to ensure the success of this thesis.

To Diana Peixoto for being the unconditional support and never ending source of inspiration and motivation, I am extremely grateful.

For reading and thoroughly reviewing my entire thesis, and providing me with much needed motivation during challenging times, I would like to express my gratitude to Francisco Formigão.

I would also like to thank my colleague Francisco Gomes who worked alongside me in ISR in whom I could rely on for critical thinking whenever necessary.

For inciting my curiosity and pushing me to pursue my dreams in uncertain times, I thank Prof. Joaquim Marques.

To my parents for not only prioritizing my education and providing the resources to achieve it but also for the unconditional support throughout my entire academic journey.

Lastly, I thank the rest of my family and friends for providing me happy moments during all these years.

Resumo

O uso de robôs *free-flyer* em aplicações espaciais tem vindo a aumentar nos últimos anos. Várias agências espaciais, como a NASA, DLR e JAXA, têm neste momento ou planeiam ter projetos relacionados com este tipo de robôs.

O aumento do interesse neste tipo de robôs pode ser correlacionado com o desejo de expandir os limites da exploração espacial. Inerente a este interesse está a necessidade de desenvolver técnicas de controlo capazes de navegar e efetuar tarefas autonomamente mantendo em mente um gasto mínimo de energia.

Esta tese apresenta um estudo de desempenho de um controlador baseado em controlo preditivo não linear aplicado ao *free-flyer* Space CoBot, desenvolvido no ISR-Lisboa, num contexto de operação em tempo real, sujeito a diferentes condições de missão que exploram os limites do seu funcionamento e as margens de operação estável e viável.

Para guiamento foi integrado um método *offline* de geração de trajetórias factíveis, capaz de lidar não só com um ambiente fortemente condicionado por paredes e obstáculos mas também por restrições nos atuadores e dinâmica do sistema. Estas trajetórias são fornecidas ao controlador como referências.

Todos os testes conduzidos neste trabalho são obtidos exclusivamente através de um ambiente de simulação baseado em ROS e Gazebo/RotorS.

Palavras-chave: Robôs *free-flyer*, Controlo Preditivo Não Linear Rápido, Otimização de Trajetórias *offline*, Controlo, Guiamento, Simulação

Abstract

The use of free-flyer robots in space applications has been increasing in recent years. Several space agencies, such as NASA, DLR and JAXA, have at this moment, or intend to have, projects involving this type of robots.

The increasing interest in these robots can be associated with the desire to expand the limits of present space exploration. Inherent to this interest is the necessity to develop control systems capable of autonomous navigation and task execution while considering a minimum energy expenditure.

This thesis presents a performance study of nonlinear model predictive control applied to the free-flyer Space CoBot, developed by ISR-Lisboa, within a real-time operation context, subjected to different mission conditions which aim to explore the limits of its operational stability and reliability.

For performing guidance tasks, an offline trajectory generation method was integrated, capable of handling not only highly confined spaces, composed by walls and obstacles but also additional constraints such as control actuation limitations and system dynamics. These trajectories are provided to the controller as references.

All tests conducted in this work are obtained exclusively through a simulation environment based in ROS and Gazebo/RotorS.

Keywords: Free flyer robots, Fast NMPC, Offline Trajectory Optimization, Control, Guidance, Simulation

Contents

Acknowledgments v
Resumo vii
Abstract ix
List of Tables xv
List of Figures xvii
Nomenclature xxi
Acronyms xxv

1 Introduction **1**
1.1 Motivation 1
1.2 Objectives 2
1.2.1 Guidance, Navigation and Control 2
1.2.2 Validation 3
1.3 Contributions 3
1.4 Thesis outline 4

2 State-of-Art **5**
2.1 Free flying robots 5
2.2 Nonlinear Control 6
2.2.1 Linear methods 6
2.2.2 Nonlinear methods 7
2.2.3 Learning methods 7
2.2.4 Optimal Control methods 7
2.2.5 Model Predictive Control 8
2.2.6 Nonlinear Model Predictive Control 8
2.3 Trajectory Optimization 9
2.4 Simulation 10

3 Background **11**
3.1 Optimal control methods 11
3.2 Nonlinear Programing 12
3.2.1 Newton's method 12

3.2.2 Unconstrained problem . . . 13
3.2.3 Constrained problem . . . 13
3.2.4 Direct Multiple Shooting . . . 14
3.2.5 Sequential Quadratic Programming . . . 16
3.3 Nonlinear Model Predictive Control . . . 17
3.3.1 Basic algorithm . . . 17

4 Proposed Approach **19**
4.1 Models . . . 19
4.1.1 Space CoBot model . . . 19
4.1.2 Space CoBot with payload model . . . 23
4.1.3 Formation with payload model . . . 25
4.1.4 Formation flight without payload . . . 27
4.2 Nonlinear Model Predictive Control . . . 28
4.2.1 Problem formulation . . . 28
4.2.2 Sequential Quadratic Programming . . . 29
4.3 Trajectory Optimization . . . 30
4.3.1 Problem formulation . . . 31
4.3.2 Iterative LQR . . . 31
4.3.3 Augmented Lagrangian iLQR . . . 31
4.3.4 Active-Set Projection Method . . . 32
4.3.5 Infeasible state trajectory initialization . . . 32

5 Implementation **33**
5.1 Space Cobot parameters . . . 33
5.1.1 Design parameters . . . 34
5.2 Payload parameters . . . 36
5.3 Formation parameters . . . 36
5.4 Simulation environment . . . 36
5.4.1 Gazebo . . . 37
5.4.2 RotorS simulator . . . 37
5.4.3 Simulation parameters . . . 38
5.4.4 Payload grasping . . . 39
5.5 NMPC controller . . . 40
5.5.1 Actuation . . . 40
5.5.2 Objective function . . . 43
5.5.3 Initialization, state and control variables constraints . . . 44
5.5.4 Solver settings . . . 44
5.6 Trajectory Optimization . . . 45

6 Results **49**
6.1 Model verification . . . 49
6.2 NMPC controller configuration . . . 50
6.2.1 Prediction horizon . . . 50
6.2.2 Weighting matrices . . . 51
6.2.3 Control constraints . . . 54
6.3 Offline trajectory generation . . . 55
6.3.1 Problems formulation . . . 55
6.3.2 Trajectory execution . . . 56
6.4 Single Space CoBot . . . 57
6.4.1 Payload carrying . . . 59
6.5 Object avoidance . . . 62
6.5.1 Single Space CoBot single obstacle trajectory . . . 63
6.5.2 Single Space CoBot with payload single obstacle trajectory . . . 66
6.5.3 Two obstacles trajectory . . . 69
6.6 Formation with payload object avoidance . . . 73
6.6.1 Approach phase . . . 74
6.6.2 Transport phase . . . 75

7 Conclusions **79**
7.1 Achievements . . . 79
7.1.1 Control and guidance . . . 79
7.1.2 Simulation . . . 80
7.2 Future work . . . 80

Bibliography **81**

A Quaternions **87**
A.1 Quaternion operations . . . 87
A.1.1 Multiplication between vector and quaternion . . . 87
A.2 Attitude error approach comparison . . . 88

List of Tables

5.1 Space CoBot design parameters 34
5.2 HQ 4x4.5-BN performance data 34
5.3 RotorS model parameters 39
5.4 ACADO solver settings 45

6.1 Trajectory optimization computation time 55
6.2 ALTRO solver settings 56
6.3 Single Space Cobot step problem formulation 57
6.4 Single Space CoBot with payload step formulation 60
6.5 Single Space CoBot single obstacle problem formulation 63
6.6 Single obstacle single Space CoBot with payload problem formulation 67
6.7 Two obstacles single Space CoBot problem formulation 70
6.8 Formation configuration obstacle problem formulation 73

List of Figures

2.1 Existing free-flyers 6

4.1 Space CoBot 3D model 20
4.2 Notation used for single propeller model with respect to the body frame B 22
4.3 Space CoBot with payload configuration 23
4.4 Visualization of vector operation to obtain rotor positions with payload 24
4.5 Formation with payload configuration 25
4.6 Visualization of vector operation to obtain rotor positions in formation 26

5.1 Space CoBot prototype 33
5.2 Visualization of RotorS framework 38
5.3 Simulation environment framework 39
5.4 ACADO implementation scheme 41
5.5 Visualization of the controller node 42
5.6 Simulation environment with controller node scheme 45
5.7 Complete simulation environment scheme 47

6.1 Verification results. 50
6.2 Position step reference for different t_s values 51
6.3 Average feedback delay vs N 52
6.4 NMPC controller step response 52
6.5 Standalone NMPC step solution evaluation. 53
6.6 Prediction horizons for y axis during step response 53
6.7 Variation induced in z axis during step 54
6.8 Rotors angular velocity during step response 54
6.9 Generated step trajectory for single Space CoBot 58
6.10 Step tracking error of position and linear velocity for single Space CoBot 58
6.11 Step tracking error of attitude and angular velocity for single Space CoBot 58
6.12 Feedback delay for single Space CoBot step trajectory. 59
6.13 NMPC performance with step trajectory for single Space CoBot 59
6.14 NMPC response to step reference with unmodelled payload 60
6.15 Generated step trajectory for single Space CoBot with payload 61

6.16 Step tracking error of position and linear velocity for single Space CoBot with payload . . 61
6.17 Step tracking error of attitude and angular velocity for single Space CoBot with payload . 61
6.18 NMPC performance with step trajectory for single Space CoBot with payload 62
6.19 Collision safety barrier created by constraint 63
6.20 Single obstacle problem simulation environment. 63
6.21 Single obstacle generated trajectory 64
6.22 Single obstacle tracking error of position and linear velocity 64
6.23 Single obstacle tracking error of attitude and angular velocity 64
6.24 Feedback delay for single obstacle trajectory 65
6.25 NMPC performance with single obstacle trajectory 65
6.26 Single obstacle trajectory execution control inputs 66
6.27 3D visualization of single obstacle trajectory execution 66
6.28 Single obstacle single Space CoBot with payload generated trajectory 67
6.29 Single obstacle single Space CoBot with payload tracking error of position and linear velocity 68
6.30 Single obstacle single Space CoBot with payload tracking error of attitude and angular velocity 68
6.31 NMPC performance with single obstacle single Space CoBot with payload trajectory . . . 68
6.32 Single obstacle single Space CoBot trajectory execution control inputs 69
6.33 3D visualization of single obstacle single Space CoBot with payload trajectory execution . 69
6.34 Two obstacles problem simulation environment. 69
6.35 Two obstacles single Space CoBot generated trajectory 70
6.36 Two obstacles single Space CoBot tracking error of position and linear velocity 71
6.37 Two obstacles single Space CoBot tracking error of attitude and angular velocity 71
6.38 NMPC performance with two obstacles single Space CoBot trajectory 72
6.39 Feedback delay for two obstacles single Space CoBot trajectory 72
6.40 3D visualization of two obstacles single Space CoBot trajectory execution 72
6.41 Formation problem simulation environment 73
6.42 Approach in formation tracking error of position and linear velocity for both Space CoBot vehicles 74
6.43 Approach in formation tracking error of attitude and angular velocity for both Space CoBot vehicles 75
6.44 NMPC performance with approach in formation trajectory for both Space CoBot vehicles . 75
6.45 Formation payload transport tracking error of position and linear velocity for both Space CoBot vehicles 76
6.46 Formation payload transport tracking error of attitude and angular velocity for both Space CoBot vehicles 76
6.47 Controller performance with generated trajectory 77
6.48 Single obstacle formation with payload trajectory execution control inputs 77
6.49 Feedback delay for single obstacle formation with payload trajectory 78

6.50 3D visualization of full formation trajectory execution 78

A.1 Comparison between attitude error vectors 89
A.2 Comparison between attitude error magnitude 89

Nomenclature

Greek symbols

$\boldsymbol{\alpha}$	Angular acceleration vector.
$\boldsymbol{\omega}$	Angular velocity vector.
$\Delta\boldsymbol{\Theta}$	Attitude error vector.
Δ	Interval or variation.
λ	Lagrange multiplier.
μ	Penalty multiplier.
Ω	Rotation velocity.
ϕ, θ, ψ	Euler angles.
ρ	Density.
τ	Rotor reaction torque.

Roman symbols

$\mathbf{p}$	Position vector.
$\mathbf{u}$	Control input vector.
$\mathbf{v}$	Linear velocity vector.
$\mathbf{x}$	State vector.
$\mathbf{a}$	Linear accelration vector.
$\mathbf{c}_{off}$	Space CoBot relative position to formation Center of Mass.
$\mathbf{F}$	Thrust force vector in body frame.
$\mathbf{J}$	Inertia matrix.
$\mathbf{M}$	Moment vector in body frame.
$\mathbf{Q}$	Intermediate quaternion multiplication matrix.

$\mathbf{q}$ Attitude quaternion.

$\mathbf{q}_e$ Quaternion error.

$\mathbf{R}$ Rotation matrix.

$\mathbf{W}_k$, $\mathbf{W}_N$, $\mathbf{Q}$, $\mathbf{Q}_f$, $\mathbf{R}_u$, Weighting matrices.

ℓ Cost function.

$\mathcal{A}$ Active constraints set.

$\mathcal{I}$, $\mathcal{B}$ Inertial and body frames.

$\mathcal{L}$ Lagrange function.

C_D Rotor drag constant.

C_M Rotor drag moment constant.

C_p Power coefficient.

C_R Rotor rolling moment constant.

C_T Rotor thrust constant.

C_t Thrust coefficient.

D Rotor diameter.

F_D Drag force.

F_T Thrust force.

fd Feedback delay.

G_i Equality constraints set.

H Objective function.

H_i Inequality constraints set.

I Identity matrix.

K_1, K_2 Rotor modelling constants.

m Mass.

M_D Drag moment.

M_R Rolling moment.

N Prediction horizon length.

t Time instant.

t_s Time step.

w_i, θ_i, ϕ_i Space CoBot design parameters.

Subscripts

0 Current instant.

i, j, k, n Computational indexes.

opt Optimal condition.

ref Reference condition.

u Predicted vector component.

x, y, z Cartesian components.

Superscripts

-1 Inverse.

$\perp$ Parallel.

opt Optimal condition.

* Conjugate.

T Transpose.

Acronyms

ALTRO	Augmented Lagrangian TRajectory Optimizer
BVP	Boundary Value Problem
CBC	Classical Backstepping Control
CIMON	Crew Interactive MObile companioN
CoM	Center of Mass
DDP	Differential Dynamic Programming
DDS	Data Distribution Service
DIRCOL	Direct Collocation
DIRTRAN	Direct Transcription
DLR	Deutsches Zentrum für Luft- und Raumfahrt
GNC	Guidance Navigation and Control
iLQR	Iterative Linear Quadratic Regulator
IP	Interior Point
ISR-Lisboa	Institute For Systems and Robotics
ISS	International Space Station
IVA	Intravehicular Activity
JAXA	Japan Aerospace Exploration Agency
KKT	Karush-Kuhn-Tucker Conditions
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
MDT	Multi-Directional Thrust

MHC	Moving Horizon Control
MPC	Model Predictive Control
MRAV	Multi-rotor Aerial Vehicle
NASA	National Aeronautics and Space Administration
NDI	Nonlinear Dynamics Inversion
NLP	Nonlinear Programming
NMPC	Nonlinear Model Predictive Control
OCP	Optimal Control Problem
ODE	Ordinary Differential Equations
PID	Proportial Integral Derivative
QP	Quadratic Programming
ROS	Robot Operating System
SDF	Simulation Description Format
SPHERES	Synchronized Position Hold Engage Reorient Experimental Satellites
SQP	Sequential Quadratic Programming
UAV	Unmanned Aerial Vehicle
URDF	Unified Robotic Description Format

Chapter 1

Introduction

This chapter will present the main objectives of this thesis as well as the motivation behind it. Firstly the motivation for this work is presented in Section 1.1. Section 1.2 sets the objectives of this thesis. In Section 1.3 the contributions of this thesis are enumerated. Lastly, the outline of the thesis is presented in Section 1.4 detailing its structure.

1.1 Motivation

In recent decades, the interest in Unmanned Aerial Vehicles (UAV) has experienced considerable increases. This is mainly due to the ability to execute actions without an onboard pilot, whether it is controlled remotely or by an autopilot system, commonly composed by several onboard elements that originate Guidance, Navigation and Control (GNC) systems [1]. This allows a vast range of applications that reduce operating costs, operations with no risk to human life or even tasks that would be otherwise impossible to manned vehicles. An even more recent trend is the use of small robots in the International Space Station (ISS). National Aeronautics and Space Administration (NASA) free-flyer robots are one example and they have been onboard the ISS for over a decade. Initially they were a platform used for many experiments from flight directors and researchers on the ground that aimed at improving formation flying and docking algorithms [2]. This research has since allowed the development of new free-flyer robots that include more features and can perform a variety of Intravehicular Activity (IVA) work in the ISS [3]. The Institute For Systems and Robotics (ISR-Lisboa) has also developed a highly maneuverable holonomic free-flyer robot designated Space CoBot [4, 5], which provides a similar platform for conducting research in this type of robots.

Space exploration has been experimenting a few changes in recent years as both private companies and public institutions are starting to make advancements towards deep space exploration. This endeavour will require a significant amount of research to make manned missions viable. One can imagine that infrastructures will also be necessary, such as refilling stations, or even temporary stations where crews can rest or perform repairs before continuing a journey. Using the ISS as reference, a station capable of housing humans is an incredible complex place that requires extensive monitoring and maintenance

to remain habitable [6]. It becomes apparent then the importance of developing capable autonomous free-flyer robots to perform these tasks in uninhabited stations.

Currently, robotic arms are already used to assist humans in maintenance tasks in the ISS, namely the *Canadarm2* [7]. However, in a remote space station there could be significant limitations to these systems. Free-flyer robots will allow more complex operations without human presence. In order to achieve this level of autonomy and unsupervised operation, several aspects have to be extensively studied, such as the vehicle interaction with the station, objects, tools, and astronauts, in the case of a manned station. Its navigation system must guarantee high awareness and risk-free maneuvers. Lastly, but definitely not less important, the control system must use the least amount of energy possible as access to energy might be severely restricted.

Almost all resources in space, such as propellant or fuel are limited. Solar power can also be limited as we stray away from the solar system, which limits electrical energy storage. Consequently, optimal navigation solutions are preferable. Model Predictive Control (MPC) is a well known field that aims to provide optimal control feedback laws, while respecting control bounds and any given constraints, for example, limitations to fuel consumption [8].

The two major motivations behind this thesis are then the desire to contribute to the improvement of the Space CoBot free-flyer robot while exploring the limits and applicability of MPC in the real-time context.

1.2 Objectives

The main objective of the present thesis is to integrate a Nonlinear Model Predictive Control (NMPC) controller for the Space CoBot and test its applicability when subjected to different situations.

1.2.1 Guidance, Navigation and Control

Typically, a flight system can be segmented in three distinct subsystems often denominated as Guidance, Navigation and Control subsystems. According to [9], the function of each subsystem can be classically defined by the determination of the desired state for the Guidance subsystem, the determination of the estimated state for the Navigation subsystem and the derivation of control commands for the Control subsystem. The focus of this thesis will be in Control and Guidance tasks. Therefore, no Navigation methods will be studied or implemented. This means that the resulting control architecture should not only control the actuation of the system, which comprises the Control task, but also find the best way of performing specific tasks, which will comprise the Guidance task. The aim is to find the limits of the applications of NMPC in the real-time operations.

Control

The controller takes on the task of controlling the actuators directly, representing a low-level controller.

A controller of this type must ensure a stable and fast computation time. It must be able to converge to the desired reference in a quick, robust and stable manner. The reference can originate from an attitude tracking task, a waypoint reference, a specific trajectory, among others. The behavior of the controller in these different situations should be analyzed.

Previous research was conducted regarding control behavior with an unmodeled payload within the Space CoBot project [4], which concluded that even though convergence to a given position reference is obtained, the addition of a heavy load causes a significant variation in attitude. This is undesirable and states the need for more robust control methods. This thesis aims to extend the previous work and introduce control solutions that take in consideration the model of the payload. Consequently, an additional objective is to explore the possibility of changing the considered models while operating. This means, for example, if a payload is to be carried by the Space CoBot system, from the point it grabs the payload, the controller considers it part of the system and not some external perturbation.

Guidance

In order to achieve autonomy, the controller has to be able to perform safe, constraint free maneuvers to achieve a given objective. This enters the realm of trajectory planning and obstacle avoidance. As this increases the complexity of the control problem, the controller is expected to struggle. The objective is to explore the NMPC limitations and observe when the controller becomes unreliable.

Environment constraints will result from adding bounds to the control inputs and state variables. Additionally, obstacles will be considered as additional space constraints.

If the NMPC becomes unreliable for a given problem complexity, additional methods should be used to compensate the NMPC limitations by separating the Control and Guidance tasks.

Ultimately, the resulting controller should be able to provide a base framework to continue NMPC research in the future.

1.2.2 Validation

Testing free-flyer vehicles is very limited as accessing a microgravity environment is difficult. Access to the ISS is not possible. Microgravity testbeds, such as parabolic flights are also very limited and would require a special setup in the airplane. Alternatively, it is possible to perform 2D motion validation using frictionless tables. One is being constructed by the ISR-Lisboa, however, it is not yet available. Therefore, creating an accurate simulation environment is extremely important as all tests will be conducted exclusively in simulation.

1.3 Contributions

In order to achieve the objectives mentioned in the previous section, several methods were implemented and analyzed during this thesis and they can be sectioned as follows:

- **Control:** Implementation of a real-time NMPC controller that can operate with different model configurations involving free-flyer robots.
- **Guidance:** Implementation of a Trajectory Optimization method for offline trajectory generation capable of handling several constraints including obstacle avoidance.
- **Simulation:** Development of a simulation composed by the RotorS simulator integrated within Gazebo; Integration of tools to provide grasping actions.
- **Experiments:** Elaboration of several experiments with varying conditions, in a simulation environment.

1.4 Thesis outline

The following work presented in this thesis is structured as follows: First, through Chapter 2 different control solutions to nonlinear problems are briefly explored in order to understand how the control method selected for this work can improve or contribute to the field. A brief review on trajectory optimization methods and available simulators is also conducted in this Chapter. Following, in Chapter 3, the theoretical background behind this work is presented in detail in order to provide insight on the topics discussed in this thesis. Chapter 4 provides a deeper exploration on how the selected algorithms were tailored to this project. Chapter 5 presents detailed information regarding the implementation of all subsystems involved in this work. In Chapter 6, the obtained results are demonstrated and discussed. Finally, in Chapter 7 the objectives of this thesis are evaluated and possible future work improvements are discussed.

Chapter 2

State-of-Art

In this chapter, a brief review is conducted on available material regarding Multi-rotor Aerial Vehicles (MRAV) control methods with the objective of understanding where the present work could be beneficial for free flyer platforms. In Section 2.1, existing free flyer platforms are introduced to understand the importance and applicability of this type of vehicles. In Section 2.2 a general review of existing control methods used in nonlinear systems is performed to a allow a better understanding of how NMPC is relevant in this field. Section 2.3 extends the base of knowledge on optimization based methods used in the trajectory optimization field. Finally in Section 2.4 available simulators are briefly analyzed in order to obtain a good approximation to a real micro-gravity environment.

2.1 Free flying robots

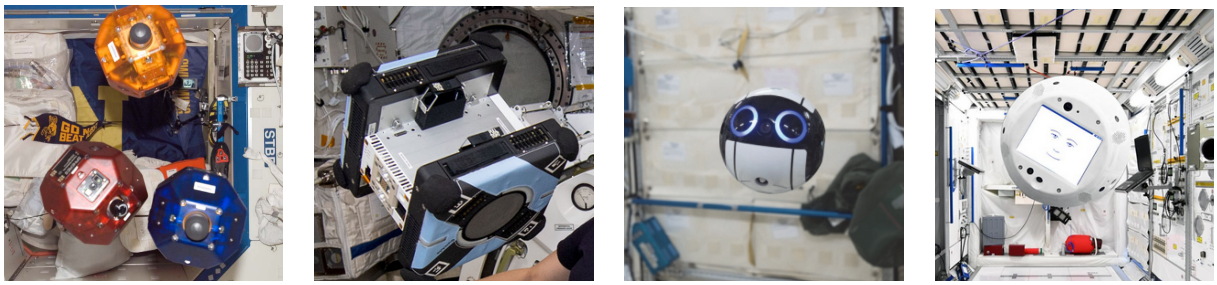
To the author's knowledge, the free-flyer robots that are or have been onboard the ISS are depicted in Figure 2.1.

NASA has been using free flyers as research platforms on board the ISS as early as 2006 through Synchronized Position Hold Engage Reorient Experimental Satellites (SPHERES) [2]. SPHERES is a formation flight testing facility consisting of three satellites operating onboard the ISS. During its operation, it used the microgravity environment to further develop formation flight and docking algorithms, fault detection and path planning in an incremental manner. However, this facility lacked a truth sensor of higher fidelity. SPHERES tests had to resort to at least two perpendicular mounted cameras so that researchers could have truth sensing of 3D operations [2].

Astrobee [3] is a similar platform that replaced SPHERES as a testbed onboard the ISS. Astrobee is expected to increase and improve SPHERES features, specially when it comes to navigation as these robots have better sensors such as depth cameras, color cameras, several processors and a collections o mapping and planning algorithms [10]. They are also equipped with a robotic arm that allow interactions with its surroundings, such as cargo transfers and maintenance tasks. A major advantage to researchers is the fact that the software in Astrobee is largely implemented within Robot Operating System (ROS) which facilitates integrations.

Japan Aerospace Exploration Agency (JAXA) has developed Int-Ball [11], which is able to move autonomously inside the ISS. It is mainly used for shooting image and video. It is particularly useful for compensating the blind spots that wall mounted cameras have.

German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt) (DLR) has also developed a free flyer designated Crew Interactive MObile companioN (CIMON) [12] which is currently on the second iteration, CIMON-2, and is the latest addition to the ISS as it is onboard as of 2020. The project aims to develop autonomous astronaut assistance through artificial intelligence. It is able to see, hear, understand, speak and fly. One of the main objectives of the project is to assist astronauts during high workloads in order to reducing their exposure to stress [12].



(a) SPHERES (b) Astrobee (c) Int-Ball (d) CIMON

Figure 2.1: Existing free-flyers. Credits from left to right: [2], NASA / Anne McClain, [11], DLR/T. Bourry/ESA.

2.2 Nonlinear Control

A controller is the element in a controlled system that commands or regulates the behavior of the system through the interaction with its sensors and actuators.

2.2.1 Linear methods

Linear control methods resort to approximate linearized models of the system whereas nonlinear methods can use the the complete dynamics model. One of the most widely used linear controllers is the Proportial Integral Derivative (PID) controller [13]. Linear Quadratic Gaussian (LQG) control, particularly the Linear Quadratic Regulator (LQR) comprise several popular optimal control techniques which can be used to achieve stable feedback control laws or implemented with several other control architectures such as in [14]. Gain scheduling is used to extend the capabilities of linear controllers. Nonlinear systems are often modeled as a set of simplified linear models, representing different operating points where the gains of the feedback control laws are tuned accordingly as in [15].

These methods are common but often limit the operation of MRAV to simple maneuvers and basic flight.

2.2.2 Nonlinear methods

To overcome some of the limitations of linear approaches, there is a broad list of techniques that resort to the nonlinear model of the system. Nonlinear Dynamics Inversion (NDI), also known as feedback linearization, comprises a group of methods that allow the use of linear tools to control a nonlinear system, as in [16]. Adaptive control is a very common method used to improve robustness of a variety of controller architectures that range from PID to NDI controllers [17]. Backstepping is a well-known recursive methodology for nonlinear systems. Generally, this method is based in Lyapunov theory, which is referred to as Classical Backstepping Control (CBC). However, several other methods have been proposed based on backstepping [18].

Also, as presented in [16], a control scheme may use different control methods to different state components.

When compared with each other, all these solutions present advantages and disadvantages regarding stability, robustness and resistance to model uncertainties, which will depend on the objective of the design. However, when handling operational constraints is a requirement, it is often necessary to augment the controller design or redesign it entirely. Anti-windup compensation [19], the augmentation of Lyapunov controllers with barrier functions [20] and reference and command governors [21] are some examples of how a controller can be augmented.

Constraints in real-world applications can take several forms that go beyond obvious actuator limits and stability constraints. They can be bounds imposed on process variables to enforce safety and efficiency, collision avoidance requirements, limited energy usage, possible failures that can reduce the functionality of a system or even time constraints.

2.2.3 Learning methods

The main characteristic of this control scheme is that a mathematical model of the system dynamics is not used. Instead, flight data is used to train a model that will represent the system. Popular methods rely on the use of Artificial Neural Networks architectures, where a model is trained trough simulation data [22]. Another learning method is the Human-based approach, where a pilot's execution of complex maneuvers is used as training data for the model [23].

2.2.4 Optimal Control methods

Optimal control aims to find an optimal control sequence for a given system such that a certain optimality criterion is achieved. Optimality is defined as the minimization or maximization of an objective function without violating any specified constraints [24]. An Optimal Control Problem (OCP) includes a cost function, containing state and control variables and a set of variables that are used to minimize the cost function, denominated optimizations variables, which are usually the control inputs of the system. These methods are often used in more complex problems with several constraints such as the rendezvous of two satellites with limited fuel or time as in [25] or systems composed by different elements such as a

satellite with a robotic arm as in [26]. Optimal control can be applied to linear or nonlinear systems. However, the implementation and algorithms involved vary significantly.

2.2.5 Model Predictive Control

MPC is an extension of optimal control methods, which consists in solving an OCP repeatedly at each time step. The feedback control law is obtained by applying the first element of the solution at each time step to the system and discarding the rest. Controllers designed with this architecture use a linear approximation of the system around an equilibrium point as presented in these works [27, 28]. By computing a new solution at each time step, linear MPC is resistant to possible perturbations and model uncertainty.

Relying on an optimization stage at each time step, MPC controllers are very dependent on the complexity of the system and computational power available, making these kind of controllers ideal for slow evolving systems, such as chemical industrial processes as in [29] or in biological processes as in [30].

In recent years, improvements in processing speeds and developments of new numerical optimization methods are allowing MPC to be considered in faster dynamic systems, such as an UAV as in [31].

2.2.6 Nonlinear Model Predictive Control

Similarly to MPC, NMPC relies on a model of the system in order to predict its evolution along a given prediction horizon. However, the full dynamic system is considered instead of a linearized model, which increases the control problem complexity significantly. When it comes to the optimization problem formulation, that is when the major differences between the two approaches become clearer. NMPC cannot rely on analytical control laws. Alternatively, it relies on discretization techniques that transform the OCP into a standard Nonlinear Programming (NLP) problem that can be solved through general NLP solvers [32]. There are two main families of NLP solvers, Interior Point (IP) and Sequential Quadratic Programming (SQP) solvers.

Currently, there is a wide variety of algorithms available based in both solver families. Recent developments are briefly presented and compared in [33], where an effort to close the gap to the real-time frame is visible. Methods that are worth mention for their extensive presence in literature as benchmark methods are the SQP based SNOPT [34] and the IP based IPOPT [35]. Many performance studies of these methods have been performed regarding computation times and solution quality while solving several tasks, such as simple steps, obstacle avoidance and operation under state bounds constraints [36]. This provides information on what to expect from these types of solvers. Additionally, to close the gap to real-time applications, variants of SQP and IP methods that take the different approach of finding approximate solutions to the NLP by stopping the process prematurely at a given time instead of iterating until convergence have been proposed in [37, 38].

NMPC is very versatile as it can take several roles within the control system.

Guidance tasks such as trajectory planning as in [39], where a NMPC is proposed to adjust a trajectory in the presence of obstacles by incorporating new constraints during operation.

Interestingly, NMPC are being considered to perform low level control tasks, which requires fast and stable rates of operation.

The hierarchical controller implemented in [40] uses a NMPC controller for the attitude of a MRAV and registered execution averages of $1\,ms$ while operating at a $5\,ms$ sampling time. Although it only controls the attitude component of the system, it represents a very reasonable computation delay.

In [41], a single non-hierarchical real-time NMPC controller implemented in a fixed wing aircraft is used to control both position and attitude for tracking a given reference trajectory while avoiding obstacles and respecting actuator bounds. The execution time required by the NMPC controller averages at $30\,ms$ while operating at a sampling time of $500\,ms$. This can be an indication of the impact of system complexity on execution time.

Also implemented in a MRAV, in [42], a hierarchical controller is used to track a feasible trajectory generated offline. A position NMPC controller computes attitude references feeding a second attitude NMPC controller, operating with sampling times of $200\,ms$ and $2\,ms$, respectively.

The application of NMPC in a MRAV with Multi-Directional Thrust (MDT) capability is studied in [43]. A NMPC controller is used to compute force inputs for each rotor to track both position and attitude references.

2.3 Trajectory Optimization

Trajectory optimization refers to a set of methods that are used to find the best choice of trajectory, usually by selecting the control inputs to the system, as functions of time [44]. In this definition, a trajectory is composed by a sequence of states or control inputs during a given time interval, starting from an initial state and ending in a desired state.

In this type of optimization, numerical algorithms solve variations of the same OCP mentioned in the previous section. However, the focus in this field is to obtain feasible trajectories instead of control inputs to apply directly to the system.

These problems can be solved with direct methods that discretize state and control trajectories in order to formulate a NLP problem as mentioned in Section 2.2.6, which tend to be versatile and robust. Common direct methods are Direct Transcription (DIRTRAN) [36] and Direct Collocation (DIRCOL) [45], which comprise different methods to transcribe the dynamics into the NLP formulation. However, the versatility and somewhat faster computations come at the cost of less accurate solutions and possible losses of dynamic feasibility. Additionally, direct methods treat both states and control variables as optimization variables which for complex system can increase the size of the NLP significantly.

Alternatively, indirect methods often treat only control inputs as optimization variables, while implicitly enforcing dynamic constraints through the system's simulation [46]. This means that in a predicted trajectory, any given state is the result of a system simulation using the previous state and respective control input. This implies that solutions obtained though these methods will be dynamically feasible.

The solvers used in this approach are highly conditioned by the initial guess. When dealing with complex systems or problems, finding solutions becomes more difficult and may require significant computation time to achieve a feasible solution.

Differential Dynamic Programming (DDP) [47] and iterative Linear Quadratic Regulator (iLQR) [48] are two indirect methods that solve the OCP by breaking it into smaller subproblems. While these methods are fast, they tend to be less robust and due to their strict enforcement of dynamic feasibility, it is often difficult to find a control sequence that produces a reasonable initialization of the problem.

An initial guess generation approach has been proposed in [49] that aims at decreasing the sensitivity of the initial guess. This could be applied to both direct and indirect methods.

Augmented Lagrangian TRajectory Optimizer (ALTRO) [46] comprises a few solvers for constrained trajectory optimization problems that aims to achieve the advantages of direct and indirect methods. It combines the introduction of slack variables that provide a relaxation of the initial problem, allowing initializations with unfeasible trajectories. An Augmented Lagrangian combined with an iLQR to quickly compute a first approximate solution and a line-search method, that takes the approximate solution and enforces the satisfaction of the dynamics and constraints. This solver is then an hybrid approach with fast computation speeds, numerical robustness, constraints handling and initialization with infeasible state trajectories. Additionally, all of these methods can be used separately.

2.4 Simulation

Performing tests with free flyers in a microgravity environment is a rather difficult task. Therefore, a good simulator is extremely important in the development of these vehicles.

The RotorS simulator [50] contains tools that allow the modeling, control and simulation of MRAV. It is a Gazebo based open source simulator implemented within the ROS environment. This package makes it relatively easy to create, test, and modify MRAV models. Furthermore, RotorS provides a fairly accurate modeling of aerodynamic elements present in MRAV, and provides the simulation of several onboard sensors, example controllers and simulation scenarios.

The NASA Astrobee Robot Software and Simulator [10] are available to the public. Its simulator is also Gazebo based running a custom propulsion system and sensors. It uses a combination of ROS, for onboard communication and Data Distribution Service (DDS), for remote messaging. This package also provides a variety of navigation and localization methods to experiment with.

As RotorS is easier to get acquainted with, it can be viewed as a first choice to develop control methods while aiming to switch to the Astrobee simulator, given that both simulators are based Gazebo within the same ROS framework.

Chapter 3

Background

In this chapter, the background theory regarding the topics required to understand the remainder of this thesis is presented. In Section 3.1, the general OCP is presented and common optimization methods are discussed. In Section 3.2, the optimization methods considered to solve general NLP problems with interest to NMPC are presented with detail. In Section 3.3, the theory regarding NMPC formulation is presented.

3.1 Optimal control methods

Optimization methods are one of the best ways to operate complex dynamic systems while satisfying certain optimality criterion. These methods allow the formulation of an OCP that can be solved by numerical and optimization solvers. Optimality is defined as the minimization or maximization of an objective function, that can be defined arbitrarily to describe the optimal solution, without violating any specified constraints. The optimal state is achieved through the selection of optimal values for the optimization variables, which are normally composed by control inputs to the system [24].

In general, an objective function includes two terms: a boundary objective J and a path integral along the entire optimization horizon, visible in (3.1). A problem with both terms is said to be in *Bolza form*. A problem with only the integral term is said to be in *Lagrange form*, and a problem with only a boundary term is said to be in *Mayer form* [44].

The terms optimization variables or decision variables are used to describe the variables that the optimization solver adjusts to minimize the objective function. These can be the initial and final time, t_0 and t_f, as well as state and control variables, $\mathbf{x}(t)$ and $\mathbf{u}(t)$, respectively [44]. The set of optimization variables will be henceforth designated by $\mathbf{z} := (t_o, t_f, \mathbf{x}(t), \mathbf{u}(t))$.

The optimization is subject to a variety of limits and constraints, namely dynamic constraints (3.2), path constraints (3.3), and boundary constraints (3.4).

This concludes the formulation of a general continuous-time OCP

$$\min_{t_0,t_f,\mathbf{x}(t),\mathbf{u}(t)} \quad F_J = J(t_f,\mathbf{x}(t_f)) + \int_{t_0}^{t_f} \ell(\mathbf{x}(t),\mathbf{u}(t),t)dt \tag{3.1}$$

$$\text{subject to} \quad \dot{\mathbf{x}} = f(\mathbf{x}(t),\mathbf{u}(t),t), \tag{3.2}$$

$$h(\mathbf{x}(t),\mathbf{u}(t),t) \leq 0 \tag{3.3}$$

$$g(t_0,t_f,\mathbf{x}(t_0),\mathbf{x}(t_f)) \leq 0 \tag{3.4}$$

There are several ways to approach an OCP. The two main approaches are indirect and direct methods.

Indirect Methods use the necessary conditions of optimality of the continuous problem to derive a Boundary Value Problem (BVP) in Ordinary Differential Equations (ODE) [51]. This BVP must be numerically solved, and the approach is often viewed as "first optimize, then discretize". The class of indirect methods encompasses the well known calculus of variations and the Pontryagin Maximum Principle. These methods often treat only control inputs as optimization variables while implicitly enforcing dynamic constraints through the system's simulation from the beginning to the end using an integrator [46]. This propagation is highly dependent on the initial state value. For this reason, producing reasonable initializations, and consequently finding a solution is often difficult. Errors introduced in the beginning of a trajectory also tend to propagate along the prediction horizon.

Direct methods transform the original infinite OCP into a finite NLP. This NLP is then solved by variants of numerical optimization methods [24]. It is often viewed as "first discretize, then optimize", which is an intuitive way of distinguishing direct and indirect methods. All direct methods are based on a finite dimensional parameterization of the control trajectory, but differ in the way the state trajectory is handled [51]. Nowadays, direct methods are the most widespread and successfully used techniques, and will be the focus of the next section.

3.2 Nonlinear Programing

The interest when using optimization methods is to find global minimizers. However, for general nonlinear problems, and in particular nonconvex problems, such global minimizers are very hard to find in practice, as it is very unpractical to evaluate the objective function F_J in all its definition space. Nevertheless, general optimization methods are able to find local minimizers [32]. For this reason, good initial guesses are often required to achieve good solutions.

3.2.1 Newton's method

Essentially all numerical methods for solving nonlinear optimization problems resort to some type of iteration with a finite set of unknowns. The fundamental approach to most iterative schemes was suggested by Newton over 300 years ago [52]. Therefore, a brief overview of this approach is essential to understand how nonlinear solvers work and how a problem can be formulated.

Following [52], assuming the objective of solving the nonlinear algebraic equations $\mathbf{a}(\mathbf{z}) = 0$, $\mathbf{a} \in \mathbb{R}^m$ for the root $\mathbf{z}^*$, starting from an estimate $\mathbf{z}$, $\mathbf{z} \in \mathbb{R}^n$, a new estimate $\bar{\mathbf{z}}$ can be computed according to

$$\bar{\mathbf{z}} = \mathbf{z} + \alpha \Delta \mathbf{z} \tag{3.5}$$

where α is a scalar step length and $\Delta \mathbf{z}$ is called the search direction, which is computed by solving the linear system

$$\nabla_z \mathbf{a}(\mathbf{z}) \Delta \mathbf{z} = -\mathbf{a}(\mathbf{z}) \tag{3.6}$$

where $\nabla_z \mathbf{a}(\mathbf{z})$ is a $m \times n$ matrix of first derivatives of $\mathbf{a}(\mathbf{z})$.

Depending on the considered optimal problem, the conditions that define an optimal solution can vary. These conditions are used to provide information about the problem to the solver.

3.2.2 Unconstrained problem

In unconstrained problems, local extrema points can be found in stationary points, where the first derivative of the objective function or the matrix of first derivatives, called Jacobian matrix $\nabla_z F_J(\mathbf{z})$, is equal to zero.

$$\nabla_z F_J(\mathbf{z}) = 0 \tag{3.7}$$

Additionally, considering $\mathbf{z}^*$ a local minimizer of F_J, to distinguish minimum points from maximum points, the second derivative of the objective function or the matrix of second derivatives, called Hessian matrix $\nabla^2_{zz} F_J(\mathbf{z})$, needs to be positive definite in an open neighborhood of $\mathbf{z}^*$ [32]. This ensures that for a vector $\mathbf{d} \in \mathbb{R}^{n_z}$ with $||\mathbf{d}|| < r$, where r is a radius such that $\nabla^2_{zz} F_J(\mathbf{z})$ is positive definite for all $\mathbf{z} \in \{\mathbf{z} \,|\, ||\mathbf{z} - \mathbf{z}^*|| < r\}$, the following holds

$$F_J(\mathbf{z}^* + \mathbf{d}) > F_J(\mathbf{z}^*) \tag{3.8}$$

These are the necessary and sufficient conditions to consider a given local minimizer $\mathbf{z}^*$ as an optimal local minimum $\mathbf{z}^{opt}$ [32].

3.2.3 Constrained problem

In problems with constraints, optimal points can be found by the Lagrangian multiplier method, which augments the objective function F_J to take in consideration the equality and inequality constraints

$$G(\mathbf{z}) = 0 \tag{3.9}$$

$$H(\mathbf{z}) \geq 0 \tag{3.10}$$

for simplicity both constraints are combined into C via

$$C(\mathbf{z}) := \begin{bmatrix} G(\mathbf{z}) \\ H(\mathbf{z}) \end{bmatrix} \tag{3.11}$$

and the Lagrangian is then given by

$$\mathcal{L}(\mathbf{z}, \boldsymbol{\lambda}) = F_J(\mathbf{z}) - \boldsymbol{\lambda}^T C(\mathbf{z}) \tag{3.12}$$

where $\boldsymbol{\lambda}$ are Lagrange multipliers. Following [52], necessary conditions for the point $(\mathbf{z}^*, \boldsymbol{\lambda}^*)$ to be a constrained optimum point require finding a stationary point of the Lagrangian that is defined by

$$\nabla_z \mathcal{L}(\mathbf{z}, \boldsymbol{\lambda}) = \nabla_z F_J(\mathbf{z}) - \nabla_z C(\mathbf{z})^T \boldsymbol{\lambda} = 0 \tag{3.13}$$

and

$$\nabla_\lambda \mathcal{L}(\mathbf{z}, \boldsymbol{\lambda}) = -C(\mathbf{z}) = 0 \tag{3.14}$$

These first-order necessary optimality conditions are usually called Karush-Kuhn-Tucker (KKT) conditions [52].

It is important to note that a generalization of constrained problems occurs when inequality constraints $H(\mathbf{z}) \geq 0$ are imposed. For a given minimizer $\mathbf{z}^*$, the constraints will fall into one of two classes. Strictly satisfied constraints, $H_i(\mathbf{z}^*) > 0$, which are called inactive constraints. The remaining active constraints are on their bounds, $H_i(\mathbf{z}^*) = 0$ [52]. Constraints restrict the feasible direction of the next iterate $\bar{\mathbf{z}}$ in (3.5). All equality constraints restrict feasible directions. However, not all inequality constraints restrict feasible directions. Only when a given minimizer $\mathbf{z}^*$ implies $H_i(\mathbf{z}) = 0$ are feasible directions restricted by inequality constraints [32]. The *active set* $\mathcal{A}(\mathbf{z})$ of any given feasible point $\mathbf{z}$ consists of the equality and inequality constraints that restrict the feasible directions of that particular point. If $\mathcal{A}$ is known, then the remaining inactive constraints can be ignored and the problem can be solved as an equality constrained problem. However, as constraints can become active of inactive, an algorithm needs to compute $\mathcal{A}$, which is not a trivial problem [52].

Apart from this generalization regarding inequality constraints, the same first-order optimality conditions (3.13) and (3.14) apply to equality and inequality constrained problems [52].

3.2.4 Direct Multiple Shooting

There are several direct methods to reformulate the infinite OCP into a finite NLP and each one can have variants or differ in terms of implementation. Therefore, for brevity only Direct Multiple Shooting will be detailed as it presents the most relevance for this work.

According to [32] a NLP problem takes the following form

$$\underset{z}{\text{minimize}} \quad F_J(z) \tag{3.15}$$

$$\text{subject to} \quad G(z) = 0, \tag{3.15a}$$

$$H(z) \geq 0 \tag{3.15b}$$

with maps $F : \mathbb{R}^{n_z} \to \mathbb{R}$, $G : \mathbb{R}^{n_z} \to \mathbb{R}^{n_g}$ and $H : \mathbb{R}^{n_z} \to \mathbb{R}^{n_h}$. The function $G(z)$ comprises all equality constraints, the function $H(z)$ comprises all inequality constraints and z represents the set of the optimization variables. Similarly to previous sections, the function $F_J(z)$ is a cost function.

Here and in the following, state and control constraints sets will be characterized by a set of functions $G_i^S : X \times U \to \mathbb{R}$, $i \in \mathcal{E}^S = \{1, \dots, p_g\}$, and $H_i^S : X \times U \to \mathbb{R}$, $i \in \mathcal{I}^S = \{p_g + 1, \dots, p_g + p_h\}$ via the equality and inequality constraints presented in (3.15a) and (3.15b).

In order to reformulate an OCP into a NLP, the dynamics need to be transformed into equality and inequality constraints, via a discretization method such as multiple shooting or full discretization [32]. Note that even though discrete time is considered, within this context, the continuous time system equations are used which are mapped by f in (3.2). This map is obtained through numerical integration methods. This mapping is used to compute the predicted trajectory x_u.

Evidently, the control input $u(k)$ will be an optimization variable in the NLP. The multiple shooting discretization includes some components of the state vector $x_u(k, x_0)$, at certain k time instants, as independent optimization variables in the problem, as opposed to the full discretization method which would include all $x_u(k, x_0)$ elements at all $k \in \{0, \dots, N-1\}$ in the NLP [32]. These new variables are called *shooting nodes* and the respective times will be called *shooting times*. In the formal description, the vector of multiple shooting nodes will be defined by $s := (s_1, \dots, s_{r_s}) \in \mathbb{R}^{r_s}$ where s_i is the ith multiple shooting node. The shooting times $\varsigma : \{1, \dots, r_s\} \to \{0, \dots, N\}$ and indices $\iota : \{1, \dots, r_s\} \to \{1, \dots, d\}$ define the time and component of the state vector x_u corresponding to s_i through

$$x_u(\varsigma(j), x_0)_{\iota(j)} = s_j \tag{3.16}$$

This implies that the components $x_u(k, x_0)_i, i = 1, \dots, d$ are determined by the iteration

$$x_u(k+1, x_0)_i = s_j \tag{3.17}$$

if there exists $j \in \{1, \dots, r_s\}$ with $\varsigma(j) = k+1$, $\iota(j) = i$ and

$$x_u(k+1, x_0)_i = f(x_u(k, x_0), u(k))_i \tag{3.18}$$

with initial condition $x_u(0, x_0)_i = s_j$ if $j \in \{1, \dots, r_s\}$ with $\varsigma(j) = 0$ and $\iota(j) = i$, or $x_u(0, x_0)_i = (x_0)_i$, otherwise. The shooting nodes s_i become part of the set of optimization variables z, which reads

$$z := (u(0), \dots, u(N-1), s^T)^T \tag{3.19}$$

To ensure that the NLP solution leads to values of s_i for which the state vector $x_u(k, x_0)$ then defined forms a trajectory of (3.26). To achieve this, following [32] a continuity condition is defined for all shooting nodes s_j with $\varsigma(j) \geq 1$ as

$$s_j - f(x_u(\varsigma(j) - 1, x_0), u(\varsigma(j) - 1))_{\iota(j)} = 0 \tag{3.20}$$

and for all shooting nodes s_j with $\varsigma(j) = 0$ as

$$s_j - (x_0)_{\iota(j)} = 0 \tag{3.21}$$

These constraints are included as equality constraints in the NLP (3.15). Its final formulation is then

$$\text{minimize} \quad F_J(z) := \sum_{k=0}^{N-1} W_k \ell(n + k, x_u(k, x_0), u(k)) + W_N \ell_N(n + N, x_u(N, x_0)) \tag{3.22}$$

$$\text{with respect to} \quad z := (u(0), \dots, u(N-1), s) \in \mathbb{R}^{n_z} \tag{3.22a}$$

$$\text{subject to} \quad G(z) = \begin{bmatrix} \left[G_i^S(x_u(k, x_0), u(k))\right]_{i \in \mathcal{E}^S, k \in K_i} \\ \left[s_j - f(x_u(\varsigma(j) - 1, x_0), u(\varsigma(j) - 1))_{\iota(j)}\right]_{j \in \{1,,,,r_s\}, \varsigma(j) \geq 1} \\ \left[s_j - (x_0)_{\iota(j)}\right]_{j \in \{1,\dots,r_s\}, \varsigma(j) = 0} \end{bmatrix} = 0 \tag{3.22b}$$

$$H(z) = \left[H_i^S(x_u(k, x_0), u(k))\right]_{i \in \mathcal{I}^S, k \in K_i} \geq 0 \tag{3.22c}$$

where W is a weighting matrix. The index sets K_i, $i \in \mathcal{E}^S \cup \mathcal{I}^S$ in these constraints indicate that some of the conditions may not be applied to all times $k \in \{0, \dots, N\}$. These constraints induce the feasible set Ω described by

$$\Omega = \{z \,|\, G_i(z) = 0, i \in \mathcal{E}; H_i(z) \geq 0, i \in \mathcal{I}\} \tag{3.23}$$

where $z \in \Omega$ are called feasible points.

This means that any minimizer z_{opt} of the NLP must be an element of Ω by definition [32].

3.2.5 Sequential Quadratic Programming

To solve any NLP of the form (3.15) SQP are common approaches. Recalling the Lagrangian function $\mathcal{L}$ in (3.12), the objective of methods based in SQP is to iteratively approximate $\mathbf{z}$ and $\boldsymbol{\lambda}$ to $\mathbf{z}_{opt}$ and $\boldsymbol{\lambda}_{opt}$ such that the KKT conditions (3.13) and (3.14) hold [51].

A SQP method achieves this by applying the Newton's method to the KKT conditions to formulate an approximation of the nonlinear problem around the current iterate, which will be denoted by $\mathbf{z}_k$ [32]. The objective is to iterate

$$\mathbf{z_{k+1}} = \mathbf{z}_k + \alpha_k \Delta \mathbf{z}_k \tag{3.24}$$

where $\mathbf{z}$ represents the vector of optimization variables defined in (3.19), α_k is the step length and the

search direction $\Delta \mathbf{z}_k$ is the solution of the Quadratic Programming (QP) subproblem that results from the application of the Newton's method according to [32] and formulated by

$$\text{minimize} \quad F_{QP}(\mathbf{z}_k) + \nabla F_{QP}(\mathbf{z}_k)^T \Delta \mathbf{z}_k + \frac{1}{2} \Delta \mathbf{z}_k^T \nabla^2_{zz} \mathcal{L}(\mathbf{z}_k, \boldsymbol{\lambda}_k) \Delta \mathbf{z}_k \tag{3.25}$$

$$\text{with respect to} \quad \Delta \mathbf{z}_k \in \mathbb{R}^{n_z} \tag{3.25a}$$

$$\text{subject to} \quad \mathbf{C}_i(\mathbf{z}_k) + \nabla \mathbf{C}_i(\mathbf{z}_k)^T \Delta \mathbf{z}_k = 0 \quad \text{for all } i \in \mathcal{W}_k \tag{3.25b}$$

where $\mathcal{W}_k$ can be seen as an approximation to the active set $\mathcal{A}(\mathbf{z}_k)$ and is updated in each iteration step. This method can then handle inequality constraints.

3.3 Nonlinear Model Predictive Control

NMPC is an optimization based control method which provides feedback control laws for nonlinear systems. NMPC originated as an extension of optimal control theory to improve robustness and resistance to model uncertainties and has become a very popular technique for difficult control problems [24].

3.3.1 Basic algorithm

Given a system with state $x(n) \in X \to \mathbb{R}^d$, which is measured at discrete time instants t_n, $n = 0, 1, 2, ..., N-1$, it is considered a controlled system if at each time instant a control input $u(n) \in U \to \mathbb{R}^m$ can influence the future behavior of the state of the system. The general objective is to obtain the control inputs $u(n)$ such that $x(n)$ follows a given reference $x_{ref} \in X$ as good as possible [32]. The objective of model predictive control, linear or nonlinear, is to use a model of the process in order to predict and optimize the future system behavior. In this work, models will take the form

$$x(n+1) = f(x(n), u(n)) \tag{3.26}$$

where $f : X \times U \to X$ is a known and nonlinear map which assigns to a state $x(n)$ and a control value $u(n)$ the successor state $x(n+1)$ at the next time instant.

Achieving the optimal input sequence will be an iterative process. Starting from the current state $x(n)$, for a given control sequence $u(0), ..., u(N-1)$ with *horizon length* $N \geq 2$, the expression (3.26) is iterated in order to obtain the predicted trajectory x_u defined by

$$x_u(0) = x(n), \qquad x_u(k+1) = f(x_u(k), u(k)), \qquad k = 0, ..., N-1 \tag{3.27}$$

At this stage, a prediction for the behavior of the system is obtained on the discrete interval $t_n, ..., t_{n+N}$, based on the given control sequence u. Optimal control methods are then applied to determine the sequence $u(0), ..., u(N-1)$ such that x_u is as close as possible to x_{ref} [24]. To this end, there must be a function $\ell(x_u(k))$ that measures the difference between $x_u(k)$ and $x_{ref}(k)$ for $k = 0, ..., N-1$, that is denominated as the cost function. To allow penalization of control inputs, the cost function can be of the

form $\ell(x_u(k), u(k))$. In this case, both the distance from a reference state and reference control input are penalized. Following [32], a common choice for this purpose is the quadratic function

$$\ell(x_u(k), u(k)) = Q||x_u(k) - x_{ref}(k)||^2 + R||u(k)||^2 \tag{3.28}$$

where Q and R are weighting parameters for the state and control, respectively. A general form of an optimal control problem is the following

$$\text{minimize} \quad F_J(x_u(.), u(.)) := \sum_{k=0}^{N-1} \ell(x_u(k), u(k)) \tag{3.29}$$

Assuming that this optimal control problem has a solution, it is given by the minimizing control sequence $u_{opt}(0), ..., u_{opt}(N-1)$, i.e.,

$$\min_{u(0),...,u(N-1)} F_J(x(.), u(.)) = \sum_{k=0}^{N-1} \ell(x_{u_{opt}}(k), u_{opt}(k)) \tag{3.30}$$

To achieve the feedback form, $u(n)$ will be defined in feedback form as $u(n) = \mu(x(n))$ where μ maps the state $x \in X$ into the set U of control values. Obtaining the desired feedback value $\mu(x(n))$ is achieved by setting $\mu(x(n)) := u_{opt}(0)$, i.e., the first element of the optimal control sequence is applied as feedback. At the following instant t_{n+1}, the process is repeated from the prediction step, meaning that the remaining control inputs from the current solution, $u_{opt}(1), ..., u_{opt}(N-1)$, are discarded [32]. In conclusion, the feedback law is obtained by an online iterative optimization over the predictions generated by the model of the system (3.26).

At each time instant t, a prediction is generated for the following $N-1$ instants, maintaining the prediction horizon constant, meaning that the horizon is advancing through time. For this reason, MPC is often referred to as Moving Horizon Control (MHC) [32].

Chapter 4

Proposed Approach

In this chapter, the formulation of the selected methods to achieve the objectives of this work are explored. In Section 4.1, the models considered throughout this work, based in the Space CoBot, will be derived. In Section 4.2 the methods considered to formulate the NMPC controller integrated in this thesis are described. Lastly, in Section 4.3, the method used to perform guidance tasks, such as generating feasible trajectories for the NMPC is detailed.

4.1 Models

Throughout this thesis, three system configurations were considered. A single Space CoBot system, a Space CoBot carrying a payload system and a formation system comprising two Space CoBots and a shared payload. An additional formation configuration without payload is considered. However, the involved vehicles are described by the single Space CoBot dynamic model. The only difference is a transformation of the provided references, as will be detailed further below.

4.1.1 Space CoBot model

The Space CoBot is an hexarotor UAV with a special rotor design configuration, that can be visualized in Figure 4.1. All propellers are represented with a blue color with the exception of one represented by the color red and serves for visual reference of the attitude of the Space CoBot.

It is modelled based on the nonlinear dynamic model described in [5], which is defined through the Newton-Euler equations of motion. However, in this work quaternions are used to describe rotation instead of Euler angles. The quaternions approach is often used not only to avoid discontinuities and gimbal-locks but also to allow faster computations, which is essential when dealing with numerical

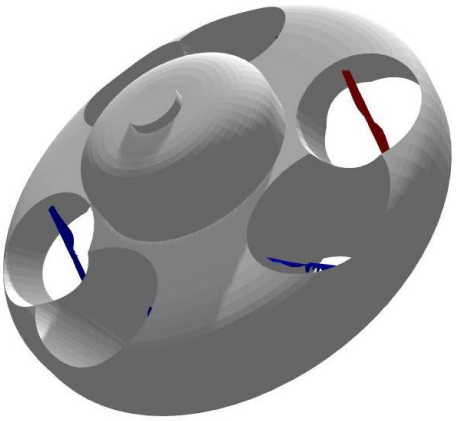


Figure 4.1: Space CoBot 3D model.

methods. The resulting model is described by

$$\begin{cases} \dot{\mathbf{p}} = \mathbf{v} \\ \dot{\mathbf{v}} = (1/m)\,\mathbf{R}(\mathbf{q})\,\mathbf{F} \\ \dot{\mathbf{q}} = (1/2)\,\mathbf{Q}(\mathbf{q})\,\boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}} = \mathbf{J}^{-1}(\mathbf{M} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}) \end{cases} \tag{4.1}$$

where $\mathbf{p} = [x\,, y\,, z]$ is the position, $\mathbf{v} = [u\,, v\,, w]$ corresponds to linear velocity, and $\mathbf{q} = [q_w\,, q_x\,, q_y\,, q_z]$ is the attitude quaternion, where q_w is a scalar and $[q_x\,, q_y\,, q_z]$ are vector components which represent the rotation axis. These components are described with respect to (w.r.t.) the inertial frame $\mathcal{I}$, and $\boldsymbol{\omega} = [\omega_x,\, \omega_y,\, \omega_z]$ is the angular velocity described w.r.t. the body frame $\mathcal{B}$. These elements constitute the state vector $\mathbf{x} = [\mathbf{p}, \mathbf{v}, \mathbf{q}, \boldsymbol{\omega}]$. The scalar m is the system's mass, $\mathbf{J}$ is the moments of inertia matrix and $\mathbf{F}$ and $\mathbf{M}$ are vectors representing the thrust force and torque generated by the rotors w.r.t. the body frame. $\mathbf{R} \in SO(3)$ is the rotation matrix obtained by considering right-hand rotation and defined by

$$\mathbf{R}(\mathbf{q}) = \begin{bmatrix} 1 - 2q_y^2 - 2q_z^2 & 2q_xq_y - 2q_zq_w & 2q_xq_z + 2q_yq_w \\ 2q_xq_y + 2q_zq_w & 1 - 2q_x^2 - 2q_z^2 & 2q_yq_z - 2q_xq_w \\ 2q_xq_z - 2q_yq_w & 2q_yq_z + 2q_xq_w & 1 - 2q_x^2 - 2q_y^2 \end{bmatrix} \tag{4.2}$$

and $\mathbf{Q}$ is a 4×3 matrix defined by

$$\mathbf{Q}(\mathbf{q}) = \begin{bmatrix} q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \\ -q_x & -q_y & -q_z \end{bmatrix} \tag{4.3}$$

Note that the dot product $\mathbf{Q}(\mathbf{q})\boldsymbol{\omega}$ results from a simplification step of the multiplication between a vector and a quaternion $\mathbf{q} \otimes \boldsymbol{\omega}$. Details regarding operations with quaternions are detailed in appendix A.1.

Propulsive system model

A common approach to model a UAV rotor is to consider the relationships that result from the momentum-blade theory [53],

$$C_t = \frac{T}{\rho\, n^2 D^4} \qquad \text{and} \qquad C_p = \frac{P}{\rho\, n^3 D^5} \tag{4.4}$$

where C_t is the thrust coefficient, C_p is the power coefficient, ρ represents the air density, D is the rotor diameter, n is the rotation velocity of the blade expressed in rps (revolutions per second), and T and P are the thrust and power, respectively. These two parameters are used to describe rotors once they are often used to present propeller performance test results. Obtaining this experimental information for a specific propeller can be a significant challenge. Alternatively, a performance study was conducted for small-scale propellers of different shapes and sizes in [54]. This allows the observation of a similar propeller and the use of approximations for the coefficients.

For modeling purposes, it is useful to describe the motor model as a function of thrust and torque. Therefore, starting from the expression (4.4) and considering that the power of a moving body, P, is expressed by $P = 2\pi\tau n$, where τ represents the generated torque, the following expressions are obtained

$$T = \rho D^4 C_t n^2 \qquad \text{and} \qquad \tau = \frac{\rho D^5 C_p}{2\pi} n^2 \tag{4.5}$$

Note that ρ, C_t and C_p are not constant parameters and vary with the propeller rotational speed and environment conditions such as pressure and temperature. However, the scenarios considered in this work do not involve high variations of environmental conditions. Furthermore, the relation C_p/C_t tends to present a small variation when the rotational speed suffers high variations according to [54]. Consequently, this relation can be considered constant without significant loss of accuracy. For simplicity sake, the constants K_1 and K_2 are defined by

$$K_1 = \rho D^4 C_T \qquad K_2 = \frac{\rho D^5 C_P}{2\pi} \tag{4.6}$$

Considering a single propeller i whose motor is rigidly linked to the body frame B, depicted in Figure 4.2, this rotor generates a scalar thrust f_i and a torque τ_i described by

$$f_i = K_1 u_i \qquad \tau_i = w_i K_2 u_i, \qquad i \in \{1, \dots, 6\} \tag{4.7}$$

where w_i is either -1 or 1 depending on whether the propeller rotates clockwise or anti-clockwise for a positive forward thrust $f_i > 0$. The actuation signal u_i is defined by $u_i = \text{sgn}(n_i)\, n_i^2$, where sgn() is a sign function, to achieve a linear relation between the actuation and forces/moments.

The orientation of a propeller w.r.t. the body frame B is represented by the unit vector $\hat{u}_i$. Consequently, the thrust $\mathbf{F}_i$ and total torque $\mathbf{M}_i$, caused by the displacement of thrust from the Center of Mass

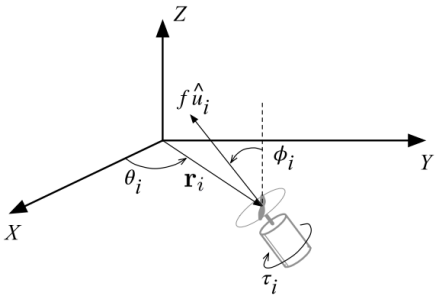


Figure 4.2: Single propeller notation. Figure from [4]

(CoM) and the propeller reaction torque $\boldsymbol{\tau}_i$ are defined by

$$\mathbf{F}_i = f_i \hat{u}_i \qquad \mathbf{M}_i = \mathbf{r}_i \times \mathbf{T}_i - \tau_i \hat{u}_i, \qquad i \in \{1, ..., 6\} \tag{4.8}$$

Each rotor is represented by a relative position to the CoM, expressed by the vector $\mathbf{r}_i$, orthogonal to the Z axis, and the unit vector $\hat{u}_i$, aligned with the propeller orientation axis. Both are uniquely defined by the angles θ_i and ϕ_i through the following expressions

$$\mathbf{r}_i = \begin{pmatrix} d\cos(\theta_i) \\ d\sin(\theta_i) \\ 0 \end{pmatrix} \qquad \hat{u}_i = \begin{pmatrix} \sin(\theta_i)\sin(\phi_i) \\ -\cos(\theta_i)\sin(\phi_i) \\ \cos(\theta_i) \end{pmatrix}, \qquad i \in \{1, ..., 6\} \tag{4.9}$$

where $d = ||\mathbf{r}_i||$ is the distance from the propeller to the CoM. All these elements can be visualized in Figure 4.2.

The resulting thrust $\mathbf{F}_i$ and torque $\mathbf{M}_i$ are obtained

$$\begin{pmatrix} \mathbf{F}_i \\ \mathbf{M}_i \end{pmatrix} = \mathbf{a}_i u_i \qquad \mathbf{a}_i = \begin{pmatrix} K_1 \hat{u}_i \\ K_1 \mathbf{r}_i \times \hat{u}_i - w_i K_2 \hat{u}_i \end{pmatrix}, \qquad i \in \{1, ..., 6\} \tag{4.10}$$

where $\mathbf{a}_i$ represents the components of force and torque of the i-th rotor described by

$$\mathbf{a}_i = \begin{pmatrix} K_1 \sin(\theta_i)\sin(\phi_i) \\ -K_1 \cos(\theta_i)\sin(\phi_i) \\ K_1 \cos(\phi_i) \\ [K_1 d\cos(\phi_i) - w_i K_2 \sin(\phi_i)]\sin(\theta_i) \\ -[K_1 d\cos(\phi_i) - w_i K_2 \sin(\phi_i)]\cos(\theta_i) \\ -K_1 d\sin(\phi_i) - w_i K_2 \cos(\phi_i) \end{pmatrix} \tag{4.11}$$

The resulting net force and torque will be the sum of the contributions of the 6 rotors. In matrix form

$$\begin{pmatrix} \mathbf{F} \\ \mathbf{M} \end{pmatrix} = \mathbf{A}\mathbf{u} \tag{4.12}$$

where $\mathbf{A} = [\mathbf{a}_1 \, ... \, \mathbf{a}_6]$ is a square matrix hereby called actuation matrix and $\mathbf{u} = [u_1 \, ... \, u_6]^T$ is the actuation input.

4.1.2 Space CoBot with payload model

One of the objectives of the Space CoBot project is to provide assistance to humans. To that end, the integration of a robotic arm can be considered to provide means of interaction with the environment. This integration is outside the scope of this thesis. Any addition to the Space CoBot system, such as an arm or a payload, will require modifications of the model defined in the previous section, due to the change in CoM. To redefine the model, one can resort to the *Parallel axis* theorem to compute the resulting inertial matrix around the CoM of the resulting system. In order to perform this modification, rigid body conditions are considered, meaning that the payload is rigidly linked to the Space CoBot. An illustration of the considered system can be observed in Figure 4.3 below.

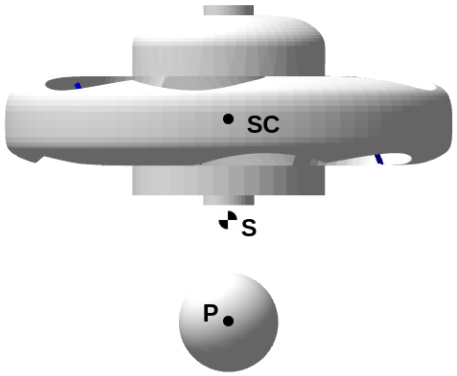


Figure 4.3: Space CoBot with payload configuration.

The new CoM represented by point S presents a displacement in the z axis only, in the body frame of reference. In general, the CoM can be found by vector addition of the weighted position vectors which point to the CoM of each object in a system. Therefore the components of the new CoM can be trivially calculated by

$$x_S = x_{SC} \tag{4.13}$$

$$y_S = y_{SC} \tag{4.14}$$

$$z_S = \frac{m_{space\ cobot} z_{SC} + m_{payload} z_P}{m_{space\ cobot} + m_{payload}} \tag{4.15}$$

Should the payload be asymmetrical, this computation has to be performed for every axis.

Following [55], the *Parallel axis* theorem states

$$\mathbf{J}_A = \mathbf{J}_G + m_{system} \left(\mathbf{r}_G^T \mathbf{r}_G \mathbf{I} - \mathbf{r}_G \mathbf{r}_G^T \right) \tag{4.16}$$

where $\mathbf{J}_G$ and $\mathbf{J}_A$ represent the system's moments of inertia matrices around the CoM and an arbitrary point A, respectively. The constant m_{system} represents the mass of the system, $\mathbf{I}$ is the identity matrix

and $\mathbf{r}_G$ is a vector representing the position of point G relative to point A.

As mentioned earlier, the objective is to obtain the total inertia of the system expressed around the CoM of the system, $\mathbf{J}_S$, which is defined by

$$\mathbf{J}_S = \mathbf{J}_{T1} + \mathbf{J}_{T2} \tag{4.17}$$

where $\mathbf{J}_{T1}$ and $\mathbf{J}_{T2}$ are defined by

$$\mathbf{J}_{T1} = \mathbf{J}_{SC} + m_{space\ cobot} \left(\mathbf{r}_{SC}^T \mathbf{r}_{SC} \mathbf{I} - \mathbf{r}_{SC} \mathbf{r}_{SC}\right) \tag{4.18}$$

$$\mathbf{J}_{T2} = \mathbf{J}_P + m_{payload} \left(\mathbf{r}_P^T \mathbf{r}_P \mathbf{I} - \mathbf{r}_P \mathbf{r}_P\right) \tag{4.19}$$

Additionally, the actuation model is modified to considered the deviation of the CoM as seen in Figure 4.4, via

$$\mathbf{r}_{p_i} = \mathbf{c}_{off} + \mathbf{r}_i \tag{4.20}$$

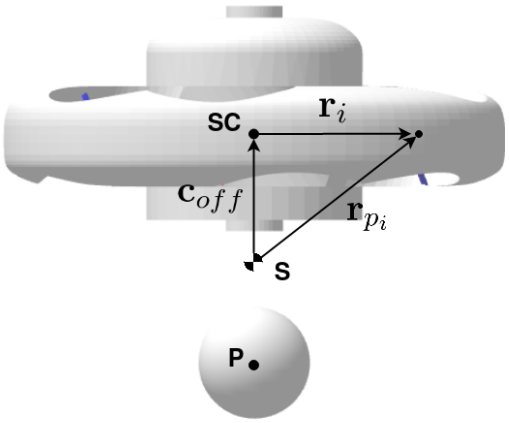


Figure 4.4: Visualization of vector operation to obtain rotor positions with payload.

The resulting actuation model can then be redefined by

$$\begin{pmatrix} \mathbf{F}_i \\ \mathbf{M}_i \end{pmatrix} = \mathbf{a}_i u_i \qquad \mathbf{a}_i = \begin{pmatrix} K_1 \hat{u}_i \\ K_1 \mathbf{r}_{p_i} \times \hat{u}_i - w_i K_2 \hat{u}_i \end{pmatrix} \tag{4.21}$$

The resulting dynamics model is then

$$\begin{cases} \dot{\mathbf{p}} = \mathbf{v} \\ \dot{\mathbf{v}} = (1/m_{system})\,\mathbf{R}(\mathbf{q})\,\mathbf{F} \\ \dot{\mathbf{q}} = (1/2)\,\mathbf{Q}(\mathbf{q})\,\boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}} = \mathbf{J}_S^{-1}(\mathbf{M} - \boldsymbol{\omega} \times \mathbf{J}_S\boldsymbol{\omega}) \end{cases} \tag{4.22}$$

where

$$m_{system} = m_{space\ cobot} + m_{payload} \tag{4.23}$$

4.1.3 Formation with payload model

To increase the applications of the Space CoBot project, a formation configuration while carrying a payload is also considered. This formation will be considered as a single system where all elements are rigidly linked, comprising a rigid body. In this case, the model will be defined around the CoM of the complete system. The resulting configuration can be observed in Figure 4.5.

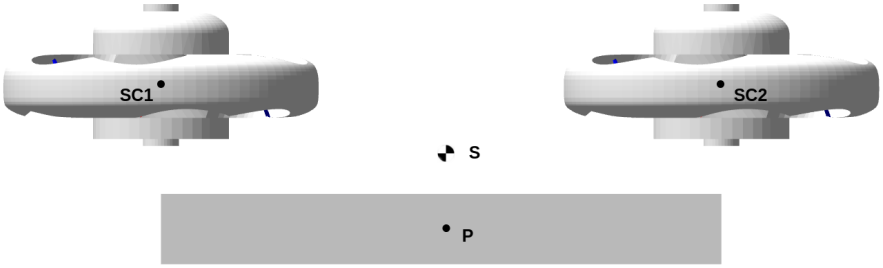


Figure 4.5: Formation with payload configuration.

Similarly to the single Space CoBot with payload configuration, the new CoM, represented by point S, can be computed by

$$x_S = x_{SC1} \quad \vee \quad x_S = x_{SC2} \tag{4.24}$$

$$y_S = \frac{m_{space\ cobot\ 1} y_{SC1} + m_{space\ cobot\ 2} y_{SC2} + m_{payload} y_P}{m_{space\ cobot\ 1} + m_{space\ cobot\ 2} + m_{payload}} \tag{4.25}$$

$$z_S = \frac{m_{space\ cobot\ 1} z_{SC1} + m_{space\ cobot\ 2} z_{SC2} + m_{payload} z_P}{m_{space\ cobot\ 1} + m_{space\ cobot\ 2} + m_{payload}} \tag{4.26}$$

The total inertia of the system can be calculated by summing the inertia of each element expressed around point S, the CoM of the system, using the same method mentioned in the previous section resulting in the following expression

$$\mathbf{J}_S = \mathbf{J}_{T1} + \mathbf{J}_{T2} + \mathbf{J}_{T3} \tag{4.27}$$

with

$$\mathbf{J}_{T1} = \mathbf{J}_{SC1} + m_{space\ cobot\ 1}\ \left(\mathbf{r}_{SC1}^T\mathbf{r}_{SC1}\mathbf{I} - \mathbf{r}_{SC1}\mathbf{r}_{SC1}\right) \tag{4.28}$$

$$\mathbf{J}_{T2} = \mathbf{J}_{SC2} + m_{space\ cobot\ 2}\ \left(\mathbf{r}_{SC2}^T\mathbf{r}_{SC2}\mathbf{I} - \mathbf{r}_{SC2}\mathbf{r}_{SC2}\right) \tag{4.29}$$

$$\mathbf{J}_{T3} = \mathbf{J}_{P} + m_{payload}\ \left(\mathbf{r}_{P}^T\mathbf{r}_{P}\mathbf{I} - \mathbf{r}_{P}\mathbf{r}_{P}\right) \tag{4.30}$$

Actuation forces must take into account the new CoM. This is solved by modifying the actuation matrix $\mathbf{A}$ of each Space CoBot to account for the new positions of each rotor relative to the new CoM. Observing Figure 4.6 for reference, let us consider one Space CoBot from the formation. The modification to its actuation matrix can be quickly implemented by registering the displacement of the CoM of the Space CoBot relative to the CoM of the system. This offset will be designated by $\mathbf{c}_{off}$ as depicted in Figure 4.6.

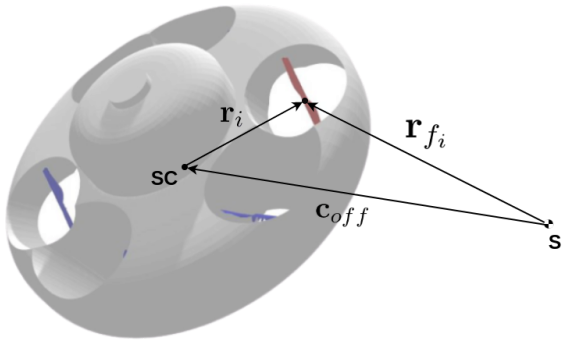


Figure 4.6: Visualization of vector operation to obtain rotor positions in formation.

By performing a simple addition of the position of each rotor, $\mathbf{r}_i$ in (4.9) to $\mathbf{c}_{off}$, the position of each rotor relative to the CoM of the system emerges, designated by $\mathbf{r}_{f_i}$ and defined by

$$\mathbf{r}_{f_i} = \mathbf{r}_i + \mathbf{c}_{off} \tag{4.31}$$

The resulting actuation model of the individual Space CoBot can then be defined by

$$\begin{pmatrix}\mathbf{F}_i\\ \mathbf{M}_i\end{pmatrix} = \mathbf{a}_i u_i \qquad \mathbf{a}_i = \begin{pmatrix} K_1\hat{u}_i \\ K_1\mathbf{r}_{f_i}\times\hat{u}_i - w_i K_2\hat{u}_i\end{pmatrix} \tag{4.32}$$

The resulting force and torque will be the sum of the contributions of all 12 rotors in the formation. In matrix form

$$\begin{pmatrix}\mathbf{F}_1+\mathbf{F}_2\\ \mathbf{M}_1+\mathbf{M}_2\end{pmatrix} = \mathbf{A}_1\mathbf{u}_1 + \mathbf{A}_2\mathbf{u}_2 \tag{4.33}$$

where $\mathbf{A}_i = [\mathbf{a}_{i_1}\ ...\ \mathbf{a}_{i_6}]$ are the actuation matrices of each Space CoBot and $\mathbf{u}_1$ and $\mathbf{u}_2$ are individual

control inputs corresponding to each Space CoBot, which compose the actuation input $\mathbf{u} = \left[\mathbf{u}_1^T \ \mathbf{u}_2^T\right]^T$.

The dynamics of the system described through the CoM of the system is then obtained by summing the contributions of each Space CoBot and are defined by

$$\begin{cases} \dot{\mathbf{p}} = \mathbf{v} \\ \dot{\mathbf{v}} = (1/m_{total})\,\mathbf{R}(\mathbf{q})\,(\mathbf{F}_1 + \mathbf{F}_2) \\ \dot{\mathbf{q}} = (1/2)\,\mathbf{Q}(\mathbf{q})\,\boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}} = \mathbf{J}_S^{-1}(\mathbf{M}_1 + \mathbf{M}_2 - \boldsymbol{\omega} \times \mathbf{J}_S\boldsymbol{\omega}) \end{cases} \tag{4.34}$$

where

$$m_{total} = 2\,m_{SpaceCoBot} + m_{payload} \tag{4.35}$$

Note that the states of the model mentioned above in (4.34) are described in the CoM of the formation while each Space CoBot has sensors positioned in their own CoM. Therefore, this information needs to be transformed to the CoM of the formation. This is achieved by following Euler's equations for rigid body dynamics. Considering the formation system as a rigid body, the linear velocity and acceleration of any point A in the body, $\mathbf{v}_A$ and $\mathbf{a}_A$, relative to the inertial frame, can be calculated given that the linear velocity and acceleration of any point B in the body, $\mathbf{v}_B$ and $\mathbf{a}_B$, relative to the inertial frame, as well as the body's angular velocity and acceleration also expressed in the inertial frame, $\boldsymbol{\omega}_{b/i}$ and $\boldsymbol{\alpha}_{b/i}$, and the position of point A in relation to point B, $\mathbf{r}_{BA} = \mathbf{p}_A - \mathbf{p}_B$, are known. The relations that result from these equations follow the expressions below

$$\mathbf{p}_A = \mathbf{p}_B + \mathbf{r}_{BA} \tag{4.36}$$

$$\mathbf{v}_A = \mathbf{v}_B + \boldsymbol{\omega}_{b/i} \times \mathbf{r}_{BA} \tag{4.37}$$

$$\mathbf{a}_A = \mathbf{a}_B + \boldsymbol{\alpha}_{b/i} \times \mathbf{r}_{BA} + \boldsymbol{\omega}_{b/i} \times (\boldsymbol{\omega}_{b/i} \times \mathbf{r}_{BA}) \tag{4.38}$$

Given that the angular velocity and acceleration, $\boldsymbol{\omega}_{b/i}$ and $\boldsymbol{\alpha}_{b/i}$, are equal at any given point of the body, it is fairly easy to transform position, velocity and acceleration data between body points.

4.1.4 Formation flight without payload

Formation flying comprises the flight of multiple objects in a synchronized manner while maintaining constant relative positions between each element of the formation. Flying in formation without a payload does not require a specific model. For simple operations, flying in formation can be achieved by making all formation elements track a specific point and a common orientation. Considering a formation comprising two Space CoBot vehicles, by resorting to (4.36) and opposing relative position references, the same path can be provided to both vehicles which will be followed via the respective transformations originated from (4.36). By providing the same orientation, possible collisions are avoided given that both will follow a unique path.

It is important to note that until this point, starting from the single Space CoBot model, one can derive the payload carrying model by altering two parameters, system mass and inertia matrix. One can also derive a formation carrying model by providing three parameters, system mass, inertia matrix and a set of relative positions that define the formation shape and the payload positioning in the formation.

Throughout this thesis, the set of relative positions corresponding to the formation are all known beforehand and considered to be constant. Regarding the payload parameters such as mass, inertia matrix and position relative to the carrying system, they are all known beforehand as their estimation is not within the scope of this work. In addition, research on how to perform such estimations has already been conducted within the Space CoBot project in [56].

4.2 Nonlinear Model Predictive Control

To achieve a NMPC formulation, an NLP must be formulated to allow NLP solvers to find control solutions at each time step while satisfying any additional imposed constraints. In this section, this problem formulation and solution are detailed.

4.2.1 Problem formulation

To perform the problem discretization, multiple shooting is used.

First the controls are discretized piecewise on a given time grid

$$\mathbf{u}(t) = \mathbf{q}_i \quad \text{for} \quad t \in [t_i, t_{i+1}] \tag{4.39}$$

Then the ODE is solved on each interval $[t_i, t_{i+1}]$ independently, starting with an artificial initial value $\mathbf{s}_i$, the so-called shooting node [32]

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}_i(t), \mathbf{q}_i), \quad t \in [t_i, t_{i+1}], \tag{4.40}$$

$$\mathbf{x}_i = \mathbf{s}_i \tag{4.41}$$

By numerically solving these initial value problems, the state trajectory pieces $\mathbf{x}_i(t, \mathbf{s}_i, \mathbf{q}_i)$ are obtained [51].

Simultaneously with the decoupled ODE solution, the finite differences in the cost function (3.28) are also numerically computed at each point of the grid

$$l_i(\mathbf{s}_i, \mathbf{q}_i) := \int_{t_i}^{t_{i+1}} \ell(\mathbf{x}_i(t_i, \mathbf{s}_i, \mathbf{q_i}), \mathbf{q}_i) dt \tag{4.42}$$

In order to ensure dynamic feasibility, the continuity conditions $\mathbf{s}_{i+1} = \mathbf{x}_i(t_{i+1}, \mathbf{s}_i, \mathbf{q}_i)$ are imposed.

The 4^{th} order Runge-Kutta integration method applied and described in [57] is used to numerically solve the integrals.

$$\int_{t_i}^{t_{i+1}} \dot{\mathbf{x}} dt = \int_{t_i}^{t_{i+1}} f(\mathbf{x}, \mathbf{u}) dt \rightarrow \mathbf{x}_{t+1} - \mathbf{x}_i \approx \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F4) \tag{4.43}$$

where

$$F_1 = hf(\mathbf{x}, \mathbf{u}) \tag{4.44}$$

$$F_2 = hf\left(\mathbf{x} + \frac{h}{2}, \mathbf{u} + \frac{F1}{2}\right) \tag{4.45}$$

$$F_3 = hf\left(\mathbf{x} + \frac{h}{2}, \mathbf{u} + \frac{F2}{2}\right) \tag{4.46}$$

$$F_4 = hf(\mathbf{x} + h, \mathbf{u} + F3) \tag{4.47}$$

and h is the time interval. Thus, arriving at the following NLP formulation

$$\underset{\mathbf{s},\mathbf{q}}{\text{minimize}} \quad \sum_{i=0}^{N} l_i(\mathbf{s}_i, \mathbf{q}_i) + J(\mathbf{s}_{N+1}) \tag{4.48}$$

$$\text{subject to} \quad \mathbf{s}_0 - \mathbf{x}_0 = 0, \tag{4.49}$$

$$\mathbf{s}_{i+1} - \mathbf{x}_i(t_{i+1}, \mathbf{s}_i, \mathbf{q}_i) = 0, \quad i = 0, ..., N, \tag{4.50}$$

$$h(\mathbf{s}_i, \mathbf{q}_i) \geq 0, \quad i = 0, ..., N, \tag{4.51}$$

$$g(\mathbf{s}_{N+1}) = 0. \tag{4.52}$$

All optimization variables can be summarized as $\mathbf{z} := (\mathbf{s}_0, \mathbf{q}_0, ..., \mathbf{s}_{N+1}, \mathbf{q}_N)$.

4.2.2 Sequential Quadratic Programming

In order to use NMPC in real-time applications involving fast dynamic processes, solution computation times must approach at least the millisecond range. Of-the-shelf algorithms are not able to accomplish this in a stable manner, meaning that certain problems can be solved near the $1\,s$ mark while increasing the complexity of the task increases the computation time to $10\,s$ or no solution is found at all, as observed in [36].

The SQP considered now, augments the objective function through the Lagrangian function $\mathcal{L}$ as in (3.12). The objective is to apply Newton's method to the KKT conditions (3.13) and (3.14) as in [52] resulting in the following linear system

$$\begin{bmatrix} \mathbf{H}_L & \nabla_z \mathbf{C}^T \\ \nabla_z \mathbf{C} & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{z} \\ -\boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla_z F_J \\ -\mathbf{C} \end{bmatrix} \tag{4.53}$$

where $\mathbf{H}_L$ is the Hessian matrix of the Lagrangian, C is the set of equality and inequality constraints and

F_J is the objective function being augmented.

$$\mathbf{H}_L = \nabla_z^2 F_J - \sum_{i=1}^{b} \lambda_i \nabla_z^2 C_i \tag{4.54}$$

where b is the size of the set of variables $(\mathbf{z}, \boldsymbol{\lambda})$ being iterated. According to [52], in this approach the search direction $\Delta\mathbf{z}$ can be computed by solving the QP subproblem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\Delta\mathbf{z}^T \mathbf{H}_L \Delta\mathbf{z} + \nabla_z F_J^T \Delta\mathbf{z} && (4.55)\\ \text{with respect to} \quad & \Delta\mathbf{z} \in \mathbb{R}^{n_z} && (4.56)\\ \text{subject to} \quad & \nabla_z \mathbf{C} \Delta\mathbf{z} = -\mathbf{C} && (4.57) \end{aligned}$$

Note that approximations of the constraints $\mathbf{C}$ are used (4.57). To increase the computation speed, an approximation of $\mathbf{H}_L$ is computed via the Gauss-Newton approach [57].

Real-time scheme

In order to reduce this feedback delay and the computation time in general, an real-time embedding strategy proposed in [37] is used. This method performs only one SQP iteration per time step and divides the calculations in two separate steps, a *preparation step* and a *feedback step*. All operations that are independent of the current state measurement are included in the preparation step and can be carried out offline. Upon the observation of the current state values the QP subproblem is constructed and solved yielding $\Delta\mathbf{u}$ and $\Delta\mathbf{x}$. This provides a control input via $\mathbf{u} = \mathbf{u}_{guess} + \Delta\mathbf{u}$ that can be immediately applied to the system [58]. The control input is maintained during a certain duration in which the *preparation step* takes place, where the solution obtained in the previous step $\Delta\mathbf{x}$ is shifted in time and used to provide a good initial guess for the next iteration. The *feedback step* includes the initial value embedding and the solving of the QP problem, which is typically faster [57]. Consequently, the feedback delay is reduced to just the computation time of the QP solver.

In this scheme, no globalization strategy is used, under the assumption that approximated solutions are enough to achieve convergence.

4.3 Trajectory Optimization

In the context of this thesis, for the NMPC to work as efficiently as possible, feasible trajectories will be provided as references instead of general references. To achieve this, the numerical trajectory optimization algorithm ALTRO proposed in [46] will be integrated in this work. This algorithm takes an indirect optimization approach to solve variations of an OCP such as 3.1 which is known for achieving dynamically feasible solutions. It comprises several optimization algorithm that can be used together or individually.

4.3.1 Problem formulation

The cost function ℓ is augmented to consider constraints forming a Lagrangian function $\mathcal{L}$ defined by

$$\mathcal{L} = \ell(x_k, u_k) + \lambda^T c_k(x_k, u_k) + \frac{1}{2} c_k(x_k, u_k)^T I_\mu c_k(x_k, u_k) \tag{4.58}$$

where c_k is the concatenated set of equality and inequality constraints, $c = (g_k, h_k)$, with index sets $\mathcal{E}_k$ and $\mathcal{I}_k$, respectively, λ are Lagrangian multipliers, μ are the penalty multipliers, and I_μ is a diagonal matrix defined by

$$I_\mu = \begin{cases} 0 & \text{if} \quad c_{k_i}(x_k, u_k) < 0 \wedge \lambda_{k_i} = 0, i \in \mathcal{I} \\ \mu_i & \text{otherwise,} \end{cases} \tag{4.59}$$

where k_i denotes the ith constraint at time step k.

Two main steps are considered to solve this problem. The iLQR iteration and the Augmented Lagrangian update.

4.3.2 Iterative LQR

The objective of this method is to approximate all nonlinear constraints and objective using first or second order Taylor series expansions at each iteration in order to operate on deviations around the nominal trajectory defined by δu_k. A iLQR iteration has two major steps, a *backward pass* that minimizes (4.58) with respect to deviations δu_k and a *forward pass* that updates the nominal trajectories $X = \{x_0, ..., x_n\}$ and $U = \{u_0, ...1, , u_{N-1}\}$ by simulating the dynamics forward with the optimal corrective deviations δu_k^{opt} starting from the fixed initial state x_0 [46]. This can be summarized by the following

$$u_k^{new} = u_k + \delta u_k^{opt} \tag{4.60}$$

$$x_{k+1}^{new} = f(x_k^{new}, u_k^{new}) \tag{4.60a}$$

This iLQR section iterates continuously until one of two conditions are reached. Either an iteration limit is achieved or a given tolerance is satisfied for the improvement of the objective function relative to the previous iteration objective function, defined by $|\mathcal{L} - \mathcal{L}_{prev}| < tolerance$ [46].

4.3.3 Augmented Lagrangian iLQR

To improve the convergence rate and quality of the solution, after an iLQR step, with λ and μ held constant, both variables can be updated according to

$$\lambda_{k_i}^+ = \begin{cases} \lambda_{k_i} + \mu_{k_i} c_{k_i}(x_k^{opt}, u_k^{opt}) & i \in \mathcal{E}_k \\ \max(0, \lambda_{k_i} + \mu_{k_i} c_{k_i}(x_k^{opt}, u_k^{opt})) & i \in \mathcal{I}_k \end{cases} \tag{4.61}$$

and

$$\mu_{k_i}^{+} = \phi \mu_{k_i} \tag{4.62}$$

where $\phi > 1$ is a scaling factor. A new iLQR step is then started. This step will keep updating these values until one of two conditions is met. Either an iteration limit is reached or a minimal constraint violation value is satisfied [46].

Both iLQR and Augmented Lagrange parameters can be tuned by the user.

4.3.4 Active-Set Projection Method

The solution from the Augmented Lagrangian-iLQR stage provides a coarse solution, described in this section by $Y \leftarrow X, U, \lambda$ for simplicity. For this reason, an active-set projection method is also implemented in ALTRO [46]. It uses this coarse solution as a warm start solution in order to find a solution that strictly satisfies the dynamics and constraints, rendering the resulting trajectories feasible. This approach takes the AL-iLQR solution and projects it onto the manifold associated with the active constraints, already presented in this work as *active set* $\mathcal{A}$. Then the algorithm uses a Newton method, already described in this work, and takes successive steps δY until reaching a desired constraint tolerance. From this point on, this method will be referred to as the ALTRO algorithm.

4.3.5 Infeasible state trajectory initialization

As mentioned before, optimization algorithms often rely on a good initial guess, or at least a feasible initial guess. This is even more evident in *indirect* methods such as this one, where dynamics are strictly enforced by forward simulation. In [46], dynamically infeasible state trajectory initialization is possible by introducing additional modifications to the dynamics with slack controls $s_k \in \mathbb{R}^n$ that provide an initial relaxation of the problem. The dynamics are then replaced by

$$x_{k+1} = f(x_k, u_k) + s_k \tag{4.63}$$

Following [46], the optimization problem is also modified by adding the cost term

$$\sum_{k=0}^{N-1} \frac{1}{2} s_k^T R_s s_k \tag{4.64}$$

and the constraints

$$s_k = 0, \quad k = 0, ..., N-1 \tag{4.65}$$

Since at convergence $s_k = 0$ is supposedly achieved, a dynamically feasible solution is still obtained.

Chapter 5

Implementation

In this chapter, details are presented regarding the implementation of all subsystems within the ROS framework, namely the simulation environment, the NMPC controller and the trajectory optimization algorithm. First, Sections 5.1, 5.2 and 5.3 detail all the model parameters used in this work. In Section 5.4 the chosen simulation environment to perform tests on the developed methods is presented in detail. The integration of the NMPC controller is detailed in section 5.5. Finally, in Section 5.6 the integration of the trajectory generation algorithm is described.

5.1 Space Cobot parameters

Although this thesis mainly relies on a simulation of the Space Cobot to test and evaluate the control system developed, the ISR-Lisboa has manufactured a working prototype of the vehicle visible in Figure 5.1. Therefore, the model described in 4.1 will now be parameterized to represent as accurately as possible the real prototype. This way, future validations on the real prototype can be compared, to some extent, with the work presented throughout this thesis.

Figure 5.1: Space CoBot prototype.

5.1.1 Design parameters

The real prototype follows the design proposed in [4] and so shall the model used for simulation. The Space Cobot comprises six rotors equally distanced to the CoM by $d = 0.16\,m$ and equally spaced between each other.

Table 5.1: Design parameters. Angles are expressed in degrees. Table from [4].

i-th rotor	1	2	3	4	5	6
θ_i	0	60	120	180	240	300
ϕ_i	55	-55	55	-55	55	-55
w_i	-1	1	-1	1	-1	1

The actuation matrix $\mathbf{A}$ from (4.12), depends on the vehicle's design parameters θ_i, ϕ_i, w_i, d, K_1 and K_2. The first three are presented in Table 5.1.

To define the constants K_1 and K_2 the desirable option would be to use propeller performance data from [54] for the propeller used in the prototype, which is designated HQ 4x4.5-BN. However, this specific propeller is not included in this database. Fortunately, this performance study is not entirely useless. As mentioned before, the relation C_p/C_t can be considered constant for varying rotor angular velocity and tends to present little difference between similar propellers. Therefore, a value for this relation will be assumed and, consequently, K_2/K_1 can also be calculated, which will be relevant to the work that follows.

Motor vendors usually provide performance data and technical specifications of the motor when coupled with different propellers. This information[1] is available for the Cobra 2204 motor and HQ 4x4.5-BN propeller, both used in the prototype. This information will allow the extraction of the K_1 constant.

Table 5.2: HQ 4x4.5-BN performance data when coupled with Cobra 2204/2300Kv motor.

Propeller	$\Omega[rpm]$	thrust $[g]$	thrust $[N]$
HQ 4x4.5-BN	23754	543	5.32

Considering the scalar thrust produced by a single propeller $f_i = K_1 u_i$, where u_i is expressed in rps^2. The relevant performance data of the used propeller is condensed in Table (5.2). It is then trivial to obtain the K_1 constant if a conversion from rpm to rps is performed, given that angular velocity data in Table (5.2) is expressed in rpm. Consequently, K_1 is simply computed by solving the following

$$f_i = K_1 u_i \Leftrightarrow \tag{5.1}$$

$$5.32 = K_1 \left(\frac{23754}{60}\right)^2 \Leftrightarrow \tag{5.2}$$

$$K_1 = 3.39 \times 10^{-5} \tag{5.3}$$

where dividing the angular velocity by 60 performs the conversion to rps.

[1] https://www.innov8tivedesigns.com/images/specs/Cobra_CM-2204-28_Specs.htm

As stated earlier, an approximate relation K_2/K_1 can be computed by resorting to the relation C_p/C_t of a similar propeller[2]. Although this relation tends to present a small variation with angular velocity, the selected value, $C_p/C_t = 0.397$, corresponds to relatively slow velocities ($\leq 9000\ rpm$). Given the propeller diameter, $D = 0.1016\,m$, the relation K_2/K_1 is then computed by

$$\frac{K_2}{K_1} = \frac{C_p D}{C_t 2\pi} \approx 6.4 \times 10^{-3} \tag{5.4}$$

The relation in (5.4) is useful once it allows the modification of the actuation matrix $\mathbf{A}$ from (4.12) to be dependent on K_2/K_1 instead of the individual constants K_1 and K_2. This is accomplished by factorizing the constant K_1 resulting in the following

$$\mathbf{A} = K_1 \mathbf{A}_N \tag{5.5}$$

where $\mathbf{A}_N = [\mathbf{a}_{N_1}\ ...\ \mathbf{a}_{N_6}]$ which will be hereafter designated by normalized actuation matrix and

$$\mathbf{a}_{N_i} = \begin{pmatrix} \hat{u}_i \\ \mathbf{r}_i \times \hat{u}_i - w_i \frac{K_2}{K_1} \hat{u}_i \end{pmatrix}, \qquad i \in \{1, ..., 6\} \tag{5.6}$$

The modified forces/moments expression is then described by

$$\begin{pmatrix} \mathbf{F} \\ \mathbf{M} \end{pmatrix} = \lambda_{rpm} \mathbf{A_N u} \tag{5.7}$$

where λ_{rpm} is used to simplify the resulting expression and comprises the K_1 constant and a conversion factor that allows the control input u_i to be expressed in rpm^2 from this point further. It is defined by

$$\lambda_{rpm} = \frac{K_1}{60^2} = 9.428 \times 10^{-9} \tag{5.8}$$

Other parameter such as mass, m, and inertia matrix, $\mathbf{J}$, were obtained experimentally in the ISR-Lisboa laboratory and are introduced below

$$m = 6.047\,kg \qquad \mathbf{J} = \begin{bmatrix} 0.0376245 & 0 & 0 \\ 0 & 0.0426381 & 0 \\ 0 & 0 & 0.0678863 \end{bmatrix} \tag{5.9}$$

This concludes the definition of all Space CoBot's design parameters that will be used in the following sections.

[2] https://m-selig.ae.illinois.edu/props/volume-2/data/gwsdd_5x3_static_0323rd.txt

5.2 Payload parameters

As visible in Figure 4.3, a $6\,kg$ spherical payload with a diameter of $0.1\,m$ is considered. Its CoM presents a displacement of $-0.2\,m$ in the z axis relative to the CoM of the Space CoBot. With this information, relative positions in (4.18) and (4.19) can be calculated

$$\mathbf{r}_{SC} \rightarrow (0,\, 0,\, 0.09961) \tag{5.10}$$

$$\mathbf{r}_{P} \rightarrow (0,\, 0,\, -0.10039) \tag{5.11}$$

The system mass and inertia matrix then become

$$m_{system} = 12.047\,kg \qquad \mathbf{J}_S = \begin{bmatrix} 0.1774 & 0 & 0 \\ 0 & 0.1825 & 0 \\ 0 & 0 & 0.073886 \end{bmatrix} \tag{5.12}$$

5.3 Formation parameters

To form a flying formation two Space CoBot vehicles are considered in this work. However, the present approach allows this number to be increased. The payload considered now takes a cuboid shape as seen in Figure 4.5, which measures $0.5 \times 0.8 \times 0.1\,m$. The CoM of each Space CoBot present a relative displacement of $0.8\,m$ in the y axis. The payload is kept at $6\,kg$ to allow possible comparisons to the single Space CoBot with payload configuration. Its CoM presents a displacement of $-0.2\,m$ in the z axis relative to the plane that contains both Space CoBot robots (xy plane). With this information, the CoM of the resulting system and relative positions in (4.28), (4.29) and (4.30) can be calculated.

$$\mathbf{r}_{SC1} \rightarrow (0,\, -0.4,\, 0.06632) \tag{5.13}$$

$$\mathbf{r}_{SC2} \rightarrow (0,\, 0.4,\, 0.06632) \tag{5.14}$$

$$\mathbf{r}_{P} \rightarrow (0,\, 0,\, -0.13368) \tag{5.15}$$

The system mass and inertia matrix then become

$$m_{system} = 18.094\,kg \qquad \mathbf{J}_S = \begin{bmatrix} 2.4957 & 0 & 0 \\ 0 & 0.37569 & 0 \\ 0 & 0 & 2.5158 \end{bmatrix} \tag{5.16}$$

5.4 Simulation environment

The Space CoBot was designed to operate in the absence of gravity. Consequently, it is not easy to experiment or perform full validations on a real prototype.

There are several open source simulators available that are easy to use and allow a quick start after a few tutorials. Most simulators have intuitive ways of implementing a certain desired model and will automatically define kinematic and dynamic relations based on a few properties, without the need to specify any equations. However, there are several effects that are not simulated, namely aerodynamic forces generated by moving bodies. Particularly, moving rotors generate complex aerodynamic forces, due to their shape, which are essential to accurately model a rotor based vehicle and reduce the gap between simulation and reality. These effects are strongly dependent on the propeller properties and are often defined using experimental methods. Therefore, one would have to model these effects manually or build plug-ins that achieve the same result.

5.4.1 Gazebo

The simulation environment used this work is based in the Gazebo simulator. It provides extensive documentation and several detailed tutorials that allow a quick introduction to this software. Gazebo uses Simulation Description Format (SDF) files to describe its models and world environments. Objects are described through *links*, which are characterized by mass and moments of inertia, and have geometric representation for visual and collision purposes. The *links* can be connected through *joints* to define specific kinematics properties. The *joints* are just a representation of motion and do not possess mass or inertia.

Motion simulation is performed through Newton-Euler equations together with a first order integrator which comprises the physics engine Open Dynamics Engine[1]. Evidently, it simulates only rigid body dynamics. One of the main reasons to use this simulator is the fact it can be used effortlessly with ROS, and it already is used as a base for more complex simulators.

5.4.2 RotorS simulator

The RotorS simulator provides tools to model MRAVs with a good level of precision. Unified Robotic Description Format (URDF) is used to define the MRAV models considered in this work through the position and orientation of individual elements while using built-in tools to automatically calculate moments of inertia and to convert the resulting models to SDF, the format used by Gazebo.

Included in this package is a plug-in that computes the lacking aerodynamic forces generated by propellers mentioned previously. The additional elements simulated for each rotor are the thrust force F_T, the drag force F_D, the rolling moment M_R and the moment originating from the drag of a propeller M_D following [59] and defined by

[1] https://bitbucket.org/odedevs/ode/src/master/

$$F_T = \Omega^2 C_T e_{z_B} \tag{5.17}$$

$$F_D = -\Omega C_D v_A^{\perp} \tag{5.18}$$

$$M_R = \Omega C_R v_A^{\perp} \tag{5.19}$$

$$M_D = -w C_M F_T \tag{5.20}$$

where Ω is the positive angular velocity of the propeller, C_T is the rotor thrust constant, C_D is the rotor drag constant, C_R is the rolling moment constant and C_M is the rotor drag moment constant. These constants are all positive. w denotes the turning direction of the rotor for which the thrust produced is positive, it can be +1 (counter clockwise) or -1 (clockwise). e_{z_B} corresponds to the unit vector pointing in the z-direction of the rotor's body frame. $v_A^{\perp}$ represents the velocity of the rotor projected in the rotor plane, described by the the same vector e_{z_B}.

RotorS provides other tools such as controllers, state estimators, sensor data simulation, wind simulation, among others. A visualization of the RotorS framework is presented in Figure 5.2.

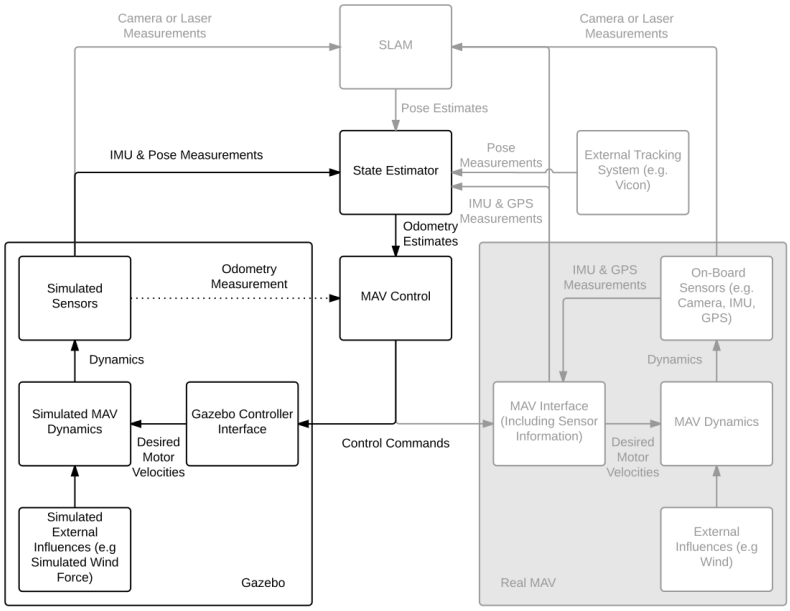


Figure 5.2: Visualization of RotorS framework. Figure from [50].

Darker blocks represent the available tools provided by this simulator. In this work, the *MAV Control* block in Figure 5.2 is replaced by the NMPC controller developed for this work.

5.4.3 Simulation parameters

To achieve an accurate model, the parameters C_T, C_D, C_R and C_M, corresponding to the constants in (5.17)-(5.20), mass, m, and moments of inertia matrix, $\mathbf{J}$, are defined in accordance with the Space CoBot design parameters presented in Section 5.1. Therefore, all elements computed by the RotorS

plug-in are modified by changing the respective constants in such a way that angular velocity can be expressed in rpm. This is already described in (5.7). Hence, to match this model the moments of inertia matrix $\mathbf{J}$ in (5.9) is manually inserted into the parameters instead of using RotorS built-in methods. This is necessary to avoid possible differences between the two models that might cause simulation errors. Given that RotorS computes the inertia of each individual element of the model separately, this can result in small inertia values and Gazebo has proven to be unpredictable when dealing with these small values.

Table 5.3: RotorS model parameters.

Thrust constant	C_T	9.428×10^{-9}
Drag constant	C_D	1.344×10^{-6}
Rolling moment constant	M_R	1.667×10^{-8}
Drag moment constant	M_D	6.400×10^{-3}
Rotor radius $[m]$	r	5.086×10^{-2}
Arm length $[m]$	d	1.600×10^{-1}

All required parameters are then presented in Table 5.3. Note that although Space CoBot does not comprise arms as a typical MRAV, the designation *Arm length* is used to describe the rotor's distance to the CoM.

The RotorS plug-in requires the rotors' angular velocities in order to compute the forces and moments to apply to the respective *links* and for setting the visual velocity of the *joints* in the Gazebo simulator. The visual velocity computation in RotorS is necessary given that angular velocity in Gazebo is expressed in rad/s. This conversion is also regulated by a constant defined in the model parameters. Note that this parameter has visual purposes only. The resulting simulation environment can be visualized through Figure 5.3.

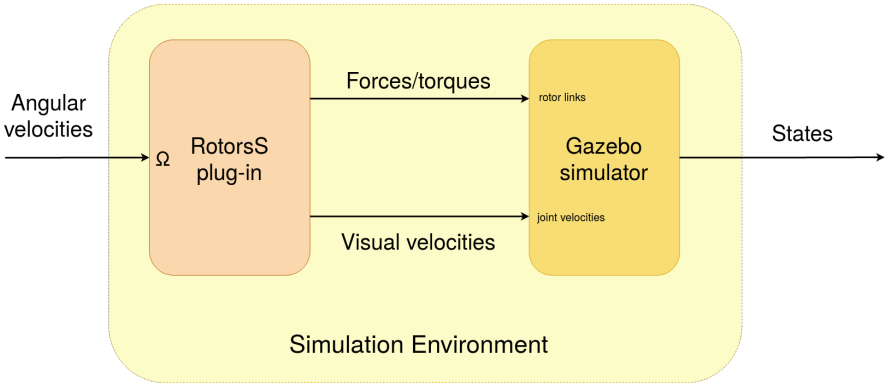


Figure 5.3: Simulation environment framework.

5.4.4 Payload grasping

In order to form a configuration with several MRAVs, each with an independent controller, RotorS initializes MRAVs individually. Therefore, there is no built-in function to create connections between objects

during simulation. Considering a formation with a payload, or the act of grasping an object, such as a payload, is then impossible with the base simulator. To allow these actions, a grasping plug-in[2] was integrated into the simulation environment. This addition allows the individual initialization of all elements of a simulation and the creation of additional Gazebo $joints$ between objects when desired to perform an attachment, and the respective destruction as well, to perform a detachment. Achieving the formation with payload configuration is then performed by creating two additional $joints$ between the vehicles and the payload, creating what can be considered a single rigid body.

5.5 NMPC controller

In order to integrate a NMPC in real-time applications, there are several open source options available. In this thesis the ACADO toolkit [60] was used to generate the real-time solver described in Section 4.2.2. This toolkit comprises several optimization methods for control and estimation, integrators, multiple shooting discretization and includes some QP solvers, such as *qpoases* [61], which is the one selected to solve the SQP subproblems (3.25) in this work.

One major advantage of this toolkit, is the ability to generate highly efficient C code of the selected solver. This eliminates much of the complexity involved in the integration of different software in ROS, which is run in C++ in this work. Also, a MATLAB interface is available that allows the configuration and testing of the selected solver. A simulation in the loop is also possible within MATLAB, which allows some testing of the solver and the respective analysis using MATLAB tools before performing the ROS integration.

The ACADO toolkit generates a set of files, which are included in a ROS node, hereafter designated by controller node, and comprises several functions that are used to interact with the generated solver. Other libraries required to run the solver are also exported and integrated within these files, such as the *qpoases* solver and an integrator. The most relevant functions are the *acado_feedbackStep* and *acado_preparationStep* functions that represent, as their names indicate, the preparation and feedback steps described in Section 4.2.2. Figure 5.4 presents a visualization of these steps and how the solver works internally.

5.5.1 Actuation

It is important to note some details regarding the integration of the ACADO solver. If the optimal control input to be found by the solver is expressed in rpm^2, the set of possible solutions will be significantly big.

Consider the angular velocity limit $|\mathbf{u}| \leq 10000^2\ rpm^2$, the set of possible solutions is then

$$\mathbf{u} \in [-1 \times 10^8, 1 \times 10^8] \tag{5.21}$$

In order to remove some complexity of the model provided to the solver, the optimal control input to be found by the solver will be expressed in force, N, and converted to angular velocity, rpm, afterwards.

[2] https://github.com/pal-robotics/gazebo_ros_link_attacher

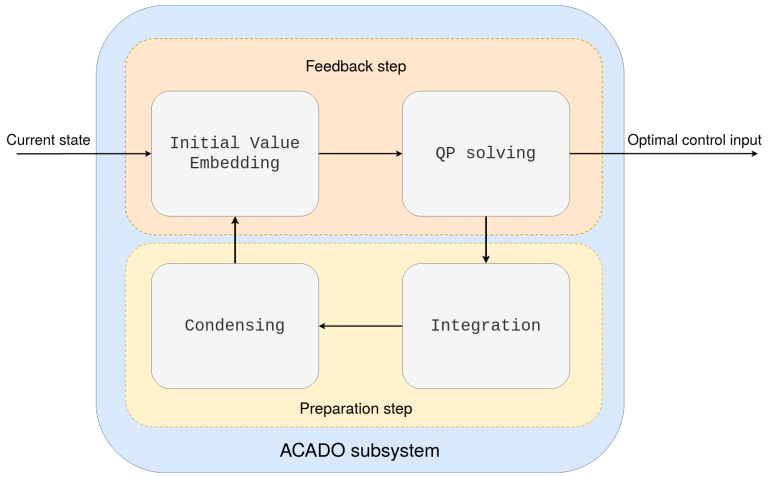


Figure 5.4: ACADO implementation scheme.

This is achieved by modifying the actuation model and defining the force control input $\mathbf{u}_f$ as follows

$$\mathbf{u}_f = \lambda_{rpm}\mathbf{u} \tag{5.22}$$

resulting in the actuation model

$$\begin{pmatrix} \mathbf{F} \\ \mathbf{M} \end{pmatrix} = \mathbf{A_N}\mathbf{u}_f \tag{5.23}$$

The possible solution set, for the same angular velocity limit example as in (5.21), is then

$$\mathbf{u}_f \in [-0.9428, 0.9428] \tag{5.24}$$

This showed a dramatic improvement in computation speed and solution accuracy. This is much faster due to the way the newton method works. The resulting controller node is now represented by Figure 5.5.

The control input $\mathbf{u}_f$ is then converted to angular velocity, $\mathbf{u}_{rpm}$, via

$$\mathbf{u}_{rpm} = \text{sgn}(\mathbf{u}_f)\sqrt{\frac{|\mathbf{u}_f|}{\lambda_{rpm}}} \tag{5.25}$$

The *controller node* is also responsible for receiving and processing sensor data from the simulator and sending control inputs to the simulator through ROS *topics*.

The models provided to ACADO are based in (4.1), (4.22) and (4.34) for the respective configurations. The parameterization of each model is detailed below. ACADO will use these models to generate the integrator, which will be responsible for predicting future steps.

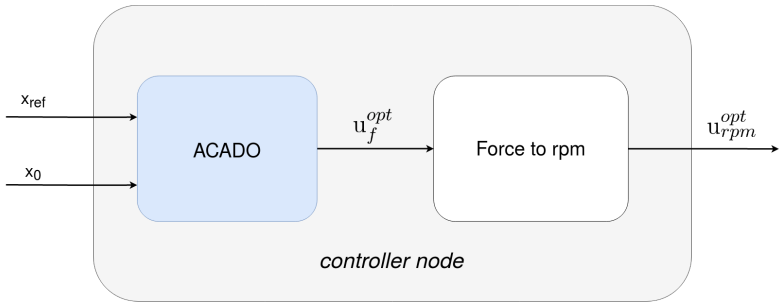


Figure 5.5: Visualization of the controller node.

Actuation matrices

The single Space CoBot configuration is parameterized by the following actuation matrix

$$\mathbf{A_N} = \begin{bmatrix} 0 & -0.7094 & 0.7094 & 0 & -0.7094 & 0.7094 \\ -0.8192 & 0.4096 & 0.4096 & -0.8192 & 0.4096 & 0.4096 \\ 0.5736 & 0.5736 & 0.5736 & 0.5736 & 0.5736 & 0.5736 \\ 0 & 0.0837 & 0.0837 & 0 & -0.0837 & -0.0837 \\ -0.0967 & -0.0483 & 0.0483 & 0.0967 & 0.0483 & -0.0483 \\ -0.1276 & 0.1276 & -0.1276 & 0.1276 & -0.1276 & 0.1276 \end{bmatrix} \tag{5.26}$$

while the single Space CoBot with payload configuration is parameterized by

$$\mathbf{A_N} = \begin{bmatrix} 0 & -0.7094 & 0.7094 & 0 & -0.7094 & 0.7094 \\ -0.8192 & 0.4096 & 0.4096 & -0.8192 & 0.4096 & 0.4096 \\ 0.5736 & 0.5736 & 0.5736 & 0.5736 & 0.5736 & 0.5736 \\ 0.8159 & -0.3239 & -0.3239 & 0.8159 & -0.4920 & -0.4920 \\ -0.0970 & -0.7551 & 0.7551 & 0.0970 & -0.6581 & 0.6581 \\ -0.1274 & 0.1274 & -0.1274 & 0.1274 & -0.1274 & 0.1274 \end{bmatrix} \tag{5.27}$$

Mass and moments of inertia matrix considered for the Space CoBot with and without payload are defined in (5.9) and (5.12), respectively.

The formation configuration with two Space CoBot vehicles is parameterized by the individual actuation matrices

$$\mathbf{A_{N_1}} = \begin{bmatrix} 0 & -0.70941 & 0.70941 & 0 & -0.70941 & 0.70941 \\ -0.81915 & 0.40958 & 0.40958 & -0.81915 & 0.40958 & 0.40958 \\ 0.57358 & 0.57358 & 0.57358 & 0.57358 & 0.57358 & 0.57358 \\ -0.1751 & -0.17258 & -0.17258 & -0.1751 & -0.34061 & -0.34061 \\ -0.097015 & -0.095555 & 0.095555 & 0.097015 & 0.0014593 & -0.0014593 \\ -0.12739 & -0.15637 & 0.15637 & 0.12739 & -0.41116 & 0.41116 \end{bmatrix} \tag{5.28}$$

for Space CoBot number 1 and

$$\mathbf{A_{N_2}} = \begin{bmatrix} 0 & -0.70941 & 0.70941 & 0 & -0.70941 & 0.70941 \\ -0.81915 & 0.40958 & 0.40958 & -0.81915 & 0.40958 & 0.40958 \\ 0.57358 & 0.57358 & 0.57358 & 0.57358 & 0.57358 & 0.57358 \\ 0.24194 & 0.30719 & 0.30719 & 0.24194 & 0.13916 & 0.13916 \\ -0.097015 & -0.059343 & 0.059343 & 0.097015 & 0.037672 & -0.037672 \\ -0.12739 & 0.41116 & -0.41116 & 0.12739 & 0.15637 & -0.15637 \end{bmatrix} \tag{5.29}$$

for Space CoBot number 2. Mass and moments of inertia matrix are defined in (5.16).

5.5.2 Objective function

To define the objective function to be minimized by the solver, a built-in function of the ACADO toolkit [62] is used to generate the following

$$\sum_{k=0}^{N-1} [h(\mathbf{x}_k, \mathbf{u}_k) - y_k] \mathbf{W}_k [h(\mathbf{x}_k, \mathbf{u}_k) - y_k]^T + h_N(\mathbf{x}_N, \mathbf{u}_N) - y_N \mathbf{W}_N [h_N(\mathbf{x}_N, \mathbf{u}_N) - y_N]^T \tag{5.30}$$

where h are called reference functions and are denoted with $h \in \mathbb{R}^{n_y}$ and $h_N \in \mathbb{R}^{n_y,N}$, $y_k \in \mathbb{R}^{n_y}$ and $y_N \in \mathbb{R}^{n_y,N}$ denote minimizing reference values that can be fixed or time-varying, and $\mathbf{W}_k \in \mathbb{R}^{n_y}$ and $\mathbf{W}_N \in \mathbb{R}^{n_y,N}$ are the weighting matrices.

To form the reference functions h and h_N, error functions composed by the differences between current state values and desired references are used. Note that the desired references are not the minimizing references $y_{0,\dots,N-1}$ and y_N mentioned above. Position and attitude errors along with control inputs are selected to construct h and h_N. The objective is to set $y_{0,\dots,N} = 0$ so that minimizing the objective function implies reducing the position and attitude errors and selecting low control inputs.

The position error vector $\Delta\mathbf{p} \in \mathbb{R}^3$ can be trivially obtained by subtraction

$$\Delta\mathbf{p} = \mathbf{p} - \mathbf{p}_{ref} \tag{5.31}$$

The attitude difference, $\Delta\boldsymbol{\Theta} \in \mathbb{R}^3$, is obtained by computing a truncated version of the quaternion error, $\mathbf{q}_e$ described in [63], yielding

$$\Delta\boldsymbol{\Theta} = \begin{bmatrix} q_w^{ref} q_x + q_z^{ref} q_y - q_y^{ref} q_z - q_x^{ref} q_w \\ -q_z^{ref} q_x + q_w^{ref} q_y + q_x^{ref} q_z - q_y^{ref} q_w \\ q_y^{ref} q_x - q_x^{ref} q_y + q_w^{ref} q_z - q_z^{ref} q_w \end{bmatrix} \tag{5.32}$$

Details on the selection of this method to compute the attitude error vector are provided in the appendix A.2.

The control inputs are used as is. Recalling that $y_{0,\dots,N} = 0$ the following objective function is formed

$$\sum_{k=0}^{N-1} [\Delta \mathbf{p}_k, \Delta \boldsymbol{\Theta}_k^T, \mathbf{u}_k^T] \mathbf{W}_k [\Delta \mathbf{p}_k, \Delta \boldsymbol{\Theta}_k^T, \mathbf{u}_k^T]^T + [\Delta \mathbf{p}_N, \Delta \boldsymbol{\Theta}_N^T] \mathbf{W}_N [\Delta \mathbf{p}_N, \Delta \boldsymbol{\Theta}_N^T]^T \tag{5.33}$$

where $\mathbf{W}_k$ is a 12×12 diagonal matrix defined by

$$\mathbf{W}_k = diag(W_p, W_p, W_p, W_\Theta, W_\Theta, W_\Theta, W_u, W_u, W_u, W_u, W_u, W_u) \tag{5.34}$$

and $\mathbf{W}_N$ is a 6×6 diagonal matrix defined by

$$\mathbf{W}_N = diag(W_{p_N}, W_{p_N}, W_{p_N}, W_{\Theta_N}, W_{\Theta_N}, W_{\Theta_N}) \tag{5.35}$$

where W_p and W_{p_N}, W_Θ and W_{Θ_N}, and W_u are scalar values associated with position, attitude and control, respectively.

Note that for the formation with payload configuration, the weighting matrix $\mathbf{W}_k$ is 18×18, where the additional diagonal elements, associated with control inputs, are also defined with the scalar W_u.

5.5.3 Initialization, state and control variables constraints

The ACADO auto-initialization routine uses the provided bounds on state and control variables to generate initial guesses. If upper and a lower bounds are provided, the initial guess will be the result of an arithmetic mean. If only one of these bounds is specified the initial guess will be equal to this bound. If there is a variable for which no bounds are specified, the initial guess will simply set to 0. This applies to the first iteration. Consequent iterations will shift the previous control solution $u^{opt}_{0,\dots,N-1}$ and obtain a new initial guess via $u^{guess}_{0,\dots,N-2} = u^{opt}_{1,\dots,N-2}$, where the last node u^{guess}_{N-1} takes the same value as u^{guess}_{N-2} or is computed through simulation [64]. The same shift applies for the state solution $x^{guess}_{0,\dots,N}$. The initial values for the differential equations are also generated from their bounds. However, these values are obtained by a simulation of the differential system with the initial guess computed for the controls, parameters, and initial states.

When one of the state bounds is violated, the shooting nodes will be projected into the feasible set enforcing the bounds. Therefore, providing these bounds can be beneficial for the initial guess generation. However, it can also originate dynamic violations as shooting nodes are not coupled in time [51].

5.5.4 Solver settings

The resulting controller is now presented.

All selected parameters for the exported NMPC controller are detailed in Table 5.4. Note that several horizon lengths N were tested during this work.

The resulting simulated environment can now be viewed as in Figure 5.6.

Table 5.4: ACADO solver settings.

Time Step	t_s	0.1
Horizon intervals	N	20
Shooting nodes	r_s	21
Discretization type	Multiple Shooting	
Sparse QP solution	Full Condensing	
QP solver	*qpoases*	
Hotstart QP	YES	
Shift solution states	YES	
Shift solution control	YES	

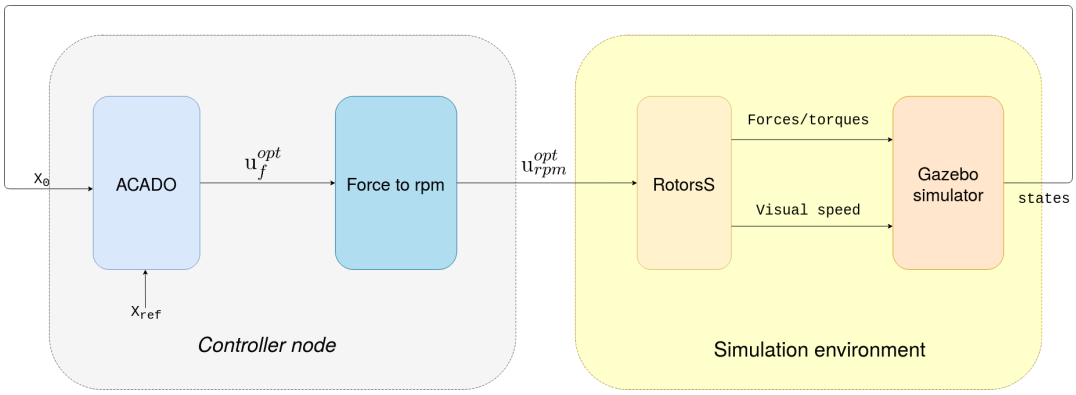


Figure 5.6: Simulation environment with controller node scheme.

5.6 Trajectory Optimization

General references such as a simple terminal waypoint or even paths composed by waypoints proved to be insufficient in obtaining a smooth and accurate solution for more complex trajectories or maneuvers. This is due to the fact that the prediction time window is too short, $2\,s$, which implies that when a desired waypoint is well outside this window, the initialization of shooting nodes causes severe discontinuities in the state trajectory. Together with the fact the considered solver only performs one iteration, it is difficult find good control solutions.

For this reason, a feasible trajectory is computed offline and fed to the NMPC controller as a reference trajectory which in theory should reduce significantly the objective function value along N, and allow better solutions. Given that this computation will occur offline, not included in the real-time framework, a different optimization approach is considered. The aim here is to obtain the best solution possible meaning that there is not an express concern regarding fast computations times.

TrajectoryOptimization.jl[3] is an open source library implemented in JULIA [65] that is used in this work to achieve offline trajectory generation capabilities.

The same system models provided to ACADO in Section 5.5.1 are used in this approach to perform the forward simulation in (4.60a). Note that in this section the full state $\mathbf{x} = [\mathbf{p}, \mathbf{v}, \mathbf{q}, \boldsymbol{\omega}]$ is considered in the objective function of an optimization along a given amount of knots N_{TO}. The time step t is selected to match the time step of the NMPC controller t_s.

[3] https://github.com/RoboticExplorationLab/TrajectoryOptimization.jl

The costs are defined by a built-in function and follows the LQR tracking formulation

$$(\mathbf{x}_{N_{TO}} - \mathbf{x}_f)^T \mathbf{Q}_f (\mathbf{x}_{N_{TO}} - \mathbf{x}_f) + \sum_{k=1}^{N_{TO}-1} (\mathbf{x}_k - \mathbf{x}_f)^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_f) + \mathbf{u}_k^T \mathbf{R}_u \mathbf{u}_k \tag{5.36}$$

where $\mathbf{Q}_f$, $\mathbf{Q}$ and $\mathbf{R}_u$ are 13×13 diagonal weighting matrices, associated to states and control, respectively. For clarity purposes, the weighting matrices $\mathbf{Q}$ will be defined by

$$\mathbf{Q} = diag(Q_p,\, Q_p,\, Q_p,\, Q_p,\, Q_p,\, Q_p,\, Q_\Theta,\, Q_\Theta,\, Q_\Theta,\, Q_\Theta,\, Q_\Theta,\, Q_\Theta,\, Q_\Theta) \tag{5.37}$$

the weighting matrix $\mathbf{Q}_f$ is defined by

$$\mathbf{Q}_f = diag(Q_{p_N},\, Q_{p_N},\, Q_{p_N},\, Q_{p_N},\, Q_{p_N},\, Q_{p_N},\, Q_{\Theta_N},\, Q_{\Theta_N},\, Q_{\Theta_N},\, Q_{\Theta_N},\, Q_{\Theta_N},\, Q_{\Theta_N},\, Q_{\Theta_N}) \tag{5.38}$$

and the weighting matrix $\mathbf{R}_u$ is defined by

$$\mathbf{R}_u = diag(R,\, R,\, R,\, R,\, R,\, R) \tag{5.39}$$

where Q_p and Q_{p_N}, Q_Θ and Q_{Θ_N}, and R are scalar values associated with position, attitude and control, respectively. Independent calibration of the matrices is now possible. It is important to note that for the formation with payload configuration, the weighting matrix $\mathbf{R}_u$ is 12×12, where the diagonal elements are also defined with the scalar R. This choice of cost function implies that the only reference parameter the cost function requires from the user is the terminal desired state $\mathbf{x}_f$.

Several constraints types are considered in this approach:

Goal constraints,

$$\mathbf{x}_{N_{TO}} = a \tag{5.40}$$

State bounds constraints,

$$\text{min} \leq \mathbf{x}_k \leq \text{max}, \quad k = 1, \ldots, N_{TO} \tag{5.41}$$

Control bounds constraints,

$$\text{min} \leq \mathbf{u}_k \leq \text{max}, \quad k = 1, \ldots, N_{TO} - 1 \tag{5.42}$$

Sphere constraints,

$$(x_k - x_c)^2 + (y_k - y_c)^2 + (z_k - z_c)^2 \geq r^2, \quad k = 1, \ldots, N_{TO} \tag{5.43}$$

The resulting objective function is the sum of all cost functions and Lagrangian terms originating from the constraints as defined in (4.58).

The optimal trajectory $\mathbf{x}^{opt} \in \mathbb{R}^{13 \times N_{TO}}$ achieved by this method is saved in a $.txt$ file. The NMPC controller will load this trajectory into an $array$ to serve as references. As the NMPC prediction horizon is significantly smaller than the obtained trajectory, the references will be updated at each time step N intervals at a time. The resulting and complete simulation environment then follows the scheme presented in Figure 5.7.

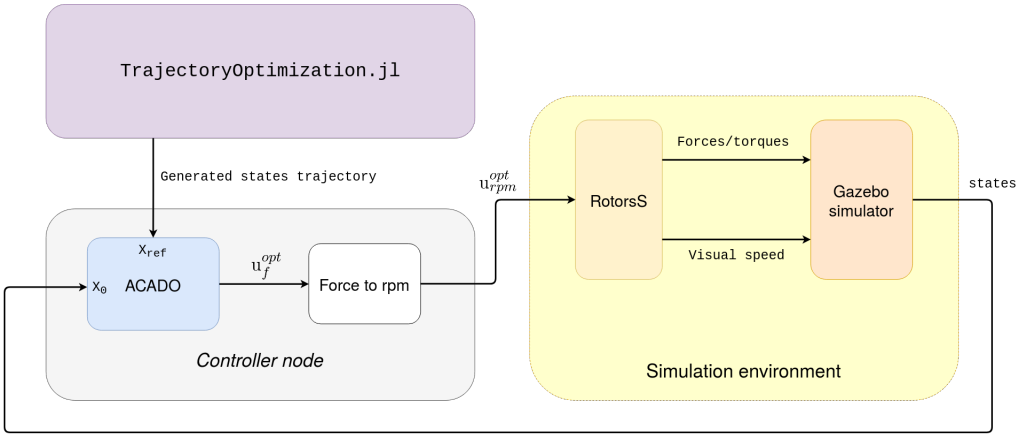


Figure 5.7: Complete simulation environment scheme.

Chapter 6

Results

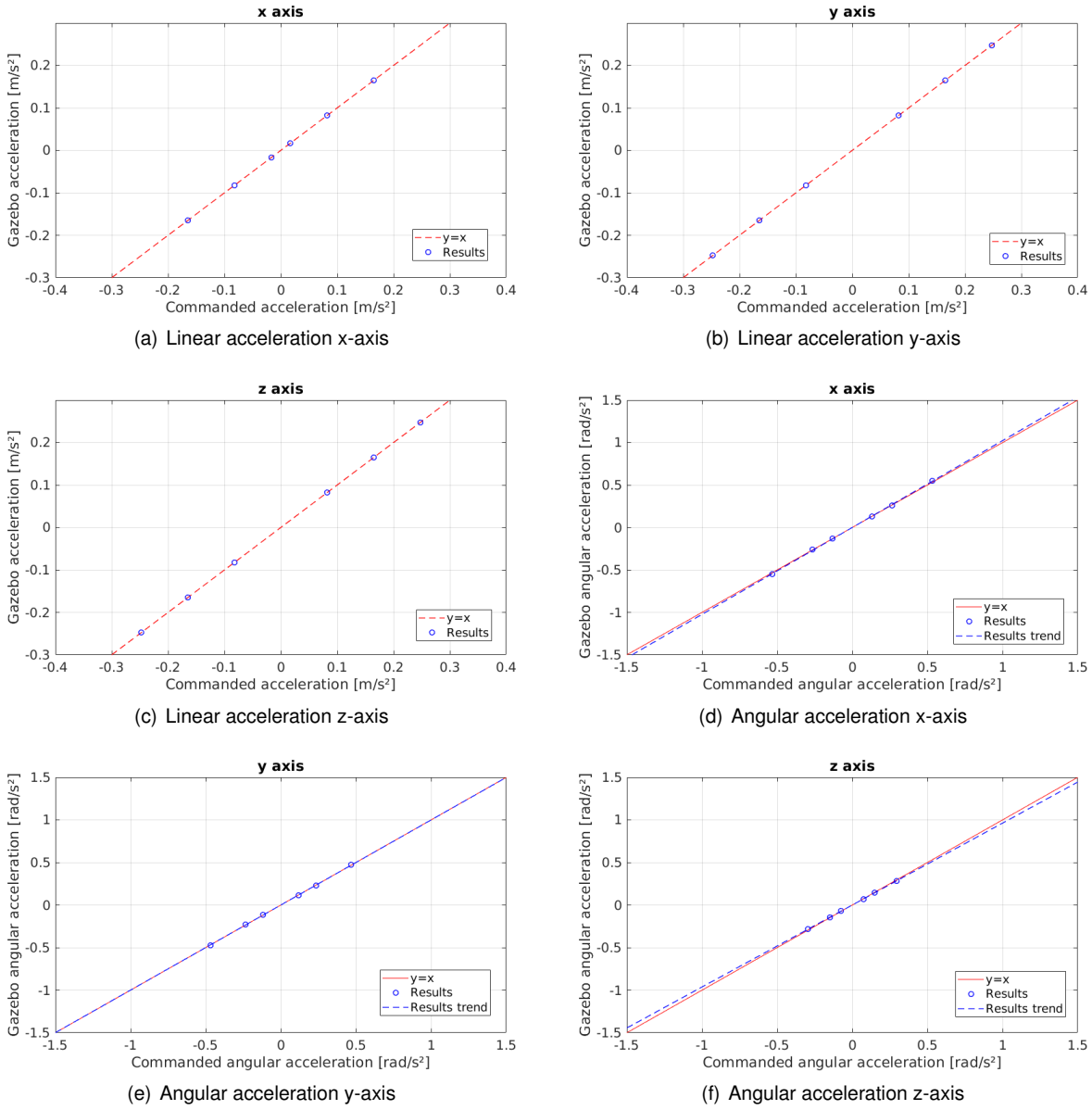
In this chapter, the optimal settings for the NMPC controller are explored through simulation results. The resulting NMPC controller is then evaluated with different system configurations. All involved parts will be tested in an incremental manner from individual algorithms to the full control scheme. First, in Section 6.1, the verification of the model used by the simulator is presented. In Section 6.2 the effect of different solver settings is analyzed along with the behavior caused by the weighting matrices and addition of variable bounds. Section 6.3 presents the trajectory generation performance and how the NMPC controller tracking will be evaluated. Sections 6.4, 6.5 and 6.6 present trajectory execution results divided in three major groups problems, unconstrained flight, flight with obstacles and formation flying, respectively. All results were obtained in a laptop computer with an Intel Core i7-4710HQ @ 2.50GHz.

6.1 Model verification

In order to verify the implicit model generated by RotorS, the simulated system response to several control inputs is evaluated. For comparison, a Space CoBot model is simulated in MATLAB through its mathematical model. The approach selected to these tests is to provide control inputs to the simulator and compare the response with the expected value, computed in MATLAB. The results can be observed in Figure 6.1.

Note that a function of the type $y = x$ is added to provide a reference for the expected values. Linear acceleration is expressed in m/s^2 and angular acceleration is expressed in rad/s^2.

Regarding linear acceleration, the RotorS simulator produces very similar responses. However, when it comes to angular acceleration, there is a small disparity visible in the results in the x and z axes. This error increases with angular acceleration, which means that the model loses accuracy. In this work, such high angular velocities are not expected to be reached during operation. Therefore, the model can still be considered accurate. Nonetheless, this could be an indication of possible errors within the simulation, which has to kept in mind.



(a) Linear acceleration x-axis

(b) Linear acceleration y-axis

(c) Linear acceleration z-axis

(d) Angular acceleration x-axis

(e) Angular acceleration y-axis

(f) Angular acceleration z-axis

Figure 6.1: Verification results.

6.2 NMPC controller configuration

It is difficult to predict how OCP settings affect the real-time controller and one can not look at a specific setting as being independent from the rest. Typically, the time step is a crucial parameter and should be the first to be set. The faster the dynamic system, the shorter the time step should be [66]. This is taken as the starting point to find a good initial configuration. Simple waypoints will be considered as references during this section, reproducing step references.

6.2.1 Prediction horizon

Computation time is essential. Therefore, the selected settings should reflect a tendency to minimize the solver's complexity. The objective is to maximize the prediction time window in order to exploit the MPC

formulation to the fullest. This is accomplished by choosing the longest prediction horizon N that does not compromise the controller operation, meaning that the computation time fits within the controller time step. Other concern to have in mind is the NMPC feedback delay, fd, that measures the interval of time between the acquisition of the current state x_0 and the application of the resulting control solution u_{opt} to the system, this corresponds to the *feedback step* in Figure 5.4. This delay should be as low as possible and takes priority over N and t_s. It is important to note that t_s should be orders of magnitude superior to fd.

Therefore, a compromise between N and t_s has to be achieved while minimizing fd. Evidently for small t_s values, the prediction time window is relatively small, and may cause overshoot problems. Higher t_s values can also cause overshoot and instability, because the control inputs lose resolution and corrective actions may take longer to achieve the objective reference.

A step reference is used to observe the behavior caused by different t_s while holding $N = 60$. The results can be visualized in Figure 6.2 below. As expected there is an interval of t_s values where no overshoot occurs.

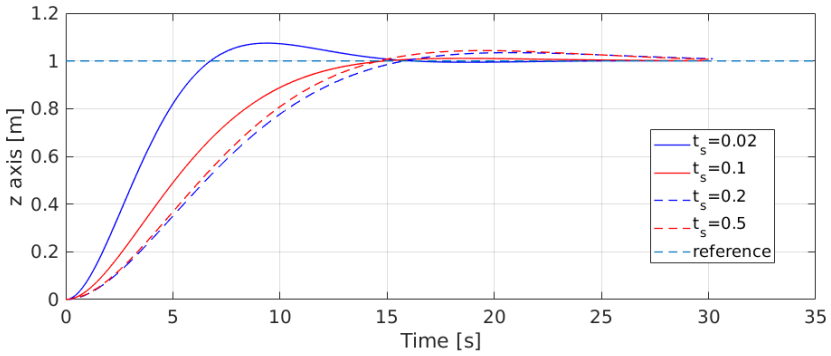


Figure 6.2: System response to different t_s values.

This is a good point to start exploring the effects of N on fd. Resorting to the same step test, different N values were used to measure fd while holding $t_s = 0.1$. The results can be observed below in Figure 6.3. Values over 60 were not considered for N given that the ACADO exporting tool allows a maximum of 61 nodes. Nonetheless, for $N = 60$, the average feedback delay $\bar{fd}$ represents over 30% of t_s which is extremely high. Higher values for t_s would mitigate this percentage. However, sacrificing control resolution would limit the NMPC controller applicability to very slow operations and reduce tracking precision.

$N = 20$ will then be considered for the remainder of this work. It is difficult to state if the selected t_s and N will be the best in all cases, given that weighting matrices can influence the system behavior as well and have been kept constant until this point. The prediction time window is $2\,s$.

6.2.2 Weighting matrices

There is a lot of room for experimentation in this topic. The weighting matrices $\mathbf{W}_k$ and $\mathbf{W}_N$ in (5.34) and (5.35), will be individually tuned through the scalars W_p, W_Θ and W_u. Evidently, different problems

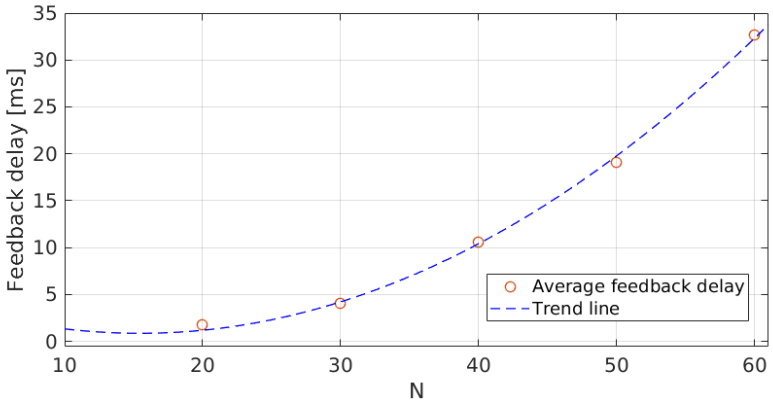


Figure 6.3: Average feedback delay for different N.

will benefit from different weighting values depending on the goal. For example, general navigation can have higher weights for position $W_p = 20$, $W_\Theta = 1$, $W_u = 1$. On the other hand, operations requiring precise attitude tracking can have higher weights for attitude $W_p = 1$, $W_\Theta = 20$, $W_u = 1$. Similarly, operations requiring the least amount of energy possible can use high weights for control $W_p = 1$, $W_\Theta = 1$, $W_u = 20$.

For the time being these matrices will be defined as $W_p = 1$, $W_\Theta = 10$, $W_u = 20$, $W_{\Theta_N} = 1000\, W_\Theta$ and $W_{p_N} = 1000 W_p$. However, several experiments with different values will take place during this work.

Considering now a time step in the position reference in the y axis of $1\,m$, presented in Figure 6.4, the importance of a good prediction time window becomes apparent.

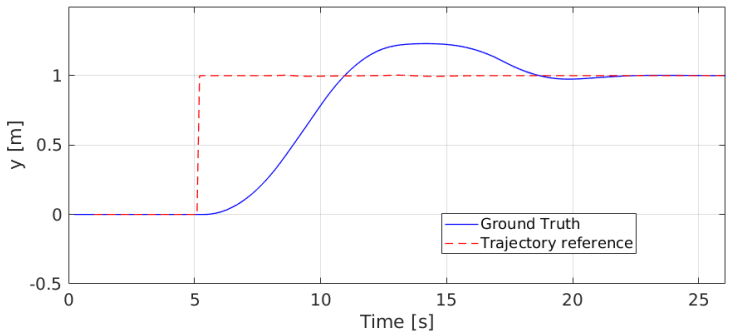
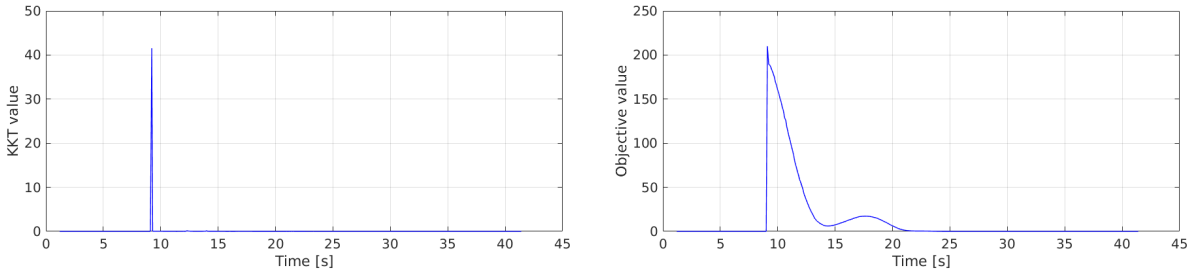


Figure 6.4: NMPC controller step response.

It is visible that the reference is reached with considerable overshoot, meaning finding a solution was more difficult. This difficulty can be evaluated through the KKT conditions violation and the Objective function evaluation along time. These results can be observed in Figure 6.5.

Throughout these next sections, solutions generated by the NMPC controller will be evaluated through the KKT and the Objective evaluation values at each iteration. Low KKT values imply that the solution obtained at each iteration presents a smooth trajectory to the desired state, which can be interpreted in this context as a good convergence indicator. In contrast, high KKT values indicate poor continuity conditions that are reflected in a non-smooth trajectory. In the real-time context, an increasing KKT value presentation, indicates that convergence capability will be eventually lost, given that at each iteration



(a) KKT values (b) Objective evaluation

Figure 6.5: Standalone NMPC step solution evaluation.

solutions become worse. Regarding the Objective evaluation, intuitively, a high value indicates loss of tracking and/or presence of high control inputs, given that the reference for all control inputs is set zero. Consequently, the Objective evaluation can only truly reach near zero values once it is stationary at the desired states.

Observing Figure 6.5, around $t = 8\,s$, when the reference approaches, the KKT and Objective evaluation values increase significantly. Solutions obtained during this phase might be poor. This is happening with a simple step reference due to the small prediction horizon. This means the reference can not be reached within the current prediction horizon. Consequently, the controller simply tries to get closer to the reference fast. When the reference enters the prediction horizon, the system simply does not possess the optimal velocity to approach it and overshoots. Observing the prediction horizon for certain iterations of the position state y in Figure 6.6, note that the predicted trajectory does not reach the reference until approximately $t = 8\,s$ in the simulation. Nonetheless, in this particular case, the controller is able to approach the reference and eventually converge, as seen in Figure 6.4. This might not be the case when considering more complex trajectories.

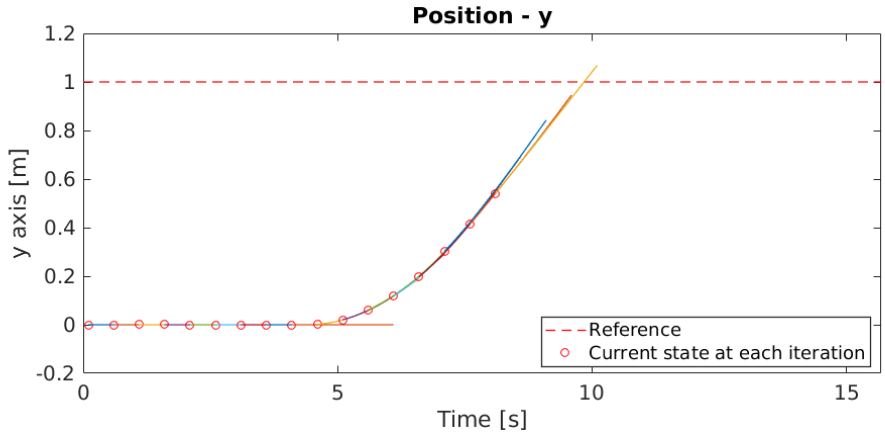


Figure 6.6: Prediction horizons for y state.

This example serves to demonstrate that it would be ideal for references to be closer to the current state. Evidently, providing simple waypoints as references to the control system will severely limit navigation capabilities. Even if a waypoint is close to the current state, a solution may be hard to find given that waypoints do not consider possible constraints or even the system's dynamics.

It is worth mentioning that during this step, a significant variation occurs in the position state z visible in Figure 6.7. This is a consequence of operating with coupled dynamics, meaning that position and attitude are controlled at the same rate. Given that the Space CoBot was designed to maximize the thrust in the z axis of the body frame, the controller might temporarily sacrifice attitude tracking or position tracking to obtain higher accelerations. This can be minimized by carefully adjusting the weight matrices.

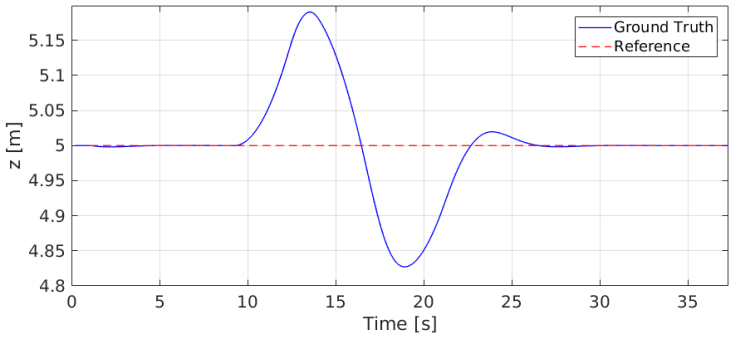


Figure 6.7: Variation induced in z axis during step.

6.2.3 Control constraints

Until this point, no additional constraints were considered, apart from dynamic constraints. The behavior of the controller is now analyzed with the additional control constraint:

$$-4000 \leq \mathbf{u} \leq 4000 \tag{6.1}$$

Note that $\mathbf{u}$ is expressed in force within the controller framework, however, for clarity, control input results will be presented in rpm.

The same step reference is used to verify if the control bounds are respected, presented in Figure 6.8.

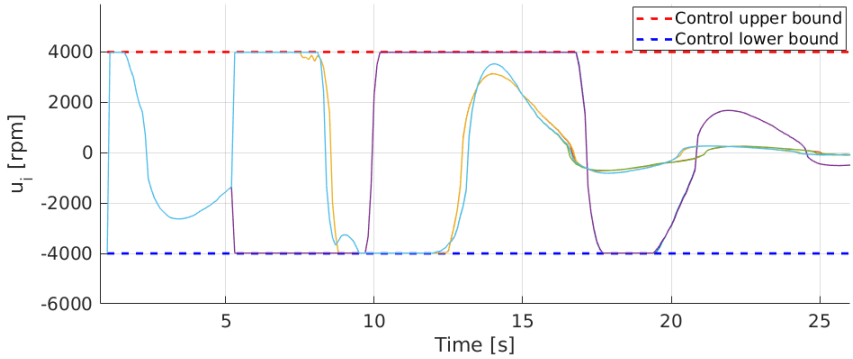


Figure 6.8: Control inputs during step response.

No significant changes are visible with the addition of control bounds. When it comes to $\bar{fd}$ values, bounded and unbounded control produce similar values presenting averages around $3.6\,ms$.

6.3 Offline trajectory generation

Waypoint navigation or path following approaches will not yield sufficiently good results to consider the NMPC controller precise and robust. An offline generation of a feasible trajectory is now considered by taking in consideration the dynamics of each configuration. As mentioned previously, ALTRO enforces strict dynamic feasibility, meaning the solution will be as precise as desired, and not just an approximation. However, this comes at the cost of increased computation times.

Table 6.1: Trajectory optimization computation time.

Trajectory	AL-iLQR	ALTRO
Unconstrained	$59\,s$	$66\,s$
State/control bounds	$86\,s$	$739\,s$
One obstacle	$134\,s$	$892\,s$
One obstacle w/payload	$1732\,s$	$2903\,s$
Two obstacles	$1557\,s$	$2732\,s$
Formation	$260\,s$	$902\,s$
Formation w/ payload	$1198\,s$	$2601\,s$

To form an idea around the required time for generating a feasible trajectory, a summary of approximated computation times corresponding to the trajectories generated and analyzed in this work is presented in Table 6.1. These are approximated times due to the fact that solving the same problem might produce slightly different computation times. A comparison between the AL-iLQR and ALTRO is performed to understand the time difference magnitude. This is relevant given that initial guesses can be provided to the algorithm. One can provide a previously computed ALTRO solution as an initial guess to warm-start a similar AL-iLQR problem and achieve good feasibility in considerable less time. The study of the compromise between tolerable feasibility and computation time is out of the scope of this thesis. Therefore, only ALTRO solutions are used as references, to maximize feasibility regardless of computation time.

6.3.1 Problems formulation

To generate the offline trajectory, a control bound is formulated to reflect (6.1). Additional bounds are added to some state variables, namely linear velocities and angular velocities to avoid extreme values. Given that at the controller level, these bounds result in poor solutions, they will be implemented at the trajectory generation level which imposes the bounds indirectly.

Applied bounds reflect the following

$$-4000 \leq \mathbf{u} \leq 4000 \tag{6.2}$$

$$-0.4 \leq \mathbf{v} \leq 0.4 \tag{6.3}$$

$$-0.3 \leq \boldsymbol{\omega} \leq 0.3 \tag{6.4}$$

Bounds are also imposed to the position state in order to simulate permanent walls, as if the simulation takes place inside a corridor shaped room $4 \times 16 \times 4\,m$, formulated by

$$-2 \leq p_x \leq 2 \tag{6.5}$$

$$-1 \leq p_y \leq 15 \tag{6.6}$$

$$3 \leq p_z \leq 7 \tag{6.7}$$

Table 6.2: ALTRO solver settings.

Time Step	t	0.1
Number of knots	N_{TO}	1001
Penalty scaling	ϕ	2
Constraint tolerance	C_{tol}	1×10^{-5}
Feasibility tolerance	D_{tol}	1×10^{-4}
iLQR iterations		500
AL iterations		40

The selected parameters for the trajectory generation algorithm are detailed in Table 6.2. These were achieved by significantly increasing the number of iterations until feasible solutions are obtained. Then gradually reduce the iteration and observing the solution feasibility. This represents an attempt to reach a compromise between the solutions feasibility and computation time. If a generated solution presents a considerably high feasibility violation, a first approach to solve this problem is to simply increase the number of iterations performed by the algorithm, at the expense of extra computation time. As mentioned earlier, computation time is not the primary concern. However, faster computations times allow less waiting time and unnecessary computation use. The default penalty scaling was also reduced to allow a more gradual constraints penalization. Therefore, as long as the resulting trajectory is feasible, settings are not changed.

The ALTRO weighting matrices will also be individually tuned through the respective diagonal elements, Q_p, Q_Θ and R. The terminal cost weights are defined by the same scalar values multiplied by a factor of 10, $Q_{p_N} = 10Q_p$ and $Q_{\Theta_N} = 10Q_\Theta$. Note that weighting matrices will be defined independently for each problem.

The obtained trajectory comprises all system states $\mathbf{x} = [\mathbf{p}, \mathbf{q}, \mathbf{v}, \boldsymbol{\omega}]$.

Given that the generated trajectory is feasible, the control bounds in the NMPC formulation will be removed. Such bounds can be kept as an extra safety measure, which can also help generate a better first guess through auto-initialization. However, without control bounds, the controller response can be analyzed in cases where the implicit bounds are not respected, which in theory should not happen under normal conditions.

6.3.2 Trajectory execution

In order to guarantee good trajectory tracking, the weighting matrices $\mathbf{W}_k$ and $\mathbf{W}_N$ are tuned to reflect a lower control cost and higher position and attitude costs as a starting point. The weights are then defined as $W_p = 20$, $W_\Theta = 20$, $W_u = 1$, $W_{\Theta_N} = W_\Theta$ and $W_{p_N} = W_p$. Terminal weights are no longer

superior as the objective is to follow a given trajectory and not a specific single reference. Changes applied to these values for different trajectories will be described prior to tracking analysis.

The tracking error of the NMPC controller is analyzed through the following errors

$$||\mathbf{p}||_e = ||\mathbf{p}_{traj} - \mathbf{p}|| \tag{6.8}$$

$$||\Delta\boldsymbol{\Theta}||_e = ||\Delta\boldsymbol{\Theta}\left(\mathbf{q}_{traj}, \mathbf{q}\right)|| \tag{6.9}$$

$$||\mathbf{v}||_e = ||\mathbf{v}_{traj} - \mathbf{v}|| \tag{6.10}$$

$$||\boldsymbol{\omega}||_e = ||\boldsymbol{\omega}_{traj} - \boldsymbol{\omega}|| \tag{6.11}$$

where $\Delta\boldsymbol{\Theta}$ is defined in (5.32). The values considered for each state correspond to ground truth data, obtained from Gazebo, and states with subscript $traj$ are the generated trajectory state values.

Note that tracking errors defined by a simple difference are practically unbounded while the attitude tracking error $||\Delta\boldsymbol{\Theta}||_e$ is bounded between 0 and 1. This has to be taken in consideration when tuning the weights of the quadratic costs.

6.4 Single Space CoBot

Given that the problem solved by the trajectory optimization algorithm is considerably constrained, the generated trajectory is evaluated through the maximum violation, C_{max}, which corresponds to the maximum between constraints and feasibility violations. Space CoBot will also be henceforth referred to by the abbreviation SC.

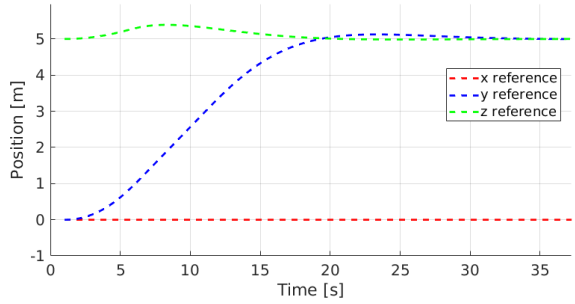
Table 6.3: Single SC step problem formulation.

	Initial condition	Terminal condition
$\mathbf{p}$	$[0,\ 0,\ 5]$	$[0,\ 5,\ 5]$
$\mathbf{q}$	$[1,\ 0,\ 0,\ 0]$	$[1,\ 0,\ 0,\ 0]$
$\mathbf{v}$	$[0,\ 0,\ 0]$	$[0,\ 0,\ 0]$
$\boldsymbol{\omega}$	$[0,\ 0,\ 0]$	$[0,\ 0,\ 0]$
Control weight	R	100
Position weight	Q_p	10
Attitude weight	Q_Θ	400

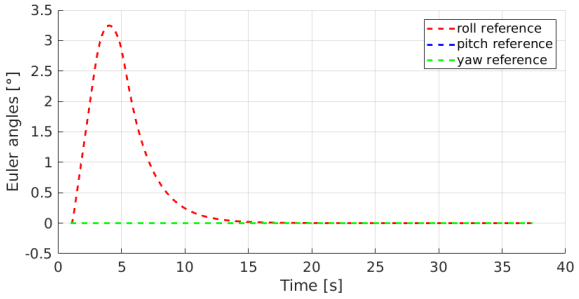
The problem solved by ALTRO is formulated in Table 6.3. A relatively high weight is selected for attitude related states to prevent variations in attitude during the step. The control weight also has a superior value when compared to the position weight matrix. The obtained trajectory can be visualized in Figure 6.9. Note that even though a trajectory is generated for the entire state $\mathbf{x}$, for simplicity, only the position and attitude trajectories are presented. A presentation of all states would yield a significantly high amount of figures that may not be relevant for analysis purposes.

This trajectory yields $C_{max} = 8.97 \times 10^{-5}$.

As expected, the generated trajectory presents variations in states not directly involved with the



(a) Position trajectory

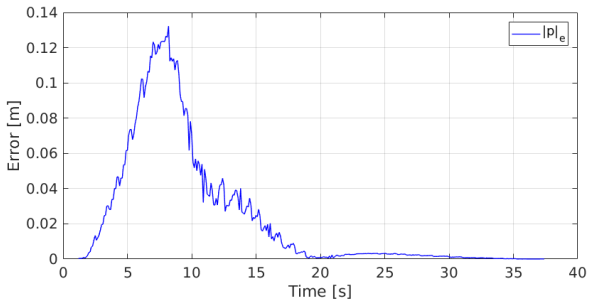


(b) Attitude trajectory

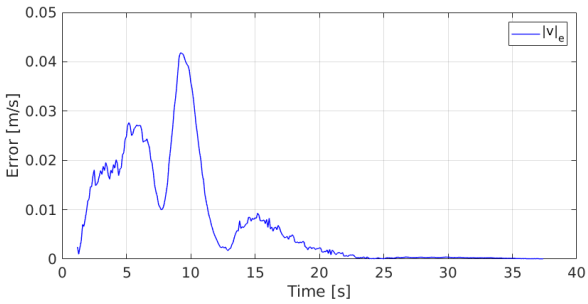
Figure 6.9: Generated step trajectory for single SC.

desired final state, namely a variation in the z axis and a slight variation in the roll angle, ϕ. Increasing the weights associated with the position would likely reproduce the same variations, given that all position state weights receive the same values. To obtain a trajectory without any undesired variation, the weights associated with the variations would have to be individually tuned.

The NMPC controller tracking error can be visualized in Figures 6.10 and 6.11 for all states of the system. Overall, the tracking errors are relatively low. However, a distinct surge in the position tracking error is visible which eventually disappears as the error converges to zero. This could be a consequence of the error introduced by the first iteration initialization where there is no information from a previous solution. Further experiments are required to verify this.

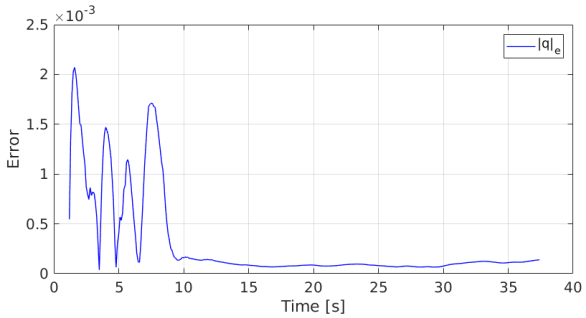


(a) Position tracking error

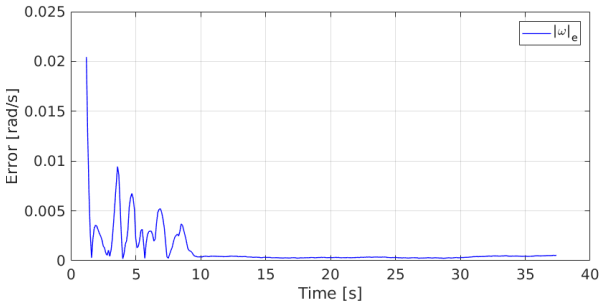


(b) Linear velocity tracking error

Figure 6.10: Step tracking errors $||\mathbf{p}||_e$ and $||\mathbf{v}||_e$ for single SC.



(a) Attitude tracking error



(b) Angular velocity tracking error

Figure 6.11: Step tracking errors $||\Delta\Theta||_e$ and $||\boldsymbol{\omega}||_e$ for single SC.

The tracking errors presented for this step trajectory will serve as baseline for comparison with in-

creasingly more complex trajectories.

Additional conclusions regarding the performance of the controller can also be achieved through the KKT and Objective evaluation values in Figure 6.13 and a distribution of fd during execution in Figure 6.12. Observing the KKT values during the surge in position tracking error, it does present several spikes that indicate discontinuities between solutions, which can result in poor control inputs. Objectives values also increase during the error surge, which is expected given that higher errors will result in higher Objective evaluations.

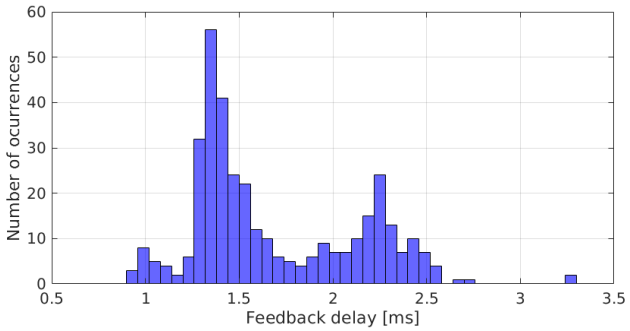
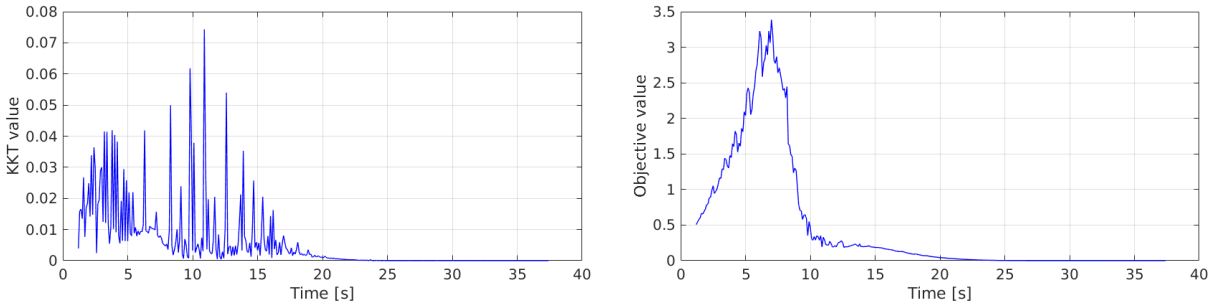


Figure 6.12: fd for single SC step trajectory.

Observing Figure 6.12, fd values do not present a sparse distribution, achieving $\bar{fd} = 1.8\,ms$. This value will also serve as baseline to compare with subsequent results. A slight improvement is visible when compared to the standalone NMPC $\bar{fd} = 3.6\,ms$ during a shorter step reference, which already shows evidence that providing feasible trajectories to the NMPC is better. This delay does not seem to affect tracking performance in any noticeable way, as convergence is obtained.

In addition to the decrease of $\bar{fd}$, providing a feasible trajectory to the controller as reference causes a significant drop in the effort of finding solutions. This is very clear in Figure 6.13 as KKT and Objective evaluation values present very small values along the trajectory.



(a) KKT value

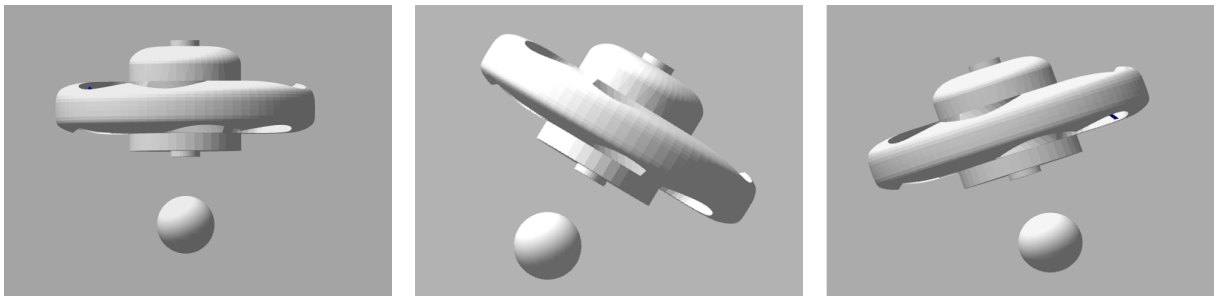
(b) Objective evaluation

Figure 6.13: NMPC performance with step trajectory for single SC.

6.4.1 Payload carrying

One objective of this thesis is to explore payload carrying with modelled payloads. This means that the system model is modified to accommodate the changes caused by the payload. Previous work within

the Space CoBot project in [4] verified that loss of attitude tracking occurs when dealing with unmodelled payloads visible in Figure 6.14, alongside with significant oscillation regarding position tracking.



(a) Start position (b) Acceleration phase (c) Deceleration phase

Figure 6.14: NMPC response to step reference with unmodelled payload.

Due to the system's CoM displacement caused by the payload, which the controller is not aware, both attitude and position tracking are severely affected even though convergence is still obtained.

With the present controller, one way to circumvent this effect could be to significantly increase the weight values related to attitude tracking. However this is not desirable as it increases the time required time to converge to the desired position, meaning position tracking is considerably reduced.

A spherical $6\ kg$ payload is considered in this scenario. The model provided to the NMPC controller and the trajectory optimization algorithm is now described by (4.22). The ability to control the system is significantly improved. Following (4.36) it is possible to provide the desired position for the payload and transform that position into the equivalent position of the Space CoBot's CoM. This is necessary given that both the controller and trajectory optimization algorithm express position at this point. Without further modifications, the controller can be used as is.

Table 6.4: Single SC with payload step formulation.

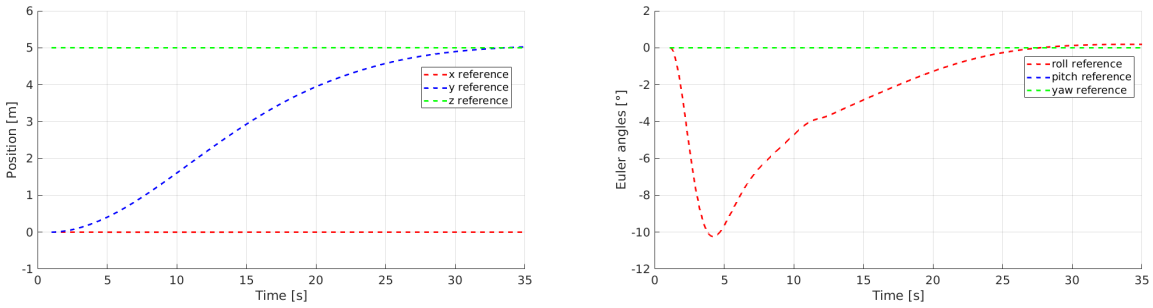
	Initial condition	Terminal condition
$\mathbf{p}$	$[0,\ 0,\ 5]$	$[0,\ 5,\ 5]$
$\mathbf{q}$	$[1,\ 0,\ 0,\ 0]$	$[1,\ 0,\ 0,\ 0]$
Control weight	R	100
Position weight	Q_p	10
Attitude weight	Q_Θ	400

The trajectory optimization problem is formulated with the parameters presented in Table 6.4. Note that information regarding $\mathbf{v}$ and $\boldsymbol{\omega}$ is omitted from this point forward as it will never change.

Even with such high attitude costs, a trajectory with variation in attitude will still yield lower overall cost, as it can be seen in the generated trajectory in Figure 6.15. Given the additional mass from the payload, the overall cost is minimized by facing the vehicle's z axis towards the desired direction. As mentioned before, the produced thrust is maximized in the z axis of the body frame of the Space CoBot.

This trajectory yields $C_{max} = 1.6 \times 10^{-2}$.

The controller performance is worse when compared to the step without payload, although it still achieves the goal. In order to improve tracking, the cost weights are increased for both position and

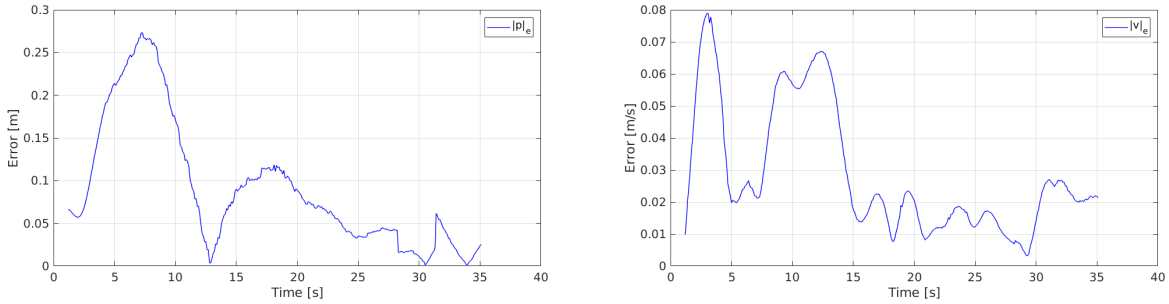


(a) Position trajectory

(b) Attitude trajectory

Figure 6.15: Generated step trajectory for single SC with payload.

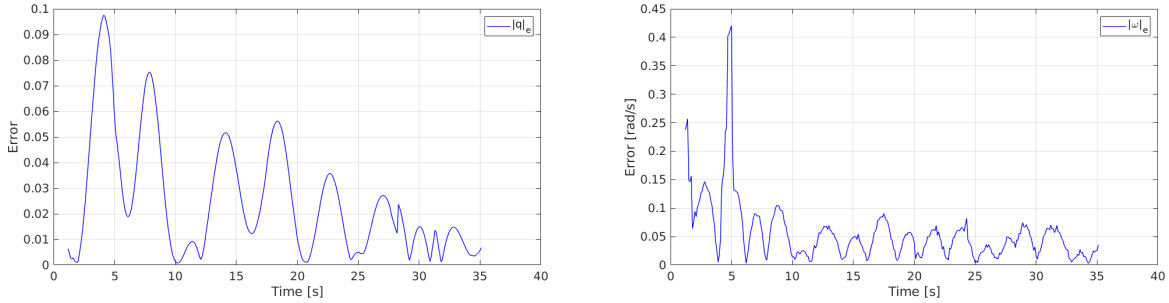
attitude, $W_p = 50$, $W_\Theta = 50$, $W_u = 1$. The resulting tracking error can be evaluated through Figures 6.16 and 6.17.



(a) Position tracking error

(b) Linear velocity tracking error

Figure 6.16: Step tracking errors $||\mathbf{p}||_e$ and $||\mathbf{v}||_e$ for single SC with payload.



(a) Attitude tracking error

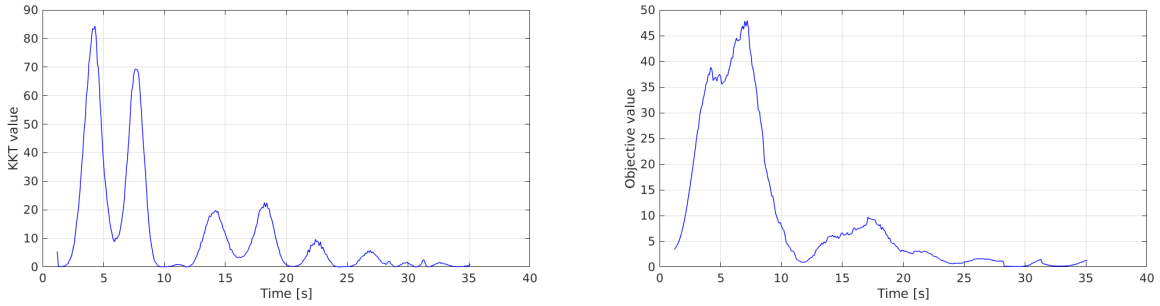
(b) Angular velocity tracking error

Figure 6.17: Step tracking errors $||\Delta\Theta||_e$ and $||\boldsymbol{\omega}||_e$ for single SC with payload.

There is a visible increase in tracking errors in this experiment. This could be an indication that the presence of the payload deteriorates the quality of the solutions. The attitude error i relatively small and comparable to previous experiments. However, higher errors in the position are verified.

Regarding the controller performance presented in Figure 6.18, although KKT values have an oscillating display, they present a visible increase, which further verifies the bad quality of the solutions obtained. The oscillating nature of the KKT and high Objective values can be attributed to the fact that the nodes where the evaluations occur are not coupled in time.

Note that increasing the costs at the controller level present a tendency to increase the KKT values. This requires some attention once higher KKT values can cause instability in the controller feedback response and quality of the solution.



(a) KKT value (b) Objective evaluation

Figure 6.18: NMPC performance with step trajectory for single SC with payload.

6.5 Object avoidance

To consider obstacles, first they are modelled and introduced in the simulation environment. Obstacles navigation is outside the scope of this work. Therefore, the NMPC controller is oblivious to their presence. However, constraints will be added to the trajectory optimization problem to represent these obstacles, namely spherical constraints (5.43). This addition is performed manually, meaning no algorithm takes part in identifying obstacles. This allows a performance study of the trajectory optimization algorithm in the presence of obstacles while leaving out the obstacle identification problem, that can be solved in future work. From the controller point of view, there is no difference from any other generated trajectory. This reflects the focus in Guidance and Control and not Navigation as stated previously.

The considered sphere obstacles have a radius of $1\,m$. Considering that position measurements are taken at the Space CoBot's CoM, a clearance radius has to be considered according to each system configuration, r_{clear}. This radius defines a sphere that comprises the system. Considering a sphere instead of exact clearance distances in all 3 axes allows a collision verification based on a single position, such as the system's CoM, regardless of the system orientation. A safety margin of $0.5\,m$ is also included in the spherical constraint to obtain a safety barrier that can be visualized in Figure 6.19.

The spherical obstacle constraint is then formulated by

$$(x_k + y_c)^2 + (y_k - y_c)^2 + (z_k - z_c)^2 \geq (1 + 0.5 + r_{clear})^2, \quad k = 1, \dots, N \tag{6.12}$$

where $(x_c,\, y_c\,, z_c)$ corresponds to the sphere's center point.

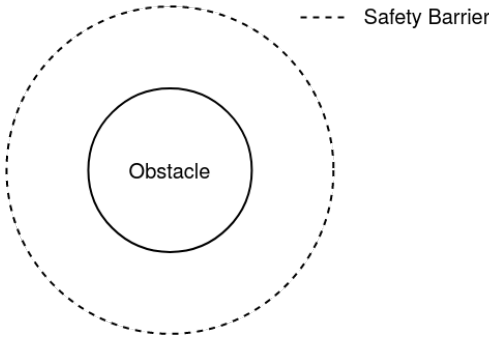


Figure 6.19: Collision safety barrier created by constraint.

6.5.1 Single Space CoBot single obstacle trajectory

A single obstacle is considered to form the first obstacle avoidance problem. The clearance radius of the Space CoBot considered is $r_{clear} = 0.5\,m$. The simulation environment can be visualized in Figure 6.20.

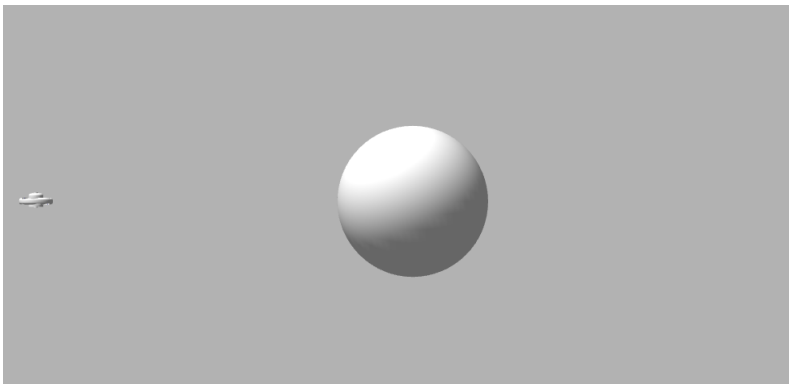


Figure 6.20: Single obstacle problem simulation environment.

Table 6.5: Single SC single obstacle problem formulation.

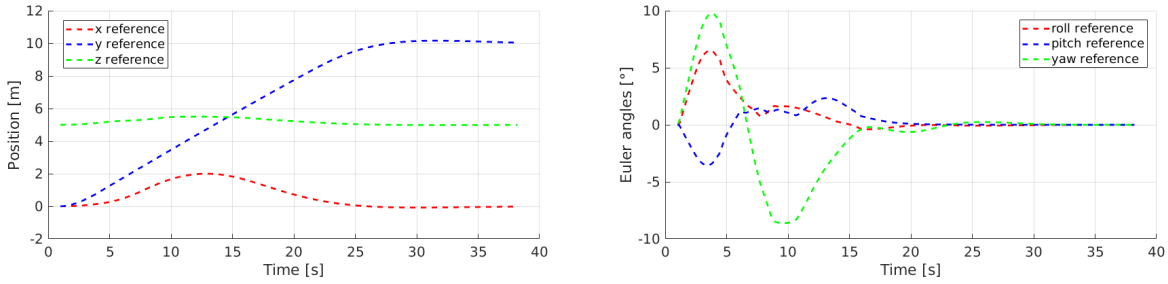
	Initial condition	Terminal condition
$\mathbf{p}$	$[0,\ 0,\ 5]$	$[0,\ 10,\ 5]$
$\mathbf{q}$	$[1,\ 0,\ 0,\ 0]$	$[1,\ 0,\ 0,\ 0]$
Control weight	R	100
Position weight	Q_p	10
Attitude weight	Q_Θ	400
Obstacles	$(0,\ 5,\ 5)$	

The problem is formulated through the parameters presented in Table 6.5. The obstacle parameter in Table 6.5 indicates the center point of the sphere.

As mentioned before, the tendency to sacrifice attitude tracking to favor faster position convergence is minimized by increasing the weight costs associated with attitude tracking. However, observing Figure 6.21, the generated trajectory shows a noticeable attitude deviation from the provided reference. This is not harmful given that the objective of this solution is to converge to the final position while avoiding

the obstacle. Should the goal be to accurately maintain a given attitude, the respective costs have to be further increased.

It is also worth noting that produced thrust in the y axis of the body frame is higher than the produced thrust in the x axis of the body frame. This is visible in Figure 6.21 as the yaw angle presents two distinct variations that consist with the orientation of the Space CoBot to the direction of motion.



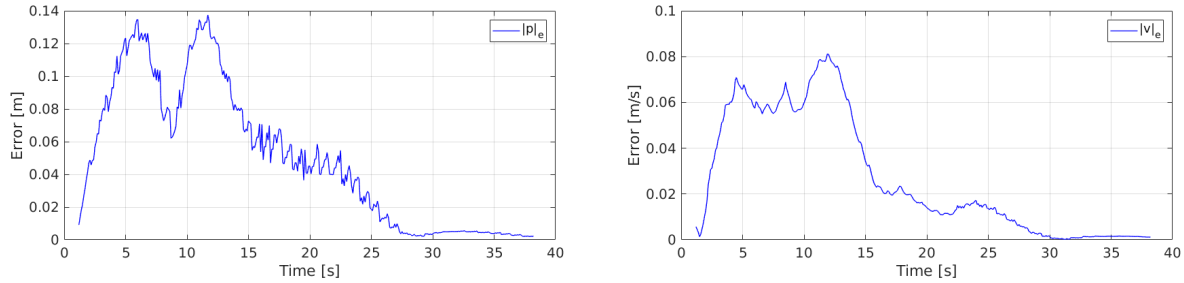
(a) Position trajectory

(b) Attitude trajectory

Figure 6.21: Single obstacle generated trajectory.

This trajectory yields $C_{max} = 1.21 \times 10^{-4}$.

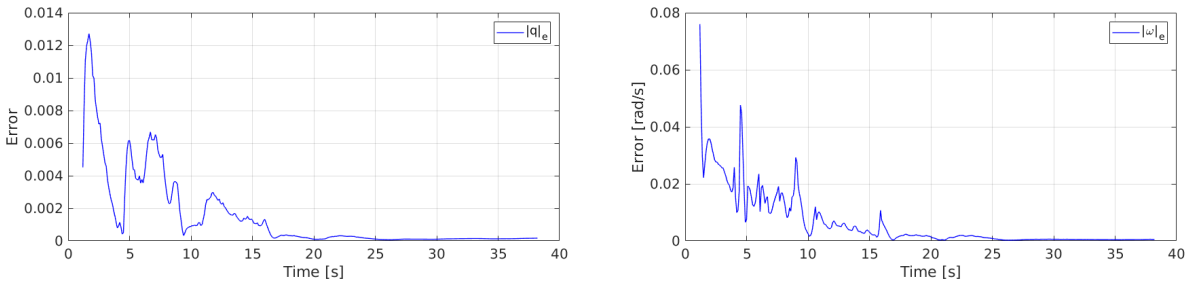
Regarding the controller tracking capabilities, presented in Figures 6.22 and 6.23, slightly higher errors can be observed when compared to the step trajectory tracking, Figures 6.10 and 6.11. Since the controller cost weights are kept constant, this slight variation can be attributed to several causes.



(a) Position tracking error

(b) Linear velocity tracking error

Figure 6.22: Single obstacle tracking errors $||\mathbf{p}||_e$ and $||\mathbf{v}||_e$.



(a) Attitude tracking error

(b) Angular velocity tracking error

Figure 6.23: Single obstacle tracking errors $||\Delta\Theta||_e$ and $||\boldsymbol{\omega}||_e$.

Important to note is the fact that attitude tracking is more demanding in this trajectory. Trying to find optimal solutions that satisfy both position and attitude convergence increases the difficulty. This is

visible as the two main surges in $\|\mathbf{p}\|_e$ coincide with the main variations of attitude. Time delays can be excluded as a cause. Observing Figure 6.24, the distribution of the controller computation is similar to previous cases with $\bar{fd} = 1.8\,m$. Even considering the cumulative delay in the control input feedback, the controller should be able to predict necessary corrections in order to keep an accurate tracking.

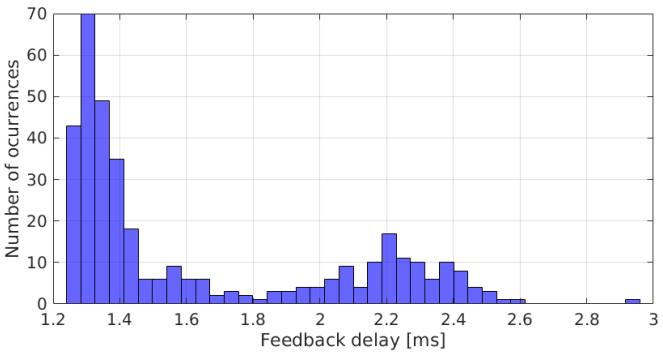
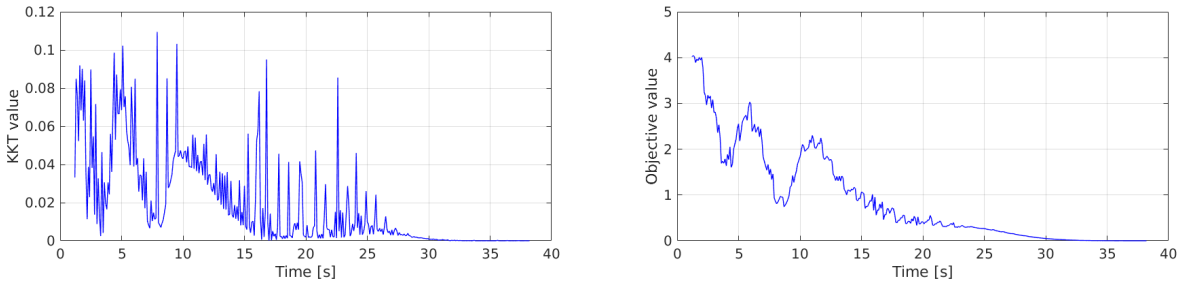


Figure 6.24: fd for single obstacle trajectory.

Observing Figure 6.25, the KKT values increase very slightly and presents significant spikes along the entire duration of the execution. This indicates the solutions being computed are worse than in previous experiments when it comes to discontinuities. Although the general tendency is to converge, which eventually happens, bad solutions are being applied when these significant variations in KKT values occur. This is further confirmed by the increase in the objective value and the visible spikes, which is expected if the tracking ability is lost. Nonetheless, this small loss of tracking is visible in the beginning of the trajectory, which means the controller is able react in order to resume an accurate tracking.



(a) KKT value

(b) Objective evaluation

Figure 6.25: NMPC performance with single obstacle trajectory.

Once again the suspicion is that this loss of tracking can then be attributed to initial poor solutions. Given a slower trajectory, the loss of tracking should be lower as the controller has more time to apply corrections.

It is important to note that the controller is able to resume tracking effortlessly which might involve using higher control inputs than the bounds imposed by the trajectory, which indeed happens as can be visualized in Figure 6.26. This is possible once the NMPC control bounds were removed. Although this should not happen, it has a positive effect in this context, meaning that when tracking errors increase or the system is perturbed, higher control inputs can be used to apply the necessary corrections to resume

tracking. Figure 6.26 also appears to confirm the suspicion that bad solutions are provided along the execution as the control trajectory is not very smooth.

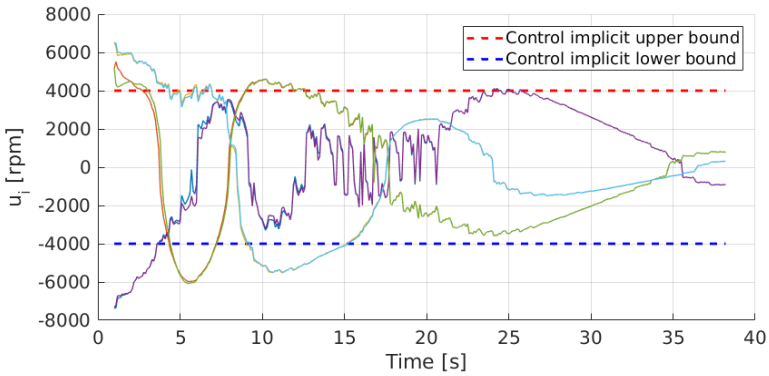


Figure 6.26: Single obstacle trajectory execution control inputs.

A 3D visualization of the trajectory execution can be observed in Figure 6.27 for visual reference. The execution is smooth and demonstrates little deviation from the reference trajectory.

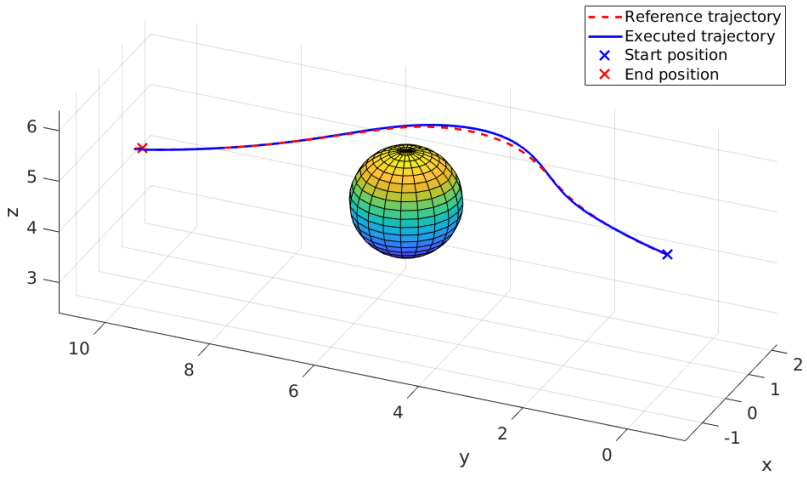


Figure 6.27: 3D visualization of single obstacle trajectory execution.

6.5.2 Single Space CoBot with payload single obstacle trajectory

The same scenario as presented in Figure 6.20 is now considered for a single Space CoBot while carrying a payload.

The problem is formulated based in the parameters presented in Table 6.6. As variations in attitude were visible in the previous cases, the attitude associated cost is increased once again.

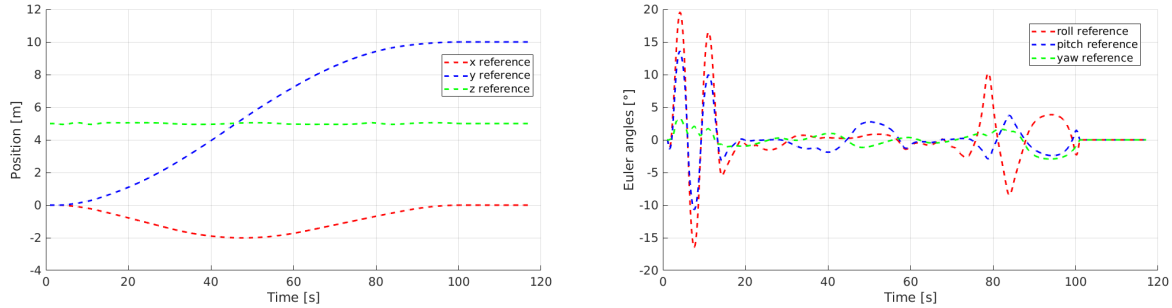
Similarly to previous cases, the generated trajectory, visible in Figure 6.28, presents a significant variation in attitude. The attitude tracking cost has been increased through different trajectories to observe if this attitude can be maintained constant. However, the optimized trajectory once again favors a loss of attitude tracking, but not in the same way as previous experiments. In this case, as attitude weights

Table 6.6: Single obstacle single SC with payload problem formulation.

	Initial condition	Terminal condition
$\mathbf{p}$	$[0,\ 0,\ 5]$	$[0,\ 10,\ 5]$
$\mathbf{q}$	$[1,\ 0,\ 0,\ 0]$	$[1,\ 0,\ 0,\ 0]$
Control weight	R	100
Position weight	Q_p	10
Attitude weight	Q_Θ	800
Obstacles	$(0,\ 5,\ 5)$	

are increased, an initial orientation is performed to maximize thrust and perform an initial acceleration. Quickly after, the orientation returns to the reference. This is an indication that it costs less to sacrifice initial attitude tracking in favor of better acceleration. This can also be associated with the fact that with the addition of a payload, and bounds on actuation, the acceleration achieved is lower when compared with the same experiment without payload, which makes the loss of attitude worth it.

Further increases to attitude weights that guarantee that sacrificing attitude tracking costs more are required to maintain the attitude constant. Note that in this particular case, the generated trajectory presents a left deviation from the obstacle. This is normal given that there might be different solutions to the problem that present the same overall cost and C_{max}. This implies that running the same problem with different initial guesses might result in different optimal trajectories.



(a) Position trajectory

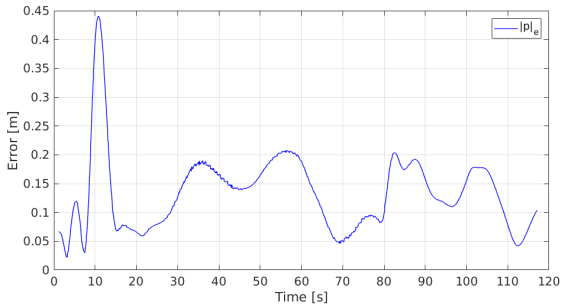
(b) Attitude trajectory

Figure 6.28: Single obstacle single SC with payload generated trajectory.

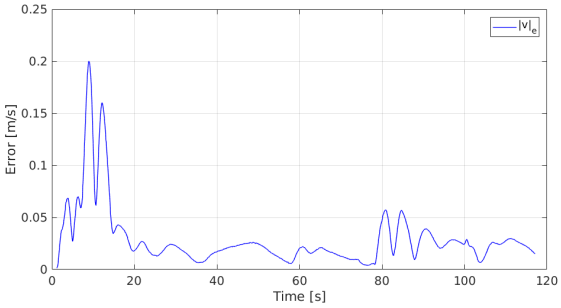
This trajectory yields $C_{max} = 6.1 \times 10^{-5}$.

Regarding tracking performance presented in Figures 6.29 and 6.30, the observed error is significantly higher when compared with the case without payload. There is a characteristic initial surge in both position and attitude related errors present in all experiments so far followed by an eventual convergence to zero. However, in this experiment, errors in position are visible throughout the trajectory. This could be an indication that the presence of the payload causes the quality of the solutions to drop. The attitude error is considerable and may be causing errors in the position as well.

When analyzing the KKT and Objective evaluation values in Figure 6.31, there is an increase in the Objective evaluation. This is expected given that the same NMPC cost weights are being used and control inputs are part of the objective function. Therefore, as the total mass of the system is much higher due to the payload so should be the control inputs. Visible in Figure 6.32 the controls are clearly

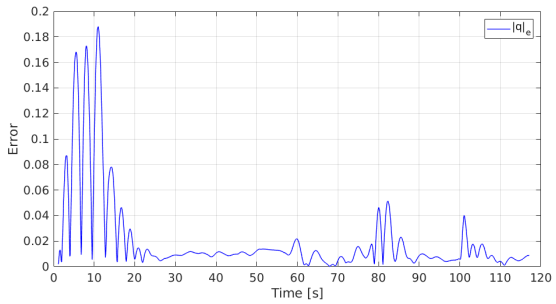


(a) Position tracking error

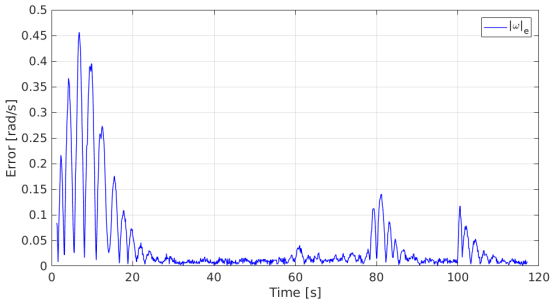


(b) Linear velocity tracking error

Figure 6.29: Single obstacle single SC with payload tracking errors $||\mathbf{p}||_e$ and $||\mathbf{v}||_e$.



(a) Attitude tracking error

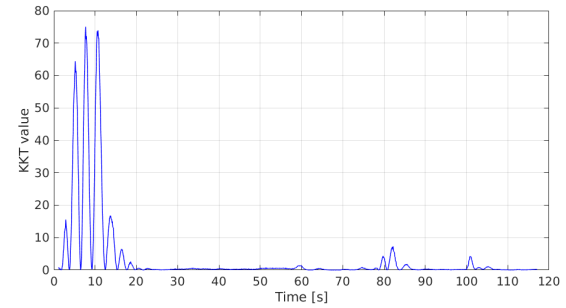


(b) Angular velocity tracking error

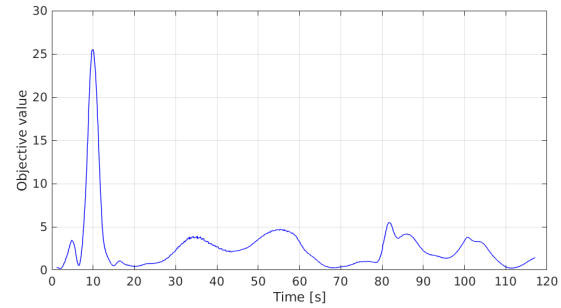
Figure 6.30: Single obstacle single SC with payload tracking errors $||\Delta\Theta||_e$ and $||\boldsymbol{\omega}||_e$.

operated closely to the implicit bounds, which is not observed in the case without payload. The very non smooth control trajectory also shows the discontinuities present between each time step. Recalling that the solver within the NMPC controller only performs one iteration, obtaining smooth control trajectories becomes more difficult with increases in system complexity.

The fd values present a similar distribution to previous experiments and $\bar{fd} = 1.8\,ms$ which is also similar to previous experiments.



(a) KKT value



(b) Objective evaluation

Figure 6.31: NMPC performance with single obstacle single SC with payload trajectory.

A 3D visualization of the trajectory execution can be observed in Figure 6.33 for visual reference. Note that the execution is not as smooth as previous experiments. There are also visible deviations from the reference trajectory at the beginning and end of the trajectory.

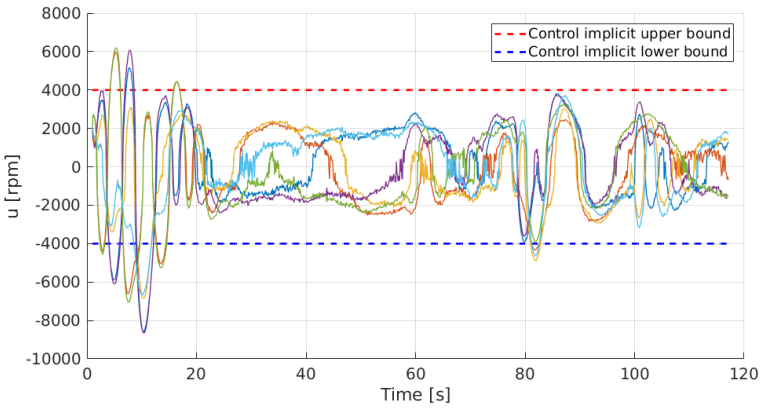


Figure 6.32: Single obstacle single SC trajectory execution control inputs.

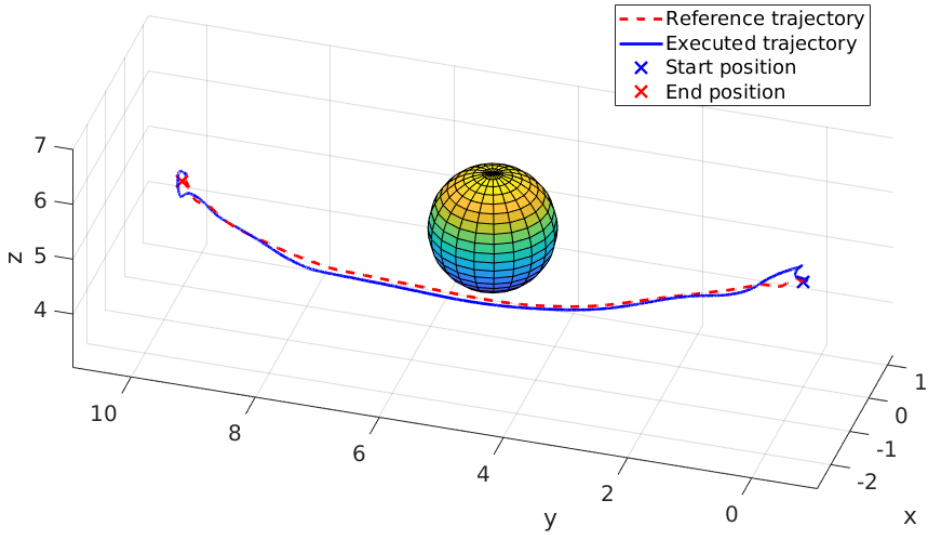
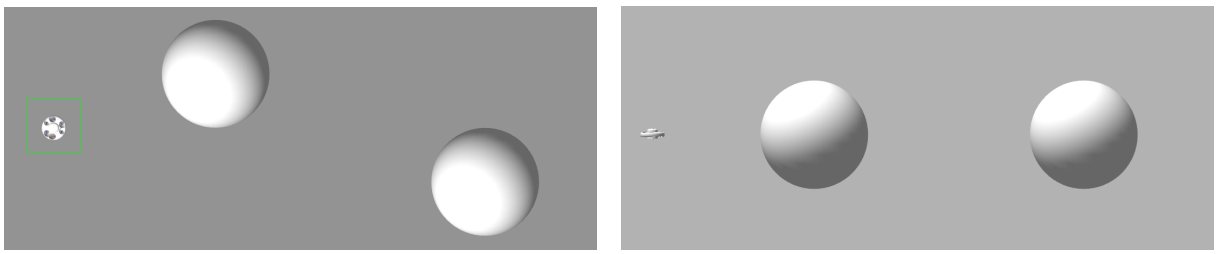


Figure 6.33: 3D visualization of single obstacle single SC with payload trajectory execution.

6.5.3 Two obstacles trajectory

Two spherical obstacles are now considered. This test is performed to evaluate the trajectory optimization algorithm capability of dealing with highly constrained problems. Therefore, this scenario is only considered once and uses the single Space CoBot configuration. The simulation environment can be visualized below in Figure 6.34.



(a) Top view (b) Side view

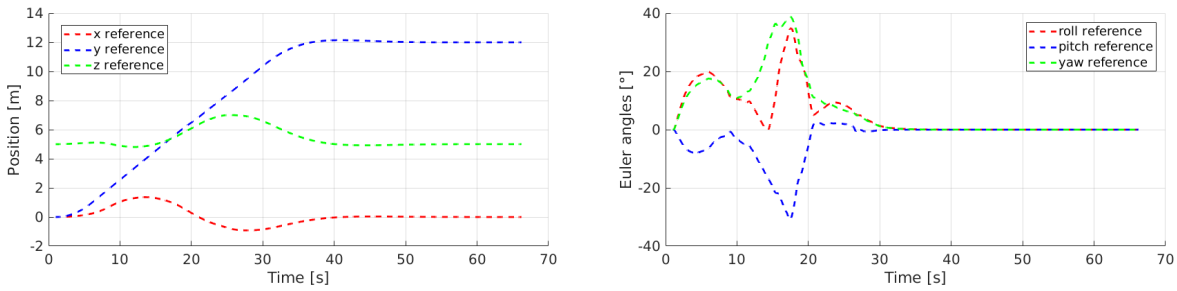
Figure 6.34: Two obstacles problem simulation environment.

Table 6.7: Two obstacles single SC problem formulation.

	Initial condition	Terminal condition
$\mathbf{p}$	$[0,\ 0,\ 5]$	$[0,\ 12,\ 5]$
$\mathbf{q}$	$[1,\ 0,\ 0,\ 0]$	$[1,\ 0,\ 0,\ 0]$
Control weight	R	100
Position weight	Q_p	10
Attitude weight	Q_Θ	1000
Obstacles	$(-1,\ 3,\ 5)\ (1,\ 8,\ 5)$	

It is considered a problem with several constraints due to the fact that, with the exception of the attitude quaternion, all states are bounded. Control is also bounded and two obstacles are considered, formulated by two constraints of the form (6.12). Additionally, the attitude associated costs were increased to limit the variation in attitude. The attitude costs are now 100 times superior to position costs. In Table 6.7, the parameters used to formulate this problem are presented.

Initial attempts to generate a solution yielded poor results, characterized by considerable high C_{max} values. The initial number of iLQR iterations is then raised to 800, which is sufficient to achieve a good feasible trajectory, presented in Figure 6.35.



(a) Position trajectory

(b) Attitude trajectory

Figure 6.35: Two obstacle single SC generated trajectory.

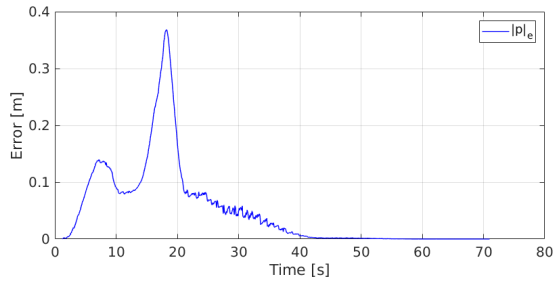
This trajectory yields $C_{max} = 4.29 \times 10^{-4}$.

It still shows significant variation in the attitude trajectory. It is even more evident in this scenario. It becomes apparent that a significant increase in the attitude is required to maintain a steady attitude throughout the entire trajectory. It has already been mentioned several times that attitude variations that benefit faster convergences are expected. This serves to demonstrate the order of magnitude required for the attitude cost when compared to position or control costs in order to achieve accurate attitude tracking. Another reason for this to be the case is that the position error is practically unlimited while the attitude error is bounded as mentioned in Section 6.3.2. Evidently, the quadratic cost formulation does not suit well this attitude error computation as costs can not be tuned proportionately to each other. Sufficiently high attitude costs will generate a trajectory without undesired variations. Meanwhile, a trajectory with several variations poses a more demanding task for the NMPC, which is the subject of study of this work.

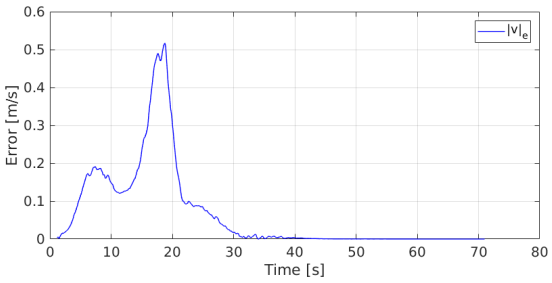
The cost weights at the controller level corresponding to position and attitude are also increased.

The aim is to force a faster convergence to the desired trajectory. This change is necessary given the significant error present in first attempts with the previous cost weights. The NMPC now reflects $W_p = 200$, $W_\Theta = 200$, $W_u = 1$.

Observing Figures 6.36, the position tracking error presents a visible surge of high values.



(a) Position tracking error

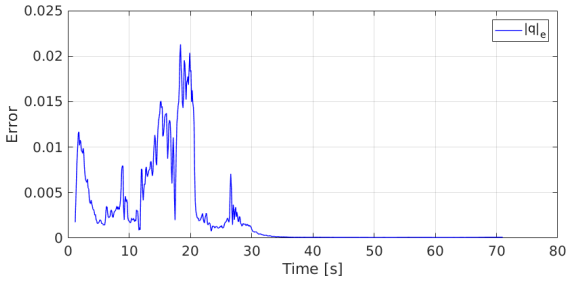


(b) Linear velocity tracking error

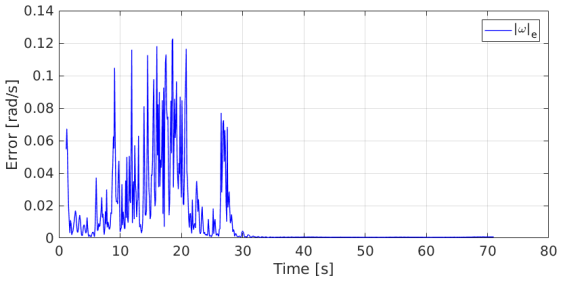
Figure 6.36: Two obstacles single SC tracking errors $||\mathbf{p}||_e$ and $||\mathbf{v}||_e$.

Other phenomenon can be identified while observing the equally high values of the linear velocity tracking error. When the controller deviates from the desired trajectory, linear velocity references may differ significantly from the velocities registered during the recuperation. Evidently this will originate higher tracking errors for velocity, which makes this error a bad analysis source when deviations such as these occur. If the linear velocity tracking error was included in the objective function of the controller, one can extrapolate that if deviations occurred, convergence would probably be lost, given that conflicting references would be provided to the objective function. However, this is not the case.

Similarly to position tracking, attitude tracking, in Figure 6.37, presents very little error while at the same time showing slightly increased angular velocity tracking errors.



(a) Attitude tracking error

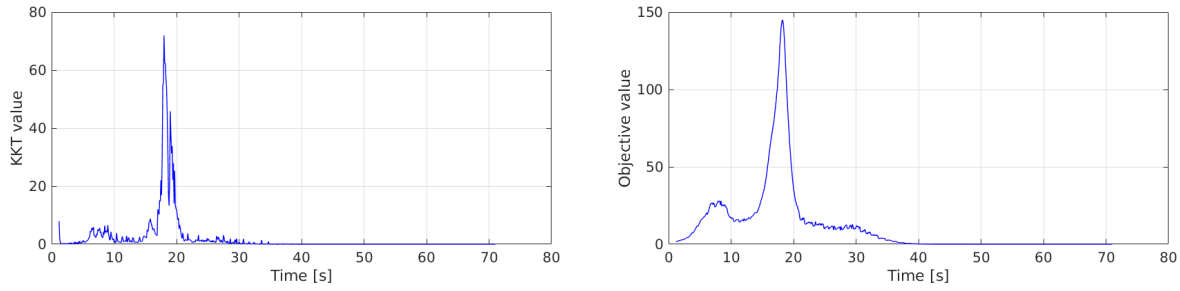


(b) Angular velocity tracking error

Figure 6.37: Two obstacles single SC tracking errors $||\Delta\Theta||_e$ and $||\boldsymbol{\omega}||_e$.

The use of higher costs yields higher KKT values, as visible in Figure 6.38. However, it is mainly a spike that soon ends and quickly returns to relatively low values. As mentioned before, if the KKT values stayed high for too long, convergence could be lost, which was not the case. The Objective evaluation value also shows higher values due to the increase in costs.

Regarding NMPC fd values, Figure 6.39 shows a different distribution when compared to previous experiments with a significant amount of occurrences above $2\,ms$. Consequently, $\bar{fd} = 1.9\,s$ which can almost be considered negligible when compared to previous scenarios considering the used time step.



(a) KKT value (b) Objective evaluation

Figure 6.38: NMPC performance with two obstacles single SC trajectory.

This small increase can be correlated with the increase of KKT values.

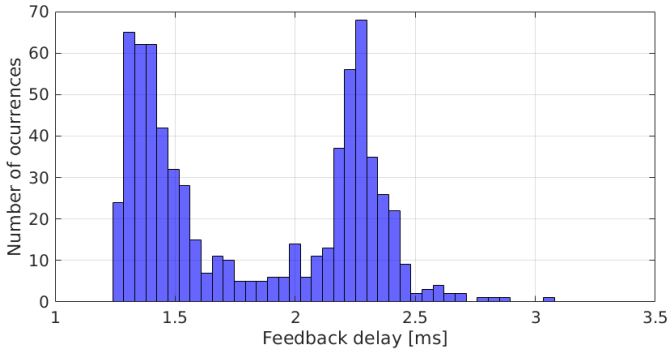


Figure 6.39: fd for two obstacles single SC trajectory.

Feedback delays appear to suffer little change when trajectory complexity increases.

A 3D visualization of the trajectory execution can be observed in Figure 6.40 for visual reference. It also presents a good tracking performance with a small loss when flying between the obstacles execution. The controller appears to sacrifice position tracking for attitude tracking as attitude tracking error is lower than previous experiments.

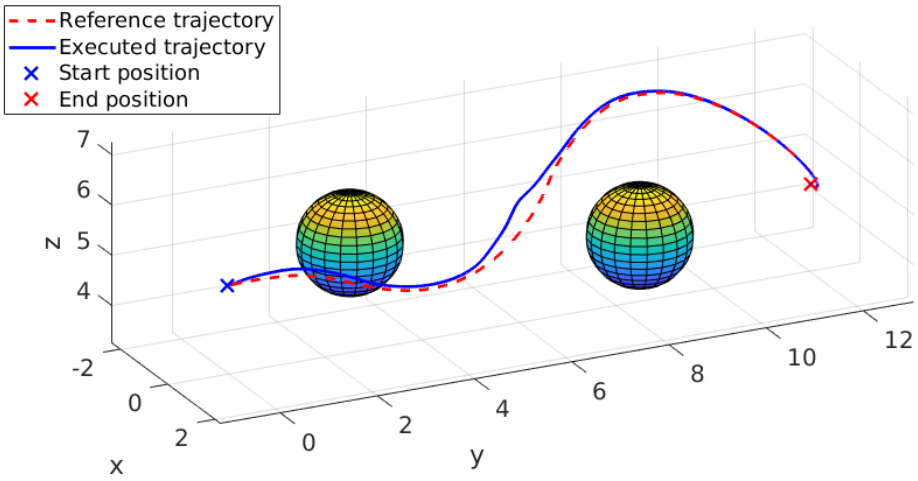


Figure 6.40: 3D visualization of two obstacles single Space CoBot trajectory execution.

6.6 Formation with payload object avoidance

The positions state bounds are modified to increase the size of the simulated corridor. Size is increased to $8 \times 16 \times 4\,[m]$. This is necessary given that the formation requires a greater r_{clear} compared to previous configurations. The considered clearance radius is now $r_{clear} = 1\,m$.

The problem considered at this stage comprises two phases, approaching a desired payload and then transporting it back to the start position. A scenario with one obstacle similar to the one in Figure 6.20 is considered.

At the trajectory optimization level, this problem will be divided into the approach problem and the transport problem. By doing this, one can apply different constraints to both phases. Additionally, considering the problem as a whole would result in a very complex problem, given the size of the resulting optimization horizon and amount of constraints. Consequently, it would require a significant amount of time to achieve a good solution, which is already very high when compared to the real-time framework, as presented in Table 6.1.

The simulation environment can then be visualized in Figure 6.41.

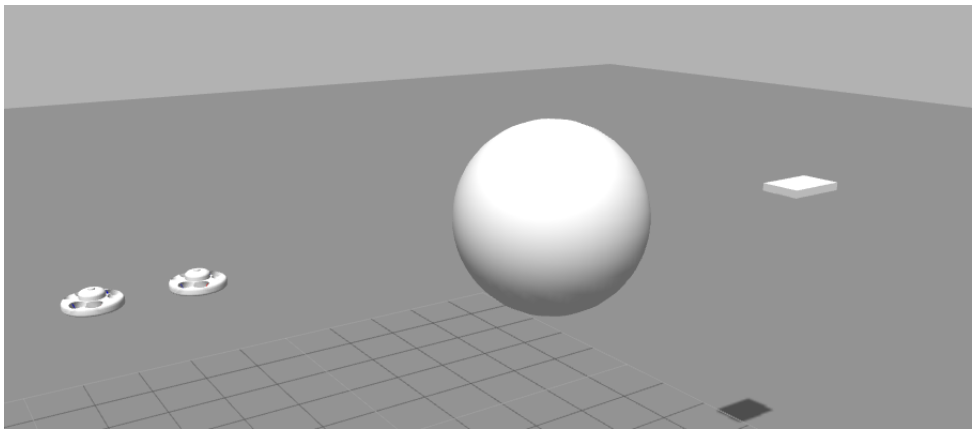


Figure 6.41: Formation problem simulation environment.

Table 6.8: Formation configuration obstacle problem formulation.

	Initial condition	Pickup condition	Terminal condition
$\mathbf{p}$	$[0,\ 0,\ 5]$	$[0,\ 8,\ 5]$	$[0,\ 0,\ 5]$
$\mathbf{q}$	$[1,\ 0,\ 0,\ 0]$	$[1,\ 0,\ 0,\ 0]$	$[1,\ 0,\ 0,\ 0]$
R		100	100
Q_p		10	10
Q_Θ		400	2000

In order to make the trajectories more interesting and visually easier to understand, both phases are forced to take different deviations from the obstacle, meaning the approach phase will favor a right deviation, and the transport phase will favor a left deviation, assuming a constant perspective on the obstacle. This is enforced by altering the required position state bounds. Additionally, the approach phase will be constrained to approach the payload from above. Both problems are formulated according to the information provided in Table 6.8.

The position state bounds considered for this problem are described by

$$-4 \leq p_x \leq 4 \tag{6.13}$$

$$-1 \leq p_y \leq 15 \tag{6.14}$$

$$3 \leq p_z \leq 7 \tag{6.15}$$

and the remaining states and control are bounded as in (6.3) and (6.4), and (6.2), respectively.

6.6.1 Approach phase

The position state bounds corresponding to the z and x axes considered during this phase are modified to reflect

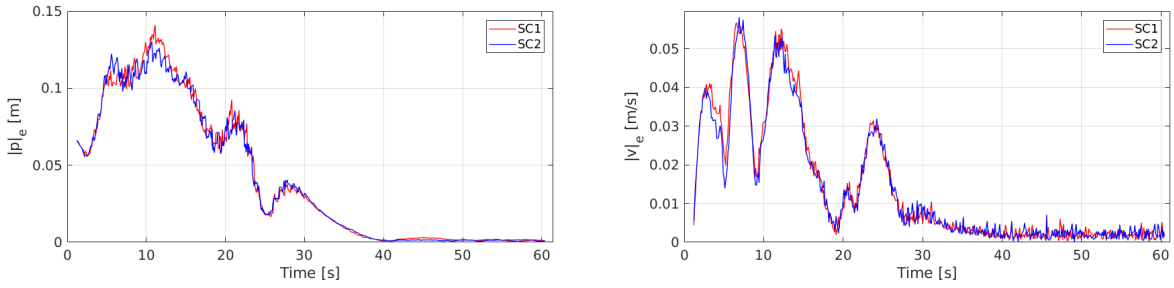
$$4 \leq p_z \leq 7 \tag{6.16}$$

$$0 \leq p_x \leq 4 \tag{6.17}$$

which is sufficient to guarantee an approach from above and a right deviation from the obstacle.

The generated trajectories yields $C_{max} = 2.12 \times 10^{-4}$.

The controller tracking errors are now analyzed for both Space CoBot vehicles. Observing Figures 6.42 and 6.43, there is small difference between both vehicles.



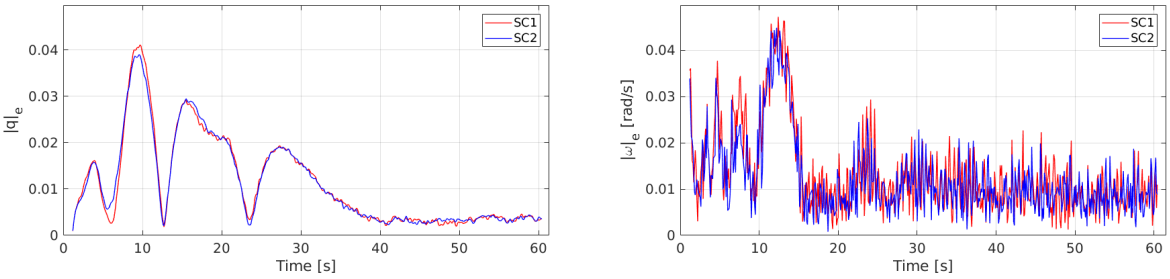
(a) Position tracking error

(b) Linear velocity tracking error

Figure 6.42: Approach in formation tracking errors $||\mathbf{p}||_e$ and $||\mathbf{v}||_e$ for both SCs.

Note that during this phase, no payload is considered, and no rigid link exists between the two vehicles. The individual controllers are operating with the single Space CoBot model. They are however tracking the CoM that results from both vehicles and the orientation of the formation as a whole. This is not strictly necessary, and simple trajectories based in the CoM of each vehicle could be sufficient. However, this does not guarantee a collision free trajectory to the payload. By making both vehicles follow a fictional point by a given distance, coincident with the resulting CoM of both vehicles, a collision is much less likely to occur as both vehicles follow unique paths.

That being said, the observed tracking errors are very similar to the tests conducted with single vehicles.

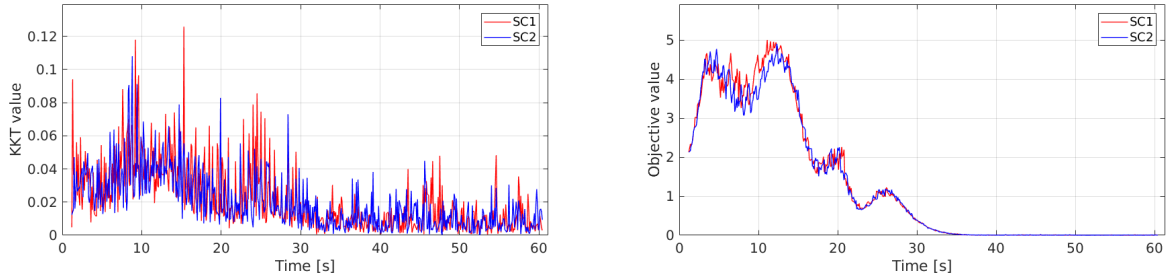


(a) Attitude tracking error

(b) Angular velocity tracking error

Figure 6.43: Approach in formation tracking errors $||\Delta\Theta||_e$ and $||\boldsymbol{\omega}||_e$ for both SCs.

Similarities with the single SC performance are also visible when observing the KKT and Objective evaluation values presented in Figure 6.44.



(a) KKT value

(b) Objective evaluation

Figure 6.44: NMPC performance with approach in formation trajectory for both SCs.

6.6.2 Transport phase

The position state bounds corresponding to the z and x axes considered in this phase are now described by

$$3 \leq p_z \leq 7 \tag{6.18}$$

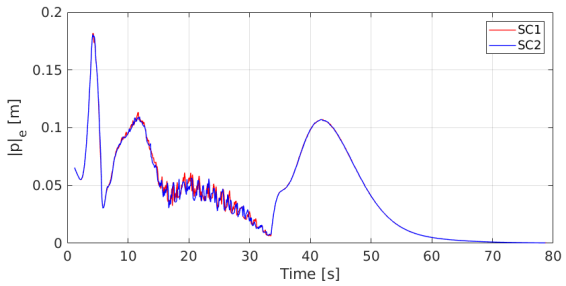
$$-4 \leq p_x \leq 0 \tag{6.19}$$

The generated trajectory yields $C_{max} = 2.23 \times 10^{-4}$.

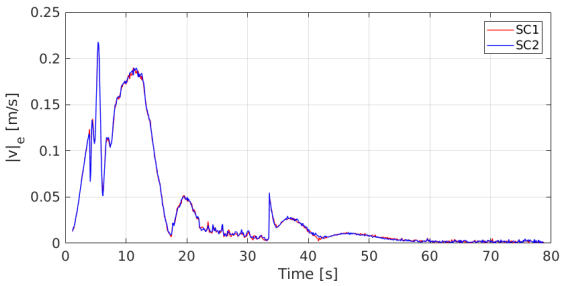
During this stage, the individual MPC controllers are now operating with the formation dynamic model (4.34). This model considers the total system as a single rigid body which implies that substantial errors in one vehicle's tracking performance will reflect substantial tracking errors on the other. It is also important to recall that the each controller comprises the computation of the actuation of both vehicles, but only uses its respective actuation input. Therefore, the cost associated with the control inputs must be further decreased while the position and attitude costs are increased. The NMPC now reflects $W_p = 500$, $W_\Theta = 500$, $W_u = 0.1$. This changes were performed once again after the poor performance of initial attempts to execute the trajectory. It is important to state that, intuitively speaking,

each vehicle expects an optimal action for the other member in the formation.

Indeed, observing Figures 6.45 and 6.46 the disparity between tracking errors of both vehicles is even smaller when compared to the approach phase.



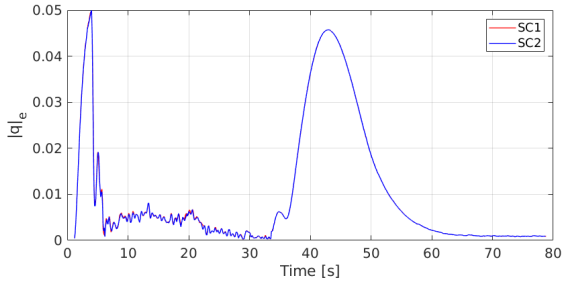
(a) Position tracking error



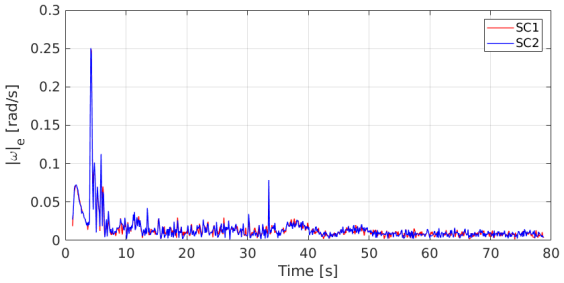
(b) Linear velocity tracking error

Figure 6.45: Formation payload transport tracking errors $||\mathbf{p}||_e$ and $||\mathbf{v}||_e$ for both SCs.

The magnitude of the tracking errors presents a slight increase when compared to previous cases. Particularly, there are two noticeable surges in the position tracking that can be attributed to acceleration and deceleration phases. This could indicate that when higher control inputs are required, the controller solutions become worse.



(a) Attitude tracking error



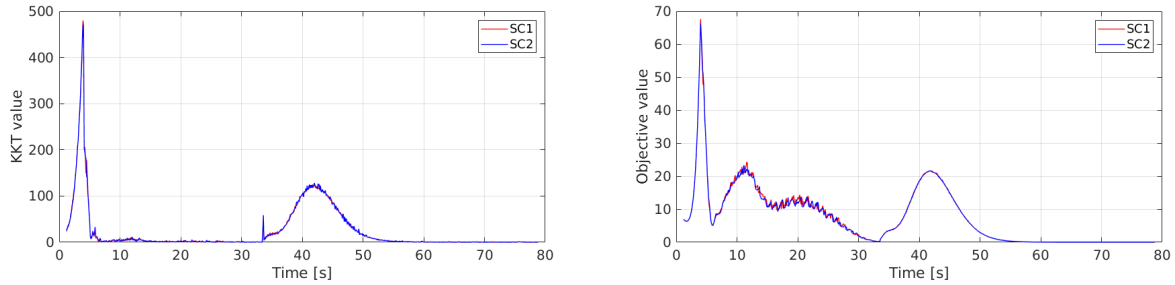
(b) Angular velocity tracking error

Figure 6.46: Formation payload transport tracking errors $||\Delta\Theta||_e$ and $||\boldsymbol{\omega}||_e$ for both SCs.

Observing the controller performance in Figure 6.47, a big spike in both the KKT and Objective evaluation values are visible during the same time intervals corresponding to the mentioned phases. The significantly high increase in KKT values indicates that the control inputs applied to the system as feedback were far from optimal, which consequently increases the Objective evaluation. This can be caused by an initial poor response to the errors registered by the controller and the consequent attempt to return to the desired trajectory, by applying increasingly aggressive corrections to the control input. This can be observed in both vehicles in Figure 6.48. Control inputs are significantly higher than implicit bounds. It is important to note that even though two vehicles are used to move a payload with similar mass to the single vehicle case, in the formation configuration, vehicles are further away from the total system's CoM, and maintaining a stable attitude while moving becomes a costly operation, requiring significantly higher control inputs.

The corrective reaction can also be verified by the return of KKT values to significantly low values after the first surge, even though considerable position tracking error is still present. This implies that

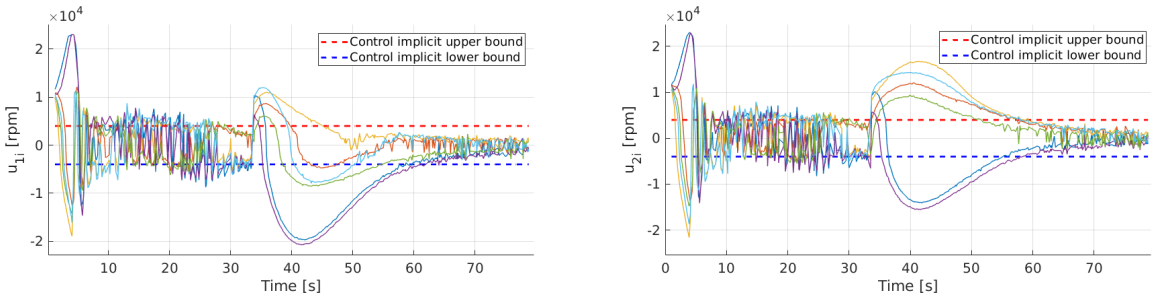
after the first surge, the controller is predicting convergence again. Note that the prediction horizon, N, was reduced from 60 to 20 for performance sake. This means that if the controller loses tracking performance too quickly, and, consequently, increases the gap to the reference, an overshoot situation might occur, as already observed in Figure 6.4. Higher N values might result in better predictions when this kind of tracking issues happen.



(a) KKT value

(b) Objective evaluation

Figure 6.47: Controller performance with generated trajectory.



(a) Space Cobot 1 control inputs

(b) Space Cobot 2 control inputs

Figure 6.48: Single obstacle formation with payload trajectory execution control inputs.

The considered model in this configuration is significantly more complex. This in turn increases fd values as visible in Figure 6.49.

The NMPC controller operates with $\bar{fd} = 12.5\,ms$ compared to $\bar{fd} = 1.8\,ms$ for the single SC model. This means that, at any given instant, not only the controller applies the optimal control input $12.5\,ms$ too late, but also maintains the previous control input, creating a double source of error. This means that increasing N is not an option in this particular case, given that it would increase fd even more. On the contrary, reducing N might help in achieving faster responses. Further reducing N is also not desirable as operating a NMPC controller with a small N defeats the purpose of using the controller at all.

Reiterating the problem at hand, high accelerations and decelerations phases will result in poor tracking once the controller operates with considerable delay. This ought to be a problem in complex maneuvers that require constant and quick changes to position and attitude. Such maneuvers don't take part in the applications of this vehicle. Regardless, it is important to note that explicit bounds need to be applied at the controller level to prevent such high inputs. This will cause the tracking error to lower much slower but converging nonetheless. Reducing linear velocity bounds in the trajectory optimization will also help by generating slower trajectories, where fd can be mitigated, which would be ideal for high

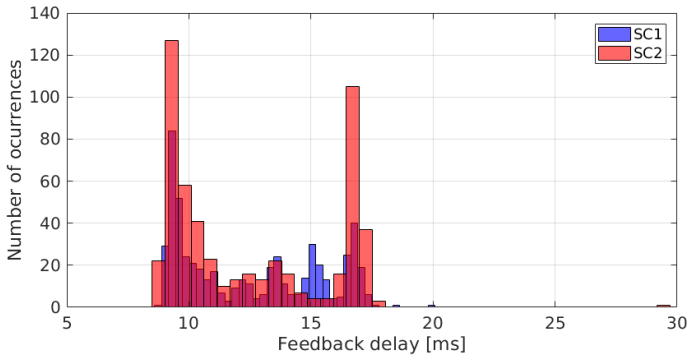


Figure 6.49: fd for single obstacle formation with payload trajectory.

mass systems.

Lastly, a 3D visualization of the complete trajectory, comprising both segments of the problem, is presented in Figure 6.50 for visual reference. The execution of the the approach phase is smooth similar to previous experiments with the single Space CoBot. The execution of the transport phase also presents a reasonable smooth execution with a small tracking error in the beginning of the trajectory.

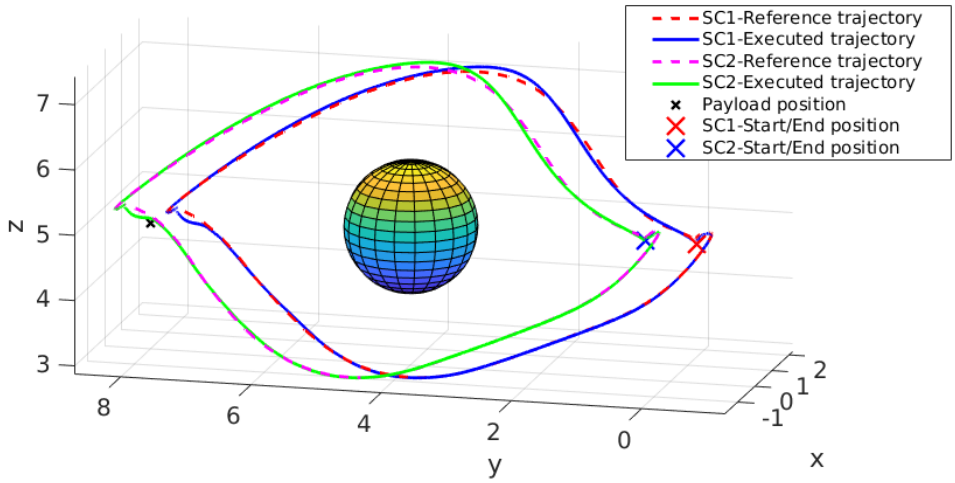


Figure 6.50: 3D visualization of full formation trajectory execution.

Chapter 7

Conclusions

This work presents a study regarding the performance of a NMPC controller for the Space CoBot free-flyer and its possible applications. This was achieved by testing the limits of the real-time formulation to the fullest while trying to perform Control and Guidance tasks in a single formulation, which proved to be unreliable.

The biggest limitation remains the computation time required to obtain control solutions. In order to speed the computation, a significant portion of the prediction horizon of the NMPC is sacrificed. This means that the controller requires another stage that can plan ahead of this horizon. To achieve this, a trajectory optimization method was integrated, which allowed better and more complete solutions. However, this method also presents a severe delay in computation and had to be considered as an offline step.

The tests performed throughout this work showing the performance of the NMPC controller demonstrate that often errors introduced in a given state will propagate to the entire state vector, as a result of optimizing all states simultaneously. These experiments also define the NMPC controller apparent limitations. Hence, under less complex scenarios the controller should also present a good performance.

7.1 Achievements

7.1.1 Control and guidance

A good exploration of the NMPC limits was conducted to obtain a controller capable of operating in a real-time frame very close to the microsecond range.

Limitations of the NMPC controller to guidance capabilities were found when testing waypoints navigation. A separate scheme was then considered with the NMPC controller responsible for Control tasks only.

Extensive experimentation was performed to obtain good configurations for different scenarios and different missions, such as weighting costs values. This allows a more detailed model parameterization for each considered configuration. This information can be used to formulate a single model that can be interacted with through a few parameters in order to better adjust to different situations. This means that

the same controller can operate with different models without recurring to weighting costs tuning or re-compiling any code. Possible so far is the operation of the Space CoBot in three different configurations, single, payload carrying or formation payload carrying, where switching between each one can be done effortlessly through few parameters.

Integrated a trajectory optimization method to take on the Guidance task, that generates feasible trajectories, even in a highly constrained environment.

7.1.2 Simulation

From this work results a simulation environment able to represent the Space CoBot with reasonable accuracy and with means to interact with elements within the simulation, such as object grasping. This can serve as a base simulation environment for future work.

7.2 Future work

The present work can benefit from several further developments to its parts. Experimentation with different error functions could yield interesting results, for example a function that makes the attitude tracking error unbounded. Better approximations for the model parameters such as K_1 and K_2 can be achieved. Different control architectures, such as a faster NMPC control within a much slower NMPC for trajectory generation as in [42], to allow an increase of the prediction horizon and more importantly, the elimination of the dependence on offline trajectory generation.

Obstacle identification methods would significantly improve the functionalities of the NMPC. For static obstacles and general mapping integrating Octomaps [67] or a similar package would provide means of identifying bounds and obstacles to constrain the trajectory optimization generation. Other vision methods can also be studied to identify additional obstacles that may not be stationary to increase navigation awareness.

Integrating other developing projects such as a robotic arm or similar to provide means of interaction with the surrounding environment would allow more realistic simulation environments.

It would be interesting to perform a study on the energy spent by this NMPC controller when compared to other control methods, as a NMPC might provide close to optimal control where energy usage is minimized, but requires significantly higher computation power.

Bibliography

[1] H. Chao, Y. Cao, and Y. Chen. Autopilots for small unmanned aerial vehicles: A survey. *International Journal of Control, Automation and Systems*, 8(1):36–44, 2010. ISSN 15986446. doi: 10.1007/s12555-010-0105-z.

[2] S. Mohan, A. Saenz-Otero, S. Nolet, D. W. Miller, and S. Sell. SPHERES flight operations testing and execution. *Acta Astronautica*, 65(7-8):1121–1132, 2009. ISSN 00945765. doi: 10.1016/j.actaastro.2009.03.039.

[3] M. Bualat, J. Barlow, T. Fong, C. Provencher, T. Smith, and A. Zuniga. Astrobee: Developing a free-flying robot for the international space station. *AIAA SPACE 2015 Conference and Exposition*, pages 1–10, 2015. doi: 10.2514/6.2015-4643.

[4] P. Roque and R. Ventura. Space CoBot: Modular design of an holonomic aerial robot for indoor microgravity environments. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem:4383–4390, 2016. ISSN 21530866. doi: 10.1109/IROS.2016.7759645.

[5] P. Roque and R. Ventura. A space CoBot for personal assistance in space stations. *IJCAI Workshop on Autonomous Mobile Service Robots*, 2016.

[6] R. Rembala and C. Ower. Robotic assembly and maintenance of future space stations based on the ISS mission operations experience. *Acta Astronautica*, 65(7-8):912–920, 2009. ISSN 00945765. doi: 10.1016/j.actaastro.2009.03.064.

[7] S. Sachdev, B. Marcotte, and G. Gibbs. Canada and the international space station program: Overview and status. *International Astronautical Federation - 55th International Astronautical Congress 2004*, 11(1):7405–7415, 2004. doi: 10.2514/6.iac-03-t.1.04.

[8] K. A. Caldas and V. Grassi. Eco-cruise NMPC control for autonomous vehicles. *2019 19th International Conference on Advanced Robotics, ICAR 2019*, pages 356–361, 2019. doi: 10.1109/ICAR46387.2019.8981639.

[9] E. S. Agency. Space Engineering – Control Engineering. *European Cooperation for Space Standardization: Space Engineering – Control Engineering*. 2004.

[10] L. Flückiger, K. Browne, B. Coltin, J. Fusco, T. Morse, and A. Symington. Astrobee Robot Software:

Enabling Mobile Autonomy on the ISS. *Int. Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, 2018. URL https://github.com/nasa/astrobee.

[11] S. Mitani, M. Goto, R. Konomura, Y. Shoji, K. Hagiwara, S. Shigeto, and N. Tanishima. Int-Ball: Crew-Supportive Autonomous Mobile Camera Robot on ISS/JEM. *IEEE Aerospace Conference Proceedings*, 2019-March:1–15, 2019. ISSN 1095323X. doi: 10.1109/AERO.2019.8741689.

[12] DLR. "CIMON - the intelligent astronaut assistant", 2018. URL https://www.dlr.de/content/en/articles/news/2018/1/20180302{_}cimon-the-intelligent-astronaut-assistant{_}26307.html.

[13] G. Szafranski and R. Czyba. Approaches of PID Control UAV. *Proceedings of the International Micro Air Vehicles conference*, pages 70–75, 2011. doi: 10.4233/uuid:3517822b-0687-48bb-82a8-748191b97531.

[14] P. Foehn and D. Scaramuzza. Onboard State Dependent LQR for Agile Quadrotors. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 6566–6572, 2018. ISSN 10504729. doi: 10.1109/ICRA.2018.8460885.

[15] J. Shah, M. Okasha, and W. Faris. Gain scheduled integral linear quadratic control for quadcopter. *International Journal of Engineering and Technology(UAE)*, 7(4):81–85, 2018. ISSN 2227524X. doi: 10.14419/ijet.v7i4.13.21334.

[16] E. Tal and S. Karaman. Accurate Tracking of Aggressive Quadrotor Trajectories Using Incremental Nonlinear Dynamic Inversion and Differential Flatness. *Proceedings of the IEEE Conference on Decision and Control*, 2019. ISSN 07431546. doi: 10.1109/CDC.2018.8619621.

[17] N. Xuan-Mung and S. K. Hong. Robust adaptive formation control of quadcopters based on a leader–follower approach. *International Journal of Advanced Robotic Systems*, 16(4):1–11, 2019. ISSN 17298814. doi: 10.1177/1729881419862733.

[18] L. Zhou, J. Zhang, H. She, and H. Jin. Quadrotor uav flight control via a novel saturation integral backstepping controller. *Automatika*, 60(2):193–206, 2019. ISSN 00051144. doi: 10.1080/00051144.2019.1610838.

[19] Anon. Pep Computer Control System. *IEEE Trans Nucl Sci*, NS-26(3 pt 1):3268–3271, 1979. ISSN 00189499.

[20] K. P. Tee, S. S. Ge, and E. H. Tay. Barrier Lyapunov Functions for the control of output-constrained nonlinear systems. *Automatica*, 45(4):918–927, 2009. ISSN 00051098. doi: 10.1016/j.automatica.2008.11.017.

[21] E. Garone, S. D. Cairano, and I. Kolmanovsky. Automatica Reference and command governors for systems with constraints : A survey on theory and applications. *Automatica*, 75:306–328, 2017. ISSN 0005-1098. doi: 10.1016/j.automatica.2016.08.013.

[22] A. Broad, I. Abraham, T. Murphey, and B. Argall. Structured Neural Network Dynamics for Model-based Control. *arXiv preprint arXiv:1808.01184*, 2018.

[23] V. Gavrilets, E. Frazzoli, B. Mettler, M. Piedmonte, and E. Feron. Aggressive maneuvering of small autonomous helicopters: A human-centered approach. *International Journal of Robotics Research*, 20(10):795–807, 2001. ISSN 02783649. doi: 10.1177/02783640122068100.

[24] W. A. Poe and S. Mokhatab. *Modeling, Control, and Optimization of Natural Gas Processing Plants*. 2016. ISBN 9780128029619. doi: 10.1016/c2014-0-03765-3.

[25] Z. Ma, O. Ma, and B. N. Shashikanth. Optimal control for spacecraft to rendezvous with a tumbling satellite in a close range. *IEEE International Conference on Intelligent Robots and Systems*, pages 4109–4114, 2006. doi: 10.1109/IROS.2006.281877.

[26] V. L. Coverstone-Carroll and N. M. Wilkey. Optimal control of a satellite-robot system using direct collocation with non-linear programming. *Acta Astronautica*, 36(3):149–162, 1995. ISSN 00945765. doi: 10.1016/0094-5765(95)00096-I.

[27] A. Bemporad, C. A. Pascucci, and C. Rocchi. *Hierarchical and hybrid model predictive control of quadcopter air vehicles*, volume 3. IFAC, 2009. ISBN 9783902661593. doi: 10.3182/20090916-3-es-3003.00004.

[28] A. Bemporad and C. Rocchi. Decentralized hybrid model predictive control of a formation of unmanned aerial vehicles. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 44(1 PART 1):11900–11906, 2011. ISSN 14746670. doi: 10.3182/20110828-6-IT-1002.00942.

[29] A. Marquez, C. Gomez, P. Deossa, and J. Espinosa. Infinite Horizon MPC and model reduction applied to large scale chemical plant. *2011 IEEE 9th Latin American Robotics Symposium and IEEE Colombian Conference on Automatic Control, LARC 2011 - Conference Proceedings*, (65), 2011. doi: 10.1109/LARC.2011.6086842.

[30] J. B. Lee, E. Dassau, R. Gondhalekar, D. E. Seborg, J. E. Pinsker, and F. J. Doyle. Enhanced model predictive control (eMPC) strategy for automated glucose control. *Industrial and Engineering Chemistry Research*, 55(46):11857–11868, 2016. ISSN 15205045. doi: 10.1021/acs.iecr.6b02718.

[31] G. Ganga and M. M. Dharmana. MPC controller for trajectory tracking control of quadcopter. *Proceedings of IEEE International Conference on Circuit, Power and Computing Technologies, ICCPCT 2017*, 2017. doi: 10.1109/ICCPCT.2017.8074380.

[32] L. Grüne and J. Pannek. *Nonlinear Model Predictive Control: Theory and Algorithms*. 2011. ISBN 978-0-85729-500-2. doi: 10.1007/978-0-85729-501-9.

[33] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl. Recent Advances in Quadratic Programming Algorithms for Nonlinear Model Predictive Control. *Vietnam Journal of Mathematics*, 46(4):863–882, 2018. ISSN 23052228. doi: 10.1007/s10013-018-0311-1.

[34] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. ISSN 00361445. doi: 10.1137/S0036144504446096.

[35] W. Andreas and Lorenz T. Biegler. *Catalogue of an exhibition of the works of Dante Alighieri March to October 1909*, volume 57. 2006. ISBN 1010700405. doi: 10.1007/s10107-004-0559-y.

[36] D. Pardo, L. Moller, M. Neunert, A. W. Winkler, and J. Buchli. Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning. *IEEE Robotics and Automation Letters*, 1(2):946–953, 2016. ISSN 23773766. doi: 10.1109/LRA.2016.2527062.

[37] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer. Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations. *Journal of Process Control*, 12(4):577–585, 2002. ISSN 09591524. doi: 10.1016/S0959-1524(01)00023-3.

[38] T. Ohtsuka. A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, 40(4):563–574, 2004. ISSN 00051098. doi: 10.1016/j.automatica.2003.11.005.

[39] D. H. Shim, H. Chung, H. J. Kim, and S. Sastry. Autonomous exploration in unknown urban environments for unmanned aerial vehicles. *Collection of Technical Papers - AIAA Guidance, Navigation, and Control Conference*, 8:6381–6388, 2005.

[40] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart. Fast nonlinear model predictive control for multicopter attitude tracking on SO(3). *2015 IEEE Conference on Control and Applications, CCA 2015 - Proceedings*, (3):1160–1166, 2015. doi: 10.1109/CCA.2015.7320769.

[41] S. Gros, R. Quirynen, and M. Diehl. Aircraft control based on fast non-linear MPC & multiple-shooting. *Proceedings of the IEEE Conference on Decision and Control*, 2012. ISSN 01912216. doi: 10.1109/CDC.2012.6426439.

[42] P. Lin, S. Chen, and C. Liu. Model predictive control-based trajectory planning for quadrotors with state and input constraints. *International Conference on Control, Automation and Systems*, 2016. ISSN 15987833. doi: 10.1109/ICCAS.2016.7832517.

[43] D. Bicego, J. Mazzetto, R. Carli, M. Farina, and A. Franchi. Nonlinear Model Predictive Control with Actuator Constraints for Multi-Rotor Aerial Vehicles. *arXiv preprint arXiv:1911.08183*, 2019.

[44] M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017. ISSN 00361445. doi: 10.1137/16M1062569.

[45] C. R. Hargraves and S. W. Paris. Direct trajectory optimization using nonlinear programming and collocation. *Astrodynamics Conference, 1986*, (3), 1986. doi: 10.2514/6.1986-2000.

[46] T. A. Howell, B. E. Jackson, and Z. Manchester. ALTRO: A Fast Solver for Constrained Trajectory Optimization. *IEEE International Conference on Intelligent Robots and Systems*, pages 7674–7679, 2019. ISSN 21530866. doi: 10.1109/IROS40897.2019.8967788.

[47] D. Q. Mayne. Differential Dynamic Programming–A Unified Approach to the Optimization of Dynamic Systems. *Control and Dynamic Systems*, 10(C):179–254, 1973. ISSN 00905267. doi: 10.1016/B978-0-12-012710-8.50010-8.

[48] S. Li, T. Liu, C. Zhang, D. Y. Yeung, and S. Shen. Learning unmanned aerial vehicle control for autonomous target following. *IJCAI International Joint Conference on Artificial Intelligence*, 2018-July:4936–4942, 2018. ISSN 10450823. doi: 10.24963/ijcai.2018/685.

[49] R. Chai, A. Savvaris, A. Tsourdos, S. Chai, and Y. Xia. Trajectory Optimization of Space Maneuver Vehicle Using a Hybrid Optimal Control Solver. *IEEE Transactions on Cybernetics*, 49(2):467–480, 2019. ISSN 21682267. doi: 10.1109/TCYB.2017.2778195Y.

[50] A. Koubaa. *Operating, Robot System (ROS) The Complete Reference (Volume 1)*, volume 1. 2015. ISBN 978-3-319-26052-5. doi: 10.1007/978-3-319-26054-9.

[51] M. Diehl and K. Mombaur. *Fast motions in biomechanics and robotics: optimization and feedback control*. 2006. ISBN 978-3-540-36118-3. doi: 10.1002/oca.4660060112.

[52] J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21(2):193–207, 1998. ISSN 15333884. doi: 10.2514/2.4231.

[53] B. W. McCormick. *Aerodynamics, Aeronautics and Flight Mechanics*. 2nd edition, 1995. ISBN 9780471575061.

[54] R. W. Deters, G. K. Ananda, and M. S. Selig. Reynolds number effects on the performance of small-scale propellers. *32nd AIAA Applied Aerodynamics Conference*, (June):1–43, 2014. doi: 10.2514/6.2014-2151.

[55] C. G. Atkeson, C. H. An, and J. M. Hollerbach. Estimation of Inertial Parameters of Manipulator Loads and Links Abstract. *The International Journal of Robotics Research*, pages 101–119, 1986.

[56] M. Ekal and R. Ventura. On the Accuracy of Inertial Parameter Estimation of a Free-Flying Robot While Grasping an Object. *Journal of Intelligent and Robotic Systems: Theory and Applications*, 2019. ISSN 15730409. doi: 10.1007/s10846-019-01040-y.

[57] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl. From linear to nonlinear MPC: bridging the gap via the real-time iteration. *International Journal of Control*, (October):1–19, 2016. ISSN 13665820. doi: 10.1080/00207179.2016.1222553.

[58] M. Diehl, H. G. Bock, and J. P. Schlöder. A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, 43(5):1714–1736, 2005. ISSN 03630129. doi: 10.1137/S0363012902400713.

[59] P. Martin and E. Salaün. The true role of accelerometer feedback in quadrotor control. *Proceedings - IEEE International Conference on Robotics and Automation*, (June 2010):1623–1629, 2010. ISSN 10504729. doi: 10.1109/ROBOT.2010.5509980.

[60] B. Houska, H. J. Ferreau, and M. Diehl. Adjoints estimation methods for impulsive Moon-to-Earth trajectories in the restricted three-body problem. *Optimal Control Applications and Methods*, 32(3): 298–312, 2011. ISSN 01432087. doi: 10.1002/oca.

[61] H. J. Ferreau. *An Online Active Set Strategy for Fast Solution of Parametric Quadratic Programs with Applications to Predictive Engine Control*. PhD thesis, Heidelberg University, 2006.

[62] B. Houska and H. Joachim. ACADO Toolkit User ' s Manual, 2011.

[63] A. B. Younes, D. Mortari Prof., J. D. Turner, and J. L. Junkins Prof. Attitude error kinematics. *Journal of Guidance, Control, and Dynamics*, 37(1):330–335, 2014. ISSN 15333884. doi: 10.2514/1.60928.

[64] M. Diehl, H. J. Ferreau, and N. Haverbeke. Efficient numerical methods for nonlinear MPC and moving horizon estimation. *Lecture Notes in Control and Information Sciences*, 384:391–417, 2009. ISSN 01708643. doi: 10.1007/978-3-642-01094-1_32.

[65] T. Koolen and R. Deits. Julia for robotics: simulation and real-time control in a high-level programming language. (May):604–611, 2019. ISSN 10504729. doi: 10.1109/icra.2019.8793875.

[66] A. Erfani, A. Rajabi-Ghahnaviyeh, and M. Boroushaki. Design and construction of a non-linear model predictive controller for building's cooling system. *Building and Environment*, 133(November 2017):237–245, 2018. ISSN 03601323. doi: 10.1016/j.buildenv.2018.02.022.

[67] S. Vanneste, B. Bellekens, and M. Weyn. 3DVFH+: Real-time three-dimensional obstacle avoidance using an octomap. *CEUR Workshop Proceedings*, 1319:91–102, 2014. ISSN 16130073.

[68] N. Trawny and S. I. Roumeliotis. Indirect Kalman Filter for 3D Attitude Estimation A Tutorial for Quaternion Algebra Multiple Autonomous. Technical Report 612, University of Minnesota, 2005.

[69] T. Lee. Exponential stability of an attitude tracking control system on SO(3) for large-angle rotational maneuvers. *Systems and Control Letters*, 61(1):231–237, 2012. ISSN 01676911. doi: 10.1016/j.sysconle.2011.10.017.

Appendix A

Quaternions

In this Appendix, some operations involving quaternions used throughout this thesis are detailed following [68]. Additionally, a comparison of methods to obtain attitude errors resorting to quaternions is presented.

A.1 Quaternion operations

The quaternion considered in this work follows the notation $\mathbf{q} = [q_w, q_x, q_y, q_z]$, where q_x, q_y and q_z are the components of the rotation axis and q_w is a scalar corresponding to the magnitude of the rotation. In this appendix a quaternion will be represented formally as $q_w + q_x i + q_y j + q_z k$, where q_w, q_z, q_y and q_z are real numbers and the symbols i, j and k satisfy the following identities:

$$i^2 = j^2 = k^2 = -1 \tag{A.1}$$

$$ij = k,\ ji = -k \tag{A.2}$$

$$jk = i,\ kj = -i \tag{A.3}$$

$$ki = j,\ ik = -j \tag{A.4}$$

$$\tag{A.5}$$

A.1.1 Multiplication between vector and quaternion

$$\boldsymbol{\omega} \otimes \mathbf{q} = \begin{bmatrix} \boldsymbol{\omega} \\ 0 \end{bmatrix} \otimes \mathbf{q} \tag{A.6}$$

$$= \begin{bmatrix} -[\boldsymbol{\omega}\times] & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix} \mathbf{q} \tag{A.7}$$

$$= \boldsymbol{\Omega}(\boldsymbol{\omega})\mathbf{q} \tag{A.8}$$

$$= \mathbf{Q}(\mathbf{q})\boldsymbol{\omega} \tag{A.9}$$

where the skew-symetric matrix operator $[\boldsymbol{\omega}\times]$ for the vector $\boldsymbol{\omega}$ is defined as:

$$[\boldsymbol{\omega}\times] = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{A.10}$$

and matrices $\boldsymbol{\Omega}$ and $\mathbf{Q}$ are defined as:

$$\boldsymbol{\Omega}(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}, \qquad \mathbf{Q}(\mathbf{q}) = \begin{bmatrix} q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \\ -q_x & -q_y & -q_z \end{bmatrix} \tag{A.11}$$

A.2 Attitude error approach comparison

Assuming the quaternion notation $\mathbf{q} = [q_w, q_x, q_y, q_z]$ the quaternion error $\mathbf{q}_e$, following [63], is defined by:

$$\mathbf{q}_e = \begin{bmatrix} q_w^{ref} q_x + q_z^{ref} q_y - q_y^{ref} q_z - q_x^{ref} q_w \\ -q_z^{ref} q_x + q_w^{ref} q_y + q_x^{ref} q_z - q_y^{ref} q_w \\ q_y^{ref} q_x - q_x^{ref} q_y + q_w^{ref} q_z - q_z^{ref} q_w \\ q_x^{ref} q_x - q_y^{ref} q_y + q_z^{ref} q_z - q_w^{ref} q_w \end{bmatrix} \tag{A.12}$$

where $\mathbf{q}$ represent the current attitude and $\mathbf{q}^{ref}$ corresponds to the desired attitude. The resulting quaternion error can then be viewed as $\mathbf{q}_e = [e_1\ e_2\ e_3\ e_4]$ where $e_4 = \pm 1$. For this reason, a truncated version of the quaternion error will be used and henceforth be considered to describe attitude error vector designated by$\Delta\boldsymbol{\Theta} = [e_1\ e_2\ e_3]$ and the norm function is used to describe the attitude error magnitude $||\Delta\boldsymbol{\Theta}||_e$.

To verify the properties of the attitude error vector $\Delta\boldsymbol{\Theta}$ and attitude error magnitude $||\Delta\boldsymbol{\Theta}||_e$, a comparison is performed with the attitude error approaches in [69] defined as:

The magnitude of the attitude error Ψ is defined by

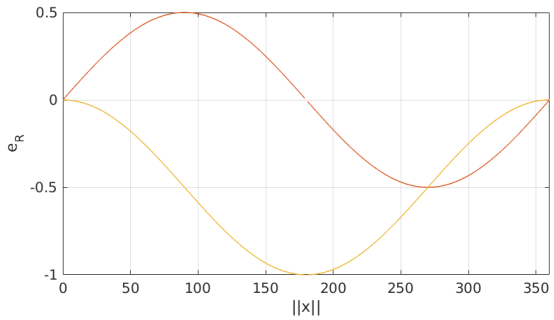
$$\Psi(R, R_{ref}) = 2 - \sqrt{1 + tr[R_{ref}^T R]} \tag{A.13}$$

and the attitude error vector e_R is defined by

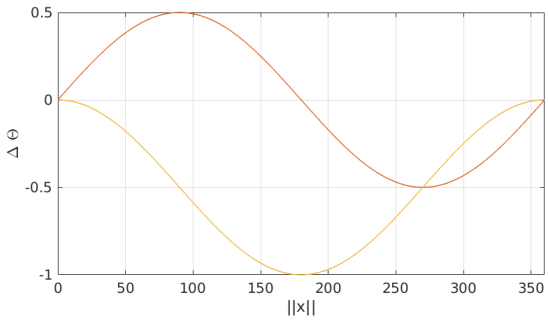
$$e_R(R, R_{ref}) = \frac{1}{2\sqrt{1 + tr[R_{ref}^T R]}} (R_{ref}^T R - R_{ref}^T)^\vee \tag{A.14}$$

where $\vee$ corresponds to the inverse skew operator and tr to the trace of a matrix.

The following Figures A.1 and A.2 present the properties of both approaches for an error of $||x|| \in [0, 360]$ degrees in θ and ψ.



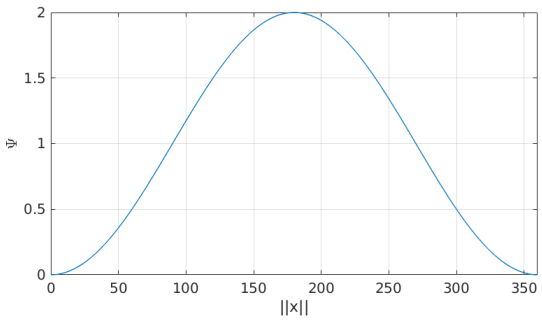
(a) Attitude error vector e_R



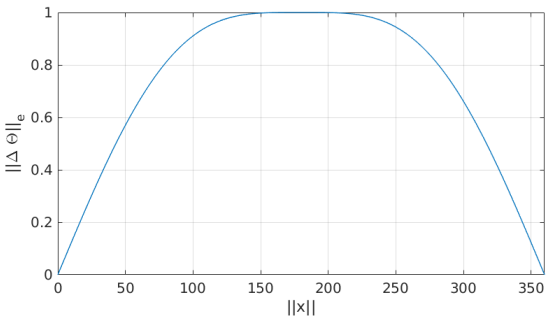
(b) Attitude error vector $\Delta\Theta$

Figure A.1: Comparison between attitude error vectors.

The attitude error vectors present practically the same properties. However, e_R presents a discontinuity at $||x|| = 180$ degrees The magnitude error functions are both bounded. The Ψ is bounded between



(a) Attitude error magnitude Ψ



(b) Attitude error magnitude $||\Delta\Theta||_e$

Figure A.2: Comparison between attitude error magnitude.

0 and 2 while $||\Delta\mathbf{\Theta}||_e$ is bounded between 0 and 1.

Both approaches present similar properties. The main advantage of the approach based in the quaternion error is the simplicity of the required computations. Therefore, this method is selected to minimize computation time within the NMPC.