POLITECNICO DI TORINO

Laurea magistrale in Ingegneria Meccanica

Politecnico
di Torino

Airsat: development of a two-dimensional free-flyer simulator

Relatore
Prof. Stefano Mauro

Candidato
Ernesto Russo

Correlatore:
Ing. Matteo Melchiorre
Ing. Laura Salamina
Dott. Davide Sorli
Dott. Martina Ferrauto

A.A. 2024/2025

Contents

List of Figures **VI**

List of Tables **VII**

List of Acronyms **VIII**

Abstract **1**

1 Introduction **2**

1.1 Objectives 2

2 State of art **4**

2.1 Spacecraft simulator systems 4

2.1.1 POSEIDYN testbed 6

2.1.2 PINOCCHIO testbed 9

2.1.3 DISCOWER ATMOS 10

3 Design of the simulator: Bill of materials and CAD representation **14**

3.1 Description of components 14

3.1.1 Metal profile 14

3.1.2 Thruster 16

3.1.3 Air bearing 17

3.1.4 Solenoid valve 18

3.1.5 Air tank 20

3.1.6 Pressure regulator 20
3.1.7 Arduino 21
3.1.8 Reaction wheel 22
3.1.9 Motor 23
3.1.10 Battery 25
3.1.11 IMU 26
3.2 CAD Design 27
3.2.1 Motor supports and tank holder 28
3.2.2 Control board support 30
3.2.3 Miscellaneous Supports 31
3.2.4 Complete CAD 32

4 Digital twin and control strategy **34**
4.1 Purpose of the model 34
4.2 Digital twin 35
4.2.1 Body 36
4.2.2 Sensors 37
4.2.3 Actuation 39
4.2.4 Control 45
4.2.5 Model results with sinusoidal trajectory 53
4.2.6 Model results with parabolic trajectory 61

5 Arduino code **64**
5.1 Arduino functions 64
5.1.1 Control logic 65
5.1.2 Data acquisition 65
5.1.3 Actuators control 66
5.2 Code structure 67

6 Design of the Control board **69**
6.1 Control board 69

6.1.1 Command block 70
6.1.2 Sensor block 70
6.1.3 Valve block 71
6.1.4 Motor block 71
6.2 Electrical circuit 73

7 Experimental results **74**
7.1 Test bench setup 74
7.2 Experimental tests 75
7.2.1 Test 1: predefined rotation 76
7.2.2 Test 2: reaction to an external disturbance 81

8 Conclusion **88**
8.1 Future works 89

A Matlab scripts **90**
A.1 Main script 90
A.2 Trajectory form 99
A.3 Angular limitation 101
A.4 w limitation 102

Bibliography **105**

List of Figures

2.1 Summary of spacecraft simulator systems 5
2.2 Line-up of the first- to fourth-generation floating spacecraft simulators used on the POSEIDYN testbed 7
2.3 Overview of the main elements of the POSEIDYN testbed 7
2.4 Schematic software architecture of the floating spacecraft simulator . . 9
2.5 PINOCCHIO platform 10
2.6 Scheme of the GNC architecture of PINOCCHIO 10
2.7 DISCOWER ATMOS 11
2.8 Avionics Plate 12

3.1 Cross-section of aluminium profiles 15
3.2 Automatic connector 15
3.3 Aluminium profiles Structure assembled 16
3.4 Thruster 17
3.5 MAGER air bearing HPC series 18
3.6 3D representation of the valve 19
3.7 Valve picture 19
3.8 Air tank 20
3.9 Pressure regulator 21
3.10 Arduino Nano ESP32 22
3.11 Reaction wheel technical drawing 23
3.12 Reaction wheel 23
3.13 Motor picture 24

3.14 Motor driver picture 25
3.15 12V 0.65Ah NiCd Battery pack 26
3.16 IMU MPU9250 27
3.17 Exploded view of motor supports and tank holder 29
3.18 Motor supports and tank holder picture 30
3.19 Control board support 31
3.20 Miscellaneous Supports 32
3.21 Airsat 3D assembly 33

4.1 Simulink Model Homepage 35
4.2 Body frame Simulink model and technical specifications 36
4.3 Reaction wheel Simulink model and technical specifications 37
4.4 Sensor block and Simulink connections 38
4.5 Sensor subsystem 39
4.6 NASA thruster equations 40
4.7 Thruster model 41
4.8 PWM subsystem 42
4.9 Entire translation motion subsystem 43
4.10 Motor driver block 44
4.11 Theorical trajectory 46
4.12 Theorical velocity profile 47
4.13 Simulink model of rotation matrix 52
4.14 Control model 52
4.15 comparison of the trajectories 53
4.16 Comparison of x velocities 54
4.17 Comparison of y velocities 54
4.18 Comparison of absolute velocities 55
4.19 Orientation comparison 56
4.20 Position and orientation error 57
4.21 Velocity error 57

4.22 motor torque output 58
4.23 motor command voltage 59
4.24 x-Force scopes 60
4.25 y-Force scopes 60
4.26 Parabolic trajectory 62
4.27 Position and orientation error of parabolic trajectory 62
4.28 Velocity error of parabolic trajectory 63

5.1 Five code states 68

6.1 Control board 70
6.2 Protection circuit of data-sheet 72
6.3 Electrical circuit 73

7.1 Test bench setup 75
7.2 Angular acceleration, velocity and position profile 77
7.3 Angular position comparison 78
7.4 Angular velocity comparison 79
7.5 Angular velocity comparison zoomed 80
7.6 Disturbance effect on angular position 82
7.7 Disturbance effect on angular velocity 83
7.8 Disturbance effect on angular position with modified script 84
7.9 Disturbance effect on angular velocity with modified script 85
7.10 Motor control torque 87

List of Tables

3.1 Air bearing performance with 4 bar relative air supply pressure 18
3.2 Parameters of motor model 24

4.1 Data of pneumatic mode 40
4.2 Sinusoidal trajectory parameters 45
4.3 Parabolic trajectory parameters 61

7.1 LQI parameters of experimental tests 76

Acronyms

ADR Active Debris Removal. 1

CAD Computer aided design. 27, 30, 32, 36

dc duty cycle. 41, 42

FSS Floating Spacecraft Simulator. 2, 6

GNC Guidance navigation and control. 8, 9

HIL Hardware in the loop. 8

IMU Inertial Measurement Unit. 12, 26, 30, 64, 65, 69, 70, 74, 78, 80, 82, 84, 85, 89

LQI Linear Quadratic Integral. 45, 47, 48, 50, 65, 66, 78, 84

LQR Linear Quadratic Regulator. 47, 48

myDAS Mini Dynamic Autonomous Spacecraft Simulator. 2

OOS On-Orbit Servicing. 1

PLA Polylactic acid. 29

PWM pulse-width modulation. 18, 39, 41, 42, 59, 64, 66, 67, 71

Abstract

Space is evolving into a dynamic environment with expanding opportunities enabled by advancing technologies. From communication satellites and Earth observation to scientific exploration and future space infrastructure, the number of missions and systems operating in orbit continues to grow. With this expansion comes the need for more advanced technologies to manage, service, and interact with objects in space, especially in complex conditions like microgravity and in the presence of uncooperative or tumbling targets. Applications such as On-Orbit Servicing (OOS), Active Debris Removal (ADR), and autonomous inspection rely heavily on robotics and precise control.

To develop and test these technologies, it's essential to have reliable simulation platforms that can reproduce, at least partially, the dynamics of space. Conducting experiments in orbit is expensive and often impractical, so ground-based simulators (particularly those that replicate microgravity conditions) are a key part of the development process.

This thesis focuses on the simulation of a robotic system designed for planar microgravity environments, with the goal of supporting research in autonomous space operations. The work includes modeling, control, and testing of a system that mimics the behavior of a free-floating satellite in space. The study covers the full development of the system, from the conception and realization of the physical robot to the modeling of its digital counterpart in MATLAB/Simulink, concluding with the design of a control strategy that ensures precise and stable performance under realistic operating conditions. As a final step, the robot was experimentally tested in a laboratory environment on a resin-coated planar surface.

Chapter 1

Introduction

At the foundation of this research lies the necessity of developing, at the Politecnico di Torino, a Floating Spacecraft Simulator (FSS) [1], conceptually similar to the Mini Dynamic Autonomous Spacecraft Simulator (myDAS) [2] platform currently in operation at the Naval Postgraduate School. The motivation behind such an initiative is twofold: on the one hand, to provide a versatile experimental facility that enables the safe testing of guidance, navigation, and control algorithms for spacecraft-like systems; on the other, to contribute to the advancement of the laboratory's capabilities in the field of space robotics and autonomous systems.

1.1 Objectives

This thesis investigates the dynamics and control of a spacecraft operating in a planar microgravity-like environment, using a testbed that approximates two-dimensional motion with minimal friction. The aim is to analyze the behavior of the system and develop suitable control strategies for trajectory tracking under conditions that resemble those encountered in space missions.

The work encompasses a comprehensive analysis of the system architecture, including a detailed description of its physical components and their integration. Particular

attention is given to the construction of a high-fidelity digital twin, which replicates the behavior of the real system and serves as a foundation for the design and preliminary testing of trajectory control strategies.

The first part of the thesis presents a brief overview of related experimental platforms used in the simulation of spacecraft dynamics. This literature review aims to contextualize the present work, highlighting the main methodologies adopted in the field and identifying the unique contributions of this study.

Chapter 2

State of art

This opening chapter is therefore devoted to a review of the state of the art, with the objective of situating the present work within its broader scientific and technological context. Particular emphasis is placed on identifying what has already been achieved in similar projects in academic environments, and on clarifying the elements of novelty that this research aims to introduce. By outlining the existing achievements and highlighting the gaps that remain, the chapter sets the stage for a deeper understanding of the scope, relevance, and potential impact of the simulator developed in this thesis.

2.1 Spacecraft simulator systems

Before delving into the specific details of the project, it is essential to examine the existing models and categories of spacecraft simulators. Such an investigation allows not only for a comprehensive understanding of the different approaches that have been developed over the years, but also for a clear identification of the conceptual framework in which the present work can be situated.

Accordingly, the following figure 2.1 [1] is presented , which illustrates the main classifications of spacecraft simulator systems. This schematic overview serves to organize the different existing approaches into well-defined categories, thereby facilitating

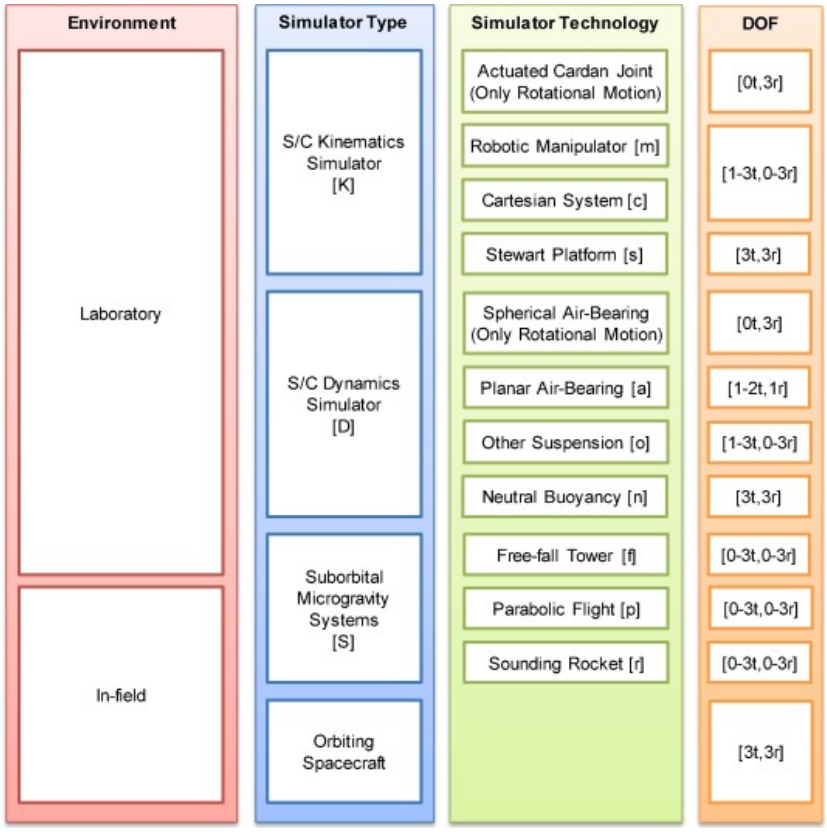


Figure 2.1: Summary of spacecraft simulator systems

the identification of the framework within which the present project can be positioned.

Through this classification, it becomes easier to place the present project within the broader landscape of spacecraft simulators. Specifically, the system under development is conceived as a laboratory-based dynamic simulator. Unlike kinematic simulators, a dynamic simulator physically generates the forces and torques acting on the spacecraft by means of real actuators - the very same type that would be employed in an actual mission [3].

In this case, the adopted simulation technology relies on a robot capable of moving in a two-dimensional plane, suspended by means of air bearings that minimize friction and emulate free-space conditions. The overall system exhibits three degrees of freedom:

two translational (along the x and y axes) and one rotational (about the out-of-plane axis).

2.1.1 POSEIDYN testbed

The primary source of inspiration for the present work is the POSEIDYN testbed [4], developed at the Naval Postgraduate School. Given its strong conceptual similarity to the system designed in this project, a concise analysis of its architecture and operating principles is of particular relevance. Accordingly, this section is devoted to an overview of the models underpinning the POSEIDYN facility, as well as a brief account of its historical development and research applications.

Since its inception, the POSEIDYN facility at the Naval Postgraduate School has undergone continuous development, leading to the realization of four successive generations of FSS, each introducing novel capabilities [4]. The FSS is designed to float on an air-bearing system, enabling planar motion with negligible friction and thus replicating the dynamical conditions of orbital free flight. As illustrated in the figure 2.4 actuation is provided by cold-gas thrusters, arranged to deliver translational manoeuvrability, while rotation is provided by a reaction wheel. Equipped with onboard sensors, such as IMUs and cameras, each FSS can operate as an independent spacecraft analogue within the Hardware-in-the-Loop framework of POSEIDYN[4]. The first generation focused on rendezvous and docking, incorporating an early prototype of a capture system[5]. The second generation expanded actuation capabilities by integrating vectorable thrusters together with a miniature control moment gyroscope, enabling more complex attitude manoeuvrers[6, 7]. With the third generation, the design philosophy shifted toward a lightweight structure, replacing the aluminium body with polycarbonate components fabricated via additive manufacturing, and introducing standardized docking interfaces. Finally, the fourth generation consolidated the use of polycarbonate structures while incorporating a dedicated standardized interface for compatibility with robotic manipulator research[8]. This evolutionary progression reflects the constant refinement of the testbed, both in terms of materials and functionality, in order to support increasingly

sophisticated experimental scenarios in proximity operations. In the following figure 2.2 [4], the successive evolutions of the robotic platforms are illustrated, while figure 2.3 [4] presents the experimental workspace in which the simulators operate.

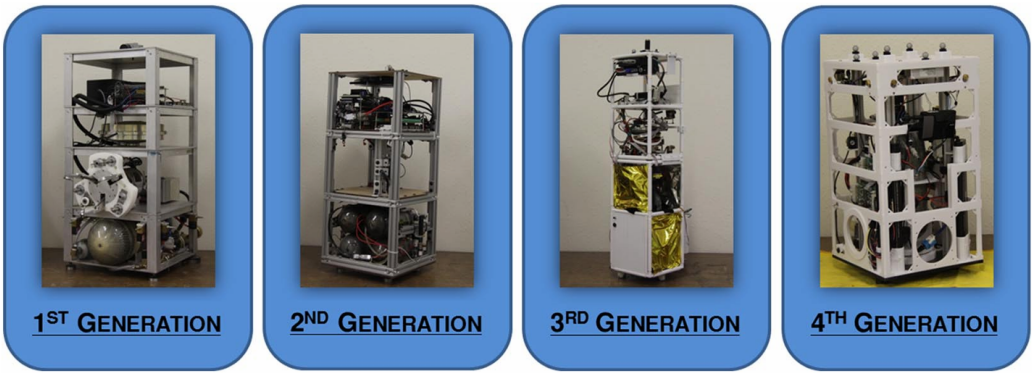


Figure 2.2: Line-up of the first- to fourth-generation floating spacecraft simulators used on the POSEIDYN testbed

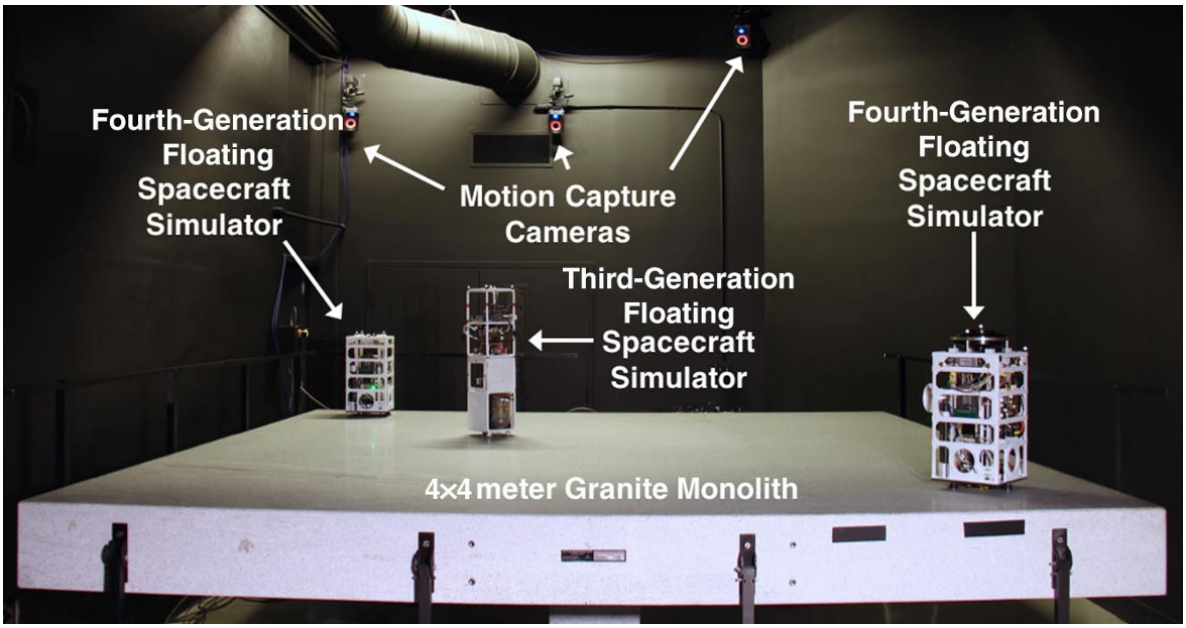


Figure 2.3: Overview of the main elements of the POSEIDYN testbed

The testbed combines the physical dynamics of the free-flying simulators with a real-time orbital dynamics simulator, implementing a Hardware in the loop (HIL) framework. In this way, Guidance navigation and control (GNC) algorithms can be tested against realistic scenarios, where the simulated space environment is continuously coupled with the actual response of the physical vehicles. This software-driven integration makes it possible to replicate proximity operations(such as rendezvous, docking, and servicing) in a safe and repeatable laboratory setting, without the risks and costs associated with on-orbit experiments. The facility therefore serves not only as a mechanical emulator of microgravity conditions, but as a versatile research environment for the validation of navigation filters, sensor fusion strategies, and autonomous control laws under conditions that closely approximate those of real space missions[4].

The software architecture of the POSEIDYN facility, illustrated in figure 2.4 [4], is structured in a modular and flexible manner, enabling seamless integration between simulated orbital dynamics, sensor data, and control algorithms. Real-time HIL execution allows the physical free-flying simulators to be directly coupled with orbital dynamics models, ensuring that both sensor readings and actuator commands reflect realistic mission scenarios. Sensor data are continuously processed and fused through dedicated filtering modules, while guidance and control outputs are translated into thruster commands in real time. This modular design makes the architecture highly scalable and reconfigurable, allowing new sensors, estimation techniques, or control laws to be incorporated without fundamental modifications to the system[4].

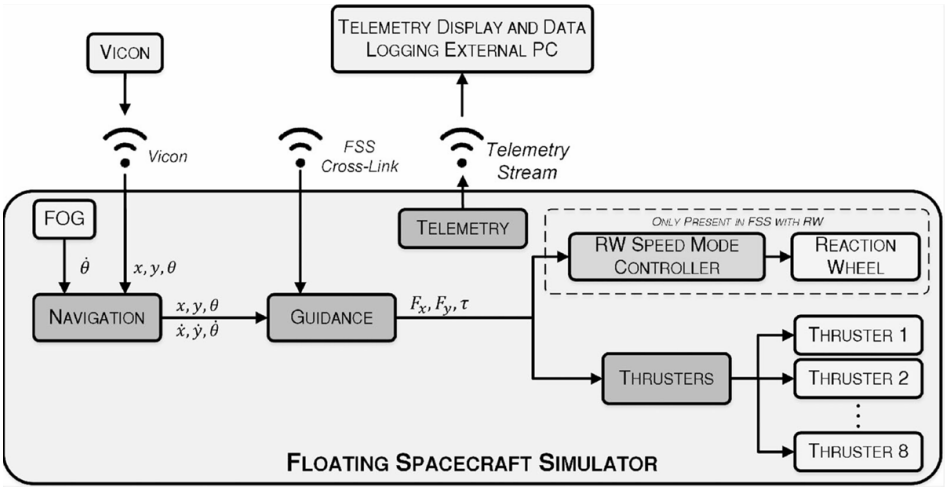


Figure 2.4: Schematic software architecture of the floating spacecraft simulator

The navigation subsystem of POSEIDYN is designed to provide real-time estimation of the vehicle's state by exploiting the complementary characteristics of different sensors. Inertial measurements offer high-frequency data suitable for short-term state propagation, but they are inherently affected by noise and drift. In contrast, vision-based observations deliver drift-free relative measurements, although at lower update rates and with greater sensitivity to environmental conditions. The core of the subsystem therefore lies in the software integration of these data streams, where filtering algorithms such as the Kalman filter are employed to fuse inertial and visual information[4].

2.1.2 PINOCCHIO testbed

A relevant contribution to ground-based experimental platforms for spacecraft GNC is represented by the PINOCCHIO project, developed at Sapienza University of Rome, whose current configuration is shown in the figure 2.5 [9]. The first stage of the project, presented in 2012[10], focused on the design and validation of a low-cost frictionless 2D testbed. The platform, levitating on air bearings and actuated by cold-gas thrusters, was conceived to reproduce a planar microgravity environment for testing guidance strategies, control laws, and navigation sensors. Particular attention was devoted to

the selection and characterization of inertial sensors in combination with an optical flow device and a Kalman filter. Experimental campaigns demonstrated the capability of the platform to perform increasingly complex manoeuvrers – from attitude acquisition to trajectory tracking and combined translational-rotational motions – highlighting both the soundness of the architecture and the limitations imposed by air supply duration. A simplified scheme of its functional architecture is depicted in the figure 2.6 [9].

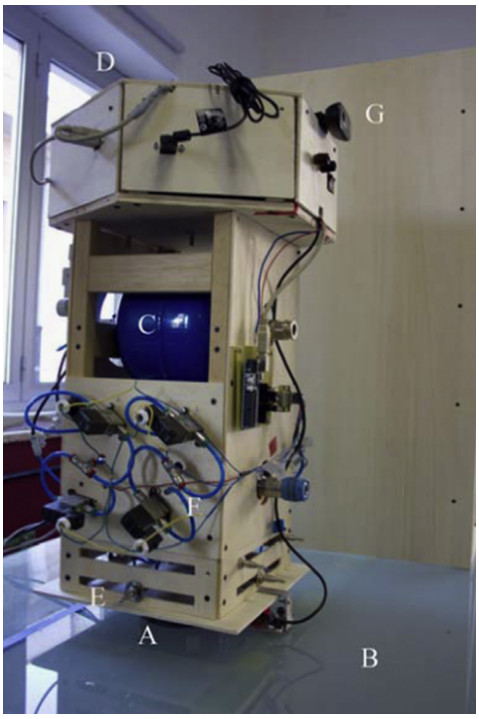


Figure 2.5: PINOCCHIO platform

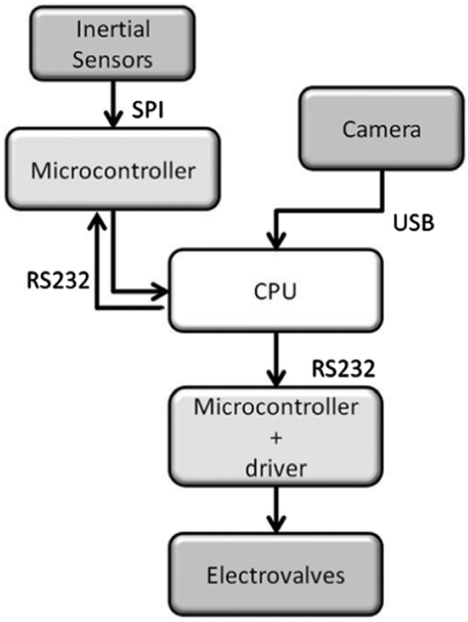


Figure 2.6: Scheme of the GNC architecture of PINOCCHIO

2.1.3 DISCOWER ATMOS

A recent and noteworthy initiative in the field of space robotics testbeds is "DISCOWER ATMOS" (Autonomy Testbed for Multi-purpose Orbiting Systems) [11], developed at the KTH Space Robotics Laboratory of Stockholm. ATMOS is conceived as an open-source, modular free-flyer platform designed to reproduce in a planar con-

figuration, the near-frictionless conditions of microgravity using an air-bearing support system. Its architecture includes a modular actuation plate compatible with solenoid thrusters and propeller-based actuators, and a payload support system that allows testing various configurations and instruments. The entire structure is shown in figure 2.7.

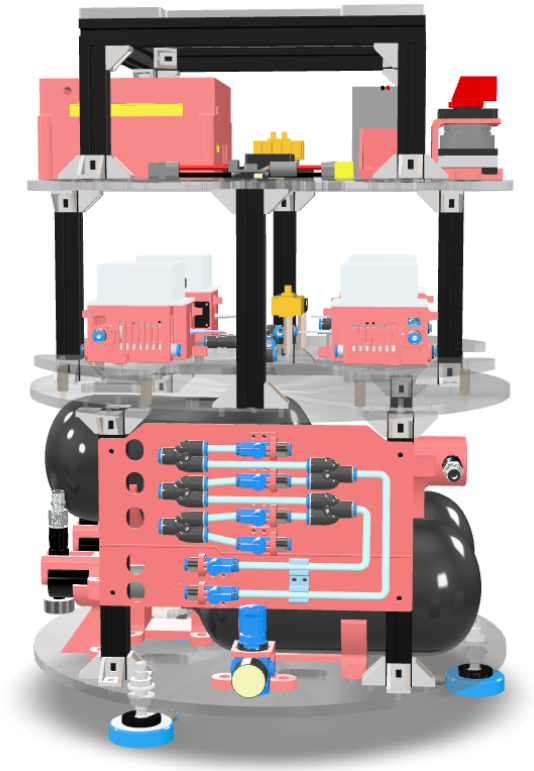


Figure 2.7: DISCOWER ATMOS

One of the most significant contributions of ATMOS lies in its goal to reduce the cost and complexity of space system validation. By providing a realistic, ground-based environment that replicates orbital dynamics, ATMOS minimizes the need for expensive in-orbit experiments and accelerates the development cycle of autonomous space systems. Researchers can perform hardware-in-the-loop simulations, software testing, and control validation directly on Earth, lowering both logistical and financial barriers

to experimentation.

Equally important is its open-source nature, which democratizes access to advanced space robotics research. The complete documentation, bill of materials, and assembly instructions are freely available online, enabling laboratories and universities worldwide to replicate and adapt the platform for their own research needs. This open-access approach fosters collaboration, transparency, and reproducibility, while also promoting the standardization of experimental methodologies within the space robotics community.

The core of the hardware architecture of ATMOS is the control platform, called "Avionics Plate", showed in the following figure, containing controllers commonly used in autonomous systems like the Pixhawk and an NVIDIA Jetson integrated processing card.

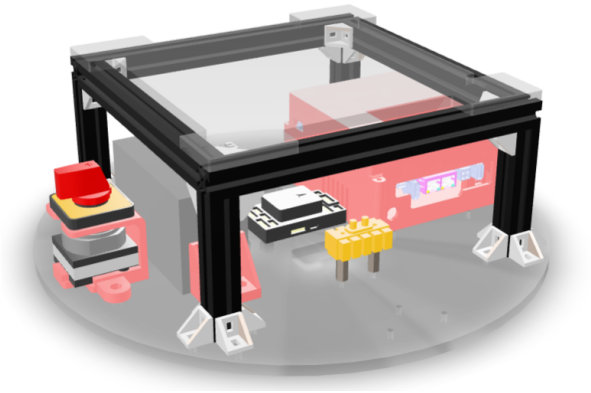


Figure 2.8: Avionics Plate

Within the context of the DISCOWER project, the NVIDIA Jetson Orin NX serves as a high-performance edge AI computing module responsible for executing advanced perception and data processing tasks. It enables real-time processing of sensor inputs such as camera feeds and supports the implementation of artificial intelligence algorithms needed for autonomous decision-making and environmental understanding. Complementarily, the Pixhawk functions as the flight controller and low-level sensor fusion unit, managing real-time control of the free-flyer platform by integrating inertial measurements from the onboard Inertial Measurement Unit (IMU) and other sensors.

It executes control commands and stabilizes the robot's attitude and motion, interfacing with propulsion systems to achieve precise manoeuvring. The integration of these two systems allows DISCOWER's ATMOS free-flyer to combine robust autonomous control with sophisticated onboard intelligence.

Chapter 3

Design of the simulator: Bill of materials and CAD representation

This chapter is devoted to the presentation of its main components, with the aim of providing a comprehensive understanding of the elements that constitute the system and of facilitating the description of the construction process and its various stages.

3.1 Description of components

3.1.1 Metal profile

The constructed metal framework closely resembles the second generation model of the POSEIDYN test bench[4]. Aluminium profiles manufactured by *Item* with a 30x24mm cross-section are employed [12], as illustrated in the figure 3.1.

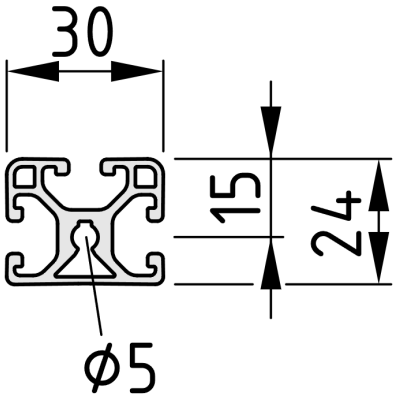


Figure 3.1: Cross-section of aluminium profiles

The connection between the various profiles is achieved using item automatic-fastening set [13], as illustrated in the figure 3.2.

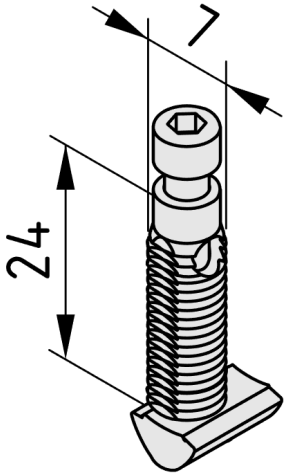


Figure 3.2: Automatic connector

These connectors are fastened into the profile grooves and do not require any additional machining. The structure consists of four $606mm$ profiles, which define the height of the robot, and an additional thirteen $300mm$ profiles, which are used both

to complete the frame and to support various component mounts. The final assembled structure is shown in the following figure 3.3.

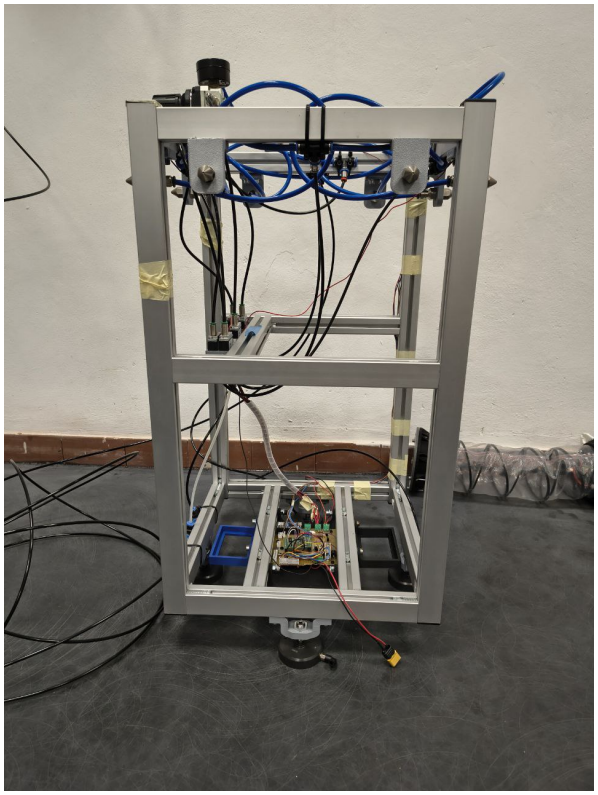


Figure 3.3: Aluminium profiles Structure assembled

3.1.2 Thruster

The thrusters installed on the robot, which are responsible for generating the translational motion required to follow the desired trajectory, consist of simple convergent nozzles with a critical diameter of $1mm$. On the rear side, they are equipped with a quick coupling for the $\varnothing 6mm$ tubing, as illustrated in Figure 3.4 below. Each thruster is composed of two separate parts that can be screwed together

Figure 3.4: Thruster

3.1.3 Air bearing

The air bearings used in the project, which serve to simulate a microgravity environment by allowing nearly frictionless planar motion, are manufactured by MAGER and correspond to the circular models of the HPC series (Figure 3.5). They feature a diameter of $\varnothing 60mm$ and a height of $18mm$.

Figure 3.5: MAGER air bearing HPC series

The performance of the bearing, evaluated at a reference pressure of $4bar$ (relative), is summarized in the table below 3.1 [14].

performances 10µm air gap h			
Load [N]	Stiffness[N/μm]	Air Consumption [l/min ANR]	-
525	48	3.1	-
performances maximum stiffness R			
Maximum sfiffness [N/μm]	Air gap [μm]	Load [N]	Air consumption [l/min ANR]
51	9.0	575	2.8

Table 3.1: Air bearing performance with 4 bar relative air supply pressure

3.1.4 Solenoid valve

Within the project, the solenoid valves are employed for both the control of the thrusters and the operation of the air bearings. Specifically, five Matrix *MX 821.100C224* valves have been used [15]: four of them control via a pulse-width modulation (PWM) signal all the thrusters, one valve for each direction, and one used as a digital valve controls the four air bearing.

The following figures 3.6 present a 3D representation of the valve, followed by a photograph 3.7 of the actual valve installed on board.

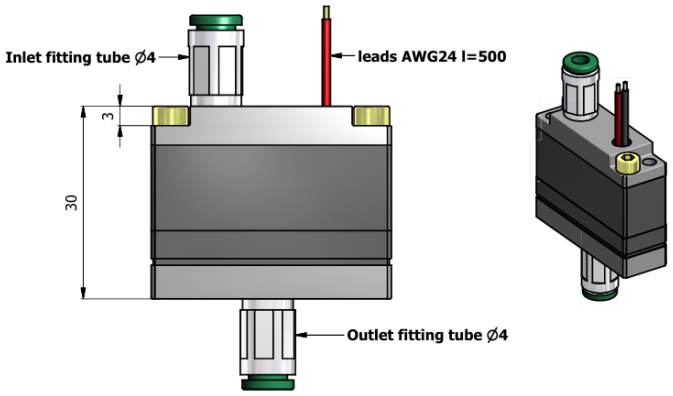


Figure 3.6: 3D representation of the valve

Figure 3.7: Valve picture

3.1.5 Air tank

The compressed air tank adopted in the project is a Smaco S400 model, featuring a capacity of $1 litre$ and a maximum operating pressure of $20 MPa$ [16]. This tank supplies the compressed air required to operate both the thrusters and the air bearings. The following image illustrates the reservoir used.

Figure 3.8: Air tank

3.1.6 Pressure regulator

To reduce the cylinder pressure to the levels required by the thrusters and the air bearings, two pressure regulators are employed: the first reduces the pressure from $200 bar$ to $7 bar$, while the second further decreases it to $4 bar$. One of the pressure regulator used is depicted in the figure 3.9 below.

Figure 3.9: Pressure regulator

3.1.7 Arduino

An Arduino Nano ESP32 microcontroller, illustrated in the following figure 3.10, is used as the onboard control unit. This board enables both the physical interfacing with the system components through its I/O pins and the development and execution of the control algorithms.

Figure 3.10: Arduino Nano ESP32

The topic of Arduino will be discussed in greater detail in the following chapters, focusing on both the design of the electrical schematic and the implementation of the control code.

3.1.8 Reaction wheel

The reaction wheel consists of a simple perforated brass disk, mounted onto the motor shaft and rotating rigidly with it. This component is responsible for generating the torque required to rotate the robot about its vertical axis. Below, a sectional drawing 3.11 of the component is provided, together with a picture 3.12 of the real component.

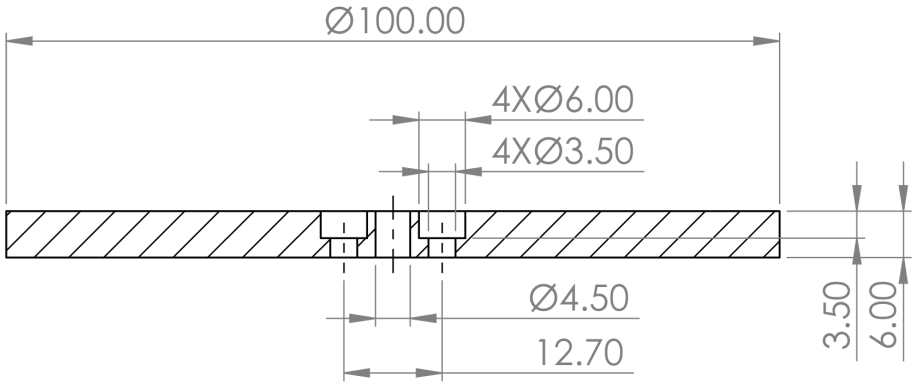


Figure 3.11: Reaction wheel technical drawing

Figure 3.12: Reaction wheel

3.1.9 Motor

The reaction wheel is actuated by a Maxon EC 45 Flat Brushless motor, characterized by an outer diameter of 42.9mm and a nominal power rating of 30W. The main technical specifications of the motor are summarized in Table 3.2, which reports the parameters with heir respective symbols, descriptions, and units of measurement for

clarity and ease of reference[17].

Name	Symbol	Value	Unit of measurement
torque constant	k_c	$25.5e-3$	Nm/A
voltage constant	k_e	$25.5e-3$	$V/(rad/s)$
Inductance	L	$0.56e-3$	H
Resistance	R	1.2	Ω
Rotor inertia	I_m	$9.25e-06$	kgm^2
Disk inertia	I_r	$1.3e-3$	kgm^2
coefficient of friction	Γ	0	$Nm/(rad/s)$
Nominal torque	C_n	$55e-3$	Nm

Table 3.2: Parameters of motor model

The figure 3.13 provides a visual representation of the component.

Figure 3.13: Motor picture

The DEC 50/5 module (Digital EC Controller) is a compact single-quadrant digital controller, specifically designed for the control of brushless DC motors (electronically commutated) with power ratings of up to $250W$. This driver is characterized by its

high versatility, enabled by a wide input supply voltage range (6 to 50VDC) [18]. The component is illustrated in the figure 3.14 below.

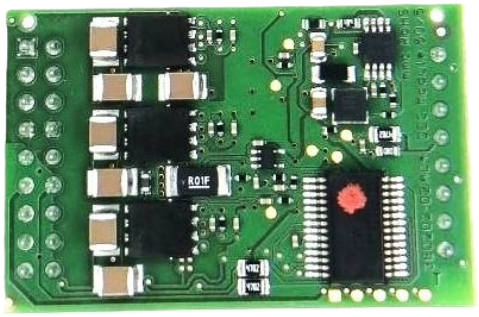


Figure 3.14: Motor driver picture

3.1.10 Battery

To accommodate the requirement for three distinct voltage levels, 12V for the motor, 24V for the valves and 5V for Arduino, two 12V, 0.65Ah RS Pro NiCd batteries are connected in series, showed in the following figure 3.15 [19]. So, this configuration supplies both a 12V output and a 24V output. Separately, the Arduino board is powered by a standard power bank. NiCd cells offer a balanced solution in terms of weight, robustness, and ease of integration, unlike lead-acid batteries, which are significantly heavier for the same capacity, and lithium batteries, which require more complex protection circuitry to ensure safe operation.

Figure 3.15: 12V 0.65Ah NiCd Battery pack

3.1.11 IMU

For the acquisition of position and velocity data, an MPU-9250 IMU sensor was employed. This component integrates an accelerometer, a gyroscope, and a magnetometer. Further details on the implementation are discussed in the following chapters. The component is depicted in the figure 3.16 below

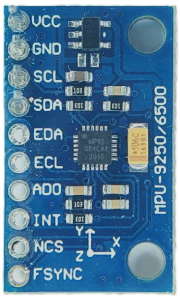


Figure 3.16: IMU MPU9250

3.2 CAD Design

Having now defined all the components that constitute the robot, it is possible to proceed with the development of the 3D Computer aided design (CAD) model. In the early stages of a project, a well-designed 3D model is essential for gaining a clear understanding of how the various elements can be assembled and, ultimately, for obtaining an accurate overview of the overall dimensions and spatial requirements of the final system.

In the SolidWorks environment, the ".STEP" files of the commercial components, downloaded directly from online catalogues, are imported, while the more generic components, or those for which no 3D CAD model is provided, are modelled directly within SolidWorks.

A series of support structures are then designed to secure the various components to the aluminium-frame body. Since these parts are intended to be manufactured using

fused-filament 3D printing on a *Bambu Lab X1C* printer, they are conceived with the goal of minimizing the need for printing supports and reducing material consumption. For this reason, a modular approach is adopted, producing smaller components that can be assembled together through interlocking features or bolts. To fasten the components to the aluminium profiles, a set of sliding T-Slot nuts manufactured by *Item* is used [20]; these are inserted into the profile grooves and feature threaded M5 holes matching the dimensions of the screws employed for the assembly.

3.2.1 Motor supports and tank holder

The motor and reaction wheel must be positioned as close as possible to the robot's central axis in order to ensure correct rotation about its own axis. However, the compressed-air cylinder is the heaviest single component of the system, and it is therefore equally important that it, too, be aligned with the robot's vertical axis. Building on this requirement, the support structure is designed with a dual function: it hosts both the motor and the reaction wheel, and it provides a stable mounting base for the cylinder.

The exploded view of the structure is shown in the following figure 3.17.

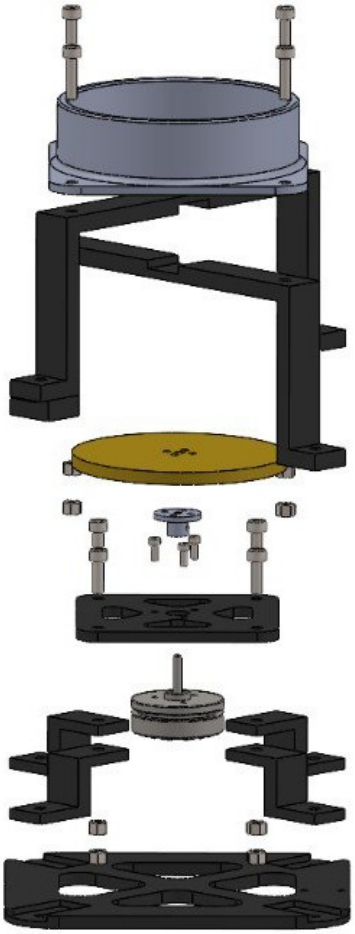


Figure 3.17: Exploded view of motor supports and tank holder

The positioning of this support takes advantage of the two profiles located in the lower part of the structure, visible in the figure 3.3.

Since these elements perform structural functions, all parts are printed in carbon-fiber-reinforced Polylactic acid (PLA), with the exception of the component that houses the cylinder which is made of standard PLA, whose primary role is to ensure correct alignment between the cylinder and the reaction wheel.

The assembled component is shown in the following image 3.18.

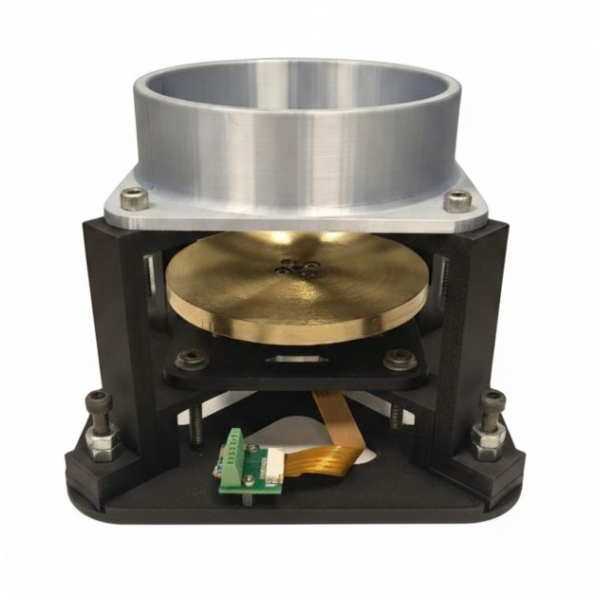
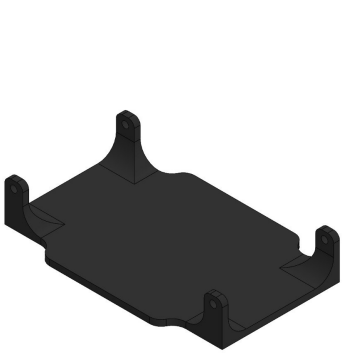


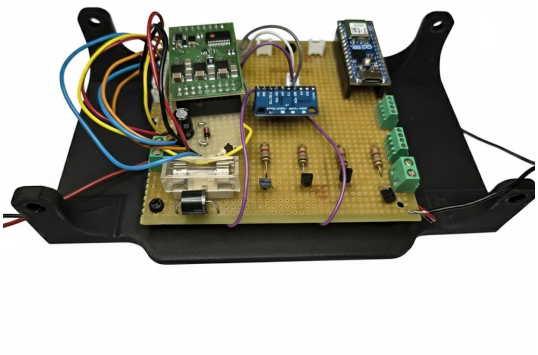
Figure 3.18: Motor supports and tank holder picture

3.2.2 Control board support

In the following chapters, the construction of the control board, which integrates all the electronic components of the simulator, including Arduino and the IMU, is discussed in detail. However, knowing the position of the board in advance is essential to allow for a more efficient assembly and soldering of the components. The board is therefore placed at the bottom of the robot to increase stability and to minimize the length of the wires connecting it to the motor and to the batteries. The space between the two profiles supporting the motor mount is utilized, allowing the control board to be positioned along the robot's central axis. The following figures show the support base CAD and the control board and its support base, connected with the aid of nylon spacers.



(a) Control board support CAD

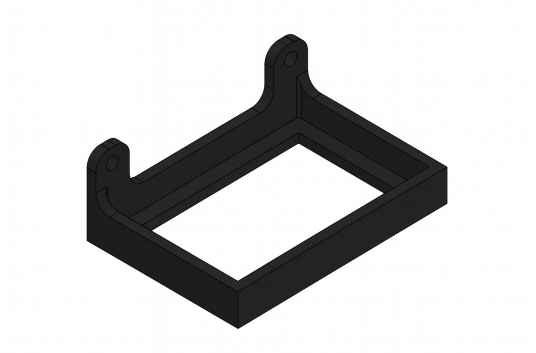


(b) Control board and its support base

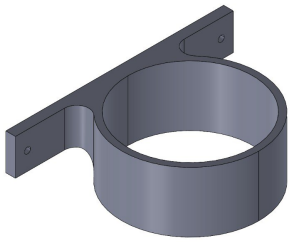
Figure 3.19: Control board support

3.2.3 Miscellaneous Supports

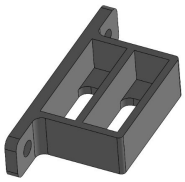
Additional support elements have been designed, such as battery enclosures 3.20a, a cylindrical holder to be fixed to the mid-height profile for the tank 3.20b, valve compartments 3.20c, air-bearing mounts 3.20d, and angled structures for integrating the thrusters 3.20e. The images of the designed components are presented below



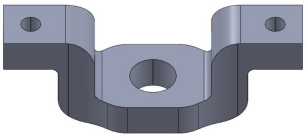
(a) battery enclosures



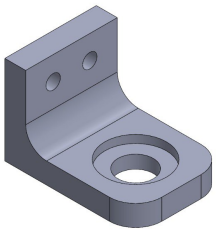
(b) Tank holder



(c) valve compartments



(d) Air bearing mounts



(e) thrusters support

Figure 3.20: Miscellaneous Supports

3.2.4 Complete CAD

By combining the 3D CAD models of the commercial components with those of the custom-designed parts, the final assembly of the simulator is obtained. An image of the complete model is reported below, with the main components highlighted using arrows.

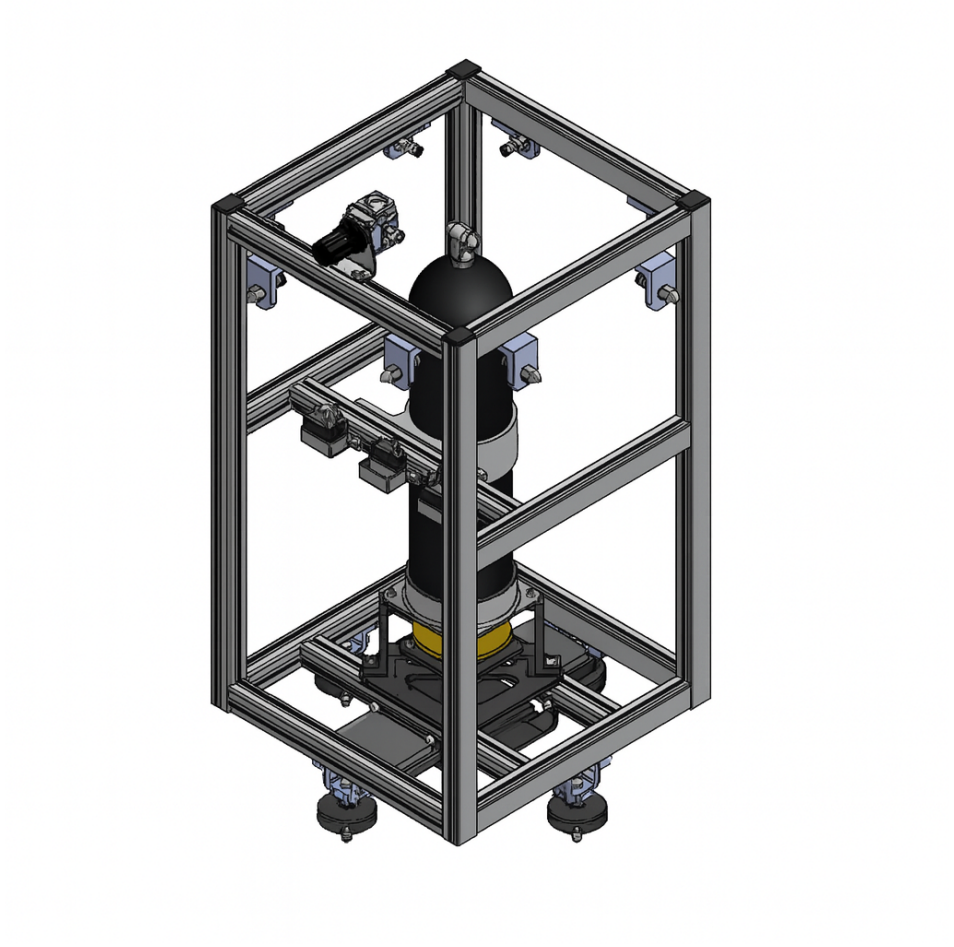


Figure 3.21: Airsat 3D assembly

Chapter 4

Digital twin and control strategy

4.1 Purpose of the model

The model has been conceived as a digital twin of the laboratory testbed, providing a virtual environment in which experiments can be carried out safely and without the risk of damaging real hardware. This framework serves as a foundation for testing and validation activities, offering both flexibility and reliability.

In the context of this thesis, the digital twin is employed to investigate control strategies for trajectory tracking. Specifically, the work focuses on the design of a controller capable of guiding the robot along a prescribed path while ensuring compliance with predefined velocity limits. Additionally, the control system is expected to maintain accurate trajectory tracking even in the presence of external disturbances, such as unexpected force perturbations. This approach not only facilitates a deeper understanding of the system dynamics but also provides a safe and effective means of assessing the performance of advanced control solutions prior to their implementation on the physical platform.

4.2 Digital twin

The development of the digital twin of the test bench was initiated using MATLAB Simulink, in conjunction with the Simscape Multibody extension. The objective of this model is to replicate the dynamic behaviour of the real system with a high degree of fidelity. To enhance the clarity and comprehensibility of the Simulink model, its architecture has been structured into four main subsystems (fig. 4.1): Body, Sensors, Actuation, and Control. This modular organization facilitates both the description and the analysis of the system, allowing each functional block to be examined in isolation before considering their interactions. In the present section, these subsystems will be introduced individually, while subsequent discussion will place particular emphasis on the Control subsystem, as it constitutes one of the core themes of this thesis.

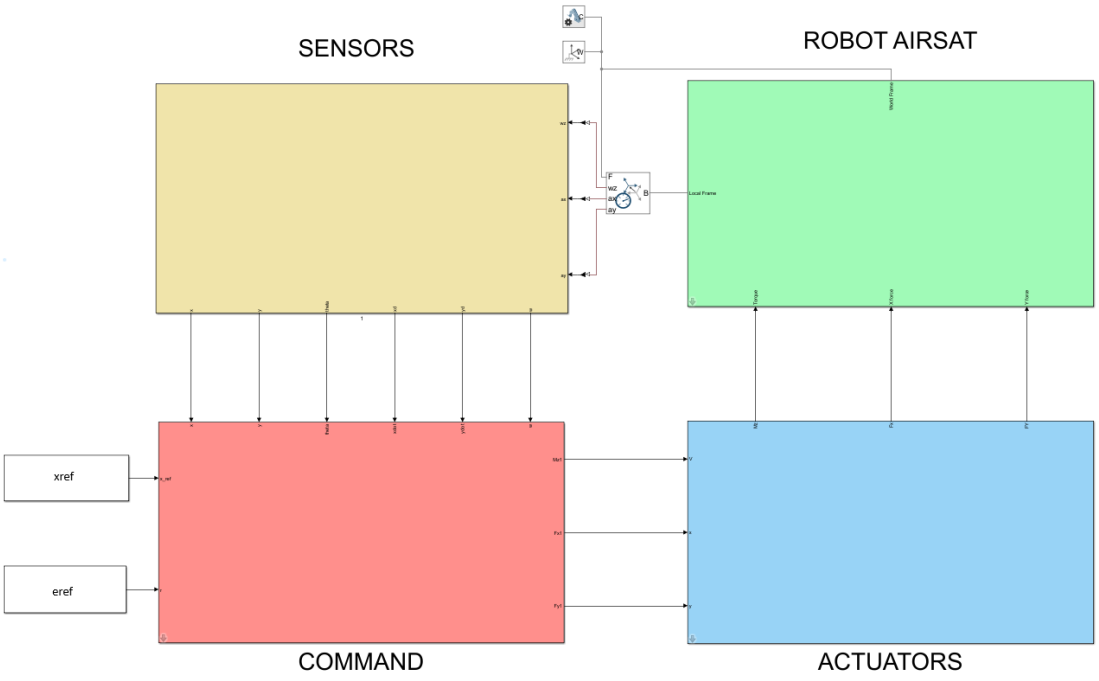


Figure 4.1: Simulink Model Homepage

4.2.1 Body

The modelling process began with the reproduction of the robot's physical structure: as shown in the following figure 4.2, the frame was represented through rigid bodies whose masses, dimensions, and spatial positions were consistent with those of the CAD model of the real system. For the sake of computational efficiency, the structural representation was simplified to a single parallelepiped, while preserving the key physical parameters necessary for accurate dynamic simulation.

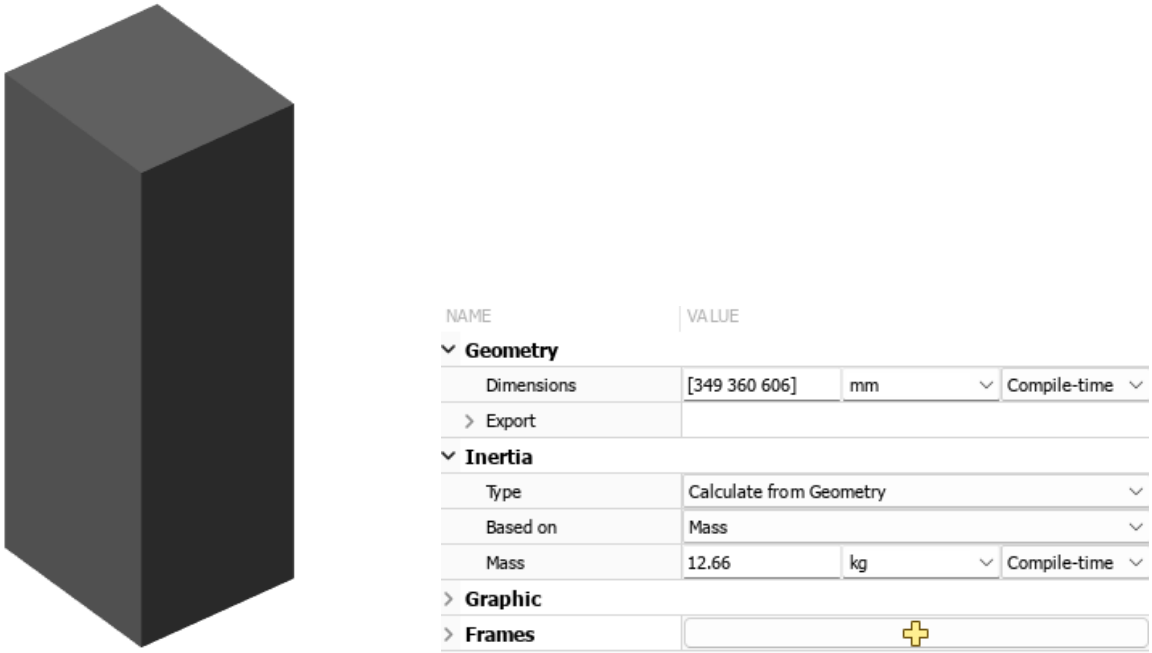


Figure 4.2: Body frame Simulink model and technical specifications

The second physical element incorporated into the model is the reaction wheel, represented in a simplified form as a rigid disk (figure 4.3) coaxial with the robot and constrained to rotate solely about its own axis through the use of a revolute joint. A rigid transformation block is employed to position the reaction wheel at the same height as in the CAD reference; however, this transformation serves only for spatial consistency within the model and does not play a critical role in the system's functional

behavior. The choice to model the reaction wheel as an ideal rigid disk was made to reduce computational complexity while retaining the essential inertial properties that influence the robot's rotational dynamics. This abstraction allows the simulation to capture the wheel's contribution to attitude control without introducing unnecessary geometric details that would have minimal impact on the overall system behaviour.

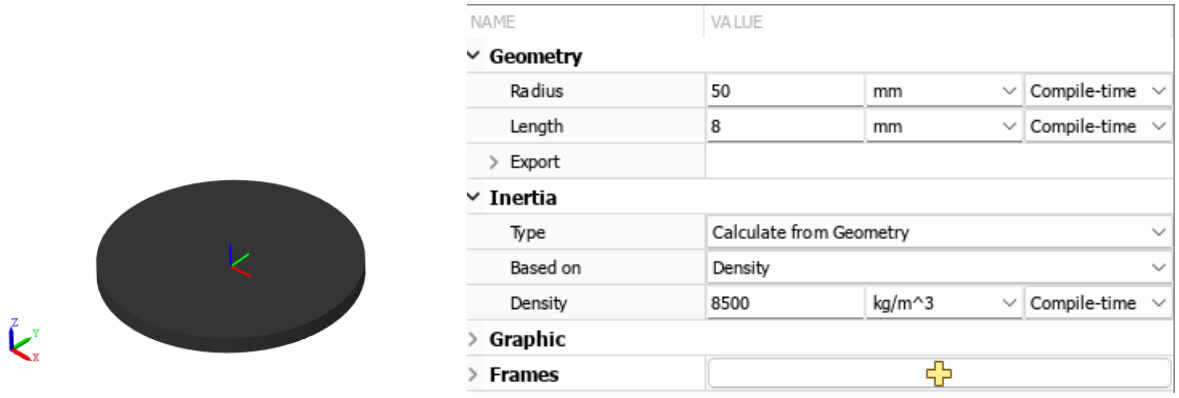


Figure 4.3: Reaction wheel Simulink model and technical specifications

4.2.2 Sensors

The sensor subsystem presents a relatively straightforward configuration. The physical sensing devices onboard the robot are represented in the model by a Transform Sensor block, as shown in figure 4.4, which measures the linear accelerations along the principal axes and the angular velocity about the axis orthogonal to the plane of motion.

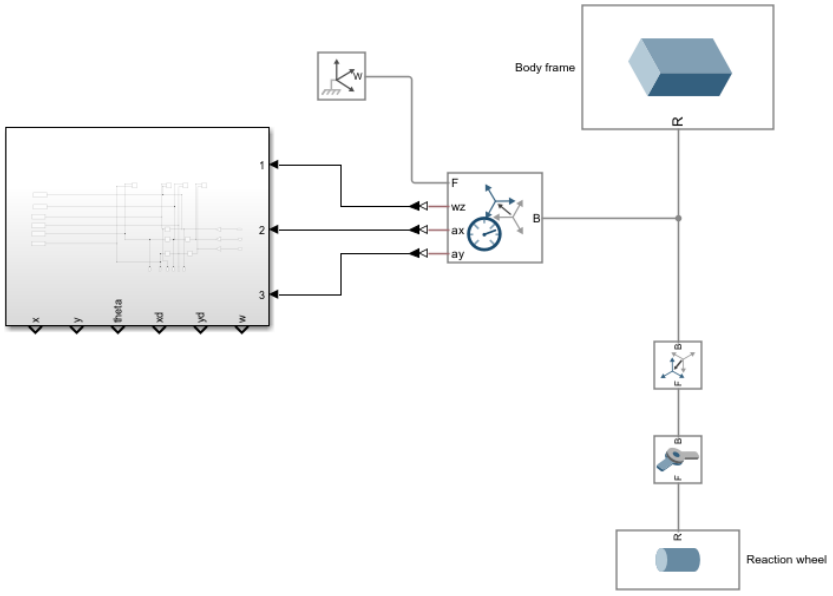


Figure 4.4: Sensor block and Simulink connections

A closer inspection of the model connections reveals that the Body Frame (denoted by B in the Simulink block, as illustrated in Figure 4.4) is attached to the robot, whereas the Follower Frame (denoted by F) is linked to the World Frame, i.e., the inertial reference system. This arrangement results in the measured values being inverted with respect to the desired sign convention. To address this, the sensor outputs are multiplied by a gain of -1 to ensure consistency in the reference frames. Subsequently, integration blocks are employed to obtain the corresponding position and orientation quantities from the measured accelerations and angular velocity. These derived signals are then utilized in the control subsystem for feedback and state estimation purposes. The following figure 4.5 represents the subsystem under discussion.

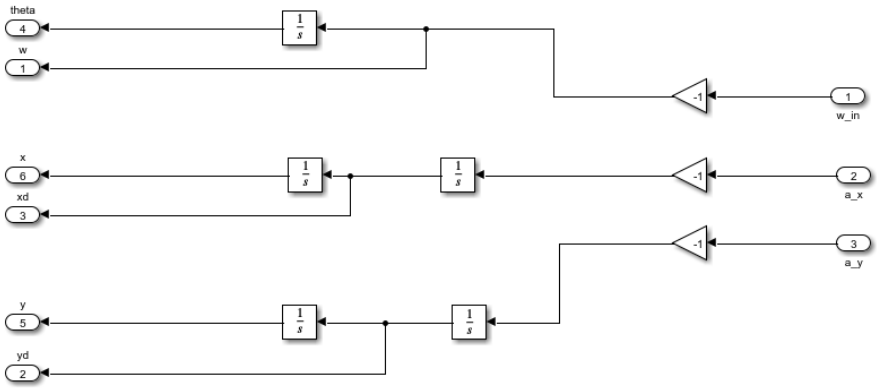


Figure 4.5: Sensor subsystem

4.2.3 Actuation

With regard to actuation, two distinct mechanisms can be identified: translational motion, resulting from the pneumatic thrust generated by the thrusters, and rotational motion, ensured by the reaction wheel. In the following subsections, the modelling approaches underlying each of these systems are examined separately, with the aim of elucidating their respective operating principles and their role within the overall dynamics of the robot.

Translation motion

As previously discussed, translational motion is provided by pairs of pneumatic thrusters and solenoid valves, which, through PWM) control, are capable of delivering a variable thrust force. The magnitude of this force is determined by the supply pressure of the thrusters, which is controlled by a pressure regulator supplied by the onboard tank. In the Simulink model, a simplification was introduced: each thruster is represented as a purely convergent nozzle, while the valve model was omitted. Preliminary tests demonstrated that including the valve dynamics had no significant impact on the overall behavior of the system, while unnecessarily increasing the computational load.

The thruster model was developed using the governing equations for convergent nozzles obtained from NASA's technical resources [21], illustrated in the following figure 4.6, and its schematic representation is shown in figure 4.7 [21].

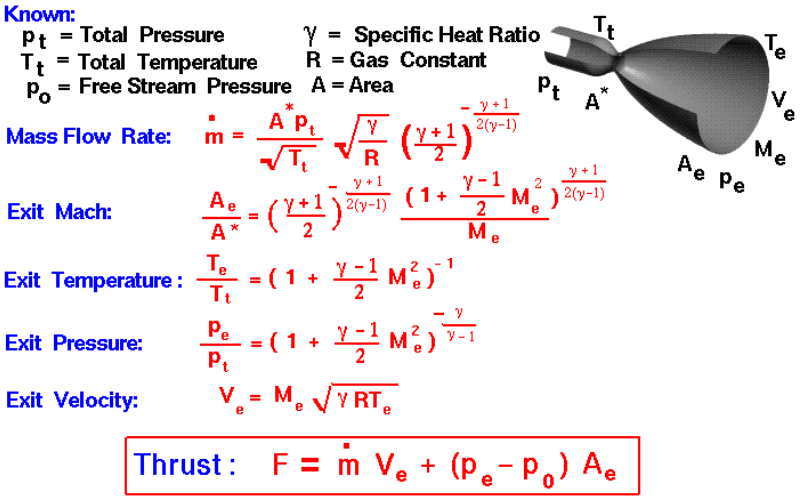


Figure 4.6: NASA thruster equations

All the parameters employed in the model are reported in the table below 4.1.

Name	Symbol	Value	Unit of measurement
Temperature	T	293	K
Heat capacity ratio	γ	1.4	–
Nozzle throat	A	$7.854e-7$	m^2
Gas constant	R_{gas}	287	$J/(kgK)$
Thrusters' supply pressure	p	7	bar

Table 4.1: Data of pneumatic mode

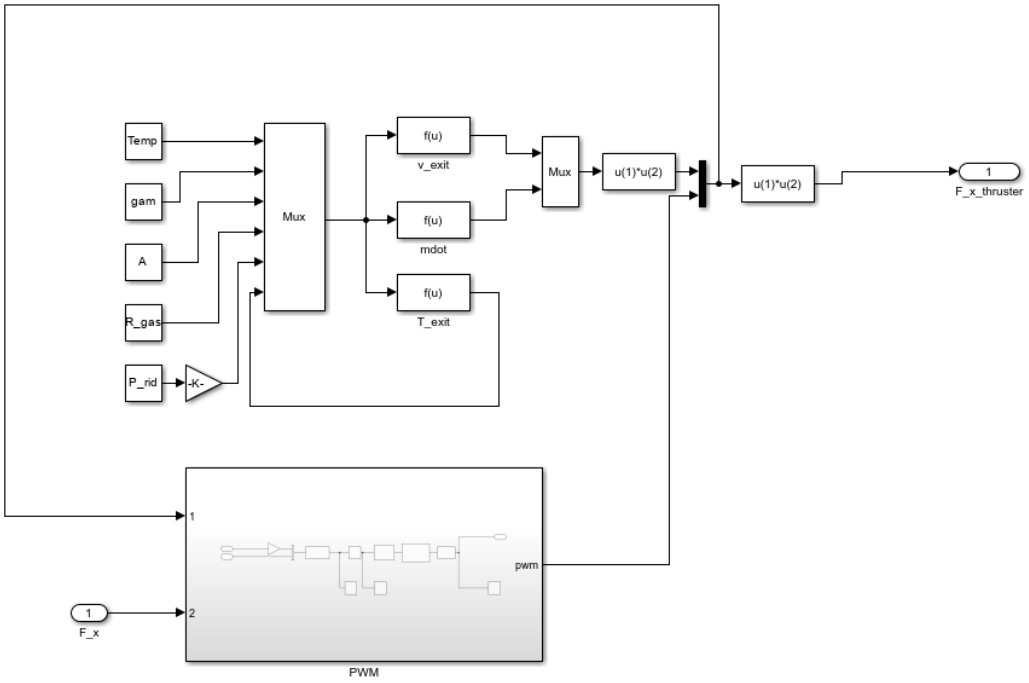


Figure 4.7: Thruster model

To obtain an accurate estimate of the force required to achieve the displacement commanded by the control system, the thruster model alone is insufficient, as it can only represent the maximum deliverable thrust. For this reason, the thruster model was complemented with a PWM block, which modulates its output force proportionally to the theoretical command signal, with a modulation period of $0.01s$ in accordance with the valve specifications. The thrust produced by each individual thruster is therefore computed as the product of its maximum deliverable force and duty cycle (dc) (output of the PWM block), the latter being a dimensionless value ranging between 0 and 1, as expressed in eq.(4.2.1)

$$F_{thruster} = F_{max} \cdot dc \tag{4.2.1}$$

A closer examination of the PWM subsystem reveals that, in addition to the stan-

dard PWM block provided by Simulink, several auxiliary blocks have been incorporated to enhance the correspondence between the model and the real system. As will be discussed in a later section, the control block generates three output signals (two force commands and one torque command) which serve as inputs to the respective actuation blocks. As shown in the figure 4.8 before reaching the PWM block, the command signal undergoes a series of preprocessing steps to ensure correct operation. First, it is normalized with respect to the force requested by the control block. The resulting value is then saturated within the range $[0, 1]$, passed through a low-pass filter, and subsequently adjusted to account for the operational constraints of the solenoid valve. Specifically, for dc below 0.1, the valve is unable to open, resulting in a null output; conversely, for dc above 0.9, the valve's response becomes slower than the input command, effectively remaining fully open and producing a constant output of 1.

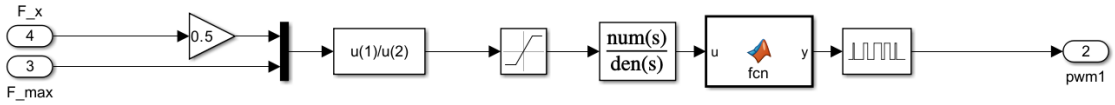


Figure 4.8: PWM subsystem

Finally, the total translational force is computed as the sum of the contributions provided by each pair of thrusters, arranged along the four sides of the robot, represented in the figure 4.9.

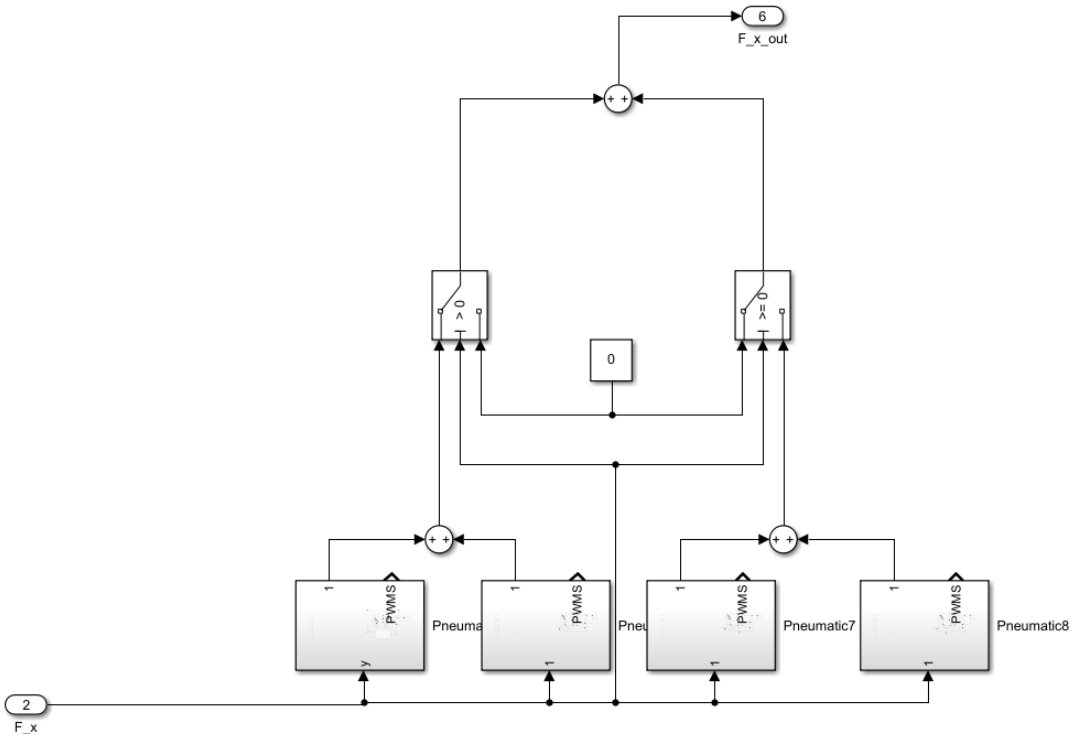


Figure 4.9: Entire translation motion subsystem

Rotational motion

Rotational motion is provided by a reaction wheel, conceptually comparable to a rigid disk driven by an electric motor. Since the rotating disk has already been modelled as part of the body subsystem, the present section focuses on the modelling of the electric motor and the corresponding command signal. The development of the motor model, simulated as a DC motor, requires reference to the fundamental electrical and mechanical equations (4.2.2) governing its operation, namely the loop (or armature) voltage equation, the torque generation equation, and the rotational equilibrium equation. These relationships provide the basis for accurately reproducing the motor's electromechanical behaviour within the simulation environment. By applying the Laplace transform to these governing equations, the system can be expressed in

the frequency domain, thereby facilitating the derivation of a compact mathematical representation suitable for implementation within the Simulink environment.

$$\begin{cases} \bar{V} = R \cdot \bar{i} + L \cdot s \cdot \bar{i} + k_e \cdot \bar{\omega} \\ \bar{C_m} = k_c \cdot \bar{i} \\ \bar{C_m} = I_r \cdot s \cdot \bar{\omega} + I_m \cdot s \cdot \bar{\omega} + \Gamma \cdot \bar{\omega} \end{cases} \tag{4.2.2}$$

Consequently, the driving torque (4.2.3) delivered by the motor can be expressed as:

$$\bar{C_m} = \frac{k_c \cdot (I_m + I_r) \cdot s + \Gamma \cdot kc}{L \cdot (I_m + I_r) \cdot s^2 + (R \cdot (I_m + I_r) + L \cdot \Gamma) \cdot s + (k_e \cdot k_c + R \cdot \Gamma)} \cdot \bar{V} \tag{4.2.3}$$

The parameters introduced in the preceding equations are summarized in Table 3.2,previously discussed. As already mentioned, the system input is a torque command, which must be converted into a voltage signal. Since the motor control voltage is required to be proportional to the motor speed, the motion torque is first used to compute the angular acceleration, which is then integrated to obtain the rotational speed. This value is subsequently normalized with respect to the maximum voltage of $12V$, with an efficiency of 0.95%. The motor driver subsystem is illustrated in the figure below 4.10.

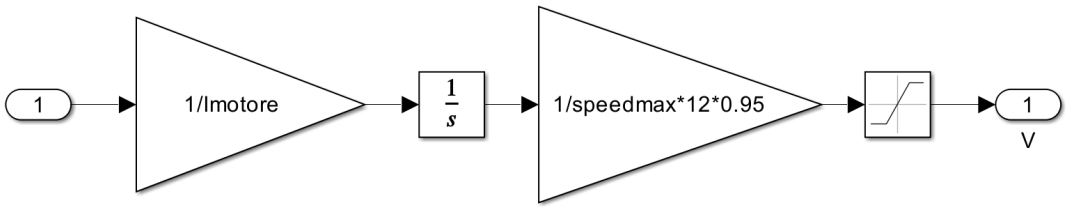


Figure 4.10: Motor driver block

4.2.4 Control

This final section focuses on the control subsystem, which represents the core of the model. Based on the outcomes of the literature review, a Linear Quadratic Integral (LQI) controller was selected as the most suitable strategy. The following discussion is therefore devoted to outlining the operating principles of this control approach and to clarifying the rationale behind its adoption in the present application. As previously outlined, the development of a trajectory control system necessarily begins with the specification of the reference path to be followed. In the present work, a sinusoidal curve has been selected, as it effectively highlights both translational and oscillatory dynamics. Nonetheless, the proposed methodology is not confined to this choice, since it can be readily extended to a broad class of paths and motion profiles. The sinusoidal function adopted as reference is defined by the following equation (4.2.4), and plotted in the figure 4.11:

$$y(x) = a_2 \cdot cos(\omega \frac{x}{a_1}) - a_2 \tag{4.2.4}$$

where the constants a_1,a_2 and ω define the scaling along the trajectory. The specific values employed in the simulations are summarized in the following table 4.2.

a_1	0.1
a_2	0.5
ω	0.1

Table 4.2: Sinusoidal trajectory parameters

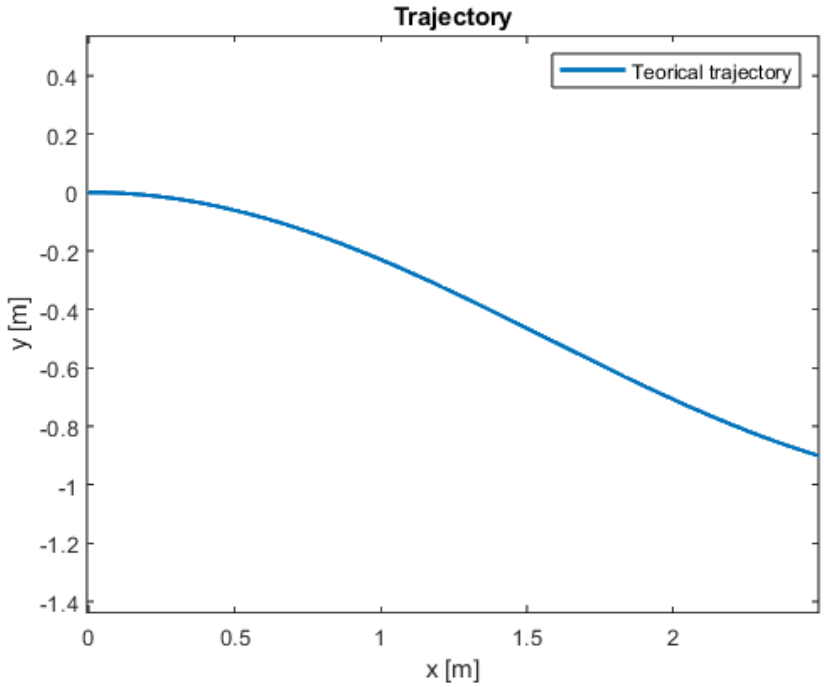


Figure 4.11: Theorical trajectory

In order to constrain both the maximum velocity and acceleration of the system, the absolute velocity of the robot along the predefined path was shaped to approximate a trapezoidal velocity profile, plotted in figure 4.12, characterized by piecewise-constant acceleration. This choice ensures a smooth yet controllable motion while respecting physical limitations. As a practical compromise, the maximum acceleration was set to $0.002 m/s^2$, and the maximum velocity was limited to $0.06 m/s$.

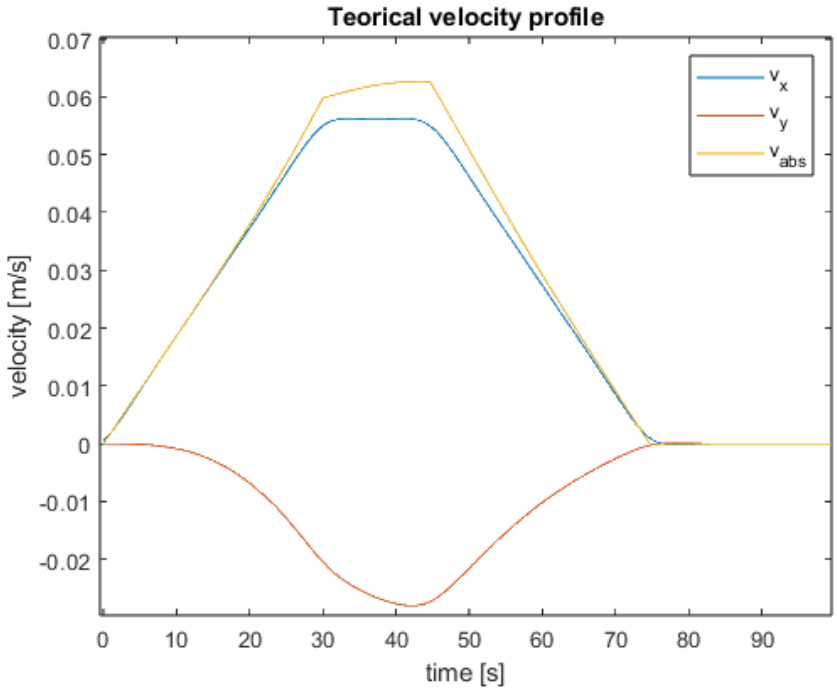


Figure 4.12: Theorical velocity profile

In order to ensure the scalability of the model to any arbitrary trajectory, identifying an algorithm capable of effectively constraining the velocity for all possible paths proved to be a non-trivial task. The adopted solution, detailed in the Appendix A) introduces a curvilinear coordinate s defined along the trajectory. By means of an external function, *Trajectory form*, also shown in Appendix A, the algorithm reconstructs the motion profile, generating either a trapezoidal velocity profile or, in cases where the imposed velocity bounds are too high to allow the full development of such a profile, a triangular one.

Linear Quadratic Integral

The Linear Quadratic Regulator (LQR) and its extension, the LQI, are optimal control strategies widely employed in modern control theory[22]. Their objective is to design a feedback controller that minimizes a quadratic cost function (4.2.5) of the form

$$J = \int_0^\infty \left(\bar{\xi}^T(t) Q \bar{\xi}(t) + \bar{u}^T(t) R \bar{u}(t)\right) dt \tag{4.2.5}$$

where $\bar{\xi(t)}$ represents the state vector,$\bar{u}(t)$ the control input, while Q and R are positive semi-definite and positive definite weighting matrices, respectively. By adjusting these matrices, it is possible to balance the trade-off between state regulation accuracy and control effort.

This formulation guarantees stability and performance in a systematic manner, overcoming the limitations of heuristic tuning approaches such as PID. Among its advantages, LQR/LQI provides robustness to modelling uncertainties, the capability to address multi-variable systems within a unified framework, and explicit control over the compromise between accuracy and energy consumption. In particular, the LQI formulation enhances steady-state performance by introducing an integral action, thus ensuring rejection of constant disturbances and maintaining trajectory tracking with negligible steady-state error. In order to advance with the study and development of the control strategy, a preliminary dynamic analysis of the system is deemed necessary, with the aim of identifying and understanding the fundamental variables involved in the regulation process.

Dynamic analysis

The planar robot under consideration exhibits three degrees of freedom: two translational (x, y) and one rotational (θ). To design an optimal controller, the system is first represented in state–space form.

The state vector is defined as:

$$\overline{x} = \left[x, y, \theta, v_x, v_y, \omega\right]^T$$

where (x, y) are the planar coordinates, (θ) is the yaw angle,(v_x, v_y) the linear velocities, and (ω) the angular velocity. The input vector collects the generalized forces and torque acting on the system:

$$\overline{u} = \left[F_x, F_y, C_m\right]^T$$

with F_x, F_y denoting the translational forces in the inertial frame and C_m the torque around the out-of-plane axis. The robot dynamics, obtained from Newton–Euler equations, can be written as:

$$\begin{cases} \dot{x} = v_x \\ \dot{y} = v_y \\ \dot{\theta} = \omega \\ \dot{v_x} = \frac{F_x}{m} \\ \dot{v_y} = \frac{F_y}{m} \\ \dot{\omega} = \frac{C_m}{I_z} \end{cases}$$

where m is the mass of the robot and I_z its polar moment of inertia. In compact matrix form:

$$\dot{\overline{x}}(t) = A\overline{x}(t) + B\overline{u}(t)$$

with

$$\left\{ A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1/m & 0 & 0 \\ 0 & 1/m & 0 \\ 0 & 0 & 1/I_z \end{bmatrix} \right\}$$

This linear formulation is valid for forces expressed in the inertial frame and serves as the basis for control design.

Control strategy

In trajectory tracking problems the goal is not merely to stabilize the system around the origin, but to ensure convergence toward a time-varying reference. For this reason, the control law is defined on the tracking error

$$\overline{e}(t) = \overline{x_{ref}}(t) - \overline{x}(t)$$

where $\overline{x_{ref}}(t)$ denotes the vector of theoretical quantities analytically derived from the prescribed trajectory. Using the error formulation guarantees that the cost function penalizes deviations from the desired trajectory, rather than the absolute state absolute state $\overline{x}(t)$. This prevents the controller from converging to the trivial equilibrium at the origin, which would otherwise be irrelevant when the reference is not zero.

LQI extension

Since an LQI controller has been adopted, it is necessary to extend the state vector $x(t)$ with an additional component representing the integral of the tracking error. This auxiliary state, denoted as $z(t)$, whose derivative is defined as

$$\dot{\overline{z}}(t) = \overline{r} - C\overline{x},$$

where $\overline{r}$ is the reference signal

$$\overline{r} = \left[x_{ref}, y_{ref}, \theta_{ref}\right]^T$$

and C the output matrix selecting the controlled variables (e.g., positions and rotation x,y,θ).

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The augmented state vector then becomes $\overline{\xi} = \begin{bmatrix} \overline{x}(t) \\ \overline{z}(t) \end{bmatrix}$

Consequently, The augmented system is then:

$$\dot{\overline{\xi}}(t) = A_{LQI}\overline{\xi}(t) + B_{LQI}\overline{u}(t) + E_{LQI}\overline{r}$$

with

$$A_{LQI}^{[9x9]} = \begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, B_{LQI}^{[9x3]} = \begin{bmatrix} B \\ 0 \end{bmatrix}, E_{LQI}^{[9x3]} = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

The final step required to solve the Riccati equation and thus obtain the optimal gain matrix K consists in selecting appropriate weighting matrices Q and R. The matrix Q, commonly referred to as the state weighting matrix or state cost matrix, penalizes deviations of the system states from their desired values. The matrix R, known as the control weighting matrix or control cost matrix, penalizes the magnitude of the control inputs. Regarding the selection of the weighting matrices Q and R, it was decided that, in the inertial reference frame, the entries of Q should be set proportional to the inverse of the square of the maximum acceptable error for each individual state. Similarly, the entries of R were chosen proportional to the inverse of the square of the maximum allowable control input[23].

$$Q = diag([1/\varepsilon_x^2, 1/\varepsilon_y^2, 1/\varepsilon_\theta^2, 1/\varepsilon_{vx}^2, 1/\varepsilon_{vy}^2, 1/\varepsilon_\omega^2, 1/\varepsilon_{int_x}^2, 1/\varepsilon_{int_y}^2, 1/\varepsilon_{int_\theta}^2])$$
$$R = diag([1/F_{max}^2, 1/F_{max}^2, 1/(C_{max})^2]);$$

By solving the equations using MATLAB's *lqr* function, as detailed in the Appendix A, it is possible to obtain the optimal gain matrix K, which is composed of two sub-matrices, K_1 and K_2. These sub-matrices allow the control input to be computed as follows:

$$\overline{u} = -K_1\overline{e} - K_2\overline{z}$$

However, it must be emphasized that the forces computed as the output of the LQI controller are expressed in the inertial (fixed) reference frame, since both the reference and measured states are defined in this coordinate system, rather than in the robot's body frame. Therefore, it becomes necessary to rotate these vectors through a rotation matrix, making use of the robot's measured orientation angle θ. The transformation is performed using the direct rotation matrix, defined as:

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

In Figure 4.13, the construction of the rotation matrix performed using Simulink is shown.

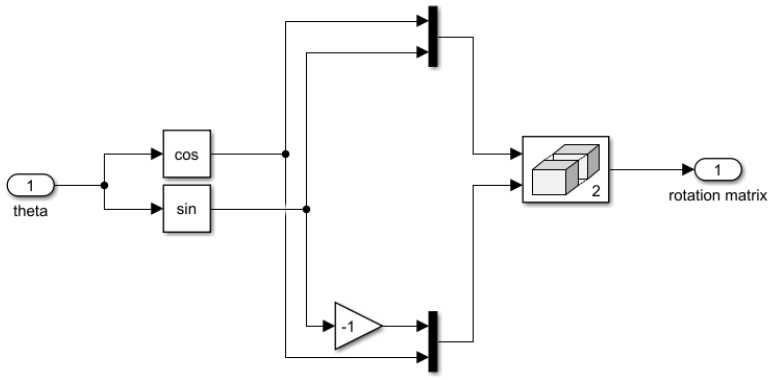


Figure 4.13: Simulink model of rotation matrix

The entire control model was therefore consolidated into the following subsystem 4.14.

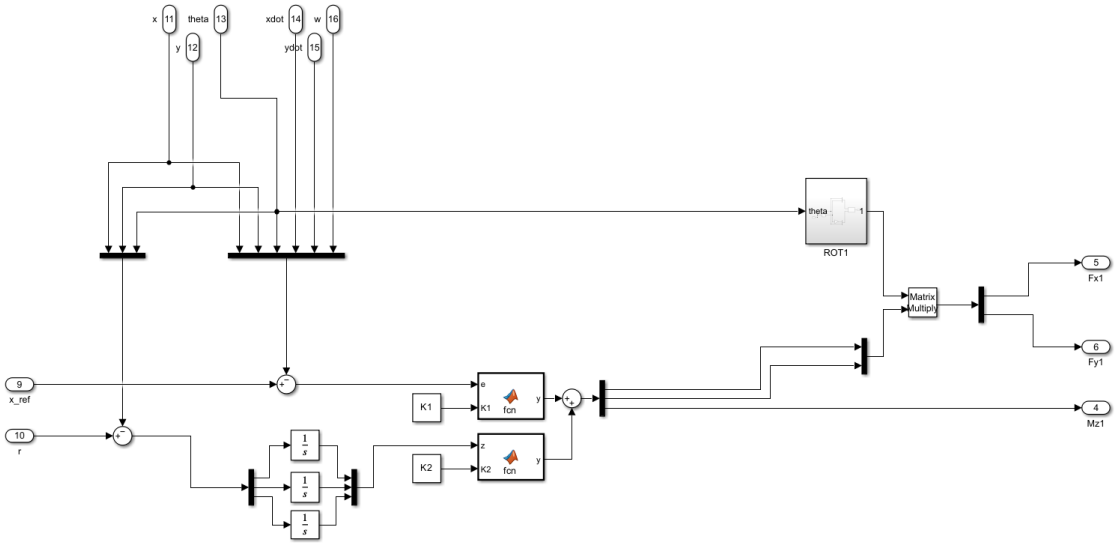


Figure 4.14: Control model

4.2.5 Model results with sinusoidal trajectory

In this final section, the results obtained from the Simulink simulation are presented. The analysis focuses not only on verifying the correct tracking of the desired trajectory and compliance with the imposed velocity limits, but also on evaluating the forces and torques generated during the motion. The first result, illustrated in the following figure 4.15, concerns the trajectories and confirms that the robot successfully maintained the prescribed path, demonstrating the effectiveness of the implemented control strategy.

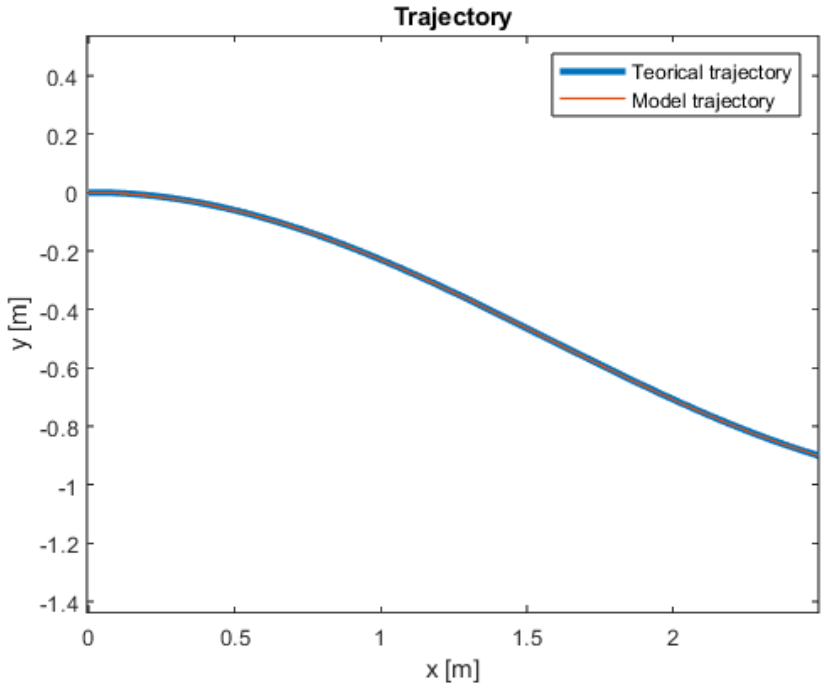


Figure 4.15: comparison of the trajectories

The following figures 4.16, 4.17, 4.18, confirm that the velocity achieved by the model closely matches the theoretical profile, providing further evidence of the accuracy of the implemented control approach.

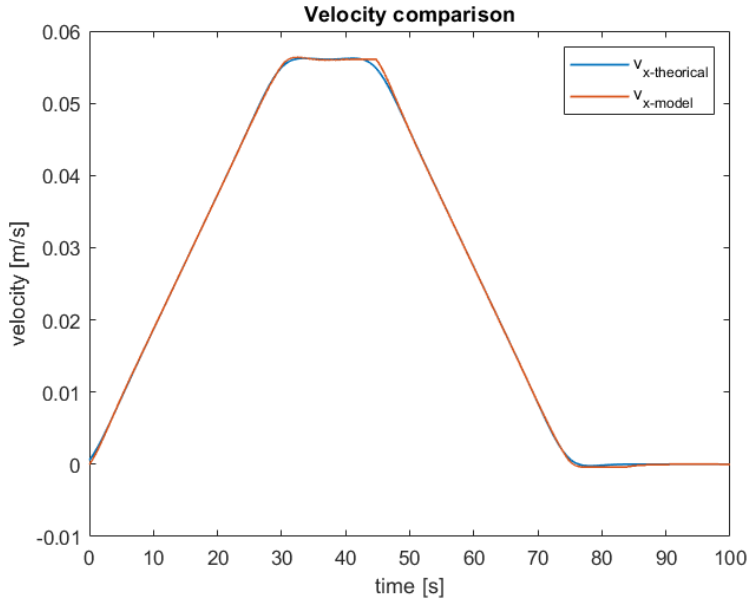


Figure 4.16: Comparison of x velocities

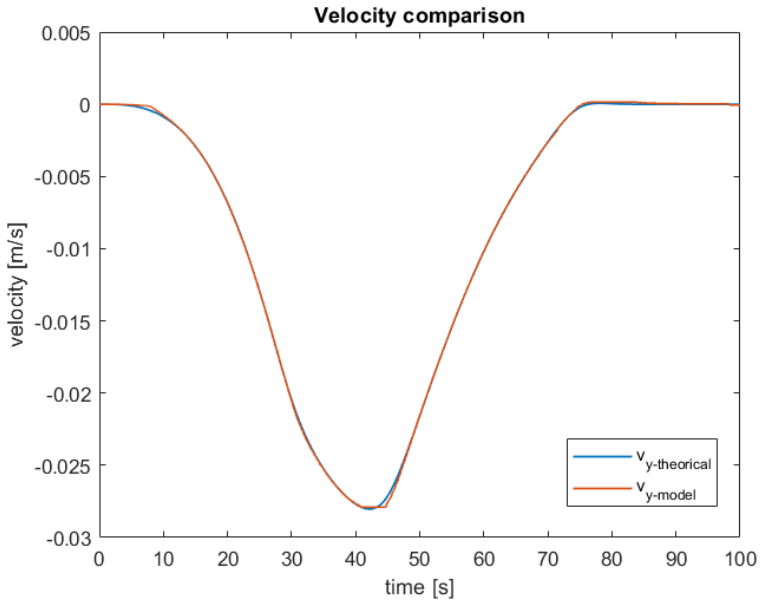


Figure 4.17: Comparison of y velocities

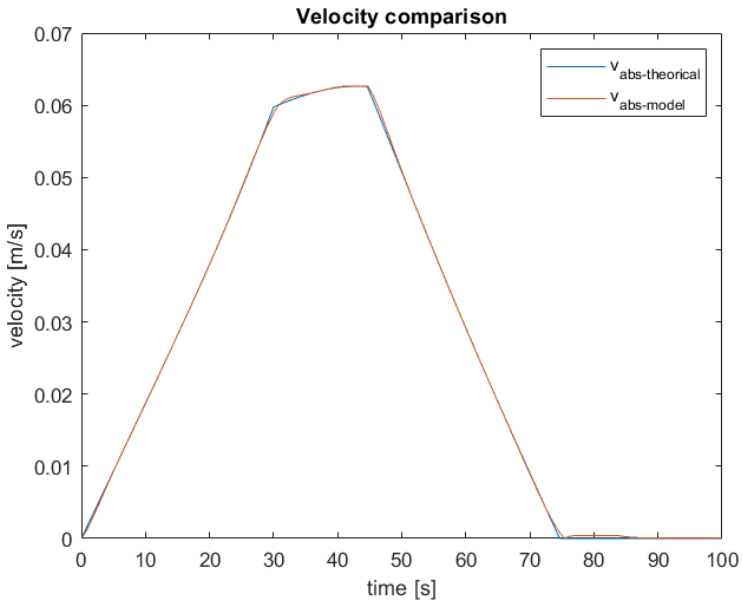


Figure 4.18: Comparison of absolute velocities

A comparison between the theoretical orientation, automatically computed by the control code to ensure that one face of the robot is always aligned with the direction of motion, and the simulated orientation obtained from the model is also presented 4.19.

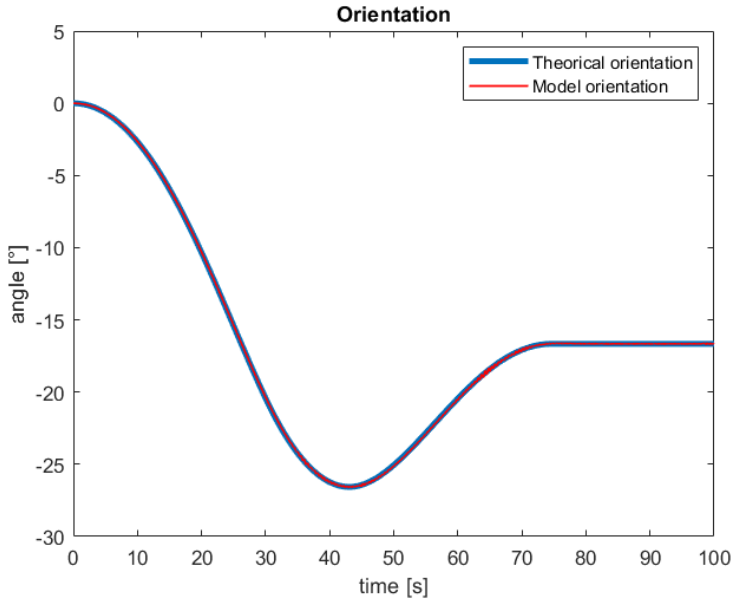


Figure 4.19: Orientation comparison

It can be observed that the two values perfectly coincide.

Below are shown the time trends of the errors obtained during the simulation. It can be observed that the position error 4.20 is clean and stable, consistently remaining below 5 mm, which indicates a high level of control accuracy. The orientation error, although very small in magnitude, appears slightly more affected by noise, a behaviour that can also be seen in the velocity error plot 4.21, where small residual oscillations are present.

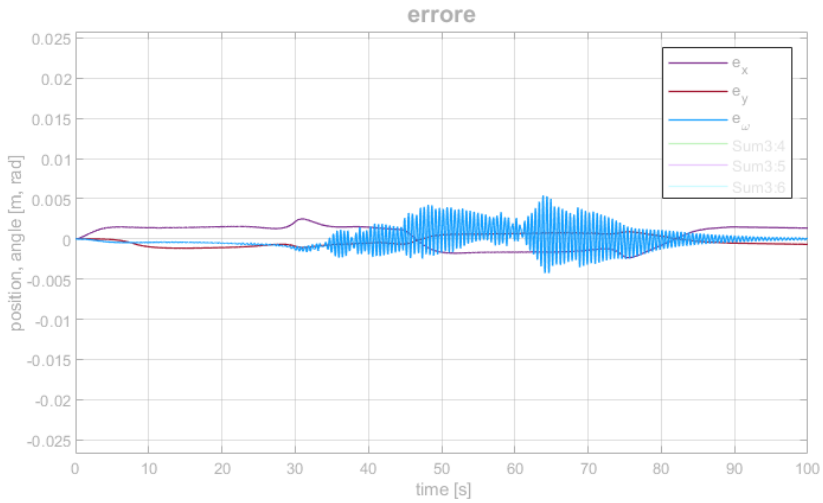


Figure 4.20: Position and orientation error

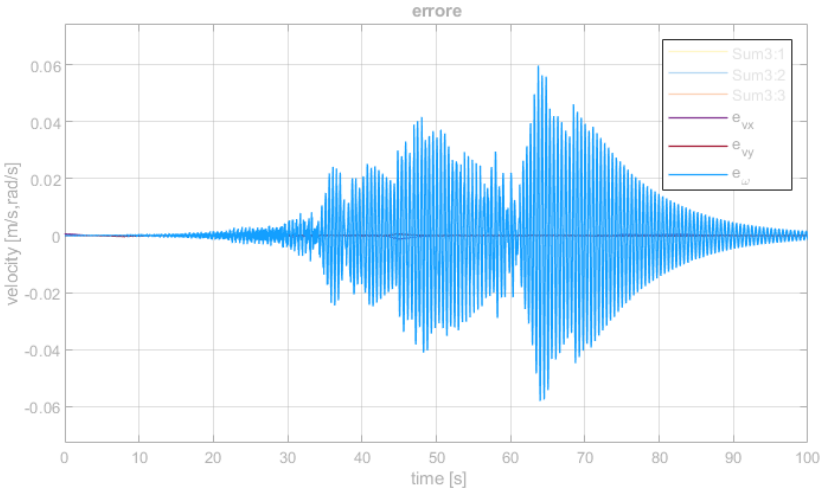


Figure 4.21: Velocity error

These disturbances are nevertheless considered tolerable, as the system maintains satisfactory and repeatable performance. With the same control parameters, the model can operate correctly even when following different trajectories, such as the parabolic trajectory whose results are shown at the end of this chapter, thus confirming the

universality and robustness with which both the control code and the simulation model were designed.

In the following figures, extracted directly from Simulink through the scope block, it can be observed that the motor command voltage remains limited, never reaching the saturation threshold of $12V$ (fig.4.23). Similarly, the torque delivered by the motor remains within the range of the maximum torque, which is approximately twice the nominal value of $55 \times 10^{-3} Nm$ (fig. 4.22).

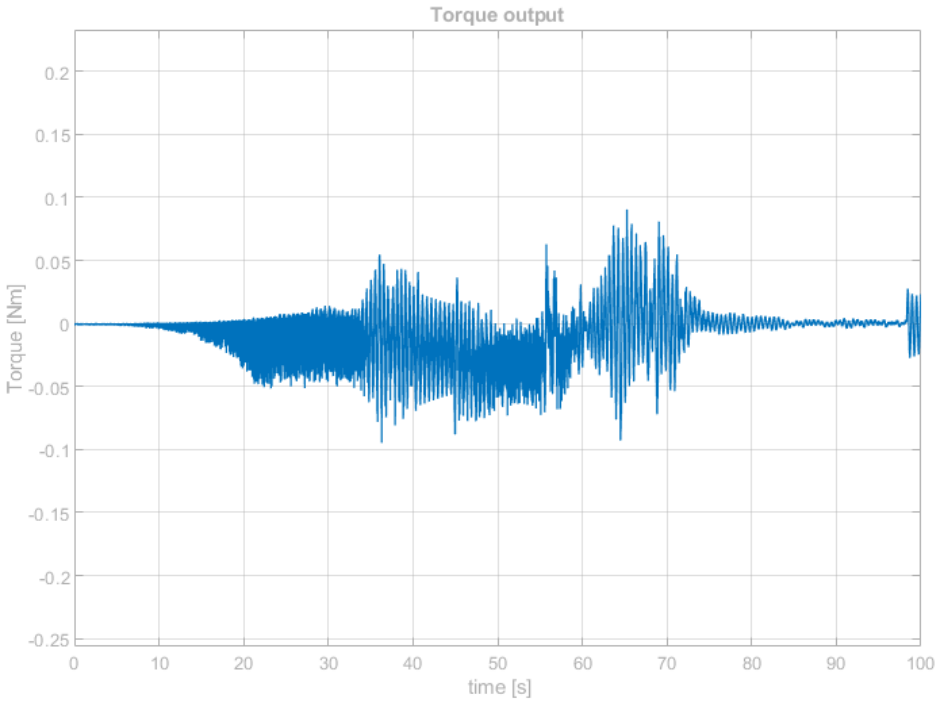


Figure 4.22: motor torque output

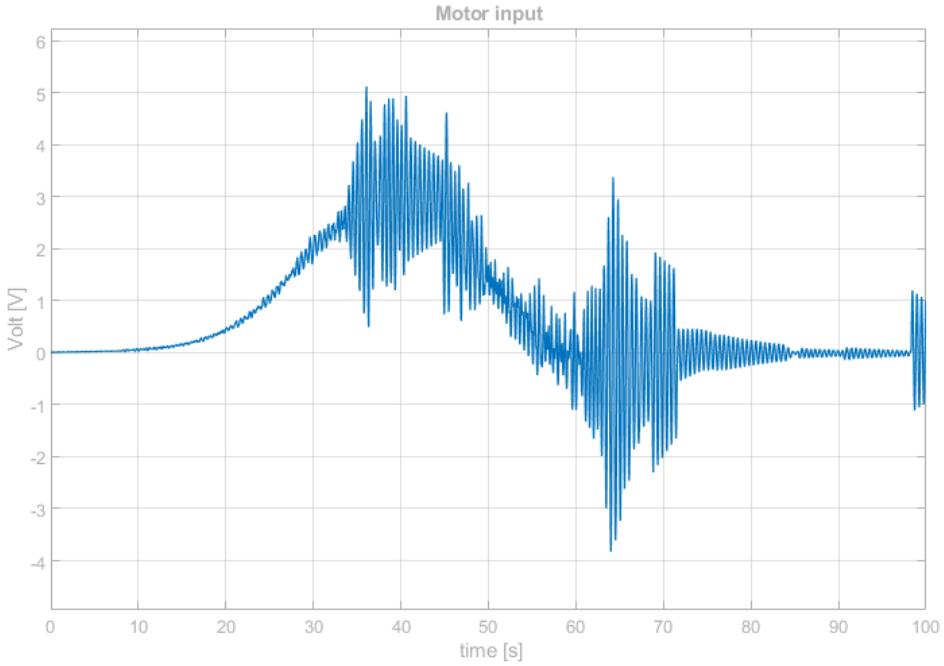


Figure 4.23: motor command voltage

Finally, the resulting diagrams of the PWM-modulated forces along the x and y axes, expressed in the robot's local reference frame, are presented (fig.4.24, fig. 4.25).

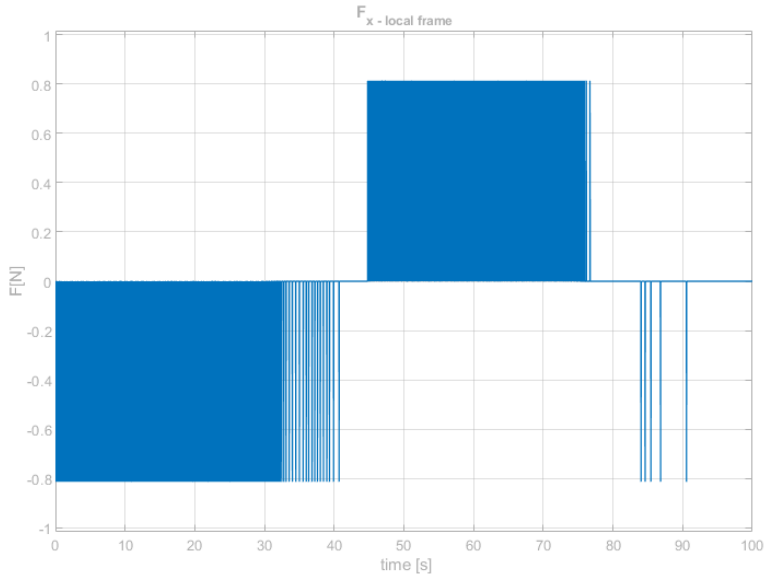


Figure 4.24: x-Force scopes

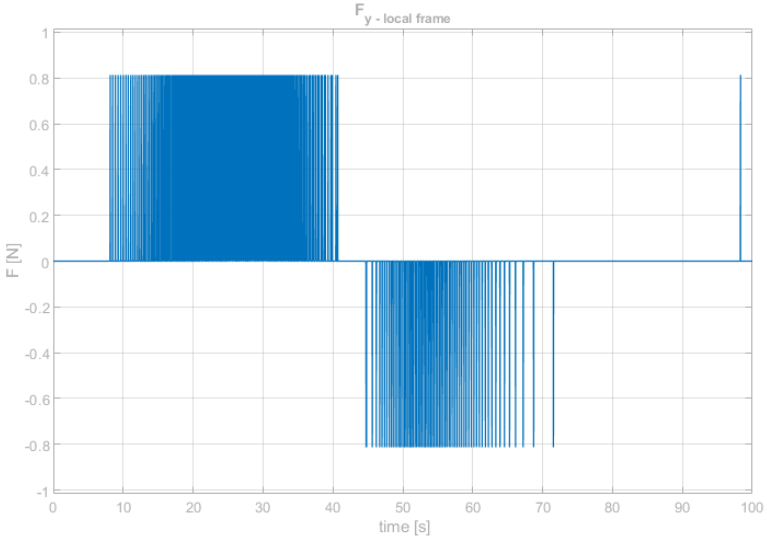


Figure 4.25: y-Force scopes

4.2.6 Model results with parabolic trajectory

To further validate the model, additional simulations are carried out using a parabolic trajectory, and some of the obtained results are presented below.

The equation of the trajectory 4.2.6 is as follows:

$$y(x) = a_2^2 \cdot \frac{x_1^2}{a_1^2} \tag{4.2.6}$$

and the parameters are listed in the following table 4.3.

a1	0.1
a2	0.03

Table 4.3: Parabolic trajectory parameters

To avoid repetitive comments, only the trajectory comparison 4.26 and the error trends over time are reported (fig.4.27 and fig.4.28). Despite a slight increase in error, still well within acceptable limits, the results confirm that the model, with the same parameters, can be effectively extended to different trajectories.

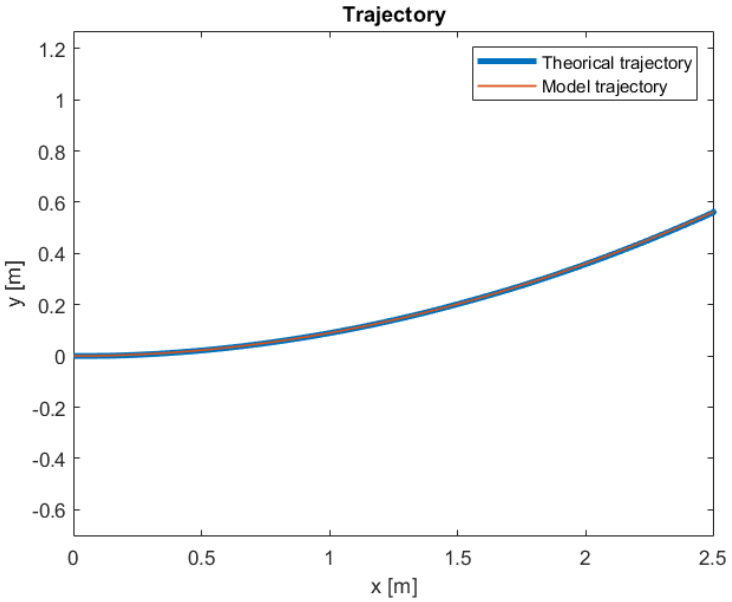


Figure 4.26: Parabolic trajectory

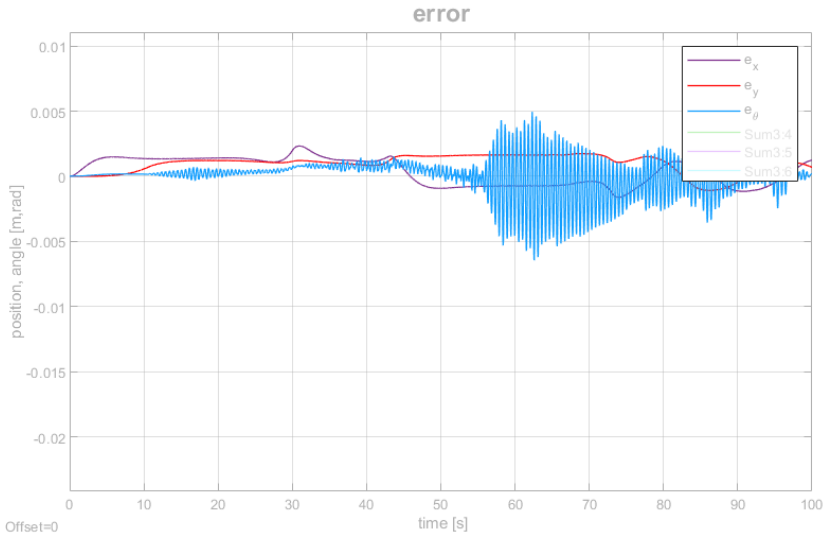


Figure 4.27: Position and orientation error of parabolic trajectory

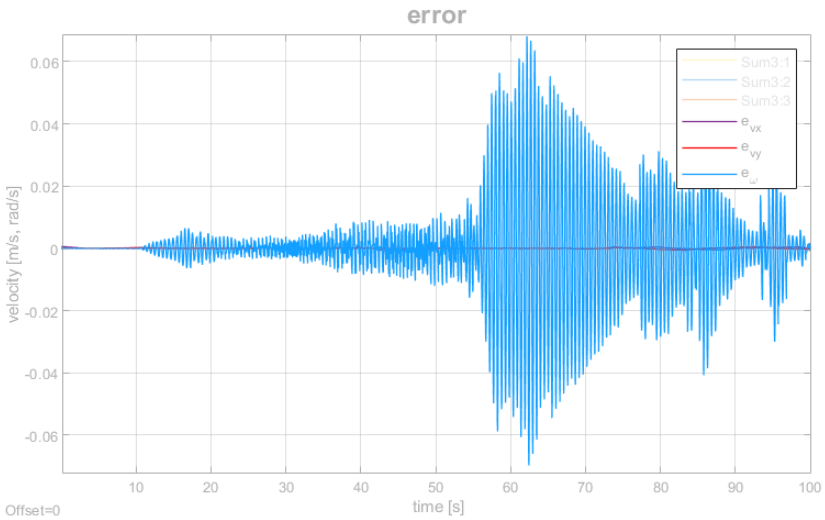


Figure 4.28: Velocity error of parabolic trajectory

Chapter 5

Arduino code

This chapter presents the script developed in the Arduino IDE to implement trajectory control and manage sensors and actuators. The discussion does not focus on the code itself, but rather on the reasoning and design choices underlying its implementation.

5.1 Arduino functions

Before proceeding, it is essential to clearly define the roles assigned to the Arduino. Its primary responsibility is the implementation of the control logic, specifically the instantaneous calculation of the forces and torques necessary for trajectory execution. In addition, the Arduino must communicate with the IMU sensor to acquire data on the robot's position and orientation on the plane. It is also responsible for managing the actuators: performing speed control of the brushless motor and operating the pneumatic valves through PWM signals. Finally, the board handles a set of user interface buttons, allowing basic interaction between the system and the operator.

5.1.1 Control logic

Given that the MATLAB script previously presented (and included in Appendix A) already generates the desired trajectory, it is sufficient to export the reference trajectory data (namely positions and corresponding velocities) into a header file *.h*, allowing them to be read directly by the Arduino script.

A similar approach can be applied to the K gain matrix obtained from the LQI controller. Since this matrix consists of constant values that remain unchanged throughout the simulation, these parameters can be imported into the Arduino code as predefined constants, thereby reducing the computational load on the microcontroller. Therefore, the Arduino is only required to compute the control input vector (denoted as u in Chapter 4) by multiplying the gain matrix K by the error vector, which includes position, velocity, and integral of position errors. The error vector is evaluated at each time step by subtracting the sensor-acquired data from the corresponding reference trajectory values.

5.1.2 Data acquisition

As previously mentioned, the only sensor available on the robot is an IMU, which provides direct measurements of linear accelerations and angular velocities. Consequently, obtaining linear velocities, positions, and orientations requires numerical integration of these signals. This introduces a critical challenge within the scope of this thesis: despite several attempts, relying solely on an inertial sensor proved highly limiting for accurately determining the robot's planar position. The resulting data often exhibited significant inaccuracies, including cumulative errors that increased over time. Conversely, the orientation estimates obtained from the IMU were found to be reliable, largely due to the implementation of the Mahony filter, which fuses the measurements from the three IMU sensors. For this specific application, however, data from the magnetometer were excluded from the filter, as experimental tests demonstrated that doing so produced more accurate and drift-resistant results under identical conditions, and

also simplified the sensor calibration process, eliminating the need for rotations around its axes. Since an existing open-source library [24] was employed for the implementation, the mathematical formulation of the Mahony filter is not discussed in detail in this thesis.

5.1.3 Actuators control

Thruster control

For the reasons mentioned above, although a control strategy for the thruster valves was theoretically developed and implemented to enable planar translation control, it was not possible to thoroughly test the proposed approach. Nevertheless, the proposed strategy involves evaluating the force components along the x and y directions and, based on their signs, activating the corresponding valves. A suitable PWM signal is then generated, proportional to the ratio between the required force and twice the maximum thrust produced by a single actuator.

Motor control

Regarding the brushless motor control, a more elaborate approach was adopted. The motor driver regulates the motor speed through an analog voltage input, which is emulated by the Arduino using a high-frequency PWM signal. However, the LQI controller provides a torque command as its output. This torque is divided by the wheel's moment of inertia and then numerically integrated to obtain the theoretical motor speed required to follow the desired trajectory. It is not sufficient, however, to simply convert this theoretical speed into an equivalent voltage level. The reason lies in the significant difference between the reaction wheel's inertia and the motor's own rotor inertia. A sudden increase in speed would cause a large current draw, while a sharp deceleration would make the motor act as a generator, feeding a potentially damaging high current back into the battery pack. To prevent such situations, a maximum acceleration limit was introduced, forcing the speed to follow a ramped profile

both during acceleration and deceleration phases, effectively smoothing current spikes. Specifically, the theoretical speed obtained from the numerical integration is compared to the actual motor speed. The difference between the two is then checked against the maximum allowable variation in speed, determined by the imposed acceleration limit. If this difference exceeds the allowed Δv, only the maximum permitted increment (or decrement) is applied to the current motor speed, producing an updated current speed. At each control cycle, this new speed replaces the previous value and is converted into a PWM signal to be sent to the motor driver. For safety reasons, the maximum duty cycle was limited to 50%, corresponding to a motor speed of approximately $1480 rpm$.

5.2 Code structure

The functions described above are integrated within the main Arduino program. Beyond the necessary setup and initialization routines, the overall code can be logically divided into five distinct stages (four sequential and one alternative) that operate cyclically.

At startup, the system enters the "IDLE" state, during which the Arduino confirms that all initializations have been successfully completed. In this state, the simulator remains inactive, waiting for a start signal to begin the trajectory execution. The start command can be issued either by pressing a physical button located on the top of the robot or by typing "start" in the Arduino IDE serial monitor. Once the command is received, the code transitions to the "READY" state, notifying the operator that the input has been correctly acquired. A message appears on the serial monitor indicating that the trajectory execution is about to begin, followed by a 3-second countdown. At the end of the countdown, the program enters the "RUNNING" state. In this phase, the actuators (valves and motor) are activated, and the trajectory control is executed according to the functions previously described. When the simulation time expires and the trajectory is completed, the system transitions to the "COMPLETE" state. Here, all actuators are reset, and a confirmation message is displayed on the serial

monitor to inform the operator that the trajectory has been successfully executed. The code then automatically returns to the "IDLE" state, ready to receive a new command. At any moment, the operator can interrupt the execution by triggering the "EMERGENCY" state, an alternative mode designed for safety. This can be activated by pressing a second button located on the robot's top panel or by typing "stop" in the serial monitor. Upon activation, all actuators are immediately disabled, and the system halts. To resume normal operation, a manual reset of the Arduino is required, after which the system returns to the "IDLE" state. The five stages described above are summarized in the following figure 5.1.

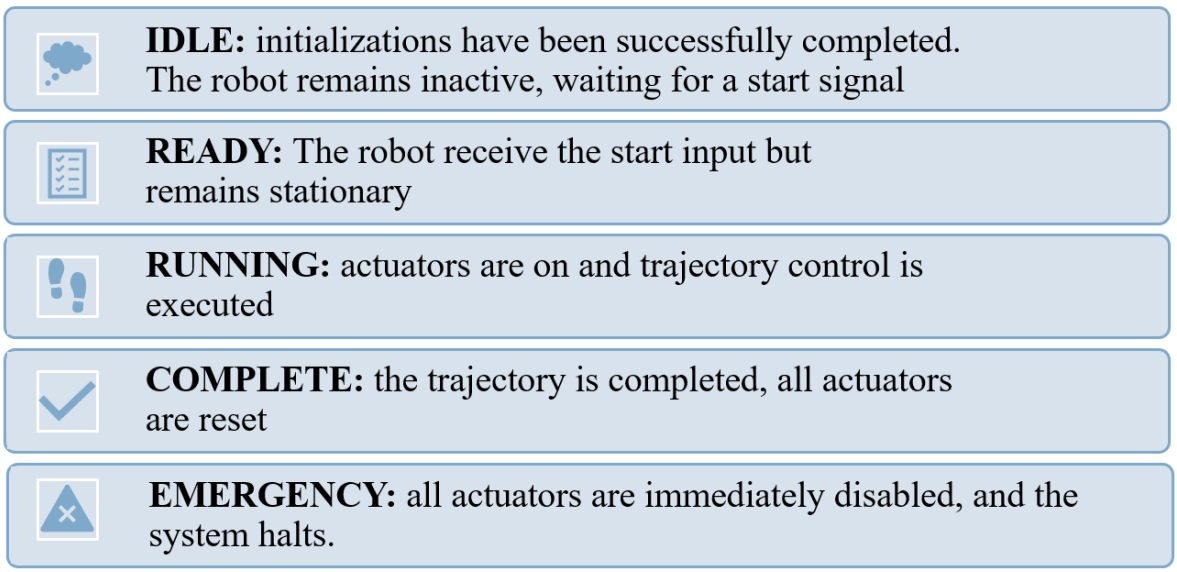


Figure 5.1: Five code states

Chapter 6

Design of the Control board

This chapter presents the control board assembled for the project, providing justification for the choices made regarding the selected components and their placement. For greater clarity, the description of the control board is divided into four functional blocks, after which the overall electrical circuit will be presented showing all the components mounted on the board.

6.1 Control board

The control board, located at the bottom of the robot, was designed and developed to be positioned as close as possible to the robot's geometric center, taking advantage of the space left free by the aluminium profiles that support the motor mounts,as previously suggested in Chapter 3. Consequently, as shown in the figure 6.1, the central area of the board is occupied by the IMU sensor.

An important aspect considered during the design phase was the modularity of the board. As can be clearly seen in the figure 6.1, the board features several connectors and terminal blocks that allow the connection of external components which cannot be mounted directly on it, such as buttons, the battery pack, valves, and the motor. This modular design greatly simplifies maintenance and troubleshooting, as the board can

be easily disconnected from the rest of the system when required.

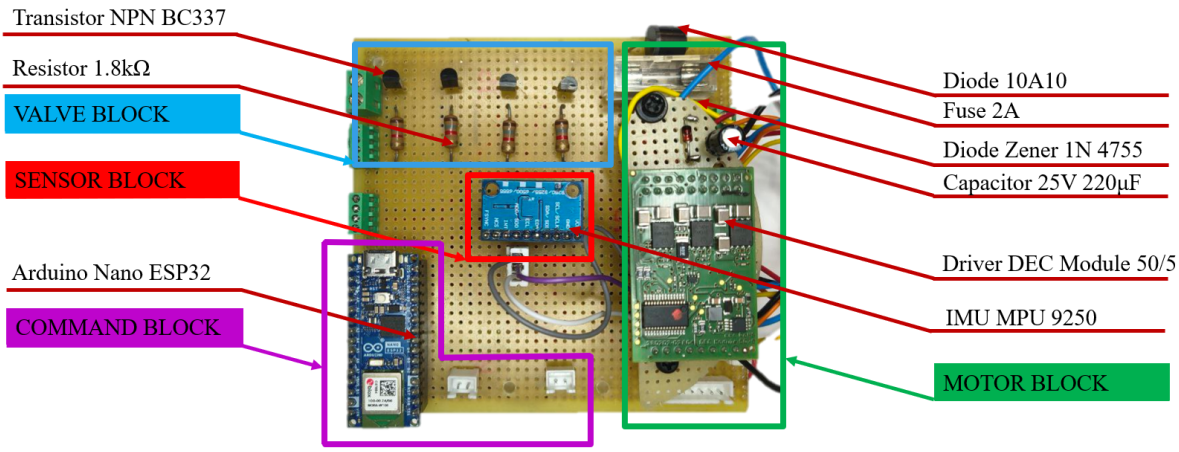


Figure 6.1: Control board

6.1.1 Command block

The command block consists of the Arduino board and the connectors for the "start" and "emergency" buttons mentioned in Chapter 5. The Arduino is positioned in one corner of the board to minimize space occupation, with its USB port oriented toward the inner side of the control board to facilitate convenient connection of programming and power cables.

Since the Arduino was the first component to be positioned, its location effectively determined the layout of all the remaining elements on the board. This placement served as a reference point for arranging the connectors, wiring paths, and other modules, ensuring both functional accessibility and an efficient use of the available surface area.

6.1.2 Sensor block

As previously mentioned, the IMU sensor (currently the only sensor mounted on the robot) is positioned at the centre of the control board, a choice made to ensure greater accuracy in data acquisition. The connection between the sensor and the Arduino is

straightforward. In addition to the $3.3V$ power supply, communication is established via the I^2C interface, a two-wire communication protocol. This interface uses two pins: *SDA*, which handles the data transmission between the sensor and the Arduino, and *SCL*, which provides the clock signal, regulating the timing and synchronization of the data transfer.

6.1.3 Valve block

This section of the board performs a crucial function for the operation of the valves, converting the PWM signal output from the Arduino pins (limited to $3.3V$) into a signal proportional to the $24V$ required to drive the valves. This voltage adjustment is achieved using four $NPNBC337$ transistors. The transistors are connected as follows: the collector is connected to the load (i.e., the valve), the emitter is tied to the board's common ground, and the base is connected to the Arduino pin providing the PWM signal. To protect the Arduino pin from excessive current, a $1.8k\Omega$ resistor is placed between the base and the pin, limiting the maximum current that can flow, according to the following equation 6.1.1, where $3.3V$ is the output voltage of Arduino, while $0.7V$ is the commutation voltage of the transistor.

$$I = \frac{3.3V - 0.7V}{1800\Omega} = 1.4mA \tag{6.1.1}$$

Additionally, as shown on the left side of the figure 6.1, three terminal blocks are present: the largest is used to connect the $24V$ power supply, while the two smaller blocks accommodate the valve power lines. The positive lines are connected in parallel, whereas the negative lines are connected to the transistor collectors, which act as a switching stage to ground. This arrangement ensures that the valves receive the correct voltage and can be safely controlled by the Arduino.

6.1.4 Motor block

This portion of the board is the largest, as it houses the motor driver, the connectors for interfacing the board with the motor, and the driver protection system, as specified

in the driver's data-sheet. The connections between the driver and the motor are not discussed here, as they strictly follow the manufacturer's technical instructions [18]. Greater emphasis, however, is placed on the driver protection circuit, which has been slightly modified compared to the recommendations in the manual. The following figure 6.2 shows the protection circuit suggested in the data-sheet, which calls for a $7A$ fast-acting fuse, a $54V$ TVS diode, and a $220\mu F$ capacitor rated at $63V$.

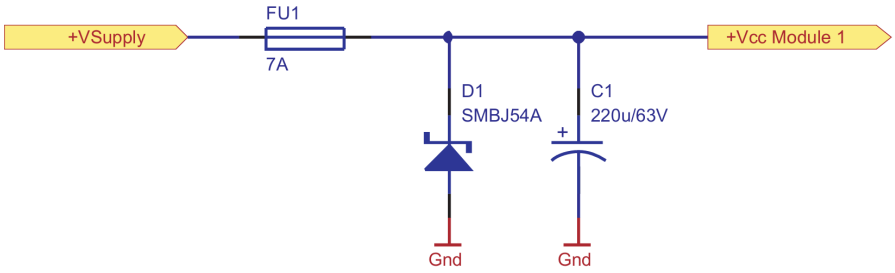


Figure 6.2: Protection circuit of data-sheet

In the present application, the motor is powered at $12V$, well below the driver's maximum allowable voltage of $50V$, and the motor has a nominal current of $2.02A$. Therefore, the protection circuit components used were adapted accordingly: a $2A$ fuse, a $43V$ 1N4755 Zener diode, and a $220\mu F$ capacitor rated at $25V$.

This protection circuit, however, safeguards only the motor driver and not the power source. To prevent potential damage to the batteries, in addition to the software-based limitations previously described in the code section, a $10A10$ diode was added in series with the fuse. This diode prevents any current generated by the motor during unintended braking from flowing back to the battery pack. The use of a power resistor was also considered to aid in energy dissipation. However, thanks to the implementation of the speed ramp, the motor's regenerative effect is significantly mitigated, making the diode sufficient for the current setup. In future iterations, a more advanced circuit could be designed to exploit regenerative braking, allowing the energy produced during deceleration to be recovered and used to recharge the batteries.

6.2 Electrical circuit

To summarize the observations made above and to clarify the various connections implemented, the following figure 6.3 shows the electrical schematic of the circuit assembled on the perfboard. The diagram highlights the main components along with their respective connections. The batteries are represented as two ideal $12V$ voltage sources.

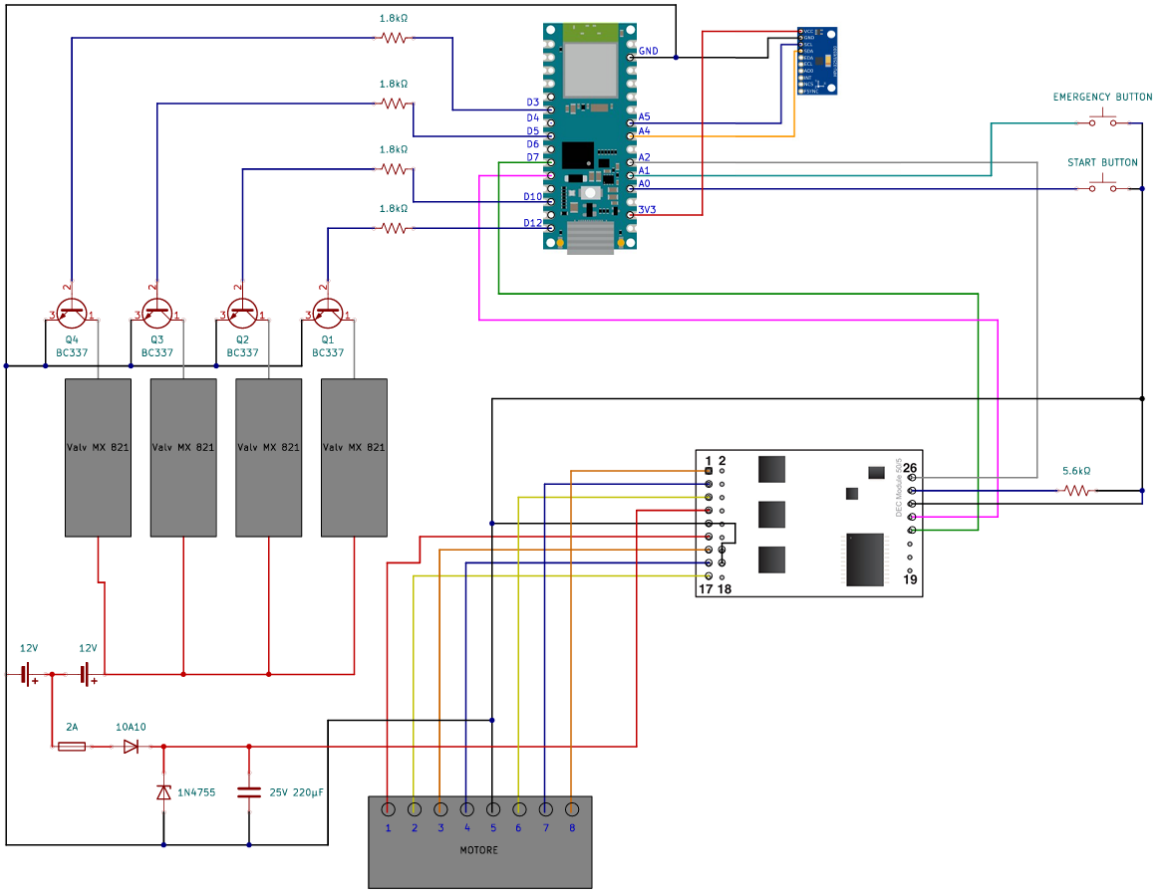


Figure 6.3: Electrical circuit

Chapter 7

Experimental results

The final chapter of this thesis is devoted to the presentation of the experimental results, with the aim of validating the design choices made throughout the development process and confirming the robustness of the implemented control strategy. Before proceeding, a few considerations must be introduced.

As previously discussed, a major limitation of the simulator lies in the use of the IMU as the sole data-acquisition sensor. While it provides quite reliable measurements of the orientation, its estimates of the position are highly inaccurate and therefore unusable. In addition to this, the resin surface originally intended for testing the robot exhibited unwanted slopes which, combined with the absence of the valves and thrusters during the experiments, ultimately made it unsuitable for proper testing. Consequently, the tests (whose results are presented in this chapter) are performed on a smooth tile surface, ensuring correct operation of the air bearings.

7.1 Test bench setup

To use the tile as a flat reference surface, a small rectangular platform is set up on a laboratory desk. It is assembled from four aluminium profiles and equipped with four screws acting as adjustable feet, allowing the height of the base to be finely tuned. Two

pictures of the of the platform 7.1a and the complete setup 7.1b are provided below.

(a) rectangular platform setup

(b) Tile setup

Figure 7.1: Test bench setup

Unfortunately, due to either a manufacturing defect or prolonged improper storage, the tile itself is not perfectly flat but slightly warped in its central region. Although the irregularity is not excessive and is barely noticeable to the naked eye, as will become clearer in the following sections, the tests carried out are significantly affected by this imperfection.

7.2 Experimental tests

The experimental tests conducted fall into two categories: in the first, a predefined rotation is commanded and the resulting response of the robot is observed; in the second, a zero-orientation hold command is applied, and the robot's reaction to an external disturbance is evaluated by manually displacing it by a given angle. This section presents the two tests and their corresponding results, addressing first the trial with a predefined rotation and then the test involving an external disturbance. The control parameters used, namely the maximum errors and maximum forces included in the Q and R matrices, which are valid for both tests, are reported in the following table

7.1.

ε_x	0.04
ε_y	0.04
ε_θ	1
ε_{vx}	0.004
ε_{vy}	0.004
ε_ω	0.1
ε_{int_x}	0.04
ε_{inte_y}	0.04
ε_{int_θ}	100
F_{max}	0.16
F_{max}	0.16
C_{max}	0.15

Table 7.1: LQI parameters of experimental tests

7.2.1 Test 1: predefined rotation

The objective of this test is to verify whether the robot is capable of performing a 30° rotation when a corresponding command, defined by a trapezoidal velocity profile, is provided. The first step consists in extending the MATLAB script presented in Appendix A to generate both the rotational trajectory and the associated trapezoidal velocity profile, since the previously developed functions are not suitable for trajectories that do not involve variations in the x and y coordinates. After specifying the desired maximum angular velocity and maximum angular acceleration, the following profiles 7.2 are obtained.

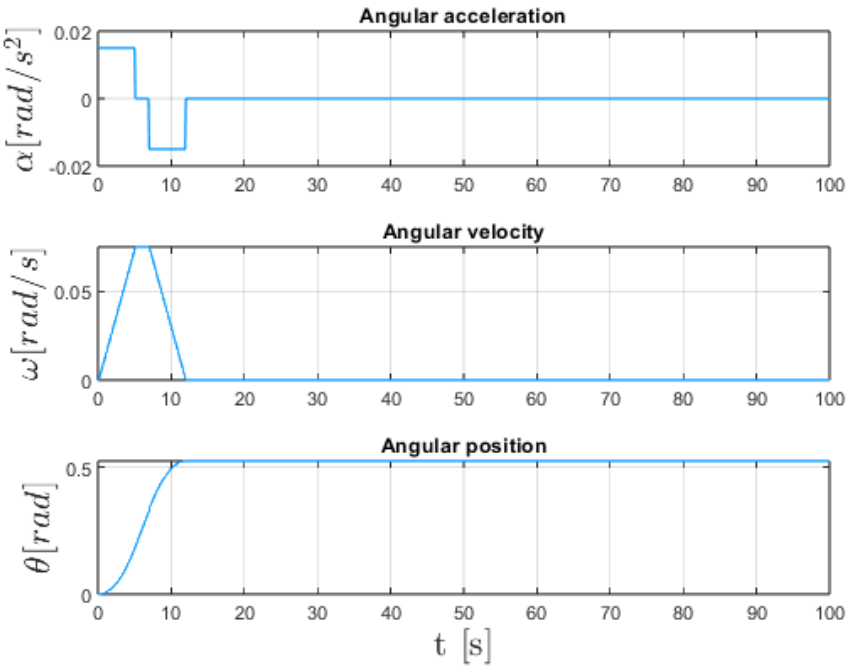


Figure 7.2: Angular acceleration, velocity and position profile

The maximum angular acceleration ($0.015 rad/s^2$) required is computed by enforcing the dynamic equilibrium between the robot and the reaction wheel. Assuming a maximum motor angular acceleration of $10 rad/s^2$, as discussed in the previous chapter, while the moment of inertia of the reaction wheel is $0.6204e-3 kgm^2$ and that of the robot is $0.2 kgm^2$, the correct value of the robot's angular acceleration is obtained from the following equation 7.2.1:

$$\alpha_a = \frac{I_{rw}}{I_a} \cdot \alpha_{rw} = 0.031 rad/s^2 \tag{7.2.1}$$

As a precaution, the calculated value has been decreased by 50%. The maximum velocity of $0.075 rad/s$has been chosen so that the robot requires 5 seconds to reach it under the imposed acceleration limit, in accordance with the laws of uniformly accelerated motion7.2.2.

$$t_{ramp} = \frac{v_{max}}{a_{max}} \tag{7.2.2}$$

The results are now presented, comparing the theoretical command set, the outcomes of the model simulation, and the results obtained from the experimental test.

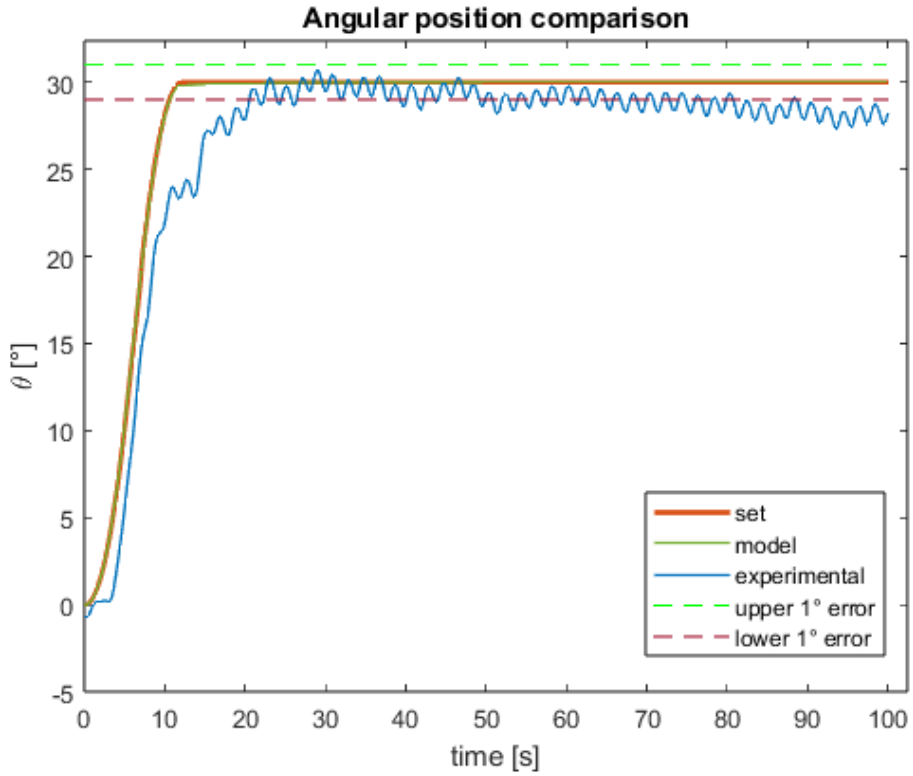


Figure 7.3: Angular position comparison

The comparison of the angular positions (fig.7.3) confirms the robustness of the developed simulator. The trajectory followed by the model perfectly matches the commanded path, making them practically indistinguishable. The experimental test results, despite all the issues previously mentioned, are still valid, with an initial maximum error of approximately 1°, according to the maximum 1° error declared in the parameters of the LQI control, which increases up to approximately 2° over the course of the test.

A similar consideration applies to the velocity curves. With the exception of an outlier measured by the IMU at the beginning of the simulation, the experimental test results are otherwise consistent and confirm the validity of the approach. Below,

both the graph comparing the angular velocities and a zoomed-in view of the curves, excluding the outlier, are presented for greater clarity.

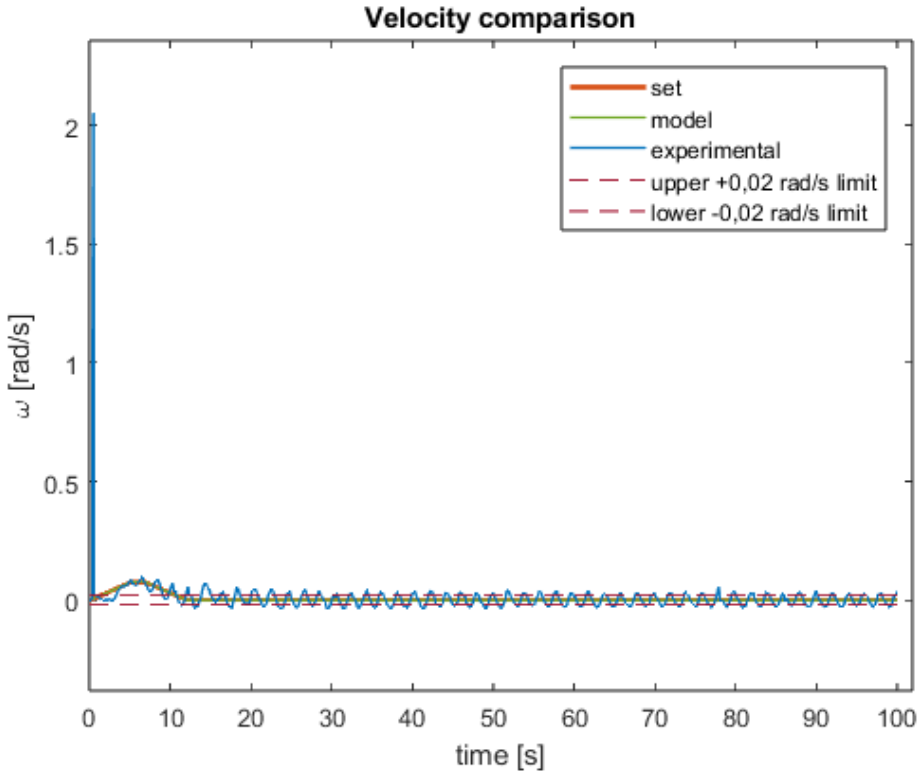


Figure 7.4: Angular velocity comparison

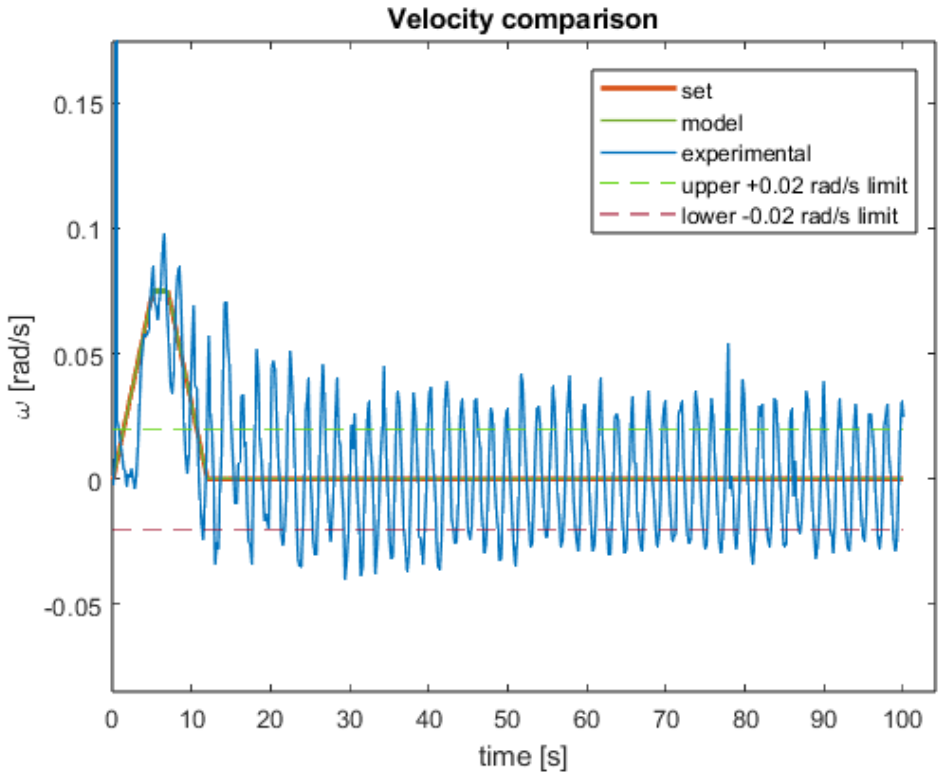


Figure 7.5: Angular velocity comparison zoomed

It is clear that in both graphs, of position and velocity, although the experimental data generally follow the theoretical profile, the measured signal appears noisy. Is more evident that when the commanded velocity is zero, the experimental signal tends to oscillate within a range of approximately $0.02 rad/s$. This oscillation consequently induces the fluctuations observed in the angular position graph as well. The reasons behind this phenomenon are manifold: the IMU 's measurement inaccuracies and the slight inclination of the test surface certainly contribute to the disturbance. However, another important factor is the behaviour of the motor driver. For command voltages below $0.1V$, the driver forces the motor to operate at its minimum speed of $62 rpm$ (approximately $6.5 rad/s$). During the braking phase, the motor reaches this minimum speed and consequently remains at that value. Conservation of angular momentum

7.2.3 requires that the robot's angular velocity also becomes constant, settling around a fixed value.

$$I_a \cdot \Delta\omega_a + I_{rw} \cdot \Delta\omega_{rw} = 0 \tag{7.2.3}$$

Nonetheless, the controller detects an increasing velocity error and responds by increasing the torque command so that the motor rotates in the opposite direction. As soon as the command voltage exceeds the $0.1V$ threshold, the motor is able to influence the robot's motion again through a variation in speed, until it once more drops back to the minimum of $6.5rad/s$, thus creating an infinite loop. A simplified way to estimate the corresponding angular velocity of the robot when the reaction wheel reaches the threshold value of $6.5rad/s$ is to consider two instants in time, t_1 and t_2. At t_1, both the motor and the robot have zero angular velocity, while at t_2 the motor rotates at the critical speed, and the robot's angular velocity is unknown. By applying the angular momentum conservation equation introduced earlier 7.2.3, the robot's angular velocity is found to be approximately $0.02rad/s$, which corresponds to the limiting value around which the velocity fluctuations are observed.

7.2.2 Test 2: reaction to an external disturbance

The objective of this test is instead to evaluate the robot's response in the presence of an external disturbance and, consequently, its ability to return to a predefined reference position. As a trajectory, a vector of zeros was assigned for both angular position and angular velocity. Before applying a disturbance in position, consisting of rotating the robot by a certain angle, data were collected in order to analyse the robot's behaviour in the absence of disturbances, around the zero position.

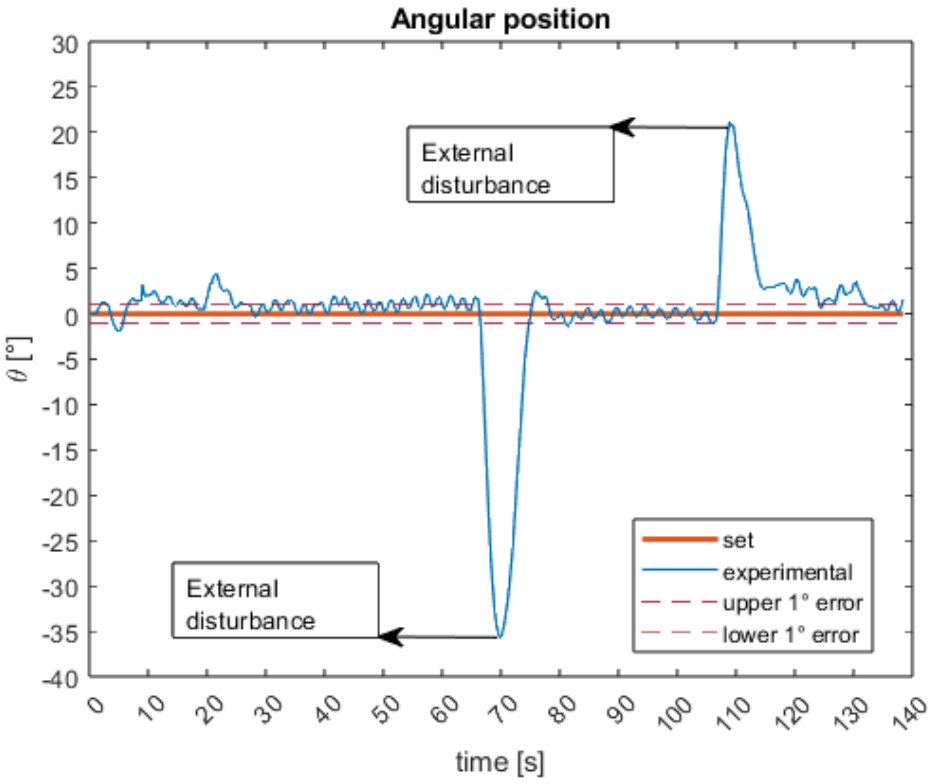


Figure 7.6: Disturbance effect on angular position

Before applying the disturbance, it is observed (fig. 7.6) that despite the command to maintain the zero position, the IMU readings indicate a deviation, occasionally exceeding the maximum error threshold of 1°. This phenomenon is consistent with the behaviour of the motor driver at low speeds, as discussed in the previous section. As illustrated in the graph 7.6, two angular displacements are applied in opposing directions. Following the first disturbance, despite the oscillations, the robot successfully returned to the requested zero position. Conversely, after the second external rotation, the system exhibited greater difficulty in re-establishing the target position. A plausible explanation for this discrepancy lies in the manual application of the disturbance; in the second instance, the robot may have been displaced from its axis, shifting it onto a region of the plane with different surface inclinations.

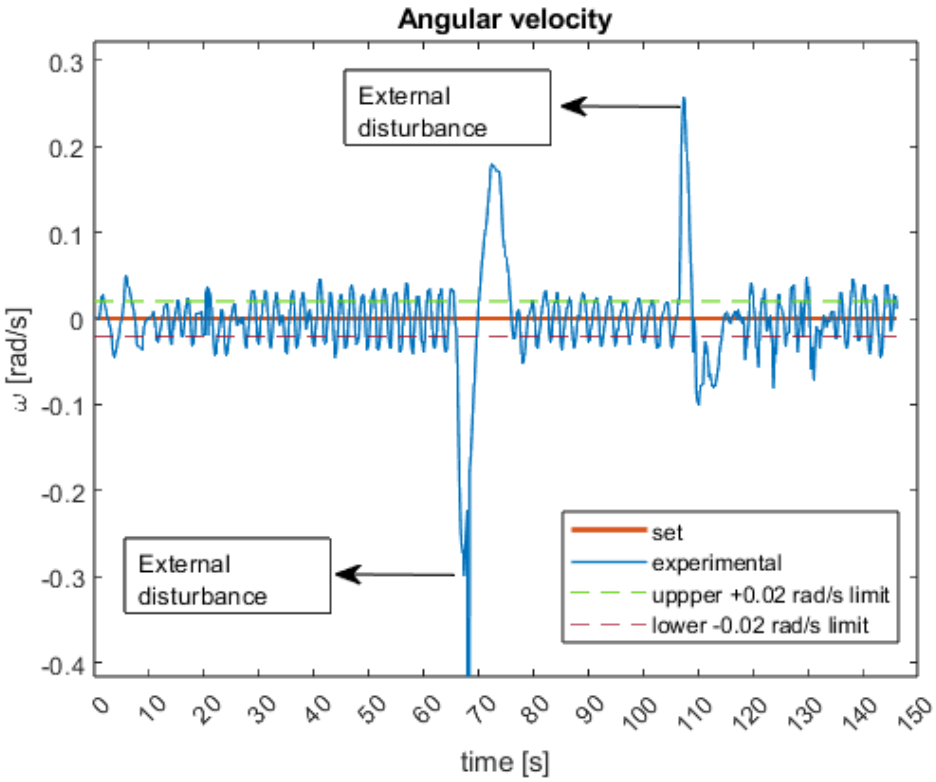


Figure 7.7: Disturbance effect on angular velocity

In the plot 7.7 showing the angular velocity over time, both the instants in which the external disturbances are applied and the previously discussed effect of the motor driver are clearly visible. In fact, apart from the moments when the disturbance is introduced, the rotational velocity remains approximately within the interval of $\pm 0.02 rad/s$. Unfortunately,similar to the previous test, this measurement also contains an outlier in the angular velocity plot 7.7 that appears shortly after the first applied displacement. Since this point is clearly attributable to a sensor reading error, it is excluded from the analysis and will not be further discussed.

From both tests, it is evident that this behaviour of the driver is particularly problematic, as it prevents a proper assessment of the controller's performance. For this reason, a modification was introduced in the Arduino code to enforce a forced shutdown

of the motor. Specifically, an additional *if* condition was implemented: whenever the theoretical torque computed by the LQI controller is less than $0.01Nm$ and the measured velocity error fall below $0.02rad/s$, the driver's *ENABLE pin* is pulled to ground, effectively disabling the motor.

This modification was tested by repeating Test 2, this time applying the external disturbance only once while the robot attempted to maintain its reference orientation.

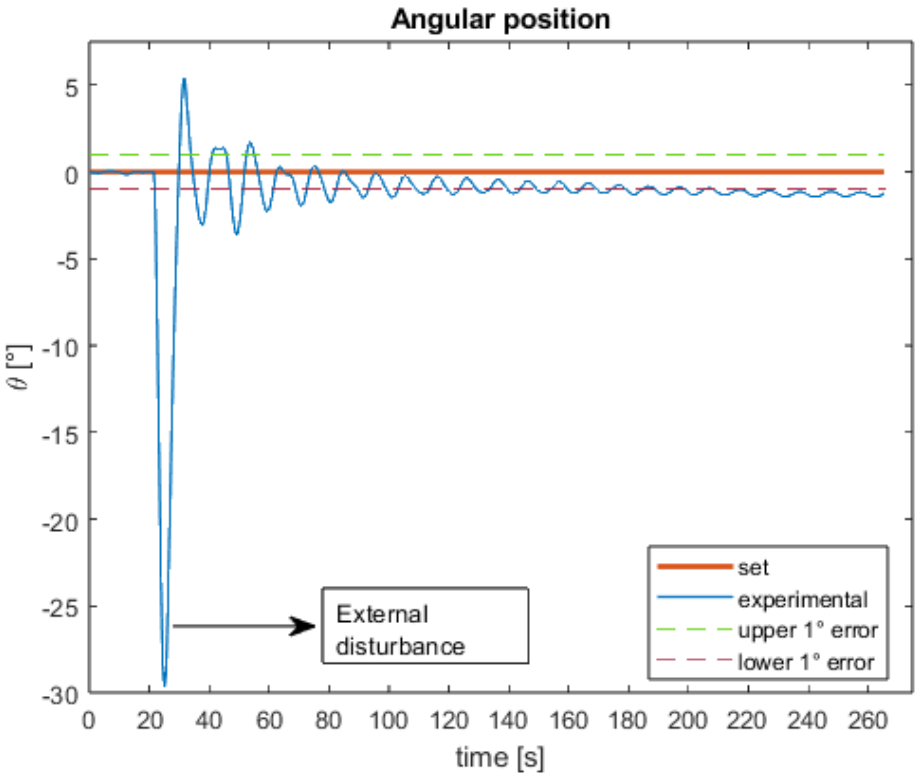


Figure 7.8: Disturbance effect on angular position with modified script

The orientation results show a markedly different behaviour compared to the previous tests. Before applying the disturbance, the robot's orientation remains at zero, as the motor is effectively turned off. Only after the disturbance is applied, the IMU detects an angular displacement, prompting the motor to react and reduce the resulting error. Once the error falls back within the allowed threshold, the motor is disabled

again. As a consequence, the residual oscillations observed in the plot 7.8 can be attributed to small micro-movements of the robot around its equilibrium position, as well as possible inaccuracies from the IMU sensor and the applied filter. The fact that the steady-state error is not exactly zero is expected and can be explained by the chosen control parameters and by the limit introduced in the Arduino code.

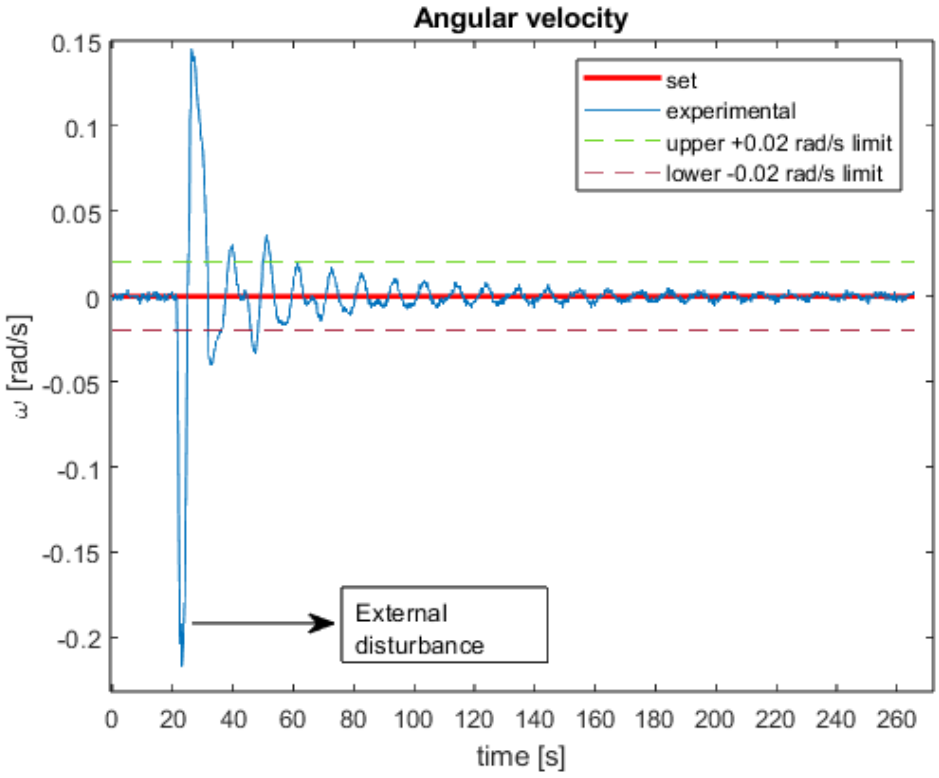


Figure 7.9: Disturbance effect on angular velocity with modified script

Similar observations can be made from the angular velocity plot. The direct readings from the IMU gyroscope confirm that, both at the beginning of the test and at steady state, the velocity, disregarding measurement noise, is effectively zero. Consequently, the oscillations observed following the disturbance are solely due to the undesired effect of gravity, which causes the robot to oscillate around a stable equilibrium position.

As previously mentioned, the implementation of the *if* condition in the Arduino

code required two checks: one on the control torque and the other on the velocity error. The limitation on the control torque excludes from the control action all those values that would require very low command voltages. This prevents the motor from operating in its non-linear low-voltage region. However, since the computation of the control torque depends on both the position error and the velocity error, and the magnetometer was excluded from the attitude estimation, the angles are obtained solely through numerical integration, which may not be highly accurate over long measurement periods. As a result, small errors can accumulate over time, potentially leading to a reported displacement even when no actual movement occurs. This effect is evident in the plot 7.8, where after approximately $200s$ the measured error exceeds the maximum allowable error of $1°$. Consequently, this cumulative error also appears in the torque plot 7.10. Therefore, if an orientation estimation error were to occur and the second constraint on velocity were not in place, the robot would attempt to compensate for a non-existent error, resulting in an unnecessary corrective motion.

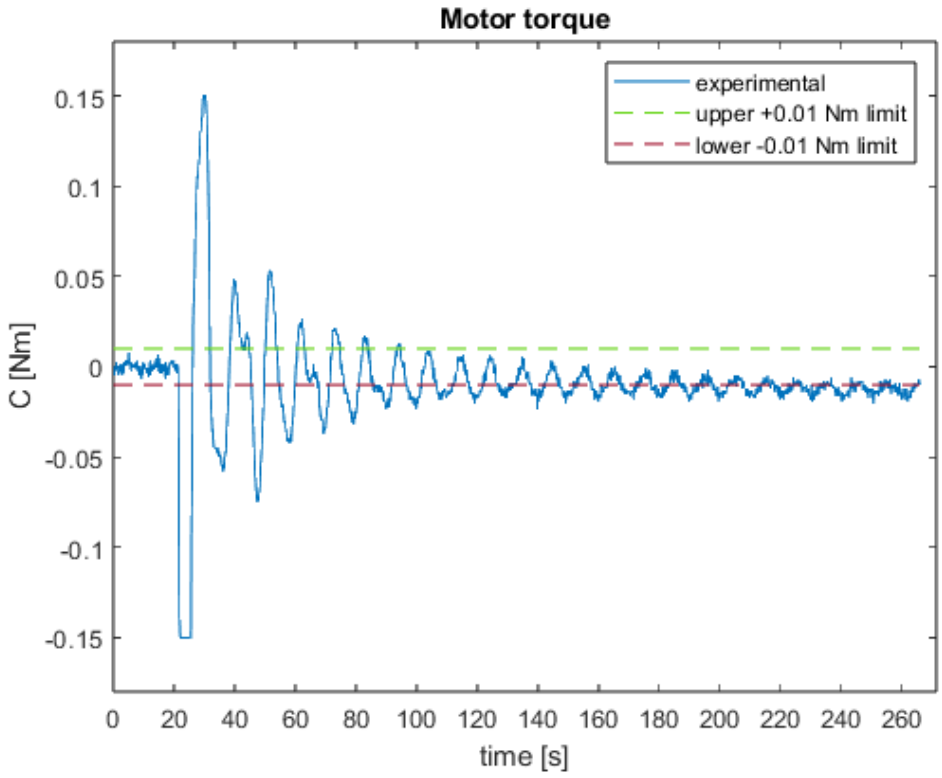


Figure 7.10: Motor control torque

Chapter 8

Conclusion

This final chapter presents the conclusions drawn from the work carried out in this thesis, summarizing the main results and evaluating the effectiveness of the implemented solutions.

The objective of developing a control strategy capable of guiding the robot along a predefined trajectory has been successfully achieved. This control logic was validated both in a virtual environment, through physical model, and in laboratory conditions. In both cases, the results were consistently positive, demonstrating the effectiveness of the proposed approach. This outcome is particularly significant considering the various challenges encountered during the experimental phase, most notably the limitations imposed by the available test surfaces, which made the validation of the simulator more complicated.

At the same time, the project is well-prepared with a view toward future developments. It provides a detailed physical model built in Simscape within the MATLAB/Simulink environment, a reliable MATLAB control script capable of delivering appropriate gains for a wide range of trajectories, and an advanced Arduino IDE command script. The latter has already proven effective in the current setup while still being structured in a way that facilitates modifications and further improvements.

From a practical and constructive standpoint, the simulator also demonstrates ro-

bustness and versatility. Its modular design, the use of slotted aluminum profiles, and the flexibility enabled by 3D printing not only simplified the assembly process but also greatly ease any future adjustments or redesigns. Equally noteworthy is the control board, which is well-organized and carefully structured, ensuring compatibility with additional components should future developments require an expanded setup.

8.1 Future works

This thesis can be considered a solid foundation for a more complex project, where new control strategies can be explored and implemented. At this stage, it is useful to reflect on the aspects that could be further enhanced. In particular, complementing the IMU with additional sensors could improve the accuracy of data acquisition, providing the opportunity to test and refine the control of the valves and thrusters, which in this work were less extensively explored due to practical testing constraints.

Additionally, the physical model developed in Simscape could be further enriched by introducing the possibility of simulating force and torque disturbances. This would allow for the virtual evaluation of scenarios such as undesired inclinations or asymmetries in the thruster jets, providing a more realistic and flexible environment to validate and optimize control strategies before implementation.

Appendix A

Matlab scripts

A.1 Main script

```
m=9.56;
base=0.349; %m
larghezza=0.360;
altezza=0.606;

Iz=1/12*m*(base^2+larghezza^2);

%Valve
P_rid=7; %bar
rapp=0.433; %b critical ratio
Cv=15.7; %Nl/bar conductance
Cv=Cv*1e-3/60; %m^3/s
Pi=1e5;%Pa initial condition
R_gas=287; %J/kg/K
Temp=293; %K
td=6e-3; %m tube diameter
Len=0.3; %m tube lenght
```

```
gam=1.4; %specific heat ratio air
dn=1e-3; %m nozzle diameter
A=pi/4*dn^2; %nozzle
pwmdt=0.01; %s

%motor
kw=374*2*pi/60; %rad/s/V
ke=1/kw; %V/rad/s
kc=25.5e-3; %Nm/A
L=0.56e-3; %H
R=1.2; %ohm
taum=17.1e-3; %s
kw_t= 17.7; %rpm/mNm
kw_t=kw_t*2*pi/60*1000;
kt_w=1/kw_t;
Imotore=92.5*10^-3*10^-4; %kgm^2
Cn_m=55e-3; %Nm
B_att=0;
speedn=2940; %rpm
speedmax=4370*2*pi/60;%rad/s

%reaction wheel
phi_r=100e-3;%m
density=7900;%kg/m3
spessore=8e-3;%m
Iwheel=1/2*(density*(pi*phi_r^2/4*spessore))*phi_r^2/4;

% trajectory
dt=0.1;
T=100;
t = 0:dt:T;  % time
```

```
49 C_max=2*54.8e-3; %Nm
50
51 % examples of trajectory x(σ), y(σ)
52
53 %sinusoide
54     % a1=0.1;
55     % a2=0.5;
56     % omega=0.1;
57     % x_t=@(time) a1*time;
58     % y_t=@(time) a2*cos(omega*time)-a2;
59
60 %parabola
61     % a1=0.1;
62     % a2=0.03;
63     % x_t=@(time) a1*time;
64     % y_t=@(time) a2^2*time.^2;
65
66 %retta
67     % a1=0.1;
68     % x_t=@(time) a1*time;
69     % y_t=@(time) 0*time;
70
71 %zero zero
72     x_t=@(time) 0*time;
73     y_t=@(time) 0*time;
74
75
76 x = x_t(t);  % x position
77 y = y_t(t);  % y position
78
79 % 2. velocity
```

```
80 dx_dt = gradient(x, t);
81 dy_dt = gradient(y, t);
82
83 % 3. abs velocity
84 velocity = sqrt(dx_dt.^2 + dy_dt.^2);
85
86 % 4. time start and time end
87 t_inizio = 0;  % Tempo di inizio
88 t_fine = 25;    % Tempo di fine
89
90 % 5. index of time start and time end
91 [~, idx_inizio] = min(abs(t - t_inizio));  % Indice per il tempo
       di inizio
92 [~, idx_fine] = min(abs(t - t_fine));      % Indice per il tempo
       di fine
93
94 % 6. Trajectory lenght
95 Leng = trapz(t(idx_inizio:idx_fine), velocity(idx_inizio:
      idx_fine));
96
97 % a_max and v_max
98 a_max = 0.002; %m/s2
99 v_max = 0.06; %m/s
100
101 x_sigma=@(s) x_t(s*t_fine);
102 y_sigma=@(s) y_t(s*t_fine);
103
104
105 [sigma, dsigma,t_flat] = Trajectory_form(a_max, v_max,t,Leng);
106
107 xr = x_sigma(sigma);
```

```
108 yr = y_sigma(sigma);
109
110
111 ds=diff(sigma);
112 dx_ds = gradient(xr,sigma);
113 dx_ds(ds==0)=0;
114
115 dy_ds = gradient(yr,sigma);
116 dy_ds(ds==0)=0;
117
118 xp_r=dx_ds.*dsigma(1:end);
119 xp_r(end)=xp_r(end-1);
120
121 yp_r=dy_ds.*dsigma(1:end);
122 yp_r(end)=yp_r(end-1);
123
124 %Filter
125 fc = 1;  % cut frequency [Hz]
126 [b, a] = butter(2, fc/(100/2));
127 xp = filtfilt(b, a, xp_r);
128 yp = filtfilt(b, a, yp_r);
129
130 v_mod = sqrt(xp_r.^2 + yp_r.^2);
131
132 xpp=gradient(xp,t);
133 ypp=gradient(yp,t);
134
135
136 %%automatic theta for different trajectories
137     % theta_in = atan2(yp_r, xp_r);
138     % theta = angular_limitation(t, theta_in, sigma,Leng);
```

```
139     % theta=unwrap(theta);
140     %
141     % w_in = diff([theta(1), theta]) / dt;  % angular velocity
142     % w_ltra = w_limitation(t, w_in, sigma);
143     %
144     % fc = 1;  % frequenza di taglio [Hz]
145     % [b, a] = butter(2, fc/(100/2));
146     % w_f = filtfilt(b, a, w_l);
147     % w=[w_l(1:9),w_f(10:end)];
148     %
149     % thetapp=gradient(w,t);
150
151 % %theta and w if x and y are 0
152  dtramp = 0.1;
153 pos_finale = pi/6;        % Final position
154 alpha_max = 0.015;
155 omega_max = alpha_max*5;
156
157 % 2. Time for trapezoidal w profile
158 t_ramp = omega_max / alpha_max;
159 pos_ramp = 2 * (0.5 * alpha_max * t_ramp^2);
160 pos_cruise = pos_finale - pos_ramp;
161 t_cruise = pos_cruise / omega_max;
162
163 % Total time
164 t_movimento = 2 * t_ramp + t_cruise;
165
166 % 3. time vector
167 tramp1 = 0:dtramp:t_ramp;
168 tramp2 = (t_ramp + dtramp):dtramp:(t_ramp + t_cruise);
169 tramp3 = (t_ramp + t_cruise + dtramp):dtramp:t_movimento;
```

```
170
171 % stop time
172
173 t_stazionamento = (t_movimento + dtramp):dtramp:T;
174
175 % 4. Trapezoidal w profile
176 w_rampa_up = alpha_max * tramp1;
177 w_cruise = omega_max * ones(size(tramp2));
178
179 % final delta t
180 dt3 = tramp3 - (t_ramp + t_cruise);
181 w_rampa_down = omega_max - alpha_max * dt3;
182 w_rampa_down(w_rampa_down < 0) = 0;
183
184 % keep w 0 at the end
185 w_zero = zeros(size(t_stazionamento));
186 % Vettore w completo
187 w = [0,w_rampa_up, w_cruise, w_rampa_down, w_zero];
188
189 % 5. theta
190 % Acceleration
191 theta_rampa_up = 0.5 * alpha_max * tramp1.^2;
192
193 % Constant w
194 pos_fine_rampa1 = 0.5 * alpha_max * t_ramp^2;
195 theta_cruise = pos_fine_rampa1 + omega_max * (tramp2 - t_ramp);
196
197 % Deceleration
198 pos_inizio_rampa_down = pos_fine_rampa1 + omega_max * t_cruise;
199 theta_rampa_down = pos_inizio_rampa_down + omega_max * dt3 - 0.5
    * alpha_max * dt3.^2;
```

```
200
201 % end of deceleration
202 theta_rampa_down = min(theta_rampa_down, pos_finale);
203
204 % keep last theta
205 theta_stazionamento = pos_finale * ones(size(t_stazionamento));
206
207 % final vector
208 theta = [0,theta_rampa_up, theta_cruise, theta_rampa_down,
       theta_stazionamento];
209
210 alphatot=[alpha_max*ones(size(tramp1)),zeros(size(tramp2)),-
       alpha_max*ones(size(tramp3)),zeros(size(t_stazionamento)),0];
211
212 %%NO automatic orientation, keep theta 0
213 % theta=zeros(size(t));
214 % w=diff([theta(1), theta]) / dt;
215
216
217 %LQI
218 x_ref = [xr;
219          yr;
220          theta;
221         xp;
222         yp;
223          w]';
224 e_ref=[xr;
225          yr;
226          theta];
227
228 xref=timeseries(x_ref,t);
```

```
229 eref=timeseries(e_ref,t);
230
231 Au = [0 0 0 1 0 0
232        0 0 0 0 1 0
233        0 0 0 0 0 1
234       0 0 0 0 0 0
235       0 0 0 0 0 0
236       0 0 0 0 0 0];
237
238 Bu = [0 0 0
239       0 0 0
240       0 0 0
241       1/m 0 0
242       0 1/m 0
243       0 0 1/Iz];
244
245 Cu=[eye(3,3) zeros(3)];
246
247 n=size(Au,1);
248 m=size(Bu,2);
249 p=size(Cu,1);
250
251 A_ext=[Au zeros(n,p);
252       -Cu zeros(p,p)];
253
254 B_ext=[Bu;
255     zeros(p,m)];
256
257 %%Optimal parameters for x y trajectories
258  Qu=diag([500 500 1 50000 50000 5000]);
259  Quu=diag([500 500 0.0000001]);
```

```
260 %Optimal parameters for x y zero
261  % Qu=diag([1 1 1 5 5 10]);
262  % Quu=diag([1 1 0.0000001]);
263
264 Q_ext=blkdiag(Qu,Quu);
265
266 Ru=diag([1/1.6^2 1/1.6^2 1/(C_max)^2]);
267
268 K = lqr(A_ext, B_ext, Q_ext, Ru);
269
270 K1 = K(:, 1:n);       % state gain
271 K2 = 1/10000*K(:, n+1:end);  % integral gain
```

A.2 Trajectory form

```
1     function [sigma, dsigma,t_flat] = Trajectory_form(a_max,
         v_max,t,Leng)
2
3     sigma = zeros(size(t));
4     dsigma = zeros(size(t));
5
6     % Calcola tempo di accelerazione
7     t_ramp = v_max / a_max;
8     L_tri =0.5* a_max * t_ramp^2;  % distanza coperta se profilo
          triangolare
9
10    if 2*L_tri >= Leng
11        % PROFILO TRIANGOLARE
12        t_flat=0;
13        t_ramp = sqrt(Leng / a_max); %0.5*L/2/a
14        for i = 1:length(t)
```

```
15              ti = t(i);
16              if ti < t_ramp
17                  sigma(i) = 0.5 * a_max/Leng * ti^2;
18                  dsigma(i) = a_max/Leng * ti;
19              elseif ti < 2*t_ramp
20                  td = 2*t_ramp - ti;
21                  sigma(i) = 1 - 0.5 * a_max/Leng * td^2;
22                  dsigma(i) = a_max/Leng * td;
23              else
24                  sigma(i)=1;
25                  dsigma(i)=0;
26              end
27          end
28      else
29          % PROFILO TRAPEZOIDALE
30          t_flat = (Leng - a_max * t_ramp^2) / v_max;  % tempo a v
                costante (2*0.5*a_max)
31          for i = 1:length(t)
32              ti = t(i);
33              if ti < t_ramp
34                  sigma(i) = 0.5 * a_max/Leng * ti^2;
35                  dsigma(i) = a_max/Leng * ti;
36              elseif ti < t_ramp + t_flat && ti>= t_ramp
37                  sigma(i) = 0.5 * a_max/Leng * t_ramp^2 + v_max/
                       Leng * (ti - t_ramp);
38                  dsigma(i) = v_max/Leng;
39              elseif ti< 2*t_ramp+t_flat && ti>= t_ramp+t_flat
40                  td = 2*t_ramp+t_flat - ti;
41                  sigma(i) = 1 - 0.5 * a_max/Leng * td^2;
42                  dsigma(i) = a_max/Leng * td;
43              else
```

```
                sigma(i)=1;
                dsigma(i)=0;
            end
        end
    end

    % Normalizza sigma tra [0, 1]
    sigma = sigma / max(sigma);
    % dsigma=dsigma/max(dsigma);
end
```

A.3 Angular limitation

```
function theta = angular_limitation(t, theta_in, sigma,Leng)

    % Inizializzazione
    theta = theta_in;

    % Trova il primo indice in cui s >= 1
    idx_stop = find(sigma >= 1, 1, 'first');

    if ~isempty(idx_stop)
        % Prendi il valore dell'ultimo theta prima di sigma = 1
        if idx_stop == 1
            theta_locked = theta_in(1); % Tutto sigma >= 1,
               blocca subito
        else
            theta_locked = theta_in(idx_stop - 1);
        end

        % Blocca i valori successivi
```

```
18          theta(idx_stop:end) = theta_locked;
19      end
20  end
```

A.4 w limitation

```
1   function w = w_limitation(t, w_in, sigma)
2
3       % Inizializzazione
4       w = w_in;
5
6       % Trova il primo indice in cui s >= 1
7       idx_stop = find(sigma >= 1, 1, 'first');
8
9       if ~isempty(idx_stop)
10          % Prendi il valore dell'ultimo theta prima di sigma = 1
11          if idx_stop == 1
12              w_locked = w_in(1); % Tutto sigma >= 1, blocca
                    subito
13          else
14              w_locked = 0;
15          end
16
17          % Blocca i valori successivi
18          w(idx_stop:end) = w_locked;
19      end
20  end
```

Bibliography

[1] Marcello Romano Markus Wilde, Casey Clark. Historical survey of kinematic and dynamic spacecraft simulators for laboratory experimentation of on-orbit proximity maneuvers. *Progress in Aerospace Sciences*, 110:100552, 2019.

[2] Josef Kulke. Design of a simplified floating spacecraft simulator. Master's thesis, Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg, August 2022.

[3] Francesco Bologna. Design, integration and testing of a small floating spacecraft simulator. Master's thesis, Politecnico di Torino and Naval Postgraduate School, July 2023.

[4] Richard Zappulla II, Josep Virgili-Llop, Costantinos Zagaris, Hyeongjun Park, and Marcello Romano. Dynamic air-bearing hardware-in-the-loop testbed to experimentally evaluate autonomous spacecraft proximity maneuvers. *JOURNAL OF SPACECRAFTAND ROCKETS*, 54(4), 2017.

[5] David A. Friedman, Marcello Romano, and Tracy J. Shay. Laboratory experimentation of autonomous spacecraft approach and docking to a collaborative target. *Journal of Spacecraft and Rockets*, 44(1):164–173, 2007.

[6] Jason S. Hall and Marcello Romano. Novel robotic spacecraft simulator with mini-control moment gyroscopes and rotating thrusters. In *2007 IEEE/ASME international conference on advanced intelligent mechatronics*, pages 1–6, 2007.

[7] Jason S. Hall and Marcello Romano. Laboratory experimentation of guidance and control of spacecraft during on-orbit proximity maneuvers. In Annalisa Milella Donato Di Paola and Grazia Cicirelli, editors, *Mechatronic Systems*, chapter 11. IntechOpen, Rijeka, 2010.

[8] Josep Virgili-Llop, Jerry V. Drew II, and Marcello Romano. Design and parameter identification by laboratory experiments of a prototype modular robotic arm for orbiting spacecraft applications. *6th International Conference on Astrodynamics Tools and Techniques (ICATT),*, Darmstadt, Germany, March 2016.

[9] Marco Sabatini, Giovanni B. Palmerini, Riccardo Monti, and Paolo Gasbarri. Image based control of the "pinocchio" experimental free flying platform. *Acta Astronautica*, 94(1):480–492, 2014.

[10] Marco Sabatini, Marco Farnocchia, and Giovanni B. Palmerini. Design and tests of a frictionless 2d platform for studying space navigation and control subsystems. In *2012 IEEE Aerospace Conference*, pages 1–12, 2012.

[11] Pedro Roque, Sujet Phodapol, Elias Krantz, Jaeyoung Lim, Joris Verhagen, Frank Jiang, David Dorner, Roland Siegwart, Ivan Stenius, Gunnar Tibert, Huina Mao, Jana Tumova, Christer Fuglesang, and Dimos V. Dimarogonas. Towards open-source and modular space systems with atmos, 2025.

[12] item s.r.l. Profilato 6 – 30×24 mm – leggero – naturale. `https://www.item24.com/it-it/profilato-6-30x24-leggero-naturale-60888?length=300&category=tecnica-dei-profilati%2Fprofilati-di-alluminio`. [Online; last consultation 11-04-2025].

[13] item s.r.l. Automatic-fastening set 8 s. `https://www.item24.com/en-it/automatic-fastening-set-8-s-bright-zinc-plated-72462?type=BR8&closedGroove=3&category=profile-technology%2Ffastening-technology`. [Online; last consultation 11-04-2025].

[14] MAGER Air Bearings. Flat air bearings: Hpr, hpc series datasheet, 2021. [last consultation 11-14-2025].

[15] Mx 8212/2 nc 2/2 solenoid valve datasheet. Product Datasheet.

[16] Shenzhen CP-Link Electronic Co., Ltd. Smaco s400 portable scuba user's manual. [last consultation 11-04-2025].

[17] Maxon. Ec 45 flat ϕ45 mm, brushless, 30 w, hall sensor, wired.

[18] maxon motor ag. *1-Q-EC Amplifier DEC Module 50/5 Operating Instructions*, 2015. Order number 380200.

[19] RS PRO. Battery pack (a700000009106018). [last consultation 11-14-2025].

[20] item s.r.l. T-slot nut st. `https://www.item24.com/en-it/t-slot-nut-5-st-m4-bright-zinc-plated-37006?material=verz`. [Online; last consultation 11-04-2025].

[21] NASA Glenn Research Center. Rocket thrust summary. `https://www.grc.nasa.gov/www/k-12/airplane/rktthsum.html`, 2025. [Online; last consultation 08-15-2025].

[22] Riccardo Bevilacqua, Andrew P. Caprari, Jason Hall, and Marcello Romano. Laboratory experimentation of multiple spacecraft autonomousassembly. In *AIAA Guidance, Navigation, and Control Conference*, Chicago, Illinois, 2009.

[23] Josep Virgili-Llop, Jerry V. Drew, Richard Zappulla II, and Marcello Romano. Laboratory experiments of resident space object capture by a spacecraft–manipulator system. *Aerospace Science and Technology*, 71:530–545, 2017.

[24] Bolder Flight Systems,flybrianfly. Invensense imu library.