

POLITECNICO DI TORINO

Master's Degree in Aerospace Engineering



**Politecnico
di Torino**



NAVAL
POSTGRADUATE
SCHOOL

Master's Degree Thesis

Design, integration and testing of a small floating spacecraft simulator

Supervisors

Prof. Marcello ROMANO

Prof. Jennifer HUDSON

Candidate

Francesco BOLOGNA

July 2023

Summary

Spacecraft Simulators allow to emulate the dynamics and/or kinematics of satellites on the ground, and play a key role in ground testing. In particular, 3-degree-of-freedom (3-DOF) Spacecraft Kino-Dynamics Simulators with planar air bearings are used for research and testing in dynamic modeling and guidance, navigation and control (GNC), performance and steering laws of actuators, and contact mechanics. This master's thesis has two main goals. The first is to realize a "miniature" orbital robotics test-bed, in which a newly designed floating spacecraft simulators (FSS) can be tested; the second is to design and test a small floating spacecraft simulator and test a GNC algorithm, which allows the vehicle to perform basic attitude and position maneuvers.

Acknowledgements

This work could not have been done without the help and involvement of a number of people. First of all, I would like to thank Prof. Marcello Romano for allowing me to work on this project. I would also like to thank Prof. Hudson and Prof. Herman for following me during my stay in Monterey. Josef Kulke was also helpful with his advice on the integration of the new vehicle. Finally, thanks to the guys from the ASTRADORS lab at Politecnico di Torino for their advice, especially Dr. Erica Scantamburlo for following me through the final stages of my project.

Table of Contents

List of Tables	VII
List of Figures	VIII
Acronyms	XI
1 Introduction	1
1.1 Objectives	1
1.2 Structure	1
2 Spacecraft Simulators	3
2.1 Introduction	3
2.2 Types and requirements	4
2.3 POSEIDYN testbed	5
2.4 Granite floor	7
3 Floating Spacecraft Simulator	13
3.1 General requirements	13
3.2 MyDas2 Vehicle	14
3.2.1 Structure	15
3.2.2 Pneumatic System	18
3.2.3 Propulsion and flotation	20
3.2.4 Electronic System	21
3.2.5 Sensors	23
4 Digital Twin	28
4.1 Equations of motion	29
4.2 Direction Cosine Matrix	30
4.3 Matlab implementation	31
4.4 Simulink implementation	33
4.5 Graphics animation	36

5	GNC algorithm implementation	38
5.1	Attitude and Position Control System	40
5.2	Simulink Integration	41
5.2.1	Support Package	41
5.2.2	Guidance	44
5.2.3	Importing sensor data	46
5.2.4	PD controller	54
5.2.5	Steering Law	56
5.2.6	Thrust Modulation	58
5.2.7	Link with On-board Computer	60
6	Hardware-in-the-loop experiments	63
6.1	Static experiments	63
6.1.1	Floating without thruster	63
6.1.2	Thrusters opened without floating	64
6.2	Attitude maneuvers	65
6.3	Position Maneuvers	69
7	Conclusion	74
7.1	Future Works	75
A	Appendix	76
A.1	Listings	76
A.2	Bill of Materials	84
	Bibliography	86

List of Tables

2.1	Primary and secondary requirements of spacecraft simulators	5
2.2	POSEIDYN requirements and features	7
2.3	DIN 876 grades of flatness	8
2.4	Requirements and constraints in the choice of granite floor	10
3.1	Characteristics of FSS at NPS	15
3.2	Key parameters of MyDas2	16
4.1	Simulation parameters	35
5.1	Accelerometer and gyroscope correction	49
5.2	Response characteristics according to K_P and K_D adjustment	54
6.1	Experimental floating duration from 2800 psi to 0	65
6.2	Experimental thruster firing duration from 2800 psi to 0	65
6.3	Parameters for attitude maneuver	66
6.4	Parameters for position maneuver	70
6.5	Results of position maneuvers	72
A.1	Electronic Parts	84
A.2	Pneumatic Parts	85

List of Figures

2.1	Reference frames for chaser and target orbiting the earth	3
2.2	Summary of spacecraft simulator systems [2]	4
2.3	POSEIDYN test-bed at NPS in Monterey [2]	5
2.4	POSEIDYN smaller test-bed at NPS in Monterey	6
2.5	Flatness Tolerance	8
2.6	New granite floor	11
2.7	Leveling foot	11
2.8	Placement of levelling feet	12
2.9	New granite floor with FSS vehicle	12
3.1	Tank container and multipurpose rings [1]	17
3.2	Scheme of MyDas2 pneumatic system [1]	18
3.3	Real pneumatic system of <i>MyDas2</i>	19
3.4	Solenoid Valve	20
3.5	Air-bearing	20
3.6	Raspberry Pi Model 4b	21
3.7	Servo motors and connection to solenoid valves	22
3.8	Voltage Converter	23
3.9	Relay	23
3.10	Inertial Measurement Unit (IMU) on <i>MyDas2</i>	24
3.11	VIVE tracker and base stations [1]	25
3.12	Schematic view of the electronic components	26
3.13	Fully assembled <i>MyDas2</i>	27
4.1	Reference frames for MyDas vehicles	29
4.2	Top view of the two reference frames	30
4.3	Dynamics equations block in Simulink	33
4.4	Operations to solve equation along ξ -axes	34
4.5	Simulation results	35
4.6	Visualization of FSS moving granite floor of POSEIDYN	37
4.7	Visualization of FSS moving granite floor of Politecnico di Torino	37

5.1	Generic GNC block diagram	39
5.2	Attitude and Position Control System for <i>MyDas2</i>	40
5.3	Steps to install the Support Package	42
5.4	Configuration Parameter for the Simulink model	42
5.5	Simulink blocks library for Raspberry	43
5.6	Strategy not permitted due to the thruster configuration	44
5.7	Correct strategy with attitude maneuver before position maneuver	45
5.8	Guidance algorithm to obtain ϕ_{des}	46
5.9	Schematic diagram of the MPU-6050 connection	46
5.10	I^2C protocol enabled by a Matlab command	47
5.11	Simulink subsystems for IMU data import	47
5.12	Correction of offset error for x-y linear acceleration	48
5.13	Correction of offset error for $\dot{\phi}$ angular velocity	49
5.14	Accelerometers Bias	50
5.15	Hardware and software architecture for VIVE tracker	51
5.16	Simulink subsystem for HTC tracker data import	52
5.17	Position of the tracker relative to the geometric center of the vehicle	53
5.18	Experimental results for x and y position correction	53
5.19	Key characteristics of a dynamics system time response	55
5.20	Simulink block for PD controller	56
5.21	Steering Law logic scheme	57
5.22	Simulink blocks for steering law	58
5.23	Simulink block for PWPFM	59
5.24	Example of filtered and modulated signal	59
5.25	Simulink blocks for OBC communication	61
5.26	Example of duty cycle for SG90 servo motors [15]	61
6.1	Static test for floating without thrusters	64
6.2	Experimental results for 90° rotation	67
6.3	Experimental results for 180° rotation	67
6.4	Experimental results for 270° rotation	68
6.5	Experimental results from run # 1 for position maneuver	71
6.6	Experimental results from run # 2 for position maneuver	71
6.7	Experimental results from run # 3 for position maneuver	72

Acronyms

3-DOF

3 Degree of Freedom

GNC

Guidance, Navigation and Control

FSS

Floating Spacecraft Simulator

NPS

Naval Postgraduate School

RVD

Rendezvous and Docking

IMU

Inertial Measurement Unit

PLA

Polylactic Acid

HPA

High Pressure Air

NC

Normally Closed

PTC

Push-to-Connect

GPIO

General Purpose Input Output

DT

Digital Twin

PWM

Pulse Width Modulation

PWPFM

Pulse Width Pulse Frequency Modulation

UDP

User Datagram Protocol

Chapter 1

Introduction

This thesis work continues the work on the building of an FSS vehicle. In particular, a twin of *MyDas* [1] is built, and a GNC algorithm is tested on it, first at the Naval Postgraduate School (Monterey), and then in a new test-bed in Politecnico di Torino. In this chapter a brief overview of the objective and the structure of this work is showed.

1.1 Objectives

The main objective of the first part of the thesis is to create a Spacecraft Dynamic Simulator [2], in the footsteps of the one at the Naval Postgraduate School (NPS) in Monterey. This is a simulator with planar air-bearings technology, with 3-DOF. Then, the study, research and acquisition of all the necessary laboratory components are presented: granite table, components of the FSS vehicle, and general laboratory tools. The guidelines in the selection of the various components of the FSS vehicle were mainly to search for low-cost off-the-shelf components. On the other hand, the market investigation of the granite table focused on tables of different sizes and weights. In addition, the necessary flatness requirements were met.

Writing, validating and testing a GNC algorithm is the second main objective of this thesis work. This part took place initially at the NPS in Monterey. First, the integration of the on-board computer with Matlab/Simulink software was programmed. So, codes for importing position and attitude data from the vehicle sensors were written. Finally, a PD control algorithm was written and tested in order to complete attitude and position maneuvers.

1.2 Structure

In this paperwork, 7 chapters will be presented.

In the Chapter 2, a brief overview of all the types of Spacecraft Simulator will be exposed. In addition, the market investigation for the granite table will be shown, as well as the final choice with all the requirements. Chapter 3 wants to be a brief summary of the previous work for the FSS vehicle. In fact, a twin of *MyDas* is built, and only a few change were made. However, the bill of materials will be presented. In the Chapter 4, the equations of motion for the FSS vehicle will be written, since they are the basis for generating a digital twin of the vehicle. Through it, all maneuvers can be simulate, imposing fires duration and thrusters direction. *Matlab/Simulink* integration with the on-board computer, sensor data import and development of the GNC algorithm involve Chapter 5. Chapter 6 discusses the set up for testing the algorithm. The control algorithm parameters, the mass and inertia features of the vehicle will be showed, as well as the results of the experiments. Finally, Chapter 7 speaks about the achievement of the goals and gives some recommendations for the future works.

Chapter 2

Spacecraft Simulators

2.1 Introduction

Spacecraft simulators arise from the need to simulate spacecraft proximity maneuvers. Wilde et al. in [2] define proximity maneuver as "operation of one orbiting spacecraft moving in close proximity to another orbiting object." Generally, operations within 100 m distance between one vehicle and another can be considered proximity maneuvers. Thus, the requirements of a spacecraft simulator generated from the particular environment in which such maneuvers take place.

First, proximity maneuvers take place in an accelerated reference system, since chasers and targets orbit a central mass. So these are subjected to accelerations and trajectories relative to the central body as well. As example, figure 2.1, shows the reference frames for a target and a chaser vehicle orbiting the earth.

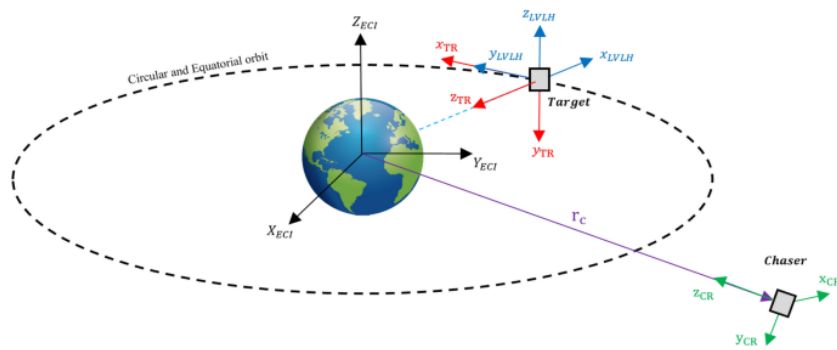


Figure 2.1: Reference frames for chaser and target orbiting the earth

In addition, the environment in which proximity maneuvers occur is quasi-frictionless, and spacecraft can move in all 6 degrees of freedom. External and internal forces also act on them.

Finally, conditions of strong contrast between light and shadow during the maneuver must also be considered [3].

2.2 Types and requirements

In order to entirely or partially replicate the conditions listed above, different types of Spacecraft Simulator were created. The table in figure 2.2 summarizes all types of spacecraft simulator system. The environment, simulator type, technologies, and DOFs determine characteristics of the simulator in order to faithfully replicate various parts of the proximity maneuver.

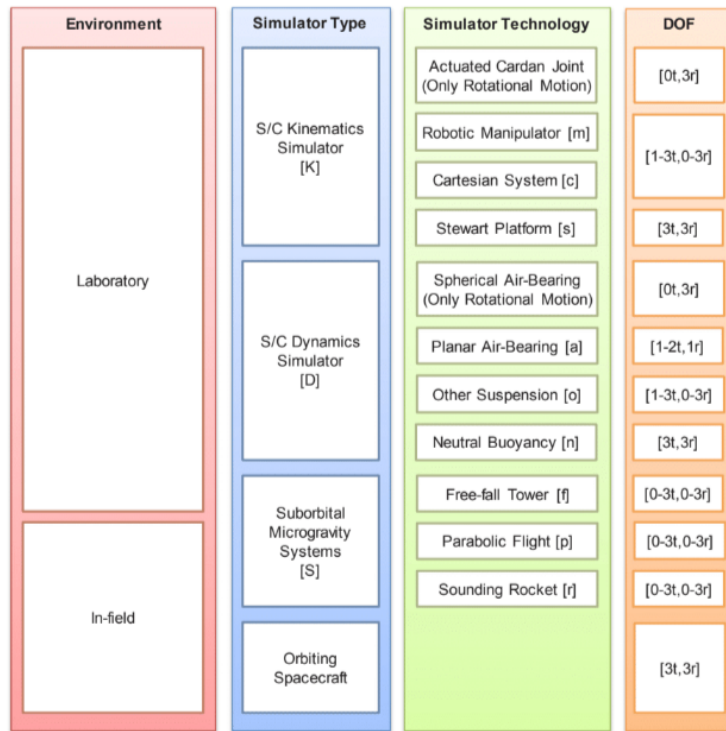


Figure 2.2: Summary of spacecraft simulator systems [2]

This thesis will not describe all simulator technologies, but it is important to emphasize the difference between a kinematic and a dynamic simulator.

The former is a system in which the simulator vehicle is not subject to the same forces and torques that it will experience in the real mission. In fact, motion is often given by electric motors that replace the spacecraft’s actual thrusters.

The latter, on the other hand, uses forces and torques delivered by real actuators, which are the same as in the real mission. This results in the need to simulate the

quasi-frictionless space environment.

All these features and peculiarities generate requirements to be followed in the design of a new spacecraft simulator. The table 2.1, briefly summarizes these requirements, which will be analyzed in detail in the next section.

Primary requirements	Secondary requirements
Mechanics of the simulation Degrees of Freedom Number of simulated object	Accessibility Availability Endurance Robustness

Table 2.1: Primary and secondary requirements of spacecraft simulators

2.3 POSEIDYN testbed

As mentioned in Chapter 1, the goal of this work is to create a new, smaller spacecraft simulator using the same environment, technology and degrees of freedom as POSEIDYN, which is present at NPS, Monterey. In order to understand the features of the new testbed, the characteristics of POSEIDYN are analyzed below.

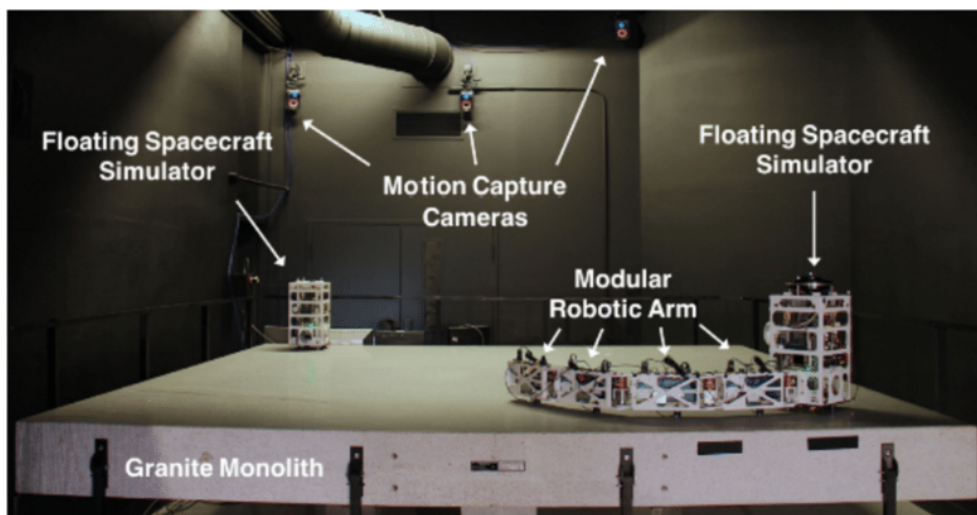


Figure 2.3: POSEIDYN test-bed at NPS in Monterey [2]

POSEIDYN is a spacecraft dynamics simulator that employs air-bearings technology to simulate 3-DOF proximity maneuvers [4]. Figure 2.3 shows the main

components of the test-bed.

Actually, all the tests performed in Monterey has been carried out in another test-bed, smaller than POSEIDYN but with the same characteristics. In this work, when POSEIDYN is mentioned, it is referred to the smaller testbed, shown in figure 2.4.

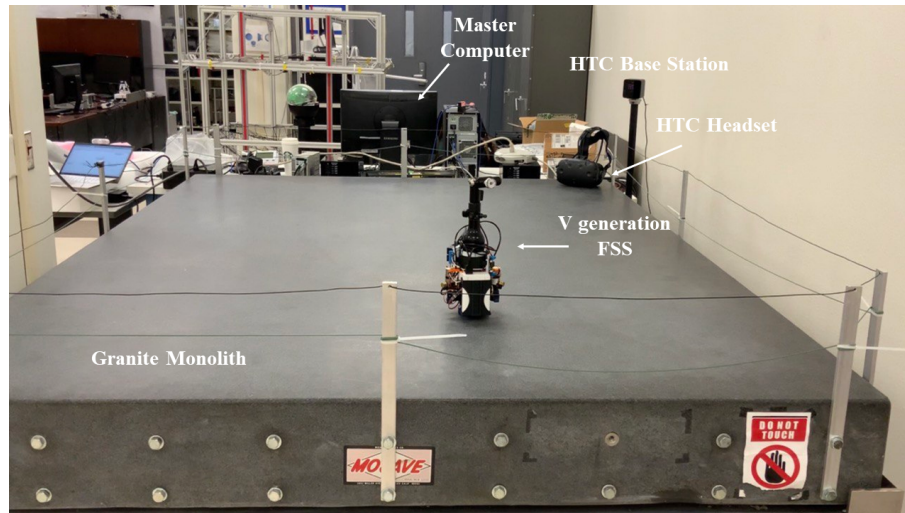


Figure 2.4: POSEIDYN smaller test-bed at NPS in Monterey

This technology uses tables or flat-floors on which special vehicles move. They are called floating spacecraft simulator (FSS) vehicle. There is a very thin film of compressed air between such vehicles and the table, so that they can move on it in a quasi-frictionless environment. POSEIDYN, is an air-bearing simulator with passive floor system: air is expelled from tanks on the vehicle itself. However, there are active floor systems, in which air is emitted from small holes in the table. In order to ensure some stability of the vehicle on the floor, requirements must be met in terms of floor flatness and leveling accuracy, as well as the choice of air bearings suitable for the weight of the vehicles.

Vehicles tracking is another important factor in simulator design. A commercial-of-the-shelf tracking system is used in the POSEIDYN to obtain data on vehicle position and attitude. In addition, onboard sensors, such as accelerometers and gyroscopes, are also usually fitted. The integration and use of these sensors will be discussed in later chapters.

Regarding secondary requirements, air-bearings dynamics simulator systems have characteristics that make them perform very well in *accessibility* and *availability*. While, kinematic simulator have better characteristics for *robustness* and *endurance*.

Accessibility refers to how easily new vehicles can be added to the testbed or existing ones can be modified. Availability, on the other hand, pertains to the

flexibility of the simulation system in meeting new test requirements.

Robustness and endurance are more vehicle-related requirements that indicate the risk of collision and the duration of the simulation before a reload, respectively.

The table 2.2 summarizes the features and requirements of POSEIDYN [4], from which the research to implement the new testbed at Politecnico di Torino started. The market survey for purchasing the necessary components will be discussed in the next section.

Requirements	
Simulator type Mechanics of the simulations Deegres of Freedom Number of simulated object Accessibility Availability Robustness Endurance	Dynamic Decoupling Dynamic from the enviornment 2 transl. + 1 rot. = 3 up to 5-6 High High < Kinematic hours, depending on the vehicle
Features	
Techonlogy Testbed Material Testbed dimensions [m] Table weighth [tons] Air-Bearings gas Tracking System	Air-berings Granite 2,4 x 1,8 15 Compressed Air HTC VIVE system

Table 2.2: POSEIDYN requirements and features

2.4 Granite floor

The main component of this type of simulator is the passive floor (or table). In fact, this usually represents the largest cost, including transportation and installation. The main characteristic of a floor is flatness. Flatness defines the permissible deviation from a perfectly flat surface. It is usually expressed in units of length, often in micrometers [μm].

For this applications, the requirement for flatness is very stringent: it ranges from 3 μm to 46 μm . The most commonly used materials for making floors are

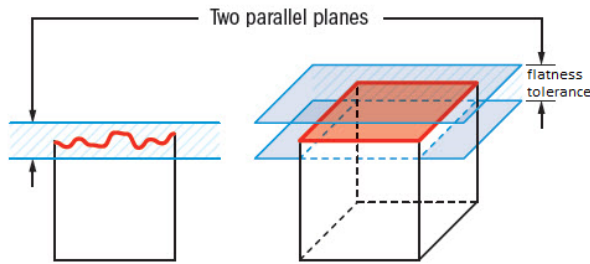


Figure 2.5: Flatness Tolerance

glass, epoxy resin and granite. The first two allow for very large floor dimensions, but they have smaller flatness tolerances than granite. Since a very large floor size was not chosen, research immediately focused on granite floors.

In Europe, the DIN 876 standard says that the error of flatness (or tolerance) is defined as the deviation from an ideal plane equidistant from two parallel planes touching the highest and the lowest point of the surface of the plane under examination, as you can see in figure 2.5. This standard defines also five grades of flatness, which allow calculation of the required flatness tolerance. In table 2.3, the grades are shown.

Grade	Tolerance [μm]
00	$2(1 + \frac{L}{1000})$
0	$4(1 + \frac{L}{1000})$
1	$10(1 + \frac{L}{1000})$
2	$20(1 + \frac{L}{1000})$
3	$40(1 + \frac{L}{1000})$
L = longest side of the floor in [mm]	

Table 2.3: DIN 876 grades of flatness

In order to achieve a very high level of flatness, research were focused on granite table with flatness $< 12\mu m$, which is the value of POSEIDYN testbed. In addition, there is another important requirements: the levelling precision. It can be achieved by the adjusting the levelling feet, supplied with the floor.

The search for the new granite floor started with relatively large floors in order to make an initial estimate of the cost and weight of the floor. Then smaller floors were evaluated, and the desired floor was then chosen.

Sizes range from very small tables, $40 \times 40cm$, to larger, $2 \times 2m$. A key factor in the choice is the weight. Granite is a material with very high specific weight, which is why very large floors mean considerable weights. In addition, high weights require very high transportation and installation costs, as well as more support feet. As a result, adjustment becomes much more complicated and unstable. In addition, very big tables are distant from the goal of this work. In fact, the new testbed is supposed to be a miniature reproduction of the POSEIDYN, where initial operation and testing of FSS vehicle components can be carried out. Finally, the delivery time of the piece was also considered, since long lead times would be incompatible with the duration of the thesis work.

So the selection process started with the analysis of the requirements and their constraints, and the final choice fell on granite table No. 4. In table 2.4, all the requirements and constraints are summarized.

Floor # 4, fully meets all the requirements and related constraints chosen. The choice of this floor resulted in:

- lowest purchase, transportation and installation costs among all alternatives;
- the best flatness tolerance. This means better test quality and better air-bearings operation;
- very low weight and minimal number of adjustment feet: the floor is "transportable" and very versatile. In addition, the presence of only 3 leveling feet allows precise and durable leveling;
- short lead times.

In figure 2.6 - 2.7, the new granite floor and the leveling feet are shown.

The test report according to the reference standard ISO8512-2/00 was carried out on the table using the "Wyler" measuring instrument. The global flatness is 1.8μ .

The levelling foot is a tilting plate type with diameter $30 mm$ and M16x1.5 adjusting screw. In order to ensure correct and durable 3-point leveling, the feet are positioned as in the figure 2.8, according to guidelines given by the manufacturer.

Finally, to ensure safe use of the floating vehicles, a barrier was built around the table. A piece was designed to insert 20-cm wood rods with a diameter of 1 cm. The rods, placed at the vertices of the table, hold the wires that act as guard rails.

Requirements and constrains					
Table #	Weight < 2500 kg	Flatness < 12 μm	n. feet < 5	Cost < 5000 €	Lead time < 6 weeks
1		•			
2		•			
3	•	•	•	•	
4	•	•	•	•	•
5	•	•		•	
6	•	•		•	
7	•	•		•	•
8	•	•		•	•
• = requirement met					

Table 2.4: Requirements and constraints in the choice of granite floor

Figure 2.9 shows the new granite floor with the FSS vehicle, which will be discussed in detail in the next chapter.

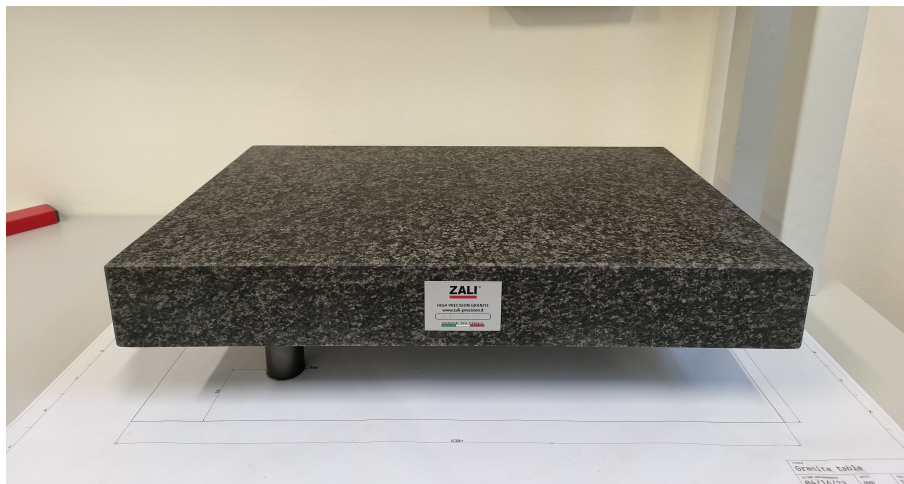
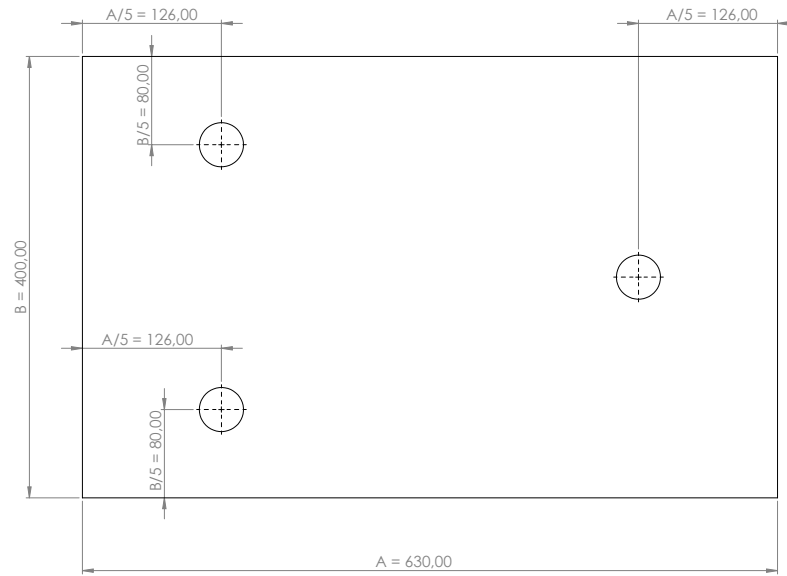


Figure 2.6: New granite floor



Figure 2.7: Leveling foot



SOLIDWORKS Educational Product. Solo per uso didattico.

Figure 2.8: Placement of levelling feet

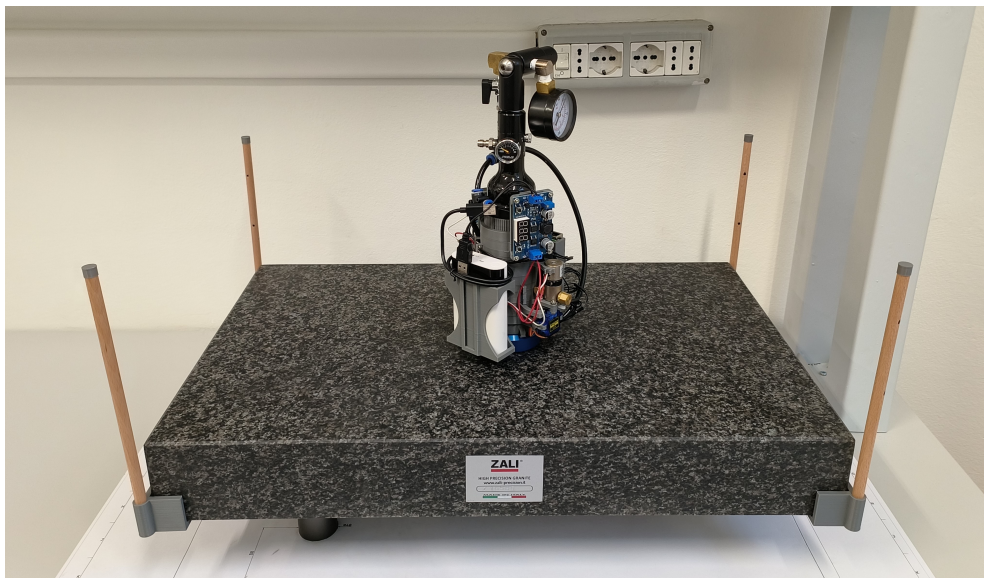


Figure 2.9: New granite floor with FSS vehicle

Chapter 3

Floating Spacecraft Simulator

In this chapter, the main elements and building techniques of floating spacecraft simulators (FSS) will be presented. The goal of the thesis work is to build a twin of the *MyDas* vehicle, built by Kulke in [1]. *MyDas* was built at NPS, Monterey, and represents the fifth generation of FSS vehicles of the lab. Since a description of the individual elements of *MyDas* is already extensively discussed in [1], this chapter aims to briefly expose the general operation of the simulator, and also describes minor modifications made to *MyDas2* in order to make it more performing. Bills of materials to build the vehicle are given in the appendix.

3.1 General requirements

Thanks to the air-bearings located on the Floating Spacecraft Simulators, these vehicles can move on the floor in a quasi-frictionless environment. Furthermore, Zapulla et al [4], claim that hardware phenomena like "delays, computational constraints, actuator response uncertainty, and sensor noise" can be simulate with this kind of vehicle and test-bed. Thus, FSSs faithfully reproduce the dynamics of the spacecraft or space object that will maneuver in space.

It is evident from this description that FSSs are complex simulators in which various subsystems are present. The list below mentions the basic subsystems required for each vehicle, but more complex simulators with secondary functions can be developed and other secondary subsystems can be built:

- **Pneumatic System** consists of hoses, pressure regulators, connectors, and valves that, fed from the air tank, supply compressed air at the right pressure to the air bearings and thrusters (if present). The size of the air tank may differ

depending on the size of the vehicle chosen and determines the *endurance* of the simulator.

- **Electronic System**, on the other hand, includes the hardware and software part of the vehicle. On-board computers, cables, servo motors, actuators and sensors are included within this subsystem. Depending on the maneuver to be performed, the complexity of the algorithm and other factors, very different configurations can be created. For this reason, these simulators possess high *availability*.
- **Structure** needs also to be mentioned, since it is how components stay connected, and it affects the shape and weight of the vehicles.

As an example, some characteristics of the 4 generations of FSS present at the NPS are shown (table 3.1).

It is possible to see how these vehicles can come in various configurations.

Different materials can be used for the structure. In recent years the trend is toward plastic materials, made by additive manufacturing. This saves weight, but also gives the testbed greater robustness, since any damage can be easily repaired by printing the damaged part again.

In addition, various actuators can be used: in the four generations at NPS, gas thrusters have always been present, but this does not preclude configurations with only reaction wheels could exist. In the next section, *MyDas* includes only the presence of vectorable gas thrusters.

As mentioned earlier, sensors play an important role in the testbed, since they are the interface between the satellite and the external environment. These can be optical sensors, such as cameras, virtual reality devices that enable tracking, but also that onboard IMU (inertial measurement unit). Choosing one sensor over another allows data of a different nature to be collected, which will then be used by the onboard computer to feed the control algorithms.

Finally, additional subsystems can be included, such as robotic arms or rendezvous & docking (RV&D) devices.

3.2 MyDas2 Vehicle

The goal of this section is to expose all subsystems of the new FSS, built at the Politecnico di Torino. *MyDas2* is a twin of *MyDas*, to which minor modifications have been made to make it more efficient. The table 3.2 summarizes the most important parameters of this vehicle, which will be explained in detail in the following subsections.




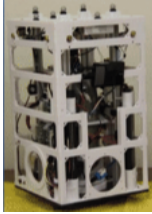
Gen. #				
	I	II	III	IV
Structure	Aluminium	Aluminium	PLA	PLA
Size [m]	$0.4 \times 0.4 \times 0.85$	$0.3 \times 0.3 \times 0.69$	unk	$0.27 \times 0.27 \times 0.52$
Weight [kg]	63	26	unk	9.8
Pneumatic	Air	Air	Air	Air
Actuator	Thruster and RW	Thruster and CMG	Thruster and RW	Thruster and RW
Sensor	IMU Vision Sensor	IMU Position Sensor	IMU iGPS	IMU VICON
Other	Capture system for RV&D	Vectorable thrusters		Robotic Arm
Reference	[5]	[6]	[7]	[8]

Table 3.1: Characteristics of FSS at NPS

3.2.1 Structure

The structure is used to contain the air tank, which is the largest and heaviest component. It must also ensure the allocation of all other components, and must provide for connection with the air bearing. To do this, a primary structure is designed, to which "multipurpose rings" can be connected.

The tank container and multipurpose rings were 3D printed, using polylactic

Subsystem	Characteristic	Parameter
Structure	Length and width	$12 \times 12 \text{ cm}$
	Height	31.5 cm
	Mass (overall)	1.7 kg
	J_z (overall)	$2.761 \times 10^{-3} \text{ kgm}^2$
Propulsion	Propellant	Air
	Equiv. storage cap.	$0.7 \times 10^{-3} \text{ m}^3 @ 0.41 \text{ MPa}$
	Operating Pressure	0.41 MPa
	Thrust (x2)	0.180 N
Flotation	Propellant	Air
	Equiv. storage cap.	$0.7 \times 10^{-3} \text{ m}^3 @ 0.41 \text{ MPa}$
	Operating Pressure	0.41 MPa
	Linear Air Bearing	65 mm diameter
	Continuous Operation	$\sim 15 \text{ min}$
Electrical & Electronics	Battery type	Lithium-Polymer (LiPo)
	Storage capacity	8 Ah @ 5V
	Continuous Operation	$\sim 12\text{h}$
	Computer	Raspberry Pi 4
Sensor	IMU	MPU6050
	Position sensor	HTC VIVE system

Table 3.2: Key parameters of MyDas2

acid (PLA). Such material turns out to be very light and with good capacity in terms of strength. In addition, it is the most widely used material in 3D printing.

The tank container is a hollow cylinder that houses the tank and allows, thanks to holes in the bottom, connection with the air bearing. On the outer surface, the tank container has 72 vertical grooves, so it can accommodate multipurpose rings. These can be oriented as desired with an angle increment of 5° . This type of structure gives the FSS a very high availability, since rings can be designed as needed while maintaining the same central structure.

The current configuration of the *MyDas2* involves the use of 12 rings. These can become up to 15 by increasing the height of the tank container. Below is a list of the rings used:

- **ring for servos (A):** is a single ring that allows the two servos to be held. These are placed 180° apart from each other. It also has two holes to screw the servos to the ring.
- **rings for dovetail (B):** with these rings it is possible to dovetail a piece. In the *Mydas2* there are two versions of these: the first has only one seat, while

the second has two seats placed at 180° . In this vehicle they are used to hold the battery, on-board computer, and two other electronic components that will be analyzed later.

- **rings for cables (C)**: allow the placement and management of electrical cables.
- **ring for adjustment (D)**: this ring does not hold any parts. Instead, it is useful to align servos. It consists of two fins that allow the two directions to be identified at 0° and 90° .
- **ring for stabilization (E)**: this ring has two holes to insert the head of the gas thrusters. Without this ring, the thrusters would only be held by the servos and would not be stabilized.
- **ring for VIVE tracker (F)**: it holds the VIVE tracker.

Figure 3.1 shows a CAD of the tank container (a) and the multipurpose rings (b).

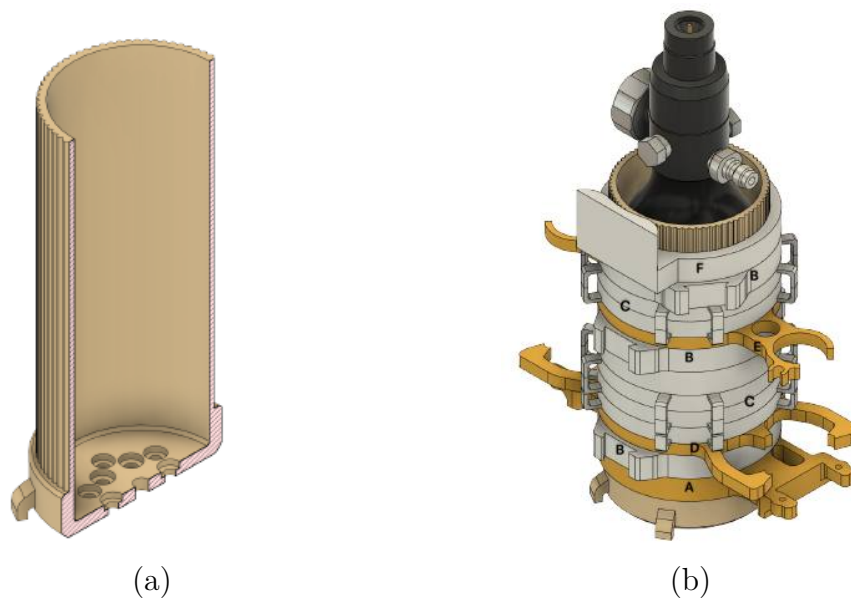


Figure 3.1: Tank container and multipurpose rings [1]

3.2.2 Pneumatic System

The pneumatic system is a key component for FSSs. In fact, it allows the vehicles to float on the granite floor and, if present, it feeds the gas thrusters. So part of the pneumatic system are the air tank, air bearings, and solenoid valves that act as gas thrusters. In addition, however, other connecting and pressure regulating devices are included in this subsystem.

Figure 3.2 gives an overview of the operation and components of the pneumatic system for *MyDas2*.

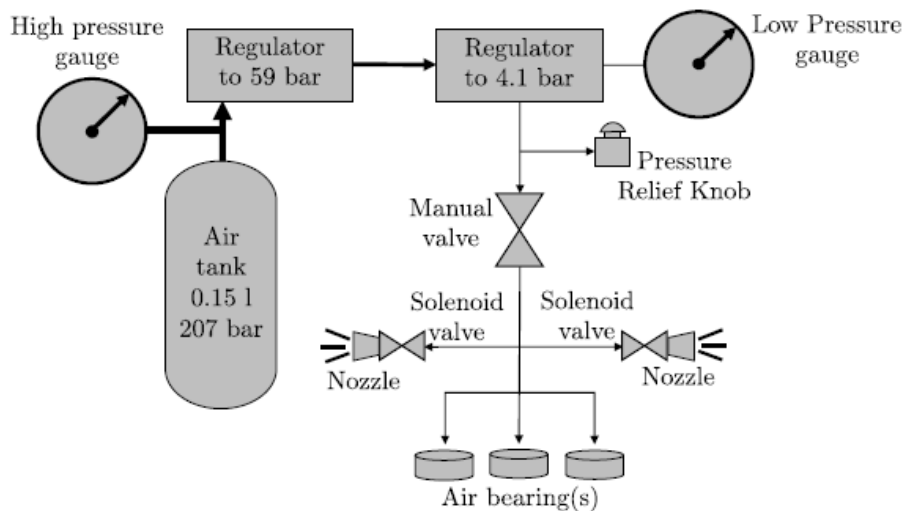


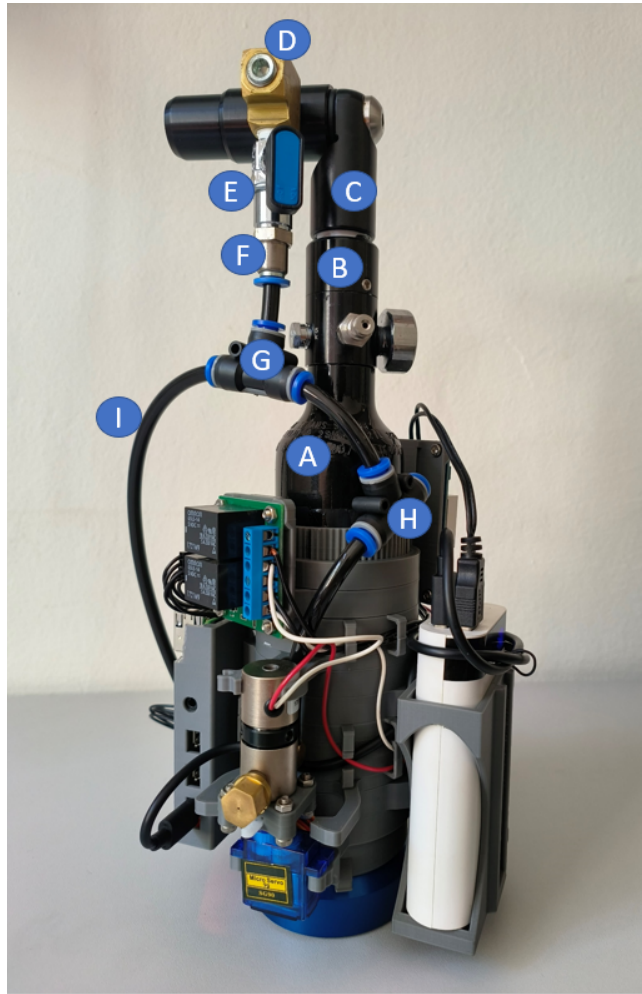
Figure 3.2: Scheme of MyDas2 pneumatic system [1]

The air-flow starts with the air tank. In order to miniaturize the vehicle, one of the smallest refillable air tanks on the market was chosen. It is an HPA (high pressure air) 13/3000 tank. It operates at 200 bar (3000 psi) and it has a volume of 0.21 L (13 ci). It also has a built-in regulator that delivers air at a pressure of 60 bar (800 psi).

Since the solenoid valves and air bearings work at operating pressure of 4.1 bar (60 bar), another pressure regulator is needed. The solution is a 90° pressure regulator by Palmers Pursuit Shop. This regulator produces an adjustable output pressure from 0 bar to 8.6 bar and can take an input pressure of up to 310 bar.

Now, the air is at the right pressure, but it needs to be delivered to the two solenoid valves and the air bearing. A system of tubes, connector, manual valve and intersection pipes allows air to be delivered out of the second pressure regulator to these components. Figure 3.3 helps to understand all the connections.

The second pressure regulator has two outputs. The first is occupied by a pressure gauge, which allows adjustment of outlet pressure from the regulator to



- | | |
|------------------------------------|-----------------------------------|
| A: Air-tank | B: Built-in pressure regulator |
| C: 90° pressure regulator | D: T-intersection pipe |
| E: Manual valve | F: 1/8 NPT to PTC |
| G: PTC to air-bearings & thrusters | H: PTC to thruster 1 & thruster 2 |
| I: Tube OD 6mm - ID 4mm | |

Figure 3.3: Real pneumatic system of *MyDas2*

4.1 *bar*. On the other side, a T-intersection pipe fitting with 1/8 NPT thread allows the flow to be directed downward. Connected to this is a manual valve, which allows air flow to be shut off or enabled. From the manual valve, a 1/8 NPT push-to-connect adapter allows the insertion of an OD 6mm - ID 4mm polyurethane hose. From this hose, with a series of push-to-connect (PTC) T-connectors, air reaches the thrusters for propulsion and the air bearing for floating.

3.2.3 Propulsion and flotation

In *MyDas2*, the actuators that provide propulsion for the vehicle are two vectorable gas thrusters. A gas thruster is a type of rocket engine which uses the expansion of a pressurized gas to generate thrust. *Vectorable* means that it is possible to direct the thrust, via dedicated servo motors.

Two solenoid valves act as thrusters in this vehicle. A solenoid valve is an electromechanical device used to control the flow of fluids or gases. It consists of a coil (solenoid) of wire and a movable ferromagnetic plunger, known as the armature, that is connected to a valve mechanism. When an electric current is passed through the coil, it creates a magnetic field that attracts the armature, causing it to move and open or close the valve. The valves used in this design are normally closed (NC). It means that only when the current passes, the valves open. In figure 3.4, the solenoid valve is shown. It is a cylindrical body, with an inlet (right) and an outlet port (left). A nozzle is connected to outlet port, to increase the efficiency of the thruster.



Figure 3.4: Solenoid Valve

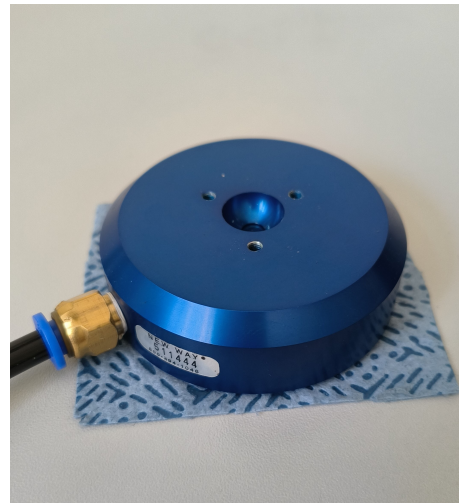


Figure 3.5: Air-bearing

Air-bearing allows the vehicle to move in a quasi-frictionless environment. In *MyDas2*, a flat round air bearing with a diameter of 65 mm is used (figure 3.5). The flight height for an ideal load of 666 N is 5 μm . Since the weight is much less than ideal, from the tables provided by the company, a flight height of $\sim 25 \mu\text{m}$ is deduced. The current air-bearing can be loaded with a pressure range 0.414 MPa - 0.552 MPa.

3.2.4 Electronic System

Like the pneumatic system, the electronic system plays a key role in the design and building of an FSS. In fact, it aims to control propulsion, via servos and solenoid valves. Also, in the development of GNC algorithms, the on-board computer gathers data from the sensors, processes it, and uses it to feed the control algorithm. So the main elements of the electronic system are the onboard computer, batteries, solenoid valves, and servos. Then there are other secondary components, such as connecting cables, voltage converters, etc... In this subsection all these components will be presented.

On-board computer It is used to control propulsion and run GNC algorithms. In *MyDas2*, Raspberry Pi 4 is used. Raspberry Pi computers are based on ARM processors and feature a set of hardware components including a CPU, RAM, network connectivity, USB ports, HDMI ports, and memory card slots. They are available in various versions, each with slightly different features and technical specifications. Raspberry Pi 4 model B (figure 3.6) is the newest version, and it has better features in terms of processor, Bluetooth connectivity, and WiFi.

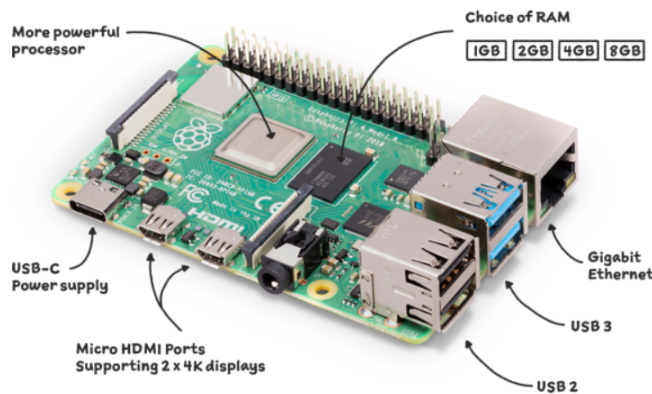


Figure 3.6: Raspberry Pi Model 4b

In addition, Matlab/Simulink offers the possibility of downloading Add-Ons for Raspberry use. Programming control algorithms thus turns out to be easier and more intuitive for those who already have some knowledge of these software.

Raspberry boards have a grid of physical pins, called GPIO (General Purpose Input-Output). These pins can be used to read digital inputs from sensors or switches, or to output digital signals to control actuators or devices. There also pins to provide a certain voltage to devices. The diagram of all connections will be presented at the end of this section.

Power Supply The solution chosen for the power supply of *MyDas2* was the 8000 *mAh* power bank from SBS. It has two 5V outputs via USB type A. One output provides 2.1A, the other 1A. The first is used to power the raspberry, while the other is used for the voltage converter. To charge the battery, a micro-USB port is available.

The choice of this type of battery turns out to be the best in terms of weight, ease of charging, duration of operation, and cost.

Solenoid Valves and Servos Solenoid valves are electronic devices that provide propulsion to the vehicle. Valves from the company GEMS were chosen for the *MyDas2*. These valves consume 2 W at 12 V. Since the raspberry provides only 3.3 V or 5V, a voltage converter is used to increase the voltage from 5 to 12 V, and a relay to distribute that voltage to the two valves.

Through servo motors, the electrovalves can rotate with a range of 180°. Micro Servo Motors 9G from RUIZHI were used for this vehicle. The voltage input is between 4.8 V and 6 V, so they can be powered directly from the raspberry. In addition, this type of servos has a built-in potentiometer. This feature allows the motor to know its position along the 180° range. The position is controlled via Pulse Width Modulation (PWM), which will be explained in the Chapter 5. In Figure 3.7, the servos are equipped with an arm (a) that via a gear wheel connects to the shaft (b). This arm is then connected via screws to the solenoid valve. This connection mechanism was designed by Joseph Kulke in [1].

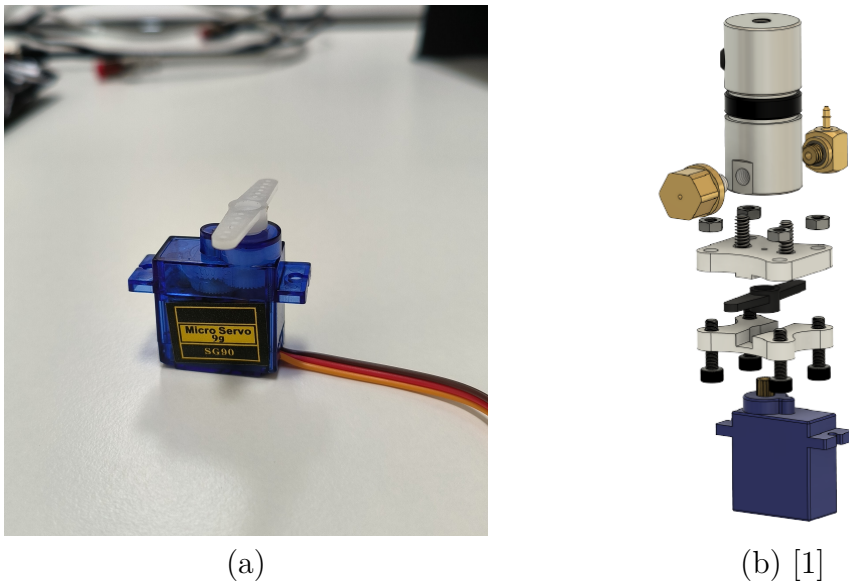


Figure 3.7: Servo motors and connection to solenoid valves

Relay and Voltage Converter The solenoid valves need 12 V to be powered properly. This voltage is not provided by the Raspberry, nor by the power bank. So, a voltage converter is necessary. This component receives 5 V voltage from the power bank, via a customized USB type-B cable, and supplies 12 V. A converter was chosen for the *MyDas2* that has a built-in potentiometer and LCD screen. The potentiometer allows, if necessary, to set an output voltage ranging from 3.5 V to 35 V, as long as it is higher than the input voltage. The LCD screen allows the output voltage to be displayed at the input, and therefore the use of a multimeter is not necessary. This allows other types of solenoid valves to be used, raising the availability of the vehicle. Figure 3.8 shows the device chosen for *MyDas2*.

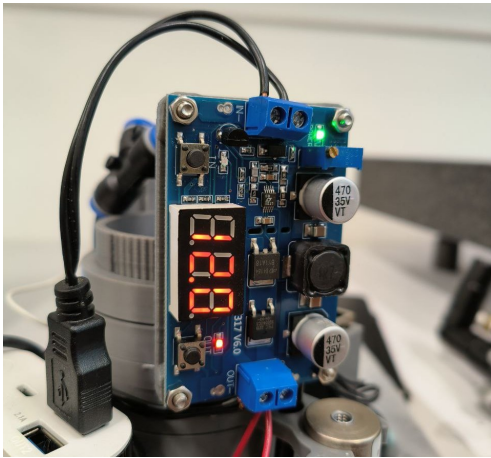


Figure 3.8: Voltage Converter

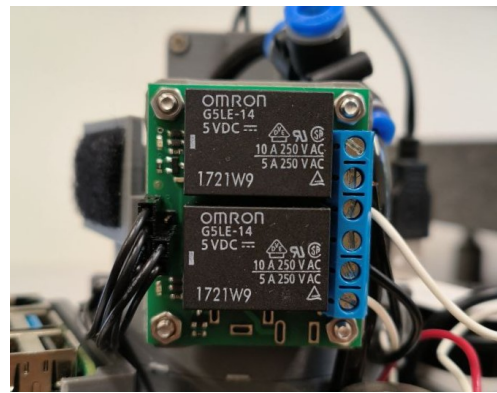


Figure 3.9: Relay

Since the valves cannot be controlled directly from the Raspberry, it is necessary to insert a relay. In *MyDas2*, a two-channel 5V relay module by Pololu is used. The pins of the relay are shown in figure 3.9. On the left side are the low voltage pins, then the ones connected to the raspberry. On the right are the high voltage outputs for the two solenoid valves.

3.2.5 Sensors

Sensors are the devices with which the vehicle communicates with the external environment, receiving data in terms of position, speed or acceleration. In this thesis work, collecting data from sensors is crucial, since one of the goals is to test a GNC algorithm. The fifth generation FSS of the NPS, of which *MyDas2* is a part, uses two types of sensors, which will be presented below.

IMU First sensor is an Inertial Measurement Unit (IMU). This device combines accelerometer and gyroscope. Accelerometer measures linear acceleration in three

axes (X, Y, and Z), allowing the determination of changes in velocity or the presence of external forces. Gyroscope, on the other hand, measure angular rate or rotational motion around the same axes. By integrating the data from accelerometers and gyroscopes over time, it is possible to estimate the object's position, velocity, and attitude. For *MyDas2* an MPU-6050 from Paradisetronic was chosen (figure 3.10).

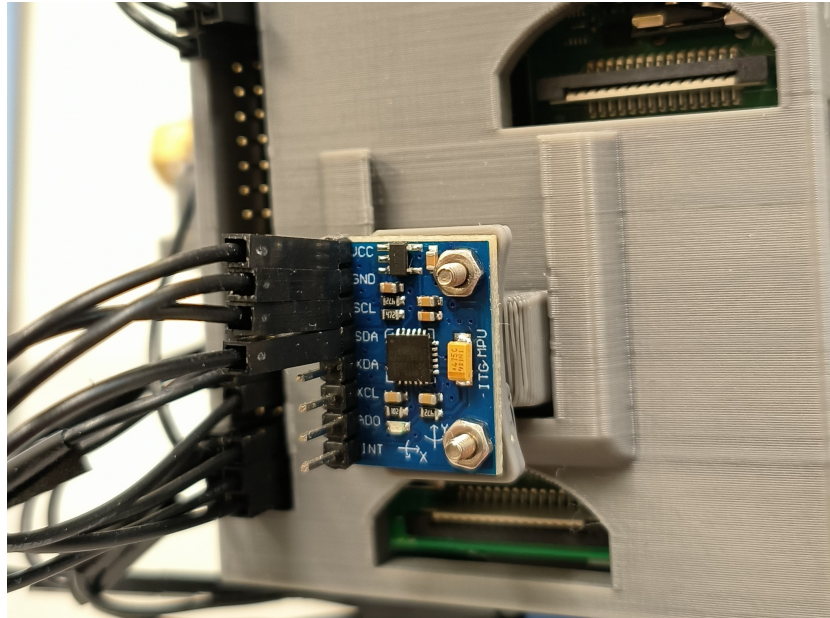


Figure 3.10: Inertial Measurement Unit (IMU) on *MyDas2*

This device has a 3-axis accelerometer and a 3-axis gyroscope. The input voltage can be 3 - 5 V, so the board can be directly connected to raspberry trough specific jumper cable. It uses the I^2C protocol to communicate with the raspberry. The master-slave architecture is used by the I^2C interface, where one device acts as the master and initiates the communication. The other devices act as slaves and respond to the master's commands or requests. The bus and the timing and sequence of the data transfer are controlled by the master device.

This will be discussed in more detail, along with the integration with the Simulink model and the code for importing data, in the chapter 5.

HTC VIVE System The second sensor is a tracking sensor. It allows tracking the position (x,y,z), orientation (θ , ψ , ϕ) and respective velocities of the vehicle. This is a VIVE tracker by HTC. Since the system is 3-DOF, only x , y , ϕ (rotation around z-axis) will be considered.

The size and weight of the tracker is very reduced. It also has an internal rechargeable battery, so it does not require any physical connection with the

on-board computer.

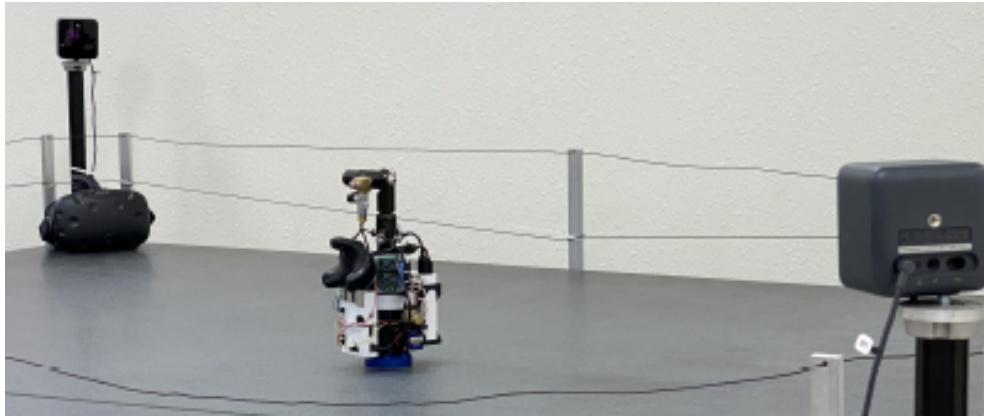


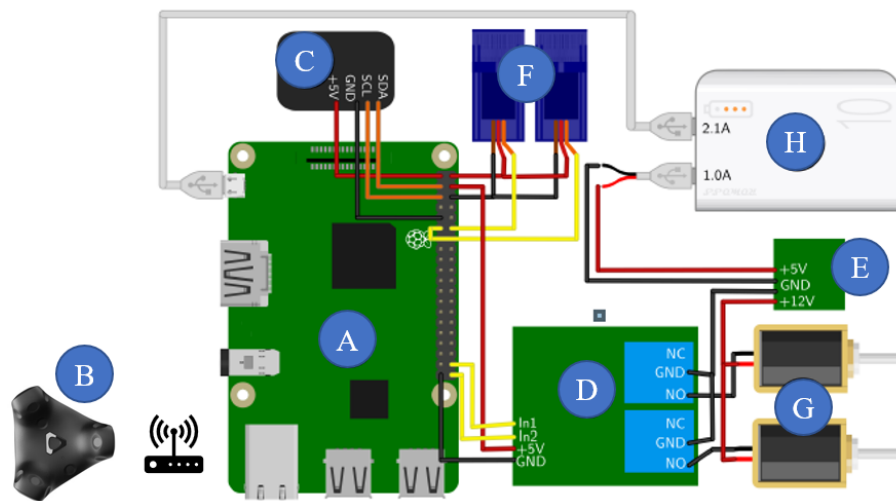
Figure 3.11: VIVE tracker and base stations [1]

The tracker works together with two Base Stations. They are two devices that track the position and speed of the tracker in space. The Base Stations are continuously powered, but they are not located on the vehicle, so this is not a problem. Once data is obtained from the base stations, the tracker communicates via Bluetooth with a computer outside the vehicle. Through the Unity and Steam VR software, data can be collected in real time and sent to the Raspberry via User Datagram Protocol (UDP).

UDP communication is a network protocol that allows data to be transferred across a computer network quickly and inexpensively. In the chapter 5, the mode of connecting and receiving data will be explored in depth.

In figure 3.11, FSS with VIVE tracker and the two base stations is shown. It is possible to see also the Headset, since it is necessary to complete the set-up.

In figure 3.12, a schematic of all the electronic components of the FSS, including the UDP connection with the VIVE tracker, is shown. Figure 3.13 shows the fully assembled vehicle.



- | | |
|----------------------|-----------------|
| A: Raspberry Pi 4B | B: VIVE tracker |
| C: IMU - MPU 6050 | D: Relay |
| E: Voltage Converter | F: Servo motors |
| G: Solenoid valves | H: Power Supply |

Figure 3.12: Schematic view of the electronic components

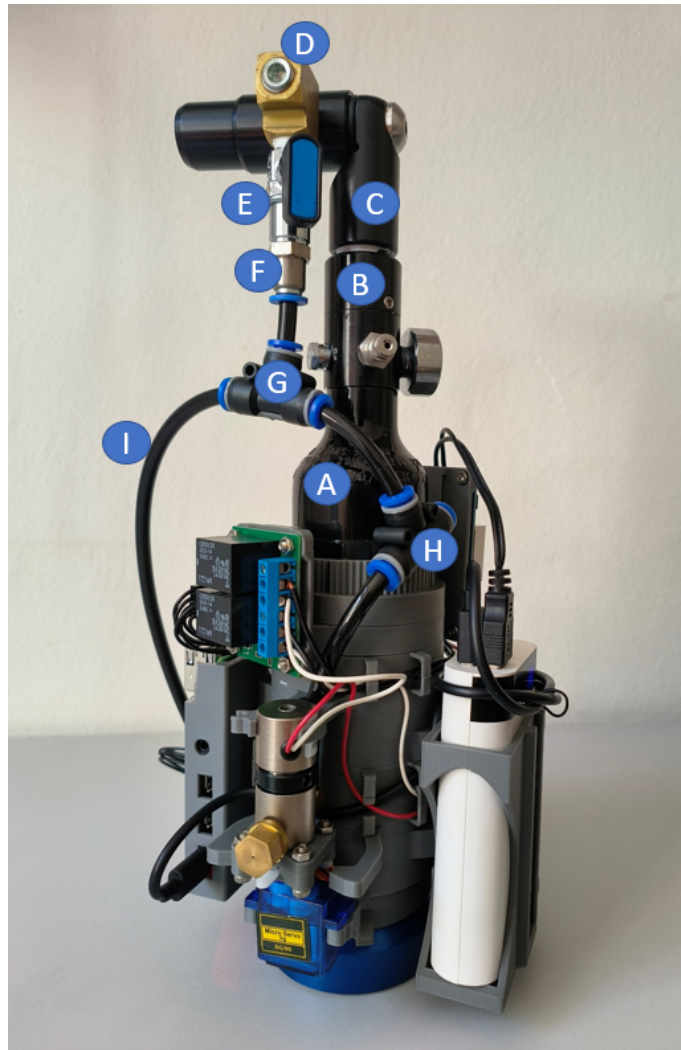


Figure 3.13: Fully assembled *MyDas2*

Chapter 4

Digital Twin

This chapter involves the creation of a digital twin (DT) of *MyDas2*. Glaessgen and Stargel in [9] claim that "a Digital Twin is an integrated multiphysics, multi-scale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin".

The main objective of a digital twin is to understand, monitor, and optimize [10] the real-world object or system. By the digital twin, simulations of scenarios can be performed, performance data can be gathered, and analysis of performance can be conducted. Furthermore, changes and improvements can be tested without affecting the physical entity.

A digital twin is created by using data from sensors, IoT (internet of things) devices, and other connected sources related to the real object or system. This data powers the creation of a real-time virtual version that can be manipulated and visualized digitally.

For *MyDas2*, the goal is to create a DT that can simulate maneuvers before performing them on the physical vehicle. This is useful for choosing the right thruster angle, or firing duration, in order to perform smoother and more controlled maneuvers. In addition, the ability to visualize maneuvers actually performed by the vehicle, through the processing of data received from the sensors, makes the test-bed even more available and efficient.

First, the equations of motion of the vehicle will be written, in order to create a physical model. Then, the rotation matrices will be presented to switch between the moving reference system, centered on the vehicle, and the fixed reference system of the granite plane and the VIVE tracker. Then the codes written in Matlab/Simulink to run the simulation will be discussed. Finally, a graphical model of the vehicle will be presented to visualize the maneuvers performed or simulated.

4.1 Equations of motion

The dynamics of the vehicles for the two translational and one rotational degrees of freedom can be described by the Hill-Clohessy-Wiltshire (HCW) equations. These equations are commonly used to model the relative translational motion between two spacecraft in near-circular orbits. The spacecraft involved are typically referred to as the Target and Chaser spacecraft.

In certain scenarios where the maneuver is short compared to the orbital period and the spacecraft remain in close proximity, the dynamics can be further simplified to a double-integrator model. This simplification allows for experimental evaluation of these methods using a ground-based test bed [11]. Recalling that there are two translational and one rotational degrees of freedom, it is possible to write the general equations of the dynamics:

$$\ddot{x} = \frac{f_x}{m}, \quad \ddot{y} = \frac{f_y}{m}, \quad \ddot{\phi} = \frac{\tau}{J_z} \quad (4.1)$$

where x and y represent the two direction on the plane; ϕ is the rotation about the third direction, perpendicular to the plane; f_x, f_y, τ are respectively the control forces along x, y and the control torque; m is the mass of the vehicle and J_z is the inertia about the vertical axis.

Figure 4.1 (from [1]) helps to understand the specific equations for *MyDas2*.

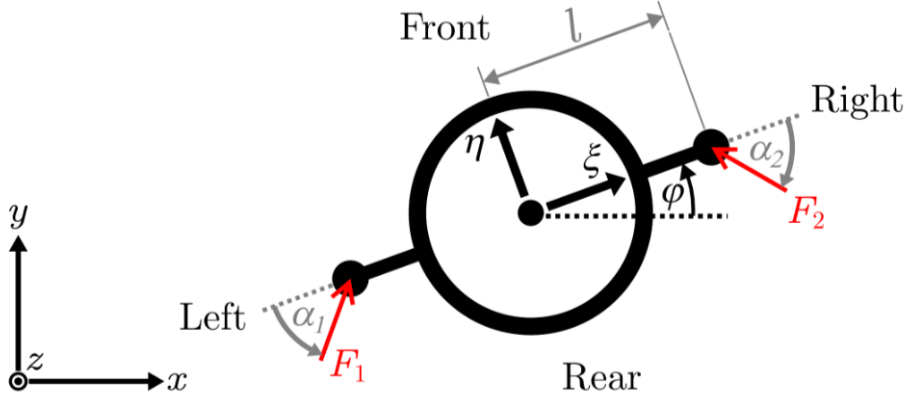


Figure 4.1: Reference frames for MyDas vehicles

At the bottom left is the fixed reference system, represented by the granite floor. It is described by the coordinates x, y, z . On the vehicle there is another reference system, which rotates with it, denoted by the coordinates ξ, η . The coordinate ϕ indicates the angle of rotation about the vertical axis of the vehicle. The two arms on the left and right end with the two thrusters present in *MyDas2*. These

can rotate by an angle α_1 or α_2 . F_1 and F_2 represent the forces delivered by the thrusters. The parameter "l" is the arm of the forces of the thrusters, which generate momentum around the z-axis.

With this parameters, it is possible to write the dynamics equations proper of *MyDas2*:

$$\ddot{\xi} = \frac{F_1 \cos \alpha_1 - F_2 \cos \alpha_2}{m} \quad (4.2)$$

$$\ddot{\eta} = \frac{F_1 \sin \alpha_1 + F_2 \sin \alpha_2}{m} \quad (4.3)$$

$$\ddot{\phi} = \frac{-F_1 l \sin \alpha_1 + F_2 l \sin \alpha_2}{J_z} \quad (4.4)$$

4.2 Direction Cosine Matrix

As mentioned above, there are two reference systems: one fixed coincident with the sides of the granite plane and one centered on the vehicle. The equations of motion are written in the vehicle-centered reference system, but the data from the VIVE tracker, are in the fixed global reference system [5]. The control algorithms will be written in the fixed reference. So, it will be necessary to write a rotation matrix that allows switching from one reference system to another.

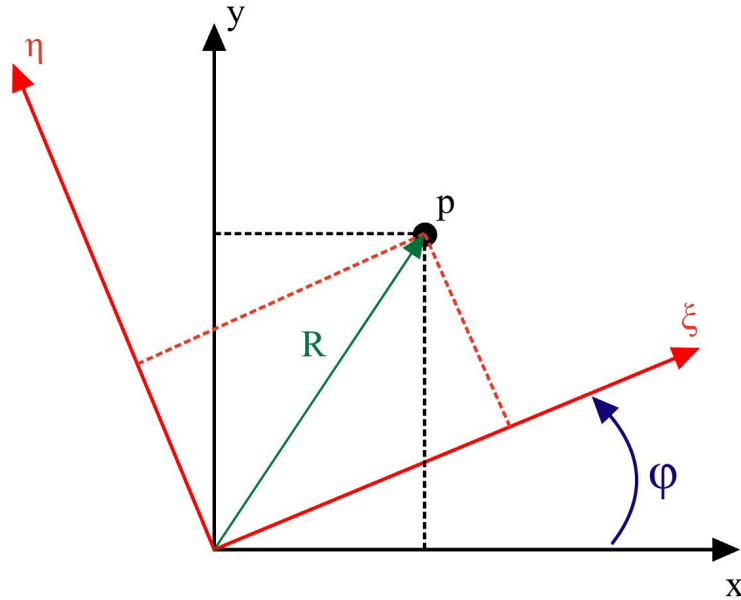


Figure 4.2: Top view of the two reference frames

The components of the vector R (figure 4.2) can be written in $\xi - \eta$ plane by using:

$$\begin{bmatrix} \xi \\ \eta \end{bmatrix} = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (4.5)$$

The third dimension along z -axis is always aligned between the two reference systems, so the Direction Cosine Matrix is a 2×2 for $x - y$ to $\xi - \eta$. It is possible, also, to obtain the vector R in $x - y$ plane by inverting the rotation matrix.

4.3 Matlab implementation

In order to solve the equations of motion, a code was written on Matlab that will be fully reported in the Appendix. This section sets out the steps and functions used to integrate these equations.

In Matlab there is a command, called "lsim" which plots the time response of a dynamic system to the input signal described by a command force and by the time. The dynamic system must be written in terms of state space equations. Thus, the first step is to write of the second-order equation as a set of first-order equations. Below is the example for the first equation in ξ , but the logic can be extended to the other two equations.

The state variables are written:

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \xi(t) \\ \dot{\xi}(t) \end{bmatrix} \quad (4.6)$$

So it is possible to write the first-order system of state equation:

$$\dot{x}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ \frac{F_\xi(t)}{m} \end{bmatrix} \quad (4.7)$$

and the output equation:

$$y(t) = x_1(t) \quad (4.8)$$

Now, writing the matrix representations of the linear system is easy:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} F_\xi(t) \quad (4.9)$$

$$y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + [0] \cdot F_\xi(t) \quad (4.10)$$

Now, the matrix A, B, C, D can be define as follow:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix} \quad D = [0] \quad (4.11)$$

It possible to define the state and the input variables (4.12) and the matrices (4.13) for the 3-DOF case:

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \\ x_5(t) \\ x_6(t) \end{bmatrix} = \begin{bmatrix} \xi(t) \\ \dot{\xi}(t) \\ \eta(t) \\ \dot{\eta}(t) \\ \phi(t) \\ \dot{\phi}(t) \end{bmatrix} \quad F = \begin{bmatrix} F_\xi(t) \\ F_\eta(t) \\ F_\phi(t) \end{bmatrix} \quad (4.12)$$

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{J_z} \end{bmatrix} \quad (4.13)$$

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

The first-order linear system in matrix form is:

$$\dot{x}(t) = [A] \cdot x(t) + [B] \cdot [F] \quad (4.14)$$

$$y(t) = [C] \cdot x(t) + [D] \cdot [F] \quad (4.15)$$

The "lsim" command takes as input:

- The dynamic system formed by the matrices A, B, C, D;

- The matrix F , which represents the input of the system: in the case of *MyDas*:

$$F = \begin{bmatrix} F_{\xi}(t) \\ F_{\eta}(t) \\ F_{\phi}(t) \end{bmatrix} = \begin{bmatrix} F_1(t)\cos\alpha_1(t) - F_2(t)\cos\alpha_2(t) \\ F_1(t)\sin\alpha_1(t) + F_2(t)\sin\alpha_2(t) \\ -F_1(t)l\sin\alpha_1(t) + F_2(t)l\sin\alpha_2(t) \end{bmatrix} \quad (4.16)$$

- The simulation time "T", intended as a vector expressed in the time units of the system and consisting of regularly spaced time samples.
- The initial conditions of the state vector " x_0 ".

Once all the matrices have been defined, the simulation can be run. The output of the simulation is a vector with 6 columns, corresponding to the 6 state variables, and number of rows equal to the duration of the simulation.

4.4 Simulink implementation

Simulink, developed by MathWorks, is a graphical programming environment and simulation tool. It finds widespread application in modeling, simulating, and analyzing dynamic systems across various domains, such as control systems, signal processing, communications, and more.

In Simulink it is possible to model using a block diagram approach; to simulate, in continuous-time or discrete-time using a variety of solvers; to visualize the results; to generate codes, to convert the models into executable code for various target platforms.

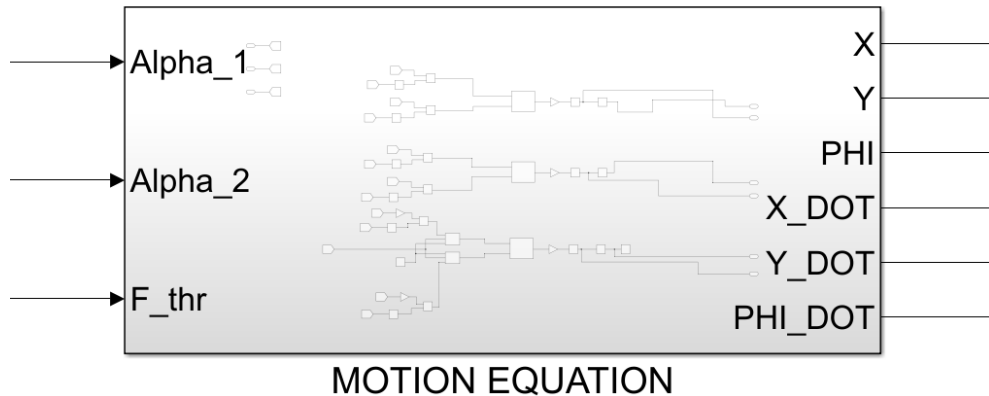


Figure 4.3: Dynamics equations block in Simulink

Simulink, closely integrated with Matlab, enables you to combine the power of Matlab's numerical computing capabilities with Simulink's simulation and modeling environment.

The steps for solving the equations of motion on Simulink are given below.

The block in figure 4.3 has as input the thrust of the thrusters (in terms of intensity and duration) and the angles α_1 and α_2 . As outputs, there are the state variables. At this stage of the modeling, F_{thr} , α_1 and α_2 are freely assigned, to verify that the dynamics is written correctly. Later, instead, they will be the output of the GNC algorithm that will be written.

Within this main block, all operations are performed to integrate the equations and obtain the desired variables.

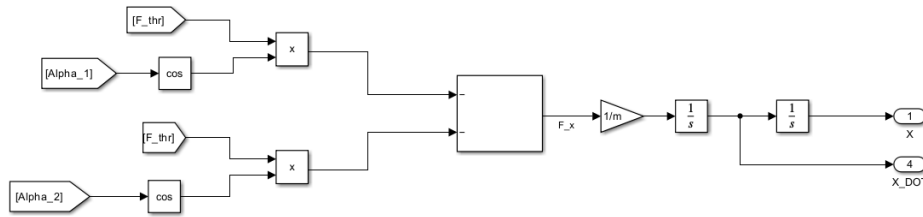


Figure 4.4: Operations to solve equation along ξ -axes

Figure 4.4 shows the double integration of the equation of motion along the ξ direction. Within the main block (figure 4.3) there are two other processes like this, one for the η direction, and one for the ϕ rotation.

On the left, the parameters F_{thr} , α_1 and α_2 are processed to obtain the force component F_ξ (eq. 4.16). So the parameter F_ξ is divided by the mass and integrated twice via the "integrator" block ($\frac{1}{s}$). From the first integration the velocity " $\dot{\xi}$ " is obtained, with the second the position " ξ ". Within the "integrator" block it is possible to set the initial conditions of integration.

At this point, you can run continuous-time simulations with various solvers and set any simulation time.

Each arrow connecting the blocks, in Simulink, represents a signal. These signals can be logged during the simulation and displayed after the simulation, via a "data inspector". Also, comparing signals between two simulation is possible.

As an example, figure 4.5 shows the results of a simulation with the parameters in table 4.1.

In this simulation, the vehicle starts from a static condition, with $\phi = 0$, so the two reference systems are aligned. Then, $\alpha_1 = \alpha_2 = 90^\circ$, so we expect motion along η according to the representation in figure 4.1. The thrusters will fire for 1 second and 10 seconds of time will be simulated.

Description	Parameter	Value
Mass	m	1.7 kg
Leverage	l	0.055 m
Inertia vertical axis	J_z	$2.761 \times 10^{-3} \text{ kgm}^2$
Left valve thrust	F_1	0.110 N
Right valve thrust	F_2	0.110 N
Angle of orientation F_1	α_1	90°
Angle of orientation F_2	α_2	90°
Time of fire	t_f	1 s
Initial Condition State vector	x_0	$zeros^{6,1}$
Time of simulation	T	10 s
Solver		ode4

Table 4.1: Simulation parameters

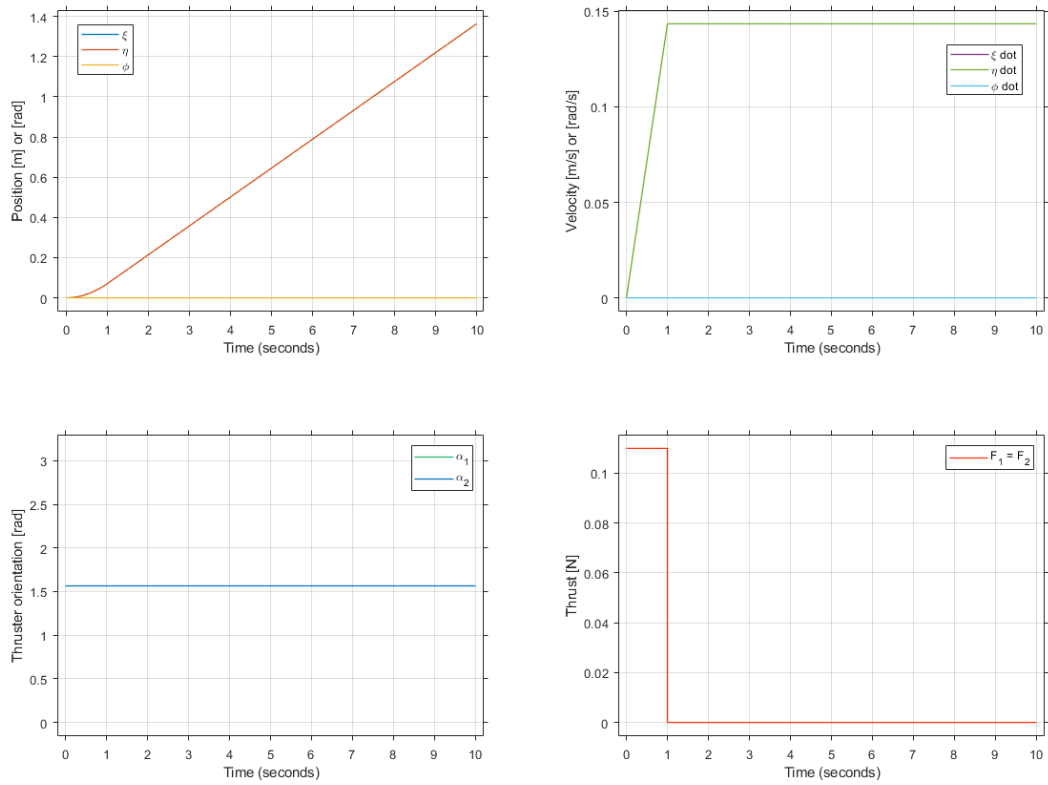


Figure 4.5: Simulation results

The simulation is successful, in fact only motion along η and an increase in velocity along this axis can be seen. The velocity increases for 1 second (firing time) and then stabilizes at that value. As mentioned before, the two reference systems coincide, since $\phi = 0$. Therefore $x = \xi$, $y = \eta$.

4.5 Graphics animation

As mentioned at the beginning of this chapter, the goal of a Digital Twin is also to be able to visualize the actual vehicle digitally. Once the results are obtained from the simulations via Matlab or Simulink, they can be processed in order to obtain a graphical animation that makes understanding the results more intuitive and immediate.

The idea of this project is to visualize *MyDas2* moving on the granite floor. The Matlab code used to create this virtual scene can be found in the appendix. To make the animation, simplify CAD models of the granite floor and *MyDas2* are first realized. So these are imported into Matlab. The "stlread" command is used to create point triangulations and connectivity matrices. So all points are assigned positions obtained from the dynamics simulation. Using a "for loop", an animated plot is created that displays the points for each time instant.

The ability to import external CAD files, greatly increases the *availability* of this project. In fact, it is possible to create planes of any size and replace them in the visualization. In figures 4.6 and 4.7, you can see the *MyDas2* on the granite floor at NPS and the new one in Turin.

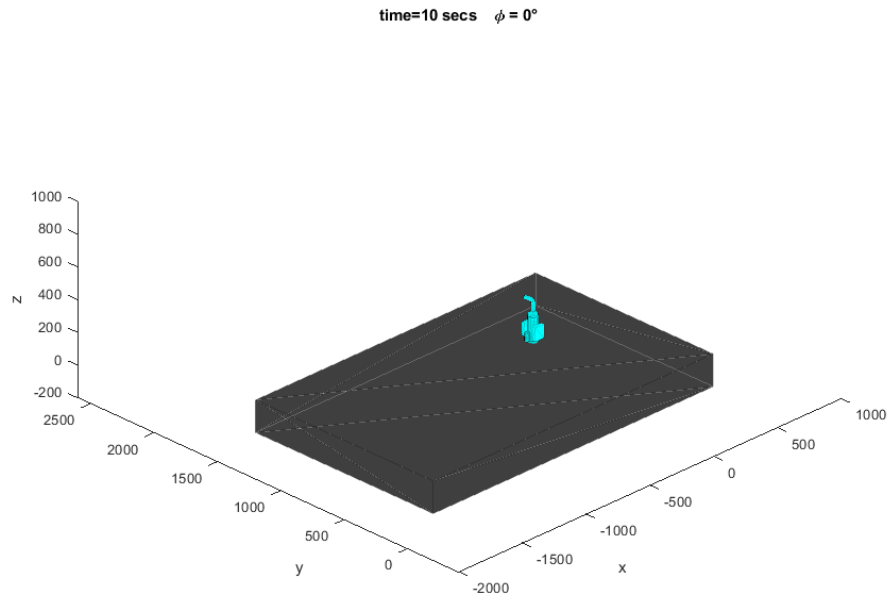


Figure 4.6: Visualization of FSS moving granite floor of POSEIDYN

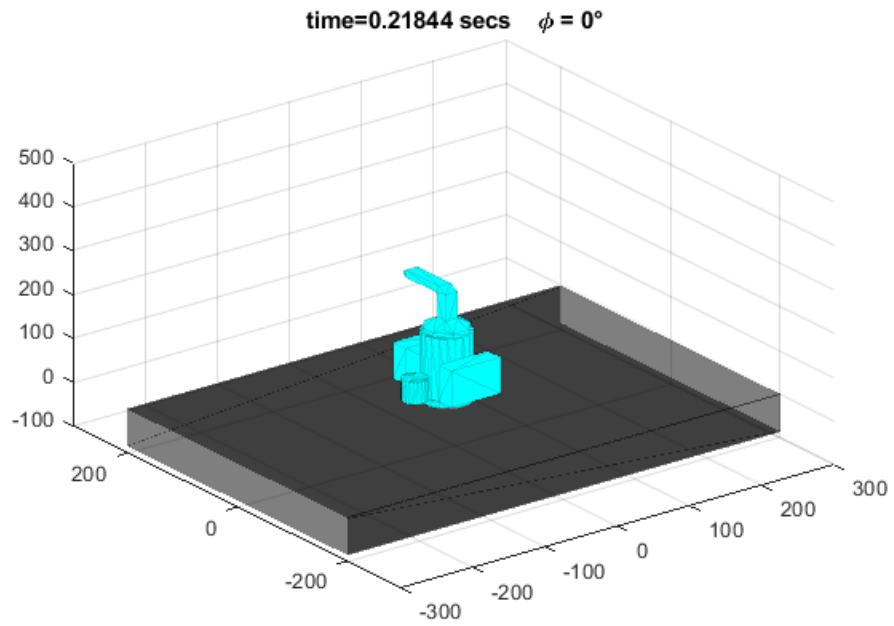


Figure 4.7: Visualization of FSS moving granite floor of Politecnico di Torino

Chapter 5

GNC algorithm implementation

A guidance, navigation, and control (GNC) algorithm is a set of procedures and instructions to control spacecraft behavior. These algorithms are designed to enable a vehicle to reach a specific destination, avoid obstacles, maintain a desired trajectory, and respond to various operating conditions. In this project, the algorithm will be used to perform an attitude maneuver, i.e. reaching a specific attitude angle ϕ , and a position maneuver, i.e. reaching a position in space x, y .

The guidance algorithm focuses on planning the optimal route and determining the necessary actions to guide the vehicle from its current position to the desired destination.

The navigation algorithm is responsible for accurately determining the vehicle's position within its surroundings. This is achieved by using sensors such as GPS, IMU, cameras, lidar, and radar, which collect data on the vehicle's position and orientation. The onboard computer processes this data to estimate the current position and correct any accumulated errors over time.

The control algorithm manages the vehicle's propulsion and control systems to execute the required actions for maintaining the desired trajectory and ensuring a safe and efficient journey. This algorithm takes into account various factors, including vehicle dynamics, motion physics, and operational constraints, in order to generate appropriate control signals for the vehicle's motors, brakes, steering, and other actuators.

In summary, the GNC algorithm integrates route planning, navigation, and vehicle control into a cohesive system that enables autonomous movement, obstacle avoidance, and reliable achievement of the desired objectives.

To better understand how the various parts of a GNC algorithm integrate with each other, it is useful to refer to a block diagram like the one in Figure 5.1.

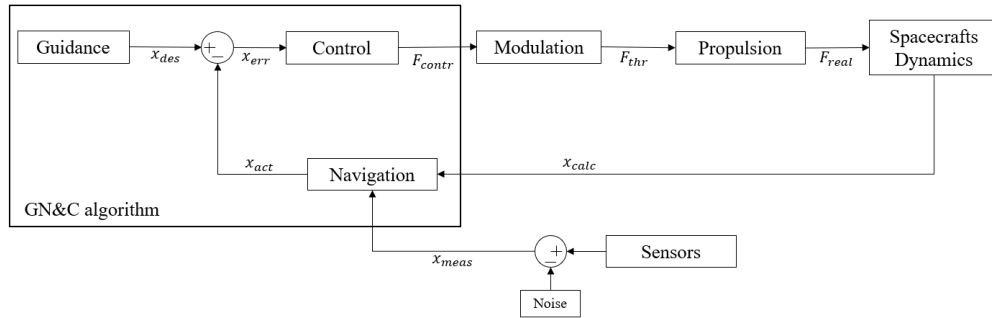


Figure 5.1: Generic GNC block diagram

Starting from the left, there is the guidance algorithm. The goal of the guidance algorithm is to provide a reference vector (x_{des}), that is, to provide a desired position and velocity (linear or angular). There are various types of these algorithms. There are "on-line" and "off-line." The former, take as input the vehicle's actual state vector and return an acceleration that must be impressed on the vehicle to reach a desired position and/or velocity. The output then varies during the maneuver. The latter, merely give the desired position and velocity of the vehicle, and these remain constant throughout the maneuver.

Further down, there is the Navigation algorithm. This is used to provide the on-board computer with the current position of the vehicle. This takes as input data from the sensors (x_{meas}) often subject to noise and, integrating it with data from vehicle dynamics, returns an estimated value (x_{est}).

With the estimated and desired state vector, the error on position and velocity can be calculated. This is what "feeds" the control algorithm. In fact, starting from the error, the algorithm returns a Control Force (F_{contr}). This force will be modulated by a modulator (F_{thr}), and it will allow the vehicle to reach the desired position and velocity. There are many types of control algorithms, more or less efficient and more or less computationally expensive.

The rest of the block diagram, together with the GNC forms what is commonly called the Attitude and Orbit Control System (AOCS) for spacecraft, but for this project it may be called the Attitude and Position Control System (APCS). In other words, it is a system that also includes the presence of the sensors, thrusters, and any other control laws of the vehicle, which is intended to maneuver the vehicle.

Obviously, each APCS is specific to each type of vehicle. In the next section, the APCS for *MyDas2* will be shown and discussed.

5.1 Attitude and Position Control System

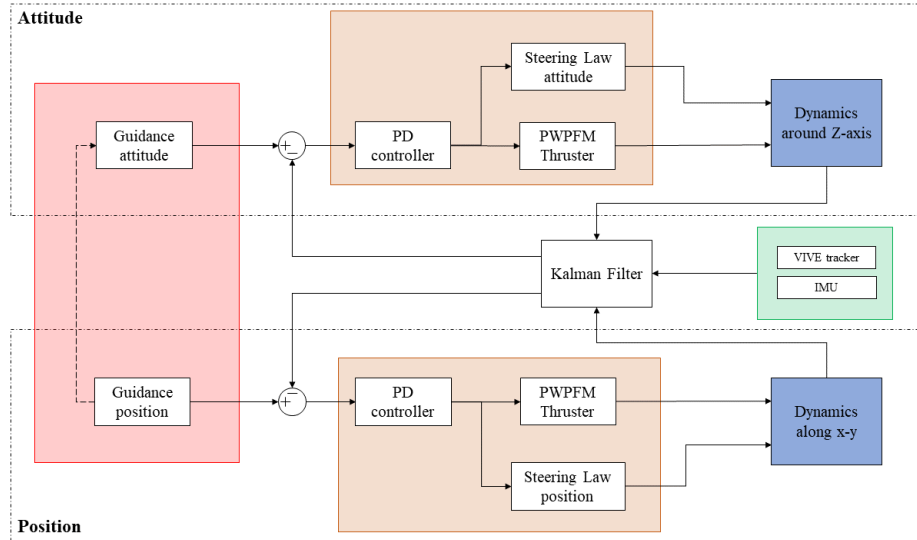


Figure 5.2: Attitude and Position Control System for *MyDas2*

Figure 5.2 shows the APCS block diagram for *MyDas2*. The diagram is divided into two parts: at the top is the part that controls attitude; down, there is position control. Given the nature of the dynamic system, attitude dynamics is decoupled from position dynamics. The reverse is not true, as will be seen later. It is therefore possible to perform isolated attitude maneuvers, but not position maneuvers. An overview of the parts is given below. In the next section all the parts will be discussed in detail with the integration in Simulink.

- **Guidance** In this project, the guidance is "off-line." This means that the vehicle's trajectory will not be calculated instant by instant, but position or attitude coordinates will be imposed, and these will remain constant over time during the maneuver. Actually, when position and attitude control are run together, an algorithm gives to the vehicle the right attitude angle to be achieved to reach the desired position. In the next section, this part will be discussed.
- **Control** Proportional Derivative (PD) controller, Pulse-Width Pulse-Frequency Modulator (PWPFM) and Steering Law control the propulsion of the vehicle. There are two different control algorithms, one for the attitude and another for the position control. Since the thrusters are vectorable, a steering law is necessary to choose the angles of the thrusters. The law for the attitude is different from the law of the position. The PWPFM is an algorithm used

to generate a square wave output signal, where the duration of the pulse is modulated based on the control signal of the PD controller.

- **Motion Dynamics** This block receive data from the "Steering Law" block and from the PWPFM. Basically, the two blocks provide F_1 , F_2 , α_1 , α_2 of the eq. 4.2 - 4.4. The double integrator gives the state vector in terms of position and speed.
- **Sensor** IMU and VIVE tracker provide the vehicle with data on its position in space. Codes will be needed to import the data correctly, and to allow the onboard computer to process it. This data, along with data from vehicle dynamics, will be corrected by a Kalman Filter to obtain an accurate estimate.

The schematic of this block diagram will be reproduced on Simulink. Each block will be considered a "subsystem," within which all the operations necessary to manipulate data and send information to the vehicle will be performed.

The interface between Simulink and Raspberry will be analyzed in the next section.

5.2 Simulink Integration

5.2.1 Support Package

First, the support package that allows Simulink and Raspberry to communicate must be downloaded and installed. Once done, The support package allows you to run algorithms written in Simulink standalone on the Raspberry Pi. In addition, the support package extends Simulink with blocks to drive Raspberry Pi digital I/O and read and write data from them.

After downloading the package, Simulink gives the option of updating the operating system already on the Raspberry, or installing a new one. When installing the new operating system, an SD card must be inserted into the master PC. On this SD card, the operating system will be installed, and once it is inserted inside the Raspberry, it will automatically connect to the Wi-Fi network that was inserted when the new operating system was installed. In addition to the Raspbian operating system, the libraries needed to read and convert code written in Simulink are also downloaded and installed. The steps to install the support package are summarized in figure 5.3.

Once the package is installed, the project can be started on Simulink. Here, the "Hardware" item must be selected in the menu, and the "hardware settings" must be checked (figure 5.4).

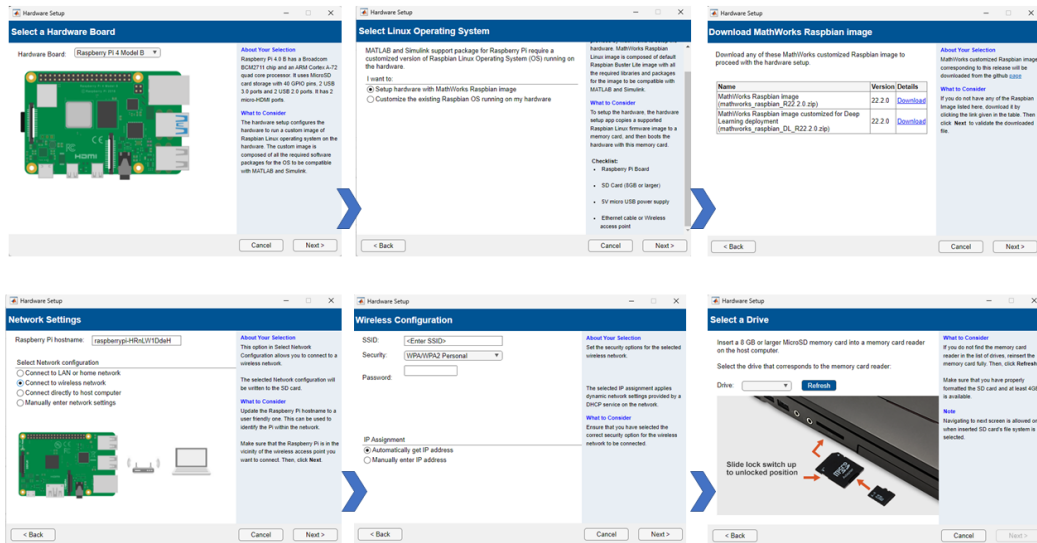


Figure 5.3: Steps to install the Support Package

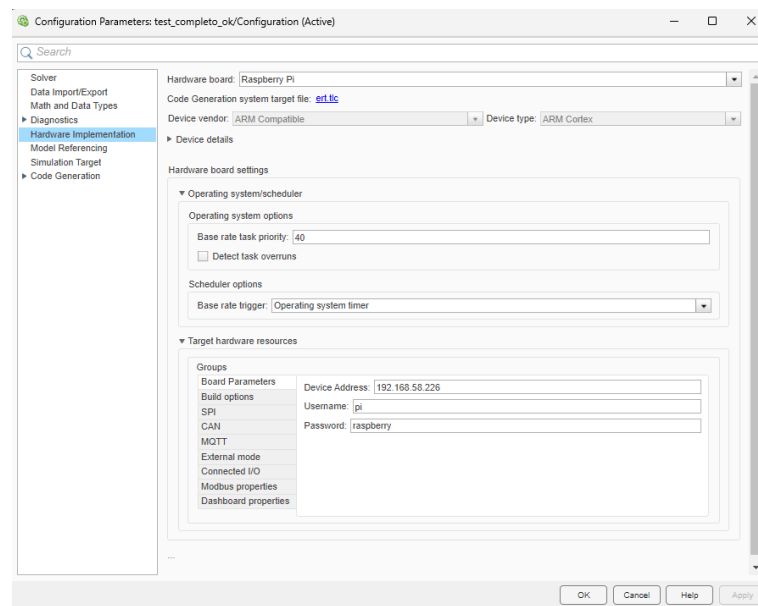


Figure 5.4: Configuration Parameter for the Simulink model

In this menu, the hardware board has to be chosen, by selecting the type, and also adding the "Device Address", username and password. In this menu, it is also possible to set all the parameters concerning the External Mode.

The External Mode is the way in which Simulink communicate with the Raspberry board. The simulation can be run in two way:

- **Run on board (External Mode):** for this project, this way is chosen. In this way, the code is run by the board and the simulation become a Real-time Simulation.
- **Connected I/O (Inputs/Outputs mode):** in this way the Simulink code is run by the master computer, which only communicates with the Pins I/O of the Raspberry. This is a useful methods to check if all the connections are well done, but this is not a Real-Time Simulation, since the time of the Simulink model is different from the real-time.

Then, the new blocks available with the package can be visualized in the Simulink library. Some examples are shown in figure 5.5.

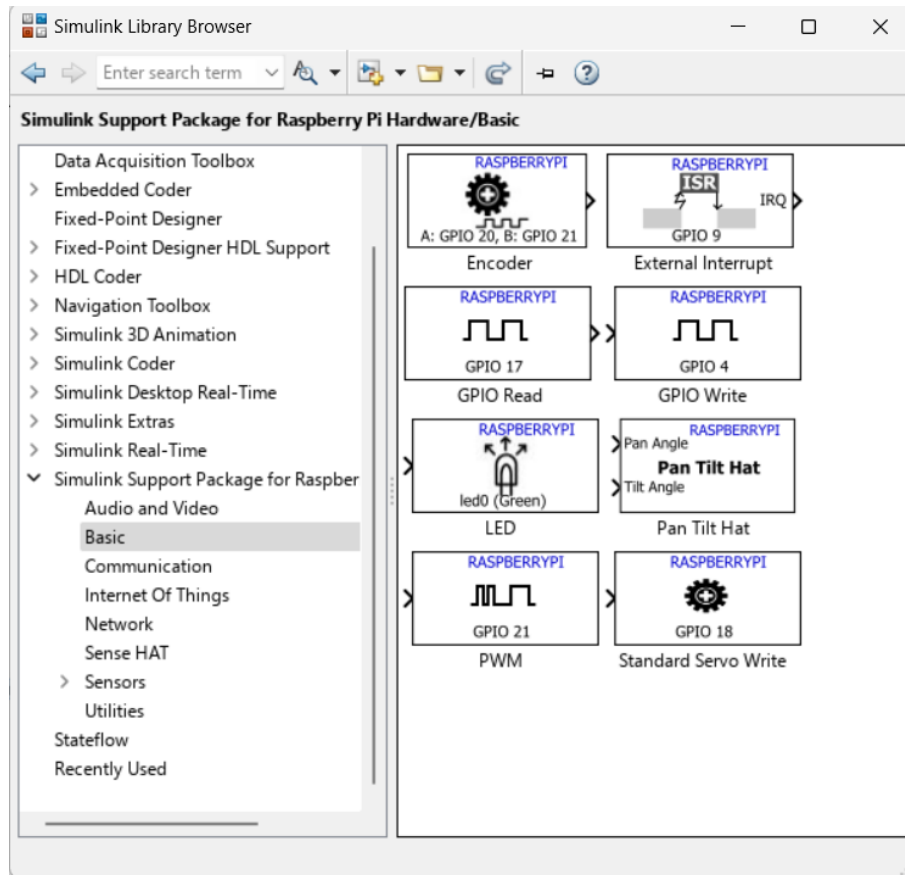


Figure 5.5: Simulink blocks library for Raspberry

5.2.2 Guidance

As mentioned before, the guidance of this GNC algorithm will be of an "offline" type. A position (or velocity) or attitude angle (or angular velocity) is set and this remains constant throughout the maneuver. The desired position or attitude values are set as variables on a Matlab file. On Simulink, on the other hand, the "constant" block is used, where these values are called from Matlab. With this block, Simulink provides the value every iteration, based on the sample time selected.

The desired position-velocity vector is provided to the vehicle in the fixed external reference system. The strategy for reaching the desired point in space is to perform an initial attitude maneuver to rotate the vehicle so that it points toward the desired position. This strategy, is dictated by the fact that the two thrusters are vectorizable, but have an angle range of 180° (from 0 to 180°). Therefore, this configuration does not allow a point in space to be reached without first making an attitude rotation.

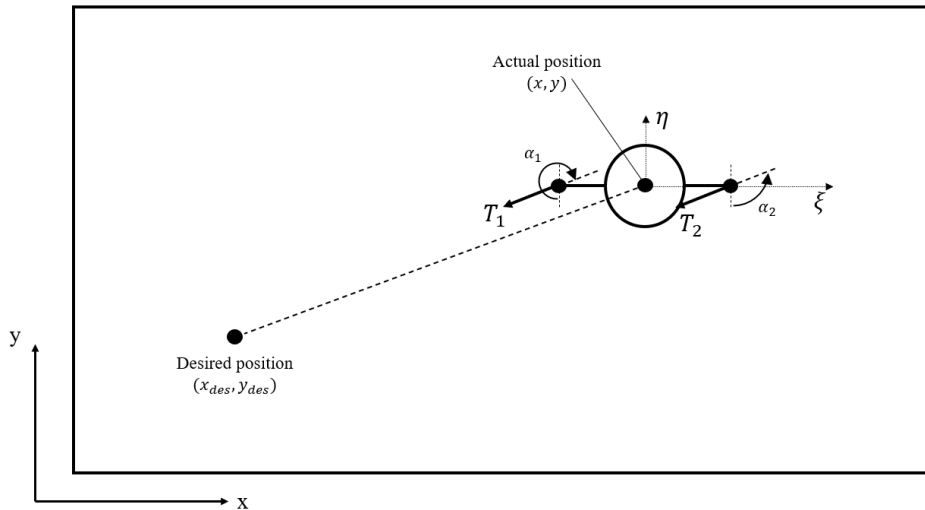


Figure 5.6: Strategy not permitted due to the thruster configuration

In figure 5.6 is represented the position maneuver without initial attitude maneuver. It is possible to see that $\alpha_1 > 180^\circ$. So the thrust T_1 can not be delivered.

The only way to complete this maneuver is to perform an attitude maneuver, in order to align the η - axis with the conjunction of the start and end points.

The configuration in figure 5.7 shows the vehicle which has completed the attitude maneuver. So, the guidance algorithm for the attitude must calculate the right angle ϕ to align η - axis to the conjunction.

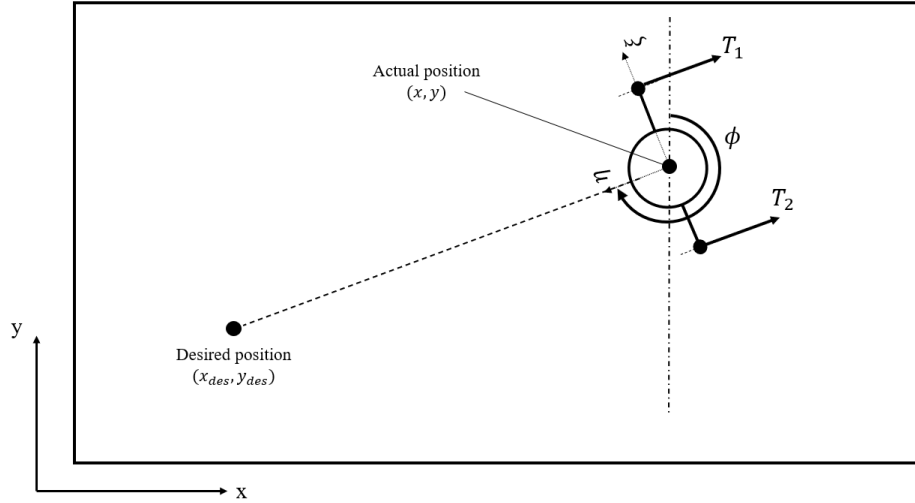


Figure 5.7: Correct strategy with attitude maneuver before position maneuver

By performing some trigonometric calculations, it is possible to write:

$$\phi = \arctan \frac{(x_{des} - x)}{(y_{des} - y)} + \pi \quad (5.1)$$

To solve this equation "atan2" command is used. This command in Matlab returns the angle in radians between the positive y-axis and a specified point in the xy-plane. Unlike the "atan" function, which only takes the x/y ratio, the "atan2" command allows determining the correct angle even for values of x that are negative or zero. It also avoids division by zero issues.

In the situation described in figure 5.7, the command "atan2" would return a negative angle. Since an angle between 0 and 360° is needed to calculate the error and feed the control algorithm, a Matlab Function is inserted to correct the data. Essentially, if the calculated angle is negative, the value of 2π will be added to it, if not it is left unchanged. The Matlab code is given in appendix.

At this point, the guidance algorithm for the attitude maneuver provides an angle ϕ_{des} .

After the attitude maneuver is completed, it can be seen that the vehicle can move straight in the direction of the desired point. In this way $\alpha_1 = \alpha_2 = 0$, so the maneuver is feasible.

So, the guidance algorithm is "off-line" in the sense that desired position and attitude is constant during the maneuver. Actually, due to the configuration of the thrusters, an "on-line" guidance algorithm is needed to perform correctly the position maneuver.

Figure 5.8 shows the Simulink blocks used to obtain the right ϕ angle.

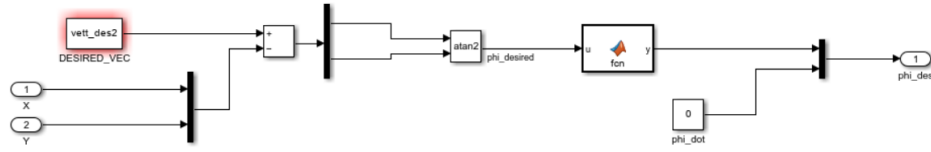


Figure 5.8: Guidance algorithm to obtain ϕ_{des}

The block "DESIRED-VEC" takes the position vector (x, y) from Matlab main and keeps it constant during the simulation. X and Y are data from the sensor and represent the actual position of the vehicle.

5.2.3 Importing sensor data

The control algorithm is fed by the error on the position and velocity of the vehicle. As mentioned before, this error is generated between the state vector provided by the guidance algorithm (x_{des}), and that obtained from reading data from the sensors (x_{act}). In this subsection, the method of data gathering from the sensors is exposed.

IMU The Inertial Measurement Unit (IMU) provides linear acceleration and angular velocity of the vehicle. Figure 5.9 shows the pins used to connect the device.

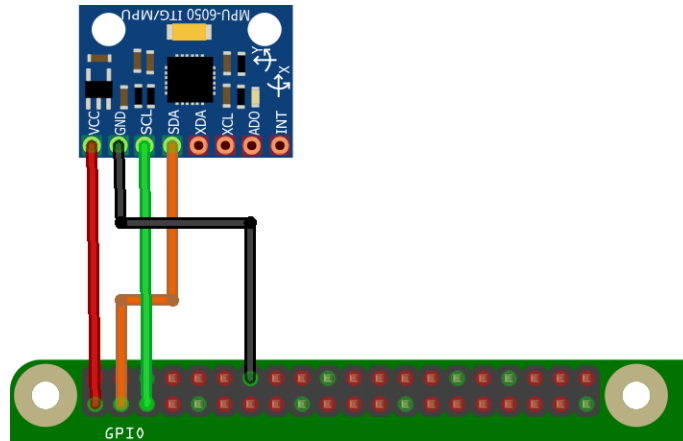


Figure 5.9: Schematic diagram of the MPU-6050 connection

This IMU device uses a I^2C protocol to communicate with the on-board computer.

The SDA (Serial Data) and SCL (Serial Clock) pins are two pins used in this type of communication. The SDA pin is used to transmit bidirectional data between the MPU-6050 and the Raspberry.

The SCL pin is used to provide a synchronization clock for I^2C communication. This clock signal is generated by the master device and is used to synchronize the data transmission on the SDA pin. The SCL signal determines the timing of the I^2C communication.

The other two pins are "VCC" and "GND." The former provides 3.3 V to the device, and the latter is the connection to ground.

Once the device is connected to the OBC, code must be written to modify the raw data. First, the I^2C protocol must be enabled in Matlab. Matlab has a library which allow the communication with the MPU-6050. After the code is run (appendix A) , the I^2C protocol is enable and Master and Slave devices are defined (figure 5.10).

```

Command Window
>> sensor=mpu6050(mydas)

sensor =

mpu6050 with properties:

                I2CAddress: 104 ("0x68")
                Bus: i2c-1
Show all properties all functions
fx >>
    
```

Figure 5.10: I^2C protocol enabled by a Matlab command

Then, Simulink code can be written. In the figure 5.11, there is the subsystem created to import data from IMU.

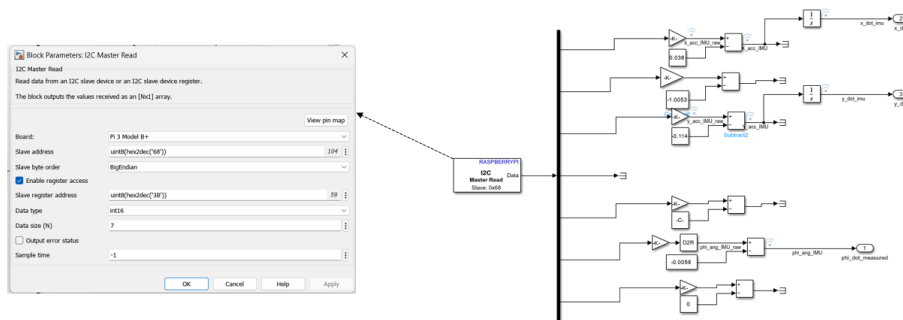


Figure 5.11: Simulink subsystems for IMU data import

After the I2C protocol has been enabled, a block from the Raspberry library is used. The block is called "I2C master read." As the description says, this block "read data from an I2C slave device or an I2C slave device register" and "the block outputs the values received as an [Nx1] array." From the data-sheet [12] of the MPU-6050, we can read that the device provides a vector containing 7 parameters (3 linear accelerations + 3 angular velocities + 1 sample time). The Sample Time is set equal to -1, which means it is equal to the one set for the whole model. "Slave address" and "Slave register address" are taken from the data-sheet.

Then, this block reads the data from the device and provides it to the on-board computer. Now, this data needs to be further corrected. First, the linear acceleration and angular velocity data, must be corrected, dividing the measured values by a certain Sensitivity Scale Factor, based on the selected Full Scale Factor.

For the accelerometer, the standard Full Scale Factor is 2 g. With this value, it is necessary to divide the measured values by the factor $q = 16384$. For the gyroscope, the Full Scale Factor is 250 °/s which implies a factor $p = 131$, to be divided by the measured value.

After this initial correction, further adjustment of the data is necessary. In fact, in these IMU platforms there is often a constant offset on the measured data. A test is then performed, with the device kept stationary, and the data for accelerations along $x - y$ and angular velocity around the z axis are measured.

Being stationary, it is expected that the average of the measured values will be as close to zero as possible. In figure 5.12 - 5.13 it is possible to observe the measured data raw and corrected.

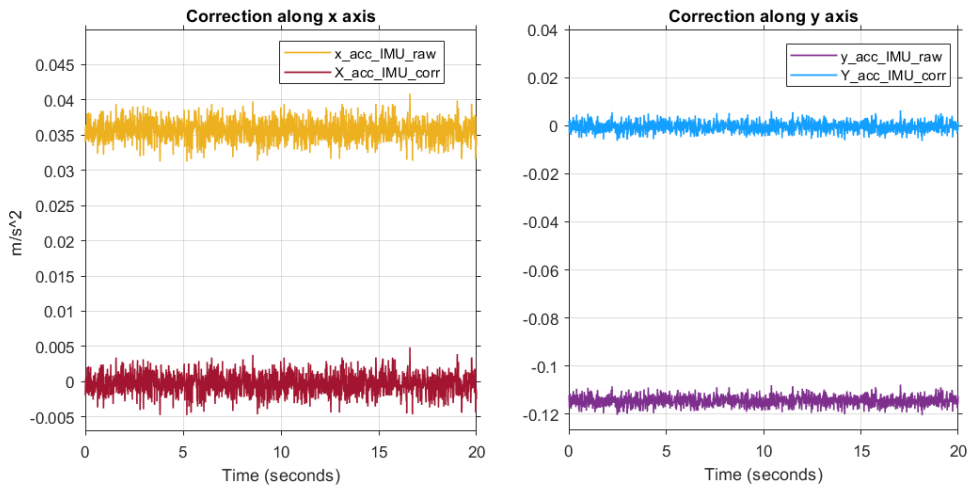


Figure 5.12: Correction of offset error for x-y linear acceleration

Then, after performing the test, data are collected for each axis and the values are mean averaged. At the next test, the average obtained is added to the new

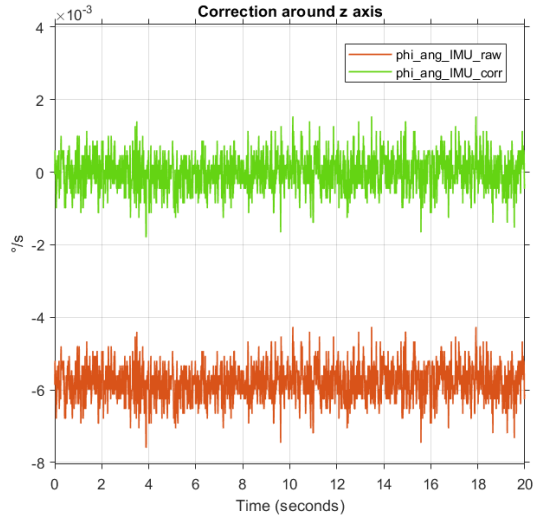


Figure 5.13: Correction of offset error for $\dot{\phi}$ angular velocity

measured values. As you can see, the average value of the measurement is brought to the value 0. The table below shows the precise readings obtained.

	Raw mean	Corrected mean
\ddot{x}	0.0357	-3.1252e-04
\ddot{y}	-0.1144	-3.9727e-04
$\dot{\phi}$	-0.0058	-9.0058e-06

Table 5.1: Accelerometer and gyroscope correction

It can be seen that the acceleration value along y is the one with the most offset, this is because the acceleration of gravity acts along that axis and disturbs can disturb the measurement. However, after correction this turns out to have a comparable mean value with that along x . Angular velocity values, on the other hand, turn out to be less subject to offset, by two orders of magnitude. This may be because in general IMU sensors, have higher sensitivity for angular velocity, and in any case this is not affected in any way by gravity acceleration. After correction, the value of angular acceleration turns out to be the most accurate.

In this way, this code provides \ddot{x} , \ddot{y} and $\dot{\phi}$ to the guidance algorithm. Actually, the control algorithm need also to be fed by the linear velocity. One way to obtain linear velocity measurements from accelerations is to integrate the calculated

acceleration values. The problem is that IMU platforms usually have very large integration errors (accelerometer bias), which must be corrected with Kalman Filters. In Figure 5.14 are measurements for velocities \dot{x} and \dot{y} using the Simulink "integrator" block. The results show a cumulative error over time. The test was carried out with the IMU board static on a flat floor.

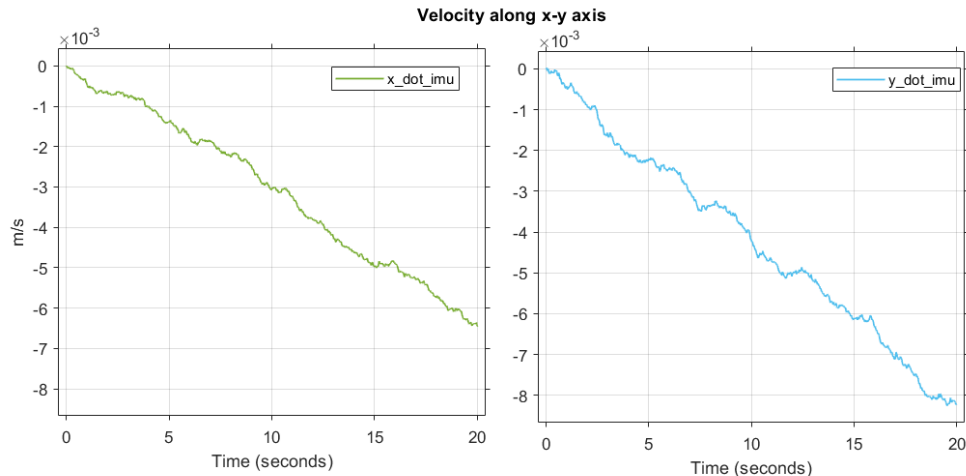


Figure 5.14: Accelerometers Bias

HTC VIVE tracker The HTC Vive Tracker is an accessory developed by HTC for use with the HTC Vive virtual reality (VR) system. It is a small, puck-shaped device that can be attached to physical objects, such as game controllers, props, or even human body, to track their movement and bring them into the virtual world. It works in conjunction with two Base Station and the Vive Pro VR.

The tracker communicates with Base Stations, which provide the location in space to the tracker. This sends the data, via Bluetooth, to a device, connected via USB to a master computer. The master computer is not the vehicle's onboard computer. On the master computer, the data is processed by Unity software, which extracts the data from Steam VR and sends it, via a UDP communication to the onboard computer. Figure 5.15, shows the architecture just described.

Unity sends data to the onboard computer via UDP communication. UDP (User Datagram Protocol) is a communication protocol that operates at the transport layer of the Internet Protocol Suite. It is a connectionless protocol, which means it does not establish a dedicated and reliable connection between the sender and receiver before sending data.

Unlike TCP (Transmission Control Protocol), which provides reliable and ordered data transmission, UDP focuses on simplicity and efficiency. It offers a "best-effort"

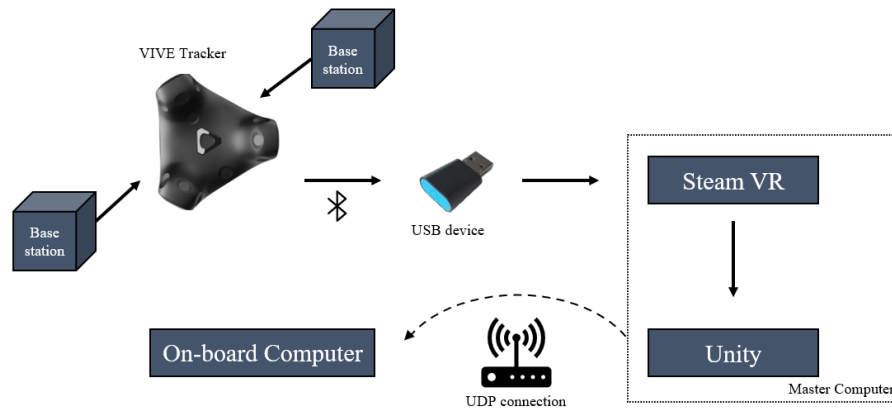


Figure 5.15: Hardware and software architecture for VIVE tracker

delivery mechanism, where packets, called datagrams, are sent from the sender to the receiver without any guarantee of delivery or order.

Here are some key characteristics of the UDP:

- **Connectionless:** UDP does not require a handshake or connection setup before transmitting data. Each UDP datagram is treated independently and can be sent without prior coordination.
- **Unreliable:** UDP does not provide acknowledgment of packet delivery, re-transmission of lost packets, or error checking. If a UDP datagram is lost or damaged during transmission, it will not be automatically recovered.
- **Low overhead:** UDP has a smaller header size compared to TCP, which reduces the amount of additional data sent over the network.
- **Fast transmission:** Due to its simplicity, UDP has less processing overhead, making it faster than TCP in terms of data transmission speed.

UDP turns out to be very useful for this project, since a real-time communication and low latency are more important than reliable data delivery.

To send data from Unity, a C code was written. It is given in the Appendix A. This code, send a vector of 7 elements: 3 position along the axis, 3 rotation around the axis, and sample time. On-board computer will take only 2 position (x, y) and 1 rotation (ϕ). The code packs the data and each elements is sent as "double". A "double" is a data type used to represent floating-point numbers with double precision. Each element "double" is represented using 8 bytes, so a total of 56 bites packed are sent by the Unity code.

Then, a code in Simulink to receive data is written 5.16.

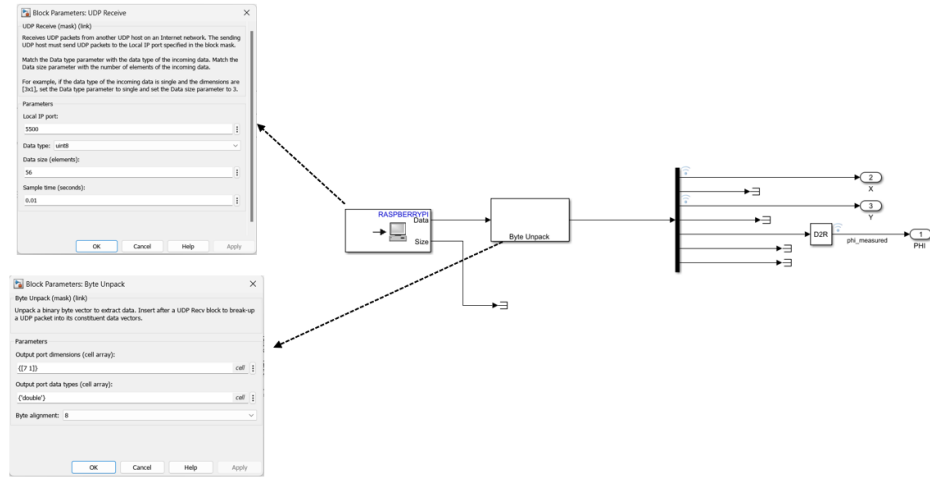


Figure 5.16: Simulink subsystem for HTC tracker data import

A first block, called "UDP receiver" is used to enable the UDP communication on the Raspberry. Master computer and onboard computer must be connected to same network. In the Unity code, the IP address of the Raspberry is selected and also a "local IP port". This number must be used in the "UDP receive" block and also the number of elements to be received is entered. Now, the data are received packed from the Unity code. A "byte unpack" block is used to unpack the data and to put them in a vector with 7 elements.

Actually, the position along x and y must be correct, since the tracker is not located exactly in the center of the vehicle (figure 5.17). Therefore, a function is implemented in Matlab (given in the Appendix), which, taken as input x , y and ϕ , returns the correct (x,y) position. Mathematically it is written:

$$\begin{aligned} x_{corr} &= x - L\cos(\phi) \\ y_{corr} &= y - L\sin(\phi) \end{aligned} \quad (5.2)$$

Basically, when the vehicle performs an attitude maneuver, position x and y must be constant. Some tests were carried out to verify the correctness of the code. The vehicle performs a 360° rotation maneuver, and x and y data were then collected before and after the correction. Figure 5.18, shows the results.

It can be seen that before the correction, during a rotation, x and y varied about 8 cm from the actual position. After correction, the total variation over the 360° rotation is only 1 cm. The calculated x and y positions are therefore much more accurate.

Finally, x , y and ϕ are ready to be used by the control algorithm to generate the control force.

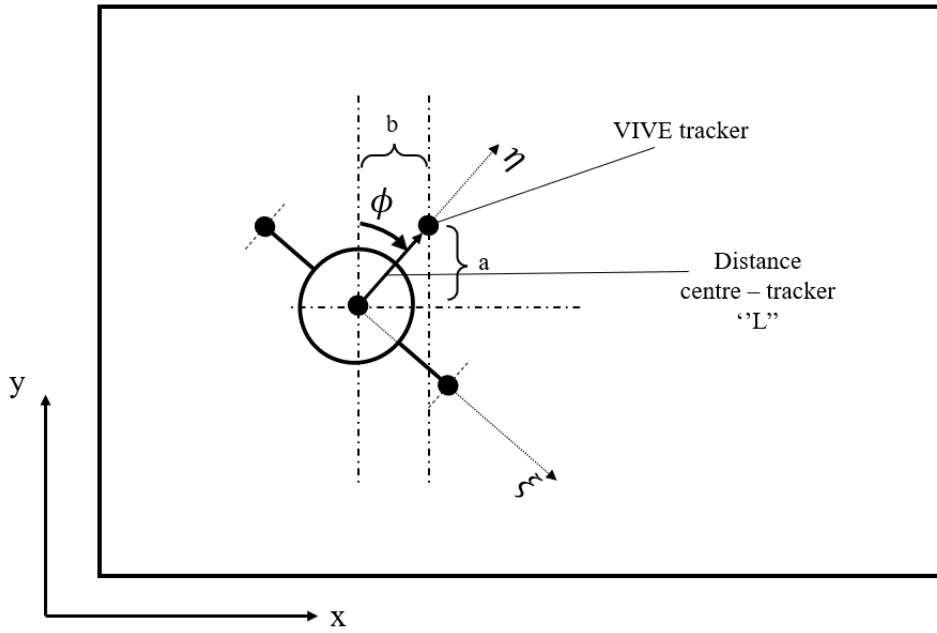


Figure 5.17: Position of the tracker relative to the geometric center of the vehicle

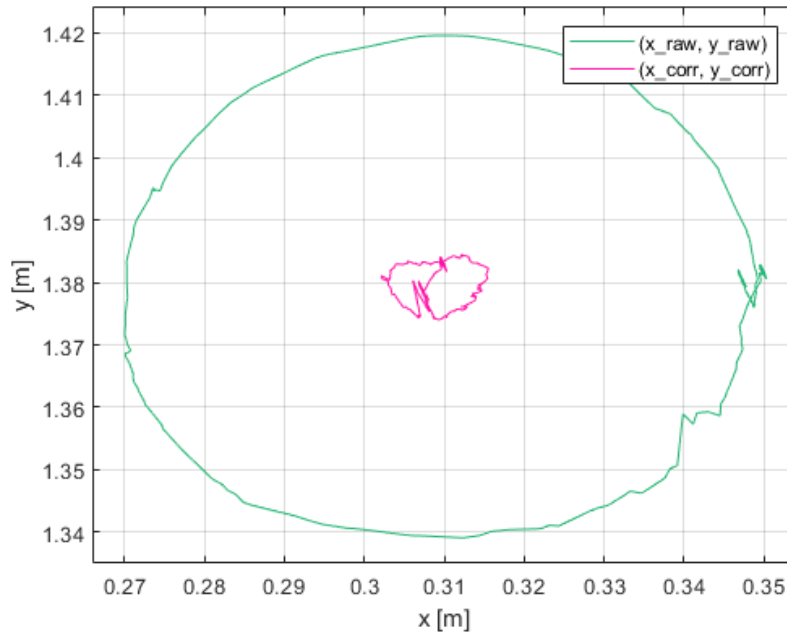


Figure 5.18: Experimental results for x and y position correction

5.2.4 PD controller

At this point, the on-board computer receives data from the sensors, which provide the current position (and velocity) of the vehicle. Guidance, on the other hand, provides the desired position (and velocity). By performing the difference between these two vectors, the error vector is obtained. This vector will be the input to the control algorithm.

For this project, a Proportional Derivative (PD) type controller was developed. Such a controller, is a type of Proportional Integrative Derivative (PID), in which there is no Integrative.

PID controllers are the first control algorithms made. The first example of PID-type was developed in 1911, while the first theoretical publication occurred in 1922. PID control is based on past error (integrative), present error (proportional) and prediction of future error (derivative). In general PD controllers follow this equation:

$$u(t) = K_P e(t) + K_D \dot{e}(t) \tag{5.3}$$

where:

- $\mathbf{u}(t)$ is the control variable, i.e. the output of the control algorithm; for this project, the control variable is the thrust of the thruster:

$$u(t) = F(t) \tag{5.4}$$

- $\mathbf{e}(t)$ is the actual error vector (position) and $\dot{\mathbf{e}}(t)$ is the derivative of the error vector (velocity). So:

$$e(t) = \begin{bmatrix} x_e \\ y_e \\ \phi_e \end{bmatrix} \quad \dot{e}(t) = \begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\phi}_e \end{bmatrix} \tag{5.5}$$

- $\mathbf{K}_P, \mathbf{K}_D$ are the gains. By varying these parameters, the time response of the system changes, as the force $u(t)$ changes. Referring to the figure 5.19, the table 5.2 how the response changes according to K_P and K_D .

	t_r	O_v	t_s	e_s
Increasing K_P	decrease	increase	small increase	decrease
Increasing K_D	small decrease	decrease	decrease	minor change

Table 5.2: Response characteristics according to K_P and K_D adjustment

The adjustment of this parameters is the tuning of PD controller. Initially, the tuning is reach by trial and error, until the closed loop system performs

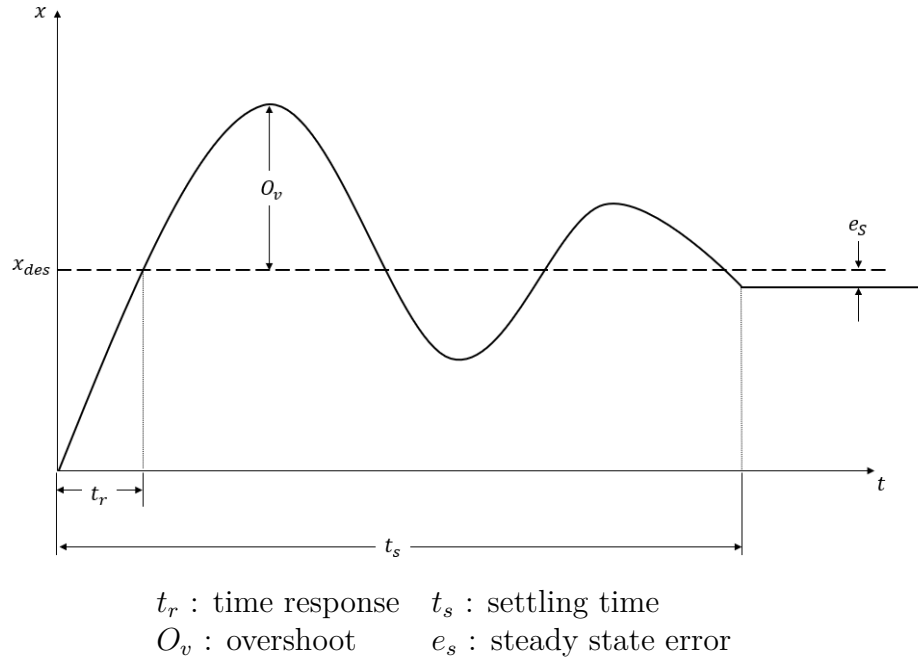


Figure 5.19: Key characteristics of a dynamics system time response

as desire. There are also mathematical models to tune the control algorithm. These models are used when the plant is complex and it is difficult to obtain mathematical model of the system.

In the Simulink code, there are two blocks for the PD controller. One is for the attitude control, the other one is for the position control, since the two maneuvers happen in different time.

The block for the attitude maneuver takes as input the vector:

$$\phi_{err} = \begin{bmatrix} \phi \\ \dot{\phi} \end{bmatrix} \quad (5.6)$$

and gives as output the force $F_{command}$.

The code is very simple, as well as the equation 5.3. A "demux" block takes the single element of the vector (5.6) and, with the block "gain", these elements are multiplied one by the K_P and one by the K_D constant. Then there is the sum. In this way, the control force is obtained, as it possible to see in figure 5.20.

For the Position Control, the scheme is the same, but the input vector is:

$$x_{err} = \begin{bmatrix} \xi \\ \dot{\xi} \\ \eta \\ \dot{\eta} \end{bmatrix} \quad (5.7)$$

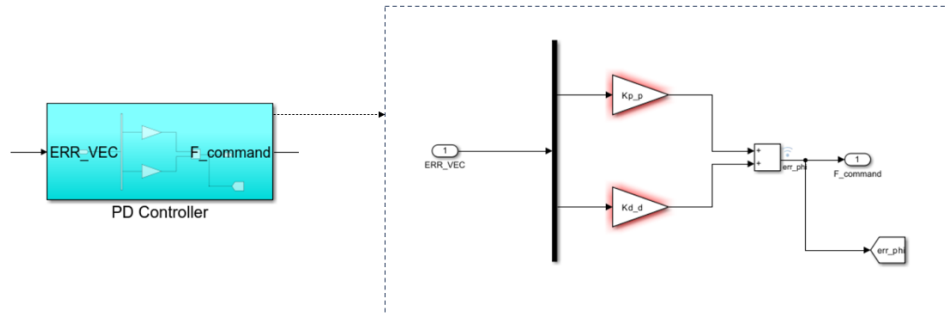


Figure 5.20: Simulink block for PD controller

and the output is $F_{command}$.

5.2.5 Steering Law

Steering law refers to the series of operations aimed at correctly directing the thrusters to obtain thrust in the correct direction. The direction of the thrusters is controlled by servos. The servos allow 180° rotation to the thrusters. Since there are only two thrusters and they have this range of motion, they will always act simultaneously. In fact, the use of only one thruster would result in too great a misalignment from the center of mass and thus in uncontrollability of the vehicle. The steering law takes as input data from the Control Law and outputs two angles. The angles then enter the block of the equations of motion, to simulate the maneuver, or force the servos to reach those angles to perform the actual maneuver.

Figure 5.21 helps to understand the steering law logic.

There are different steering laws for attitude maneuvering and position maneuvering. The basic logic on which they are based is the same, but the resulting angles will be different.

On the left, there are two situations that may arise during the position maneuver, while on the right it relates to the attitude maneuver. On the left, based on the position of the vehicle relative to the desired position, the control algorithm returns a positive or negative $F_{command}$ value. The steering law investigates the sign of $F_{command}$ and returns angles as in the figure, to deliver thrust in a given direction. T_1 and T_2 represent the thrust delivered by the thrusters, and always have the same value between them. It is possible to see, that in the position maneuver, the angles α_1 and α_2 always have the same value to each other. On the right, in contrast, there is the attitude maneuver. As mentioned above, the logic is the same as in the position maneuver: the steering law investigates the sign of $F_{command}$, which depends on the difference between ϕ_0 and ϕ_{des} . In this case, however, in order to

obtain pure rotation, the angles α_1 and α_2 are opposite to each other.

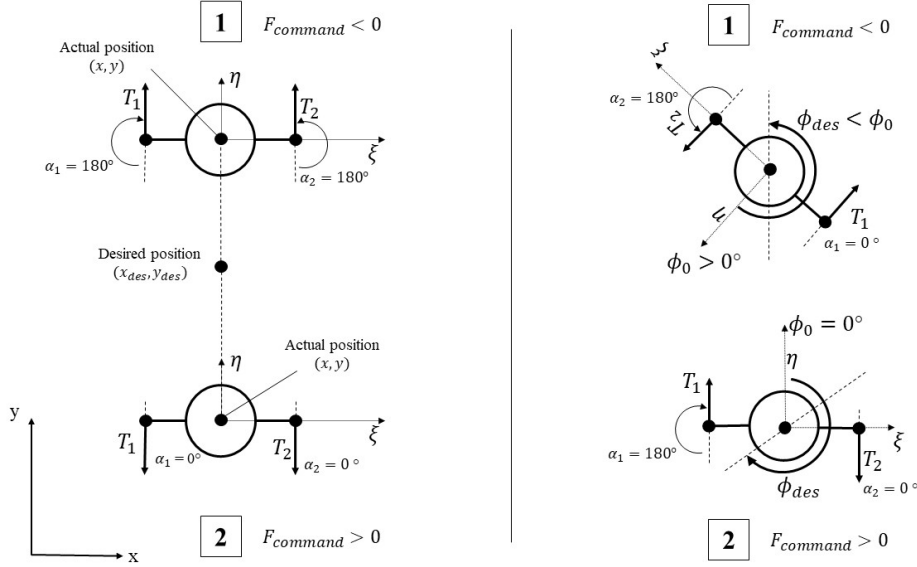


Figure 5.21: Steering Law logic scheme

Theoretically, in this way, the maneuvers are decoupled. Mathematically, for position control it is:

$$\alpha_1 = \begin{cases} 0^\circ & \text{if } F_{comm} > 0 \\ 180^\circ & \text{if } F_{comm} < 0 \end{cases} = \alpha_2 \quad (5.8)$$

For attitude control it is:

$$\alpha_1 = \begin{cases} 180^\circ & \text{if } F_{comm} > 0 \\ 0^\circ & \text{if } F_{comm} < 0 \end{cases} \quad \alpha_2 = \begin{cases} 0^\circ & \text{if } F_{comm} > 0 \\ 180^\circ & \text{if } F_{comm} < 0 \end{cases} \quad (5.9)$$

In Simulink the code is shown in figure 5.22. The block "sign", takes as input the $F_{command}$ and outputs 1 for positive input, -1 for negative input, and 0 for 0 input. As it possible to notice, the angles α_1 and α_2 do not vary between 0° and 180° , but between $-\pi/4$ and $\pi/4$. This happens for two reasons. The first, is that the equation of motions need a range from $-\frac{\pi}{2}$ and $\frac{\pi}{2}$, where $-\frac{\pi}{2} = 0^\circ$ and $\frac{\pi}{2} = 180^\circ$ referring to figure 5.21. This value will be corrected to enter the communication block with the servos, which need an angle between 0° and 180° . The second concerns the thrust provided by thrusters. After initial tests, in fact, it was noticed that the thrust was oversized for the weight of the vehicle. In order to achieve smoother maneuvers, therefore, it was chosen to direct the thrust at smaller angles, so as to have less effective thrust during maneuvers. Clearly, this is not the optimal

choice in terms of propellant consumption, but for an initial control test it can be accepted.

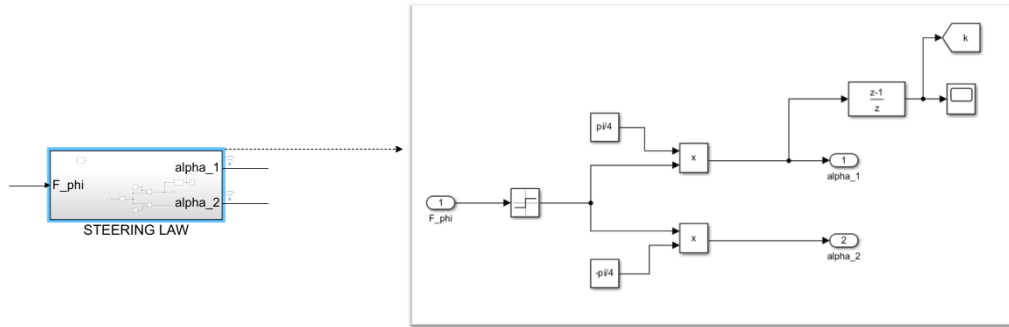


Figure 5.22: Simulink blocks for steering law

Only the Simulink block for the attitude maneuver is shown in the figure 5.22. For the position maneuver, the logic is the same. It is important to note that the position maneuver occurs only along the η axis of the vehicle. This is because, as mentioned at the beginning of the chapter, the position maneuver occurs after the vehicle has performed the rotation maneuver and aligned η to the conjunction with the desired point.

5.2.6 Thrust Modulation

The control algorithm outputs a force that varies continuously over time. Thrusters, however, take on only two discrete states: on or off. To use the thrusters, the continuous force then must be modulated [13]. The Schmitt trigger is one of the simplest thruster control method, often described as a relay with hysteresis and a deadband. It establishes its minimum pulse-width based on the changing inertia of the spacecraft [14]. Although not classified as a pulse modulator, this control technique can be extended using the pulse-width pulse-frequency modulator (PWPFM). The PWPFM and derived-rate modulator incorporate a first-order lag filter in either the feedforward or feedback paths, respectively. Another alternative is the pulse-width modulator (PWM), which exhibits similar behavior to a PWPFM but is simpler in construction. Unlike the Schmitt trigger, the static characteristics of the PWM, PWPFM, and derived-rate modulator are not influenced by the spacecraft inertia, making them advantageous options. Consequently, the PWM, PWPFM, and derived-rate modulator have found widespread use in spacecraft thruster modulation. In this work, a PWPF modulator will be used.

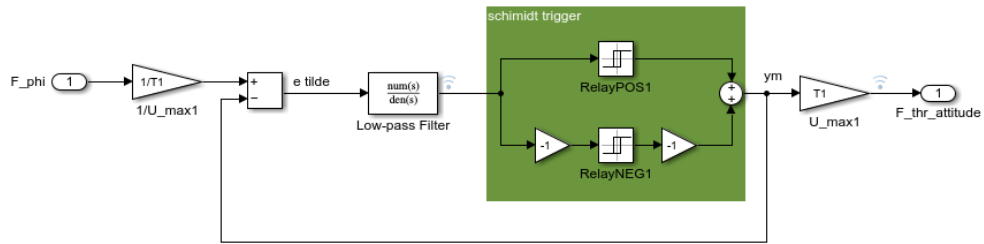


Figure 5.23: Simulink block for PWPFM

The main element of this modulator is the Schmitt Trigger, which consist of a double relay with hysteresis, separated by a dead band. In order to provide a quasi-linear steady-state response, a low pass filter is add. The input of the filter is the difference between the control signal and the modulator output, and it is represented by a first order system ($\frac{K_f}{\tau s + 1}$). The output of the signal is the Schmitt Trigger activation signal. The output of the modulator remains zero until this signal remains below the activation threshold U_{on} . If the signal exceed this value, the relay is on and gives 1 as output. The relay stays on until the input drops below the value of the switch off point, U_{off} . When the relay is off, it gives 0 as output.

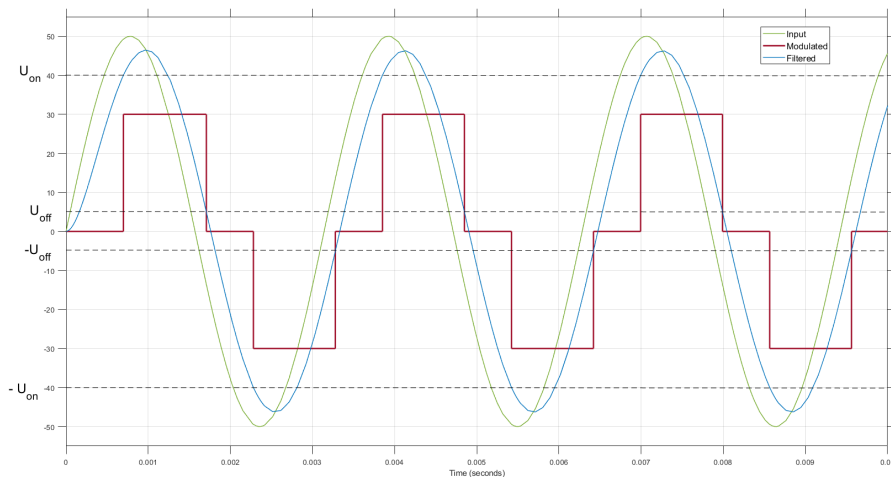


Figure 5.24: Example of filtered and modulated signal

Figure 5.24 shows an example of modulation of a sinusoidal signal. In red, it is

possible to see the signal modulated. It fluctuates between set values. In this way, a discrete value can be provided to the thrusters. In this work, that value oscillates between 0 and 1, which results in on or off for the solenoid valves.

So, the PWPF modulator parameters to be defined are:

- $U_{on}, U_{off}, \tau, K_f$ are respectively activation threshold, deactivation threshold and low pass filter first order system parameters.
- **Dead band:** it is defined as:

$$e_{db} = \frac{U_{on}}{K_f} \quad (5.10)$$

this parameters is useful for some measured tag values which can fluctuate rapidly above and below a critical threshold. The dead band is a buffer to prevent this fluctuations.

- $\Delta t_{on} = t_{off} - t_{on}$ is the fire duration of the thruster. It can be defined also as:

$$\Delta t_{on} = -\tau \cdot \ln\left(1 - \frac{U_{on} - U_{off}}{U_{max}K_f}\right) \quad (5.11)$$

In the next chapter, the values of all these parameters will be given.

5.2.7 Link with On-board Computer

Now that the GNC algorithm is complete, it is necessary to enable communication with the propulsion components. This involves then imposing a desired angle on the servos, and indicating values 1 and 0 (on or off) to the solenoid valves. As mentioned earlier, the support package enables communication directly with the PINs on the raspberry.

The block in Simulink takes as input α_1, α_2 and F_{thr} both for position and attitude maneuver. This block does not have any output, since it communicates with the Raspberry. Each PIN in the Raspberry, has a corresponding number that identifies it. For solenoid valves, the corresponding numbers for GPIOs are 20 and 21. For servos, the PINs are 17 and 18. Using the blocks in figure 5.25, the GPIO pin number can be entered. The input signal will then be sent to the chosen PIN.

For the thrusters, the signal is 1 or 0, which means valves opened or closed. For the servos it is different. The position of the servos is controlled via PWM (Pulse Width Modulation). The basic principle of PWM for servo motor control involves sending a series of pulses with varying widths to the servo motor. The width of each pulse determines the position of the servo motor's shaft. The SG90 motors chosen for the *MyDas2* work with a control signal whose pulse width varies between 1 to 2 milliseconds (ms) [15]. The control signal consists of a series of pulses that are

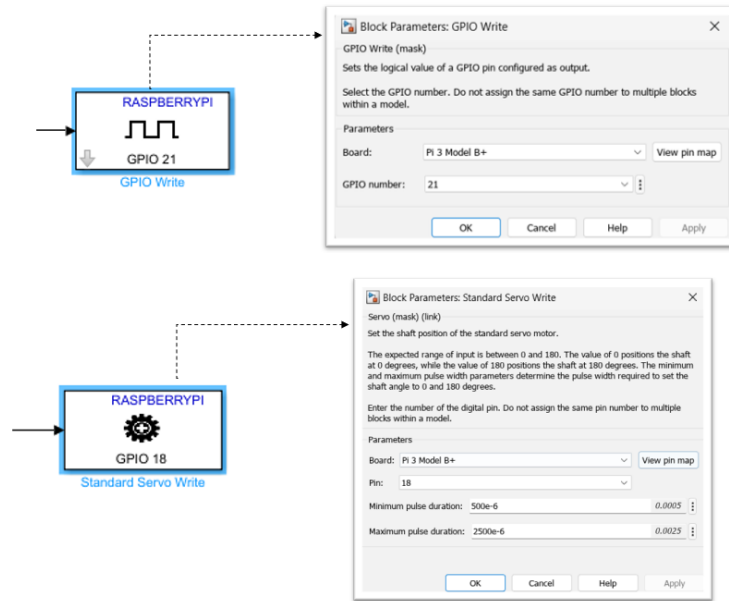


Figure 5.25: Simulink blocks for OBC communication

repeated at a fixed frequency. For SG90 the frequency is 50 Hz, which means that the pulses repeat every 20 milliseconds. The pulse width determines the position of the servo motor. For example, if the pulse width is 1 ms, the servo motor will rotate to its minimum angle (0°), while a pulse width of 2 ms will rotate it to its maximum angle (180°). Intermediate pulse widths will correspond to positions between the minimum and maximum angles. In PWM, the duty cycle (figure 5.26) is used to control the position of the servo motor. The duty cycle represents the percentage of time the signal is "high" (on) within one control cycle.

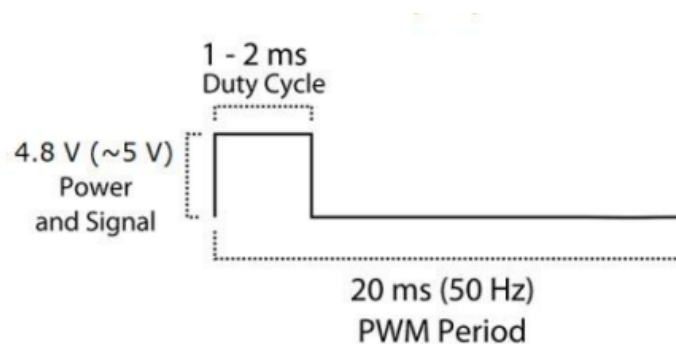


Figure 5.26: Example of duty cycle for SG90 servo motors [15]

The block "standard servo write" takes as input the angle in degrees and creates a duty cycle. The minimum and maximum pulse width duration has to be entered. For SG90 the minimum pulse duration is 0.5 milliseconds and the maximum is 2.5 milliseconds.

Chapter 6

Hardware-in-the-loop experiments

This chapter will present all the tests performed with the *MyDas* vehicle, using the code described in the previous chapter. The tests were conducted in Monterey, at the Naval Postgraduate School.

First, the "static" tests, which had already been carried out by Josef Kulke in [1], were reproduced. These tests aimed to evaluate the endurance of the vehicle after nearly 7 months of inactivity. First, endurance was evaluated with the use of air bearings only, then also with active thrusters. Next, tests were performed to perform attitude maneuvers. Finally, tests were performed to do attitude and position maneuvers simultaneously.

6.1 Static experiments

6.1.1 Floating without thruster

The first test that was performed was the floating duration test without the thrusters. It is used to evaluate the proper functioning of the air bearings and to assess leaks in the tube-bearing connection. During this test, no sensors are active, nor is the on-board computer turned on. Essentially, the air tank is filled to 2800 psi, the vehicle is placed on the granite floor, and the manual valve is opened. The thrusters have a "normally closed" operation, so they do not interfere with the test, and the compressed air only feeds the air bearing. The vehicle is allowed to float, and the time between when the valve is opened and when the vehicle stops floating is measured. To properly estimate the time when the vehicle is actually "stopped" on the plane, throughout the test the vehicle is kept moving on the floor.

The table 6.1 shows the results obtained and compares them with those obtained

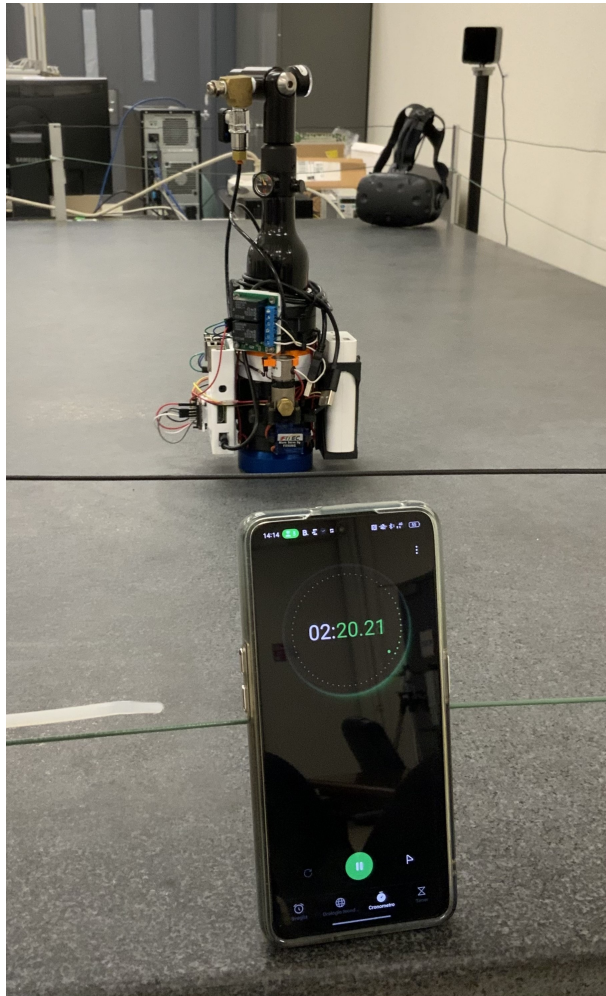


Figure 6.1: Static test for floating without thrusters

by Kulke in [1]. The average floating duration obtained in the previous tests, was measured with a different procedure: the measurement was made between the pressure 2500 psi and 1000 psi and then the result was multiplied by 2 to obtain a value between 3000 psi and 0. With this procedure, the times are slightly lower. This discrepancy may be related to the fact that at higher pressures, micro-leaks are generated that are not present at lower pressures. Thus, the floating time from 1000 to 0 psi is longer than that between 2500 and 1000 psi.

6.1.2 Thrusters opened without floating

The second static test involves the thrusters. This test is much more impactful on vehicle endurance, since the thrusters consume much more propellant than the air

	Floating time [min]
Run 1	15:12
Run 2	15:03
Run 3	15:17
Average	15:11
Previous Average [1]	13:12

Table 6.1: Experimental floating duration from 2800 psi to 0

bearing. The experimental setup is the same, except that the onboard computer is on, as it is necessary to control the valves. Again, the air tank filled to 2800 psi. The time between the opening of the solenoid valves and the time when the measured pressure goes to 0 psi is measured. In this case, the measurement was made by opening the two valves at the same time. The times are compared again with the previous times in table 6.2.

	Firing time [sec]
Run 1	38
Run 2	38
Run 3	39
Average	38
Previous Average [1]	36

Table 6.2: Experimental thruster firing duration from 2800 psi to 0

The test by Kulke was carried out in the same manner as the floating time. That is, the time required to go from 2500 psi to 1000 psi was measured, and this time was multiplied by 2. Again, it can be seen that the times in the new tests are longer. In fact, in the last test the new and old results are very similar. This can be explained by the fact that the firing time is an order of magnitude lower than the floating time, and the effect of micro-leakage is less noticeable.

6.2 Attitude maneuvers

This section describes the experimental setup and test results concerning attitude maneuvers. The goal of these tests is to verify that the vehicle can reach a certain angle ϕ . Successful tests will be presented with vehicle rotation of 90° , 180° and 270° , starting from an angle $\phi \sim 0$. It is important to note that a Kalman filter is not used in these tests, but sensor data are directly used to know the current position and speed of the vehicle.

Before running the code on the onboard computer and performing the maneuver, some steps need to be taken, which are given below:

1. Fill the air tank to a pressure of 2800 psi;
2. Turn on the onboard computer and power the voltage converter connected to the thruster valves;
3. Connect the base stations to the power supply and turn on the Vive Tracker;
4. Run the code on Unity on the external computer (this will enable Unity to enable the UDP protocol to send the Vive Tracker data to the on board computer);
5. Open manual valve to enable compressed air flow to the air bearings and solenoid valves;
6. Run the code on Simulink.

Table 6.3 shows the configuration parameters for the attitude control tests.

	Description	Parameter	Value
Spacecraft	Mass	m	1.7 kg
	Leverage	l	0.055 m
	Inertia vertical axis	J_z	$2.761 \times 10^{-3} \text{ kgm}^2$
PD control	Proportional Gain	K_P	0.3
	Derivative Gain	K_D	0.3
PWPFM	Activation Threshold	U_{on}	0.2
	Deactivation Threshold	U_{off}	0.001
	Low pass Filter Num	K_f	1
	Low pass Filter Den	τ	0.2885
Guidance	Initial Conditions	$(\phi_0, \dot{\phi}_0)$	(0 rad, 0 rad/s)
	Desired Condition	$([\phi_{des}], \dot{\phi}_{des})$	$([\frac{\pi}{2}, \pi, \frac{3}{2}\pi] \text{ rad}, 0 \text{ }^\circ/\text{s})$
Simulation	Sample time	dt	0.01 s
	Number of Runs		3

Table 6.3: Parameters for attitude maneuver

Three runs were performed, to reach the angles $\phi = \frac{\pi}{2}, \pi, \frac{3}{2}\pi$. In figures 6.2, 6.3, 6.4 the test results are shown.

For each test, data are reported for the on/off of the thrusters, the angle α_1 (and thus also α_2), and the state variables ϕ and $\dot{\phi}$.

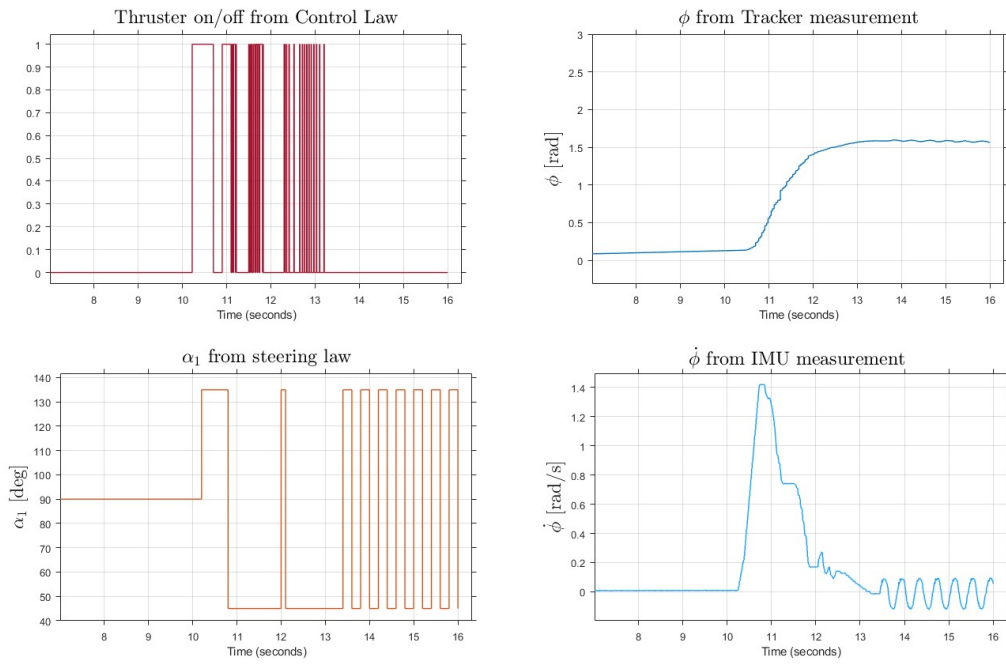


Figure 6.2: Experimental results for 90° rotation

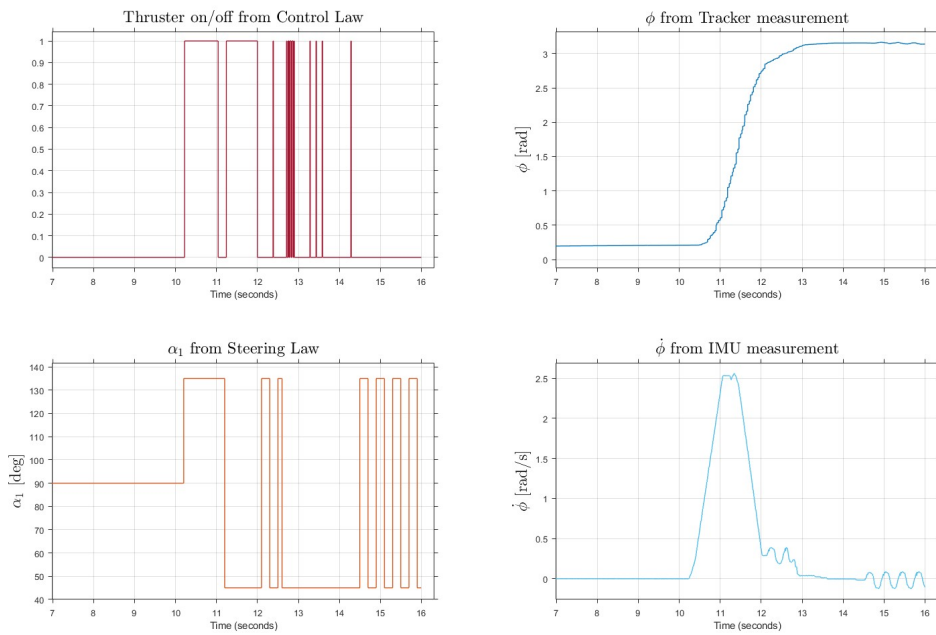


Figure 6.3: Experimental results for 180° rotation

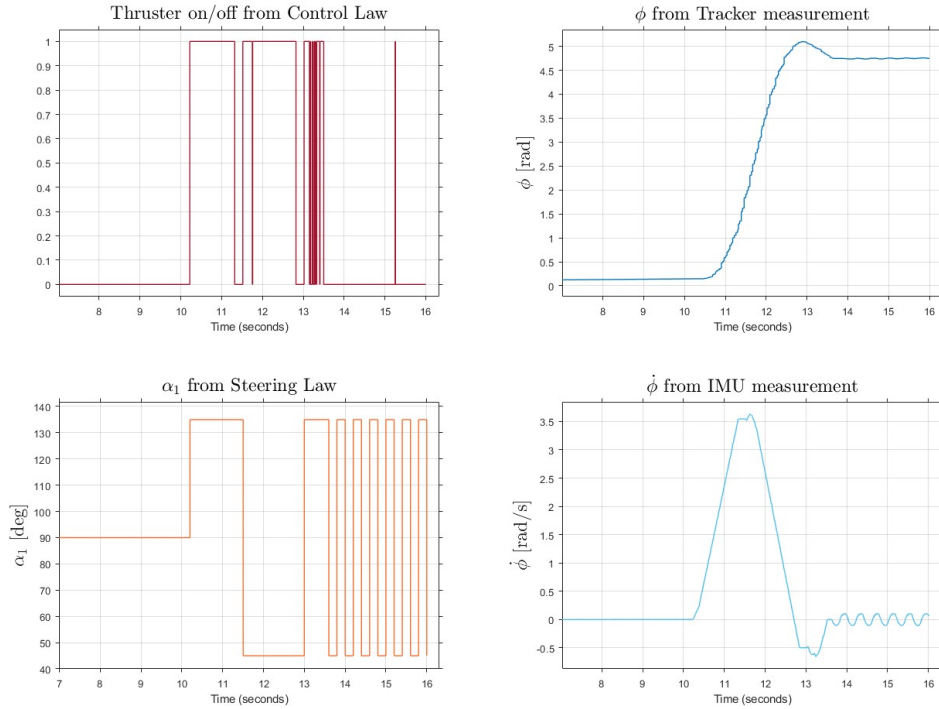


Figure 6.4: Experimental results for 270° rotation

The simulation starts from $t=0$, but for the first 10 seconds the vehicle does not perform any maneuvers. This time was needed for the operator to position the vehicle on the plane and to direct it to $\phi = 0$, which is the initial point of the maneuver.

- 90° rotation:** The duration of the maneuver is about 3 seconds. It can be seen that in the last 3 seconds the angular velocity $\dot{\phi}$ oscillates and does not stabilize. This is due to the fact that the servos, in the last 3 seconds, oscillate between 45° and 135° abruptly. This oscillation depends on the F_{com} varying between very low values around zero as the position ϕ_{des} has been reached. Indeed, it can be seen that the thrusters, after 3 seconds, no longer fire. The response of the system is very satisfying. There is no presence of overshoot, and the maneuver duration is quite fast but also smooth. The maneuver is thus successfully completed.
- 180° rotation:** The duration is about 4 seconds. Also in this there is oscillation of angular velocity, for the same reasons as before. The ϕ angle is correctly achieved, again without overshoot and with a very smooth maneuver. Thruster modulation performs better, as the thrust is less discontinuous and fewer shots are fired. The maneuver is therefore successfully completed.

- **270° rotation:** The duration of the maneuver is about 4.5 seconds. In this maneuver, a higher angular velocity is achieved than previous maneuvers, and overshoot is also present. The maneuver was performed with the same parameters as the previous ones for the PD controller and PWPFM, to simulate a worse maneuver where the vehicle has to perform small or such large rotations. However, the result remains satisfactory, and the maneuver can be said to be completed this time as well.

6.3 Position Maneuvers

In this paper, a position maneuver is defined as a maneuver that takes the vehicle from one point to another in the granite floor. As mentioned in the chapter 5, given the configuration of the thrusters, the position maneuver also involves an attitude maneuver, which allows the vehicle to align its η axis with the conjunction to the desired point. Thus, this type of maneuver is more complex computationally and in terms of controlling variables than the attitude maneuver. In fact, Position and Attitude Control must interact with each other in order to complete the maneuver correctly. In the attitude maneuver presented in the previous section, instead, only attitude control was active. In the tests that are given below, it was chosen to perform the maneuver in this way:

1. The position guidance sets a point for the vehicle to reach on the floor. This point is entered in the main Matlab code;
2. The attitude guidance calculates the angle ϕ_{des} , as described in the equation 5.1;
3. After 10 seconds, necessary for the pseudo-derivative on x and y to converge to a solution [5], the attitude maneuver begins;
4. As seen in previous tests, the worst attitude maneuver no longer lasts 4.5 to 5 seconds. Therefore, a time of 6 seconds is set to perform the attitude maneuver. After 6 seconds, the conjunction between the starting point (x_0, y_0) and the desired point (x_{des}, y_{des}) is aligned with the η axis of the vehicle. Then the position maneuver can start, which controls the vehicle along the η axis only, since the error on ξ is reduced to 0 due to the rotation. Thus, on the x-y plane, the vehicle controls the position along x and y, while on its reference system, only along the η axis.

Also in these tests, there is no Kalman Filter, but data from the sensors are used as the current position and velocity of the vehicle. In this case, however, the position control algorithm needs the velocities along x and y of the vehicle. This

data, is not directly provided by any of the sensors placed on the vehicle, so it must be calculated. As shown in figure 5.14, integrating the data from the accelerometer is not possible, if the bias is not corrected precisely by a Kalman Filter.

The second strategy would be to perform the pseudo-derivative of the position provided by the VIVE tracker, and thus obtain an estimate on the vehicle speed. The problem is that this data is very subject to noise, so it is necessary to insert a filter that mitigates the effect of noise. The filter is a "Low Pass Filter" type, while the pseudo derivative is calculated simply as:

$$\dot{x} = \frac{(x_k - x_{k-1})}{dt} \quad (6.1)$$

where x_k is the x position received in the instant t and x_{k-1} is the x position received in the instant $t - dt$.

In this section, results from 3 runs of position maneuver will be presented, and all the critical aspects of the maneuver will be discussed. Table 6.4 shows the configuration parameters for the attitude control tests.

	Description	Parameter	Value
Spacecraft	Mass	m	1.7 kg
	Leverage	l	0.055 m
	Inertia vertical axis	J_z	$2.761 \times 10^{-3} \text{ kgm}^2$
PD control attitude	Proportional Gain	K_P	0.3
	Derivative Gain	K_D	0.3
PD control position	Proportional Gain	K_{P-pos}	0.2
	Derivative Gain	K_{D-pos}	0.95
PWPFM	Activation Threshold	U_{on}	0.2
	Deactivation Threshold	U_{off}	0.001
	Low pass Filter Num	K_f	1
	Low pass Filter Den	τ	0.2885
Guidance	Initial Conditions	(x_0, \dot{x}_0)	(-0.27 m, 0 m/s)
		(y_0, \dot{y}_0)	(1.6 m, 0 m/s)
		$(\phi_0, \dot{\phi}_0)$	$(\frac{\pi}{4} \text{ rad}, 0 \text{ rad/s})$
	Desired Condition	(x_{des}, \dot{x}_{des})	(-0.27 m, 0 m/s)
		(y_{des}, \dot{y}_{des})	(1.2 m, 0 m/s)
Simulation	Sample time	dt	0.01 s
	Pseudo-derivative time		10 s
	Attitude Maneuver time		6 s
	Number of Runs		3

Table 6.4: Parameters for position maneuver

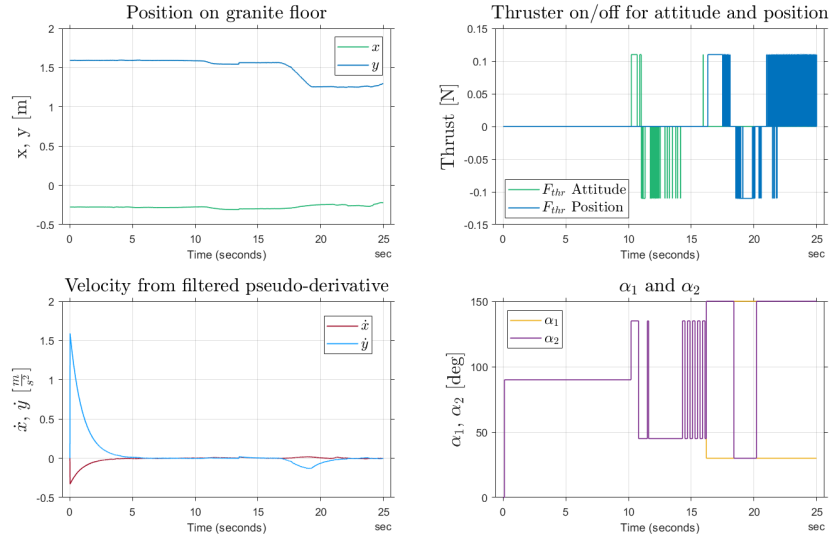


Figure 6.5: Experimental results from run # 1 for position maneuver

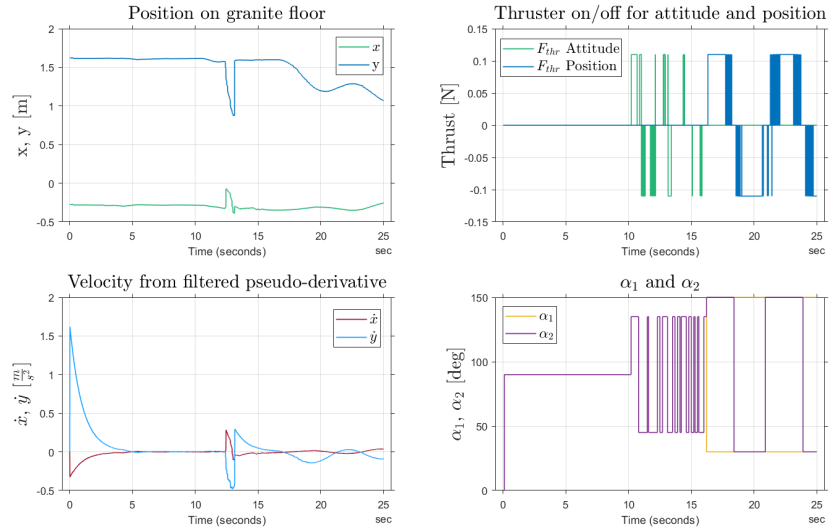


Figure 6.6: Experimental results from run # 2 for position maneuver

Results on the main parameters of the dynamical system have been reported in figure 6.5, 6.6, 6.7. These are 3 runs, in which the vehicle, starting from a position on the plane $(x_0, y_0) = (-0.27m, 1.6m)$, intends to reach the position $(x_{des}, y_{des}) = (-0.27m, 1.2m)$. Basically, the maneuver involves a displacement of 40 cm along the Y axis of the granite floor reference system. The vehicle is

positioned at the point x_0, y_0 with an angle $\phi = 45^\circ$. As mentioned before, the first maneuver involves pure rotation to point toward the desired point.

It can be seen from the results how the first 10 seconds are needed to converge the pseudo-derivative \dot{x}, \dot{y} to 0. So, after 10 seconds, the rotation maneuver starts. After another 6 seconds, the rotation maneuver is concluded, and the position maneuver begins. The first two maneuvers are concluded after about 15 seconds, and the results are satisfactory. The maneuver is not very precise, but it shows how the steering law, PWPFM and PD control work. The run #3 is a little bit different from the first two. It lasts 45 seconds, so it is a slower maneuver, and the starting point and final point are different. Also this maneuver can be considered completed. The data of the maneuvers are shown in the table 6.5.

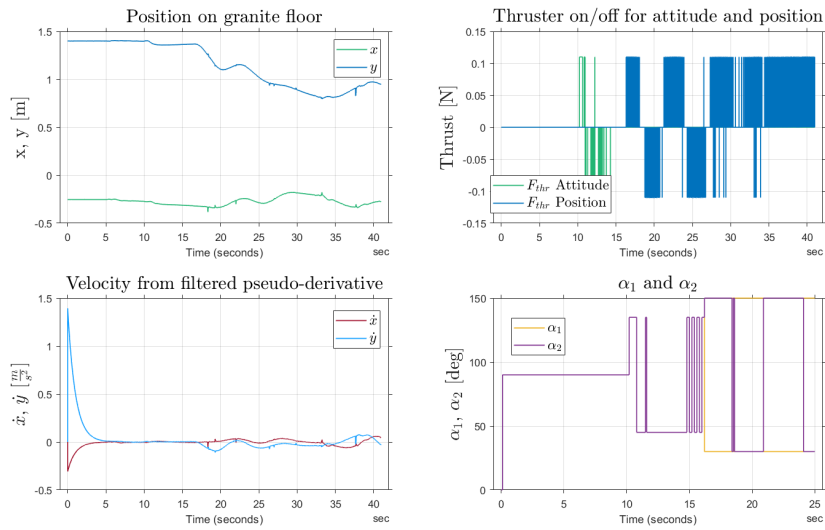


Figure 6.7: Experimental results from run # 3 for position maneuver

Parameter	Run 1	Run 2	Run 3
Duration	25 s	25 s	41 s
Starting Point (x, y)	$(-0.27, 1.59)$	$(-0.27, 1.62)$	$(-0.25, 1.40)$
Final Point (x, y)	$(-0.222, 1.3)$	$(-0.26, 1.06)$	$(-0.27, 0.94)$
Attitude Thruster ON	1.12 s	1.55 s	1.11 s
Position Thruster ON	4.67 s	6.22 s	10.09 s

Table 6.5: Results of position maneuvers

The maneuvers have been completed. In the graphs, position values along x-y (top left), velocities (bottom left), thruster ignition (top right), and values of α_1

and α_2 (bottom right) are shown.

Run 1 For run 1 the vehicle starts from position $(x, y) = (-0.27, 1.59)$ m. The desired point is $(-0.27, 1.2)$ m. In the velocity graph, it can be seen that the low-pass filter is acting to correct the error on the pseudo-derivative calculation. After 10 seconds, the attitude maneuver begins. This can be seen from the thrusters graph and the angles α_1 and α_2 . It is possible to see that the range of servos varies between attitude and position maneuvering. In fact, in the former case the range varies between 135° and 45° , in the latter case between 150° and 30° . As described in the previous chapter, angles different from 180° and 0° were chosen in order to have more control and smoother maneuvers. After several trials, these values were the appropriate ones to perform the two maneuvers. Moreover, the values of α_1 and α_2 seem to contradict the steering law described in figures 5.21. In fact, during position maneuvering it should be $\alpha_1 = \alpha_2$ and not vice versa. This discrepancy stands in the fact that the servos are specularly mounted in the vehicle, and therefore it is necessary to provide these angles as input to obtain the logic described in figure 5.21.

After another 6 seconds, the attitude one begins. Indeed, it is possible to see how the vehicle moves toward the desired point, from the attitude graph along x and y . After about 25 seconds, the maneuver is completed and the vehicle is at position $x = -0.22$ and $y = 1.3$ m. There is an error of 5 cm along x and 10 cm along y .

Run 2 Run 2 is almost a copy of run 1. Starting from a position $(x, y) = (-0.27, 1.62)$ m, the vehicle reaches position $(-0.26, 1.03)$ m. The error on x decreased (1 cm), while the error along y remained almost unchanged (11 cm).

Run 3 Run 3 is different from the first two. The vehicle starts from $(x, y) = (-0.25, 1.4)$ m and the desired point is $(-0.27, 1)$ m. The maneuver is performed more slowly by changing the constants of the PD controller. Specifically, $K_P = 0.2$ and $K_D = 1.2$. Therefore, the parameter K_D is larger compared to the previous runs. Therefore, the vehicle will have a longer settling time. In fact, the maneuver is completed in about 40 seconds. The endpoint is $(-0.27, 0.94)$ m obtaining an error of 0 on x and only 6 cm on y .

Chapter 7

Conclusion

The goals of this thesis work were achieved. The first was to design and build a small-scale Floating Spacecraft Simulator. The design was taken from Josef Kulke's design, making modifications to some electronic components. The weight and size of the vehicle is reduced compared to the older generations of FSS present at the Monterey NPS. This drastic reduction in weight and size is part of the project of miniaturization of satellite components. A testbed was also set up to test the vehicles. A granite plane, also small in size, was purchased. A laboratory was set up to acquire data from the position tracker. The electronics of the new vehicle have been tested and it is working properly. The vehicle can also be controlled with an XBOX controller. The pneumatic connections were tested and showed that the vehicle does not have any kind of pressure loss and is ready to start floating on the granite plane. The vehicle could not be tested on the new granite plane in Turin due to delays on the delivery of the suitable compressor.

Before a GNC algorithm was implemented, a digital twin of the vehicle was created to simulate maneuvers and estimate simulation parameters. In addition, a code was also written that allows the vehicle's maneuvers on the granite plane to be displayed on the screen, knowing the vehicle's position moment by moment.

Thus, a GNC algorithm was tested for the first time on this type of FSS. This included writing guidance, data acquisition and control algorithms. The data acquisition part from the VIVE tracker in on-line mode had never been tested before, and it produced very satisfactory results. Basic position and attitude maneuvers were performed, with appreciable results even considering the short time available to perform the tests. The maneuvers performed returned useful data to evaluate the effective maneuverability of the vehicle. In fact, the only two steerable thrusters offer an excellent solution in terms of weight and complexity, but they represent an obstacle to the vehicle's maneuverability. The maneuvers, however, are to be considered successful.

7.1 Future Works

Regarding the design and integration of the FSS, the goal is to make the vehicle smaller and lighter. Researching a lighter material for the structure could be one solution. But also look for lighter electronic and pneumatic parts. In addition, designing a different structure, perhaps not involving rings, could also lead to making the structure lighter. In addition, a docking mechanism could be added, to maneuver with multiple satellites together.

The digital twin represents a prototype and can be improved in many aspects. One of these could be the introduction of a mass ejection law from the vehicle that would improve the accuracy in calculating state variables. In addition, the testbed is quasi-frictionless type. Some residual accelerations are always present. One idea would be to compute and include these external disturbances in the motion equations to get a simulation as close to the reality as possible. Finally, a more user-friendly interface could be designed, allowing more immediate management of firing duration and servo angles.

The GNC algorithm can certainly be improved. The first improvement is definitely to include a Kalman Filter, to get sensor data less susceptible to error. Then, other guidance and control algorithms can be added, such as Artificial Potential Field (APF) or Sliding Mode. This is to be able to perform even more complex maneuvers, for example, with collision avoidance. All the work done so far is "open source," so it will be made available to make changes and improve the project.

Appendix A

Appendix

A.1 Listings

```
1 clear all
2 clc
3
4 % Spacecraft data
5 m=1.533; %mass [kg]
6 l=0.055; %leverage of the thrust [m]
7 Jz=2.761e-3; %z axis inertia [kgm^2]
8
9 %Thrust data
10 T1=0.110; %Thruster 1 [N]
11 T2=0.110;
12 F=T1;
13
14
15 % out=sim('untitled')
16
17 %PID CONTROLLER
18 vett_des1=[-0.27 1 0 0]; % x , y , x_dot , y_dot
19 vett_des2=[-0.27 1]; % x, y
20 vett_des=[3/2*pi 0]; % theta , theta_dot
21
22 %initial condition
23 x0=0;
24 y0=0;
25 phi0=0;
26 x_dot0=0;
27 y_dot0=0;
28 phi_dot0=0;
29
```

```
30 %PD CONTROL x-y
31 Kp=0.2;
32 Kd=0.95;
33
34 %PD CONTROL phi
35 Kp_p=0.3;
36 Kd_d=0.3;
37
38 %PWM x-y
39 Uon=0.02;
40 Uoff=0.001;
41 Kf=1;
42 tau=0.2885;
43
44 %PWM phi
45 Uon_phi=0.02;
46 Uoff_phi=0.001;
47 Kf_phi=1;
48 tau_phi=0.2885;
```

Listing A.1: Main Matlab Code

```
1 clear all
2 clc
3
4 % Spacecraft data
5 m=1.533;
6     %mass [kg]
7 L=0.055;
8     %leverage of the thrust [m]
9 Jz=2.761e-3;
10    %z axis inertia [kgm^2]
11
12 %Simulation
13 t_end=6;
14 dt=0.05;
15     %sample time
16 T=(0:dt:t_end)';
17     %Simulation time [s]
18
19 %Thrust data
20 alpha_1=linspace(-pi/2,-pi/2,length(T));
21     %angle of orientation F1 [rad]
22 alpha_2=linspace(-pi/2,-pi/2,length(T));
23     %angle of orientation F2 [rad]
24 T1=0.126;
25     %Thruster 1 [N]
26 T2=0.126;
27     %Thruster 2 [N]
28
29 % External Forces (fire of 1s from t0=0)
30 tf=1;
31     %time of firing [s]
32 F1=zeros(1,length(T));
33 F1(1:tf/dt +1)=T1;
34 %F1(tf/dt +1 + 1/dt:tf/dt +1 + 1/dt+tf/dt)=-T1;
35
36 F2=zeros(1,length(T));
37 F2(1:tf/dt +1)=T2;
38 %F2(tf/dt +1 + 1/dt:tf/dt +1 + 1/dt+tf/dt)=-T1;
39
40 %% State space system
41 A=[0 1 0 0 0 0
42    0 0 0 0 0 0
43    0 0 0 1 0 0
44    0 0 0 0 0 0
45    0 0 0 0 0 1
46    0 0 0 0 0 0];
```

```
39 B= [0 0 0
40     1/m 0 0
41     0 0 0
42     0 1/m 0
43     0 0 0
44     0 0 1/Jz];
45
46 C=eye(6);
47 D= zeros(6,3);
48
49
50 F1_z=F1.*sin(alpha_1).*L;
51 F2_z=F2.*sin(alpha_2).*L;
52
53 U_z=-F1_z+F2_z;
54     %control vector z
55
56 F1_x=F1.*cos(alpha_1);
57 F2_x=F2.*cos(alpha_2);
58
59
60 U_x=F1_x-F2_x;
61     %control vector x
62
63 F1_y=F1.*sin(alpha_1);
64 F2_y=F2.*sin(alpha_2);
65
66 U_y=F1_y+F2_y;
67     %control vector y
68
69 F= [U_x;U_y;U_z];
70
71 x0=zeros(6,1);
72     %initial condition
73
74 S=ss(A,B,C,D);
75
76 X=lsim(S,F,T,x0);
```

Listing A.2: Digital Twin Main code

```

1 clear all
2 clc
3 %data import
4 load("simul_tesi.mat");
5 Dati=extractTimetable(data);
6
7 %%
8 % phi=-Dati.phi_measured;
9 % T=Dati.Time;
10 phi=Dati.phi;
11 T=Dati.Time;
12 % x=Dati.Data_7*1000;
13 % y=Dati.Data_8*1000;
14 % z=zeros.*x;
15 x=Dati.x*1000;
16 y=Dati.y*1000;
17 z=zeros.*x;
18
19 %% granite table Torino
20 %
21 % granite=stlread("granite_table_torino.stl");
22 % P_g = granite.Points; %access the vertex data from
    triangulation
23 % C_g = granite.ConnectivityList; %access the connectivity data
    from triangulation
24 % P_g(:,1) = P_g(:,1) - (630/2); %add to each vertex's x value
25 % P_g(:,2) = P_g(:,2) - (400/2);
26 % P_g(:,3) = P_g(:,3) - 85;
27 % fv_g = triangulation(C_g, P_g); %Combine both components back
    into a triangulation variable
28 % anim_2=trimesh(fv_g, 'FaceColor','k', 'EdgeColor','k', 'FaceAlpha
    ', '0.5', 'EdgeAlpha', '0.5');
29 %
30 % xlim([-300,300])
31 % ylim([-300,300])
32 % zlim([-100,500])
33 % hold on
34
35 % granite table Monterey
36 granite=stlread("granite_table_Monterey.stl");
37 P_g = granite.Points; %access the vertex data from triangulation
38 C_g = granite.ConnectivityList; %access the connectivity data
    from triangulation
39 P_g(:,1) = P_g(:,1) - (1600); %add to each vertex's x value
40 P_g(:,2) = P_g(:,2) + (200);
41 P_g(:,3) = P_g(:,3) - 200;
42 fv_g = triangulation(C_g, P_g); %Combine both components back
    into a triangulation variable

```

```

43 anim_2=trimesh(fv_g, 'FaceColor','k', 'EdgeColor','k', 'FaceAlpha', '
    0.5', 'EdgeAlpha', '0.2');
44
45 xlim([-2000,1000])
46 ylim([-400,2600])
47 zlim([-200,1000])
48
49 xlabel('x')
50 ylabel('y')
51 zlabel('z')
52 hold on
53
54 %FSS
55 fv=stlread("fss_3.stl");
56 P = fv.Points; %access the vertex data from triangulation
57 C = fv.ConnectivityList; %access the connectivity data from
    triangulation
58
59 % phi=linspace(0,pi/2,1000);
60 pos=[x,y,z];
61
62 %ANIMATION
63 for i=1:1:length(T)
64 P_n=P;
65 %modifica Z
66 P_n(:,3) = P(:,3) - min(P(:,3));%add to each vertex's x value
67
68 %modifica x-y
69 R=[cos(phi(i)) -sin(phi(i)) 0
70     sin(phi(i)) cos(phi(i)) 0
71     0 0 1];
72 P_xy=(R*(P_n)') + pos(i,:);
73
74
75 fv = triangulation(C, P_xy); %Combine both components back into a
    triangulation variable
76 anim=trimesh(fv, 'FaceColor', 'c', 'EdgeColor', 'k', 'EdgeAlpha', '0.1');
77 pause(0.1)
78 delete(anim)
79
80 title("time=" + string(T(i)) + "s" + ' ' + "\phi = " + ...
81     round(rad2deg(phi(i)),0) + " ")
82 % + ' ' + "x =" + round(X(i),3) + "m" + ' ' + "y =" + round(Y
    (i),3) + "m")
83 end

```

Listing A.3: Digital Twin graphics animation code

```
1     using UnityEngine;
2 using System.Net.Sockets;
3 using System;
4 using System.IO;
5 using System.Text;
6
7
8 public class Socket : MonoBehaviour
9 {
10     // Use this for initialization
11
12     internal Boolean socketReady = false;
13     NetworkStream theStream;
14     StreamWriter theWriter;
15     StreamReader theReader;
16
17
18
19     void FixedUpdate()
20     {
21
22         setupSocket();
23         Debug.Log("socket is set up");
24
25     }
26
27
28     public void setupSocket()
29     {
30         string serverIP = "192.168.16.163";
31         int port = 5500;
32
33         try
34         {
35             double [] values = new double [] { transform.position.x,
transform.position.y, transform.position.z, transform.eulerAngles.
x, transform.eulerAngles.y, transform.eulerAngles.z, Time.time };
36             byte [] sendBytes = new byte [values.Length * sizeof(double
)];
37
38             for (int i = 0; i < values.Length; i++)
39             {
40                 byte [] bytes = BitConverter.GetBytes(values[i]);
41                 bytes.CopyTo(sendBytes, i * sizeof(double));
42             }
43
44             UdpClient udpClient = new UdpClient();
```

```
45         udpClient.Send(sendBytes, sendBytes.Length, serverIP,
46         port);
47     }
48     catch (Exception e)
49     {
50         Console.WriteLine(e.ToString());
51     }
```

Listing A.4: C# code for sending data from Unity to Raspberry via UDP

A.2 Bill of Materials

Product	Description	Vendor	Price Amount
Jumper cables	M-F 3" black	Polulu	3,20 € Pack of 10
Jumper cables	M-M 3" black	Polulu	3,20 € Pack of 10
Jumper cables	F-F 3" black	Polulu	3,20 € Pack of 10
USB cable	Type A to micro	Amazon	1,99 € 1
USB cable	type A to VCC/GND	Amazon	1,99 € 1
Servo	Micro Servo Motors 180° range	Amazon	7,12 € 2
Relay	2-Channel SPDT Relay Carrier	Polulu	10,04 € 1
Voltage Converter	from 3-35 VDC to 3-35 VDC	Amazon	7,99 € 1
Raspberry Pi4	On board computer 2 GB RAM	Kubi	104,96 € 1
Micro SD	Flash Memory 32 GB	RS Components	13,05 € 1
Power Bank	Power Supply 8000 mAh	Amazon	31,86 € 1
IMU Sensor	3 axis accelerometers 3-axis gyroscope	Amazon	11,80 € 1
Tracker	VIVE Tracker 3.0	HTC	163,48 € 1
Base Station	Vive Base station	HTC	485,02 € 2
VR	Vive Pro 2	HTC	1.023,95 € 1
		Total	1872,85 €

Table A.1: Electronic Parts

Product	Description	Vendor	Price Amount
Air tank	Field Tank 0.21L-13ci	Paintball Shop	44,95 € 1
Air Bearings	Air bearing flat 65mm diam.	New Way	279 € 1
Pressure regulator	90° Boulder Air Pneumatic Regulator	Palmer Pursuit Shop	127,95 € 1
Solenoid Valve	24 V normally closed	RS Components Components	44,23 € 2
Manometer	0-200 psi range 1/8 NPT	Amazon	18,52 € 1
Air Fitting	Adapter 1/8 NPT M-M	Amazon	10,20 € 2
Air Fitting	1/8 NPT F T intersection	Amazon	15,15 € Pack of 2
Manual Valve	2-Way Brass Mini BallValve	Tameson	6,06 € 1
Air tube	OD 6mm - ID 4mm 10 mt	Amazon	11,99 € 1
Air Fitting	Brass Elbow HoseBarb 4 mm - G1/4"Rotable	Tameson	13,18 € 2
Air Fitting	Brass Elbow Hose Barb 4 mm - M5 Rotable	Tameson	10,64 € Pack of 2
Air Fitting	Push-in Fitting 6mm x G1/8"	Tameson	7,45 € Pack of 5
		Total	589,32 €

Table A.2: Pneumatic Parts

Bibliography

- [1] Joseph Kulke. «Design of a Simplified Floating Spacecraft Simulator». MA thesis. Hamburg: Helmut-Schmidt-Universität / Universität der Bundeswehr Hamburg, 2022 (cit. on pp. 1, 13, 17, 18, 22, 25, 29, 63–65).
- [2] Markus Wilde, Casey Clark, and Marcello Romano. «Historical survey of kinematic and dynamic spacecraft simulators for laboratory experimentation of on-orbit proximity maneuvers». In: *Progress in Aerospace Sciences* 110 (2019), p. 100552. ISSN: 0376-0421. DOI: <https://doi.org/10.1016/j.paerosci.2019.100552>. URL: <https://www.sciencedirect.com/science/article/pii/S0376042119300466> (cit. on pp. 1, 3–5).
- [3] Markus Wilde. «Visual Augmentation Methods for Teleoperated Space Rendezvous». PhD thesis. Technische Universität München, 2012 (cit. on p. 4).
- [4] Richard Zappulla, 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». In: *Journal of Spacecraft and Rockets* 54.4 (2017), pp. 825–839. DOI: 10.2514/1.A33769. eprint: <https://doi.org/10.2514/1.A33769>. URL: <https://doi.org/10.2514/1.A33769> (cit. on pp. 5, 7, 13).
- [5] Marcello Romano, David A. Friedman, and Tracy J. Shay. «Laboratory Experimentation of Autonomous Spacecraft Approach and Docking to a Collaborative Target». In: *Journal of Spacecraft and Rockets* 44.1 (2007), pp. 164–173. DOI: 10.2514/1.22092. eprint: <https://doi.org/10.2514/1.22092>. URL: <https://doi.org/10.2514/1.22092> (cit. on pp. 15, 30, 69).
- [6] Jason Hall and Marcello Romano. «Laboratory Experimentation of Guidance and Control of Spacecraft During On-Orbit Proximity Maneuvers». In: Mar. 2010. ISBN: 978-953-307-041-4. DOI: 10.5772/9132 (cit. on p. 15).
- [7] R. Bevilacqua, T. Lehmann, and M. Romano. «Development and experimentation of LQR/APF guidance and control for autonomous proximity maneuvers of multiple spacecraft». In: *Acta Astronautica* 68.7 (2011), pp. 1260–1275. ISSN: 0094-5765. DOI: <https://doi.org/10.1016/j.actaastro.2010>.

- 08.012. URL: <https://www.sciencedirect.com/science/article/pii/S0094576510003000> (cit. on p. 15).
- [8] Josep Virgili-Llop, Jerry II, and Marcello Romano. «Design and Parameter Identification by Laboratory Experiments of a Prototype Modular Robotic Arm for Orbiting Spacecraft Applications». In: Mar. 2016 (cit. on p. 15).
- [9] Edward Glaessgen and David Stargel. «The digital twin paradigm for future NASA and U.S. air force vehicles». In: Apr. 2012. ISBN: 978-1-60086-937-2. DOI: 10.2514/6.2012-1818 (cit. on p. 28).
- [10] Yang Wenqiang, Yu Zheng, and Shaoyang Li. «Application Status and Prospect of Digital Twin for On-Orbit Spacecraft». In: *IEEE Access* PP (July 2021), pp. 1–1. DOI: 10.1109/ACCESS.2021.3100683 (cit. on p. 28).
- [11] Richard Zappulla, Hyeongjun Park, Josep Virgili-Llop, and Marcello Romano. «Real-Time Autonomous Spacecraft Proximity Maneuvers and Docking Using an Adaptive Artificial Potential Field Approach». In: *IEEE Transactions on Control Systems Technology* 27.6 (2019), pp. 2598–2605. DOI: 10.1109/TCST.2018.2866963 (cit. on p. 29).
- [12] *MPU-6000 and MPU-6050 Product Specification*. PS-MPU-6000A-00. Rev. 3.4. InvenSense. Aug. 2013 (cit. on p. 48).
- [13] Richard Zappulla, Josep Virgili-Llop, and Marcello Romano. «Spacecraft Thruster Control via Sigma–Delta Modulation». In: *Journal of Guidance, Control, and Dynamics* 40.11 (2017), pp. 2928–2933. DOI: 10.2514/1.G002986. eprint: <https://doi.org/10.2514/1.G002986>. URL: <https://doi.org/10.2514/1.G002986> (cit. on p. 58).
- [14] Bong Wie. *Space Vehicle Guidance, Control and Astrodynamics*. American Institute of Aeronautics and Astronautics, Inc., 2015 (cit. on p. 58).
- [15] Unknown. *Datasheet : SG90*. URL: <http://www.datasheet-pdf.com/PDF/SG90-Datasheet-TowerPro-791970> (cit. on pp. 60, 61).