Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Guida, navigazione e controllo per il docking autonomo di un Cubesat

Guidance, Navigation and Control for autonomous docking of a Cubesat

Relatore
Prof. Stefano Ghidoni

Correlatore
Prof. Francesco Branz

Laureando
Federico Favotto

Anno Accademico 2021-2022
11-Apr-2022

Thanks to my family and friends, who have supported me during the university. Thanks to Stefano Ghidoni and Francesco Branz who reviewed this thesis and accompanied me in my work of this project. Thanks to Omar, Francesco and Andrea for the beautiful moments we spent together. Thanks (and good luck) to the ERMES team: Alessandro, Giuliano, Miroljub, Fabio, Nicola.

Abstract - EN

In recent years there has been growing interest in autonomous docking manoeuvres. In aerospace they are important and of scientific interest because they allow the development of larger structures or the removal of debris in orbit. The fields of application are manifold, so Experimental Rendezvous in Microgravity Environment Study (ERMES) has as its main objective the design and test of an autonomous docking manoeuvre between two free-flying CubeSats mock-ups. The two mock-ups involved in the experiment are both equipped with a Guidance Navigation and Control (GNC) system and mechanical docking interfaces. During the manoeuvre, they work in a Target-Chaser configuration, where the Chaser is active and the Target is cooperative. In addition, the Chaser is equipped with a cold gas propulsive system based on expendable CO2 cartridges, while the Target is equipped with three reaction wheels. This thesis proposes a software architecture to implement a collaborative GNC system based on the Robot Operating System (ROS). The implementation proposal uses navigation based on the fusion of information from two IMU and a AprilTag detection system, with the aim of improving state estimation. In fact, it is believed that a ROS-based system can benefit the development of the experiment: thanks to the availability of related Open Source libraries, it is possible to increase the complexity of the on-board computation systems without impacting on the development time. The implementation uses a navigation system based on the fusion of the information coming from two IMUs and from the recognition of AprilTags, placed on the face of the Target, with the aim of improving the localization and the estimation of the system status. A test to verify the correct integration of the components is also proposed, and future tests to improve performance under certain conditions are described.

Abstract - IT

Negli ultimi anni è cresciuto sempre più l'interesse verso le manovre di docking autonomo. In ambito aerospaziale esse son importanti e hanno interesse scientifico perché permettono lo sviluppo di strutture più grandi, oppure la rimozione di detriti in orbita. I campi di applicazione sono molteplici, ed ecco che Experimental Rendezvous in Microgravity Environment Study (ERMES) ha tra i principali obiettivi quello di progettare e testare una manovra di docking autonomo tra due CubeSats mock-ups. I due mockup son provvisti di un sistema di Guida, Navigazione e Controllo e delle interfacce di docking meccanico. Durante la manovra, lavorano in una configurazione Target-Chaser, dove il Chaser è la parte attiva e il Target è cooperativo. Inoltre il Chaser è provvisto di un sistema di propulsione basato su cartucce CO_2 usa e getta, mentre il Target è dotato di tre ruote di reazione. Questa tesi propone una architettura software per implementare un sistema GNC collaborativo basato su Robot Operating System (ROS). Si crede infatti che un sistema basato su ROS possa avantaggiare lo sviluppo dell'esperimento. Infatti, grazie alla disponibilità di librerie Open Source ad esso correlate, è possibile aumentare la complessità dei sistemi di computazione di bordo senza impattare sui tempi di sviluppo. L'implementazione utilizza un sistema di navigazione basato sulla fusione delle informazioni provenienti da due IMU e dal riconoscimento di AprilTag, posti sulla faccia del Target, con l'obiettivo di migliorare la localizzazione e la stima dello stato del sistema. Viene inoltre proposto un test per verificare la corretta integrazione delle componenti, e descritti test futuri per migliorare le prestazioni in certe condizioni.

Contents

List of Figures

List of Acronyms

APDCS Attitude and Position Determination and Control System. 11, 13

ASDS Autonomous Satellite Docking System. 5

ATV Automated Transfer Vehicle. 5

CPOD CubeSat Proximity Operations Demonstration. 5

CSI Camera Serial Interface. 13, 16

DoF Degrees of Freedom. 6, 8, 9, 11, 16, 41, 42

EKF Extended Kalman filter. 27, 49, 52

ELGRA European Low Gravity Research Association. 2

ERMES Experimental Rendezvous in Microgravity Environment Study. 1–4, 7, 8, 11–13, 16, 17, 19, 21, 23–28, 30, 32, 35–37, 42–45, 49, 51, 52, 57, 61

ESA European Space Agency. 1, 2

ESDP Experiment Safety Data Package. 3

EV Electrovalve. 16, 23, 24, 37, 38, 41, 42, 61

FELDs Flexible Electromagnetic Leash Docking system. 6

FYT Fly Your Thesis!. 1, 2

GPS Global Positioning System. 51

HLS High Level System. 23, 34, 53

I2C Inter Integrated Circuit. 13, 14, 23, 24, 38, 43

IMU Inertial Measurement Unit. 11–14, 17, 18, 24, 26, 27, 34, 37, 42, 46, 48, 49, 51–53, 55

IR LED Infrared Light-Emitting Diode. 11, 26, 43

ISS International Space Station. 1, 5

LLS Low Level System. 23, 24, 32, 37, 38, 41–43, 48, 50, 51, 53

MLS Medium Level System. 23, 24, 35, 37, 41, 53

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor. 12, 16

OBCS Onboard Computer System. 1, 11–13, 17

PACMAN Position and Attitude Control with MAgnetic Navigation. 6, 18, 43

PWM Pulse-width modulation. 13, 16, 38, 41, 61

ROS Robot Operating System. 35, 36, 41–45, 48, 49, 51, 53, 61

SPHERES Synchronized Position Hold, Engage, Reorient, Experimental Satellites. 5

ToF Time of Flight. 11, 13, 14, 26, 37, 42, 48–50, 52

UART Universal Asynchronous Receiver-Transmitter. 13, 24

USB Universal Serial Bus. 16

Introduction

This dissertation presents the design and development of the control software of the experimental rendevouz in microgravity environment, hence the name Experimental Rendezvous in Microgravity Environment Study (ERMES). ERMES is an experiment that has in its main objetives to perform a docking manoeuvre between two free-flying CubeSats mock-ups. The mock-ups involved in the experiment are equipped with an Onboard Computer System (OBCS) and a mechanical docking interface. They are referred as "Chaser" and "Target". The chaser will be able to move using a cold gas propulsion system based on expendable CO_2 cartridges. The target on the other hand will be able to change its orientation using three reaction wheels. The chaser will approach the target in a calculated trajectory and will lock the docking interface when it is in the final position.

In order to test the system in a reduced-gravity environment, ERMES is selected in Fly Your Thesis! (FYT) programme by European Space Agency (ESA) in the 2022 edition. This allows us the opportunity to test and perform the experiment in a real microgravity environment.

This thesis will introduce the design of the experiment, the organisation of the *Fly Your Thesis!* programme, and how the parabolic flights offered by Novespace[1] work. Then it will present the hardware design in the Chapter 1. The software design is explained in the Chapter 2 followed by its implementation in the Chapter 3

Fly Your Thesis!

The *Fly Your Thesis!* programme is an opportunity provided to master and PhD students by ESA. Along with the *Drop your Thesis!* and the *Spin your Thesis!*, FYT is part of the Hands-on space projects of ESA Education. In FYT a team of students has the opportunity to fly technological research or scientific experiments related to engineering, physics, chemistry, biology in microgravity conditions. FYT is the only microgravity platform among those listed that allows scientists to interact with their own experiment while it is experiencing weightlessness, rather than doing it by remote on a robotic capsule or in the International Space Station (ISS) where only astronauts can interact with it. The programme launches a call for proposal once per year, during which master and PhD students can send their

[1]Novespace site: `https://www.airzerog.com/scientific-research-services/`

Figure 1: The Airbus A310 Zero-G

proposal of an experiment designed for parabolic flights.

After the presentation of the experiment proposal, ESA will shortlist the top teams who have met the eligibility criteria. The teams will present their experiment to a review board composed of experts from ESA Human Spaceflight and Robotic Exploration Directorate, ESA Education Office, European Low Gravity Research Association (ELGRA) and Novespace [1].

Some teams will be selected to fly in the campaign, in the 2022 campaign ERMES is one of them, hence this thesis. To accomplish the goal to prepare the experiment, a series of assignments to design and test the experiment is planned with ESA. This should be done in accordance with the security measures, ESA and Novespace experts will review all the work progress between each step.

In the last four-month period there will be the flight campaign. It is a two-week long event that will take place in Bordeaux, France. The teams will partecipate in three flights of 30 parabolas each, experiencing about 20 seconds of microgravity (and 20 of hypergravity, about $1.8g$).

When the microgravity experiment is done the teams can retrieve their data, and publish a report.

Novespace and the A310 ZERO-G

Novespace's main activity is to organise scientific parabolic flights for research in microgravity on the Airbus A310 Zero G [2]. These scientific flight campaigns have been operated for many years. In April 2015, Novespace began operating its third aircraft, the Airbus A310 Zero G (Fig.: 1). This is a modified version of the airplane to accomodate the experiments in the cabin. They also have a separated cabin to experience weightless in <a safe way. Novespace can provide even different levels of g-force, to simulate different space environments. In the case of FYT the parabolas are performed to provide near $0g$ microgravity. The

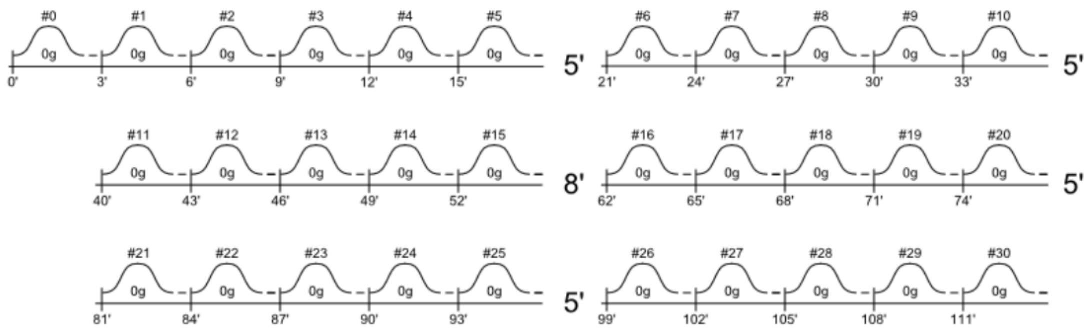

Figure 2: The schedule of the series of parabolas

level of microgravity is around $\pm 5 * 10^{-2}$.

After the takeoff the plane will go in a zone decided with regards to the weather. Then it will perform a parabola as test, this is necessary to let the people to get used of the manoeuvre without worrying about the experiments. Then six series of five parabolas each are performed with a five minutes pause between each series.

Due to the nature of the flight, there are additional safety measures compared to standard passenger flights. The Experiment Safety Data Package (ESDP) is a safety document that must be updated by the team and analyzed with Novespace every three weeks. This allows the technical contact, provided by Novespace to each team, to review the ESDP and advise if there is something to fix. The design of the experiment will be frozen after June 2022 and the experiment must be built at least four weeks before the flight.

What is a parabolic flight

A parabolic flight is an aircraft flight that performs a series of manoeuvres, called parabolas, each providing 20 seconds of microgravity or weightlessness, during which scientists are able to test their experiments in low gravity conditions and obtain data that would otherwise not be possible on Earth.

To be weightlessness means to be in a free fall. To achieve this result the plane must follow a certain path. There are 4 forces that act on the plane: weight, drag, thrust, lift. In normal condition a plane has a lift equals to the weight force. And if it is going at constant velocity it will have the thrust equal to the drag force. In this condition the objects in the cabin experience the normal gravity acceleration. To achieve a microgravity environment in the plane, it needs to be in a zero-lift angle of attack. It also must balance thrust to have no acceleration along the longitudinal axis. In these conditions inside the cabin the object will have near 0 acceleration relative to the airplane. In this thesis those condition could be referred as “zero-gravity” for convenience.

Now, the weightlessness condition is clear, but what happens when the plane needs to regain altitude? It will climb and inside the cabin there will be $1.8g$ apparent weight. In the ERMES experiment there are not any benefit of this

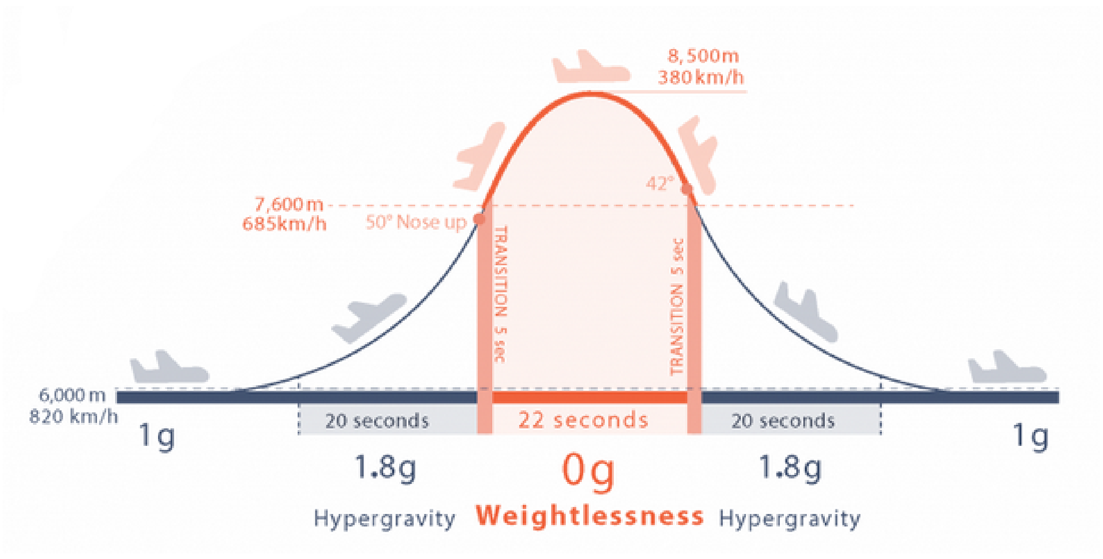

Figure 3: Apparent weight during the parabola - scheme from the Novespace site

phase, but sometimes there are experiments that can make use of both phases. In the Figure 3 there is the acceleration experienced in the cabin during the flight.

ERMES

The main goal of ERMES is the design and development of a test in a microgravity environment for an autonomous docking manoeuvre between two free flying CubeSat mock-ups equipped with Guidance Navigation and Control (GNC) systems. The two mock-ups will work in a Target-Chaser configuration.

The Chaser has control over its attitude and position, while the Target only over its attitude. The manoeuvre will be accomplished by releasing the CubeSat mock-ups from their initial electromagnetic constraints into a free flying condition, then the dedicated localization and proximity navigation software will control attitude and position of the Chaser and slowly approach and dock the Target, that in the meanwhile will work cooperatively by contrasting attitude disturbances.

The ERMES major objective can be achieved thanks to a series of low level goals: the design and development of the Target and Chaser, their structure and all of the hardware interfaces; the design and development of the Guidance Navigation and Control software, able to locate autonomously the Target, compute the trajectory to reach it and send the commands to the actuators; finally, the design and development of miniaturized docking interfaces, suitable for this particular application.

In general, the ERMES experiment aims to prove the feasibility and versatility of autonomous docking manoeuvres between small satellites. This general objective can be achieved thanks to the validation and testing of the GNC systems in relevant reduced-gravity environment and the validation of the manoeuvre configuration and selected approaches.

The two mock-ups involved in the manoeuvre work in a Target-Chaser configuration, in which the Chaser actively performs the manoeuvre to reach the Target, that, in the meanwhile, acts cooperatively by contrasting unwanted attitude disturbances. Moreover, the manoeuvre will be performed autonomously by the mock-ups. They will be released by a Release Structure composed of a holding mechanism to support the mock-ups prior to the start of the experiment and a slider to withdraw the locking mechanism so that it does not interfere with the experiment. The mock-ups are equipped with probe-drogue miniaturized docking interfaces [3], composed of an active probe on the Chaser and a passive drogue on the Target.

The connection between these two interfaces is obtained with a physical insertion of the two and then a mechanical interlock thanks to a servo motor that rotates the tip of the probe.

The experiment is composed by three main phases:

1. **release** of the mock-ups: the Release Structure removes the magnetic constraint and retracts the locking mechanism;

2. **proximity** navigation: the chaser will approach the target by controlling its velocity and attitude, while the target will maintain its attitude;

3. **docking phase**: the chaser will lock the target activating the mechanical interlock;

After each maneuvre, an operator relocates the mock-ups on the Release Structure to prepare the system for the next parabola.

Related work

Autonomous space systems have always been an interesting topic in the scientific community because they allow particularly useful applications related to large structure assembly, active space debris removal or formation control. The Automated Transfer Vehicle (ATV) [4] is one of the first examples of an autonomous space system that carried out on multiple occasions rendezvous with the ISS. Autonomous Satellite Docking System (ASDS) [5] from Michigan Aerospace Corporation had the goal to demonstrate the capability of autonomous docking manoeuvres between satellites on orbit. More recently, Crew-2 Mission [6] performed an autonomous docking manoeuvre with the ISS. Regarding small satellites studies some examples are: Synchronized Position Hold, Engage, Reorient, Experimental Satellites (SPHERES) [7] aboard the ISS that consists of a series of miniaturized satellites developed by the Massachusetts Institute of Technology used to test flight formations, rendezvous and autonomy algorithms in view of future implementations; CubeSat Proximity Operations Demonstration (CPOD) [8] mission led by Tyvak Nano-Satellite Systems focused on a docking manoeuvre of two 3U CubeSats. Finally, regarding autonomous small satellite systems and, in particular, software architectures for autonomous operational capabilities, Astrobee [9] [10] can be cited: it has been built on the legacy of SPHERES and it is composed of three autonomous cubic shape robots that help the astronauts in all their duty in the ISS. Moreover, the University of Padova has a grounded heritage on autonomous docking manoeuvres studies for example: Flexible Electromagnetic Leash Docking system (FELDs) [11] studied an electromagnetic soft docking technology, participating in the Drop Your Thesis! Programme achieving a post-docking mechanical connection with a flexible wire; Autonomous Rendezvous Control And Docking Experiment - Reflight 2 (ARCADE-R2) [12] correctly performed three release operations and two docking procedures between 2-Degrees of Freedom (DoF) vehicles; Position and Attitude Control with MAgnetic Navigation (PACMAN) [13] demonstrated how a Chaser can correct autonomously the relative attitude with respect to the Target through electromagnetic actuators. Moreover, regarding the miniaturized docking interfaces, a probe-drogue solution suitable for small satellites has been developed and proven functional in the laboratory on a low friction table, able to guarantee a docking manoeuvre with tolerance to misalignment [3].

1

ERMES Hardware Design

This chapter will elaborate on the hardware design. This is not a definitive design, because further tests could require a redesign of some parts. A number of constraints were presented in the introduction and are summarised below:

- Chaser and Target are independent from external informations and commands (exceptions for security and starting the experiment);
- The Chaser will approach the Target;
- The manoeuvre must be under 8 seconds;
- The sensors should not rely on external features.

1.1 Chaser

The chaser is a two unit (2U) CubeSat mock-up[1]

It has an alluminum structure that can contain all the hardware and provide stiffness to the mockup. All the hardware inside is positioned with some priorities, some consumables must be chaged during the flight so they must be accessible in a short time. The entire body frame of the chaser is covered with foam sheets. They serve to protect it from possible impacts during the experiment.

A preliminary design of the chaser is shown in Fig. 1.1. The docking port has a movable part that allows to lock the chaser with the target. It also has a sensor to know if the docking port is in the correct position with the other half before the locking. Both the motor and the sensor are yet to be fixed, because ERMES will try to test two different docking mechanism. A common electrical interface need to be defined. In the software part we assumed that the sensor is a digital sensor with two different values: high when the docking interface is in the locking position, low otherwise. The locking mechanism is assumed as a logic output. Both of them are not included in this version of the circuit.

[1]The standard defines a (1U) cubesat rigorously [14]. The standard defines one unit as $10 \times 10 \times 11.35$ cm^3, this allows an internal space of $10 \times 10 \times 10$ cm^3.

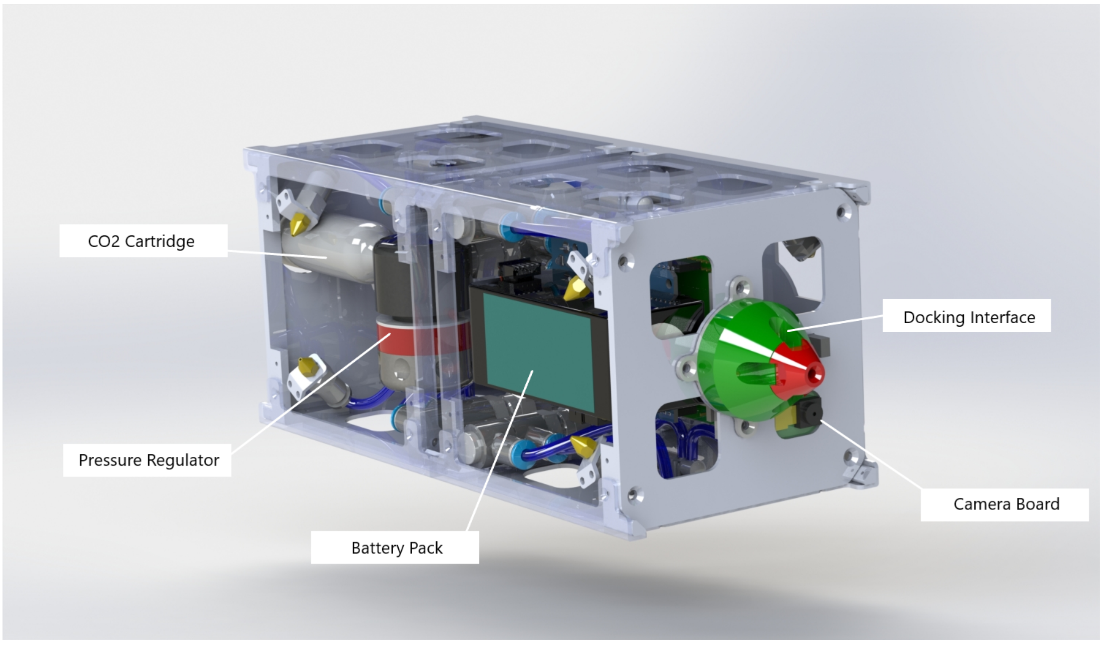

Figure 1.1: Preliminary design of the chaser

1.1.1 Propulsive System's Hardware

ERMES chaser is equipped with a cold gas propulsive system to actuate the position and attitude control. It can control 6 DoF.

The propulsive system is based on expendable CO_2 cartridges and characterized by a set of 8 controlled thrusters. The cartridge contains 16g of CO_2.

The thrusters are tilted by ±45° with the respect to z axis, ±30° with the respect to y axis and ±60° with the respect to x axis. To move or rotate along a single axis four thrusters must be actuated together as shown in Table 1.1. The figure 1.2 shows a design of the positions and orientations of the thrusters. Lets take for example an acceleration along the y positive axis, the thrusters to open are the 1,2,5,6, the resulting force would be along the y axis. Similarly, we can identify all other combinations. To control this configuration the chaser will have

	Translation		Rotation	
	+	−	+	−
x	1 − 4 − 5 − 8	2 − 3 − 6 − 7	3 − 4 − 5 − 6	1 − 2 − 7 − 8
y	1 − 2 − 5 − 6	3 − 4 − 7 − 8	1 − 4 − 6 − 7	2 − 3 − 5 − 8
z	5 − 6 − 7 − 8	1 − 2 − 3 − 4	2 − 4 − 6 − 8	1 − 3 − 5 − 7

Table 1.1: Thrusters to open to perform an acceleration along or around one axis.

a pneumatic circuit better explained by the fig. 1.3. An exhaustive list of the components of the pneumatic circuit is given below:

- `C1`: 16g CO_2 threaded cartridge;

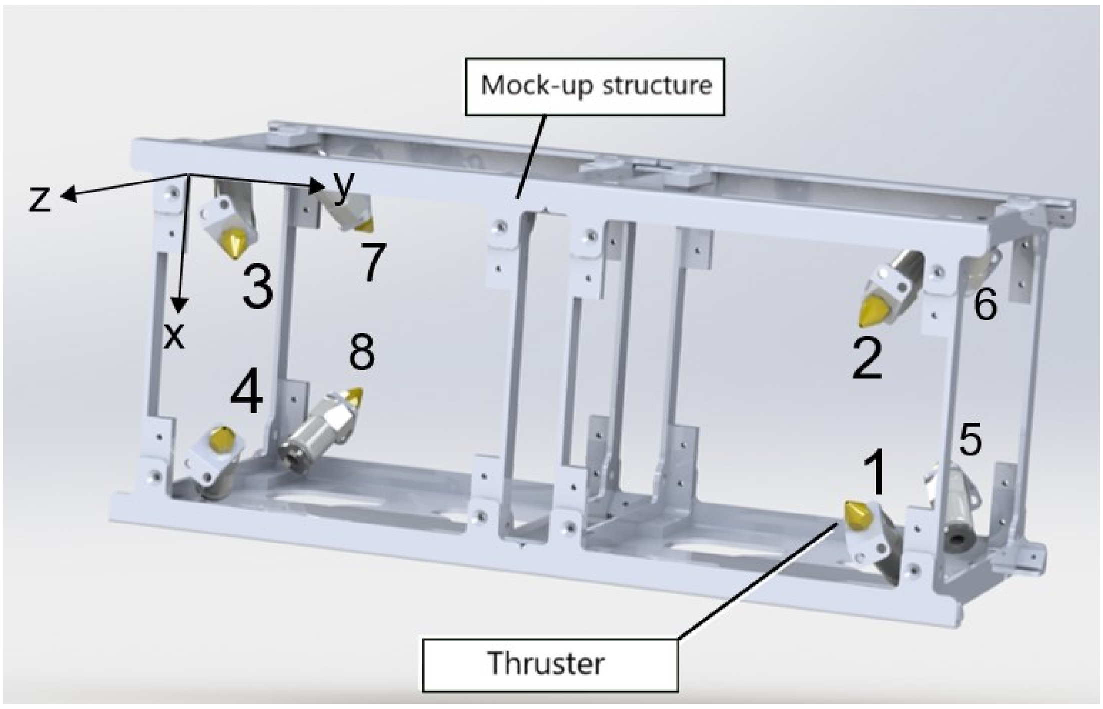

Figure 1.2: The trusters are placed in this configuration, this allows to control $6DoF$ with 8 thrusters

- `PR1`: Pressure regulator;
- `RV1, RV2`: relief valves[2]
- `T1-21` and `D1-5`: Tubing and adapters;
- `SV1-8` Solenoid electrovalves;
- `N1-8` Convergent nozzles.

The CO_2 inside the cartridge is around 57bar at standard condition of temperature, assuming the liquid state. The volume is below 0.5L according to Novespace guidelines (PRES 01-04) [15]. The type of cartridge chosen is commonly used in cycling activities: they are used to inflate tyres, therefore they are easy to find and buy.

The pressure regulator stands a maximum pressure of 190bar of input and can regulate pressure between 0 and 5.5bar. It sets a working pressure around 2.5bar. According to the MATLAB program used to study the fluid-dynamics of the pneumatic system, it allows to generate over 100mN of thrust: in the ambient pressure of the laboratory (1bar) and the parabolic flight environment (0.8bar). This value of thrust allows to have a low-pressure system that generates enough amount of thrust to perform the manoeuvre.

[2]Novaspace requires a two point of failure pneumatic system (REF/PRES 01-04 of Guidelines)

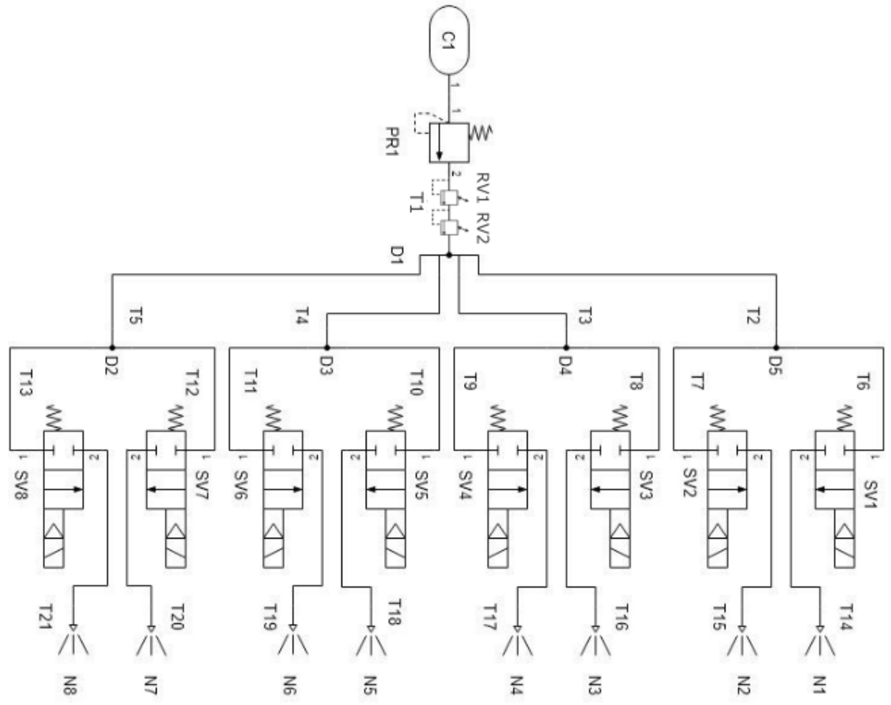

Figure 1.3: This is the propulsive circuit on board of the Chaser

Both relief valves have been implemented for safety reason [15]. They act as a safety relief point for any unwanted increase of pressure. The selected configuration resists up to 10bar in order to avoid any damages to the electrovalves. This guarantees two point of failure as required.

The selected tubes are all PVC 4×1mm due to their relatively high flexibility.

The choosen electrovalves can be controlled by a 12V or 24V input voltage. This allows the CO_2 stream to exit the respective nozzle and provide thrust when the valve is powered. The nozzles are simply convergent, instead of a classic convergent-divergent configuration due to the necessity of avoiding supersonic flows since the experiment takes place at standard atmospheric pressure and not in a vacuum chamber. In fact, supersonic flows lead to shock waves, that could cause unwanted increase in pressure in the pneumatic system and, consequently, damages to the system or simply alter its performance. The choice of a convergent solution allows also to model the thrust output linearly with the respect to the pressure set on the regulator. For pressure higher than 1.8 bar, the flow exits at a sonic state (Mach number equal to 1) and, therefore, the exit pressure is a function of the total pressure set on the regulator.

$$p_{sonic} = p_{total} * \left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}} \tag{1.1}$$

$$T = A_{exit} * p_{total} * \Gamma_{(\gamma)} \tag{1.2}$$

Like shown in Eq. 1.1 and the thrust is linear.

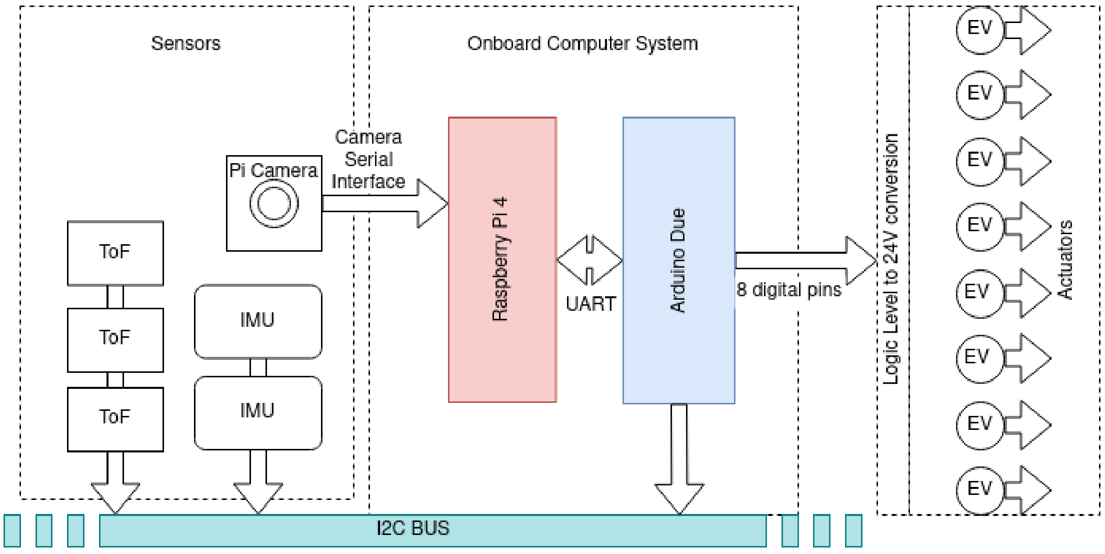

Figure 1.4: An comprehensive scheme of the Attitude and Position Determination and Control System (APDCS) components, and the connections between them.

1.1.2 Attitude and Position Determination and Control System

To achieve an high level of situation awareness of the chaser, different kind of sensors are needed. The chaser also needs an OBCS to process the information and provide adeguate commands to the actuators. The OBCS of the chaser would also calculate the trajectory of the manoeuvre.

The sensors used by the system are:

- 3 Time of Flight (ToF) sensors;
- 2 Inertial Measurement Unit (IMU);
- Raspberry Pi Camera V2 Noir.

The camera is used to implement a computer vision system. The principal algorithm is the AprilTag Detection[3] [16] [17] [18]. It is a fiducial tag recognition system better discussed in Section 3.4.1. The camera is without the infrared filter to allow the implementation of an Infrared Light-Emitting Diode (IR LED) based computer vision system. If implemented, the choice of IR LEDs is to avoid disturbing other experiments in the cabin of the aircraft.

Three ToF sensors will provide the distance of the target. The VL6180x model is selected because it has a range of 20cm. Three DoF sensors can also provide orientation information in particular conditions. They cannot always provide a reliable orientation information because the face of the target is not completely flat. According to the datasheet these sensors are pretty unreliable under 1cm [19]. The Figure 1.5 reports the typical range performance measured

[3]The AprilTag 3 is the version used in ERMES

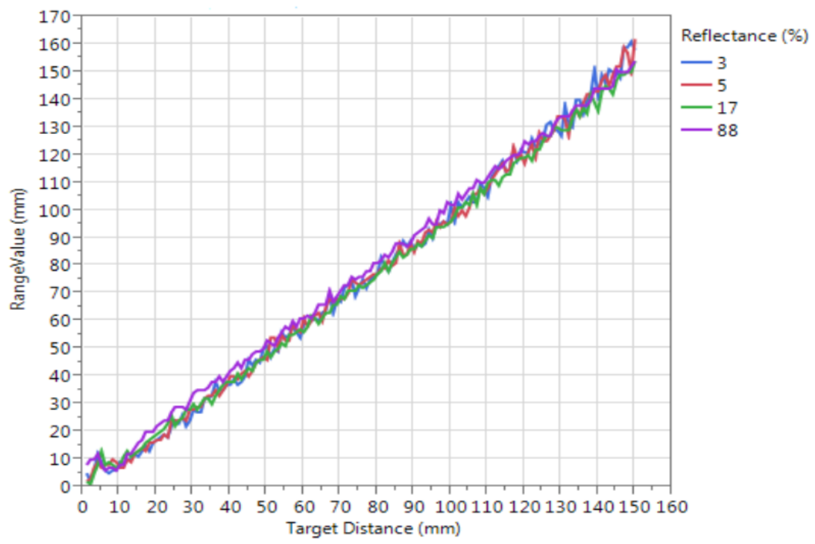

Figure 1.5: Typical ranging performance of VL6180x as reported by the datasheet

by the manifacturer. According to the datasheet the software will consider their measures below 1cm as invalid. So their measurements are considered only in particular conditions.

A good argument is that two IMU are useless or a risky choice, because if they provide two measures in contradiction between them it is a problem. ERMES will use two sensor positioned in different way to achieve a better estimation of the real state of the system. Sections 2.2.1 and 3.4.3 present methods to obtain a good result from this combination of sensors. Focusing to the hardware selection, the IMUs are two Pmod NAV sensors, they have the LSM9DS1 chip module that provides a 3-axes accelerometer, 3-axes gyroscope and 3-axes magnetometer. If future tests show that there is a need for a more precise sensor, one of the Pmod NAV sensors can be kept, to be coupled to a sensor with higher accuracy, like the Epson M-G364PD or the Phidget Spatial 1044_1B. The choice of Pmod NAV is due to its wide use in a lot of projects and its compatibility with Arduino. It is also an inexpensive sensor compared to other IMU in the market. We think that fusing of a lot of cheap sensors measures could be good enough for our purposes. But, because the hardware design isn't frozen yet, the software must consider the possibility that the IMU model will change.

The actuators are the nozzles that are activated by the opening of the corresponding solenoid valves. As mentioned above, the solenoid valves open with a 12 or 24 volt voltage, so a suitable circuit needs to be implemented to transform the low voltage signals (3.3V in the case of the chaser) from the OBCS to the voltage needed by the solenoid valves. It was therefore decided to use a logic level MOSFET[4] as a switch, which will open the power supply to the solenoid valves when the digital signal is high. Each signal is generated by a predetermined digital

[4]Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)

port of the OBCS, and the signal generation is based on Pulse-width modulation (PWM) management, like explained in the Section 3.3.1.

The computer units are the Raspberry Pi 4 and the Arduino Due, both are part of the OBCS. The Raspberry will provide the computational power to achieve an autonomous guidance and navigation. The Arduino is used to provide a low level computing system, better suited to generate the PWM signals. The two units are connected together with a serial connection, using the UART[5] devices made available by the outputs of the boards. The camera is connected to the Raspberry through the Camera Serial Interface (CSI), all other sensors are connected to the Arduino via a common Inter Integrated Circuit (I2C) bus. The software design assumes as mandatory to connect the camera to the Raspberry, and at least one of the IMU to the Arduino.

The Raspberry and the Arduino versions are choosen based on the compatibility with the software libraries used. There is a valid argument about the version of the Arduino, because an Arduino Nano could achieve the designed task. But, due to the amount of data that needs to be read and processed by the Arduino we opted to the safer choice. The different in weight is negligible if compared with other systems in the chaser, like the eight electrovalves or the battery rack.

The ToFs and the IMUs use an I2C connection with the computer system. If both IMUs will be connected to the arduino there could be also an I2C multiplexer. This is because, according to some tests, the data flow is more reliable using a multiplexer instead of changing the chip selection pin of both accelerometer and magnetometer. For the three ToF sensors, it is possible to change the I2C addresses at the startup.

1.1.3 Electronics and power supply

Battery

The chaser needs a battery to power the APDCS described in the previous paragraph. The safety parameters to which ERMES is subject require that lithium batteries cannot be implemented. Batteries and items that contain batteries are sensitive components as they may overheat and ignite in certain conditions and, once ignited, may be difficult to extinguish or may spread corrosive substances. That is why specific cares and limitations apply when batteries are embedded in experiments. Limitations are based on the IATA requirements dealing with the transportation of batteries and on the parabolic flight particularities. In view of the above, the choice was made to use rechargeable NiMH batteries. Additional precautions must be implemented, so the battery will be enclosed and protected from short circuit. A switch should be able to isolate the battery from the rest of the circuit, all the circuit needs to be compliant to design rules against fire. This means also that all the cables must have the appropriate size.

According to all of the safety requests, we think that is better to buy an appropriate battery pack already assembled. The chaser can use a battery of 24V,

[5]Universal Asynchronous Receiver-Transmitter (UART)

it will be composed by many different cells, already connected and encapsulated by the vendor. On the market there are different capacities, ERMES will opt to the smallest battery possible that allows to accomplish at least five repetition of the experiment, and with a 20% of safety margin. That's because the battery is the heaviest component in the free floating object, so it makes sense to optimize the weight of it. During the flight it is possible to carry additional batteries provided they are in suitable containers, and to change them between breaks in the series of parabolas. It would be useful to size the battery so that it can only be changed after 15 parabolas, i.e. during the longest break, but achieving such a long battery life could cost in terms of premature consumption of the CO_2 cartridges.

Changing of battery means that the system needs a complete restart, between each parabola there is 1 minute and 40 seconds, and in this time one member of the team has to enter the safety net and reposition both mockups. In this time it is unthinkable to change the battery, and probably also the cartridge. In the preliminary design, a battery change was programmed after 15 parabolas, and a cartridge change every 5. It has been proven in previous tests[6] that a cartridge can keep the flow from one nozzle for 40 seconds. Now one could argue that each experiment last 8 seconds, more or less, and needs to activate 4 nozzles at time. But it is necessary to consider that the thrust is not always on, and the chaser has an initial velocity due to the release system design. So, in the end, the possibility to change the cartridge every 5 experiment is not so far from reality.

If the cartridge cannot last safely for 5 consecutive experiments, it makes sense to reduce the battery capacity and change it every 5 experiments, just like the cartridge. This could reduce the consumption of the CO_2 used to perform the manoeuvre. Additional precaution could be taken to reduce the propellant use, but because they are related to the software, they are better discussed in the Section 2.2.2.

Electrical Circuit

The circuit can be divided in 3 main sections: power management, logic level boards and the interface with actuators.

Each section can be outlined by its function. The power management includes the battery and the necessary circuit to step down the voltage to power the other parts. The logic level boards are characterized by voltages around 5V or 3.3V, they are the Arduino Due and the Raspberry Pi 4, and the converter from the logic level signal to the 24V allows solenoid valves to be opened.

The block scheme represented in Fig. 1.6 shows the connections between the main parts. The battery power the entire system with a step down for the two computer boards. The three ToF sensors are directly connected to the I2C of the Arduino. From each ToF and IMU there is a chip selection pin connection.

[6]Some of those tests are mentioned in this dissertation because they also regard the control system

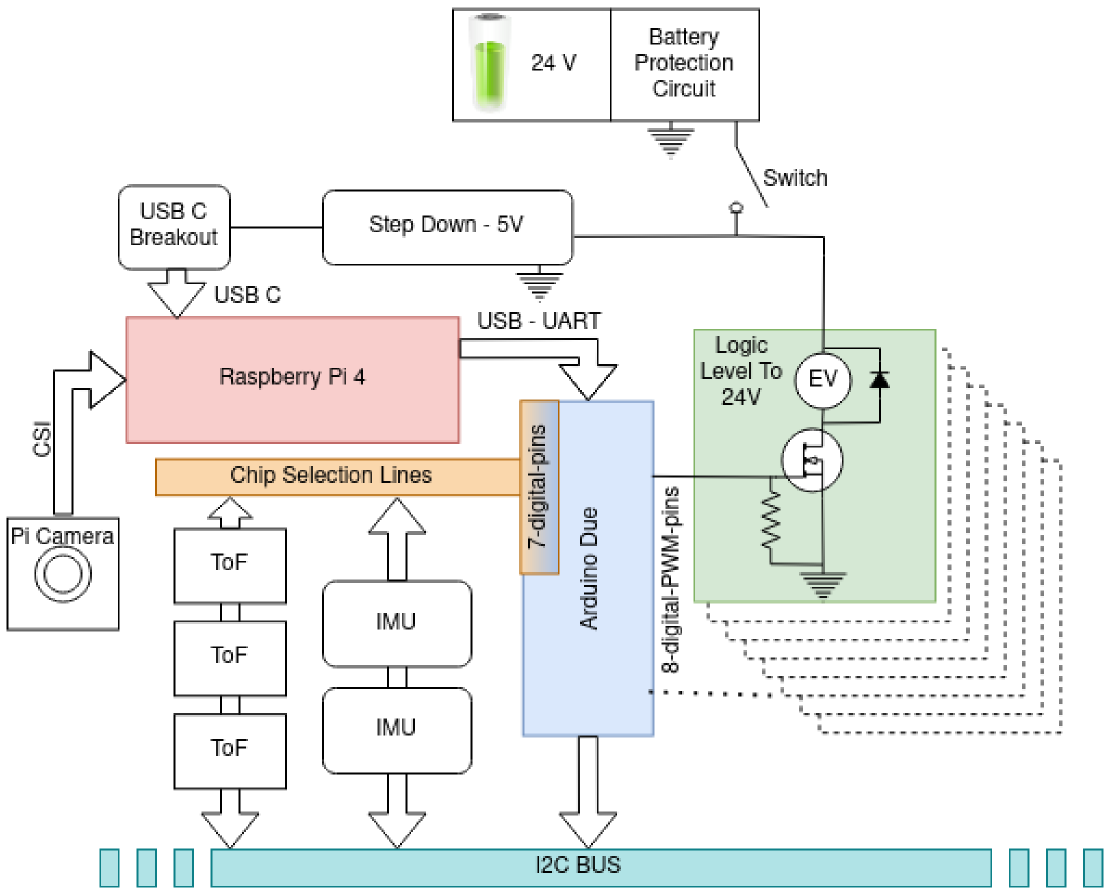

Figure 1.6: Electrical Circuit Scheme

The Arduino is connected to the Raspberry via a standard micro USB[7] cable, this provides also the power input required.

After the battery there is a protection fuse for the computer units, and another fuse for the electrovalves. The step down provide an usb connection with $5V$ output. Note that the Raspberry will power itself and also the Arduino. The Pi Camera is connected with the CSI to the Raspberry.

The last part is the circuit that transforms the logic level signal to an appropriate signal to open the EVs[8], for each electrovalve there is a MOSFET and a resistor. The diode is a freewheeling. All of the signal cables have a relative small size, but the connection of the high power part of the circuit must have appropriate sized cable.

Another important note is that the battery will have a connector that does not allow a reverse polarity connection. That's for safety reasons, and also because the battery needs to be changed in flight.

The Arduino Nano Alternative

Like mentioned before, the chaser will use the Arduino Due. But there is an argument about the choice of an Arduino Nano instead.

The Arduino Due has a lot of advantages, for example it has 11 PWM pins, it has a lot of memory and a very good processor. Those features are very useful in a testing phase, like it will be explained in the section 3.3.

The main drawback is that consumes a lot of power. Even if it has some low power modes [20].

From the circuit described above we can substitute the Arduino Due with the Arduino Nano and a PWM board. If the PWM board can output up to 1.2A per pin and 12V it can substitutes the circuit that converts the logic level signal: the green part of the Scheme 1.6.

1.2 Target

The target is a 1U CubeSat mock-up. Like the chaser it has an alluminum structure to contain the hardware and provide stiffness. All its edges and corners are covered with protections to prevent damage from shocks. The passive drogue of the docking interface is attached to the front face of the cubesat and it has no sensor or activation mechanism. On the same face are the fiducial tags necessary for the localization by the chaser. If the use of IRLEDs is required, they will also be placed here in a known pattern.

The target has three reaction wheels to allow it to control the 3DoF needed to maintain its attitude. The reaction wheels are enclosed in a case, and they

[7]Universal Serial Bus (USB), the Arduino Due has two ports: native port and programming port, in ERMES we use the programming port to connect the two devices

[8]Electrovalve (EV)

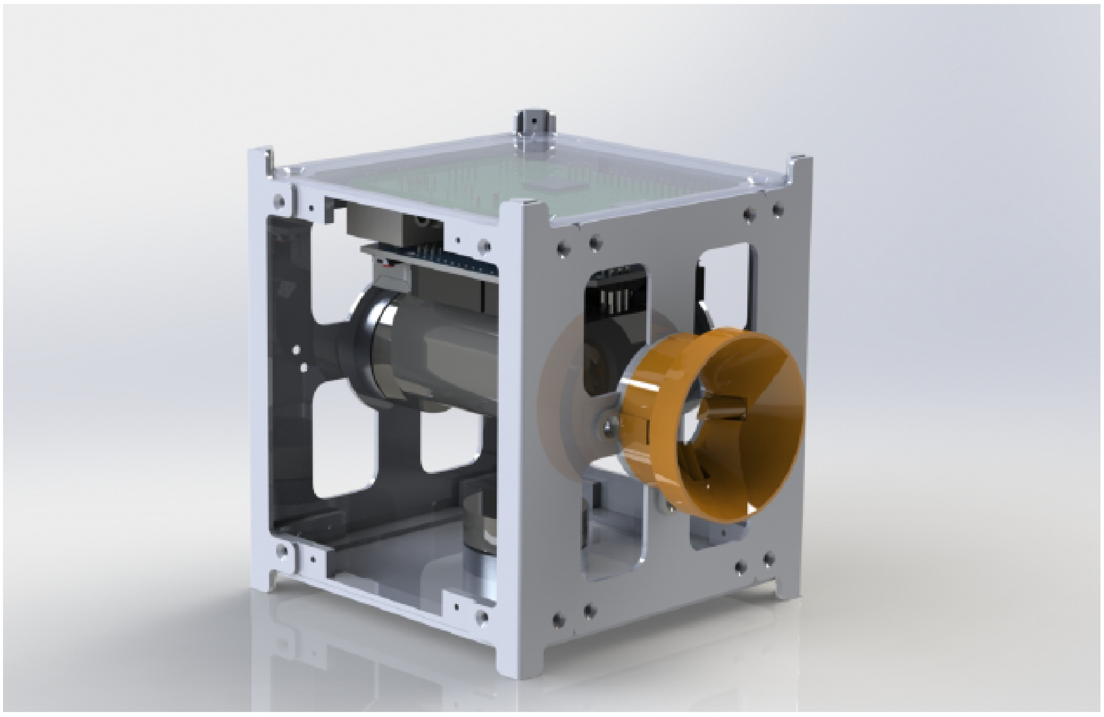

Figure 1.7: Preliminary design of the target

are composed by a DC motor connected to a flywheel. They will spin in different velocities to allow the target to maintain the correct attitude.

The preliminary design of the cubesat mockup is shown in the Figure 1.7.

The OBCS of the target is composed by an IMU, an Arduino Mega board, and 3 DC motors. The OBCS is powered by a battery. The implementation of the battery will follow the same rules of the chaser's one, but the battery will be smaller in both voltage and total capacity.

In order to allow testing of the target control system, a WiFi interface must be implemented, which could also remain for the final design, as it could become useful in the case of a manoeuvre that involves sending certain commands to the target by the chaser, as it is explained in Section 2.2.2. In fact the Raspberry Pi 4 of the chaser has both bluetooth and WiFi interfaces and it could send commands via wireless.

1.3 The Release System and External Setup

To support the manoeuvre ERMES needs some fixed hardware. This added hardware provide these functionality:

- release of the experiment;
- collecting data;
- power supply;

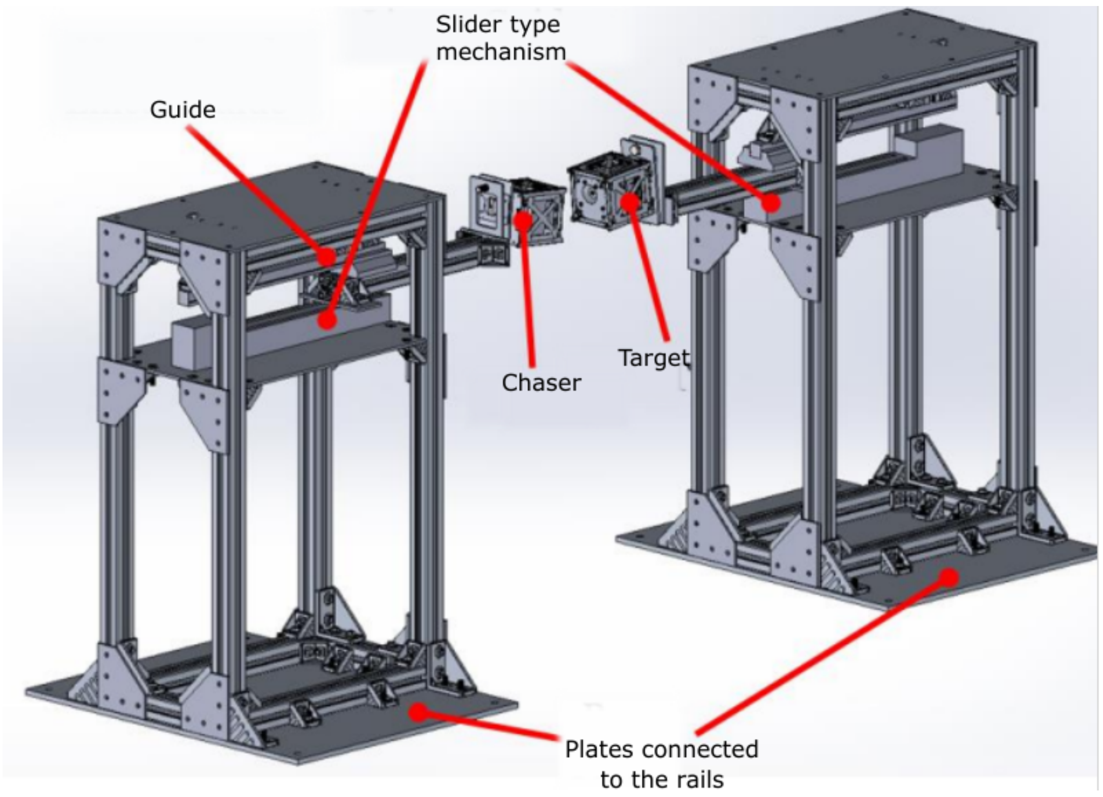

Figure 1.8: Release structures of the experiment

- telemetry monitoring base.

The experiment area contains two structures for the release of cubesat mockups. They are built according to novespace guidelines, and they are fixed on 2 baseplate on the aircraft floor. In the Figure 1.8 there is a preview design, but the final appearance of each structure will be enclosed with panels, then covered with foam like mentioned before.

These structures provide a release system based on linear guides. At the end of each linear guide an electromagnet is fixed which will hold the corresponding cubesat. The linear guides allow the acceleration of the chaser, the release of the mockups, and the withdrawal of the electromagnets so as not to interfere with the experiment. Those structures will be similar of the ones used by PACMAN, mentioned in the introduction. Due to the nature of the parabola, it is a good idea to implement an automated release system for the experiment. The goal is to release both mockups when the perceived acceleration is as close to 0 as possible. To achieve this goal it is necessary to add an IMU unit attached to the structures, and connected with the supporting laptops.

The experiment area is completely enclosed in a safety net. The floor inside is covered with mattresses to reduce impact damage during the experiment phases. Outside the net there are two support plates that provide anchorage points for the external hardware supporting the experiment. One of them is used to fix the power distribution units, such as the distribution socket and the power supplies needed to keep the experiment's electronic instruments running. In the case of ERMES there will be two power supplies for the laptops and another to power the linear guides and their electromagnets. Each laptop is attached to another support plate, which in addition to the laptops, is used to store the cartridges and the additional batteries. ERMES needs two laptops, one of them is used to saving the data, starting the experiment and monitoring the telemetry, the other one is a backup in the case the first one fails.

To retrieve additional data for the experiment, there will be 2 cameras positioned in the upper part of the safety net. They should be able to observe and register the experiment from 2 different angles. The precise model of the cameras has yet to be defined. Their features include internal memory and the ability to record in 1080p at 60fps, and if possible, they should be battery powered. Their video will be used after the experiment to analize the manoeuvre. Another option is to consider a wide angle camera, like the one used in the PACMAN experiment [13].

The amount of data registered in the experiment needs additional memory space. The external storage media should be provided for data and backups. The solid state drives should be preferred to achieve this goal, because the hard disks have moving parts, and they could experience some problems during the parabolas[9].

[9]The hard disk park the lens at the border when they are off, to prevent the damage of the disk itself. In ERMES experiment they should be on during the parabolas to save the data, so this is a possible point of failure.

2
Software Design

This chapter describes the software architecture decided upon for the system and attempts to explain the interaction between the various main components. The implementation will be discussed in the next chapter, while this one focuses on outlining software features and constraints.

The software can be divided into three different levels, each corresponding to a different computational unit. Each level provides functionalities and has different constraints. Each component belongs to a certain level and can interact with others. Figure 2.1 shows a general diagram of the architecture with an outline of the components and their interactions.

Among the main components, the four key ones are Guidance, Navigation, Control and Actuator Control System. Starting with the last one, in the ERMES project the actuation control means a component capable of executing simple commands. This subsystem allows the chaser to move or rotate with respect to a single certain axis. Guidance, Navigation and Control interact so closely that they are usually referred to as GNC, denoting the whole system. In the case of the chaser, these subsystems are described separately, but they still interact closely with each other.

Guidance is used to calculate the trajectory to be followed by the system. This is calculated using the location data provided by the navigation. Finally, it communicates the planned trajectory to the control system.

Navigation, also referred to as localization, is responsible for estimating the position of the target relative to the chaser, and the chaser state (position, orientation, velocity, acceleration). To do this it must use the information produced by the sensors, including the vision system. In addition to the sensor information, localization uses movement estimates calculated by the control system.

Control refers to the manipulation of the forces required to execute the calculated trajectory. In the case of ERMES, the applicable forces are dependent on the opening of the corresponding nozzles. Therefore the control component must calculate the commands to be communicated to the low-level system, in the form of direction, applied force and ignition time.

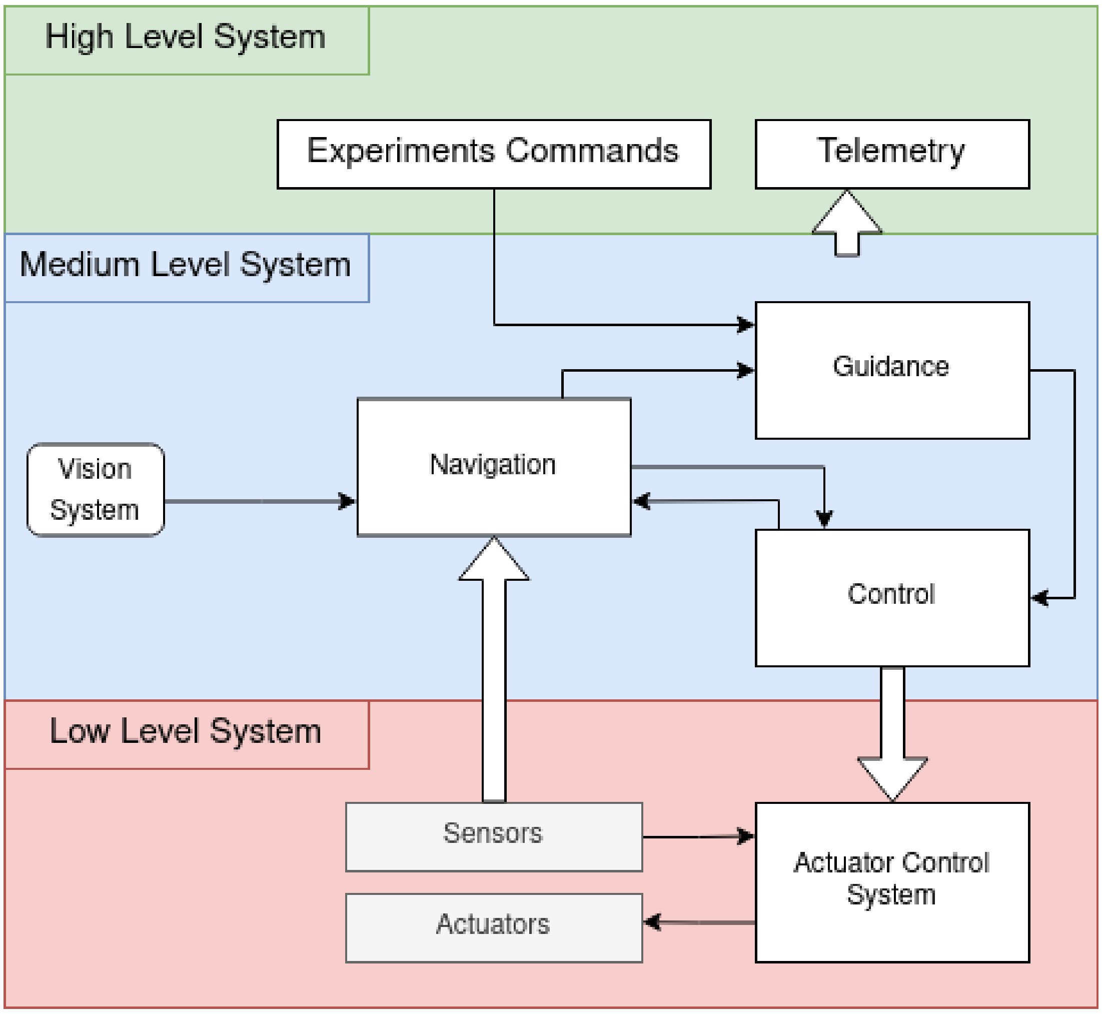

Figure 2.1: Software architecture divided by levels and grouped by functionality. In gray the interfaces with the respective hardware.

Outside the main components, it is necessary to interface with the experiment by the users. Therefore, the high-level system must provide the necessary interface both to start the experiment and to observe its telemetry.

Referring instead to the computational unit that deals with each level, in ERMES project it was decided:

- the Low Level System (LLS), with the actuator control system, runs on the Arduino Due;
- the Medium Level System (MLS), with the guidance, navigation and control, runs on the Raspberry Pi 4;
- the High Level System (HLS), that takes care of the communication between the operators and the chaser, runs on the laptop.

This subdivision is partially inspired by the work of the Astrobee team [9] [10] [21]. The Astrobee system consists of three cubed-shaped robots, software and a docking station used for recharging. The robots use electric fans as a propulsion system that allows them to fly freely through the microgravity environment of the station. In this system there is a particular feature interesting for ERMES project: the free flyer can autonomously dock with its charging station. ERMES needs to dock with another free flying object, but some techniques used for Astrobee are useful in ERMES project too.

2.1 Low Level System

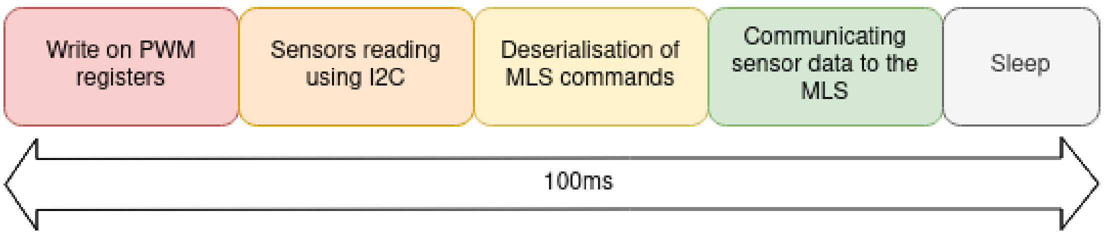
Write on PWM registers
Sensors reading using I2C
Deserialisation of MLS commands
Communicating sensor data to the MLS
Sleep
100ms

Figure 2.2: The tasks to be performed by the Low Level System within the time limit. In the ERMES project it is set at 100ms

The low-level system is a cycle that repeats itself at a predefined interval, and performs the following activities (also shown in Figure 2.2):

- controls the opening of the EVs;
- reads data from sensors connected to the I2C line;
- deserialises commands received from the MLS;
- packages and sends sensor data to the mid-level system.

LLS activities must be performed within a certain time limit. It is set at 100ms by the design. This time limit must be strictly adhered to, i.e. no cycle repetition must exceed 100ms. It means that the system is hard real time [22]. This is because if a deadline is missed, the next command may interfere with the command still running, resulting in less stable control. Due to the difficulties of an hard real time system implementation, this level should have the minimum level of features required to work. For the ERMES project the feature to control the opening of the EVs is essential. Reading commands from the MLS is linked to the previous activity, so it is also considered vital to the LLS. Reading sensors is instead optional, during ermes development it was chosen to keep the management of sensors connected to the I2C line at this level. This is because after some tests made, it was verified that it is possible to maintain the time deadline even if this activity is inserted. Given the premises, the choice is also correct because it allows for more sophisticated future developments in the event that the current implementation is not sufficient to guarantee system stability: for example, it is possible to consider IMU data to control the EVs with the IMU feedback.

The communication between the LLS and the MLS is serial based on UART. The information transmitted in commands from the MLS contains the direction of thrust, the force of thrust, and the time expressed in number of cycles. Messages from the LLS to the MLS must contain sensor data. Apart from the previous two, there is no other form of communication between the two levels.

Given the premise of communication, the LLS must be able to understand which nozzles to open when a command is given, according to the Section 1.1.1. It must also send the correct signal to the outputs PINs to which the actuators are connected to guarantee an opening consistent with the force expressed in the command received.

2.2 Medium Level System

The MLS relies on the localization, and on the coordinate system of the experiment. In ERMES the reference system follows the conventions outlined in the REP-103 document [23].

Starting with the reference system of the free floating objects, the center of mass is the origin, the axes are disposed according to the standard: the x positive axis is the forward one, y positive is the left and z positive is the up. All systems are right handed, this means they comply with the right hand rule. In both the chaser and target, the front face is the one with the docking port. The top face, in the case of ERMES, can be any of the side faces: it has been decided to be the one facing upwards while the objects are still fixed in the release system. The reference system of the chaser and the target is shown in Figure 2.3, the colour scheme used for the axes was chosen for consistency with the programs used in the development.

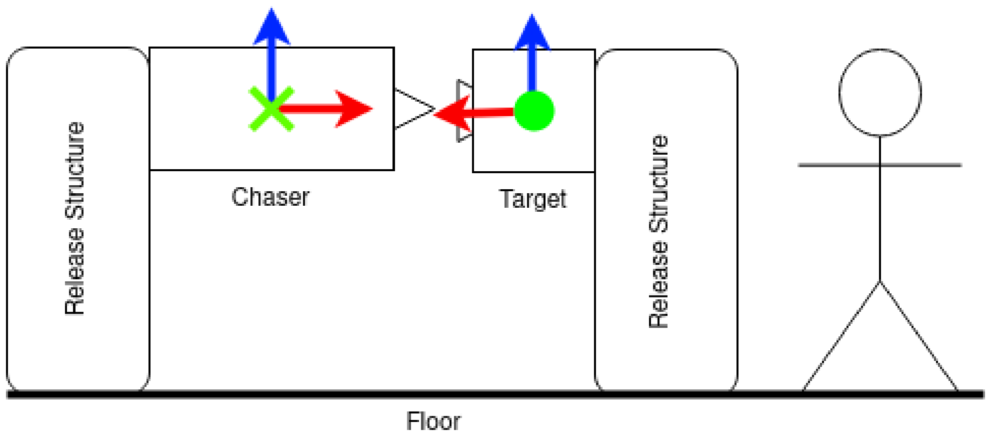

Figure 2.3: The axes are x in red, y in green, z in blue. The $\times$ means that it enters the sheet, the circle means that it comes out.

In this thesis, the target software will not be discussed in detail, but some features are key to understand some design choices of the chaser software. The first one is the ability of the target to maintain a given attitude. According to the design its software should start to control the orientation just after the release. The second key feature is the ability to communicate via WiFi. This feature is mainly used to test the software of the target. In the chaser control system, the target's WiFi capability is used to communicate attitude commands when using a collaborative manoeuvre, as described in section 2.2.2.

2.2.1 Navigation

In the control theory, the navigation is the determination, at a given time, of the state of the vehicle, in this case the chaser. Its state is determined by its position, velocity, and orientation (attitude). In ERMES the state will include the acceleration along each axis. In the navigation system, the state of the chaser s_c, referred to a coordinate system F_1, is defined by:

- the position referred to F_1
- the velocity relative to F_1, but referred to F_c, where F_c is the coordinate frame of the chaser itself (as described in the section before);
- the linear acceleration relative to F_1, but referred to F_c;

and it can be expressed by:

$$s_c = \begin{bmatrix} x_{F_1} & y_{F_1} & z_{F_1} & roll_{F_1} & pitch_{F_1} & yaw_{F_1} \\ \dot{x}_{F_c} & \dot{y}_{F_c} & \dot{z}_{F_c} & r\dot{o}ll_{F_c} & pi\dot{t}ch_{F_c} & y\dot{a}w_{F_c} & \ddot{x}_{F_c} & \ddot{y}_{F_c} & \ddot{z}_{F_c} \end{bmatrix} \tag{2.1}$$

For example, if the state of the chaser with respect to the target reference system is considered (depicted in Figure 2.3): the first six terms represent the position of

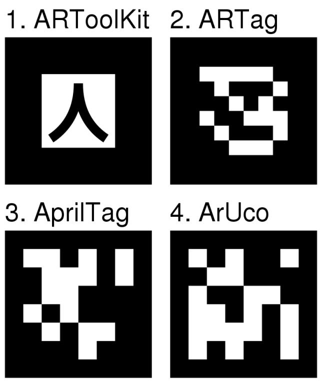

Figure 2.4: Comparison of some fiducial markers used in computer vision applications

the chaser with respect to the target, the other terms indicate the velocities along and around the axes of the chaser's reference system with respect to the target. To determine the status, the chaser needs data from sensors, including a vision system capable of locating the target. There are a lot of solutions for a computer vision based system, but ERMES focused in a fiducial tag based recognition. Fiducial tags are objects positioned in the field of view of the camera for use as a point of reference or a measure, examples are shown in Figure 2.4. In ermes, fiducial tags are used to recognise the position of the target, so the target will have at least one tag with a defined pattern and a known size. The computer vision software will recognize the sticker and compute the position and orientation of the target with respect to the camera. Another computer vision algorithm to consider is the IR LED based system. It has taken a back seat because it complicates the design of the target circuit, and also there are some known issues with the use of a LED based system in the airplane cabin: it may interfere with other experiments[1].

The sensors data should be passed by the low level system, so the medium level system can process them. Considering the nature of the data recorded by the sensors, in addition to the position relative to the target provided by the vision system, there are also:

- the acceleration along the axes and velocity around them, provided by the IMU;
- the relative distance between target and chaser, provided by the 3 ToF.

Information from the control system is also used in determining the status of the chaser. At each control cycle, a predicted state is generated in accordance with the command sent. In ERMES, the control system sends commands in the form of an applied force. The mass of the chaser is known. Therefore the Control can generate a prediction of what the next state of the chaser will be. To do this it

[1]In the last flight campaign (2021) one experiment used a LED based system to locate one object, and it interfered with another experiment. There are precautions that could be taken to avoid it, so if future tests show that it is an essential feature, the additional work will be justified

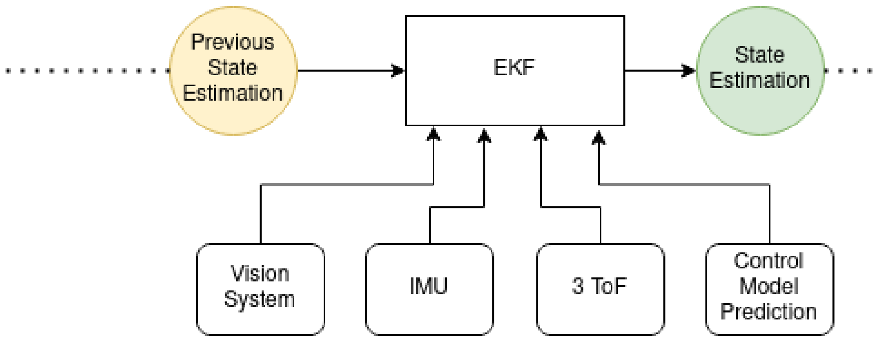

Figure 2.5: The Localization component will estimate one state at every step. This is the scheme for the target-related localization

must integrate acceleration with respect to time, and use the previous state as an initial condition. Given the previous state of the system s_{t-1}, and an acceleration along x positive of $A_x \mathrm{m/s^2}$, considering 100ms as the time step, the control will generate the $\hat{s}_t$. The following equation shows the calculation of the $\dot{x}_{\hat{s}_t}$ term as an example:

$$\dot{x}_{\hat{s}_t} = \dot{x}_{s_{t-1}} + A_x * 0.1 \tag{2.2}$$

The first state used by the control is initialised with the initial conditions of the experiment, which are known. The navigation, considering all the data from the sensors explained above, considering the predictive model of the control system and also considering the status generated in the previous step, will generate the estimate of the current status of the chaser. Considering also that each piece of information has an uncertainty associated with it, the localization can also estimate the uncertainty of the current generated state. To do this, a filter is needed, and in particular in ERMES it was decided to use the Extended Kalman filter (EKF). It is the nonlinear version of the Kalman filter [24] which linearizes about an estimate of the current mean and covariance. The Figure 2.5 shows a diagram of how localization works at each step.

In this design, the localization finds the estimate of the chaser state referred to the target coordinate system, shown in the right of the Figure 2.3. This could be a problem for the control algorithm: for example, if the target rotates a few degrees on itself to control its attitude, there will be an apparent velocity in the previously explained localization estimate. In order to solve this problem, a dual localization system is implemented, consisting of both the system explained in the paragraph above and an identical system but relying only on a few sensors. The second system will base its estimation only on data from the IMU, and on information calculated by the control's predictive model. Its initialisation will be the same as the previous system, because the initial state of the experiment is known. The coordinate system of this Localization is the position of the target at the initial moment of the experiment. As the manoeuvre continues, the position estimate given by this system will tend to accumulate error. However, this estimate is continuous. In the previous system, assuming that the chaser momentarily loses

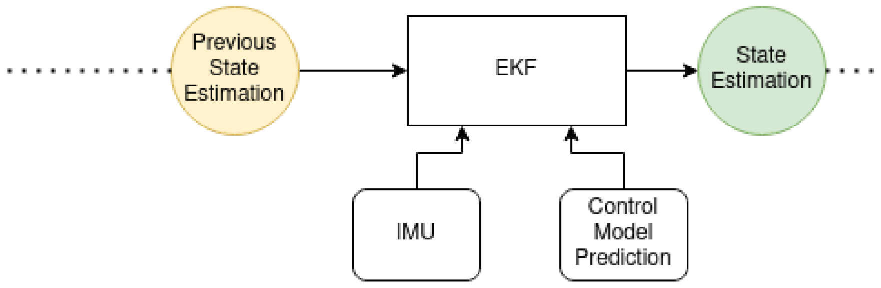

Figure 2.6: The Localization component will estimate one state at every step. This is the scheme for the odom-related localization

visual contact with the target, and regains visual contact shortly thereafter, new data from the vision system would correct the estimate, which would instantly jump to the new position. This position jump would result in an untrue speed estimate. In the second estimator, however, since there are no sensors dependent on external factors included in the estimate, there is no such problem. A diagram of this second estimator is shown in Figure 2.6. For convenience, the reference system of this estimator will be called "odom".

2.2.2 Guidance

Guidance refers to the determination of a trajectory from the chaser current location to the target as well as desired changes in velocity and rotation.

The trajectory in ERMES consists of a series of states, each containing position, rotation and linear and angular velocity information. The general state s_i is represented as:

$$s_i = \begin{array}{l}\big[x_{F_t} \quad y_{F_t} \quad z_{F_t} \quad roll_{F_t} \quad pitch_{F_t} \quad yaw_{F_t} \\ \qquad \dot{x}_{F_c} \quad \dot{y}_{F_c} \quad \dot{z}_{F_c} \quad r\dot{o}ll_{F_c} \quad pi\dot{t}ch_{F_c} \quad y\dot{a}w_{F_c}\big]\end{array} \tag{2.3}$$

Where F_t means that it relates to the target reference system, and F_c means that it is the velocities along and around the orientation axes of the chaser. To simplify, it is assumed that all velocities are always zero in the intermediate states of the trajectory. The only exception is the final state which will have $\dot{x}_{F_c}$ equal to the velocity decided for the docking port according to its limits. Thus, the state s_i becomes:

$$s_i = \begin{bmatrix} x_{F_t} & y_{F_t} & z_{F_t} & roll_{F_t} & pitch_{F_t} & yaw_{F_t} & \dot{x}_{F_c} \end{bmatrix} \tag{2.4}$$

The generation of the trajectory is entrusted to a planning strategy, executed by a component called "planner". The planner must generate a series of states. Between one state and the next the value of only one term must change. The planner must also generate execution time limits for each state, taking into account the physics of the system. In ermes, 3 planners are considered: naive planner, trapezoidal planner, collaborative planner.

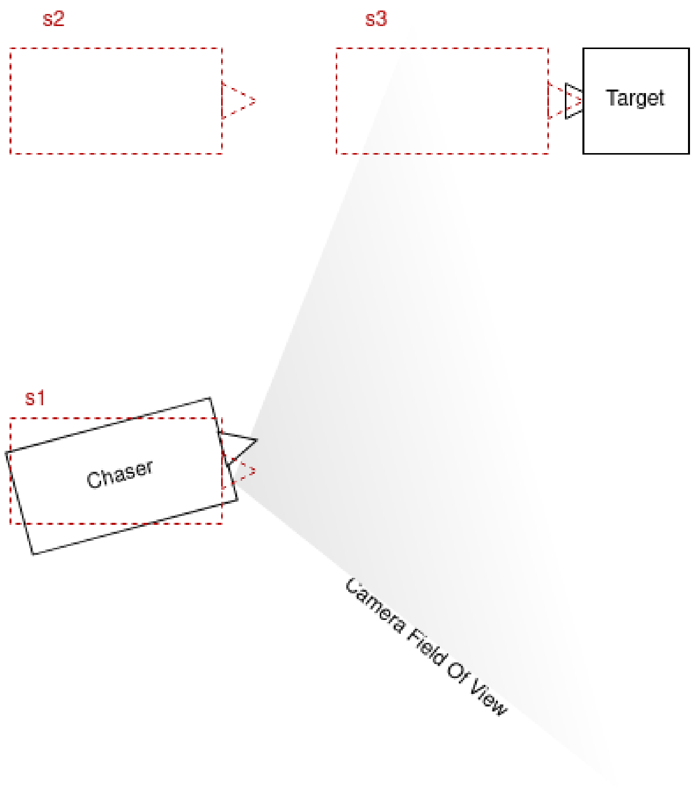

Figure 2.7: This is a 2D example of a trajectory generated by the Naive Planner

Naive Planner

This planner, considering the estimated position of the target, will generate a series of states whose values change one by one starting from the penultimate one. Figure 2.7 shows an example of a planned 2D trajectory.

In order to plan the time limits, must find t_i to minimise the following:

$$\min \sum_i t_i \tag{2.5}$$

Where t_i is the time between the previous state and the state i. And according to the following constraint:

$$t_i \geq T_{p-min} * (1 + A_{tr}) \tag{2.6}$$

Where T_{p-min} is the minimum time to move from the previous state to the s_i according to the physical model, and A_{tr} is a parameter decided in the project that represents the relative added time to allow the adjustment commands.

This planner is very simple, but if the target exits the camera field of view, the chaser loses the relative localization: if the field of view of the chaser camera

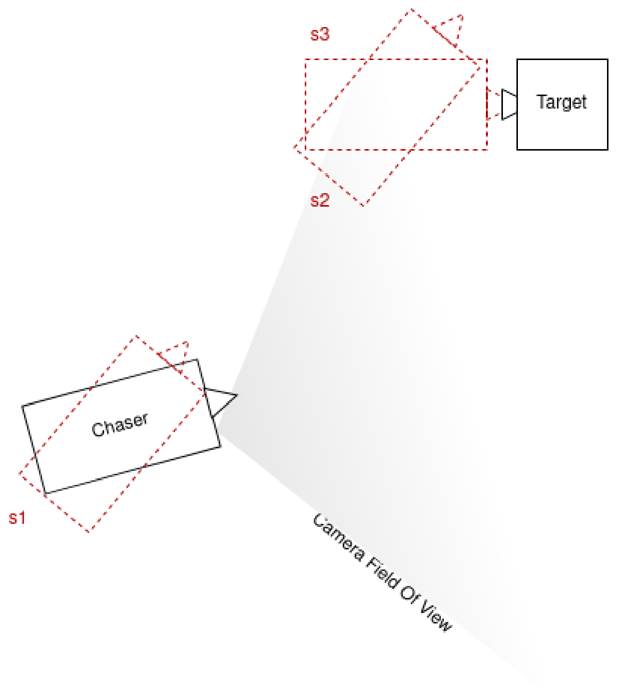

Figure 2.8: This is a 2D example of a trajectory generated by the Trapezoidal Planner

in the Figure 2.7 were narrower, the navigation would lose the information given by the vision system.

Trapezoidal Planner

To solve the field of view problem, ERMES takes inspiration from Astrobee software, in their implementation they use a trapezoidal planner. It plans a straight line between different poses, with terminal velocity equals to 0.

In the ERMES case scenario the planner needs to:

- generate a series of states with a position unchanged but with the orientation equal to the direction of the vector from the center of mass of the chaser and the desired position;
- generate a state in the desired position;
- generate a series of states in the same position but with the orientation equal to the direction of the vector from the center of mass of the chaser and the docking port of the target. The velocity along this direction is lower than the maximum allowable velocity.

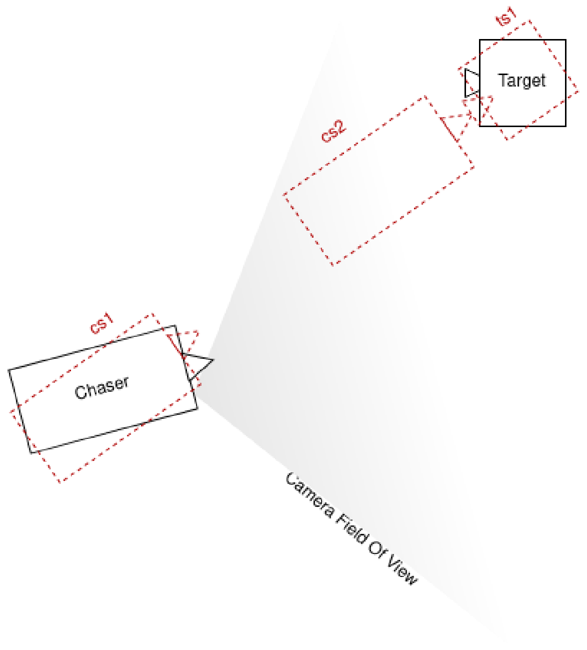

Figure 2.9: This is a 2D example of a trajectory generated by the Collaborative Planner

In this way, the chaser will always have visual contact with the target except at the end, when the camera may not be able to focus anyway. This planner can be easily understood by looking at Figure 2.8

Collaborative Planner

Lets start by optimizing the trapezoidal planner: the key movement that slows the system is the change of orientation.

The goal is to send to the target appropriate commands, via WiFi, to get it oriented towards the chaser, using the localization provided by the chaser itself. The generation of states is similar to the previous one, but some are referred to the chaser, and some to the target.

The states generated by the planner will be:

- **Target**, a series of states to adjust the orientation of the target, so that its positive x-axis points to the position of the centre of mass of the chaser;
- **Chaser**, generation of a series of states to adjust the orientation of the chaser so that its x-axis points to the centre of mass of the target;
- **Chaser**, generation of the final state near the target but with a velocity towards it different from zero.

Figure 2.9 gives an example of the states generated by the Collaborative Planner in a 2D trajectory.

Considerations on Planning Strategies

The planner usually calculates the trajectory considering the collision avoidance, in the ERMES experiment this feature is not necessary. But, the same group of motion planning algorithms are designed to allow the specification of various constraints, for example in terms of time or fuel consumption. From this point of view there is the possibility to use one algorithm to optimize the trajectory planned by the chaser. Unfortunately, these approaches require a lot of development and optimisation, and in the case of the ERMES project there is not enough time. Furthermore, the calculation of the trajectory itself with these methods may take too long in terms of computation. The planners considered above have algorithms that produce the sequence of states in linear time.

2.2.3 Control

The control takes care of generating the messages that can be executed by the LLS. They are a tuple with two main elements: the force required and the time. They represent an impulse. To generate valid commands for the LLS, it needs the configuration parameters that define the physical model of the chaser. This means that the control knows the mass, the maximum force it can apply along a certain axis, the available force levels due to the discretization of the system.

Using the trajectory provided by the guidance, the control must generate messages allowing it to move from one point on the trajectory to the next. Since it has been assumed that guidance always generates trajectories in which acceleration along only one axis is required between each point and the next, the control could generate two commands at most between two points. If the system were ideal, these two commands would be acceleration and deceleration along the above-mentioned axis.

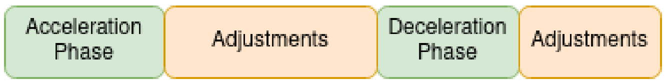

Figure 2.10: Breakdown of time allocated for each control activity. It is not to scale, and depends on the parameters decided by the project.

Due to the imperfections of a real system, this method cannot be used. Therefore, the ERMES control plans the above accelerations, with time constraints: between two different points of the trajectory it can use a percentage of the time for the two commands of acceleration and deceleration, and keep the rest for the commands of trajectory adjustment. The percentage of the time available for the acceleration and deceleration commands is a parameter set by the project, and can be tweaked to have the right compromise: the lower this parameter, the longer the time needed to travel the distance between one point on the trajectory and the next.

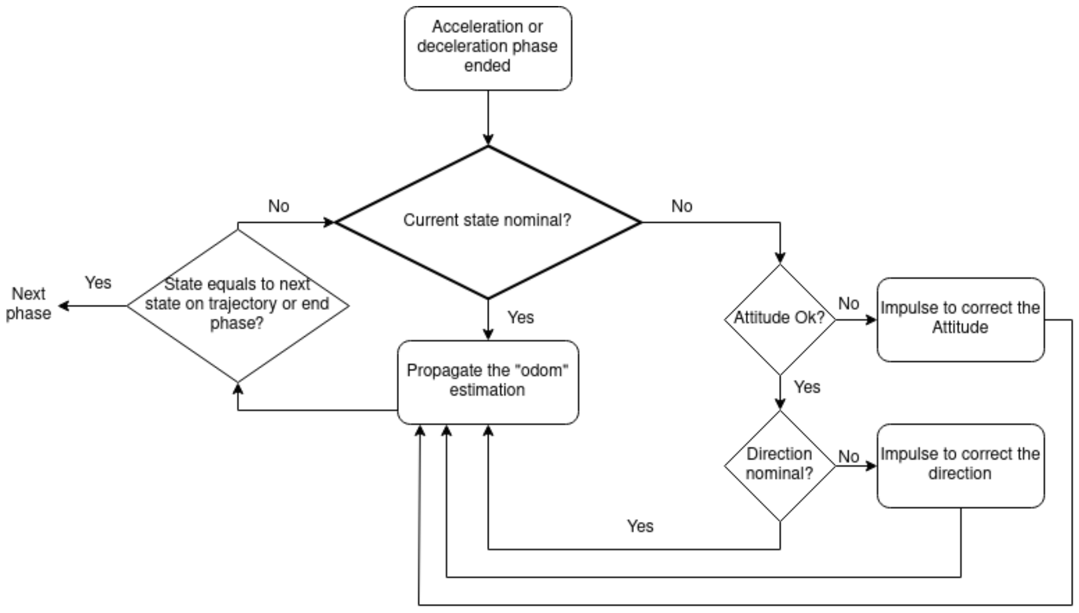
Acceleration or deceleration phase ended
Current state nominal?
No
No
Yes
Next phase
Yes
State equals to next state on trajectory or end phase?
Propagate the "odom" estimation
Attitude Ok?
No
Impulse to correct the Attitude
Yes
Direction nominal?
No
Impulse to correct the direction
Yes

Figure 2.11: A possible decision cycle for the trajectory adjustment phase.

During the adjustment time, the control will generate the necessary impulses to maintain a certain attitude and direction. The attitude and the translation movement are corrected with the information received from the location relative to “odom”. Using the information from the navigation relative to target, it could estimate the real target position in the odom coordinate system, and use this information to correct the manoeuvre.

Figure 2.12 shows an example of trajectory correction. The first figure shows the target and the planned trajectory on the odom reference system. In the second figure, a position update has arrived, and the position of the target relative to the odom reference system has changed by $\boldsymbol{e}$. The magnitude of the vector $\boldsymbol{e}$ is smaller than a parameter assigned to the control system: the control then updates the positions of the trajectory with respect to the odom reference system. This update will cause the generation of impulses by the adjustment cycle. In the third figure, the target has rotated on itself, so the location relative to odom shows a rotation. The control will ask the guidance if trajectory recalculation is necessary. In this example it is assumed that recalculation is not necessary, and therefore the control continues. In the last image the target is translated by $\boldsymbol{e}$, its magnitude is greater than the system parameter, so the control requires a new trajectory. This is the case if the localization system has previously estimated an incorrect position and the new estimates have corrected it.

Each command that the control generates affects the status of the chaser, and therefore has an expected result. The control uses the generated pulse messages to propagate the state received by the “odom”-localization: it adds the effects of the generated commands to the last received state. This was explained in

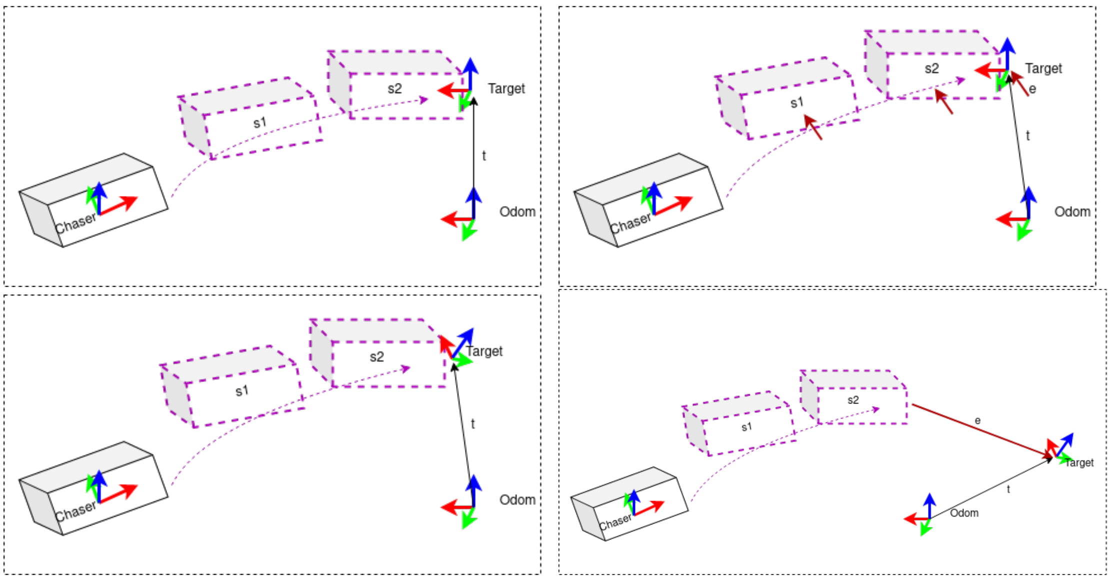

Figure 2.12: An example on control decisions

Section 2.2.1, and the Equation 2.2 shows the example in the case where the command is an acceleration along x.

The control operates at a fixed frequency of 10Hz, and, between one point on the trajectory and another, uses its control cycle to calculate the adjustment pulses and to propagate the state estimate according to the predictive model. Figure 2.10 shows the division of time to be allocated to the two activities: generation of acceleration and deceleration pulses, and generation of addition pulses. While in Figure 2.11 there is a general scheme of the adjustment phase.

2.3 High Level System

The HLS is used to interact with the experiment. The two main tasks of the HLS are to start (and stop) the experiment and to make the chaser telemetry available to the user.

To do so the high level system needs to communicate with the medium level system. The communication between the HLS and the glsmls is based on a wireless network.

To start the experiment and release the two mockups, an automatic release system based on electromagnets was designed, as described in Section 1.3. The exact moment of release is managed by the HLS through an automatic program based on the information of the IMU located in the release structure. The software plans to release the experiment when the jitter in acceleration is below a certain threshold. This software runs on the laptop connected to the IMU: the first one. In case it has any problems, the release software must be run on the second laptop, so the IMU must also be able to be connected to the latter..

Telemetry, on the other hand, can be viewed from both laptops of the HLS. This allows it to be recorded in double copy.

3
Software implementation

In this chapter, the software implementation chosen for ermes will be presented. Its structure is described in Chapter 2.

This chapter will discuss the main points of Robot Operating System (ROS) [25], which is the framework chosen for the development of ERMES. It will also highlight the key functionalities of the adopted libraries, focusing on those used in ERMES. Some calibration tests have not yet been performed, so the procedure for doing them and what to consider a good result will be described.

3.1 ROS overview

ROS is a set of software libraries and tools that help build robot applications. It contains drivers and state-of-the-art algorithms. It also provides powerful debugging tools. ROS is an open-source, meta-operating system for robots. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. The ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server [25]. ROS is not a realtime framework, though it is possible to integrate ROS with realtime code.

ERMES uses ROS to implement the MLS. This allows to integrate some algorithms, like the Tag detection, or the Extended Kalman Filter. This is a key value, because the algorithms are already tested in multiple applications.

The executables in ROS are called nodes. The code structure is composed by different nodes, each one has a "main". Each node will run in a different process[1].

[1]There are exceptions like nodelets

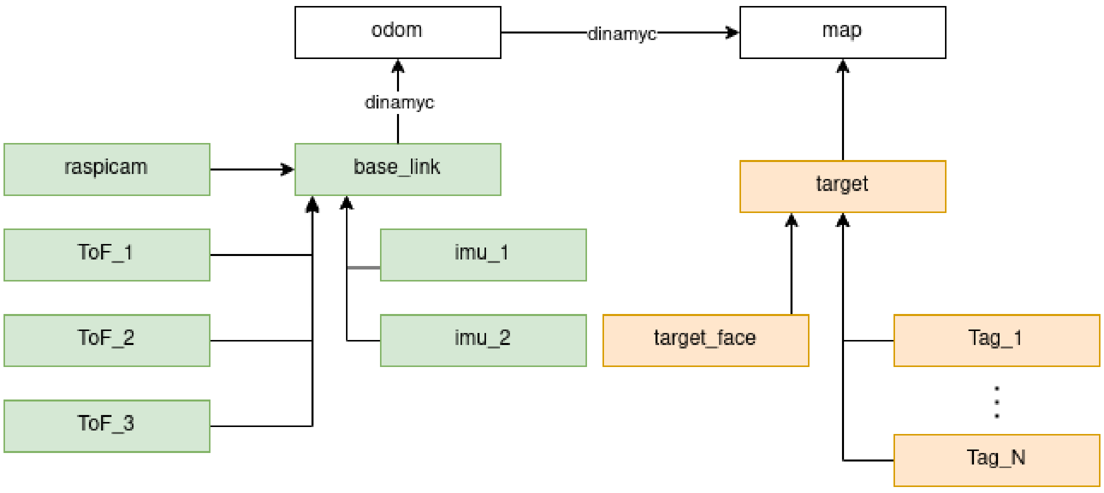

Figure 3.1: Relations between different coordinate systems. The arrow means "is child of".

To communicate between different nodes ROS uses topics.

A topic is composed by an header and the body of the message. In a ROS system the nodes can be distributed in different machines in the same network. ROS works with a principal node, called "roscore". It is the server that allows the subscription and publishing of the topics.

In ERMES the "roscore" is executed in the Raspberry, that runs Ubuntu 20.04 Server.

ROS uses workspaces to build and source the code. In ERMES chaser there is one workspace to build the additional libraries, and another to build the projects developed by the ERMES team.

In the laptop there will be a ROS installation that uses the chaser as a server. This allows the operator to check the telemetry with RViz[2], but also to save the data.

3.2 Reference System

In ROS each object has a coordinate system, for example as seen in Section 2.2, target and chaser have two different coordinate systems, like shown in the Figure 2.3. The coordinate system is called "frame". These frames are managed by the tf library [26]. The tf library was designed to provide a standard way to keep track of coordinate frames and transform data within an entire system such that individual component users can be confident that the data is in the coordinate frame that they want without requiring knowledge of all the coordinate frames in the system. Transforms and coordinate frames can be expressed as a graph with the transforms as edges and the coordinate frames as nodes. In this representation the net transform is simply the product of the edges connecting any

[2]RViz is a telemetry tool provided by ROS to show the output of the topics in a graphical environment.

two nodes. The graph can exist with one or more disconnected subgraphs and the transform can be computed between nodes within the subgraphs, but not between disconnected subgraphs.

In the ERMES system the tf tree is shown in the Figure 3.1. Note that the two edges between "map" and "odom" and between "odom" and "base_link" are dynamic. In fact, they are estimated by the localisation and therefore the tree is not initially fully connected. Each node represents a component of ERMES; the image shows the frames on the target in orange and those on the chaser in green. As explained in Section 2.2, the general reference system is based on the target coordinate system, so "map" is the root of the tree and it coincides with the target frame shown in the Figure 2.3. The main frame is choosen according to the application [27].

3.3 Low Level System

The low level system should integrate the Arduino program. The frequency for the programme cycle is 10Hz. the minimum time a group of EVs can remain open is 100ms. This also fixes the amount of possible commands manageble by the system: in 8 seconds of manoeuvre there is the time for 80 different commands. This is mainly due to the time it takes for the solenoid valves to open and close. The system has been designed with the aim of reducing the impact of the transition time, of opening and closing the EVs, on the total opening time.

As discussed in the Section 2.1 the features that the LLS should have are:

1. close and open the electrovalves;
2. read the first IMU;
3. read the second IMU;
4. read the three ToF;
5. empty the serial buffer;
6. send sensors data to the MLS;
7. deserialize the command;

To build an hard real time system we need to be sure that in the worst case of code branching, all the actions are performed before the 100ms time limit. This must also be tested by forcing the worst branch the code can follow, because when the system is completely integrated, it could be very difficult to identify.

To open the electrovalves there should be a table that associates each axis with the PINs where the EVs are connected in the circuit. The program should keep trace of the command in execution, to remember to close the EVs when it's terminated. A precaution to build more robust software is to close all EVs every time there isn't any command in the queue. So the close phase should close all the EVs.

The reading of the sensors is a point of concern, because it needs to wait the I2C, so this phase needs to be short enough even when all sensors will timeout.

During the development of the LLS, it is assumed that only one command can be executed at a time. In fact, the hardware design specifies that EVs should be opened in groups of 4.

3.3.1 Exploiting the Electrovalves

In the preliminary discussion of the LLS it has been decided that the electrovalves should function with a PWM. This could give some advantages and a finer control, useful in the final step of the maneuvre. Unfortunately the EVs don't work like the LED[3], and their average response is not linear with the input.

First lets briefly introduce what is a PWM. The Pulse-width modulation is a method to reduce the average power delivered by an electrical signal. Lets fix a frequency f, a periodic signal with a period of T seconds, and the possible values for the signal as $\{0, 1\}$. In a period T we defined the signal with the values 1 for $T * D$ seconds and 0 for $T * (1 - D)$ seconds, $0 \leq D \leq 1$. D is called duty cycle, and varying D results in a variation of the average power of the signal.

A PWM signal is to be used in the project to vary the output of the EV, mantaining the relation in the Eq.: 3.1:

$$F_{thrust} = F_{max-thrust} * D \tag{3.1}$$

Where $F_{max-thrust}$ is the maximum thrust of a nozzle with the electrovalve completely open, D is the duty cycle mentioned before. In the implementation the D needs to be discretised, so the PWM generator, that the Arduino provides in some ports, was used to obtain this result [28].

Different frequencies have been tested: 122.55Hz, 30.64Hz. According to the [28] document those frequencies can be obtained with the use of PWM registers, so it will not interfere with the LLS cycle. Higher frequencies than 122.55Hz do not work with the EVs chosen.

The D value is represented by the discrete values from 0 to 255. 255 is associated with 1, when the signal is always in its higher state.

The results are not linear and the Eq. 3.1 is not respected, so the output needs to be remapped to have a result similar to the Eq. 3.1. In the Chaser 16 different levels of thrust are sufficient to have a fairly fine division of the maximum thrust. The chosen frequency is 30.64Hz, because the response seems more stable. To remap the PWM values to obtain a response similar to function 3.1, it is necessary to find 16 values such that: 0 is 0, 15 is the maximum (in this case 135mN), and the remaining once are assigned so that the response is proportional. In general,

[3]In a PWM the LED will blink rapidly, and the perceived quantity of light is a factor of the maximum light that the diode can emit

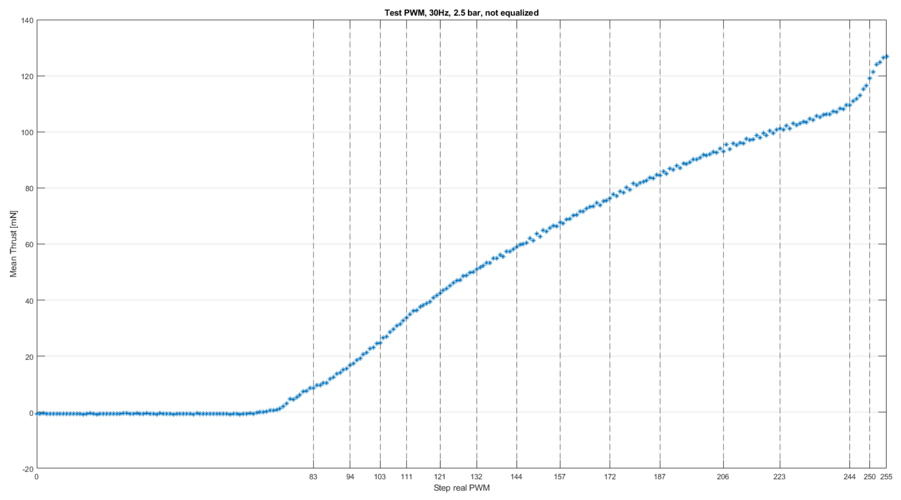

Figure 3.2: This graph reports the response in thrust of one nozzle if controlled with a PWM. The dotted lines represent the values of the PWM to map to the 16-levels values

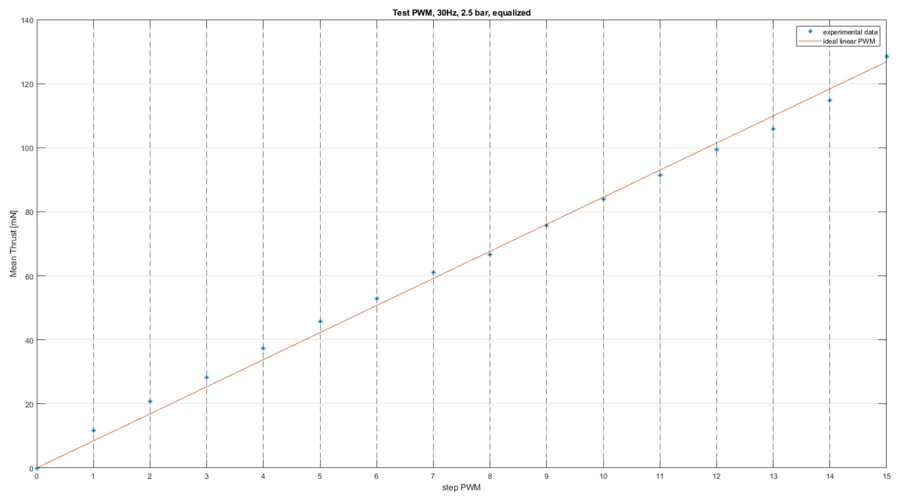

Figure 3.3: This graph shows the relation between the thrust of the nozzle and the PWM with the remapped values: step number 15 is $D = 1$

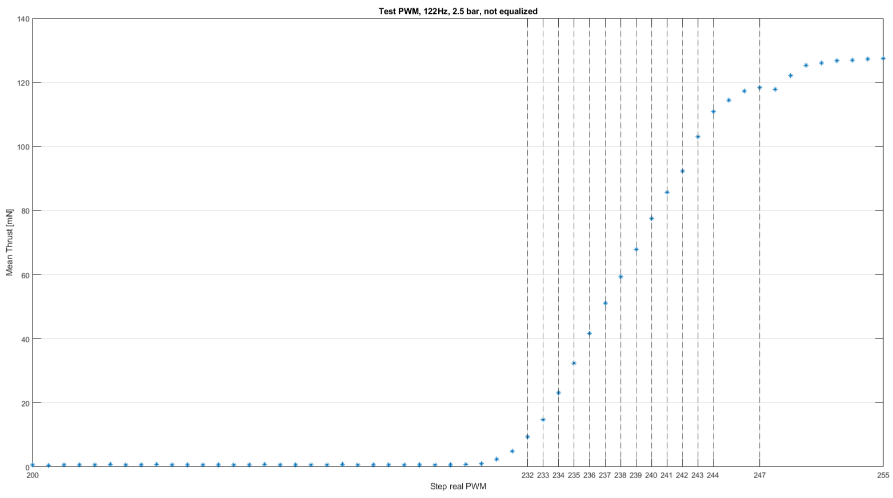

Figure 3.4: The response in thrust of one nozzle if controlled with a PWM. The dotted lines represent the values of the PWM to map to the 16-levels values. The points between 0 and 200 are omitted, because the mean thrust for them is 0

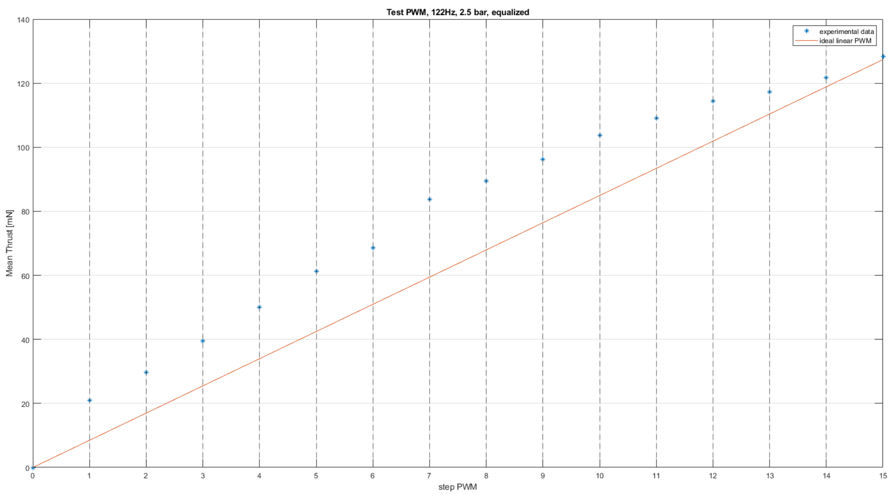

Figure 3.5: The relation between the thrust of the nozzle and the PWM with the remapped values: step number 15 is $D = 1$

given N bit of PWM precision, a PWM step is:

$$F_{step} = \frac{F_{max-thrust}}{2^N - 1} \tag{3.2}$$

And the general force of PWM value p, $0 \leq p \leq 2^N - 1$:

$$F_{PWM}(p) = F_{step} * p \tag{3.3}$$

Starting from a PWM accuracy in bits greater than N, called $\hat{N}$ in the non linear PWM response $\hat{f}$, for each p from 0 to $2^N - 1$ it is necessary to find the values of $\hat{p}$ such that:

$$\hat{f}(\hat{p}) = F_{step} * p \tag{3.4}$$

The pair $p, \hat{p}$ represents the map for a linear response with respect to p. In the implementation, an array of 16 elements contains the corresponding $\hat{p}$-values. The index of the array is p and it is the access key to the map. The final run of the experiment with 16 different levels outputs the graph shown in the Fig. 3.3. The values of $\hat{p}$ are represented by the dotted lines in the Figure 3.2.

3.3.2 Communication between LLS and MLS

The LLS needs to communicate with the MLS, so there are two ways: the first one is to use ROS as a message layer, this is memory intensive and can overhead the serial connection, but it's very easy to implement and integrate with the other nodes in the Raspberry. The second is to develop an ad hoc serial protocol for this system. The first approach is useful to quickly test some functions of the MLS, even before the LLS is developed. This method has a major flaw: it's slow. It can manage 2 topics, but when the system will be expanded it could miss the 100ms deadline. This is bad for the LLS, because, like it was explained in the Section 2.1.

As seen at the beginning of the section, the worst case scenario is when the loop needs to close the EVs, open another group of EV, read the sensors, send the data, read the buffer and deserialize the command. Please note that the worst case does not happen in every cycle. It is important to optimize the communication. The minimum amount of data required for communication is analysed below. For a command, must be considered:

- 3 bit for the DoF to activate, one of the possibilities is “no DoF”, all shut down.
- 5 bit for the PWM level: 4 as PWM, 1 as sign;
- cycles of duration, the maximum is 80, so 7 bit at worst;

A command could be less than 2 bytes. However, for data communication to the MLS, the following are required:

- 6 bytes for each accelerometer;

- 6 bytes for each gyroscope;
- 6 bytes for each magnetometer;
- 1 byte for each ToF;

Considering the above, an ad hoc protocol could greatly reduce the amount of data passing through the serial line. In fact, ROS messages have a large header and the data is not transported in the most efficient way (For example, the bytes needed for an IMU ROS message are 172). The optimal solution would be to send the minimum necessary data through a custom protocol, and then enrich it with the necessary information through a special node in the MLS.

3.3.3 Improving the LLS

During future tests of the system, it may turn out that the thrust given by the 4 EVs controlled by the LLS does not precisely follow the corresponding axis. There are two main ways to solve this problem: close the control loop to the LLS as well, or create a calibration matrix.

The first one needs to use the IMU data. Closing the cycle and balancing the output of the EV could be difficult, because there is the need to process the IMU data. Also the LLS should integrate the velocities around the axis, or combine the data from the two IMU to retrieve the torque. Adding to all of this, there are the problems related to the edge cases, like when the command lasts only one cycle or when the command requires the lowest thrust.

The problem can also be solved with a calibration system, as mentioned above. It has some advantages: the results are always static and it is faster for the processor.

To calibrate the system the deviation from the correct path should be measureed. This could be difficult for the DoFs of rotation, but doing this only for the translations might be useful anyway. At the end, using the data collected, a calibration table must be constructed, that means that for each EV used in each command, what is the thrust offset to apply. The results must be validated by the same test, the expected result is that the deviation is lower than before.

3.4 Navigation

The navigation of ERMES chaser relies on two main part:

- the computer vision system;
- the sensors data from the LLS.

Three main approaches were considered for the vision system: a LED based system, a features recognition and a fiducial tag localization. In the Section 2.2.1 it was mentioned that a system based on fiducial tags was chosen. Please

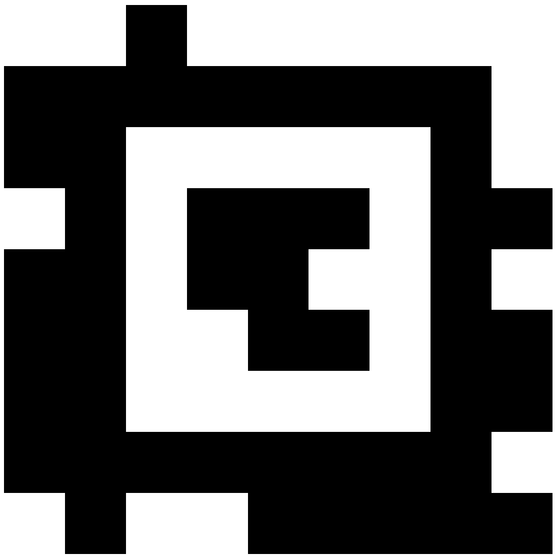

Figure 3.6: This is an example of AprilTag, it's code is 4, and it's format is "41h12".

note that all of these possibilities are meaningful, for example the Astrobee [10] experiment uses the features recognition on the handle of the docking port. The PACMAN experiment used the IR LED based system.

The sensors can be read by the LLS, but, if necessary, the entire I2C line can be moved on the Raspberry. In both cases the navigation layers consider only some types of messages, so the data needs to be processed before feeding them to the navigation.

As mentioned in the Section 3.2, the navigation will also publish the TF from "`map`" to "`odom`" and from "`odom`" to "`base_link`". This means that, during data preprocessing, nodes manipulating data in this subsystem cannot rely on the above-mentioned TFs, because the TF-tree has two missing branches.

3.4.1 AprilTag

AprilTag is a visual fiducial system, useful for a wide variety of tasks including augmented reality, robotics, and camera calibration. Targets can be created from an ordinary printer, and the AprilTag detection software computes the precise 3D position, orientation, and identity of the tags relative to the camera [17]. In ERMES the algorithm version 3 [16] was used, with its ROS wrapper. AprilTag uses an embedded 2D-coded marker for tag detection and to differentiate it from the other tags. The visual marker tag can be of any size with a square dimension. The tag is printed on a white background with a black outline square. Inside the square is an embedded black bar-code. In ERMES, the "41h12" format was used, which also features part of the barcode on the outside of the black border, an example is repoted in the Figure 3.6.

The AprilTag algorithm uses the features to recognize the tag in the frame, than finds the border and evaluates the position. Please keep in mind that the

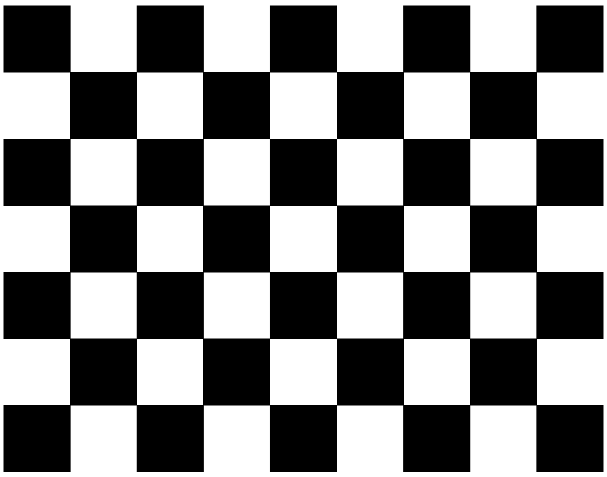

Figure 3.7: This is an 8×6 checkerboard used in calibration.

size of each tag and their code is known to the algorithm.

With AprilTag it is possible to detect a single Tag, or a bundle where each tag is in a known fixed position relative to each other. The use of bundles will result in a better localization of them, the drawback is that they take up a lot of space. They are resilient to partial occlusion, but in the case of ERMES this feature is not important. It is possible to change the format of the tags used, looking for one that is better optimised for the ermes use case. A good tag distribution solution is to position the first tag correspondent to the camera when the chaser is aligned with the target, so it will stay in frame even when the chaser approaches the target. Then apply other tags in the front face, and evaluate the possibility to put other tags in each face.

Camera

The raspberry camera must be used to detect the AprilTags. Drivers are needed to do this, and in ERMES it has been chosen to use the "raspicam_node" library [29]. It provides a ROS node that publish two main topics: "`image_raw`" and "`camera_info`". The first one publishes the uncompressed image according to the parameters, the second one publishes the camera information necessary to the rectification. The camera information must be obtained through calibration. After that it is possible to configure the said node to publish the correct information. To calibrate the camera the algorithm implementation of the Zhang approach [30] was used. ROS provides a node to execute the program, this node will subscribe to the "`image_raw`" topic, it will recognize the checkerboard and it saves the necessary frames. To make the calibration a chessboard 8×6 must be printed out, it is shown in the Figure 3.7. The algorithm implemented in ROS uses the OpenCV library, and in [31] there is a useful guide.

At the end of the calibration the node outputs two files, one of them is the

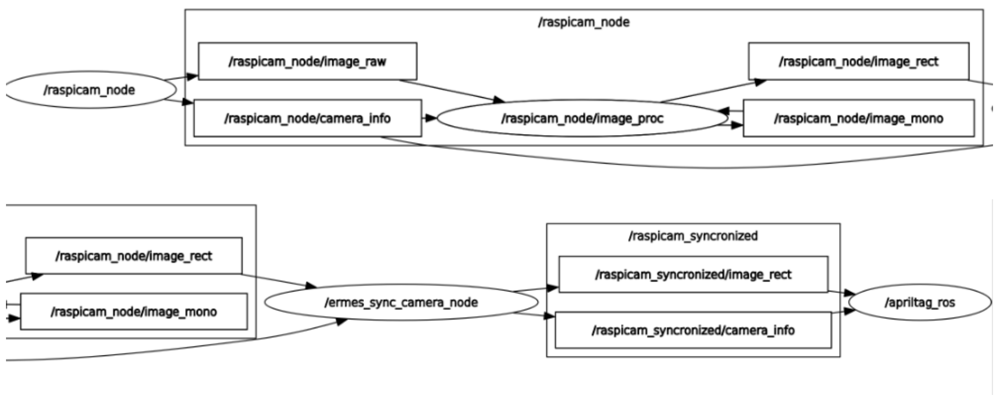
/raspicam_node
/raspicam_node
/raspicam_node/image_raw
/raspicam_node/camera_info
/raspicam_node/image_proc
/raspicam_node/image_rect
/raspicam_node/image_mono
/ermes_sync_camera_node
/raspicam_syncronized
/raspicam_syncronized/image_rect
/raspicam_syncronized/camera_info
/apriltag_ros

Figure 3.8: This is the output of the rqt graph of ROS, it represents the image pipeline.

yaml file needed for the rapicam node. The other contains the same information but in a different format. In ermes, a resolution of 1280×960 at $10fps$ were chosen as the camera's operating parameters. In addition, the focus was adjusted so that the depth of field is between 4cm and 20cm.

The AprilTag algorithm has additional operating parameters, but in this version the default ones have been maintained, with the exception of the number of processors that can be used, which has been increased to 4.

Image Pipeline

The camera will produce two topics, the image and the information, but the image is not rectified. To rectify the image ROS provides a nodelet inside the package "`image_proc`". This package uses OpenCV to rectify the input image using the camera info associated with the topic. It will publish the topic with the rectified image.

At this point in the pipeline, the image is rectified and can therefore be used by AprilTag's algorithm, but the camera information was published at the previous step, so it is out of sync with the rectified image, as can be seen in Figure 3.8. This problem exists because in the implementation of ROS it is foreseen that a camera can have variable focus, so each frame is associated with a different information message, with the calibration for that particular focal length relative to the frame. In the case of ERMES this leads to the need to add a synchronisation system between the two published topics. This is because the node processing the image to detect the apriltags requires the messages to be perfectly synchronised. Therefore, a special synchronisation node was created ("ermes_sync_camera_node" in the Figure 3.8).

The AprilTag node will subscribe to the output topics of the synchronization node. It will publish the tag detections topic, on which the node to transform the tag detections to odometry messages will subscribe to.

At this point, simple pre-tests were performed to confirm that the entire pipeline was executed within 100ms. Using the parameters on the AprilTag to

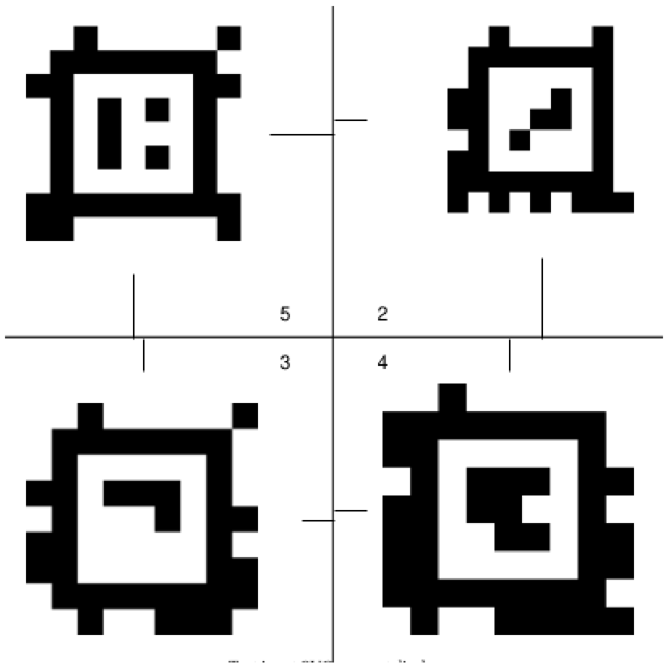

Figure 3.9: This is an example of how to place apriltags on the target, each of them is independent, but being in a fixed position relative to each other, it is possible to define a bundle. The numbers are the Tag ID used. The lines are guidelines for the necessary configuration.

achieve the best precision, the tag family "41h12", the maximum resolution of the camera frame, the frequency where the pipeline was stable was higher than 10Hz. Given the settings used, it might be worth thinking about lowering some settings that would lead to faster algorithm execution and thus a higher refresh rate.

3.4.2 Preparing Sensors Data

To use the sensor data in the localization node, they need to be prepared in a certain format. The navigation accepts odometry[4], IMU, pose and twist messages. Each one must be in the correct reference system according to the REP-103 [23].

Odometry from Fiducial Tags

An odometry message is composed by the pose (position and orientation) and the twist (linear and angular velocity). In the Section 3.4.1 it was described how to obtain the apriltags poses referenced to the camera frame. In order to use this information, their position must be converted to the target coordinate system ("`map`"). Figure 3.9 shows a test positioning of the AprilTags. As these 4 AprilTags are bound to each other, and the relative position between them is fixed, they can be grouped together in a bundle. A bundle Tag is defined by the

[4]odometry messages are composed by the pose and twist with covariance

main Tag, and the relative positions to it of the other AprilTags in the group. In the example, the Tag in the bottom right corner has been chosen as the main one, and the relative positions of the other Tags have been set out in the configuration file. The detection of the position of a Tag bundle is much more accurate because the algorithm can rely on 4 estimates. However, it must be completely in the camera's field of view to be detected.

To publish the positions the following strategy is used:

- If the tag bundle is detected, the pose published by the topic will be the one obtained by the bundle pose;
- If it's not detected, all of the tag topics will publish their pose letting the filter to manage them.

In this way there is always at least one pubblication of the pose from the AprilTag process.

Now that the pubblication strategy is defined, each tag detection must be converted to a chaser pose referenced to "`map`". With the tree of transforms, the algorithm can transform the pose from the "`raspicam`" frame to the "`base_link`", then it will invert it. Once this has been done, the position is referred to the reference system of the Tag, looking at the tree in 3.1, it is clear that it must be converted to a position referring to "`map`", and the static link exists and can be used to do this. After the geometry calculation the node will reconstruct the pose message, with the covariance matrix, and publish it.

From the node that publishes the poses, the average velocity between different time samples can be derived. It is assumed that:

- The velocity needs to be calculated from consecutive tag detections.
- The published twist topic needs to be referenced to the "`base_link`", not the "`map`", according to the localization implementation [32].

For simplicity lets assume that there is only one tag that is always detected. The twist node will subscribe to one of the chaser pose topics published by the previous node. At the first iteration it will save the old pose. When a pose estimation arrives, the node will calculate the velocity using the two poses, each pose has an associated time, because it's inherited from the topic header.

To do that the algorithm will invert both poses, so they represent a point position in space referenced to the chaser frame. In this way the node can ignore the target orientation in the calculations. The linear velocities on each axis are just the difference between the old pose and the new pose, divided by the difference in time, showed in the following:

$$v_x = \frac{x_{old} - x_{new}}{t_{new} - t_{old}} \tag{3.5}$$

$$v_y = \frac{y_{old} - y_{new}}{t_{new} - t_{old}} \tag{3.6}$$

$$v_z = \frac{z_{old} - z_{new}}{t_{new} - t_{old}} \tag{3.7}$$

To calculate the twist is easier to start from the target coordinate system, so the position of the chaser in the target frame. This allows to use some quaternion properties. In ROS the quaternion is defined as $q = (x, y, z, w)$ where the component are:

$$q = x + yi + zj + wk \tag{3.8}$$

There is a trick to find the relative rotation between 2 quaternion in the same frame, it is widely used in robotics. The quaternion q_r, to go from q_1 to q_2, is in relation with them according to:

$$q_2 = q_r * q_1 \tag{3.9}$$

from this lets multiply both sides with the inverse of $q_1^{-1} = q_{1inv}$ and obtain:

$$q_r = q_2 * q_{1inv} \tag{3.10}$$

To invert a quaternion, simply change the the sign of the w term. So the algorithm will calculate q_r.

Now the node needs to convert q_r in roll, pitch and yaw, then they can be divided by the delta time.

At the end both linear and angular velocity are calculated, and the node can publish the topic after the filling of the covariance matrix. The velocity covariance should be the sum of the two pose covariances.

IMU

The IMU data can be transported by the IMU message created by ROS. It is directly created in the LLS. As discussed in the Section 3.3, in the future the LLS will only send data through a custom protocol, and a node in the MLS will package it to conform to ROS standards.

ToF

To use the ToF data a fake pose message is created. If at least one of the ToF send a valid range[5], the node will publish a pose message with the x dimesion equals to the range augmented by the x value of the “`base_link`”-“`tof_N`” transform and the x value of the “`map`”-“`target_face`” transform[6]. The covariance needs to be a big value for all other variables but x.

On the other hand, if all three of the ToF send a valid range, it could be possible to calculate *yaw* and *pitch*. Before calculating them, the node should check the previous position estimation to verify that the beams do not intersect the drogue port of the target.

[5]The range needs to be between the minumum and the maximum

[6]This transform doesn’t exist at the moment, because it was needed only for the ToF, and only for the x value, so it is a simple parameter

At the end the node calculated a fake pose of the target with respect to the chaser. But the positions need to be referenced to the target frame. So, before publishing, it needs to invert it.

3.4.3 Extended Kalman Filter

In estimation theory, the EKF is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance. In the case of well defined transition models, the EKF has been considered [33] the de facto standard in the theory of nonlinear state estimation.

The `robot_localization` package [32] provides an implementation of the EKF filter for ROS.

ERMES uses two instances of this filter, one node to estimate the state relative to "`map`" and one relative to "`odom`".

Each node has a frequency. In the case of ERMES it was set at 10Hz if it is decided to mantain the sensor update cycle at the same frequency of the control cycle. Otherwise, if it is assessed that it is better to raise the frequency of publication of states, it can be set to 30Hz. There is no problem if the computer vision algorithm is slower than the chosen frequency, because the filter will propagate its state when it receives the first sensor data anyway.

As mentioned before, there are two estimators that use different frames, to configure that, all frames must be defined. "`map_frame`","`odom_frame`" and "`base_link_frame`" are set in the same way on both filters. Then, "`world_frame`" is set to "`map`" for the filter that references to the target, and to "`odom`" for the filter that references to the fictitious frame. In this setup the node that references the target will publish the transform from "`map`" to "`odom`", and the node that references the "`odom`" will publish the transform from "`odom`" to "`base_link`". For the state of the system, each node will publish the state vector referenced to the "`world_frame`" specified before.

For each data source there is a vector of 15 Boolean variables representing which reference system variables to consider for such data source when making the prediction. The order of the variables is the following:

$$\begin{bmatrix} x & y & z & roll & pitch & yaw \\ \dot{x} & \dot{y} & \dot{z} & \dot{roll} & \dot{pitch} & \dot{yaw} \\ \ddot{x} & \ddot{y} & \ddot{z} & & & \end{bmatrix} \tag{3.11}$$

The IMUs will have the acceleration along $\ddot{x}$, $\ddot{y}$ and $\ddot{z}$, the twist along $\dot{roll}$, $\dot{pitch}$, $\dot{yaw}$ set to true.

The ToF pose source will have the x position set to true, and also $pitch$ and yaw orientation. Note that if the position is invalid the node will not publish the pose data on the topic, this is not a problem, the filter will publish the propagation state anyway.

The odometry from fiducial tags is composed by two topics, like explained in the Section 3.4.1. This means that for the position the filter will consider all 6 position variables: x, y, z, $roll$, $pitch$, yaw. And the respective $\dot{x}$, $\dot{y}$, $\dot{z}$, $\dot{roll}$, $\dot{pitch}$, $\dot{yaw}$ velocity variables for the twist.

In the Section 2.2.3 it was shown how the control system will produce commands expressed by accelerations. To integrate those commands in the filter, it needs to produce odometry messages, like it was mentioned inthe Section 2.2.3. The covariance of this predictive model is correlated by the performances of the LLS and it can be retrieved in the experiment to calibrate the LLS (Section 3.3.3).

The last mention is to the "relative" parameter. For the estimator that references to odom, this parameter will be true for the fictitious odometry messages from the control node. With this parameter, the filter will initialize the first state estimation as the origin of the system. This guarantees a continuous state estimation with an initialization parameter.

The result of this implementation is composed by two nodes that operate in the Raspberry. They will publish an odometry message, and the acceleration message. With both topics there is the 15 dimensions state presented in the Section 2.2.1. It will also the pubblish the transform from different frames. This is needed for the correct visualization of the state of the system in RViz, but also it is needed by the control cycle, because, like it was mentioned in the Section 2.2.3, it will convert the states of the trajectory from one reference system to another to determine whether trajectory recalculation is necessary.

Finally, what happens in flight will be something like this:

- The chaser program will start, it has not yet been released;
- The apriltags will be detected because the two mockups are still in fixed positions, so they are definitely in the camera's field of view;
- The navigation node publish the first message;
- The control publish the first state, with position relative to target, and velocity equals to zero.
- The navigation publish the message, this time the "odom" state estimation is valid because it receives the first data from the "relative" source;
- The experiment start the release of the system;
- The chaser and target are released;
- If the target is out of the field of view of the camera, it's position is already estimated, and the chaser just try to counteract the unexpected rotation;
- The target will return on the field of view, the guidance would probably recalculate the trajectory;
- The target is now in ToF range, so the estimation of the distance is more precise.

This system should be able to counteract the initial rotation given by the release system. Even if the tags will be out of frames, the propagation of the state will publish the orientation expected, so the control can send the counter rotation command to the LLS.

The two nodes to publish two different state estimations are usually used for other purposes. For example in a robot that uses the Global Positioning System (GPS), due to the nature of GPS, it will use two localizations: one local and one global. In the ERMES experiment the tag detection is like the GPS.

In the Chapter 4 there is reported an integration test of the system.

3.4.4 Calibrate the sensors

The system described in the Section 3.4.3 relies on the covariance of the sensors. In a simplistic way, if the covariance is high enough, the data point will be ignored. If the covariance is too low, the filter will jump between 2 different estimantion every time it receives a data point. So it's important to estimate the covariance of the data of each sensor.

There are two ways: the first one is starting from the manifacturer datasheets, and increase it according to the performances of the filter. This works but if possible should be avoided. The second way is testing each sensor with a ground truth, in the final hardware setup, retrieve a dataset and calculate the covariance.

The use of a motion capture system is proposed as ground truth. In an optical motion capture system, multiple synchronized cameras are installed around the target capture volume, and 2D images are captured from each camera. 2D positions are calculated, and the overlapping position data are compared to compute the 3D positions via triangulation. OptiTrack[7], which is a ROS-compatible[8] system, may be available in the laboratory.

AprilTag Detection Covariance

The proposed setup is similar to the work done by [34]. They used a Motion Capture device as groud truth. In ERMES the OptiTrack should be used as ground truth, as mentioned before. A ROS node should publish the position of the chaser and target on a topic in the OptiTrack frame reference. Then the OptiTrack measures should be transformed from its reference to the reference used by the chaser: "`map`". Enough data must be collected to have a sufficiently accurate estimate for the variance of each variable. Please note that the covariance of each AprilTag could be different, because it depends on the dimension. Than if the tag bundle is used in the experiment, its covariance should also be estimated.

The twist covariance is the pose covariance doubled.

IMU Covariance and Calibration

The IMU's covariance estimation is more difficult. The proposed covariance is the square of the offset recorded by the IMU calibration. ERMES will use two IMU, so after calibration it is necessary to check that the covariance estimates

[7]`https://optitrack.com/`

[8]`http://wiki.ros.org/mocap_optitrack`

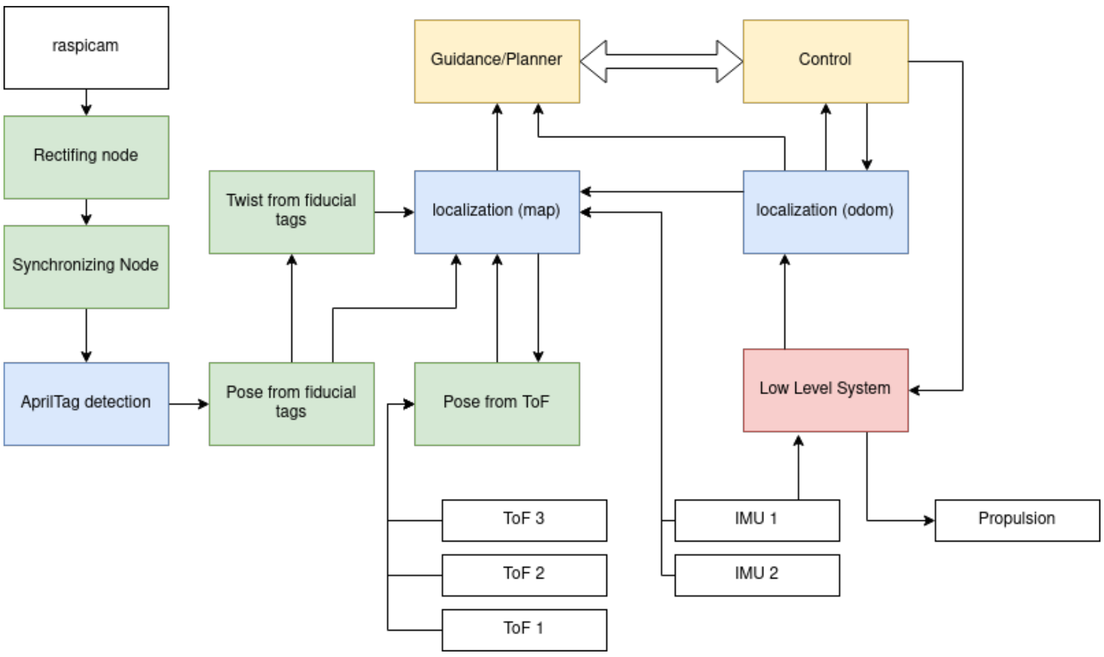

Figure 3.10: Diagram of interactions between the various components of the ERMES GNC. Arrows represent data flows.

overlap sufficiently. Otherwise, the estimate produced by the EKF might jump from one data point to another every time it receives an update from one of the two IMU.

ToF Covariance

The ToF covariance could be collected in the same experiment of the AprilTag covariance. There is also the possibility of relying on the data provided by the manufacturer, but this was collected in an environment too different from ERMES' use case. In fact ERMES should collect these data using the face of the Target.

3.5 Guidance and Control nodes

The guidance and control are two different nodes. They both receive the messages from the localization.

The guidance publishes a trajectory in the form of a vector of states, each state has 7 variables and they are all assumed with acceleration equals to 0, they have the form expressed in Equation 2.4 of Section 2.2.2. This trajectory is calculated accordingly to the strategy choosen for this phase. The implementation assumes one pubblication before the release of the chaser. The guidance listens to the topic published by the localization relative to the target. The control could request a recalculation in some circumstances. The guidance needs to account the physics limits of the physics system, for example the different acceleration limits in the various axes.

The control node tries to generate acceleration commands, then check if the nominal trajectory given by the execution of the command will result in the next state. It can generate commands every 100ms according to the LLS implementation. The control node then publish a odometry message every 100ms, using as the first reference the first target localization estimation. Then it could integrate the acceleration commands and retrieve the expected velocity and pose at each time step. Obviously the localization generated from this data is imprecise in the long term, but its precision in the short term depends on the calibration of the LLS. The estimation is also continuous, the control should use this reference system for the execution of the trajectory, then the guidance will produce another trajectory using the target reference system if the target moves. Theoretically the control node shouldn't use the localization relative to the target, but only the position referenced to "`odom`".

From the outside view, it can be summarized as follow: the guidance will produce a trajectory in the frame of the "`odom`" using the data in the frame of the "`map`". The control will try to follow this trajectory in the frame of the "`odom`", keeping the trajectory updated according to the strategy described in the Section 2.2.3. Please note that in the meantime "`odom`" moves from its original position, so the transform is changed. This approach works not only for the accumulated error by the "`odom`" localization, but also for the movement of the target.

3.6 Retrieve Data and HLS

To save all the data ROS provides the command `rosbag`, with this command executed on one laptop, ROS can create a file that saves the output of one or more topics. This is better than saving the data in another database system, because it allows to reproduce the same data in the simulator.

To control the experiment the HLS needs one node that listens to the IMU data from the sensor installed in the structure. With this data it has to decide when is the best time to send the start signal to the main node of the chaser. Note that the launch of the system needs to be done before the start of the parabola. This allows the MLS to record the initial possible angular velocity, so it can act accordingly when it is released.

If the rosbag saves also the topics published by the HLS, all the data are timing correlated. Please note that the image topics shouldn't be saved by the laptop. A better procedure is to run another ssh terminal from the laptop that saves the image topics in the chaser itself. The transport of image topics through the network is discouraged.

4 Integration Test

A basic test was designed to check the operation of the navigation system. It has the following objectives:

- check that missing branches of the TF tree are published;
- ensure that localization topics are published;
- check accuracy of localization in normal condition;
- check accuracy when the target is out of focus;

To check the first two requirements, just start the system and see if the missing branches are published, and if the topics are present.

In order to check the others, 4 tags were printed on the sheet at known sizes and relative distances from each other. This sheet of paper was attached to a metal plate, so that it would stand straight, and fixed to a bench vice. This will simulate the target, in fact the AprilTag have been placed within a rectangle the size of its face. To simulate the Chaser, the Raspicam was attached to a pre-built model, which can also accompany the Raspberry. An Arduino and an IMU were

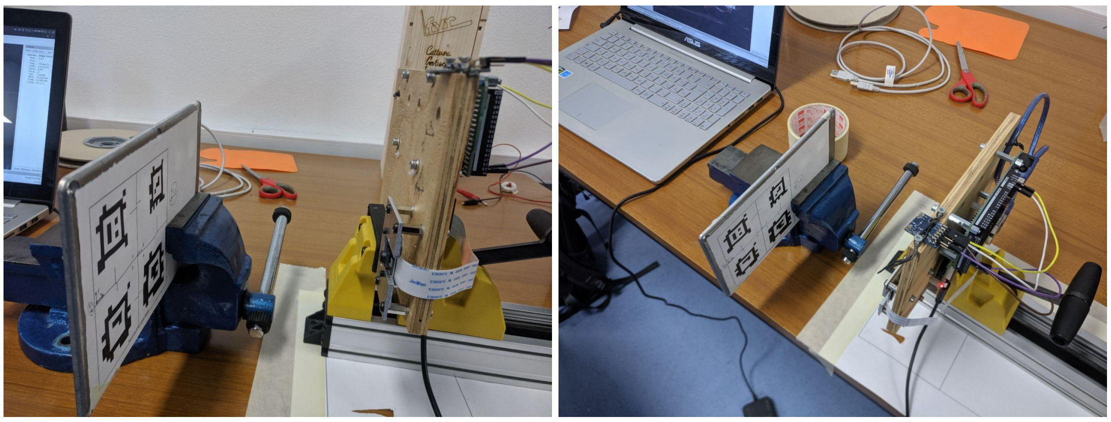
Figure 4.1: Setup for the Integration Test

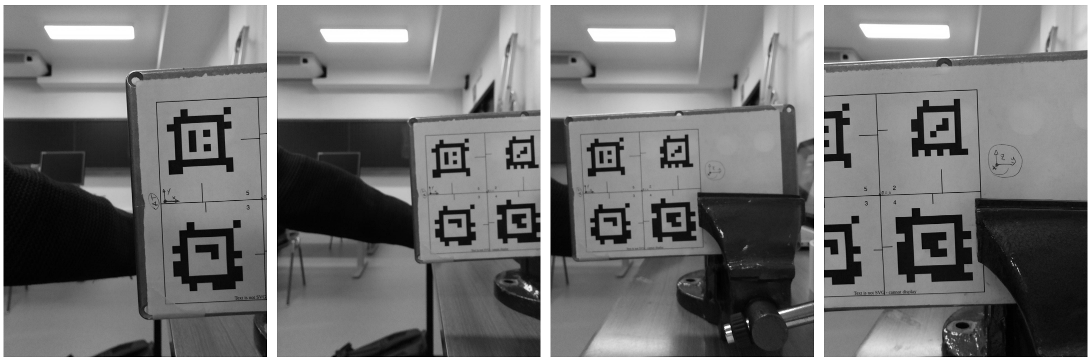

Figure 4.2: Field of view in each position, in the order: A, B, C, D

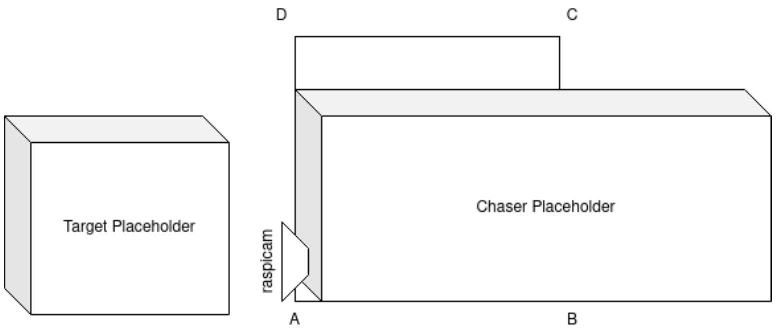

Figure 4.3: Setup for the Integration Test (Not in scale)

added to it. The ground truth of the experiment is a sheet of paper with a square of side 10cm, fixed to the table, on which the chaser can be moved. The laptop is connected to the Raspberry via WiFi, to check the system with RViz. The setup is shown in the Figure 4.1

4.1 Execution and Results

Before the test, the camera was adjusted to have a focus range of 5cm to 20cm, and then the calibration was repeated. It was verified that the calibration was correct by checking the rectified images with rqt_image_view. The chaser was placed at the four corners of the test square to verify the frame captured by the camera. The photos are in the Figure 4.2. They are in the same order as the chaser route for the experiment, that is reported by the scheme in the Figure 4.3.

The second objective of the experiment is easy and it is a requirement for the other steps, it was checked that the list of published topics included those related to navigation, from which the message data will be extracted. The first objective is shown in the Figure 4.4.

A telemetry node was created, that listens to the odometry messages, and write the x and y coordinates on a log file. Several repetitions of the experiment were carried out to record the data. One set of repetitions was made with the

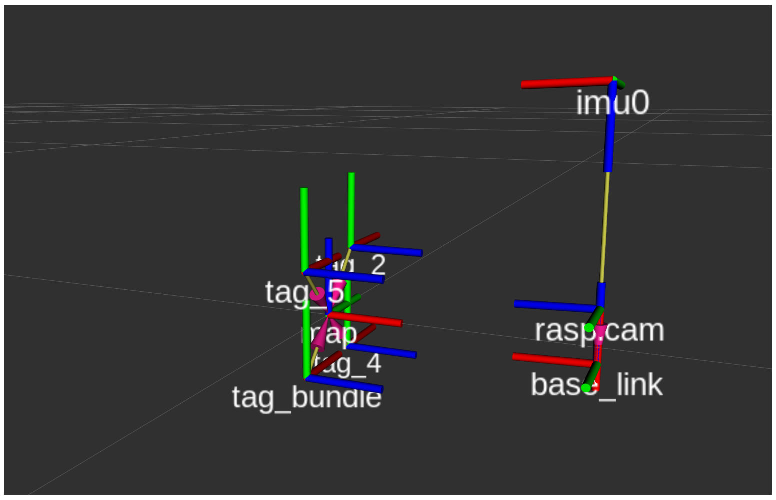

Figure 4.4: The RViz view of the tree of the integration test, the “map” frame is used as coordinate system for the 3D viewer.

initial position of the target at approximately 13cm, so that during the experiment the distant part was out of focus. The other set of repetitions was made starting at about 9cm, so that the whole experiment was in optimal conditions.

In the Figure 4.5 there is the path of one repetition. As expected, the accuracy away from the target tends to degrade, which can be seen despite the flaws in the execution of the experiment. Then some parameters are calculated:

- average minimum distance from the nearest point of the rectangle;
- variance of the minimum distance from the nearest point of the rectangle;
- maximum minimum distance from the nearest point of the rectangle;

Those parameters are calculated for the entire rectangle, and for the lower edge. That is because the lower edge, from A to B, is the most interesting in ERMES. The results are reported in the Table 4.1.

4.2 Comments

The results are as expected but the distance from square is a somewhat misleading measure. In fact, the test should be improved with a ground truth referred to the individual point. Apart from that, the most interesting edge for ERMES project is also the one with the best performance, at least from what the test shows.

There is room for improvement when the chaser is far from the target, but the test can be considered a very good first result: it shows the correct integration of the system, and provides the track followed to see where the algorithm can be improved. This test also has advantages in terms of time, as it takes only a short time to set up, and leaves room for more measurements, even repeated

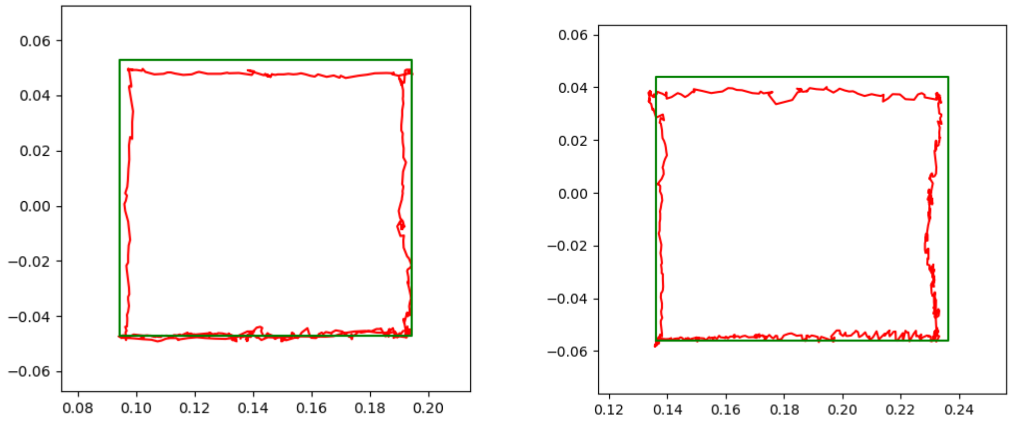

Figure 4.5: The path recorded by the Chaser, on the left the camera was always in focus, on the right it was not. These are 2 example of the repetitions. The green square is a 0.1×0.1m, starting from the first record of the repetition. For simplicity: looking at this graph is like looking at the experiment from above.

	In focus	Out of focus
avg (mm)	1.560	2.688
var (mm^2)	2.628	4.454
max (mm)	6.4	10.2
avg AB (mm)	0.692	1.628
var AB (mm^2)	0.432	1.069
max AB (mm)	3.4	3.9

Table 4.1: Results of the integration tests, on the right column the results of the test with part of the square outside the focus area

ones. It could also be a good preliminary test for the future, and would serve to identify implementation errors before more demanding and time-consuming tests are carried out.

5 Conclusion

This thesis described the operation of the ERMES experiment. It then showed the hardware choices to enable it to run: describing the two free floating objects. However, the main part is the development of software that can enable the manoeuvre that the experiment aims to achieve. In order to understand the main functionalities of such software, its architecture was discussed in Chapter 2. A solution was found to the problem of navigation for the chaser, and a control system was described that could exploit this functionality. During development, a way was also found to use the EVs with a PWM signal, while maintaining a linear response in terms of the resulting force from the system. This method was tested and its results are satisfactory and presented in Chapter 3. In the same Chapter the implementation techniques adopted to obtain this software system have been described, including navigation based on two EKF filters, which is the main part of this thesis. In order to demonstrate the correct integration of the navigation system, a basic model was assembled with the necessary components to operate the system. The description of the test of this model and the results obtained are given in the last chapter.

Certainly the EV management and the implementation of the navigation system are very good results, however the navigation integration test has some flaws, as described in the corresponding chapter. Indeed, in future developments would be appropriate to repeat the described test in a better controlled environment. In addition, all sensors would have to be calibrated in order to obtain more meaningful results from the EKF filters used.

This thesis opens up the opportunity to develop a control system based on two Localizations, which makes it possible to distinguish the movements of the chaser from the apparent movements given by the adjustment of the target attitude. It is hoped that future developments will make good use of this functionality. After the flight campaign, it would be interesting to considering the use of ROS 2: the real time version of ROS. In recent years ROS 2 has made interesting developments, finally becoming a system stable enough to be used in projects. A good development of the project could be base the successor of ERMES using ROS 2 with a real time kernel.

Bibliography

[1] ESA, “Fly Your Thesis! programme phases.” https://www.esa.int/Education/Fly_Your_Thesis/Fly_Your_Thesis!_programme_phases, 2022. [Online; accessed 21-Mar-2022].

[2] Novespace, “The Airbus A310 Zero-G.” https://www.airzerog.com/the-airbus-a310-zero-g/, 2022. [Online; accessed 21-Mar-2022].

[3] F. Branz, L. Olivieri, F. Sansone, and A. Francesconi, “Miniature docking mechanism for cubesats,” *Acta Astronautica*, vol. 176, pp. 510–519, 2020.

[4] P. Amadieu and J. Heloret, “The automated transfer vehicle,” *Air & Space Europe*, vol. 1, no. 1, pp. 76–80, 1999.

[5] G. Ritter, A. Hays, G. Wassick, G. Sypitkowski, C. Nardell, P. Tchory, and J. Pavlich, “Autonomous satellite docking system,” in *AIAA Space 2001 Conference and Exposition*, p. 4527, 2001.

[6] NASA, “Crew-2 Astronauts Safely Splash Down in Gulf of Mexico.” https://blogs.nasa.gov/crew-2/, 2022. [Online; accessed 21-Mar-2022].

[7] A. Fejzic, S. Nolet, L. Breger, J. How, and D. Miller, “Results of spheres microgravity autonomous docking experiments in the presence of anomalies,” in *59th International Astronautical Congress, Glasgow, Scotland*, 2008.

[8] J. Bowen, A. Tsuda, J. Abel, and M. Villa, “Cubesat proximity operations demonstration (cpod) mission update,” in *2015 IEEE Aerospace Conference*, pp. 1–8, IEEE, 2015.

[9] M. Bualat, J. Barlow, T. Fong, C. Provencher, and T. Smith, “Astrobee: Developing a free-flying robot for the international space station,” in *AIAA SPACE 2015 Conference and Exposition*, p. 4643, 2015.

[10] M. G. Bualat, T. Smith, E. E. Smith, T. Fong, and D. Wheeler, “Astrobee: A new tool for iss operations,” in *2018 SpaceOps Conference*, p. 2517, 2018.

[11] D. Petrillo, M. Buonomo, A. Cavinato, F. Chiariotti, M. Gaino, F. Branz, R. Mantellato, L. Olivieri, F. Sansone, A. Francesconi, *et al.*, “Flexible electromagnetic leash docking system (felds) experiment from design to microgravity testing,” in *66Th International Astronautical Congress, IAC-15 E*, vol. 2, 2015.

[12] M. Barbetta, A. Boesso, F. Branz, A. Carron, L. Olivieri, J. Prendin, G. Rodeghiero, F. Sansone, L. Savioli, F. Spinello, *et al.*, “Autonomous rendezvous, control and docking experiment-reflight 2,” ESA/CNES Small Satellites Systems and Services Symposium, Porto Petro . . . , 2014.

[13] M. Duzzi, R. Casagrande, M. Mazzucato, F. Trevisi, F. Vitellino, M. Vitturi, A. Cenedese, and A. Francesconi, “Electromagnetic position and attitude control for pacman experiment,” in *Proceedings of the 10th International ESA Conference on Guidance, Navigation & Control Systems, Salzburg, Austria*, vol. 29, 2017.

[14] A. Mehrparvar, D. Pignatelli, J. Carnahan, R. Munakat, W. Lan, A. Toorian, A. Hutputanasin, and S. Lee, “Cubesat design specification rev. 13,” *The CubeSat Program, Cal Poly San Luis Obispo, US*, vol. 1, no. 2, 2014.

[15] B. Verthier, A. Jaquemet, T. Villatte, A. Jolion, and F. Gai, “NOVESPACE Experiment Design Guidelines in Parabolic Flight,” Guidelines GDL-2021-03, NOVESPACE, 2021.

[16] M. Krogius, A. Haggenmiller, and E. Olson, “Flexible layouts for fiducial tags,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2019.

[17] E. Olson, “AprilTag: A robust and flexible visual fiducial system,” in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, IEEE, May 2011.

[18] J. Wang and E. Olson, “AprilTag 2: Efficient and robust fiducial detection,” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[19] STMicroelectronics, “Proximity and ambient light sensing (ALS) module,” Datasheet DocID026171 Rev 7, STMicroelectronics, 2016.

[20] Atmel, “SAM3X / SAM3A Series,” Datasheet Atmel-11057C-ATARM-SAM3X-SAM3A-Datasheet 23-Mar-15, Atmel, 2015.

[21] K. Albee, M. Ekal, C. Oestreich, and P. Roque, “A Brief Guide to Astrobee’s Flight Software,” Guidelines Revision 1.1, 2021.

[22] H. Koptez, “Real-time systems: design principles for distributed embedded applications,” 1997.

[23] T. Foote and M. Purvis, “Standard Units of Measure and Coordinate Conventions.” `https://ros.org/reps/rep-0103.html`, 2010. [Online; accessed 21-Mar-2022].

[24] R. E. Kalman, “A new approach to linear filtering and prediction problems,” *Transactions of the ASME-Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[25] ROS, “Robot Operating System.” https://www.ros.org/, 2022. [Online; accessed 21-Mar-2022].

[26] T. Foote, “tf: The transform library,” in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, Open-Source Software workshop, pp. 1–6, April 2013.

[27] W. Meeussen, “Coordinate Frames for Mobile Platforms.” https://ros.org/reps/rep-0105.html, 2010. [Online; accessed 21-Mar-2022].

[28] Arduino, “Secrets of Arduino PWM.” https://docs.arduino.cc/tutorials/generic/secrets-of-arduino-pwm, 2022. [Online; accessed 21-Mar-2022].

[29] UbiquityRobotics, “raspicam node repository.” https://github.com/UbiquityRobotics/raspicam_node, 2022. [Online; accessed 21-Mar-2022].

[30] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[31] OpenCV, “Camera Calibration and 3D Reconstruction.” https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, 2022. [Online; accessed 21-Mar-2022].

[32] T. Moore and D. Stouch, “A generalized extended kalman filter implementation for the robot operating system,” in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.

[33] S. J. Julier and J. K. Uhlmann, “Unscented filtering and nonlinear estimation,” *Proceedings of the IEEE*, vol. 92, no. 3, pp. 401–422, 2004.

[34] S. M. Abbas, S. Aslam, K. Berns, and A. Muhammad, “Analysis and improvements in apriltag based state estimation,” *Sensors*, vol. 19, no. 24, p. 5480, 2019.