Guidance, Navigation and Control of a Sounding Rocket

Albéric de Lajarte

Master thesis of Robotics
Section de Microtechnique

Swiss Federal Institute of Technology in Lausanne

Supervisor Dr. Colin Jones, Associate Professor, Automatic Control Laboratory
Assistant Supervisor Petr Listov, Doctoral Assistant, Automatic Control Laboratory

June, 2021

Acknowledgements

First, I would like to thanks my parents and my family for their unconditional support throughout these years at EPFL.
Then my gratitude goes to the EPFL Rocket Team, who gave me the opportunity of a great adventure with wonderful friends and challenges which taught me so much.
Finally, I would like to thank the Automatic Control Laboratory, and especially Professor Colin Jones for trusting my project and to Petr Listov who has been a constant support during the months of hard work. Your precious help and feedback brought this project much further that I could have hoped.

Abstract

A new framework for development and testing of Guidance, Navigation and Control (GNC) algorithms is proposed to support the efforts of the student association EPFL Rocket Team in developing a new GNC system for sounding rockets.

This framework includes a complete hardware and software architecture based on the Robot Operating System (ROS), a highly modular middleware that allows the deployment of GNC in most Linux compatible machines, such as the popular Raspberry Pi for embedded applications, or personal computer for development and simulations.
This architecture is completed with a new real-time rocket simulator developed as a standalone ROS package to support testing of the GNC in Software-in-the-Loop, Processor-in-the-Loop or Hardware-in-the-Loop environment with very little modifications of the GNC.

Two applications of this framework are presented in this thesis. The first one is a GNC developed to control the maximum altitude of the sounding rocket Bellalui-2, built this year by the EPFL Rocket Team and scheduled to be launched at 1200 meters in July 2021.
The second one is a more experimental GNC to control a complete Thrust Vector Control system that could be integrated with the next generation of rockets developed by the EPFL Rocket Team.

List of Tables

List of Figures

Table of Contents

Introduction

Guidance, Navigation and Control (GNC), is a spacecraft subsystem dealing with trajectory planning and motion control. Specifically, it refers to three components of this subsystem: Navigation estimates the position, velocity and attitude of the spacecraft; Guidance defines a trajectory from the current state to the target point, and the Control algorithm controls the spacecraft actuators to follow the trajectory given the current estimated state. This subsystem is thus a critical part of any spacecraft aiming to perform complex autonomous missions.

The EPFL Rocket Team (ERT) is a student association aiming to develop sounding rockets from the ground-up. Its goal is to give students first hands-on experience of aerospace engineering projects through international competitions, researches, and educational projects. To investigate advanced control and navigation techniques, a new project called Icarus was started in September 2020 inside the EPFL Rocket Team. This projects does not aim at developing a particular product, but rather explore various technologies related to GNC, such as active rocket stabilization, Thrust Vector Control (TVC), controlled apogee and landing.
After a semester of preparation in autumn 2020, a small team of 12 students was form to work on a Thrust Vector Control prototype, and the associated test facility for ground qualification.
Supervision of the team has been a part of this master thesis, as well as creating a framework to develop a reliable GNC for the new rocket by designing a new GNC architecture, a real time simulator for software and hardware testing, and applying these tools and methods to first GNC prototypes. This report details the results of this work, which should serve as a basis for future control project supported by Icarus in the future.

Rocket system

This section provides the reader some basic knowledge of a rocket dynamic. The rocket is assumed a rigid body, rotating around its center of mass, and subject to the four main forces:

- **Thrust**: is the force created by the rocket engine by expelling propellant at high speed through the nozzle. This force is proportional to the mass flow rate and propellant speed.
- **Weight**: is the force due to Earth gravity and the rocket mass. Weight of the rocket changes throughout the flight due to the propellant burn.
- **Drag**: is the aerodynamic force due to air resistance, slowing down the rocket at high speed. It is colinear with the relative motion of the fluid.
- **Lift**: is the aerodynamic force due to the fluid being redirected by the rocket surface at high speed. It is proportional to the rocket angle of attack α, i.e. the angle between the rocket $\hat{\mathbf{z}}^b$ axis and the relative wind, which is the apparent velocity of the rocket in the reference frame of the wind.

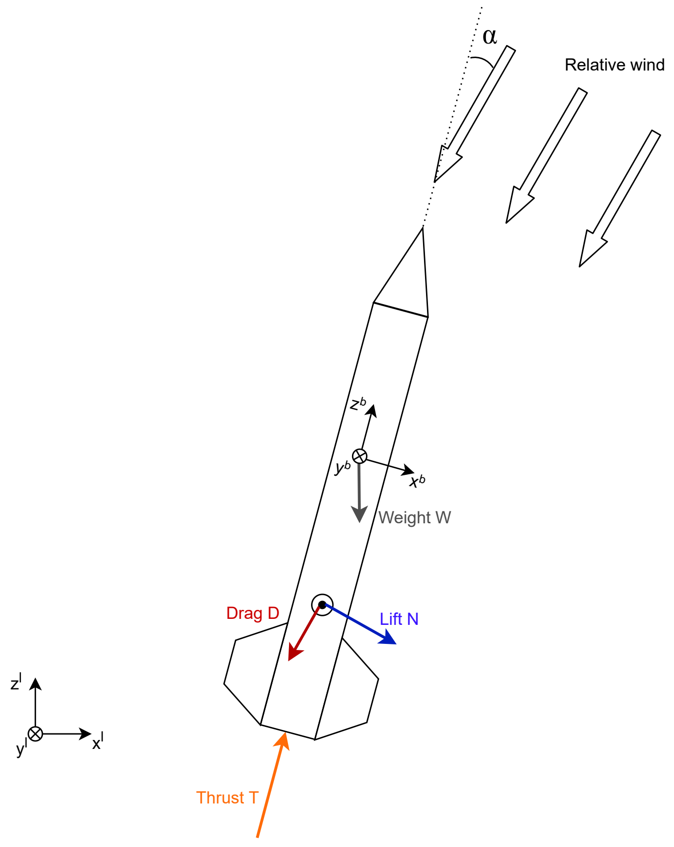


Figure 1: The four principal forces of a rocket, and the effect of relative wind

In this thesis, two reference frames are considered: the local-level frame is attached to the launch-pad, with its $\hat{\mathbf{z}}^l$ axis pointing up and $\hat{\mathbf{x}}^l$ axis pointing to the North, and the body frame is attached to the rocket's center of mass, with the $\hat{\mathbf{z}}^b$ axis (roll) in the direction of the rocket length, toward the nosecone, and the $\hat{\mathbf{x}}^b$ and $\hat{\mathbf{y}}^b$ axes aligned with those of the on-board sensors, perpendicular to the rocket surface.

A common misconception is to think about rocket as a pendulum, with its center of rotation at the top, which could thus be stabilized by having its center of mass close to the bottom. In fact, most small sounding rockets rather rely on passive stabilization from the lift force. This force is perpendicular to the fluid motion (relative wind) and its application point is at the center of pressure, whose position is defined mainly by the rocket geometry. By placing fins at the bottom of the rocket, the center of pressure can be moved below the center of gravity, so that the lift force can be used as passive stabilisation torque, aligning the rocket in the direction of relative velocity ($\alpha = 0$). The relative wind being the vector sum of the rocket velocity and the external wind velocity, a launch rail is used at the beginning of the flight to accelerate the rocket vertically at high speed.

Another way to stabilize a rocket is to steer the thrust force in order to create a torque that will be controlled by the on-board GNC. For this a TVC system has to be developed to either steer the rocket engine, or to interact with the exhaust gas to control its direction.
The thrust magnitude being usually very high, a thrust redirection of only 5 to 10 degrees is typically necessary to control the rocket rotation around its $\hat{\mathbf{x}}^b$ and $\hat{\mathbf{y}}^b$ axes (pitch and yaw). By stabilizing the rocket in different orientations during the thrust phase, lateral forces can be generated with respect to the local-level frame, which allow to control the complete rocket position and velocity.
This method is widely used in commercial launchers, as it allows for full control of the rocket states, and can actively compensates external perturbations during the flight. In our case, it cannot fully replace the passive fins stabilization, as our rocket also has a coasting phase during which the rocket engine is shutdowned, and thus the TVC is ineffective

Chapter 1

GNC Architecture

The first step in this project was to develop a new software and hardware architecture for the GNC and define interfaces with the rocket electronics and test bench for on-ground validation.
Overall, a complete development and testing framework for our GNC was developed. This chapter presents the objectives of this new architecture, and describes both hardware and software parts.

1.1 Objectives

First, a description of the previously developed architecture will be presented, to better explain the rationale behind the new one.
The main electronics of the rocket, called the avionics (AV), is a modular stack of microcontrollers connected together via CAN bus. The previous navigation and control algorithm were implemented directly on one of these microcontrollers which had the advantage of simplifying the interface, as the avionics stack can simply be extended with a dedicated microcontroller for the GNC. The biggest drawback of this method is the low computational performance which constrain the complexity of the algorithm. The second issue is due to the complexity of programming these microcontrollers which led to the algorithms being first developed and tested on the computer with languages such as Matlab or Python, then ported to embedded C with no guarantee that the tests done on simulation were relevant anymore. These two issues were probably the cause of the many failed flights regarding the navigation and control, as the algorithms deployed on the microcontroller were usually too simple and not tested enough on the real hardware.
From these observations, it was clear that a better framework was needed for the Icarus project, as the guidance, navigation and control algorithms that will be developed for this project are quite more complex. The main objectives of the new architecture are presented below:

- **Portability**: The development environment should be very similar to the deployment environment, so that the modifications needed to embedd the GNC are minimized. Ideally, it is the deployment environment that should be changed to be closer to what the student are used to work with, i.e. higher level framework and programming language compared to bare-metal C.
- **Performance**: By providing enough processing power to run the GNC, the student can focus on developing more complex algorithm with little compromise due to hardware limitation, and less time spent on tuning and deployment.
- **Modularity**: The architecture shall be easily adaptable to the different prototypes and GNC that will be developed at the EPFL Rocket Team. These different projects include changes in the system, mission objective, sensors and actuators.
- **Testing**: It shall be easy to connect the GNC to different test environments, like simulator or test bench, so that the GNC can be tested on Software-in -the-Loop (SIL), Processor-in-the-loop (PIL) and Hardware-in-the-loop (HIL).
- **Compatibility**: This architecture shall rely as much as possible on the current architecture and existing hardware, to avoid having to redevelop existing and flight proven functions.

1.2 Hardware architecture

In this section, we describe the current avionics module and the new hardware that was developed to deploy GNC. As explained before, the avionics is a modular system of four custom microcontroller boards called host boards, which are distributed on the rocket and communicating together via a CAN bus. Each board has specialized hardware connected to it to perform specific functions:

- **Sensor board**: Host four IMUs (BNO055) with 3D gyroscope and accelerometers, and four barometers (BME280). The board continuously sample the sensors, and merge them into one sensor value by removing outliers, and averaging the rest. This allows to have more robust and more precise sensor data at 21 Hz.
- **Telemetry board**: Communicate with the ground segment to receive ignition commands, and transmit flight data via radio telemetry.

- **Propulsion board**: Handle rocket engine operations and monitoring by controlling the main valve to control the thrust and measuring pressure and temperature of the engine in real time.
- **GNC board**: New board developed by the Icarus project. Host a Raspberry Pi 4 (Compute module) to run the GNC algorithms.

A complete block diagram of the hardware is presented on figure 1.2. It can be seen that the host boards are distributed on two locations: the avionics bay and the valve bay (Propulsion), with each having its own power supply. The general architecture of the avionics has been developed during project Eiger (2019) and thanks to its modularity allowed the different boards to be easily upgraded throughout the years. It also allowed to add our newly developed GNC board without the need to create new hardware interface. Once manufactured, our host board could simply be plugged on the CAN and power bus of the avionic bay, and attached to the structure of avionic module similar to the other telemetry and sensor boards.

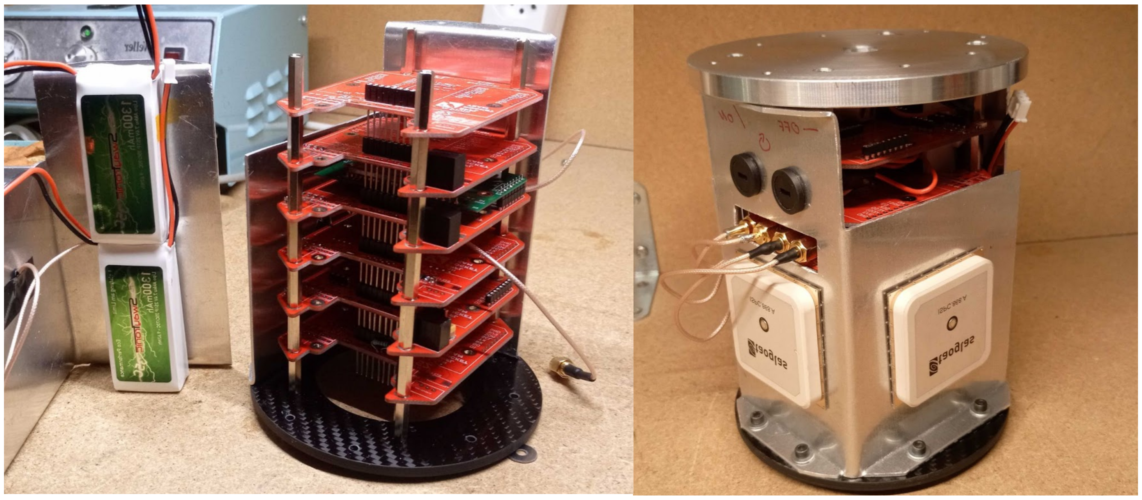


Figure 1.1: Assembled avionics module, without (left) and with (right) its outer aluminium protection shell. Note that more host boards are shown here to demonstrate the possibility to extends the stack with new boards

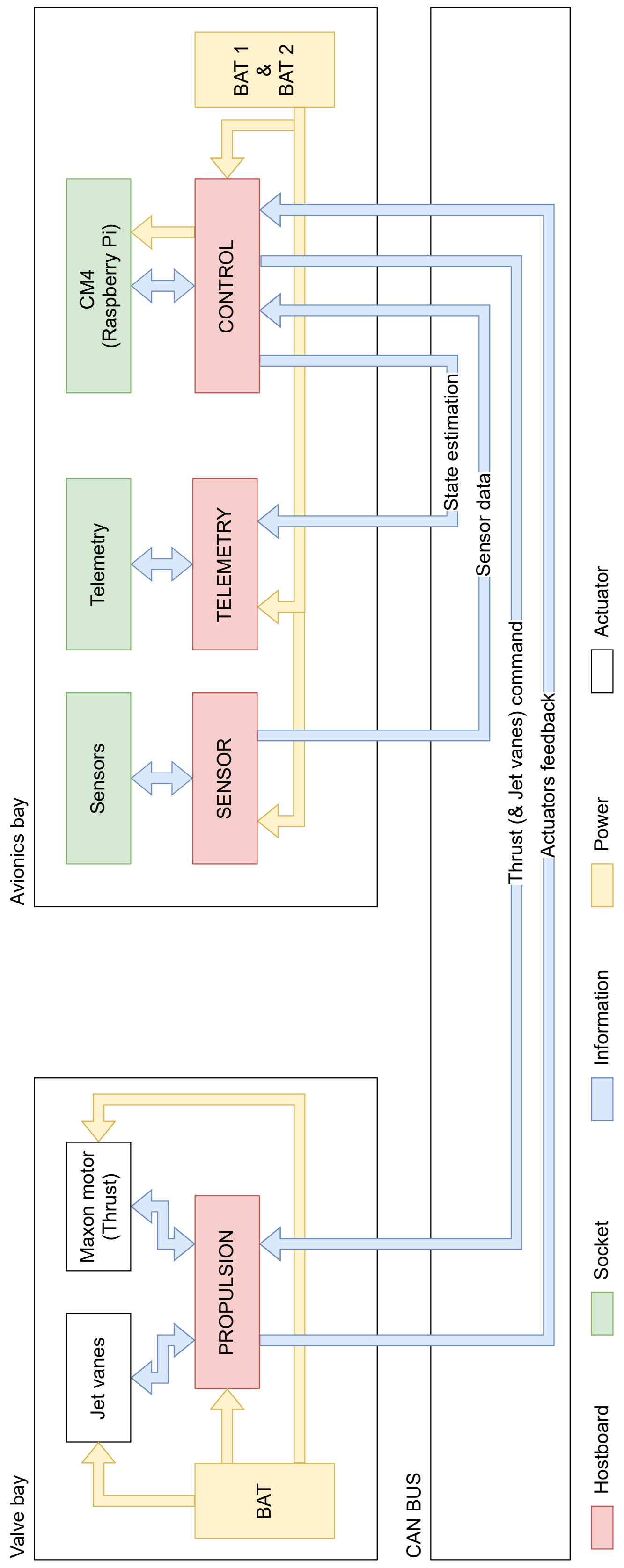


Figure 1.2: Complete block diagram of the augmented avionics to include the GNC board. *Image credit: Iacopo Sprenger*

1.2.1 GNC module

The new GNC board developed by project Icarus is briefly presented here. The development of this board was done by Iacopo Sprenger during a semester project, and is thus explained in more detail in his report [1].

At the heart of this module is a Raspberry Pi 4 board, which satisfy the requirements of **Performance** and **Portability** described previously. This single-board computer runs Linux (Raspian) which allow standard C++ and Python libraries to be used, and has very high performance compared to the STM32 F4 microcontrollers used on the host board, while being easy to embed on our AV module thanks to its small size. A summary of its important specifications is shown on table 1.1.

Name	Raspberry Pi Compute module 4
Processor	BCM2711 quad-core ARM Cortex-A72
Frequency	1.5GHz
RAM	4GB
Storage	8GB eMMC
Wireless	Bluetooth and WiFi
Dimensions	55*40*4.5mm
Mass	12g
Max. power	15W

Table 1.1: Specifications of the computer board used for the GNC

To interface with a host board, a custom PCB was developed to route power and communication from host board to Raspberry Pi. As Raspberry Pi requires more power than the host board can deliver with its buck converter, this Printed Circuit Board (PCB) integrates its own 30W buck converter, which is directly connected to the battery pin of the host board. To reduce power consumption, the "Enable" pin of the Raspberry Pi is also controlled by the host board, so that the Raspberry Pi is kept in the sleep mode during all ground operations, and is only activated a few minutes before liftoff.

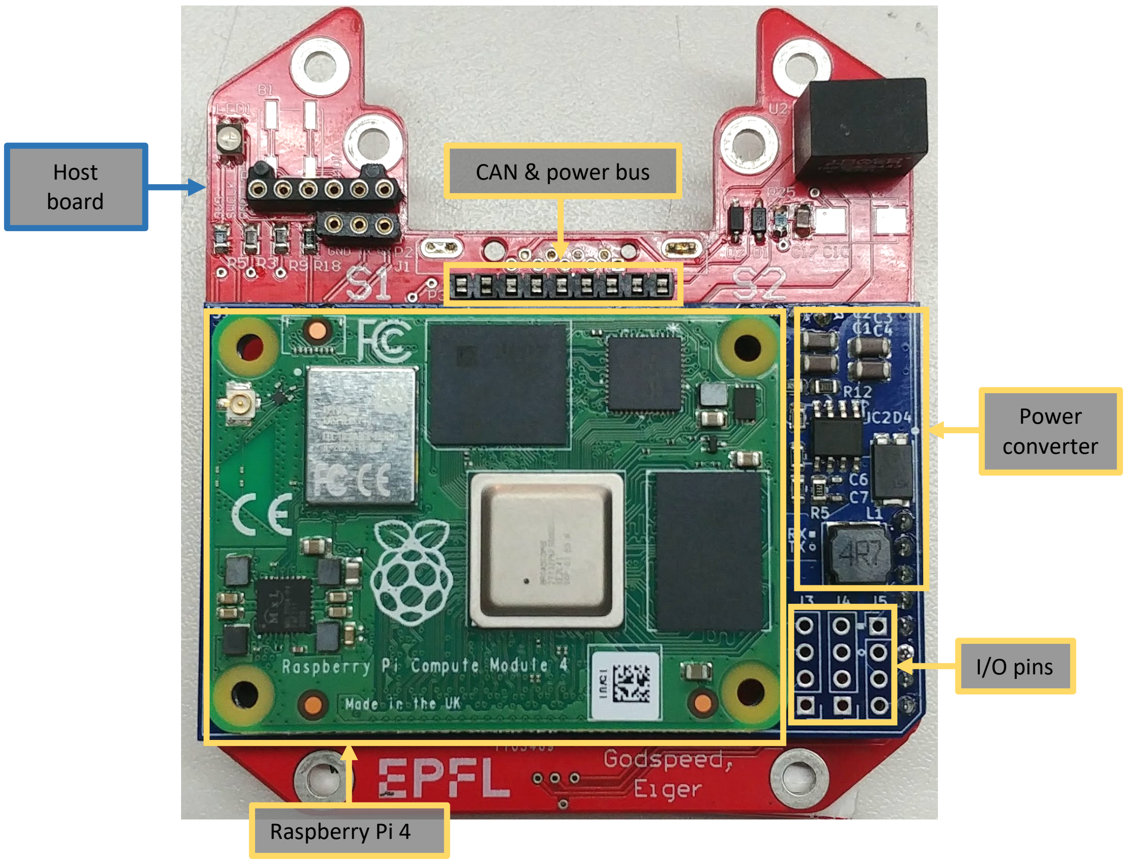


Figure 1.3: Assembled GNC board and its functions. One can see the three layers of hardware: in red is the host board, in blue is the custom PCB, and in green is the Raspberry Pi

1.3 Software architecture

1.3.1 Overview

All the GNC algorithms are directly deployed on the Raspberry Pi using the ROS framework [2], while the sensor input and the actuators output are managed by the avionics. This separation is very suitable for testing, as it allows to place the GNC algorithms as is in simulation environment, test bench, or the real hardware with very little modifications.

The general block diagram of this architecture is presented on figure 1.4.

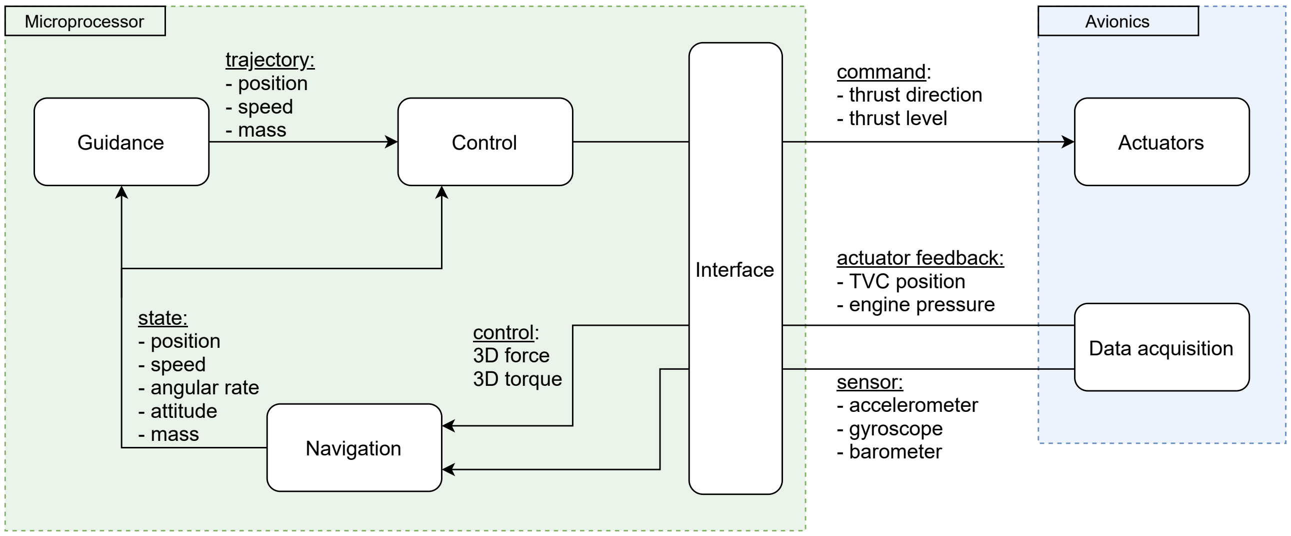


Figure 1.4: General architecture of the GNC algorithms

On the green box are all the programs running on the Raspberry Pi. The rest of the avionics is represented with the blue box and provides mainly sensor input (Data acquisition) and actuator output. The interface with the GNC is done on the Raspberry Pi, handling the low level UART communication with the host board, and converting data between ROS messages and UART packets.

The Guidance, Navigation and Control algorithms will be explained in more detail on the following sections. However, the exact implementation of these algorithms will be presented in the following chapters, as they depend on the type of mission and system considered.

Navigation

The Navigation algorithm is responsible of estimating the full state of the rocket in real time. Using the sensor data provided by the AV, it has to reconstruct the position, velocity, attitude, and angular rate. The mass of the rocket is also estimated, as it varies significantly during the flight.
Traditionally, a Kalman filter is used to merge sensor data with a prediction of the rocket state from the rocket equation of motion. This allows to have a high refresh rate of around 100Hz, as the state prediction can be done independently of the sensor update.

Guidance

The Guidance algorithm has the task of taking the high level objectives defined by the user, such as the target apogee and the state constraints, and finding an optimal trajectory to the targeted state defined by the user.
Typically, the rocket equations of motion are used to connect these two states, and knowledge of the rocket properties such as mass, aerodynamic coefficients or actuators limits are used to compute a feasible trajectory. Fuel consumption serves as a minimisation criterion.
This guidance strategy is thus very flexible and could be used as much for ascent guidance, as it could be used for descent and landing planning. As it will be presented in the following chapters, a Model Predictive Control (MPC) is a very good candidate for this algorithm, as it naturally integrates knowledge of the system dynamics and user constraints to find an optimal trajectory.
To take into account external perturbations and model errors, this trajectory should be recomputed periodically and sent to the Control algorithm at around 5Hz.

Control

The Control algorithm is the last stage of the GNC. Using the state estimation from Navigation and the trajectory target from Guidance, it has to send commands to the actuators to control the thrust direction and magnitude.
Because the Guidance is already working on the long term trajectory, the Control is focused more on short term targets, without having knowledge of the flight long-term objectives. This is quite important, because besides following the Guidance trajectory, the Control also has to stabilize the rocket close to the vertical orientation, which requires a higher control frequency of around 50Hz

to compensate for external perturbations.

By controlling the thrust direction of the engine, the Control can slightly pitch the rocket away from the vertical and control its horizontal position and velocity. As the rocket is mainly vertical, the altitude and vertical speed are primarily controlled by the thrust magnitude. Finally the propellant consumption is controlled by the duration and magnitude of the main thrust. By blindly following a trajectory in position, velocity and mass as a function of time, the Control can thus meet the objective of propellant consumption defined by the Guidance optimal trajectory, without having to solve itself the complete flight optimization problem.

In theory, a simple controller such as the cascade controllers used for drones could be used for Control. However, the current strategy is to use MPC due to its capability to handle directly complex system.

Time synchronization

The algorithms are running asynchronously, meaning that they continuously do their work at a fixed frequency, without waiting for information from other algorithms.

For example, the Guidance and Control just use the last received state estimation to compute the optimal trajectory and control, which makes them more robust in case Navigation fails or its state message is lost. One exception to this is the Interface algorithm which is synchronized to input and output messages to relay them directly and minimize delays.

To have the different algorithms working well together, it is important to have a common time and state machine. For this, a dedicated algorithm is used to estimate the phase of the flight, and the time since liftoff, which serves as a reference to all algorithms. The phases are:

- **Idle**: The initial state of all algorithms at startup. Corresponds to the time when the rocket is on the launch rail.
- **Rail**: The engine is ignited and kept at full throttle so that the rocket gains a maximum of speed while it is on the launch rail. All algorithms start to run, except for the Control.
- **Launch**: The rocket has exited the launch rail and is now fully controllable. All algorithms are thus fully active.

- **Coast**: The engine has been stopped, and the rocket is decelerating due to gravity and drag, No more Thrust Vector Control is possible, Control is stopped.

1.3.2 ROS framework

ROS (Robot Operating System) [2] is a middleware which provides a set of programming tools for robotic projects. One of the key feature of ROS is its peer-to-peer connectivity which allows communication between several processes called nodes. ROS is language-neutral, so different nodes can use different programming languages and can be distributed across different machines on the same network. This allows for very modular programming as each function can be separated into different nodes. Several nodes can be regrouped into a package, which provides a coherent set of functions. Popular open-sources packages includes for example Rviz and RQt for 3D visualization and building graphical interfaces respectively. As the project can easily integrates several packages, we can use these packages as is.

GNC package

The algorithms described in the previous section are all integrated in one package. This includes the algorithms, configuration files and scripts utilities. Figure 1.5 shows the minimum organization of a GNC package. The functions of each folders are:

- */config*: Contains the YAML configuration files used to define the rocket and launch environment parameters.
- */launch*: Contains ROS launch files to start different nodes combinations, depending on the environment: real flight, software in the loop (SIL) or processor in the loop (PIL).
- */log*: Contains one log file per flight. Use *rosbag* to replay the content of the file.
- */script*: Contains python scripts needed by the user for data parsing and analysis, for instance.
- */source*: Contains the source code of the GNC algorithms.

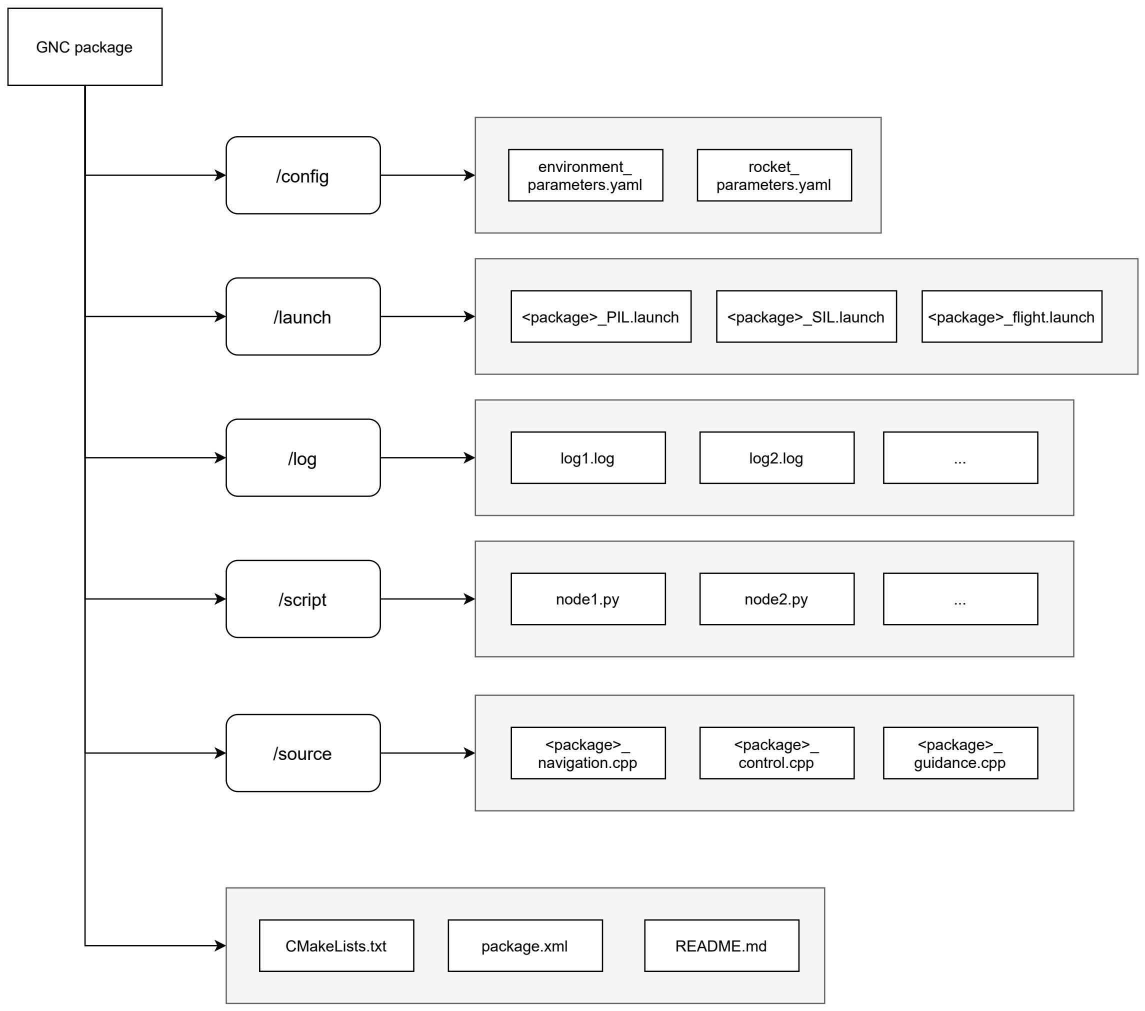


Figure 1.5: File tree of a typical GNC package. Each "<package>" has to be replaced with its real package name

Of course this organization can be completed with new folders and files to suits the needs of each project. The idea is to have a separate package for each new GNC project. Common utilities like post-processing scripts and custom ROS messages definition are placed on the simulator package that will be presented in chapter 2, as these are common to all GNC projects.

This modular architecture will allow to develop multiple GNC projects for the drone and rockets prototypes that will be presented in the following chapters. Once tested on the computer, these packages can be directly compiled and run on the Raspberry Pi, making the GNC adjustable to various simulation and tests environments as it will be shown in the following chapter.

Chapter 2

Simulator

The first step to test the GNC project is simulations. The simplest form of this is a SIL configuration where the GNC software is directly linked to an external simulation program on the same machine. This allows fast development and debugging of the GNC as it doesn't need to be deployed every time a change is made.
However, this is usually not enough for aerospace applications, the hardware that will be running the GNC needs to be tested. In this project, we separate these tests in two configurations: first the embedded processor is tested with PIL configuration with the GNC deployed on the hardware and connected only to the simulator. Finally, the real actuators can be tested HIL configuration, by having the full system mounted on a test bench, and direct measurement of the actuators being used in real time to drive the simulation. The latter configuration has not been implemented yet, but would allow very reliable ground testing of the GNC.

Until now, the existing simulator did not have these features. A previous Matlab simulator [3] has been used until now to do open-loop simulations of rocket flights to help design the rocket and its engine, but this simulator cannot interface with complex GNC to do closed-loop simulations, is not real time, and would be hard to link with our hardware.
Thus, a new simulator was developed from the ground-up. This chapter explain all its features, as well as the development and test process.

2.1 Features

To make the simulator highly compatible with the GNC architecture described previously, the ROS framework is used. Most of its features are distributed on different nodes running asynchronously, which makes the simulator very modular to be configurable and adaptable.

The complete block diagram of this simulator is shown on figure 2.1. The HIL configurations is presented as it is the most complex one. For PIL configuration, the *test bench* module is simply removed. For SIL configuration, the *TVC system* and *CAN interface* are also removed, so that the *GNC algorithm* can be directly connected to the *Simulator*. But in general the working principle is the same for all configurations. The *Simulator* computes in real time the full state of the rocket based on its previously computed state and the commands from the GNC. This state is used to emulate sensor data that are sent to the GNC to update its control commands.
As the simulator uses ROS, many utilities can be used, like the complete logging of nodes interactions, 3D visualization and live plots, and simulation configuration with YAML files.

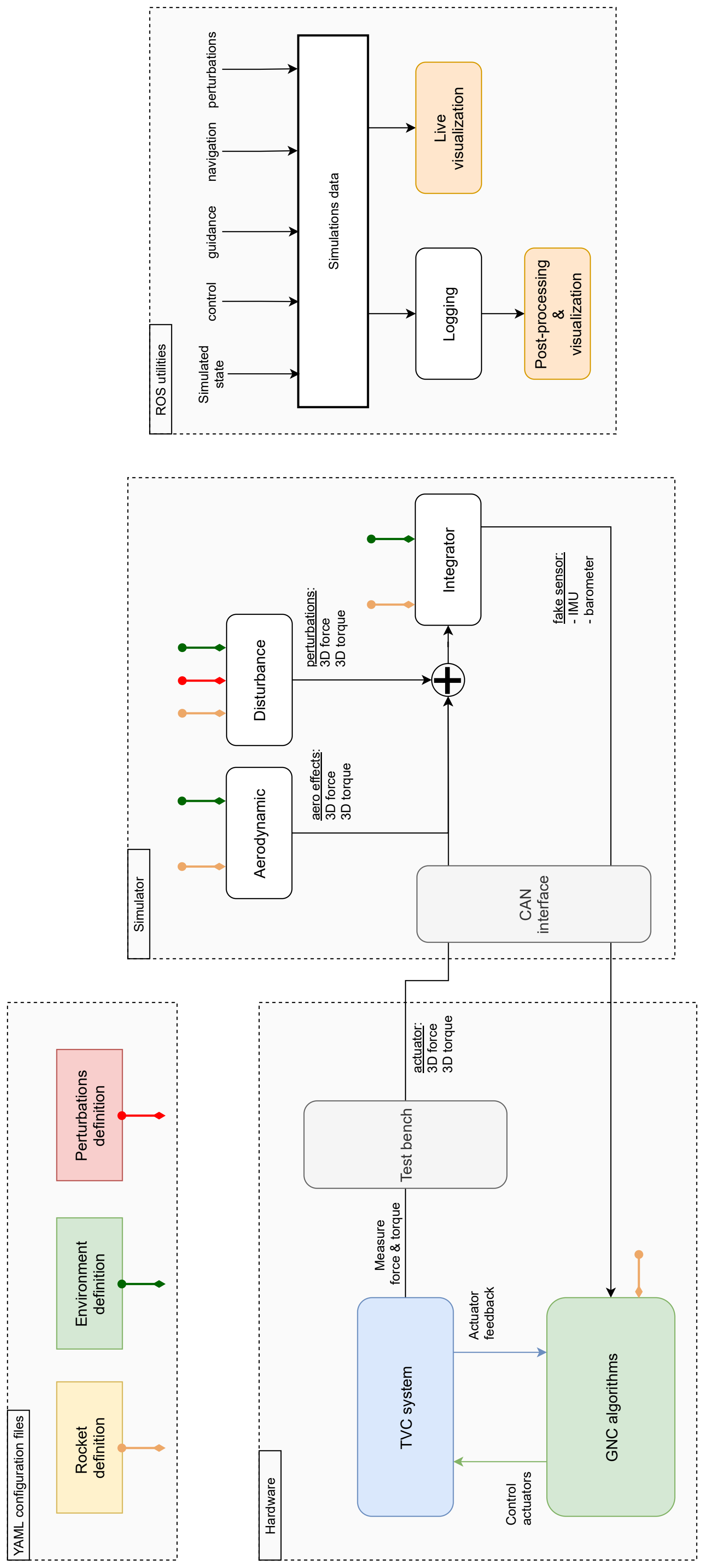


Figure 2.1: Complete block diagram of the new simulator, in HIL configuration

2.1.1 Rigid body simulation

The dynamic of a rocket is modelled as a six-degree-of-freedom rigid body with varying mass [4]. One node called *integrator* is in charge of summing all forces and torques acting on the rocket, and by using Newton-Euler equations, we can derive the ordinary differential equations of the rocket motion. These equations are then numerically integrated using *Boost Odeint* C++ library.

The first force to consider is the weight $\mathbf{w}$ of the rocket:

$$\mathbf{w}(m_p, p_z) = -m_{tot} * g * \hat{\mathbf{z}}^l = -(m_{dry} + m_p) * \frac{\mu_E}{(R_E + h_0 + p_z)^2} * \hat{\mathbf{z}}^l \tag{2.1}$$

with m_{dry} the dry mass of the rocket, and m_p the propellant mass, varying with time. The variation of gravitation is modelled using μ_E, the standard gravitational parameter of the Earth, and the distance of the rocket to the center of the Earth with R_E the Earth radius, h_0 the ground altitude and p_z the altitude above ground level. This force depends thus on the propellant mass and vertical position states. It is colinear with the vertical axis $\hat{\mathbf{z}}^l$ of the local-level frame.

The second force is the controlled force and torque generated by the rocket actuators. It is primarily composed of the thrust of the engine, but can also take into account other actuators such as airbrakes or cold gas thrusters:

$$\begin{cases} \mathbf{f}_c^l = \mathbf{R}^B * \begin{bmatrix} f_x^b & f_y^b & f_z^b \end{bmatrix}^T \\ \boldsymbol{\tau}_c^l = \mathbf{R}^B * \begin{bmatrix} \tau_x^b & \tau_y^b & \tau_z^b \end{bmatrix}^T \end{cases} \tag{2.2}$$

These two vectors being in body frame, they have to be transformed to the local-level frame using the rotation matrix $\mathbf{R}^B$:

$$\mathbf{R}^B = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x * q_y - q_w * q_z) & 2(q_x * q_z + q_w * q_y) \\ 2(q_x * q_y + q_w * q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y * q_z - q_w * q_x) \\ 2(q_x * q_z + q_w * q_y) & 2(q_y * q_z + q_w * q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix}$$

This rotation matrix is computed using the quaternion state $\mathbf{q} = (q_w, q_x, q_y, q_z)$ with q_w the real part and (q_x, q_y, q_z) the vector part of the quaternion.

Finally, the aerodynamics effects $\mathbf{f}_a$ and $\boldsymbol{\tau}_a$, and external random perturbations $\mathbf{f}_p$ and $\boldsymbol{\tau}_p$ are added. Due to their complexity, they are computed on two separate nodes and will be presented in the following sections.

The four forces and four torques can be summed to get the total force and torque $\mathbf{f}_{tot}$ and $\boldsymbol{\tau}_{tot}$ which can be used as the starting point of the differential equation:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^l \\ \mathbf{v}^l \\ \mathbf{q} \\ \mathbf{w}^l \\ m_p \end{bmatrix} \qquad \dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v}^l \\ \frac{\mathbf{f}^l_{tot}}{m_{tot}} \\ 0.5 * \mathbf{w}^l \otimes \mathbf{q} \\ \boldsymbol{\tau}^l_{tot} * (\mathbf{R}^b * \mathbf{I}^b)^{-1} \\ \frac{f_z}{g_0 * Isp} \end{bmatrix} \tag{2.3}$$

The state vector $\mathbf{x}$ is composed of the position $\mathbf{p}$, the velocity $\mathbf{v}$, the quaternion $\mathbf{q}$, the angular rate $\mathbf{w}$ and the propellant mass m_p. All states are in local-level frame.
The inertia matrix $\mathbf{I}^b$ is in body frame for simplicity reason, and is used to compute the angular acceleration from the total torque. The parameter Isp is the specific impulse of the rocket engine, relating engine thrust force to the propellant mass variation.

Using the odeint C++ library, these differential equations can be solved with the Dormand-Prince method (*runge_kutta_dopri5*). Currently, the integration step size is 10ms, and a new integration is made every 10ms so that the simulation is real time. On a standard laptop, the integration computation takes less than 200 microseconds, so if need be, the integration step could be reduce even more to get a better resolution.
The previous version of this node was programmed in Python, so due to performance issues, the maximum integration period that could be obtain was only 50ms, which made the simulation unreliable and diverging after some time.

2.1.2 Aerodynamics effects

Our rocket launches being at low altitude, aerodynamic forces have very important effects that need to be taken into account.
Due to the complexity of modeling these effects using analytical equations, an already existing Python transcription of the previous matlab simulator was used and wrapped on a separate ROS node. This program uses the current state of the rocket, its geometry, and environmental conditions such as wind speed and ground pressure to compute the drag and lift forces and torques.
A brief description of these forces will be given here. For the full details of the aerodynamics effects, please refer to [3].

The drag force is the resistance of an object when moving through a fluid. It is colinear and in the direction of the relative fluid motion:

$$D = \frac{1}{2} * \rho * C_d * A_{ref} * v_{rel}^2 \tag{2.4}$$

The fluid relative velocity is v_{rel}, and its density is ρ. The geometry of the object is represented with its cross-section surface A_{ref} and its drag coefficient C_d. This drag coefficient is dependent on the mach number and angle of attack of the rocket, but in practice is constant for most of the flight.

Similarly, the lift force can be expressed as

$$L = \frac{1}{2} * \rho * C_{n\alpha} * \alpha * A_{ref} * v_{rel}^2 \tag{2.5}$$

With $C_{n\alpha}$ the lift coefficient. Note that here the lift increases with the angle of attack α, which is the angle between the $\hat{\mathbf{z}}^b$ axis of the rocket and the fluid motion. Because the rocket has lateral velocity, and a constant wind speed is used throughout the simulation, α varies typically between 0° and 90° during the flight, reaching its maximum at liftoff and apogee due to the low rocket velocity.
Both of these forces are applied at the center of pressure of the rocket, which is determined by

the geometry of the rocket as the weighted average of the lift coefficient of each rocket component: fins, body, nosecone, etc.
By design, the center of pressure is placed between 2 and 5 body diameters away from the center of pressure, so that the lift force generates a corrective moment to stabilize the rocket at low angle of attack.

2.1.3 Perturbations

To test the robustness of the GNC, and add variability to our simulations, random perturbations forces and torques are added. These perturbations represents small effects that are difficult to model with certainty or that are inherently stochastic.
These effects are modelled in a separate node called *disturbance.py*, and tuned using the *perturbations_parameters.yaml* file.
At the moment, only two types of perturbations are modelled. The first one is due to wind gusts acting non-uniformly on the rocket. This is modelled using the drag equation 2.4, and horizontal gusts of wind with a velocity following a Gaussian distribution around the constant wind of the simulation. The non-uniformity of the wind gust is modeled by applying the resulting force at a random point along the rocket length, which can thus generates destabilizing torques.

The second perturbation is the thrust misalignment with respect to the center of mass. This can be due to the rocket engine being not centered well, or tilted with respect to the rocket $\hat{\mathbf{z}}^b$ axis, or the center of mass not being in the center of the rocket. All of these results in lateral forces creating a constant torque. To add variability, the exhaust gas is also modelled as a jet with small random variation in direction.
These two effects are represented on figure 2.2. Many other effects could be added on this node to assess the robustness of the GNC to manufacturing errors such as fin tilt creating a roll ($\hat{\mathbf{z}}^b$ axis) torque or drag asymmetry due to the rocket being not perfectly symmetric.

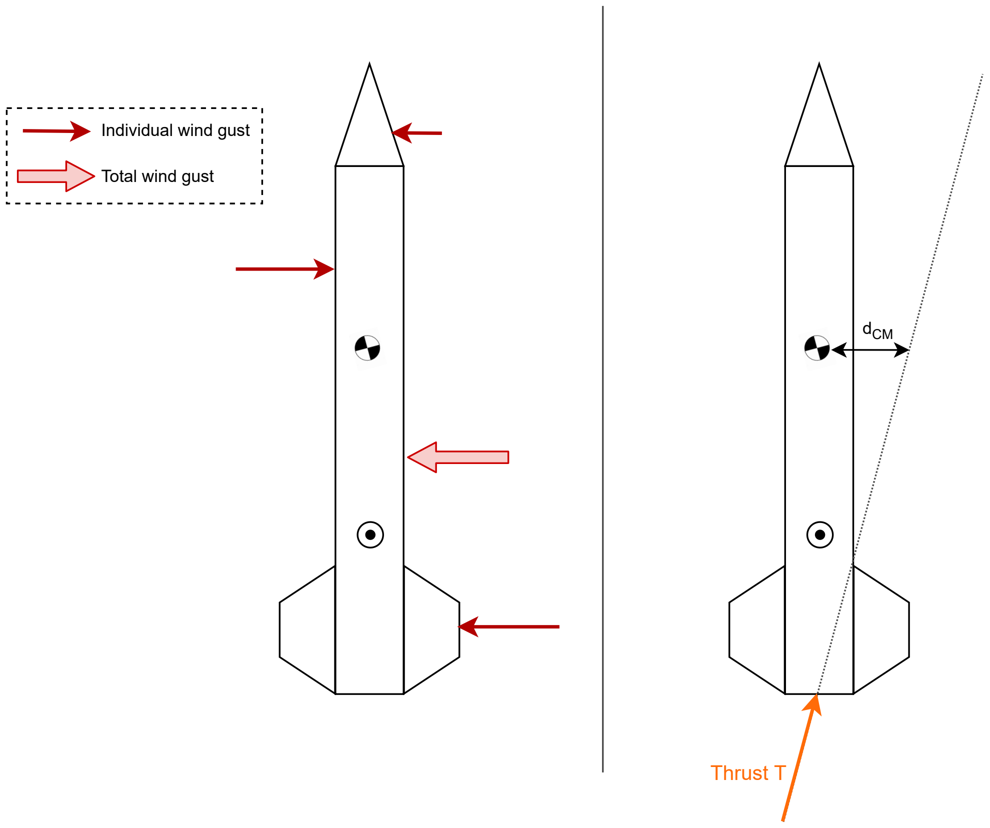


Figure 2.2: Wind gust perturbations (left) and thrust misalignment perturbation (right)

2.1.4 Sensor simulation

To interact with the GNC algorithm, the simulator has to send information in the form of sensor measurements, as the GNC expects them during a real flight. Currently, the avionics has three types of sensors: three-axes accelerometer and three axis gyroscope integrated into an Inertial Measurement Unit (IMU), and barometer. These measurements are relatively simple, and can thus be directly generated when the state is updated inside the *integrator* node [5].

The accelerometer model is:

$$\mathbf{a}^b_{IMU} = \mathbf{R}^{b^{-1}} * (\dot{\mathbf{v}} + g * \hat{\mathbf{z}}^l) + \mathbf{r}_a \ , \ \mathbf{r}_a \sim N(\boldsymbol{\mu}_a, \boldsymbol{\sigma}^2_a) \tag{2.6}$$

$\dot{\mathbf{v}}$ is the time derivative of the velocity in local-level frame to which the gravity vector $g*\hat{\mathbf{z}}^l$ has to be added to get the true acceleration due do external forces. Then using the inverse rotation matrix

$\mathbf{R}^{b^{-1}}$, this acceleration is transformed to the body frame in which our strap-down IMU is attached. Finally, Gaussian noise is artificially added with the vector $\mathbf{r}_a$ having mean and variance $\boldsymbol{\mu}_a$ and $\boldsymbol{\sigma}_a^2$.

The gyroscope model is a bit simpler, as we simply have to transform the angular velocity state $\mathbf{w}^l$ from local-level to body frame, and add predefined noise:

$$\mathbf{w}_{IMU}^b = \mathbf{R}^{b^{-1}} * \mathbf{w}^l + \mathbf{r}_w \ , \ \mathbf{r}_w \sim N(\boldsymbol{\mu}_w, \boldsymbol{\sigma}_w^2) \tag{2.7}$$

Finally the barometer data is directly generated from the vertical position state:

$$h_{baro} = p_z + r_b \ , \ r_b \sim N(\mu_b, \sigma_b) \tag{2.8}$$

This set of sensor data could easily be extended with this method, for example to simulate data from a Global Navigation Satellite System (GNSS) receiver or a magnetometer, which are real sensors likely to be integrated in the avionics in the future.

2.1.5 Hardware connection

The features described above allow to run SIL simulations by having the GNC and the simulator in two separate ROS packages on the same computer and started together with a dedicated launch file. Custom ROS messages allow efficient communication of the simulated sensor and the GNC commands between the two packages.
To run PIL simulations, the GNC package has now to be deployed on the Raspberry Pi, while the simulator package is kept on the computer. The first idea was thus to use a feature of ROS which allow nodes to be started on different machines of the same local network, with very little modification to the packages themselves. However, this method proved to be unreliable due to do the high communication delay over network of around 6ms, with spikes up to 15ms.

A different approach was thus taken. As the avionics consists of a modular stack of board, it is easy to remove or shutdown the sensor board, which will be replaced by the simulator, sending simulated sensor data to the AV CAN bus instead of the sensor board.

As shown in figure 2.3, the same hardware architecture is used, as presented in chapter 1, and the Simulator can simply be plugged in the CAN bus as any other host board would be.

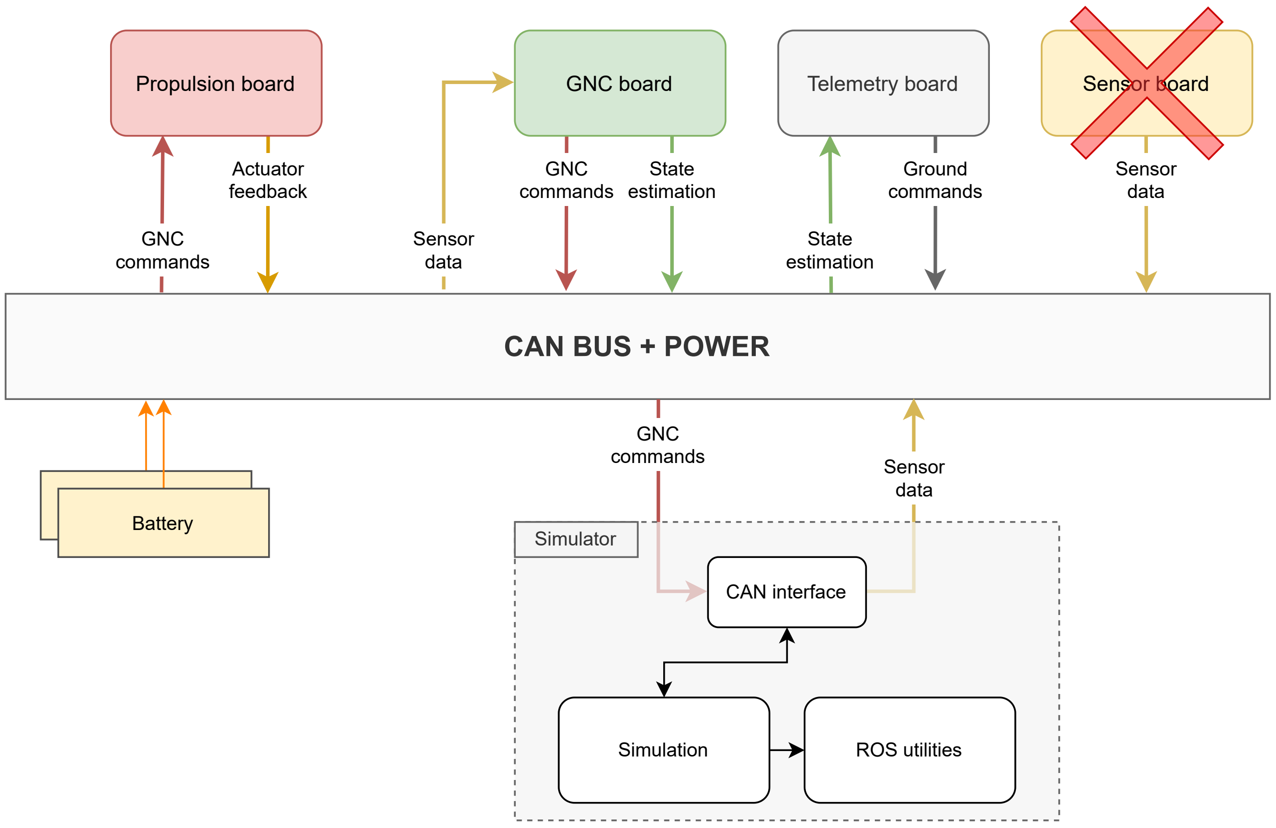


Figure 2.3: Block diagram of the PIL and HIL setup

A dedicated node called *CAN_interface* takes care of the ROS messages to USB message conversion, then a USB to CAN hardware adapter is used for the physical connection to the CAN bus. This configuration allows any host board to be removed or added depending on what needs to be tested. A minimal configuration with only the *GNC board* and the Simulator communicating via CAN is thus possible, but other host boards can be added to test their compatibility with the GNC. Typically the *Telemetry board* can transmit the GNC activation command to power on the Raspberry Pi, and send back to the ground station the state estimation from the GNC, and the *Propulsion board* can control the actuators to assess the effect of the GNC commands.

To avoid interference with the simulation, the *Sensor board* has to be completely deactivated. As the *Propulsion board* also sends back actuator feedback, it must be deactivated too if in PIL configuration, as the rocket engine won't be used, and thus the Simulator provides actuator feedback.

To run HIL simulations, the *Propulsion board* is activated and connected to the rocket engine as usual. A complete 6DOF test bench measuring all forces and torques is also used to feedback the real effect of the actuators to the Simulator, as it was shown in figure 2.1.
For now, the HIL configuration is not fully ready, as the test bench is still under development and some work is still needed to configure the Simulator properly to work with this test bench. Typically the simulation part of the Simulator should be deployed on a dedicated Raspberry Pi so that it can retrieves directly the force and torque measurement from the test bench's electronics. The visualization tools from ROS can be kept on a laptop connected to the same ROS network as the Raspberry Pi, in order to provide feedback to the user in real time. This configuration would also allow the user and expensive computer to be kept at a safe distance, and place the less risky equipment such as the Raspberry Pi and the test bench close to the rocket engine for maximum performance.

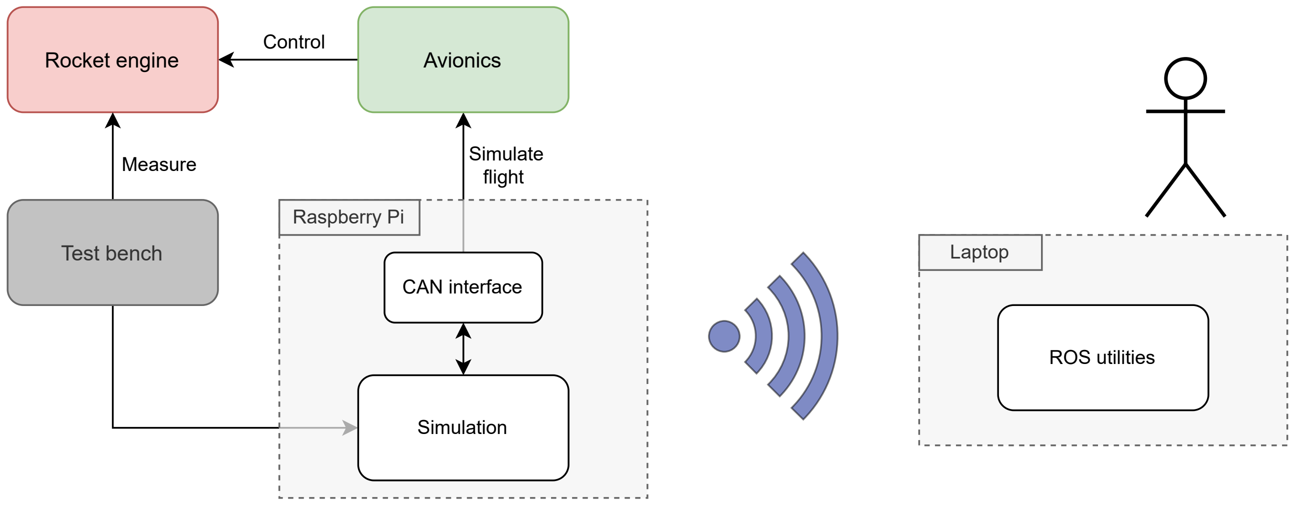


Figure 2.4: High level block diagram of the HIL configuration

2.2 Validation

Before using this Simulator, it is important to thoroughly test it, as a large part of the ground testing will rely on the accuracy of our simulations.

2.2.1 Unit test

The first simple tests are units test. These are simple motions that are easy to analyse such as 1 DOF translation and rotation. Here, the result of the simulator can directly be compared to those of analytical formula.

We start our series of unit tests with a free fall, without any perturbations or aerodynamic effects.

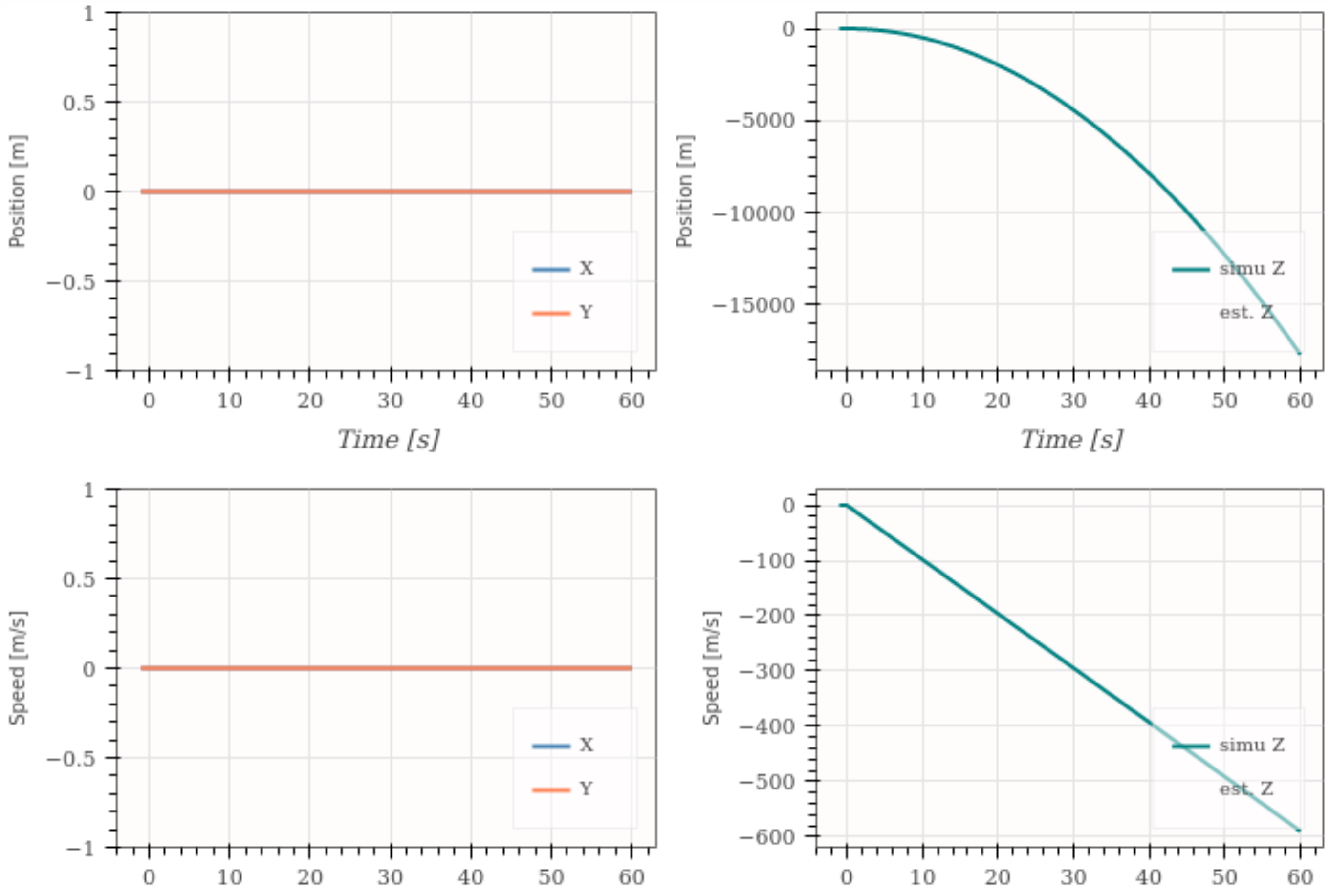


Figure 2.5: 3D position and velocity of the rocket in free fall. No thrust, aerodynamic force, or external perturbations are added

We see in figure 2.5 the classic parabola for the altitude, and linear curve for the vertical speed. The final position and altitude at 60 seconds matches perfectly the theory. Lateral position and speed, as well as the attitude of the rocket are all kept constant as expected. Exactly the same results were obtained regardless of the initial rocket orientation.

Then a hovering test is made. The rocket thrust is kept vertical, and compensating perfectly the weight, The Isp parameter is overwritten with a very high value so that mass variation due to the rocket thrust is kept minimum.

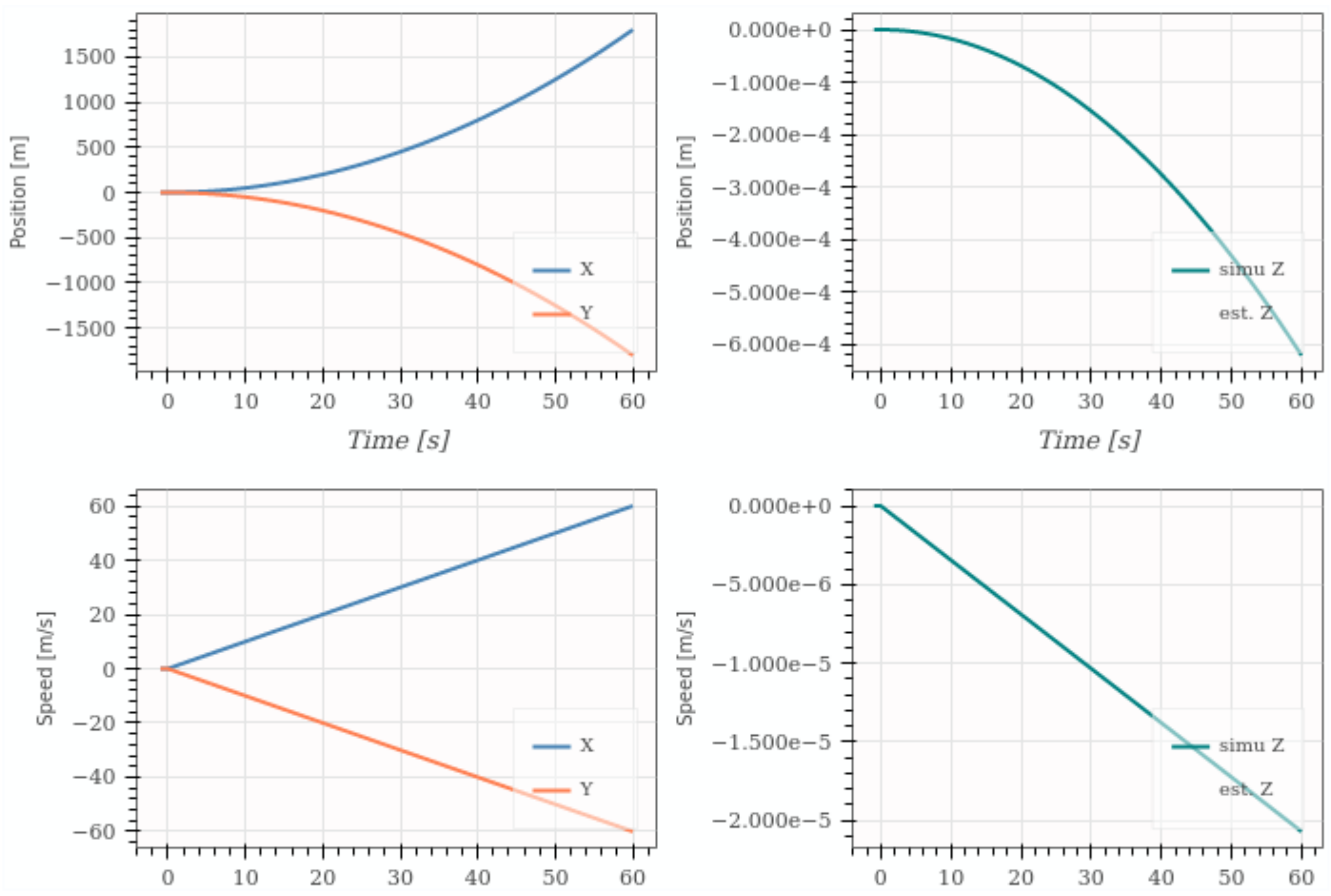


Figure 2.6: 3D position and velocity of the rocket when hovering. A lateral thrust is used to have 1g of acceleration in X and -1g in Y

As we can see in figure 2.6, slight numerical imprecision result in a deviation of -0.6mm in altitude after 60s, which is quite acceptable. Lateral motion is exactly as expected due to 1g of acceleration in $\hat{\mathbf{x}}^l$ and -1g in $\hat{\mathbf{y}}^l$.

The last test of the 3D rigid body simulation involves both rotations and translations in 3D. The rocket is initially tilted at 45° around the $\hat{\mathbf{y}}^l$ axis, and has an angular velocity around the $\hat{\mathbf{z}}^l$ axis of one rotation per second. The only forces are the constant weight of the rocket and the thrust of the engine, which is scaled to exactly compensate the weight of the rocket in the $\hat{\mathbf{z}}^l$ axis. The total force on the rocket is thus:

$$\mathbf{f}_{tot} = \mathbf{R}_z(\theta) * \mathbf{R}_y * \mathbf{f}_c^b + \mathbf{w} \tag{2.9}$$

with $\mathbf{w}$ the weight and $\mathbf{f}_c^b = \begin{bmatrix} 0 & 0 & m_{tot} * g * \sqrt{2} \end{bmatrix}^T$ the rocket thrust in body frame. A first constant rotation matrix $\mathbf{R}_y$ is used to represent the initial 45° tilt, and a second rotation matrix $\mathbf{R}_z(\theta)$ represents the rotation around the $\hat{\mathbf{z}}^l$ axis due to the angular rate w_z.Then $\theta = w_z * t$, with t the time since the start of the simulation.
After simplification of this equation and double integration to get the velocity and position, we have the following analytical result:

$$\mathbf{f}_{tot} = m_{tot} * g * \begin{bmatrix} cos(w_z * t) \\ sin(w_z * t) \\ 0 \end{bmatrix} \quad \mathbf{v} = \frac{g}{w_z} * \begin{bmatrix} sin(w_z * t) \\ 1 - cos(w_z * t) \\ 0 \end{bmatrix} \quad \mathbf{p} = \frac{g}{w_z^2} * \begin{bmatrix} 1 - cos(w_z * t) \\ w_z * t - sin(w_z * t) \\ 0 \end{bmatrix} \tag{2.10}$$

The average force over time is zero, so the velocity vector is constant. This constant equal zero for the $\hat{\mathbf{x}}^l$ and $\hat{\mathbf{z}}^l$ axes, so their position is constant, whereas the average velocity along the $\hat{\mathbf{y}}^l$ axis is positive, which creates a close to linear increase in position. This was verified in the simulation with the 3D visualization tool, and the graph shown in figure 2.7.

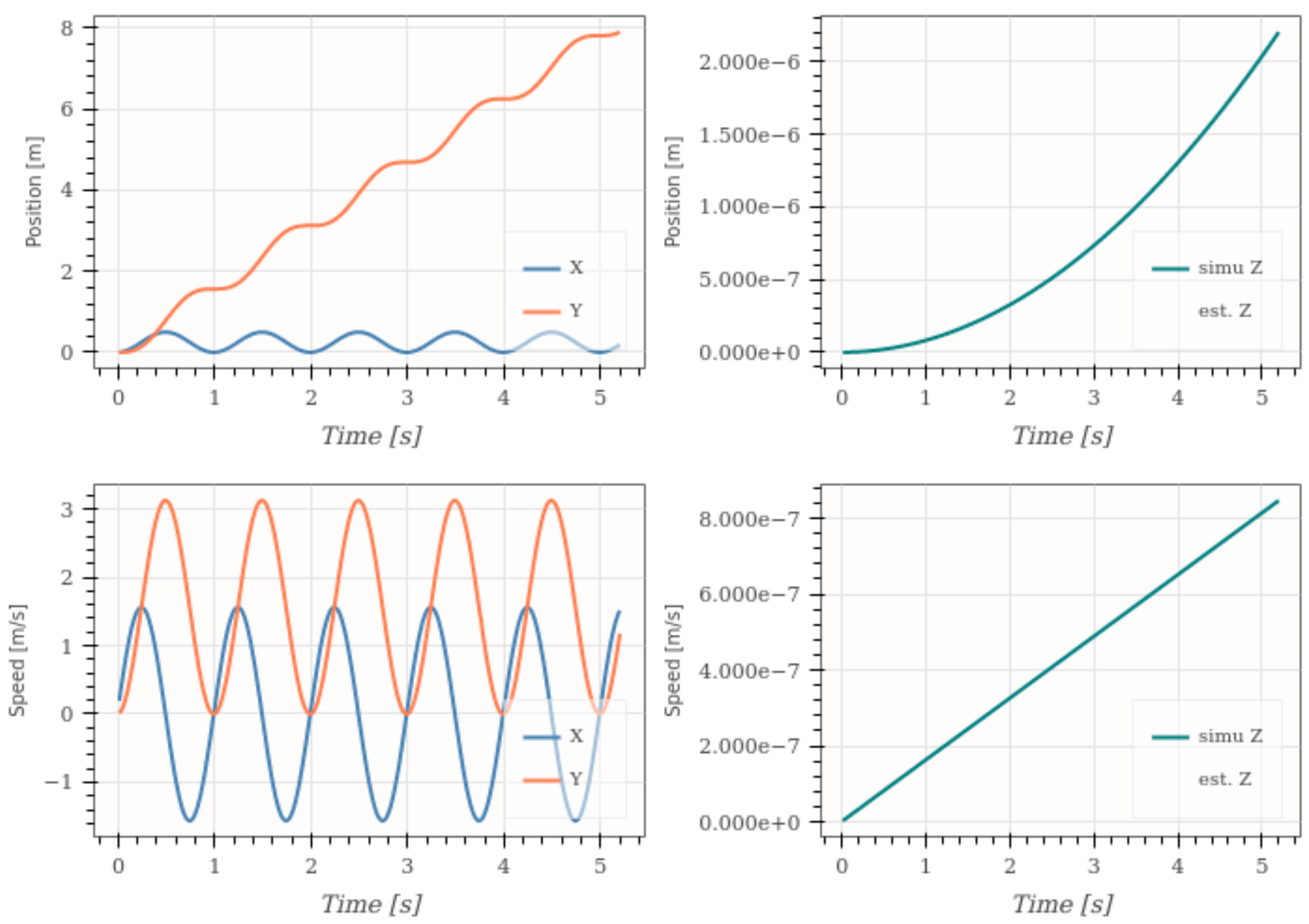


Figure 2.7: Simulation of a rotating rocket around the vertical axis, with initial tilt and weight compensating thrust.

2.2.2 Flight test

The complete simulator is tested by comparing it with flight data from 2020 high altitude rocket launch. This flight aimed at reaching 1200m with a solid propellant motor so no throttling was possible. However, the same avionics and sensors were used during the flight so we can directly compare the simulated IMU and barometric data to the logged sensor data.
The specifications of this flight are summarized in table 2.1.

Due to lack of data about the environmental condition of this flight, typically wind speed, atmosphere conditions and launch rail orientation, it is not possible to perfectly match the simulation and flight data. Thus this test serves more as a high-level test to verify all the features of the simulator are working as expected for a typical flight.

Specification	**Value**
Location and time	Wasserfallen 2020
Rocket model	Eiger/Bellalui 1
Target altitude	1200m
Motor	Solid motor (M2400)
Liftoff mass	45.25 kg
Length	4.16m
Sensor data	IMU, barometric altitude, temperature
Logging frequency	16.7 Hz

Table 2.1: Rocket launch specifications

Another difficulty is that the solid-propellant rocket motor used for these kind of flights have high variability of up to 10% in their thrust curve, resulting in different acceleration profiles for different motors of the same category. We can observe this on figure 2.8, where the simulated acceleration along the $\hat{\mathbf{z}}^b$ axis of the rocket, computed from the nominal thrust curve given by the manufacturer, has significant difference compared to the measured acceleration during the flight.
However, this difference is mainly in term of distribution of the thrust over time, so the average acceleration error is only $1.5m/s^2$. This means that the total energy (total impulse) given to the rocket by the motor is very similar, and we should expect the simulated apogee altitude to be close to the measured one.

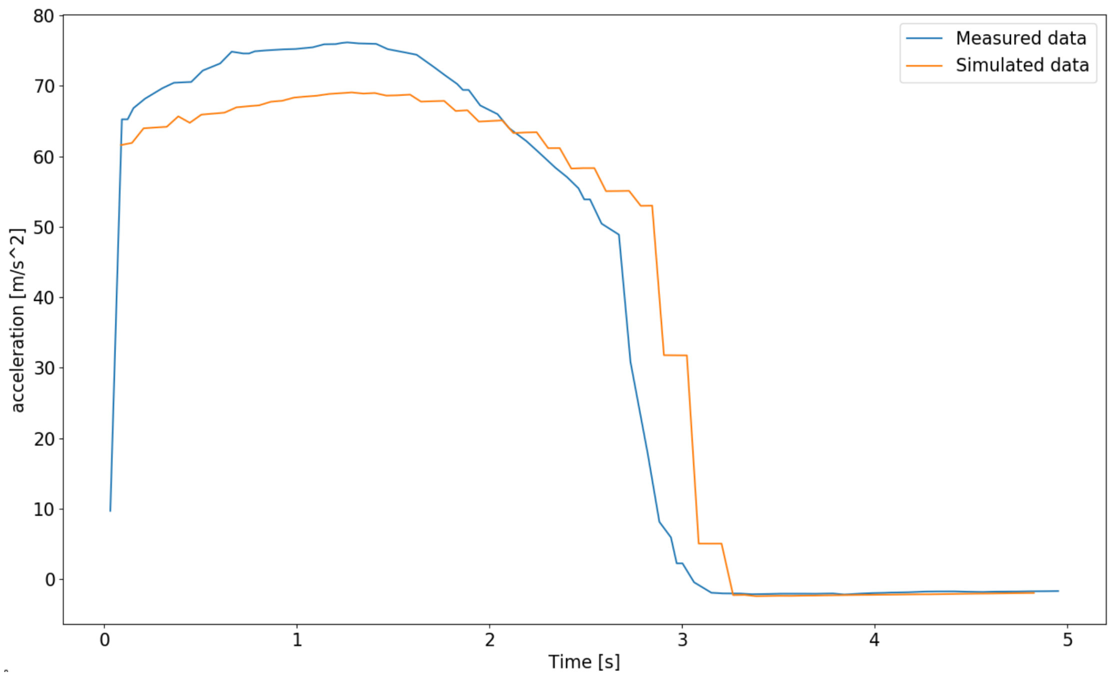


Figure 2.8: Simulated $\hat{\mathbf{z}}^b$ acceleration at 21Hz compared to the acceleration measured by the on-board IMU

The resulting altitude of the rocket over time is presented in figure 2.9. We see similar altitude at the apogee, though the flight profiles are slightly different, probably due to the different thrust curve of the rocket engine.

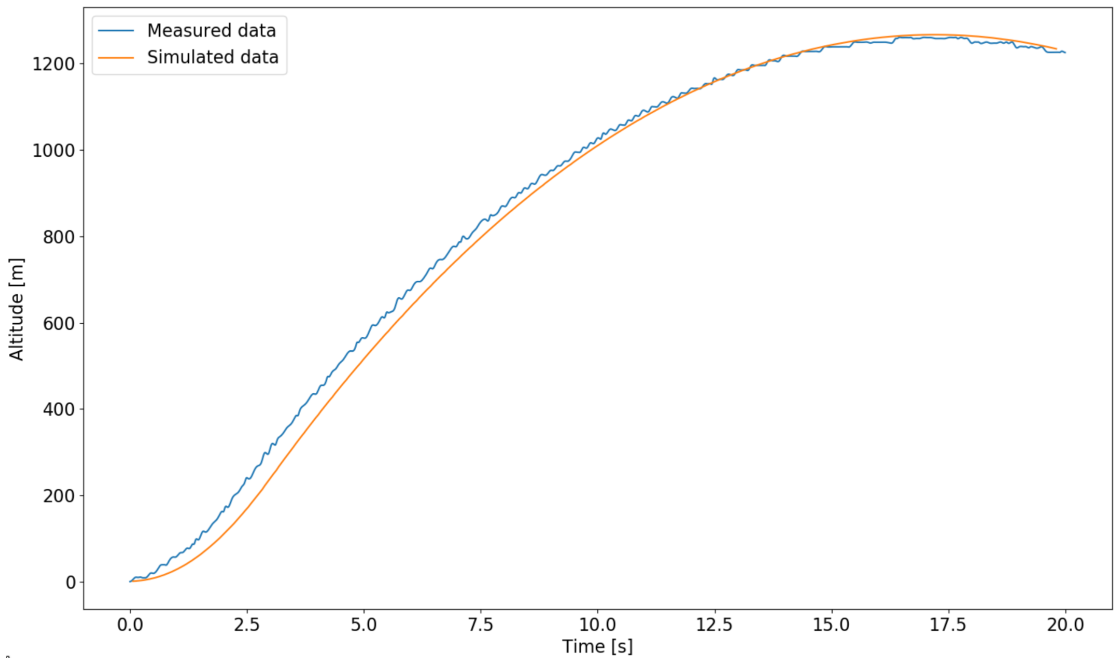


Figure 2.9: Simulated altitude compared to the barometric altitude measured by the on-board barometer

The last type of sensor to verify is the three axis gyroscope. The simulated sensor data and the measured data are compared in figure 2.10.

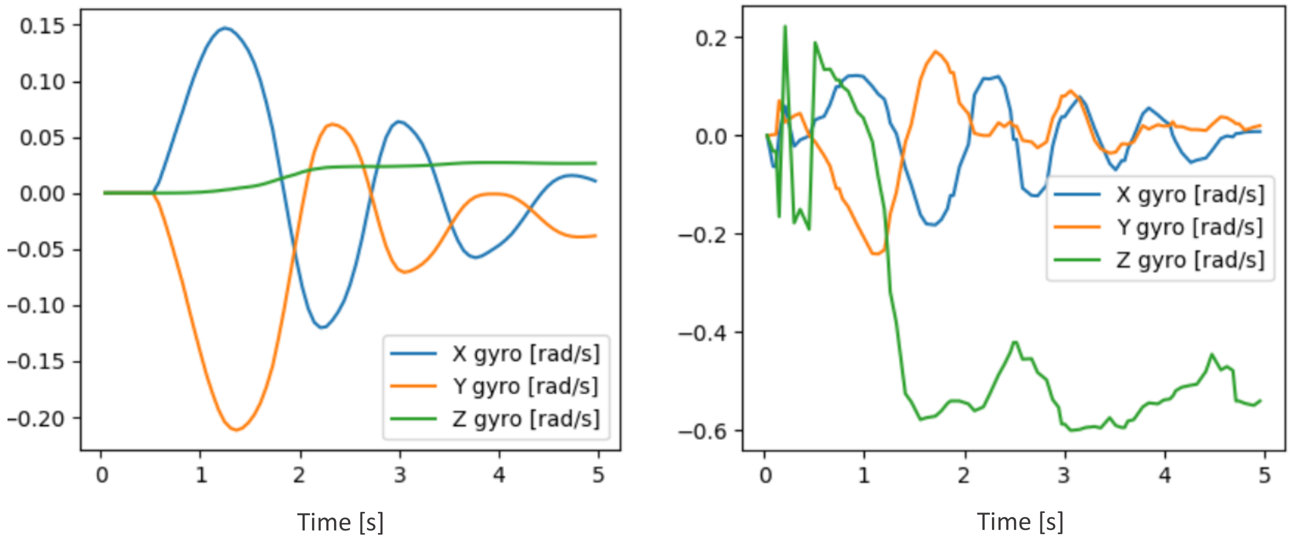


Figure 2.10: Simulated gyroscope data (left) compared to the angular rates measured by the on-board IMU

The major difference to be observed is the difference of roll angular rate ($\hat{\mathbf{z}}^b$ axis). From what was measured during the flight, it seems that slight tilting of the fins due to manufacture imperfections led to spinning of the rocket. This effect is well known and sometimes used to spin-stabilize sounding rockets [6]. However, this effect is not modelled in the simulator, as the fins are assumed to be perfectly straight. As discussed in section 2.1.3, it could be added in the perturbations modelling node in the future to get more realistic flight simulation.

The two other axes are more interesting, as they play an important role in the stability of the rocket. As we see in figure 2.10, the simulation is very similar to the flight data, with the oscillations in the $\hat{\mathbf{x}}^b$ and $\hat{\mathbf{y}}^b$ axes having close amplitude and frequencies.

Conclusion

In this chapter, the main features of the simulator have been presented, as well as some of the important tests that have been made to demonstrate its performance and reliability.

Though more complex to install and use than the previous Matlab simulator of the EPFL Rocket Team, or existing open-source simulator such as OpenRocket [7], it is also more adapted to the development of GNC projects due to its real-time capability and the possibility to easily connect it to external GNC algorithms and hardware.

Due to its modular architecture, it is easy to add new features or better models of the flight dynamics and perturbations. However, as it will be shown in the following chapters, the current version of this simulator is quite sufficient to develop GNC algorithms to flight-ready versions. It it thus highly advised to the following students who will be working on the Icarus project to continue using this simulator for their personal projects.

Chapter 3

Altitude control

3.1 Mission overview

3.1.1 Motivation

To win the Spaceport America Cup, the rocket has to reach precisely a predefined altitude at the apogee. Thus the strategy of the EPFL Rocket team is to have active control during the flight to compensate for simulations error and uncertainties on the environment and motor performance.
Previous projects called project Matterhorn [8] and project Eiger [9] relied on airbrakes which are aerodynamic surfaces that can be extended out of the rocket to increase its drag, and simple algorithms for navigation and control of these airbrakes. Unfortunately, in three years this strategy has never been successful, mainly due to lack of time and proper tools to test the GNC on the ground and its integration on the rocket. This year, the Bellalui 2 project still intends to actively control the altitude of its apogee using the throttling capability of its newly developed hybrid rocket engine to modulate the thrust during the flight.
In addition to being useful for the mission of project Bellalui 2, this project provides a simple use case to demonstrate the effectiveness of the simulator presented in chapter 2 for ground qualification, and will be the first in-flight demonstration of the software and hardware architecture presented in chapter 1.

3.1.2 Interface

Following the architecture defined in chapter 1, the GNC module is integrated on the avionic (AV) subsystem of the Bellalui 2 rocket. An overview of the rocket configuration is shown in figure

3.1, with the Avionic in the center of the rocket. Below it is the hybrid engine with its custom electronics to control the thrust of the engine.

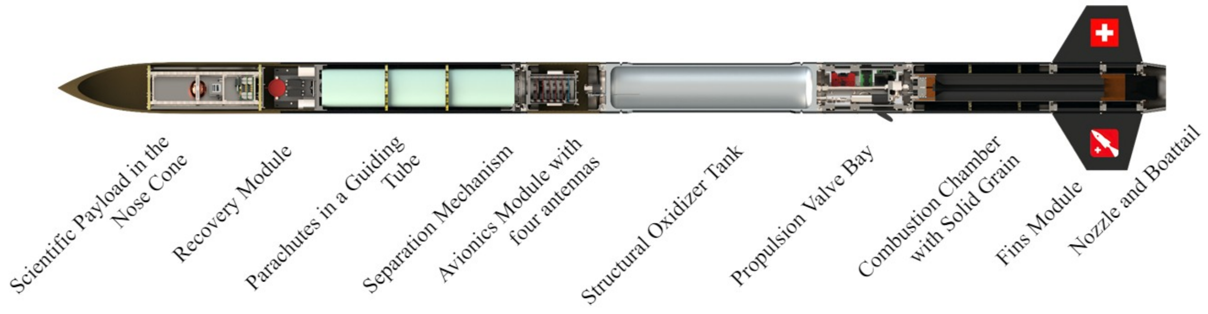

Scientific Payload in the Nose Cone
Recovery Module
Parachutes in a Guiding Tube
Separation Mechanism
Avionics Module with four antennas
Structural Oxidizer Tank
Propulsion Valve Bay
Combustion Chamber with Solid Grain
Fins Module
Nozzle and Boattail

Figure 3.1: Bellalui 2 rocket configuration

The avionics provides to the GNC barometric and IMU data. The IMU is a three axis accelerometer and gyroscope sensor, and the barometer provides directly the altitude above ground level, correcting for temperature variations.
To provide more reliable data, a pre-processing algorithm was developed by the avionics team to merge sensor data from four IMUs and four barometers before sending them to the GNC. Each sensor data is sent at a fixed frequency of 21Hz.
Then the propulsion's electronics manage the low-level control of the rocket engine, receiving high level thrust commands from the GNC, and sending back pressure and temperature data of the engine.

Finally, the main specifications of this rocket are presented on table 3.1. Note that the loaded mass and the burn time can be adjusted to reach the desired target apogee and maximum speed.

Specification	**Unit**	**Value**
Length	mm	3363
Cross-section diameter	mm	156.8
Dry Mass	kg	32.947
Max. loaded mass	kg	41.647
Thrust range	N	1500 to 3000
Burn time	s	11.5
Max speed	m/s	229
Targeted apogee	m	3048

Table 3.1: Bellalui 2 rocket specifications

3.2 GNC algorithms

In this section, the Navigation and Guidance algorithm used for the mission are presented.
The same notation is used as in the previous chapters, with the vectors in lowercase bold font, and the matrices in uppercase bold font. The same two reference frames are also used: the body frame (superscript b), attached to the rocket with the $\hat{\mathbf{z}}$ axis along its length, and the local-level frame (superscript l) , attached to the launch rail with the $\hat{\mathbf{z}}$ axis up and the $\hat{\mathbf{x}}$ axis pointing to the North.

3.2.1 Navigation

We use the standard approach [10] [11] of using an Inertial Navigation System (INS) in conjunction with a Kalman filter to estimate the full state of the rocket. A barometer is used to correct the vertical position and velocity errors from the direct integration of the IMU measurements. This method makes the Navigation quite flexible to be upgraded in case new sensors such as a GNSS or magnetometer would be integrated by the avionics team to increase the precision and stability of the state estimation.
The requirements to make the Navigation comply with the mission are presented in table 3.2.

ID	**Name**	**Description**
2021-LV-KF-OP-01	Sensor	The navigation shall use the available IMU (3DOF accelerometer and gyroscope) and barometer data from the avionics
2021-LV-KF-OP-02	Output	The Navigation shall estimate 3D position, velocity, attitude, and the mass of the rocket
2021-LV-KF-FCT-02	Frequency	The Navigation shall compute a new state estimation at a frequency of 100Hz
2021-LV-KF-VF-01	Testing	The Navigation shall be tested on simulation, with the avionics module on the ground, and during a test flight
2021-LV-KF-INT-01	Communication	The Navigation shall transmit the vertical velocity and position in real time to the avionics via the CAN bus

Table 3.2: Requirements for the Navigation algorithm

INS Model

By integrating the 3D acceleration and gyroscope data from the IMU, the complete 3D position, speed and orientation of the rocket can be estimated using dead reckoning [12]. The mass variation of the rocket due to propellant consumption is assumed to be proportional to the rocket thrust.
By noting $\mathbf{x}$ the state vector composed of the 3D position $\mathbf{p}$, velocity $\mathbf{v}$, attitude quaternion $\mathbf{q}$, angular rate $\mathbf{w}$ and mass m, we find $\dot{\mathbf{x}}$ the time derivative of $\mathbf{x}$ from Newton-Euler ordinary differential equation:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^l \\ \mathbf{v}^l \\ \mathbf{q} \\ \mathbf{q} \circ \mathbf{w}_{IMU} \circ \mathbf{q}^{-1} \\ m \end{bmatrix} \qquad \dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v}^l \\ \mathbf{q} \circ \mathbf{a}_{IMU} \circ \mathbf{q}^{-1} - \mathbf{g} \\ 0.5 * \mathbf{w} \circ \mathbf{q} \\ \vec{0} \\ \frac{T}{g_0 * Isp} \end{bmatrix} \tag{3.1}$$

with $\mathbf{a}_{IMU}$ and $\mathbf{w}_{IMU}$ respectively the acceleration and angular rate measurement from the IMU in body frame. All the states in $\mathbf{x}$ are in local-level frame, so the attitude quaternion $\mathbf{q}$ is used to transform the IMU measurements from body frame to the local-level frame. Noting the quaternion state $\mathbf{q} = (q_w,\ q_x,\ q_y,\ q_z\ )$ with q_w the real part and $(q_x,\ q_y,\ q_z\ )$ the vector part of the quaternion, we can transform any vector $\mathbf{v}^b$ from the body frame to the local-level frame $\mathbf{v}^l$ using [13]:

$$\mathbf{v}^l = \mathbf{q} \circ \mathbf{v}^b \circ \mathbf{q}^{-1} \tag{3.2}$$

We also have $\mathbf{g} = \begin{bmatrix} 0 & 0 & g_0 \end{bmatrix}^T$ the vector of gravity assumed to be constant and co-linear with the $\hat{\mathbf{z}}$ axis of our local-level frame. Finally we have the specific impulse of the motor Isp which is a parameter of the rocket engine that relates the mass variation with the thrust of the engine T. This thrust is measured during the flight with a pressure sensor located inside the combustion chamber.
To satisfy requirement *2021-LV-KF-FCT-02* (table 3.2), these equations needs to be solved to estimate the full state every 10ms. Due to its simplicity of implementation, the Runge-Kutta method of order four was chosen to solve equation 3.1 by numerical integration.

Kalman filter

The main issue with the simple INS model is that the position, velocity and attitude states will inevitably drift over time, as biases inside the IMU will be integrated too. To get a better estimation of the vertical position and velocity, Kalman filtering is used to fuse the barometric altitude measurement and the INS estimation.

The basic principle of Kalman filtering is to use a model of a system to predict its evolution, and measurements of the system to update our current estimation.
A simplified system is considered by extracting the sub-state $\mathbf{x}_{KF}$ from the full state $\mathbf{x}$ provided by the INS. This new system consists of only the vertical position and speed as they are the most important states to estimate for this mission, and the only one we can reliably correct with only the barometric measurement. The sub-state model is:

$$\mathbf{x}_{KF} = \begin{bmatrix} p_z \\ v_z \end{bmatrix} \qquad \dot{\mathbf{x}}_{KF} = \begin{bmatrix} v_z \\ 0 \end{bmatrix} = \mathbf{A} * \mathbf{x}_{KF} \tag{3.3}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{3.4}$$

With $\dot{\mathbf{x}}_{KF}$ the time derivative of our sub-state mode. This continuous-time state-space representation can be discretized exactly using Euler's method, as higher order terms are all equal to zero.

$$\mathbf{F}_k = e^{\mathbf{A}*\Delta t} = \mathbf{I} + \mathbf{A} * \Delta t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{3.5}$$

$$\mathbf{x}_{KF}(k+1) = \mathbf{F}_k * \mathbf{x}_{KF}(k) \tag{3.6}$$

$\mathbf{F}_k$ is the discrete-time state transition matrix, and Δt the time interval between two samples k and $(k+1)$. In our case, the prediction frequency is 100Hz by requirement, so $\Delta t = 10ms$.

Now that we have a model of our system, we need to relate the barometric measurement h_{baro} to our estimated state vector $\mathbf{x}_{KF}$ using the observation matrix $\mathbf{H}$:

$$h_{baro} = p_z + r_{baro} = \mathbf{H} * \mathbf{x}_{KF} + r_{baro} \tag{3.7}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix} \tag{3.8}$$

The barometric measurement h_{baro} directly measures the altitude above ground level, and we assume that white noise $r_{baro} \sim N(0, \sigma_{baro}^2)$ is added to the true altitude. The variance σ_{baro}^2 of the barometer was directly measured on static conditions.

The state estimation also has uncertainty, modeled with a normal distribution, and estimated with the covariance matrix $\mathbf{P}$. This matrix is estimated recursively, using the the discrete-time state transition matrix $\mathbf{F}_k$:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k * \mathbf{P}_{k-1|k-1} * \mathbf{F}_k^T + \mathbf{Q} \tag{3.9}$$

This estimation is initialized with $\mathbf{P}_0$, representing the uncertainty on the initial state x_{KF0}. Derivation of $\mathbf{P}_0$ is detailed in section 3.2.1. The process noise $\mathbf{Q}$ was set to $0.002 * \mathbf{P}_0$ after some tests with the simulator.

We now have a model of our system and of our measurement, and an estimation of their uncertainties.

From these, the optimal [14] state estimation is obtained by first computing the Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} * \mathbf{H}^T * (\mathbf{H} * \mathbf{P}_{k|k-1} * \mathbf{H}^T + r_{baro})^{-1} \tag{3.10}$$

and use it to combine the *a-priori* state prediction $\mathbf{x}_{KF_{k|k-1}}$ and the measurement h_{baro} to get the *a-posteriori* state $\mathbf{x}_{KF_{k|k}}$ and covariance matrix $\mathbf{P}_{k|k}$:

$$\mathbf{x}_{KF_{k|k}} = \mathbf{x}_{KF_{k|k-1}} + \mathbf{K}_k * (h_{baro} - p_z) \tag{3.11}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k * \mathbf{H}) * \mathbf{P}_{k|k-1} \tag{3.12}$$

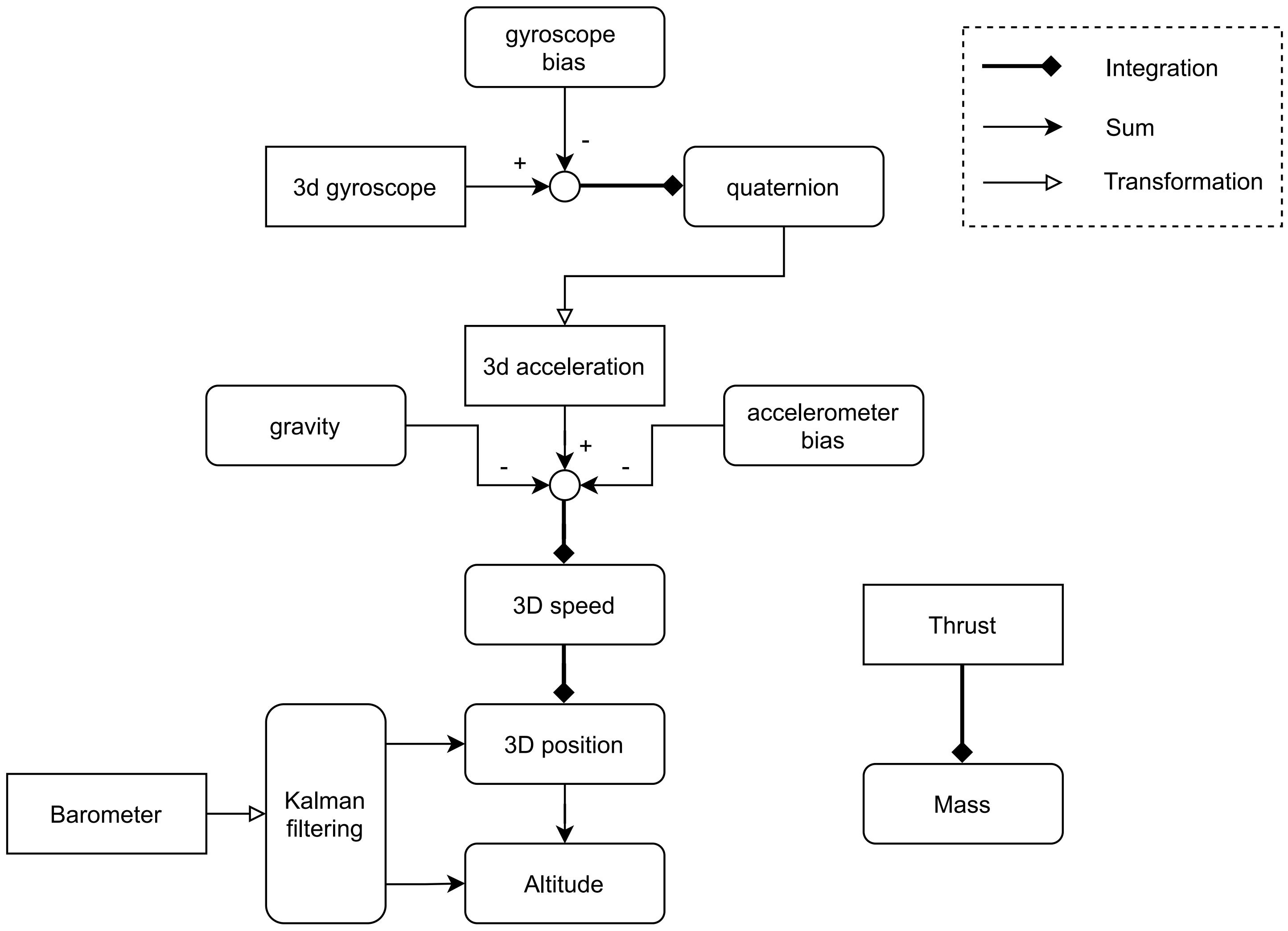


Figure 3.2: Complete navigation algorithm

Delay compensation

It was observed that the biggest source of errors in this model comes from delays in the inputs and outputs of the GNC.

For this mission, the input delay is mostly important at liftoff, as it creates uncertainty on the initial state. This initial error comes from the initial delay between the real liftoff time and the moment the GNC receives high acceleration signal that triggers the start of the navigation algorithm. This delay cannot be avoided, as starting the Navigation before liftoff would result in significant drift due to the long integration time, as it will be shown in section 3.3.2.

Due to high total acceleration of 50 m/s^2 to 62 m/s^2 at liftoff, this results in the rocket having already significant speed and altitude when the navigation starts, that we can approximate as:

$$\begin{cases} v_{z0} = a_0 * \tau \\ p_{z0} = \frac{1}{2} * a_0 * \tau^2 \end{cases} \tag{3.13}$$

with a_0 the initial acceleration and τ the initial delay resulting in the initial error in speed and position v_{z_0} and p_{z_0}.

To estimate this initial delay, we can first list all possible source of delays present at liftoff.

Delay type	Description	Value [ms]
Ignition delay	The time it takes for the engine combustion to create the high acceleration defined as liftoff trigger	4
Mechanical delay	The time it takes the acceleration created by the engine to reach the accelerometer (speed of sound in carbon fiber tube ≈ 10km/s [15])	0.1
Sensor delay	The times it takes the accelerometer to measure the high acceleration due to the sampling period	≤ 48
Communication delay	The delay to relay the IMU data from the AV to the Raspberry Pi	3
GNC startup delay	The time it takes for the different GNC nodes to generate a first state estimation	10 to 20

Table 3.3: Sequence of delays from real liftoff to GNC startup

As we can see in table 3.3, the biggest source of delay is the sampling period of the sensors, so the total delay can vary between 17.1ms and 75.1ms, and typical initial speed at Navigation startup will be between 0.86m/s and 4.66m/s. Using equation 3.3, we can thus initialize v_{z_0} and p_{z_0} with the mean delay and liftoff acceleration to have a reduced initial error and faster convergence of the Kalman filter correction.

With this initial delay, we can also have an estimation of the initial state covariance P_0. We model this delay variation as a Gaussian process with a mean of $46.1ms$ and a variance of $93.5ms^2$ to have a confidence interval of 99.7%. So $\tau \sim N(\mu, \sigma^2) = N(46.1, 93.5)$. Using formula 3.13, we can estimate the covariance matrix of the initial state to be $\mathbf{P}_0 = \begin{bmatrix} 0.00055 & 0.012 \\ 0.012 & 0.265 \end{bmatrix}$.

The output delay is the time interval between a GNC command and the execution of the real actuator. In this case, this delay is mostly important at the shutdown time, as during this time interval the rocket engine is still ignited, so the rocket will continue to accelerate which will result in the rocket overshooting due to the excess speed. As for the input delay, we can list all sequence of delays at shutdown.

Delay source	Description	Value [ms]
Navigation	Time delay to recompute a state estimation and apogee prediction at 100Hz	≤ 10
Communication	The delay to relay the command from the Raspberry Pi to the AV module	3
Motor control	The delay to send low level control to the valve actuator	≤ 50
Engine shut-down	The time for the actuator to fully close the valve and stop the combustion	100

Table 3.4: Sequence of delays from GNC command to engine shutdown

This mean output delay can then be used to have a better prediction of the real altitude and vertical speed at the engine shutdown time. To do this, simple Euler integration of the current acceleration measured by the IMU is used to shift these two states in time.

3.2.2 Guidance

To reach the required apogee, a Model Predictive Control (MPC) algorithm is developed to control the thrust of the engine during its burn time. A faster, lower level algorithm, is also added to send the engine shutdown command at the right time and reach the apogee on a ballistic trajectory. The Guidance algorithm has to satisfy some requirements:

Table 3.5: Requirements for the Guidance algorithm

ID	**Name**	**Description**
2021-TVC-TP-INT-01	Input	The Guidance shall use the state estimates by the Navigation algorithm and the targeted apogee as inputs
2021-TVC-TP-INT-02	Output	The Guidance shall compute a thrust level in the feasible range defined by propulsion
2021-TVC-TP-PHYS-01	Frequency	The optimal thrust command and trajectory shall be computed at least every 200ms
2021-TVC-TP-PHYS-02	Communication	The Guidance shall transmit the commanded thrust level in real time to the Propulsion's electronics via the CAN bus

MPC model

For this algorithm, a simplified model neglecting attitude dynamics is used.:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^l \\ \mathbf{v}^l \\ m \\ T^b \end{bmatrix} \qquad \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, \alpha) = \alpha * \begin{bmatrix} \mathbf{v}^l \\ \frac{cos(\theta) * T^b * \hat{\mathbf{z}} - \mathbf{d}}{m_{tot}} - g_0 \\ \frac{T}{g_0 * Isp} \\ \dot{T} \end{bmatrix} \qquad \mathbf{u} = \begin{bmatrix} \dot{T} \end{bmatrix} \tag{3.14}$$

The position $\mathbf{p}^l$ and velocity $\mathbf{v}^l$ states are in the local-level frame, noted with the superscript l. The thrust command T^b in body frame is added to the state vector, as its rate of change will be controlled by the input $\mathbf{u}$. Then the state dynamics are computed considering only the dominant forces: rocket thrust, gravity and drag (by order of importance). As the thrust is in body

frame, the angle to the vertical θ is computed before each MPC iteration to project the effective rocket thrust on the vertical axis. The 3D drag force is defined as $\mathbf{d} = C * \mathbf{v}^2$, with C the drag constant encompassing aerodynamic coefficient, frontal surface and air density: $C = 0.5 * \rho * S * C_d$, computed for each axis of the rocket. This model of the drag force is quite a simplification, as it should in theory take into account the external wind, and the effect of the angle of attack on the drag constant C. However it proved to be sufficient for this problem, as these effects are mostly important during the phases of low velocity and thus low drag.
The α coefficient is used to scale the state dynamics, and is thus a parameter of the optimal control problem, used by the solver to find the quickest trajectory from the current state up to apogee.
To find this parameter, we need to solve a minimum-time problem, whose general formulation is:

$$\underset{u(\cdot),x(\cdot)}{\text{minimize}} \int_{t_0}^{t_f} 1 \cdot d\tau \quad \text{subject to: } \forall \tau \in [t_0, t_f] : \begin{cases} \dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau), \mathbf{u}(\tau)) \\ \mathbf{x}(t_0) = \mathbf{x}_0 \\ \mathbf{x}(t_f) = \mathbf{x}_f \\ \mathbf{x}_{min} \leq \mathbf{x}(\tau) \leq \mathbf{x}_{max} \\ \mathbf{u}_{min} \leq \mathbf{u}(\tau) \leq \mathbf{u}_{max} \end{cases} \tag{3.15}$$

Which gives us the command $\mathbf{u}$ that will control our system starting from the initial state $\mathbf{x}_0$ to arrive to the final state $\mathbf{x}_f$ in the minimum time.
We can reformulate this problem using the scaling parameter α:

$$\underset{u(\cdot),x(\cdot),\alpha}{\text{minimize}} \int_{0}^{1} \alpha \cdot d\tau = \alpha \quad \text{subject to: } \forall \tau \in [0, 1] : \begin{cases} \dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau), \mathbf{u}(\tau), \alpha) = \alpha * f(\mathbf{x}(\tau), \mathbf{u}(\tau)) \\ \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{x}(1) = \mathbf{x}_f \\ \mathbf{x}_{min} \leq \mathbf{x}(\tau) \leq \mathbf{x}_{max} \\ \mathbf{u}_{min} \leq \mathbf{u}(\tau) \leq \mathbf{u}_{max} \end{cases} \tag{3.16}$$

Which is completely equivalent to equation 3.15 and allow to use directly α as a parameter to optimize in the minimum-time problem.

To solve this problem, the *PolyMPC* library [16] developed by the Automatic Control laboratory of EPFL is used, taking care of the discretization, interpolation and numerical optimization of the problem.

Firstly, the problem 3.16 is discretized using direct collocation method and Chebyshev polynomials:

$$\mathbf{x}(t) = \sum_{k=0}^{N} \mathbf{x}_k \phi_k(t) \qquad \mathbf{u}(t) = \sum_{k=0}^{N} \mathbf{u}_k \phi_k(t) \tag{3.17}$$

$$\begin{cases} \phi_k(t) = cos(k * \arccos(t)) \\ t_k = cos(\frac{\pi k}{N}) \end{cases} \tag{3.18}$$

which gives us a list of states $\mathbf{x}_k$ and control vectors $\mathbf{u}_k$ where the dynamics and constraints of 3.16 are enforced. To reach a good balance between accuracy and computation resources, two segments with each five collocation points were used.

By extending this list of state and control variables with the parameter α, we can now solve this problem using non-linear optimization problem theory. The method used in *PolyMPC* is the Sequential Programming Method [17] which solves the non-linear problem as a sequence of simpler quadratic problems. The advantage of this method is that with only a few iteration, the solution can be quite close to the optimal solution, so for real-time applications, a compromise can be made between accuracy of the solution and computation time. Using the *Eigen* implementation of *PolyMPC* we can solve this problem on the Raspberry Pi with a limit of ten SQP iterations, and a maximum computation time of 180ms.

Once the problem is solved, only the first command $\mathbf{u}_k(k = 0)$ is sent to the low-level thrust controller of the rocket engine.

Because this thrust command corresponds to the minimum-time trajectory, it should minimize fuel consumption. Indeed, for low altitude and low speed launches, the major loss is due to gravity drag, which is the amount of time the rocket engine is used to fight against gravity. So at this scales, the most efficient trajectory with respect to fuel consumption is to use the maximum amount of thrust for the shortest amount of time [18]. If in the future this algorithm would be used for high altitude launches, a term could be added in the cost function to directly maximize the final mass of fuel in order to take into account all types of loss, including drag loss that will play a more important role.

To make the OCP solution feasible, a few constraints on the state and control are added:

State	Constraint	Value
Propellant mass m	Should be positive, and less than the initial loaded mass of propellant	$[0\,;\, m_0]\, kg$
Thrust level T_b	Cannot exceed a maximum value due to engine design, and cannot be negative	$[0\,;3000]$ N
Altitude p_z	Should be above ground level, and below targeted apogee altitude to avoid overshoot	$[0\,;p_{zf}]$ m
Vertical speed	Should be positive (as we only consider the ascent phase) and below supersonic speed	$[0\,;330]$ m/s
Thrust rate	Should always be negative to avoid re-ignition strategy of the MPC during the flight, and with limited magnitude	[-3000 ; 0] N/s

Table 3.6: List of constraints for the Guidance OCP

Apogee prediction

The MPC presented in the previous section should in theory be sufficient to fully control the thrust of the engine during the flight. At one point the MPC should command zero thrust to the engine in order to reach the target apogee without any overshoot, which would shutdown completely the engine as we currently have no re-ignition capability during the flight. The rocket would then enter the coast phase of the flight, during which only gravity and drag forces subsist and slow down the rocket so that zero velocity is reached at the targeted apogee altitude.
The time when the rocket engine is shutdown is thus critical, so to add robustness, a lower level algorithm is implemented to verify the apogee based on the current state. If the MPC sends a shutdown commands to the engine (zero thrust) but the predicted apogee is lower than the target one, this command is ignored. Otherwise, if the predicted apogee starts to be above the targeted one, the MPC is bypassed, and a shutdown command is directly sent to the engine.
To predict the apogee, a simple analytical formula is used from [19]:

$$\begin{cases} h_f = h_0 + \frac{ln(1+\frac{v_0^2 * B}{g_0})}{2*B} \\ B = \frac{0.5*\rho*C_d*S}{m_{tot}} \end{cases} \tag{3.19}$$

with h_f the predicted apogee, h_0 the current altitude and v_0 the current vertical speed. The parameter B summarizes the effect of drag on the rocket speed with the air density ρ, the aerodynamic drag coefficient C_d, the frontal surface S and the rocket mass m_{tot}.

This simple algorithm alone is enough to control the apogee altitude, so the main reason to have the Guidance MPC is to have an algorithm that could easily be extended with other actuators such as airbrakes for a better control. Additionally, the MPC generates a smoother thrust curve compared to the abrupt shutdown of the apogee prediction algorithm, which reduces the problem of output delay (see 3.2.1). For project Icarus, having this MPC gives us a way to qualify its performance during a test flight, as this algorithm is very similar to the Guidance that will be implemented on the complete TVC rocket.
Unfortunately, as the propulsion team of Bellalui could not develop in time the low level thrust controller of their engine, Bellalui's rocket currently has no throttling capabilities. Thus for the next flight, the Guidance commands will only be logged for research purposes, and not be used to control the engine.

3.3 Implementation and qualification

The Navigation and Guidance algorithms have been implemented following the architecture described in chapter 1, with each algorithm running on a separate C++ ROS node and exchanging data asynchronously. The following sections describe the implementation and test of these algorithm in simulation and on the hardware.

3.3.1 Simulation

The first step in this development was to develop the GNC on the computer and test it on a simulated environment. A separate package called "bellalui_gnc" was created, hosting mainly

Guidance and Navigation, as well as a small node called *av_interface* to interface the input/output of the package with either the simulation or the real hardware.

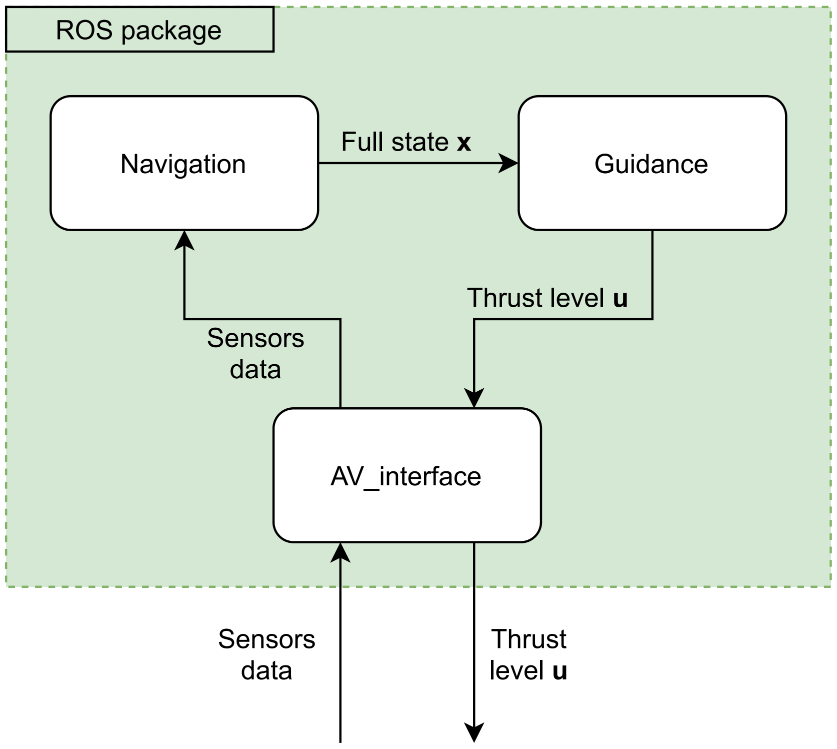


Figure 3.3: Simplified block diagram of the GNC ROS package used for Bellalui 2

The simulation was setup to send data every 48ms, mimicking the real avionics. To have a faster state refresh rate, the Navigation numerically integrate the IMU data at 100Hz, and only perform the correction step of the Kalman filter whenever new sensor data is available. The Guidance uses this state to compute the optimal trajectory every 200ms and sends it to the interface node. To reduce delay, this node is synchronized with input and output data using callback functions.

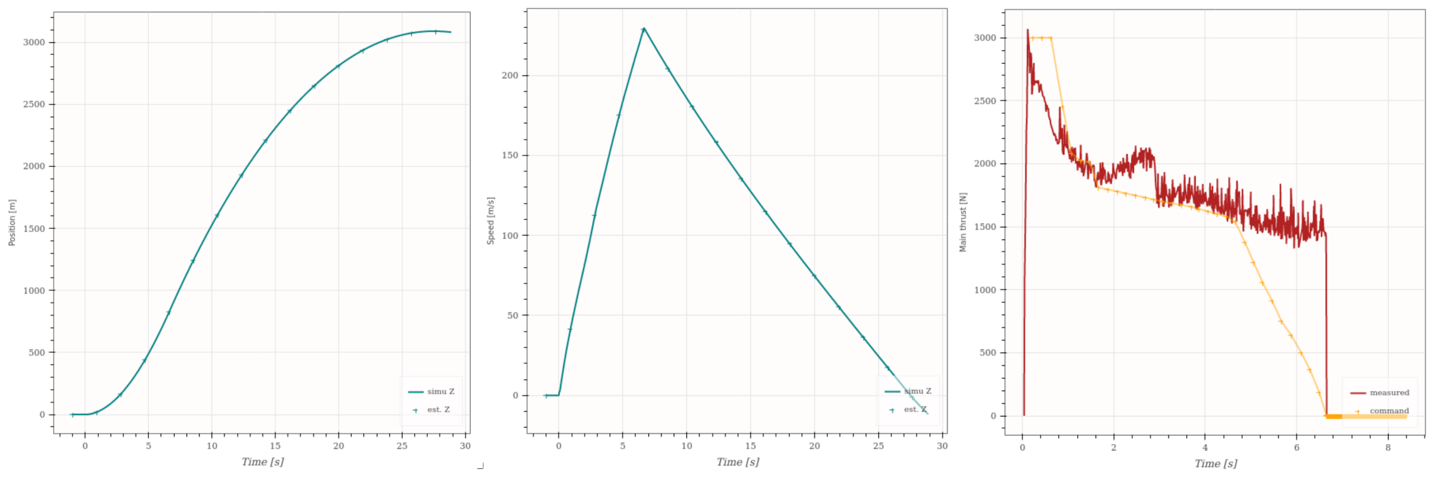


Figure 3.4: Altitude (left), Vertical speed (middle) and Thrust (right). The cross markers correspond to the navigation estimation and the commanded thrust level, and the plain line are the simulation data

The result of a typical simulation is presented on figure 3.4 for a targeted apogee of 3048m and a final apogee of 3088m, which corresponds to a precision of 1.3%. We can see the precision of the Navigation (cross markers) which matches well the simulation for the vertical position and velocity. Here the thrust command (yellow line) of the Guidance is not used to control the flight, so the rocket engine is at full throttle all the time (red curve). Nevertheless, we can see the Guidance thrust level is always trying to be at full throttle to minimize fuel consumption at the beginning of the burn, then steadily converges to zero thrust at the same time the apogee prediction algorithm shutdown the engine, which we see on the red curve as the abrupt vertical line at 6.7s. Note that here the Guidance thrust command is better than the full throttle, as a high thrust right before engine shutdown will result in additional speed due to the rocket continuing to accelerate during the delay between shutdown command and effective engine shutdown.

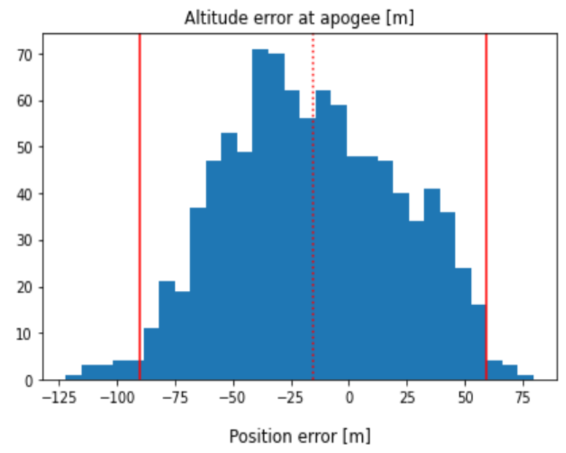
Monte Carlo simulation

To assess the performance of this GNC over all environmental conditions and flight profiles, a setup to do batches of simulation was developed on the simulator. External perturbations and internal sensors errors were varied randomly between each simulation to see their effect on the flight performance. A summary of the varied parameters is presented on table 3.7.

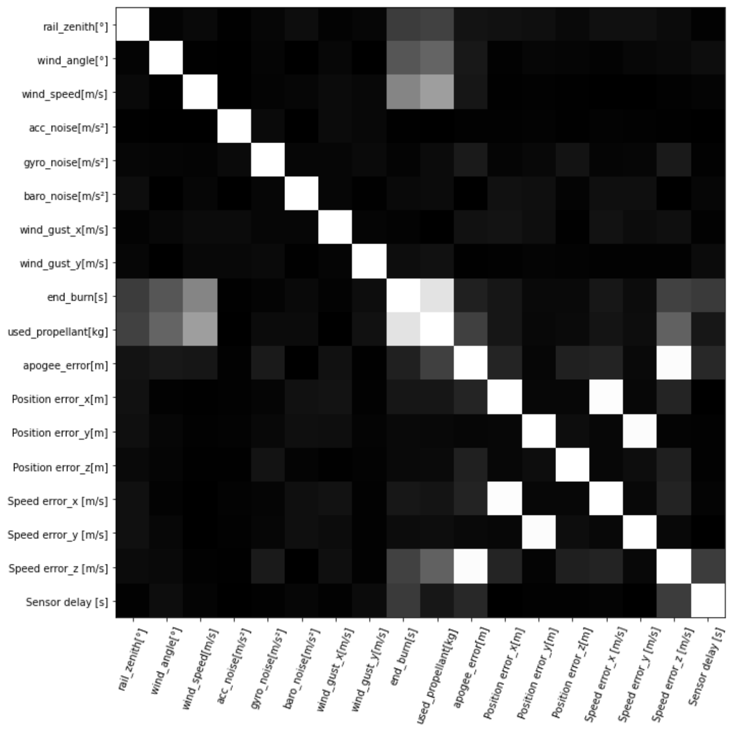
Table 3.7: Monte Carlo parameters

Parameter	**Description**	**Range**
Rail zenith	Initial angle from vertical	[0 ; 10] °
Rail azimuth	Initial angle from North. Also defines wind direction	[0 ; 180] °
Wind speed	Speed of the constant wind	[0 ; 15] m/s
Wind gust speed (X and Y)	Speed of the wind gust added to unstabilize the rocket	[0 ; 10] m/s
Accelerometer noise	Standard deviation of the Gaussian noise added to the true acceleration	[0 ; 0.2]m/s^2
Gyroscope noise	Standard deviation of the Gaussian noise added to the true angular rate	[0 ; 0.05] rad/s
Barometer noise	Standard deviation of the Gaussian noise added to the true altitude	[0 ; 1] m
Sensor period	Time period between each new sensor data	[40 ; 56] ms

The current setup run each simulation in real time and sequentially, which takes 46s per simulation. This allow to do around 1000 simulations to have a good dataset. The results of the last batch of simulation are presented on figures 3.5a and 3.5b. The mean error is -15m and 95% of the flights have a final apogee error between -90m and 59m. The worst flight had an error of -122m, which is still a precision of 4% relative to the targeted 3048m.
One very useful result of such analysis is the Pearson coefficient matrix shown in figure 3.5b which allow to trace back the source of the apogee errors by showing its relation with the other simulation parameters. High correlation is shown on the grayscale image as close to white.



(a) Histogram of altitude error at apogee, for a target apogee of 3048m. Two sigma interval is shown with plain red lines



(b) Pearson correlation between simulations parameters and simulation results

The most interesting line to look at is the apogee error line, which shows a high correlation with the vertical speed error at engine shutdown. This correlation can easily be explained as this velocity estimation is the main parameter used on the apogee prediction equation (3.19). Basically, if the speed estimation error is too important, this equation will predict a too high apogee as this additional speed will be integrated over the whole coast phase. This result in the engine being shutdown too early. From equation 3.19 we find a relation of around 16m of apogee error per m/s of vertical speed estimation error. This number corresponds roughly to the duration of the coast phase, and was also confirmed statistically from the Monte Carlo dataset.

It is after this observation that the Navigation was extended to the full model presented in section 3.2.1 with the barometer correcting both altitude and vertical speed. The result of this modification is shown on figure 3.6, where we see that without the Kalman filter correction, the initial speed error due to the initial delay is kept throughout the flight, whereas with the Kalman filter, this error is corrected in around 1s. With the delay compensation, the error is corrected in 0.2s due to the lower initial error.

This example is a good illustration on the effectiveness of the Monte Carlo simulation to highlight small errors that can be hard to see on simple simulations, due to the large velocities and accelerations.

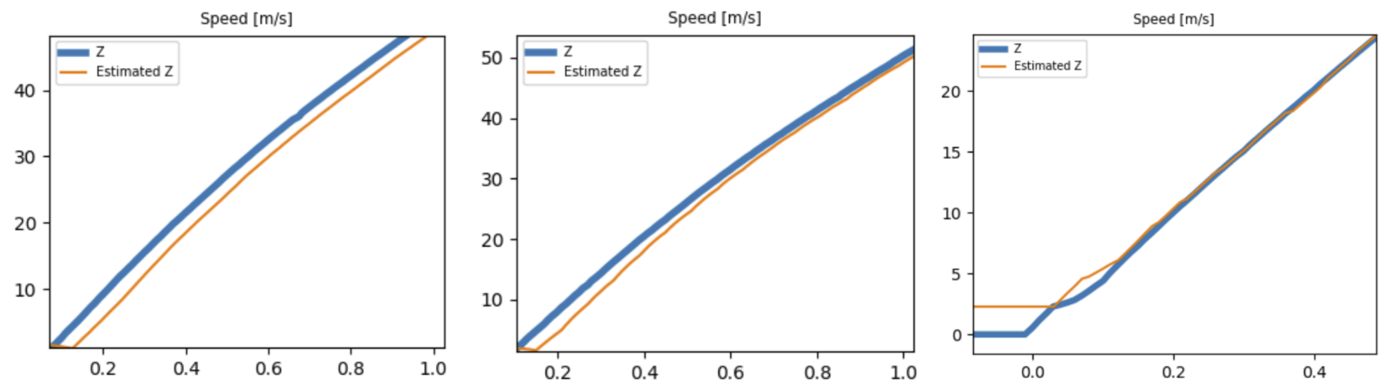


Figure 3.6: Vertical velocity in [m/s] as a function of time in seconds. Without (left), with (middle) kalman filter correction, and with delay compensation (right).

We can do the same analysis on the B parameter to find around 4.3m error at apogee for every percent of error on this parameter estimation. This error mainly come from mismodeling of the rocket parameter like its mass or aerodynamic coefficient, and thus requires extra attention to be corrected before each flight to adapt to the different rocket models and configurations.

The last important result from this Monte Carlo simulation is the absence of correlation between apogee error and external perturbations and configuration such as sensor noise or rail angle, which suggest overall a good robustness of the GNC.

3.3.2 Hardware test

Once the GNC algorithms is validated with the simulation, it is necessary to test it thoroughly on the real hardware. Thanks to the modular architecture used in this project, exactly the same ROS package can be used on the on-board computer, which in our case is a Raspberry Pi 4. The only difference is with the interface node which is configured to communicate directly to the avionic instead of the computer simulation.

Processor-in-the-loop setup

To simulate a flight, the simulator is connected directly to the avionics's GNC module. The simulation sends sensor data to the GNC hardware module, and receives the GNC commands to update the simulation.

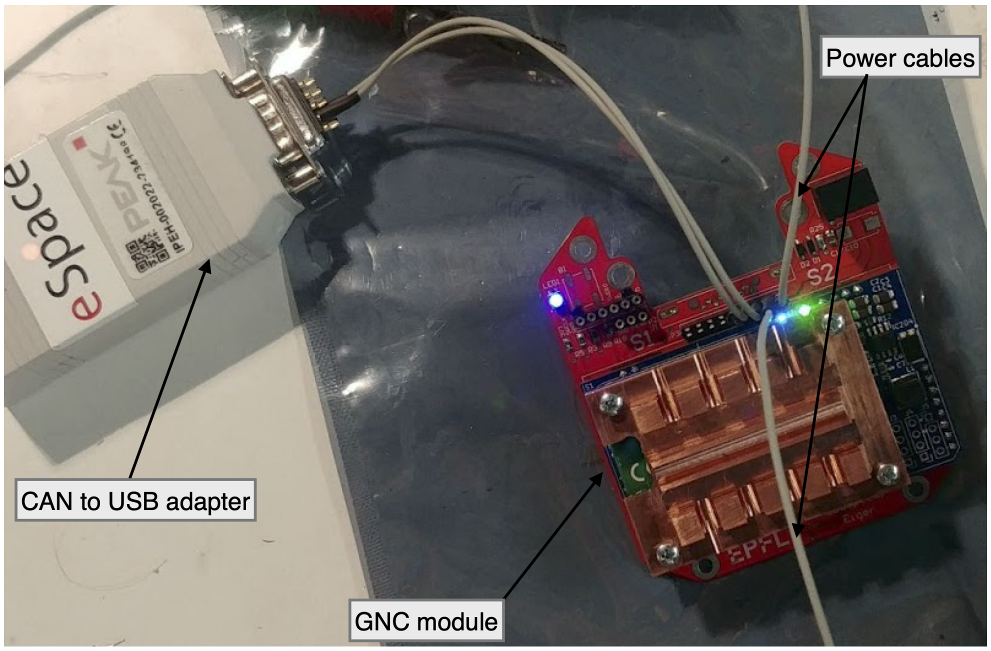


Figure 3.7: Processor in the loop hardware setup

As we see in figure 3.7, the interfaces to perform this test are minimal, and are the same as those used in-flight: CAN bus and power cables.

Using this setup, critical delay issues due the UART communication speed were addressed, and the processing power of the Raspberry Pi could be verified to respect the 200ms period of MPC computation. After these modifications, a final monte carlo simulation was done with the PIL setup, which showed very similar performance as with the simulation.

Sensor validation

Because the simulator is not able to capture all imperfections of the real sensors used on the rocket, it is important to test the Navigation with the real sensor data on the ground.

To do this, the complete AV module was used with the Navigation estimating the rocket state on simple cases. First on a static and horizontal position, then by rotating the module in all axes before replacing it back to the initial position, and finally by moving it to a known position. The complete procedure can be found in Appendix A.

The result of a 30 seconds long static test is presented on figure 3.8.

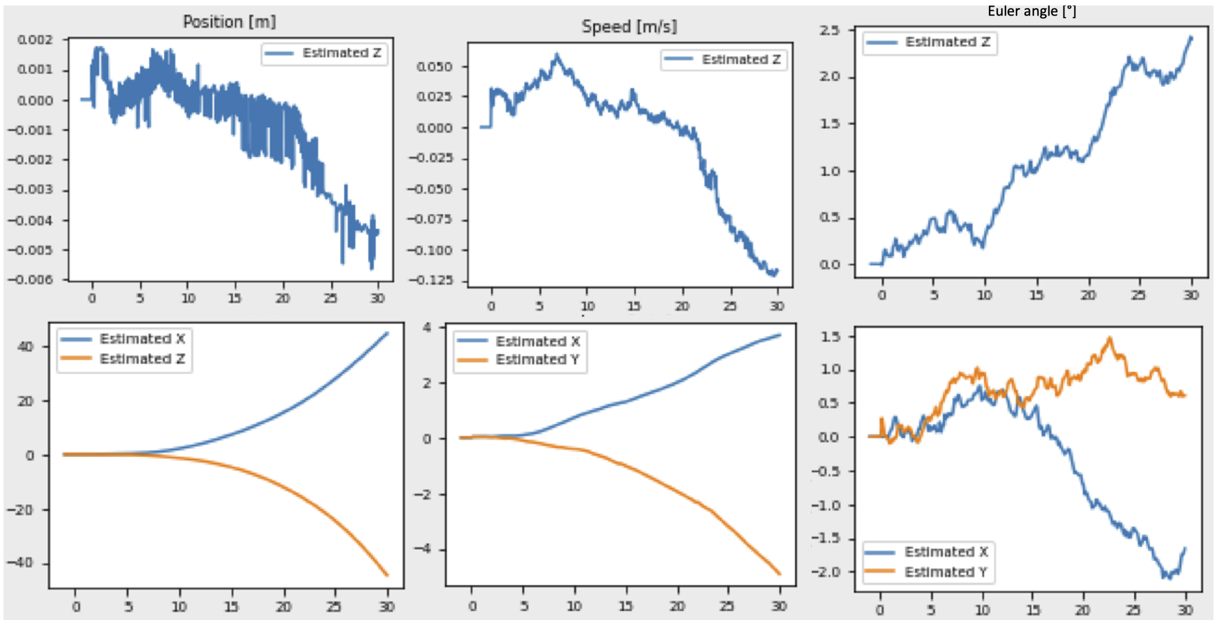
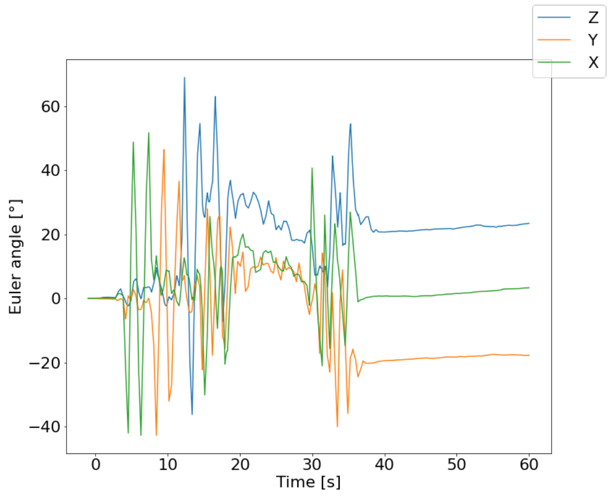


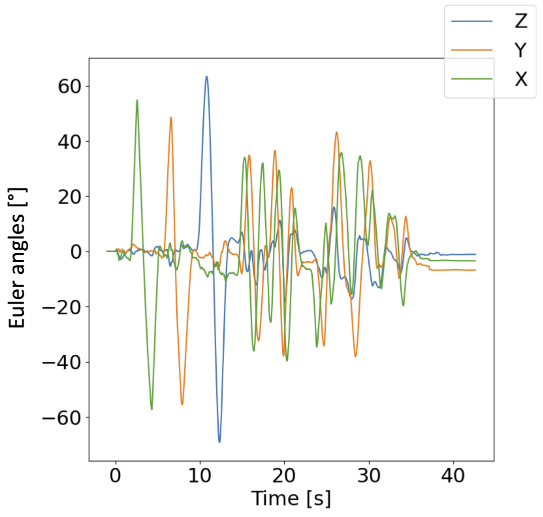
Figure 3.8: State estimation during a static, horizontal test of 30s. The left column is the position, middle column is velocity, and right column is the attitude (Euler angles)

We see that by estimating the gyroscope bias and removing them before integration allow to have quite stable attitude estimation for the duration of a flight. Thanks to the barometer, the altitude and vertical speed are also quite stable. The biggest error is on the horizontal velocity and position estimation, which due to accelerometer errors drifts significantly during the flight, rendering them currently unusable for future position control.

The result of the dynamic test also showed interesting result. Due to a low sampling frequency of the gyroscope, the Navigation algorithm could not track properly the rotation of the AV module, which resulted in a constant error at the end of the test (figure 3.9a). This was corrected by increasing the sampling frequency to its current maximum of 20.8Hz, as seen in figure 3.9b.



(a) Euler angles estimation with 3Hz sampling frequency



(b) Euler angles estimation with 20.8Hz sampling frequency

Conclusion

The GNC for Bellalui 2 is now fully ready to be used in a real flight, thanks to the thorough testing performed on the ground.

Unfortunately, due to the Coronavirus pandemic and bad weather, the test launch was postponed to the end of July, so the result of this launch could not be presented in this report. However, the current results presented in section 3.3 already show the benefits of using the tools and methods developed during this thesis to catch bugs and errors without launching the rocket.

The current results also show the difficulty to obtain high apogee precision with the current set of sensors and actuators. It is highly advised to the EPFL rocket team to add new sensors such as magnetometer or GNSS to improve the current state estimation. Developing the low level thrust throttling or adding airbrakes would also give more control to the GNC. These modifications would greatly increase the robustness and precision of the GNC for altitude control, and will be probably mandatory for the future Thrust Vector Control of the rocket.

Chapter 4

Trust Vectoring Control

This chapter describes the first steps in the development of a Thrust Vector Control (TVC) system for the EPFL Rocket Team. This project being quite large and complex, it will require the work of students over the span of several semesters. The achievement of this semester team are thus mainly on first designs and proof-of-concepts that should serve as a basis for the following semesters.

4.1 Motivation

Thrust Vector Control (TVC) is a method to control the direction of a rocket engine thrust to control the attitude of the rocket. In turns, this defines the rocket direction of flight which is used to control its position and velocity. Compared to the GNC developed in chapter 3, a TVC system allows the complete control of the state, and is thus widely used in today commercial rocket launchers [20].
Developing a TVC system and its associated GNC is thus a very interesting project for the EPFL Rocket Team in its objective of developing state of the art technologies in the field of rocket launchers. It is also the logical next step that will allow more ambitious missions such as sub-orbitals flights which cannot rely on passive aerodynamic stabilization due to the low air density at high altitude. For a similar reason, Thrust Vectoring is also a key-technology for enabling control during low-speed phases of the missions, such as for vertical take-off without launch rail [21] or for precise controlled landing [22].

Besides these long term objectives, the current mission profiles at low altitude and speed could already benefits from having more control over the vehicle during its thrusting phase.
The first benefits would be a better control of the rocket attitude, as the current passive stabiliza-

tion is highly dependent on the external wind perturbations (see [3] on chapter 3) which is difficult to predict or measure. As shown in figure 4.1, this perturbation results in the rocket having usually an angle of 5° to 10° with respect to the vertical during the thrust phase which leads to unwanted horizontal accelerations.

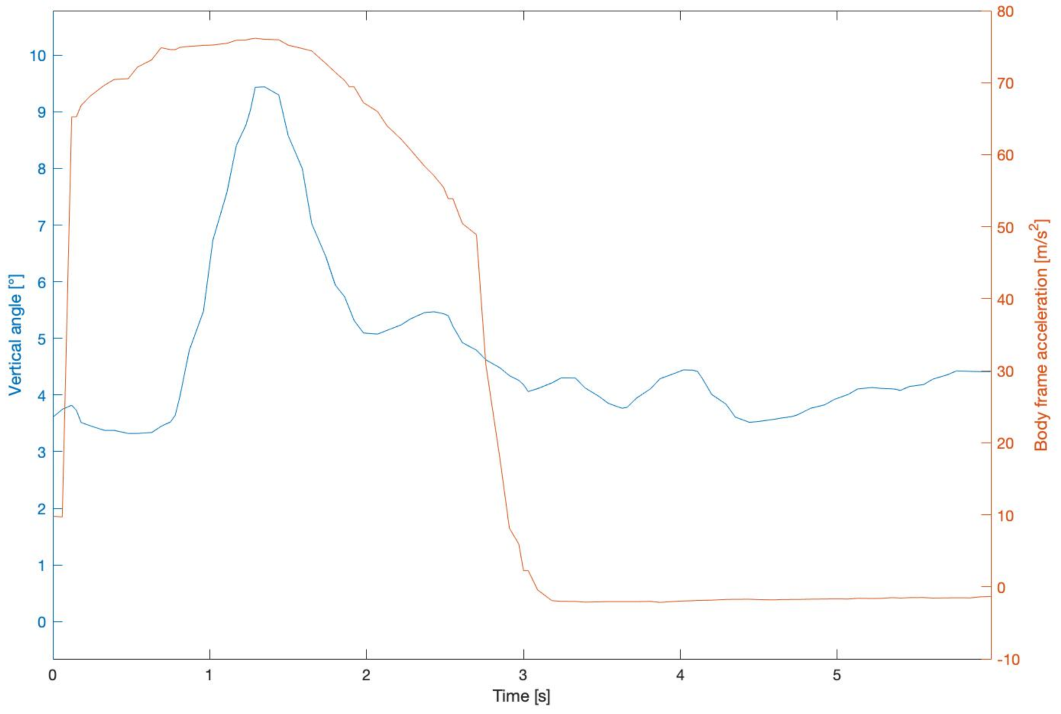


Figure 4.1: Estimated angle with respect to the vertical (blue) and $\hat{\mathbf{z}}^b$ acceleration (orange) as a function of time. These data comes from the launch of the rocket Bellalui 1 in 2020

To compensate this effect, the rocket is usually accelerated vertically at high speed on a launch rail, so that the wind speed becomes significantly lower than the rocket speed. However, this results in additional inefficiencies due to the launch rail friction with the rocket launch lug, which are small plastic parts attached to the rocket and sliding inside the launch rail. Thus, the second advantage of the TVC would be to reduce the drag and friction due to the launch rail and launch lugs. The fins size and its associated drag could also be reduced by up to 30% [23], as the stabilization during the critical liftoff phase would be ensured by the TVC system. Overall these two points would improve the efficiency of the rocket, by increasing the thrust effectiveness, and reduce the drag effect.

Another possible benefits of the TVC is on the control of the horizontal position and velocity of the rocket. Because of the vertical angle during the thrust phase, the rocket accelerates horizontally, and thus drifts significantly away from the launch pad. Because of the uncertainty on wind speed and direction, the guidelines for a safe rocket launch requires a secured launch zone diameter of at least half of the altitude apogee [24], which proved to be very difficult to find in Switzerland. With the help of the TVC system, this vertical angle could be minimized, and the horizontal position drift could be compensated by controlling the flight direction of the rocket with the TVC.

4.2 Guidance and Control of a TVC system

This section first explains the general theory of controlling the attitude, position and velocity states of a rocket using Thrust Vector Control. Then a first implementation of a guidance and control algorithm using Model Predictive Control will be used to demonstrate the capability of a TVC system to achieve the objectives presented in the previous section.

4.2.1 Control strategy

Because the controlled force of a TVC system is applied at the bottom of a rocket, far away from the center of mass which is usually located near the middle of the rocket, the effect of the TVC is mainly to create torques around the $\hat{\mathbf{x}}^b$ and $\hat{\mathbf{y}}^b$ axes of the rocket to control its attitude, rather than directly using the 3D force vector to control the speed and position of the rocket.
The usual strategy [25] to control the position and velocity of a rocket is shown in figure 4.2.

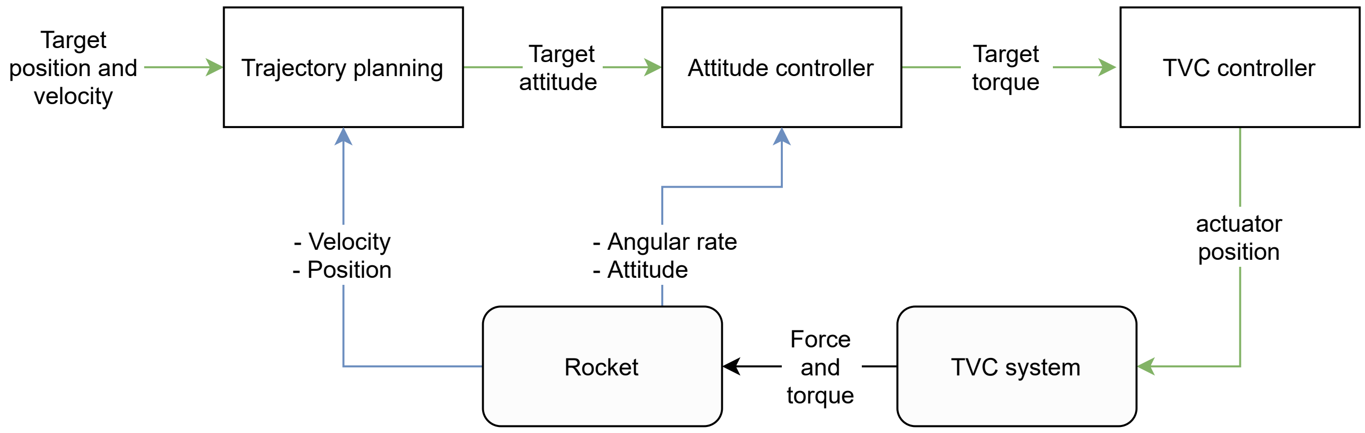


Figure 4.2: Black diagram of the sequence of control used for a typical rocket GNC

To reach the target position and velocity, the rocket is rotated in the direction of the flight by the attitude controller. If we neglect aerodynamic effects, the rocket can be stabilized in any orientation as long the vector of force $\mathbf{f}_c^b$ passes through the center of mass of the rocket.
Indeed, if we write down the sum of torque on the rocket, we have:

$$\sum \boldsymbol{\tau}^l = \vec{\mathbf{0}} \times \mathbf{g}^l + \mathbf{R}^b * (\mathbf{d}_{CM} \times \mathbf{f}_c^b) = \vec{\mathbf{0}} \tag{4.1}$$

The vector $\mathbf{g}^l$ is the force of gravity applied at the center of mass, and thus producing no torque on the rocket. $\mathbf{R}^b$ is the rotation matrix derived from the attitude of the rocket, and $\mathbf{d}_{CM}$ is the vector from the nozzle of the rocket engine (where the thrust force is applied) to the center of mass of the rocket. So to reach equilibrium, we only need:

$$\mathbf{d}_{CM} \times \mathbf{f}_c^b = \vec{\mathbf{0}} \Rightarrow \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} \times \begin{bmatrix} f_{cx} \\ f_{cy} \\ f_{cz} \end{bmatrix} = \begin{bmatrix} d_y f_{cz} - d_z f_{cy} \\ d_z f_{cx} - d_x f_{cz} \\ d_x f_{cy} - d_y f_{cx} \end{bmatrix} = \vec{\mathbf{0}} \tag{4.2}$$

However, the rockets developed by the EPFL Rocket Team have only one rocket engine, which is aligned with the center of the rocket. The center of mass is also placed by design at the center of the rocket, so $d_x = d_y = 0$, and:

$$\boldsymbol{\tau}_c^b = \mathbf{d}_{CM} \times \mathbf{f}_c^b = \begin{bmatrix} -d_z f_{cy} \\ d_z f_{cx} \\ 0 \end{bmatrix} = \vec{\mathbf{0}} \Rightarrow f_{cx} = f_{cy} = 0 \tag{4.3}$$

To compensate external perturbation torques such as those due to aerodynamic forces, or if we need to rotate the rocket ($\sum \boldsymbol{\tau}^l \neq \vec{\mathbf{0}}$) to a new attitude, we can use the TVC system to generate lateral body forces f_{cx} and f_{cy}, so that the vector of thrust force is not passing through the center of mass.

Once the rocket is stabilized to a new attitude, the projection of the thrust force to the local-level frame is used to control the position and velocity of the rocket, similarly to how a quadcopter control its position.

This strategy is usually implemented as a cascade of controllers, as shown in figure 4.2. In the next section, we will explore the possibility to use MPC to find this strategy directly from the model of the system, without having to detail it explicitly in the controller. This could allow a better balance between the objective of stabilizing the attitude and angular velocity, with the objectives of position and velocity tracking.

4.2.2 Guidance, Navigation and Control

Control formulation

A Model Predictive Controller is implemented to control the rocket with the TVC model presented in section 4.2. This controller has no model of the actuators, and is thus working only with high level forces and torques commands in body frame:

$$\mathbf{u} = \begin{bmatrix} f_{cx} \\ f_{cy} \\ f_{cz} \\ \tau_z \end{bmatrix} = \begin{bmatrix} u[0] \\ u[1] \\ u[2] \\ u[3] \end{bmatrix} \tag{4.4}$$

The attitude of the rocket can be completely controlled with the torque vector $\boldsymbol{\tau}_c^b$ derived from the input $\mathbf{u}$:

$$\boldsymbol{\tau}_c^b = \begin{bmatrix} -d_z f_{cy} \\ d_z f_{cx} \\ \tau_z \end{bmatrix} = \begin{bmatrix} -d_z u[1] \\ d_z u[0] \\ u[3] \end{bmatrix} \tag{4.5}$$

And the controlled $\mathbf{f}_c^b$ force acting on the rocket is:

$$\mathbf{f}_c^b = \begin{bmatrix} f_{cx} \\ f_{cy} \\ f_{cz} \end{bmatrix} = \begin{bmatrix} u[0] \\ u[1] \\ u[2] \end{bmatrix} \tag{4.6}$$

These controlled forces and torque are used to control the state vector $\mathbf{x}$, defined as:

$$\mathbf{x} = \begin{bmatrix} \mathbf{p}^l & \mathbf{v}^l & \mathbf{q} & \mathbf{w}^l & m_p \end{bmatrix}^T \tag{4.7}$$

with $\mathbf{p}^l$ and $\mathbf{v}^l$ respectively the 3D position and velocity of the rocket in local level-frame, $\mathbf{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}$ the quaternion representing the attitude of the rocket, $\mathbf{w}^l$ its angular rate in local-level frame, and m_p is the mass of propellant.

The dynamic of this state vector is computed using the ordinary differential equation of a rigid body in 3D space, with varying mass. First, the sum of the forces and torques acting on the rocket are computed. An approximation of the equations presented in section 2.1.1 is used to reduce computations, so only the thrust of the rocket, its weight, and the drag force are considered. First the weight force is computed using a constant gravitational acceleration of $g_0 = 9.81 m/s^2$:

$$\mathbf{w}(m_p)^l = -m_{tot} * g_0 * \hat{\mathbf{z}}^l = -(m_{dry} + m_p) * g_0 * \hat{\mathbf{z}}^l \tag{4.8}$$

The thrust of the rocket is simply the transformation of the body frame controlled force $\mathbf{f}_c^b$ to the local level-frame, using the quaternion state $\mathbf{q}$:

$$\mathbf{f}_c^l = \mathbf{q} \circ \mathbf{f}_c^b \circ \mathbf{q}^{-1} \tag{4.9}$$

Finally, the drag force is approximated as colinear with the rocket $\hat{\mathbf{z}}^b$ axis, and thus produces no torque:

$$\mathbf{d}^l = \mathbf{q} \circ \begin{bmatrix} 0 \\ 0 \\ 0.5 * C_d * \rho * S * v_z^2 \end{bmatrix} \circ \mathbf{q}^{-1} \tag{4.10}$$

with C_d the drag coefficient of the rocket, ρ the air density and S the frontal surface of the rocket which are constant parameters. The advantage of this simple drag model is that does not require angle of attack estimation, as the EPFL Rocket team as no means to measure it. For the same reason, the lift force is neglected. Thus these approximation only holds for small variation of the rocket's attitude, so that the angle of attack is kept relatively small during the flight.

The sum of forces f_{tot}^l and torques $\boldsymbol{\tau}_{tot}^l$ can then be computed as:

$$\begin{cases} \mathbf{f}_{tot}^l = \mathbf{w}(m_p)^l + \mathbf{f}_c^l + \mathbf{d}^l \\ \boldsymbol{\tau}_{tot}^l = \mathbf{q} \circ \boldsymbol{\tau}_c^b \circ \mathbf{q}^{-1} \end{cases} \tag{4.11}$$

and used to find $\dot{\mathbf{x}}$ the time derivative of the state $\mathbf{x}$:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} \mathbf{v}^l \\ \frac{\mathbf{f}^l_{tot}}{m_p + m_{dry}} \\ 0.5 * \mathbf{w}^l \circ \mathbf{q} \\ \mathbf{q} \circ (\boldsymbol{\tau}^b_c * \mathbf{I}^{b^{-1}}) \circ \mathbf{q}^{-1} \\ \frac{\|\mathbf{f}^l_c\|}{g_0 * Isp} \end{bmatrix} \tag{4.12}$$

with m_{dry} the mass of the rocket without propellant, and Isp the specific impulse of the rocket engine which relates the total thrust $\|\mathbf{f}^l_c\|$ to the propellant mass variation $\|\mathbf{f}^l_c\| = \dot{m}_p * (g_0 * Isp)$. The matrix of inertia $\mathbf{I}^b$ is used to compute the angular acceleration first in body frame, then is transformed to the local-level frame.

Using this system, we formulate the following MPC problem:

$$\begin{aligned} &\underset{u(\cdot),x(\cdot)}{\text{minimize}} \int_0^{t_f} [\mathbf{x}_e(\tau)^T \mathbf{Q} \mathbf{x}_e(\tau) + \mathbf{u}_e(\tau)^T \mathbf{R} \mathbf{u}_e(\tau) + (cos(\theta) - 1)^2 * q_\theta] \cdot d\tau + \mathbf{x}_e(t_f)^T \mathbf{Q}_f \mathbf{x}_e(t_f) \\ &\text{subject to: } \forall \tau \in [0, t_f] : \begin{cases} \dot{\mathbf{x}}(\tau) = f(\mathbf{x}(\tau), \mathbf{u}(\tau)), \ \mathbf{x}(0) = \mathbf{x}_0 \\ \mathbf{x}_e(\tau) = \mathbf{x}(\tau) - \mathbf{x}_t \\ \mathbf{u}_e(\tau) = \mathbf{u}(\tau) - \mathbf{u}_t \\ cos(\theta) = q_w^2 - q_x^2 - q_y^2 + q_z^2 \\ \mathbf{x}_{min} \leq \mathbf{x}(\tau) \leq \mathbf{x}_{max} \\ \mathbf{u}_{min} \leq \mathbf{u}(\tau) \leq \mathbf{u}_{max} \end{cases} \end{aligned} \tag{4.13}$$

which minimizes with a quadratic cost function the distance of the state $\mathbf{x}$ to the target state $\mathbf{x}_t$, and the input $\mathbf{u}$ to the target input $\mathbf{u}_t$ defined by the guidance algorithm:

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{p}^l_t \\ \mathbf{v}^l_t \\ \mathbf{q}_t \\ \vec{\mathbf{0}} \\ m_{p_t} \end{bmatrix} \qquad \mathbf{u}_t = \begin{bmatrix} 0 \\ 0 \\ f_{t_z} \\ 0 \end{bmatrix} \tag{4.14}$$

The guidance algorithm is the same formulation and implementation as the one described in section 3.2.2. It computes the optimal trajectory from the current state to the apogee, which is used to define the target position $\mathbf{p}_t^l$, velocity $\mathbf{v}_t^l$ and propellant mass m_{p_t} that the control algorithm should reach within it horizon time t_f. As the guidance does not consider rotations in its model, the control algorithm simply minimizes the weighted norm of the angular rate and distance to the vertical orientation defined by $\mathbf{q}_t = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$. In practice, the target quaternion $\mathbf{q}_t$ is not directly used in the cost function, but rather the angle θ between the $\hat{\mathbf{z}}^b$ and the $\hat{\mathbf{z}}^l$ axes is minimized by computing the cosine of this angle from the quaternion state.
To help the control algorithm, the optimal thrust f_{t_z} computed by the guidance is also provided. The other inputs are targeted to zero to help stabilize the system and reduce propellant consumption.

Implementation and Results

The Model Predictive Controller is implemented using the *PolyMPC* library in the same manner as it was described in section 3.2.2. Using the *Eigen* implementation for maximum performances, the problem is solved with a limit of one SQP iteration so that the computation time is kept under 50ms when solved on a laptop in a simulation environment. This problem is then solved every 50ms, and as suggested by Raphaël Linsen during his semester project [26], this computation time is compensated by propagating in time the current state of the rocket by 50ms before solving the problem.
Another implementation detail which helps to solve the MPC problem faster is to scale the input and state vector so that their numerical values have similar orders of magnitude [27]. The input vector is thus scaled to have its values between -1 and 1, with -1 corresponding to the minimum force or torque, and +1 corresponding to the maximum. The state vector has only its position and velocity vector scaled to be in units of kilometers rather than meters, so they have magnitude of 10^{-2} to 10^{0}, similarly to the other states.

To test the GNC, a new package called *generic_gnc* was developed and connected to the simulator developed in chapter 2. The Guidance and Navigation algorithm are the same as those used in chapter 3.
First the attitude stabilization was tested for low altitude and speed types of flight, and without the

guidance algorithm. Only the weight q_θ and the matrix $\mathbf{R}$ have non-zero value in equation 4.13. The matrices $\mathbf{Q}$ and $\mathbf{Q}_f$ are both null matrices so that the control algorithm only tries to keep the rocket vertical.

The result of this test are shown in figure 4.3. We see that the control algorithm compensates the initial angle of the rocket in around 3 seconds by using a force f_{cx} to create a torque around the $\hat{\mathbf{y}}^b$ axis.
A residual angle of around 1.5° is present, probably due to an equilibrium between the weight on the input $\mathbf{R}$ and on the objective of minimizing the vertical angle θ with the weight q_θ.

$$\mathbf{R} = \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \end{bmatrix} \qquad q_\theta = 10^4$$

The weight on the inputs was necessary to avoid oscillations of the inputs and the rocket.

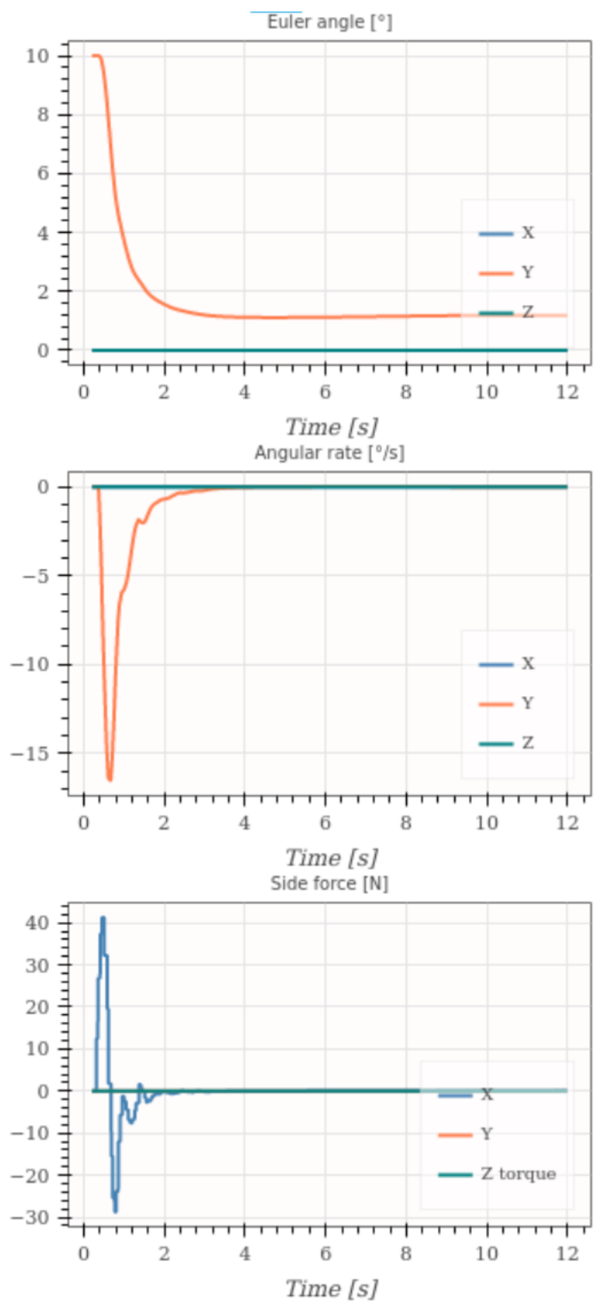
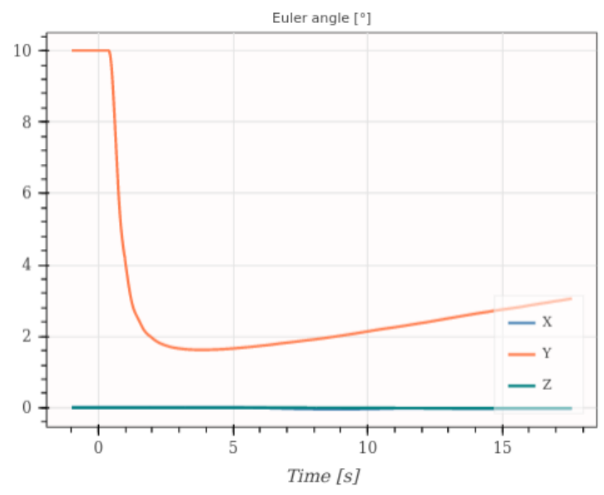


Figure 4.3: Euler angle (top), angular rate (middle) and controlled lateral forces (bottom)

This simulation uses a rocket with fins sized to have the center of pressure very close to the center of mass, and with no external wind, so that there is very little perturbations due to the lift force.
Thus, the next the same simulation is run with normal sized fins to have passive stabilization, and an external wind of 5m/s. The result is shown in figure 4.4, where we see a similar behaviour, with the initial angle being corrected in around 3 seconds, but this time the increasing speed of the rocket creates a new equilibrium point for the vertical angle,

which thus rises over time.



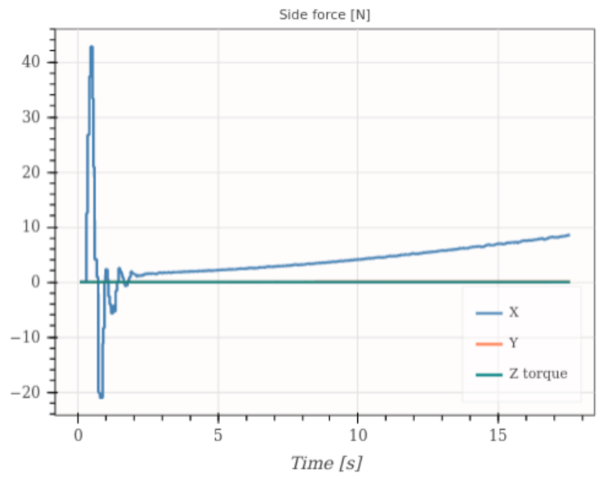


Figure 4.4: Euler angles (left) and controlled lateral forces (right) with a wind of 5 m/s

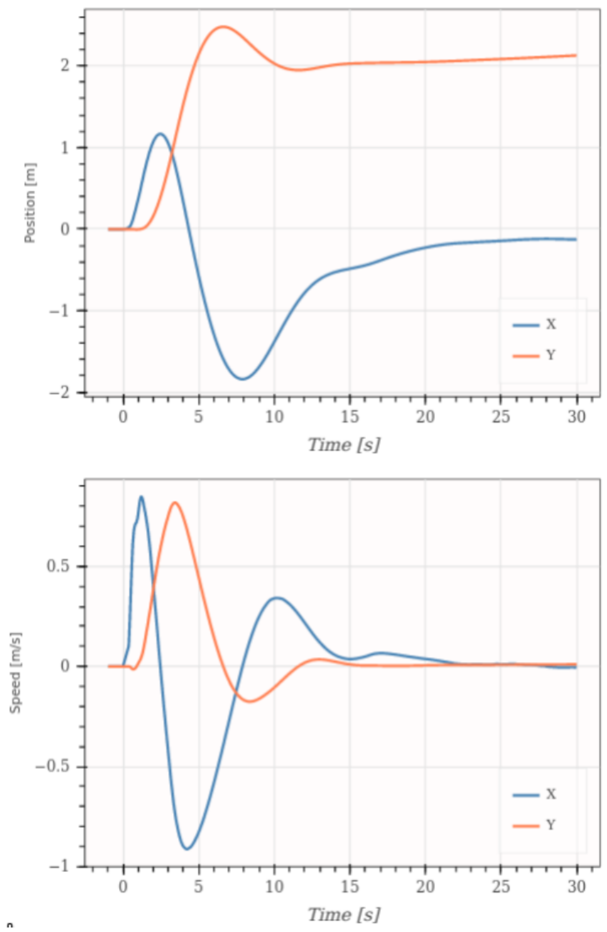
Better results could be obtained by modelling the lift force, or by adding an estimation of the unknown perturbations to directly compensate them, instead of compensating the resulting error. This was confirmed by the flight tests done with the drone, which estimates the unknown force and torque perturbation to achieve more accurate tracking [26].

The last unit test is on the position and velocity control. Because of the initial angle of 10°, the rocket accelerate along the $\hat{\mathbf{x}}^l$ axis and thus drifts away from the launch pad. The controller also has a constant target of 2 meters in $\hat{\mathbf{y}}^l$, so it has to simultaneously stabilize the rocket vertically, while tracking a 2D horizontal position.

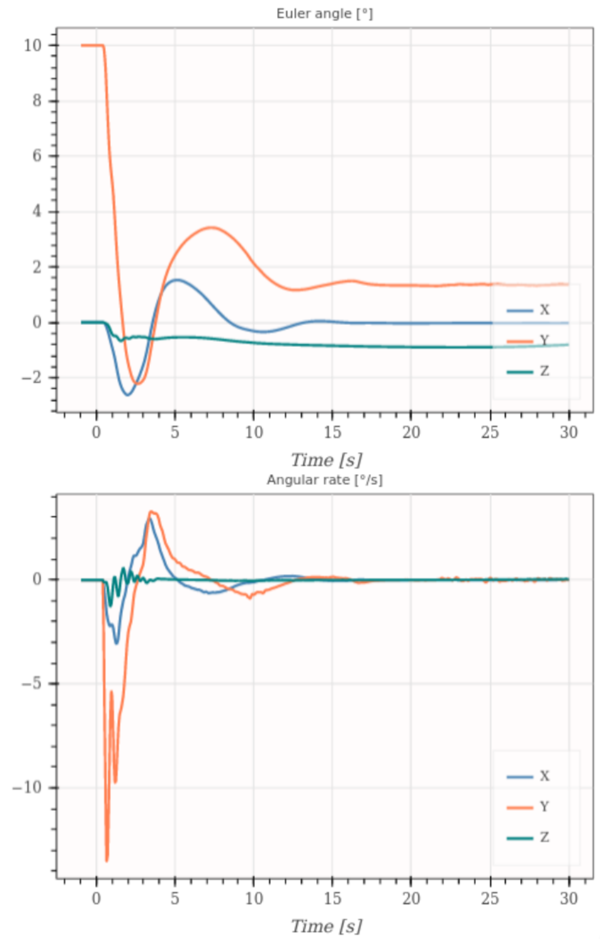
For this test, the weight matrices $\mathbf{Q}$ and $\mathbf{Q}_f$ are added to balance between tracking the error in attitude and the error in position.

$$\mathbf{Q}_{14*14} = \mathbf{Q}_{f\,14*14} = \begin{bmatrix} 50 & 0 & 0 & 0 & 0 & \\ 0 & 50 & 0 & 0 & 0 & \vdots \\ 0 & 0 & 0 & 0 & 0 & \mathbf{0} \\ 0 & 0 & 0 & 0.5 & 0 & \vdots \\ 0 & 0 & 0 & 0 & 0.5 & \\ & \dots & \mathbf{0} & \dots & & \mathbf{0} \end{bmatrix} * 10^3$$

As seen in figure 4.5a, the controller is able to reach the target position and stabilize around it at low velocity. While maneuvering, it is also able to keep the rocket's attitude close to the vertical, as we see in figure 4.5b. Small errors in position tracking seems to be due to the drift of the state estimation from the Navigation algorithm.



(a) Horizontal position (top) and velocity (bottom)



(b) Euler angles (top) and angular velocity (bottom)

Figure 4.5: Simultaneous tracking of the position and attitude of the rocket, with a wind of 5m/s

The last test is a simulation of a typical flight that could be done in Switzerland, targeting an altitude of 2000 meters, and using our latest iteration of a large scale rocket called Bellalui-2 (see table 3.1).

For reference, a first simulation is done without TVC. A 9 meters launch rail is used to accelerate the rocket at liftoff, and four fins of 17 centimeters span are placed at the bottom of the rocket to stabilize it passively. Thus this configuration corresponds to a typical rocket launch that we can do with our current technology.

The engine thrust is kept at its maximum during the whole flight (no throttling), and the low-level altitude control algorithm presented in section 3.2.2 is used to stop the engine at the right time to reach the targeted altitude of 2000 meters. The results of this flight are shown in figure 4.6.

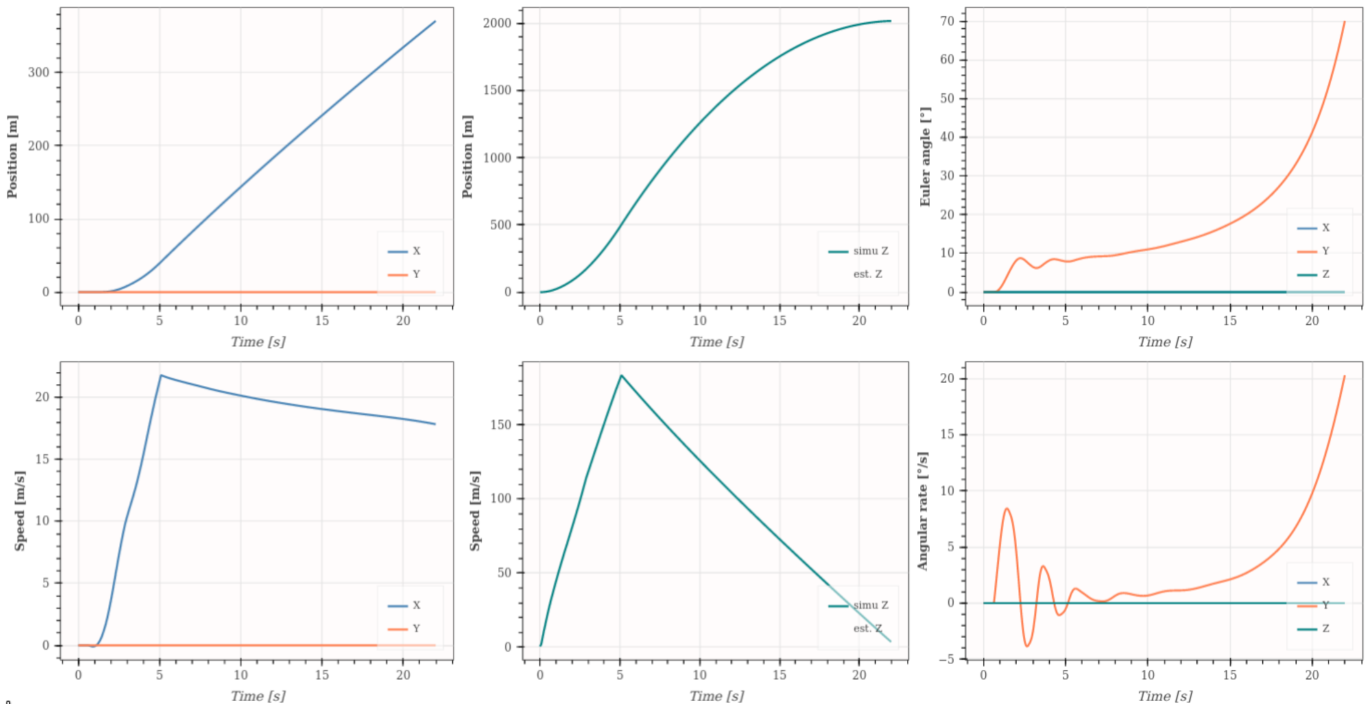


Figure 4.6: Reference flight without TVC, with a constant wind speed of 5m/s

The motor is shutdown 5 seconds after liftoff, and the rocket reaches an apogee of 2020 meters, 22 seconds after liftoff.

The rocket is launched vertically, but because of the wind, it stabilizes with the fins at around 10° during most of the flight. As usual, the rocket goes horizontal at apogee due to the constant horizontal wind speed.

Another result of this simulation is the horizontal position drift of 370 meters away from the launchpad at apogee, and the final horizontal speed of 18 m/s.

Next the same simulation is run with the TVC activated. The launch-rail is not used, and the fin size is reduced to 10 centimeters of span so that there is almost no passive stabilization due to the lift force. Thus the controller has full control on the rocket trajectory during the thrust phase.

Similarly to the previous test, the thrust phase last 5.1 seconds, and an apogee of 2020 meters is reached after 22 seconds of flight. The result is shown in figure 4.7.

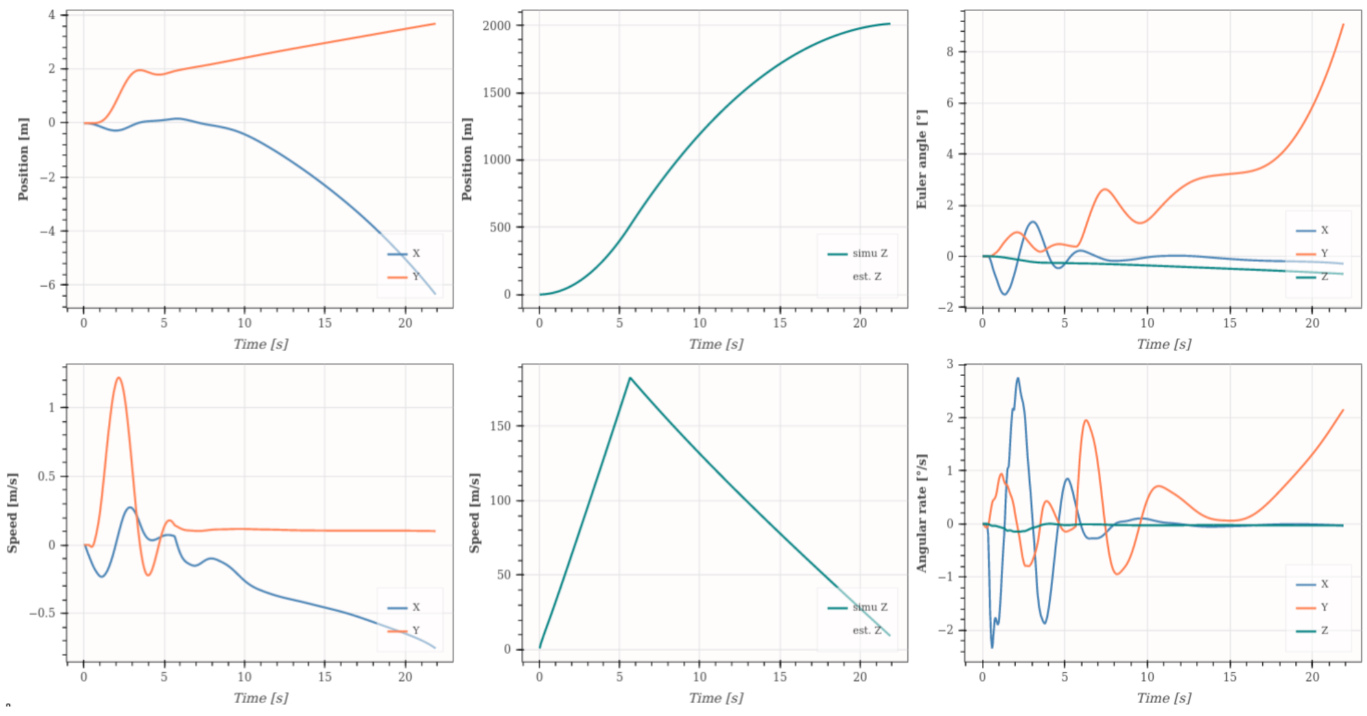


Figure 4.7: Reference flight with TVC activated, with a constant wind speed of 5m/s

The most notable difference is the low Euler angles of less than 2° during the 5.1 seconds of TVC control. When the TVC is deactivated at the rocket engine shutdown, the rocket is at its maximum speed, so the very low passive stabilization is enough to keep the rocket close to the vertical.
The second important result is the small horizontal velocity and position. At apogee, the rocket has only drifted by 7.5 meters away from the launchpad, compared to the 370 meters drift without TVC. The norm of the horizontal velocity is only 0.8 m/s, which highly reduces the stress on the parachute system deployed at apogee, compared to the usual 18m/s observed on the previous simulation without TVC.

A summary of these two flight is presented in table 4.1.

Parameter	Without TVC	With TVC
Burn time [s]	5	5.1
Max speed [m/s]	185	185
Apogee error [%]	1	1
Mean vertical angle during thrust [°]	6.4	1.0
Mean vertical angle during ascent [°]	23.8	3.9
Horizontal position drift at apogee [m]	370	7.5
Horizontal speed at apogee [m/s]	18	0.8

Table 4.1: Simulation comparison for a typical high-power flight, with and without TVC

4.2.3 Conclusion

In this section, the general theory to control the complete state of a rocket using Thrust Vector Control was presented. Then a Model Predictive Controller was developed and integrated with a navigation and a guidance algorithm to be tested in a Software-In-the-Loop configuration.
The results show the effectiveness of the TVC to control the complete position, velocity and attitude of the rocket, which could allow more efficient trajectory tracking, reduced launch area due to a lower position drift, and safer parachute deployment due to the lower horizontal speed at apogee.

These results are thus a first step in developing a Thrust Vector Control system for the rockets of the EPFL Rocket Team, which would give more control on the launch trajectory, and give the possibility to start more complex mission such as high altitude launch or propulsive landing.

4.3 Flight model development

4.3.1 Technology trade-off

The model presented in the previous section is a generic model which does not consider any real implementation of the TVC system. This allow it to be adaptable to various TVC system and vehicles.

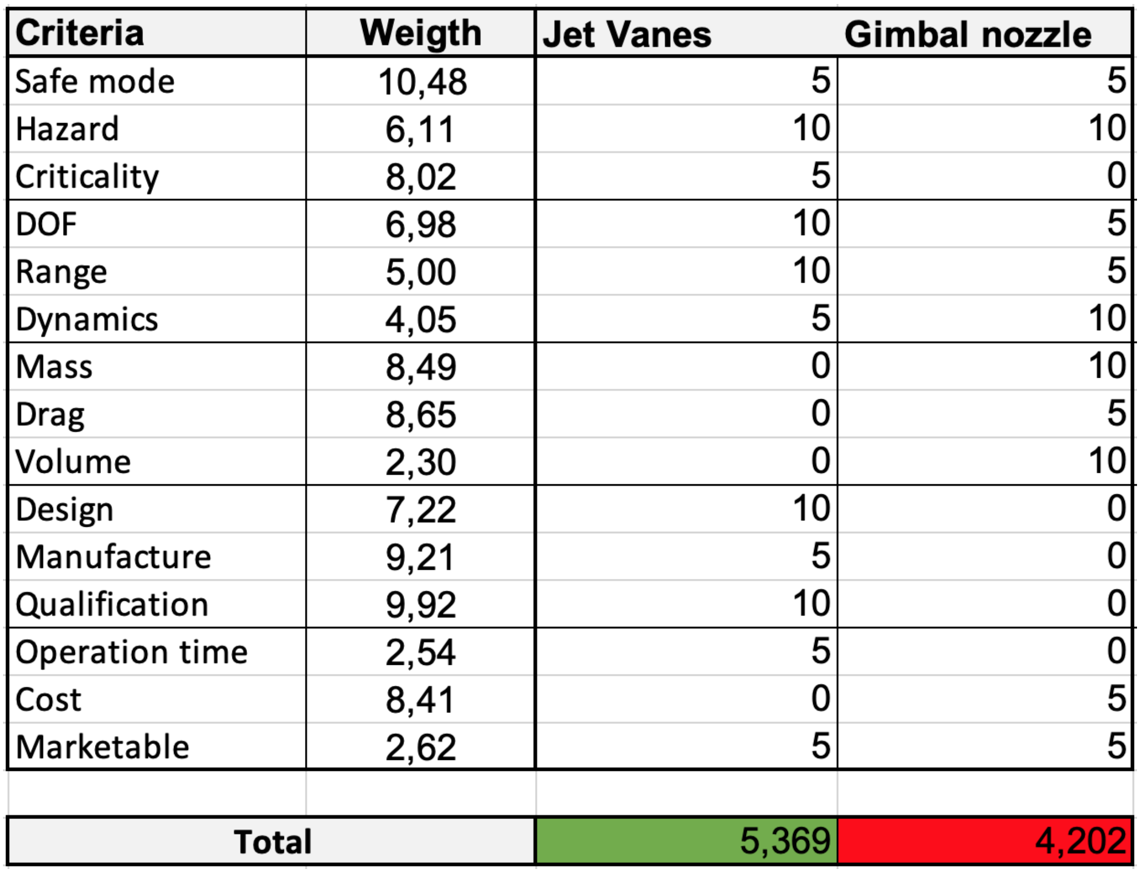
However, the long term objective of the team is to implement a real TVC system on a large-scale rocket. This type of rockets are typically three to four meters long, and weight between 35 and 45 kg at liftoff (see section 3.1.2).

First, a technology trade-off was done to chose the most suited technology for our rockets. The alternatives that were considered are:

- **Gimbal engine**: The whole rocket engine is mounted on a two-degree-of-freedom (2 DOF) mechanism to allow two rotations of the rocket engine thrust. This system can only generate pitch and yaw torque with one engine.
- **Jet vanes**: Three of four independent aerodynamic surfaces are placed at the exit of the rocket engine to redirect partially its plume, and thus its thrust. This system can generate roll torque ($\hat{\mathbf{z}}^b$ axis) by swirling the exhaust plume.
- **Liquid injection**: A set of valves controls the injection of a pressurized liquid near the exit of the gas to disturb its flow and control its direction.
- **Gimbal nozzle**: Similar to the gimbal engine, but only the nozzle part of the rocket engine is rotated, which makes the mechanism more compact and modular.

Out of these four options, two were quickly discarded for the large-scale rocket. First the gimbal engine, though state-of-the-art technology for commercial launchers due to its efficiency, cannot be used in our rockets, as our long rocket engine does not have enough room to be rotated inside our slim rockets. The liquid injection technology was discarded due to its complexity in the modelisation and implementation of the system.

To decide on the remaining two technologies, a complete trade-off was made, including definition and ranking of trade-off criteria. The final result is shown in figure 4.8. The complete trade-off process can be found in appendix B.



Criteria	Weigth	Jet Vanes	Gimbal nozzle
Safe mode	10,48	5	5
Hazard	6,11	10	10
Criticality	8,02	5	0
DOF	6,98	10	5
Range	5,00	10	5
Dynamics	4,05	5	10
Mass	8,49	0	10
Drag	8,65	0	5
Volume	2,30	0	10
Design	7,22	10	0
Manufacture	9,21	5	0
Qualification	9,92	10	0
Operation time	2,54	5	0
Cost	8,41	0	5
Marketable	2,62	5	5
Total		🟩 5,369	🟥 4,202

Figure 4.8: Ranking of the criteria

The Jet-vane solution clearly stand out in this trade-off study. Despite its lower efficiency due to the added mass and drag on the vanes, it has the advantage of being highly modular as it can be developed as a separate module to be placed at the exit of the rocket engine. This means that an already existing large-scale rocket, such as Bellalui-2 or Bellalui-1 can be directly used and augmented with the TVC module, even though these rockets were never designed to integrate a TVC system in the first place. Another important advantage of the jet vanes is that they allow for roll control of the rocket.

On the other hand, the gimbal nozzle solution was deemed more difficult to develop, due to the complexity of the sealed rotating joint that would have to withstand the extreme temperature and pressure of the combustion chamber. This technology also requires heavy modification of the

rocket engine to replace its current static nozzle, which would increase the risks to damage the engine.
Thus, it was decided that the long-term objective of project Icarus would be to develop a Jet Vane Thrust Vector Control (JV-TVC) system as a separate project to be integrated in the rocket Bellalui-2.

However, due to the complexity of this project, it was also decided to develop small-scale flight models that would use gimbal engine TVC. Indeed this technology is simpler, more efficient, and could easily be integrated in first flight model prototypes. The rest of this section thus describes the development of a small drone that integrates a type of gimbaled engine TVC for early flight demonstration. Then the first results of the jet vane TVC system developed by the team are presented.

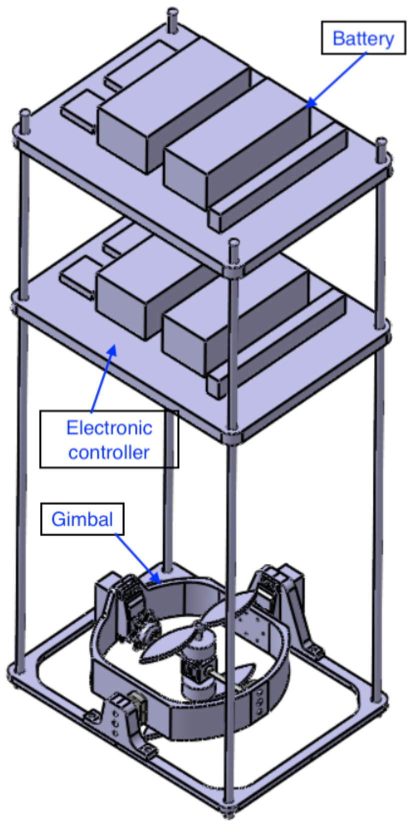
4.3.2 Drone model

It is common when developing GNC algorithms for a spacecraft or a launcher to start with a small scale version of the final product. This simplified model serves as a test platform to deploy and test first flight software with reduced cost and risks. One example of this is the "grasshopper" prototype from the aerospace company SpaceX, or more recently the "FROG" prototype developed by the Centre National d'Etudes Spatiales (CNES) which will serve as a first step in the project of developing a reusable launch vehicle for the family of Ariane rockets [28].
To support the long-term development of a TVC system for our rockets, a small and cheap Unmanned Aerial Vehicle (UAV) called "the drone" was designed, built, and used for first in-door flights experiments, using similar GNC architecture and development plan as what was presented in the previous chapters.

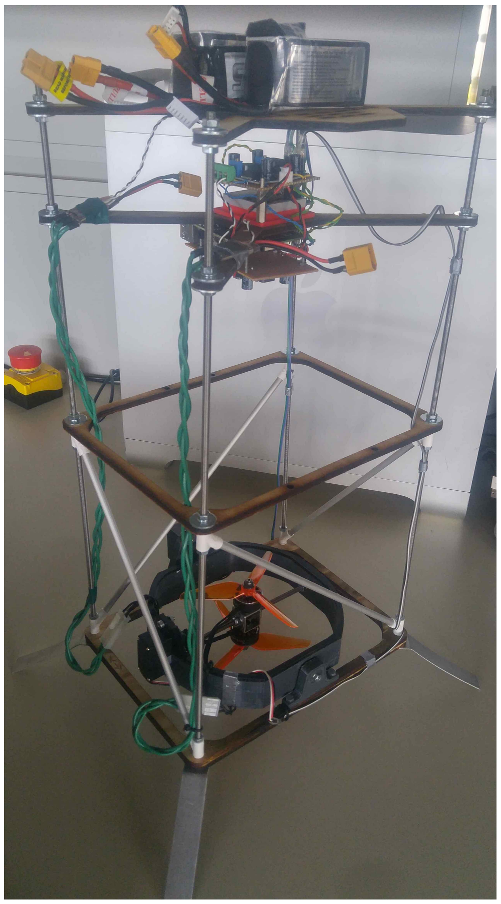
The drone was designed to mimic the dynamics of a rocket, with the center of mass placed above the controlled thrust. The TVC consists of two contra-rotating propellers placed on a 2 DOF gimbal.
The system identification, controller design, and tests flights were conducted by Raphaël Linsen during his semester project [26]. My work on this project was on the high-level design, the manufacture of the drone, and the support during the drone tests (see appendix C).

Mechanical design

The drone consists of three main parts: the TVC stage, electronic stage and battery stage. These are placed vertically on a tower-like structure, with the TVC stage at the bottom providing the controlled thrust with four actuators, the electronics stage in the middle hosts the power electronics and microcontroller, and finally at the top is the battery stage, which offset the center of mass away from the TVC stage.



(a) Drone initial concept



(b) Manufactured drone

The three stages are made out of lightweight plywood and connected together with four M4 threaded steel rods, so that the different stages could easily be moved or replaced for this first version. As shown in figure 4.9b, transverse aluminium rod and an empty plywood stage were added between the TVC and the electronic stage to add rigidity to the structure.

All the Computer-Aided Design (CAD) parts were modeled by Théotime Lemoine, a member of the EPFL Rocket Team.

Electronic design

The electronics is composed of a commercial drone board from *ST Electronics* which provides IMU, barometer and magnetometer sensor data. This board is connected via UART to a Raspberry Pi 4 which runs the GNC using ROS (see chapter 1). The Raspberry Pi also controls the four actuators using 50Hz Pulse-Width-Modulation: two brushless motors are used to provide thrust and roll control using two contra-rotating propellers, and two servo-motors are used to gimbal the brushless motors along two axes. One large battery is used to directly power the brushless motors through their Electronic Speed Controllers (ESC), while a smaller battery is used to power the two electronics boards and the servomotors.

Conclusion

A summary of the drone specification is shown in table 4.2. Overall this first version was very successful as a cheap and rapid to develop test platform. Thanks to the work of Raphaël on the GNC, it demonstrated first in-door controlled flight using the proposed GNC architecture and simulator developed during this thesis.

Specification	**Unit**	**Value**
Dimensions	mm	530*190*250
Mass	kg	156.8
Dry Mass	kg	1.53
Center of mass height	mm	205
Thrust range	N	4.9 to 19.6
Flight time	s	133 (2min 13s)

Table 4.2: Drone V1 specifications

A second version of this drone is being designed by Théotime Lemoine to be lighter and more precise in its physical characteristics. This new version will also integrate the rocket's avionics to support more realistic outdoor flight. The results of this work could also serve as a basis to develop a first small-scale rocket with gimbal TVC, which would be an intermediate flight prototype before the large scale rocket TVC.

4.3.3 Jet vane model

In this section are described some of the first results obtained by the Icarus team on the development of a jet-vane TVC (JV-TVC) system for the rocket Bellalui-2. This is a complex project involving mechanical and electronics design, fluid simulations, and extensive ground testing and characterization which necessitated the work of five technical semester projects. Supervision of the team was part of the work done for the thesis.

Mechanical design

The JV mechanism is designed to be a separate module containing the four independent vanes and their actuators. An image of the mechanism developed by Copenhagen Suborbitals for their rocket Sapphire is shown in figure 4.10. A very similar design is being developed at the moment, with the vane axis supported on a bearing, and actuated by a dedicated servo-motor.

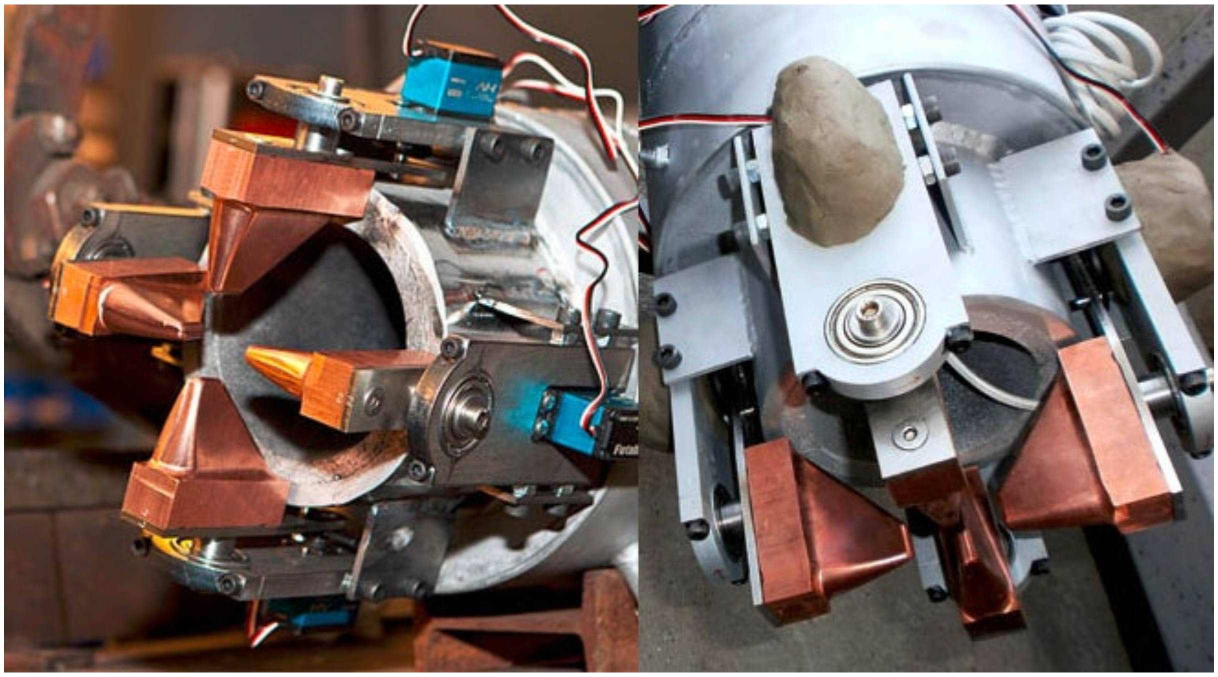


Figure 4.10: Jet vane mechanism from Copenhagen Suborbitals

For safety reason, the jet vane should return to its resting position in case of power failure, so that the rocket thrust is brought back to being colinear with the rocket $\hat{\mathbf{z}}^b$ axis. To do this, the center of pressure of each vane is placed behind their axis of rotation, so that the lift force create a torque stabilizing the vane around zero angle of attack.

The four vanes are attached to an aluminium plate that can be bolted to the rocket bottom plate, which makes the mechanism simple to assemble and highly modular.

Electronic design

The work on the electronic has been done by Iacopo Sprenger [1]. Besides the development of the electronic interface presented in section 1.2.1, he is working on the low-level electronics controlling the sensors and actuators of the Propulsion sub-system, including the four servo-motors to control the four jet vanes orientations.

We chose the servo-motor model *XL-330-M288-T* from Dynamixel due to its small size, powerful features, and ease of integration. The specifications of this servo-motor are presented in the following table:

Specification	**Unit**	**Value**
Dimensions	mm	20*34*26
Mass	g	18
Stall torque	N.m	0.52
No load speed	rpm	104
Resolution	°	0.09
Supply voltage	V	3.7 to 6
Communication protocol	-	Half duplex serial (TTL)

Figure 4.11: Dynamixel XL-330 servo-motor

The Transistor-to-Transistor Logic (TTL) serial communication protocol makes it very easy to control as the servo-motor can be directly connected to the propulsion microcontroller board. Thus, except for the wiring and connectors, no special hardware had to be developed to integrates these motors. The current Bellalui-2 rocket electronics can be re-used as is, with the GNC algorithms deployed into the upper electronics stage (avionics), and sending high level commands to the lower electronics stage (propulsion). These servo-motors can receive commands in position or speed, have configurable step response, and sends back their current position and current consumption, which could be used for torque estimation of the jet vanes. The last important feature is the ability to daisy-chain the four servomotors, which simplify their integration in the rocket by minimizing wiring.

Jet vane model

Compared to a gimbal TVC where the thrust of the engine is directly redirected by the gimbal, a JV-TVC has a more complex model, depending on aerodynamic effects between the supersonic exhaust gas and the jet vane.
Thus, the first step to develop a model is to use Computational Fluid Dynamics (CFD) to simulate the rocket engine and its interaction with the jet vanes.
This was the semester project of Théo La Marca [29] who setup the CFD simulation using *Fluent* and used it to find a relation between aerodynamic forces and angle of attacks of the vanes, as seen in figure 4.12.

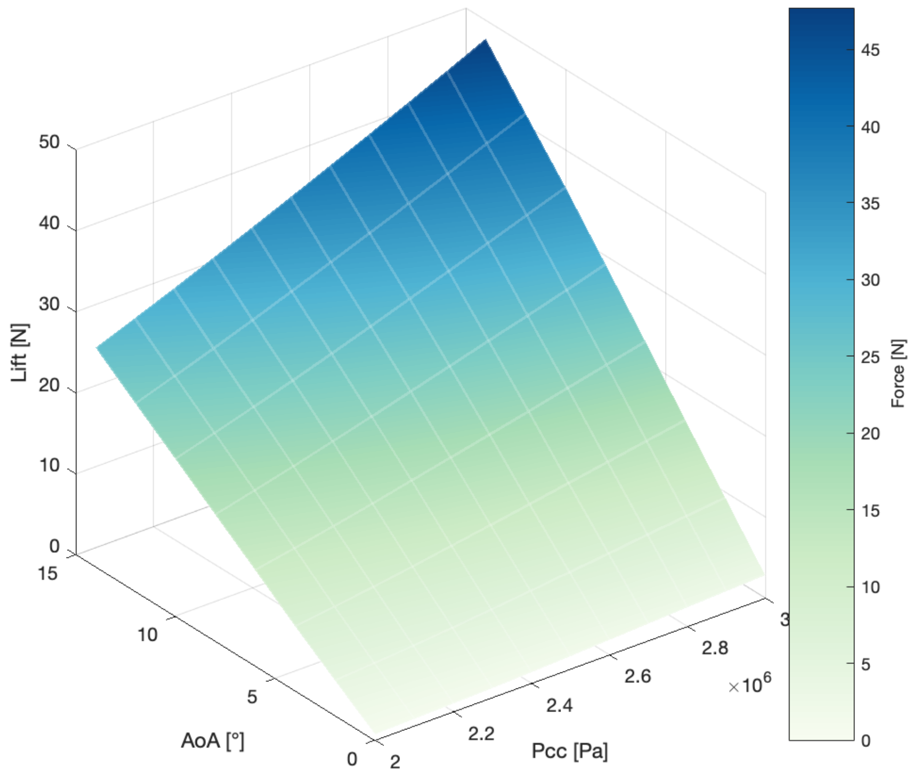


Figure 4.12: Lift force of one vane, for various angle of attack and combustion chamber pressure

The lift force is perpendicular to the flow, so it is directly a force in either the $\hat{\mathbf{x}}^b$ or $\hat{\mathbf{y}}^b$ axis. We see that the lift force is dependent on the combustion chamber pressure, which defines the thrust of the rocket engine. Depending on the main thrust, we thus have between 50 and 96 Newtons of lateral force because each axis has two vanes to control it. For reference, the simulations presented in 4.2.2 have a limit of 50 Newton of lateral force.

The same type of fluid simulations were done to find the drag force of the jet vanes, which creates a force in the $-\hat{\mathbf{z}}^b$ axis of the rocket, opposite to the thrust force. All these simulations gives us a first model of the JV-TVC for equation 4.4.

The next step to have a more reliable model would be to directly measure these forces during a static fire test of the engine, with the JV-TVC mechanism mounted on our test bench.
For this, Arnaud Muller developed [30] a platform to measure 3D forces and torques with an assembly of cheap 1D load cells, as seen in figure 4.13.

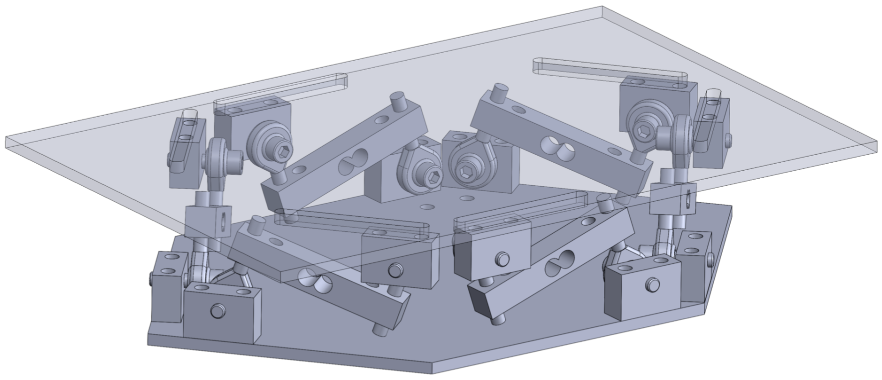


Figure 4.13: CAD model of the 6 DOF measuring platform

Besides allowing system identification for the TVC system that will be developed, this measuring platform could also be used for complete Hardware-In-the-Loop by providing force feedback to the simulation, as presented in chapter 2.

Thermal analysis

The last development of this project was to study different materials and shapes for the jet vanes. Indeed, the exhaust gas is estimated to have temperature between 2500 and 3000 Kelvin, with flow velocity around Mach 2.5. This very harsh environment can destroy the vane in a few seconds if not taken into consideration.
A project [31] conducted by Usama Qayyum was dedicated to simulate the thermal behaviour of different materials. At the same time, simple experiments were conducted to place jet vanes in a static fire test of the engine to directly measure the impact of the rocket engine on them.

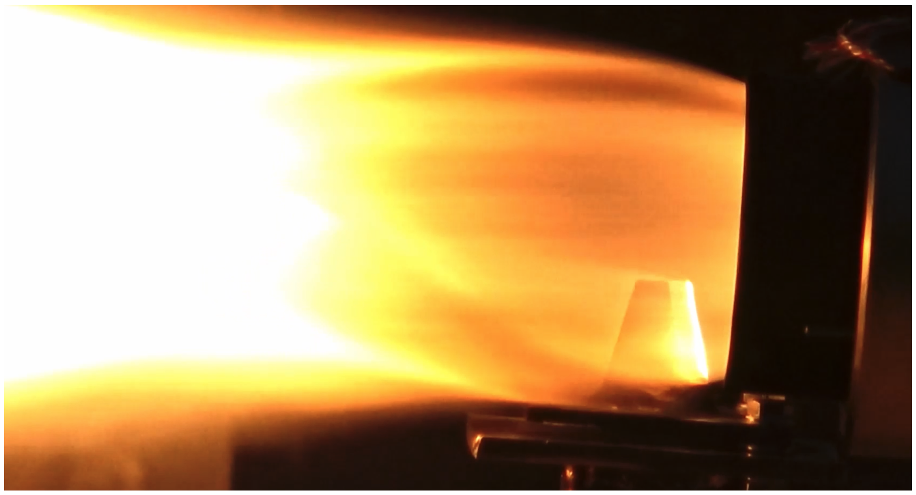


Figure 4.14: Thermal test on a steel vane during a static fire test of our hybrid rocket engine

This way, steel, graphite and copper vanes were tested, and compared with the result of the simulation. Steel vanes were quickly discarded as they melt in less than 3 seconds, graphite withstand very well the temperature but showed signs of erosion which could impact its aerodynamic properties during the flight, and finally copper vane survived for a short static fire test, and would thus need to be tested again on a more realistic static fire test of 5 to 10 seconds.

Summary

This section briefly presented the first developments of the Jet-Vane TVC system that could be integrated in Bellalui-2 in the future. All the aspects of this system have been studied in this first project, including the mechanism, electronics, simulations and test bench, but all will need much further development to be ready for ground tests and qualification.

The results of this future test campaign will be critical to determine if this system is safe and ready for a first controlled flight of a large-scale rocket.

Conclusion

In this chapter, the Thrust Vector Control technology was presented and its importance for a rocket launcher was demonstrated by developing a simple Software-in-the-Loop setup. The different objectives of attitude, position and velocity control were met in the simulation environment, and confirmed its potential for the EPFL Rocket Team as a promising technology for better trajectory control and to allow more complex missions in the future.
The first results of two flight models were also presented. One is a drone which demonstrated the capability of the EPFL Rocket Team to develop complete Guidance, Navigation and Control for a Thrust Vectored Controlled UAV, and the other is a Jet-Vane TVC system for our rocket Bellalui-2 which could be the first large scale rocket to be actively controlled with a TVC system developed by a team of students.

It is clear that both of these projects will need more development in order to reach the ambitious goals set by the team. The drone project should continue on its iterative improvement, by gradually adding more complexity and features of a real rocket launcher, like embedded sensors for outdoor flights or more realistic flight profiles. On the other hands, the JV-TVC project should focus on finishing the current development of its mechanism, electronics and vane material and shape optimization, in order to raise its technology maturity, and allow real hardware tests and characterization.

Conclusions

This thesis presents a new framework to develop advanced GNC for the EPFL Rocket Team. It includes a complete definition of the software and hardware architecture, as well as reliable tools for simulation and analysis that accompany students from the GNC development to its qualification through Software and Processor-in-the-Loop to its deployment on flight hardware.
These tools and methods have demonstrated their effectiveness in three different projects: the first one is a small UAV which demonstrated controlled flight and position tracking, the second one is a GNC for a sounding rocket that was developed and tested to a flight-ready level, and the last one is a research GNC project on thrust vector control which went through its first phase of development in a simulation environment to assess its performance and value for the EPFL Rocket Team.

Unfortunately, due to lack of time and canceled rocket launches, the two GNC developed for sounding rockets were not tested during flight. The focus of this thesis was thus on ground testing and qualification to demonstrate that with the proper tools, a large part of this process can be done without flight tests. Nevertheless, it is critical that the results obtained through simulations are validated with the upcoming rocket launches to qualify both the GNC and the method proposed in this thesis.
Besides these results, additional work may be needed in the future to complete the simulator with new features for better physics modeling, new tools for analysis, and a more intuitive user interface. One of the major default of this tool is its complexity that could limit inexperienced students to use it. Adding scripts to simplify the installation and automate repetitive tasks could help this.

Overall, this project was a great experience in learning how to develop GNC algorithms and the process of qualifying them for aerospace application. Besides some setbacks during the project, the results obtained demonstrate the added value of this tool for the EPFL Rocket Team and will hopefully encourage students to use it in their future projects.

[illegible]

Appendix A

INS ground test procedure

INS PERFORMANCE

Filename:	2021_TVC_TP_INS_PERFORMANCE.docx
Prepared by:	Albéric de Lajarte
Checked by:	
Approved by:	

REVISION HISTORY

Revision	Description	Date
Baseline	First complete draft	29/03/2021
Rev. 02	Updated dynamic test procedure	05/05/2021
Rev. 03	Result of test 1, 2 and 3	15/05/2021

TABLE OF CONTENTS

1 INTRODUCTION

To estimate the full state of the rocket during the flight, an Inertial Navigation System (INS) is developed. This algorithm simply integrates IMU data to estimate orientation, velocity and position in real time.

A Kalman filter on altitude and vertical velocity is added to correct these two states with a barometer.

This algorithm having no model of the rocket, it is very simple to test on the ground, but has the problem of being very dependent on good sensor data for good performances.

This test aims to assess the performance of this algorithm for basic, known motions.

2 REFERENCE DOCUMENTS

Table 2-1 Reference Documents

Ref	Description	Doc. Number	Issue
[RD01]	[RD02] Acceptance REDV test procedure.pdf	S3-D-SET-1-2	1.0

3 DEFINITIONS AND ABBREVIATIONS

RD	Reference Document
GNC	Guidance, Navigation and Control
INS	Inertial Navigation System
MPC	Model predictive Control
ROS	Robot Operating System
TVC	Thrust Vector Control

4 VERIFICATION PROCESS

4.1 Verification objectives

The first objective of this test is to quantify the drift of the orientation, velocity and position estimates over time due to sensor bias.

The second objective is to measure its precision in orientation and position.

4.2 Verification method

To find the drift over time, the avionics is firstly kept still in a fixed position and orientation for around 30s. In a second test, the avionics is moved and rotated by hand during 30s, then brought back to the initial orientation and position to assess the influence of motion on drift.

For the precision test, the avionics is constrained to move along a simple predefined path. The measured position and orientation are compared to the predicted one to find the precision.

4.3 Test procedures

The complete avionics is used, with the sensor board mounted on its structure. The GNC module is integrated on the AV stack. The AV stack is mounted on its structural module on which modified coupler with squared features are attached to provide a square reference.

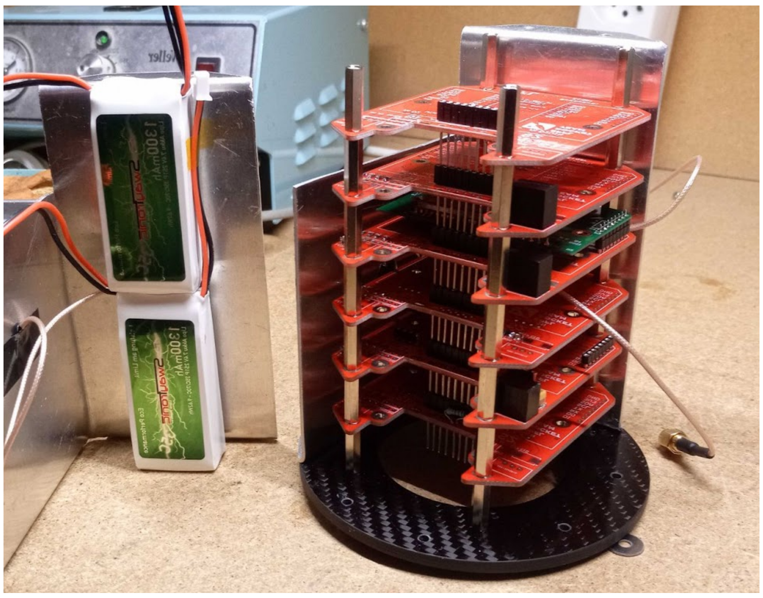


Figure 1: Avionic module with integrated IMU and barometer

The GNC algorithms are running on a Raspberry pi 4 compute module, and all data are logged using the ROS logging tools.

The automatic calibration routine is used to estimate the accelerometers and gyroscope bias by comparing their measurement to the known initial pose.

4.4 Test conditions

The test is done on a flat and horizontal table. Two perpendicular surfaces are attached to the table to provide a reference.

A last test is made in the MED building to test the overall behavior of the algorithm by moving the module back and force along the stairs.

4.5 Test setup and equipment

Reference setup:

A straight beam is fixed to a flat and horizontal marble table. Using shims and an inclinometer, the table can be made horizontal with a precision of less than 0.2°.
The beam provides another reference surface at 90° to the marble table on which the avionics module will be slide on.
Two other blocs at both end of the table are used to provide mechanical stops and thus a length reference of 50cm.

MED setup:
The module is initialized on a flat table at the bottom of the stairs. A person carries the module up and down the stairs, and the altitude and vertical velocity of the module are monitored in real time by a user at the ground station.

Equipment:

- ❏ Flat table
- ❏ Inclinometer
- ❏ Clamp (x2)
- ❏ Square bloc (x2)
- ❏ Rubber hammer
- ❏ Ground station software

4.6 Step-by-step test procedure

1. Static drift test
   1.1. Position the AV vertically module on the flat table, +X axis on the direction of the track (initial pose)
   1.2. Power it on and wait 10s.
   1.3. Start the GNC algorithms: “bellalui_flight.launch”. Wait for the calibration to finish (5s)
   1.4. Lightly hit the module with a soft hammer to create an acceleration higher than 1.5g
   1.5. Wait 30s in without moving the module
   1.6. Stop the GNC algorithms and the AV. Retrieve the log file.

2. Dynamic drift test
   2.1. Position the AV vertically module on the flat table, +X axis on the direction of the track (initial pose)
   2.2. Start the GNC algorithms: “bellalui_flight.launch”. Wait for the calibration to finish (5s)
   2.3. Lightly hit the module with a soft hammer to create an acceleration higher than 1.5g

2.4. Move the module by hand to create rotations in all axes of less than 30° at a rate of roughly 1Hz, and translations of less than 50cm in all axes
2.5. Stop the GNC algorithms and the AV. Retrieve the log file.

3. Dynamic precision test
    3.1. Position the AV vertically module on the flat table, +X axis on the direction of the track (initial pose)
    3.2. Start the GNC algorithms: “bellalui_flight.launch”. Wait for the calibration to finish (5s)
    3.3. Lightly hit the module on the Z axis with a soft hammer to create an acceleration higher than 1.5g
    3.4. Move the module along the +X axis for 30cm (track length), and back to its original pose (-X translation) in less than 5s
    3.5. Stop the GNC algorithms. Retrieve the log file.
    3.6. Repeat the procedure to test each axis in the same manner, making sure to restart the algorithm between each test, and to modify the initial pose in the GNC configuration

4. Full motion test
    4.1. Position the AV vertically module on the flat table, marking the direction of the X axis and the initial 3D position.
    4.2. Power on the module and wait 10s.
    4.3. Activate the Raspberry pi from the ground station. Wait until seeing 1mm as initial position in the ground station software
    4.4. Lightly hit the module on the Z axis with a soft hammer to create an acceleration higher than 1.5g
    4.5. Start logging the data on the ground station software
    4.6. Carry the module up the stairs at normal walking speed. Turn around and go down the stairs to carry the AV module at its initial pose.
    4.7. Stop the GNC, AV and GS. Retrieve all logging files.

4.7 Pass-fail criteria

Test	Pass	Fail
Attitude drift	- The pitch and yaw and roll angles are kept below 5° during the two drifts test	- The orientation diverges from vertical (more than 5°)
Translation drift	- The X and Y position are below 50m for the two drift tests. - The Z position is below 2m for the two drift tests. - The Z speed is below 1m/s for the two drift tests.	- The X and Y position diverges above 50m - The Z position diverges above 2m - The Z speed diverges above 1m/s
Position precision (test 3)	- Each axis should have a postion error of less than 50cm - The residual speed should be below 1m/s	- The position error is larger than 50cm - The speed has drifted above 1m/s at the end of the test

5 TESTS RESULTS

Test 1:

The first test lasted 3min of continuous integration. The attitude, vertical velocity and altitude showed very good results, while the lateral positions and velocities drifted quickly away from the reference.

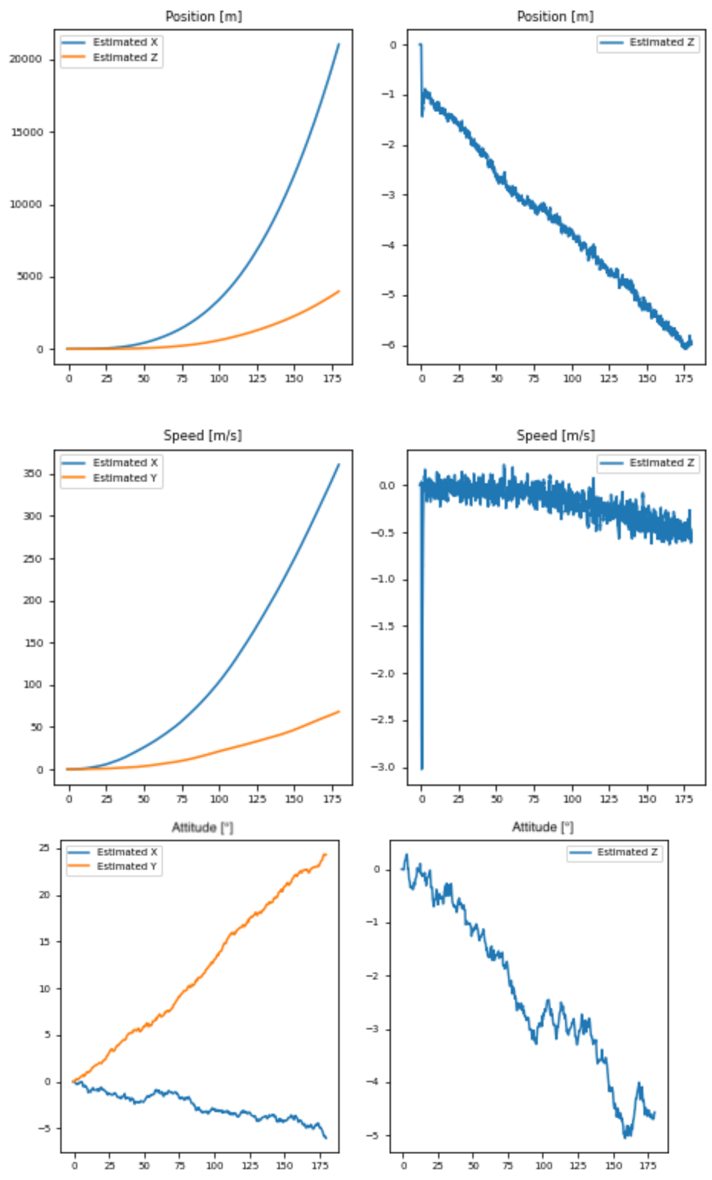


Figure 2: Position (top), Velocity (middle) and Euler angles (bottom) as a function of time in seconds

We see that the vertical velocity and altitude are very stable thanks to the barometer correction. The attitude is also acceptable, staying below 5° for a typical flight duration (30s). However the horizontal position drifts too quickly, making its estimates unusable.

Criteria	Pass / Fail	Reason
Attitude drift	Pass	
X/Y translation drift	Fail	Accelerometer errors accumulates too quickly. Need a better calibration method
Z position drift	Pass	
Z speed drift	Pass	

Test 2:

The test lasted 60s, with 40s of dynamic phase. The biggest difference was on the attitude estimation which performed significantly worst, accumulating a bias after the dynamic phase.

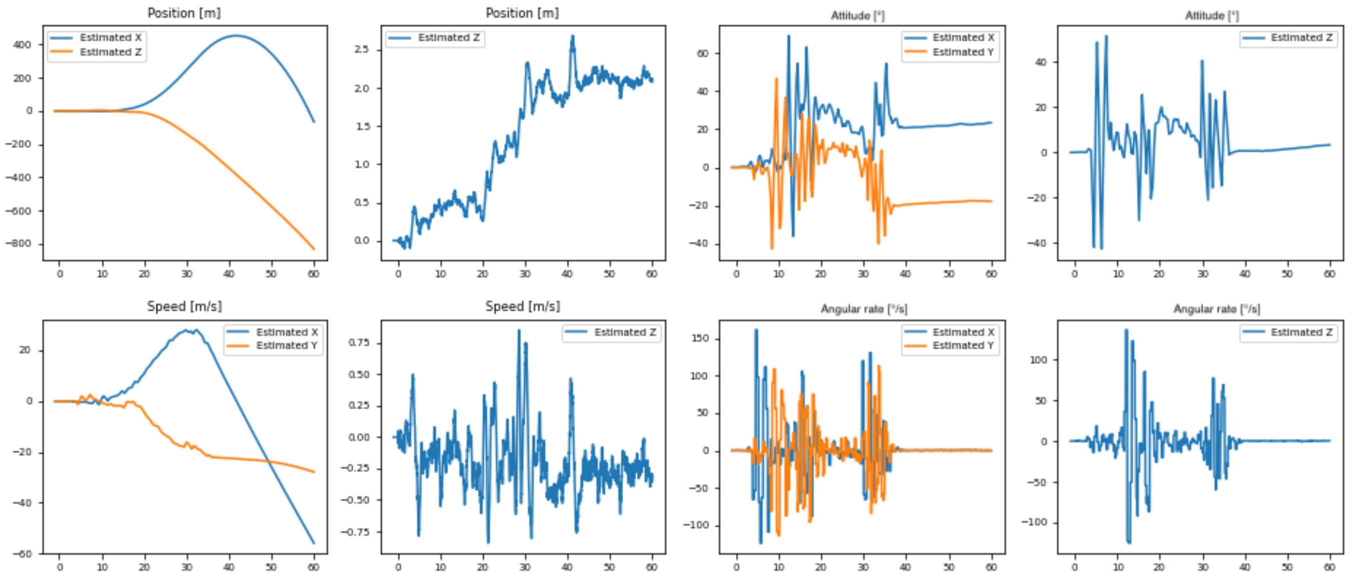


Figure 3: Position, velocity, attitude and angular rate as a function of time

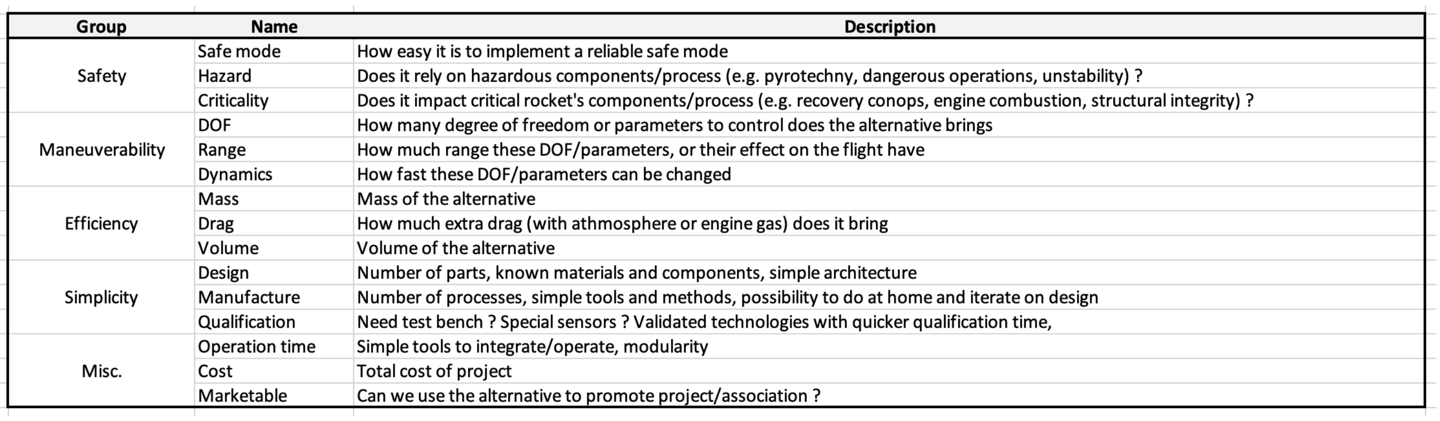
We see that two of the three Euler angles drifted to ±20° after the dynamic phase. This could be corrected by increasing the sampling frequency or the calibration time.

The position and velocity estimates have similar performance.

Criteria	Pass / Fail	Reason
Attitude drift	Fail	Bias after dynamic phase
X/Y translation drift	Fail	Accelerometer errors accumulates too quickly. Need a better calibration method
Z position drift	Pass	
Z speed drift	Pass	

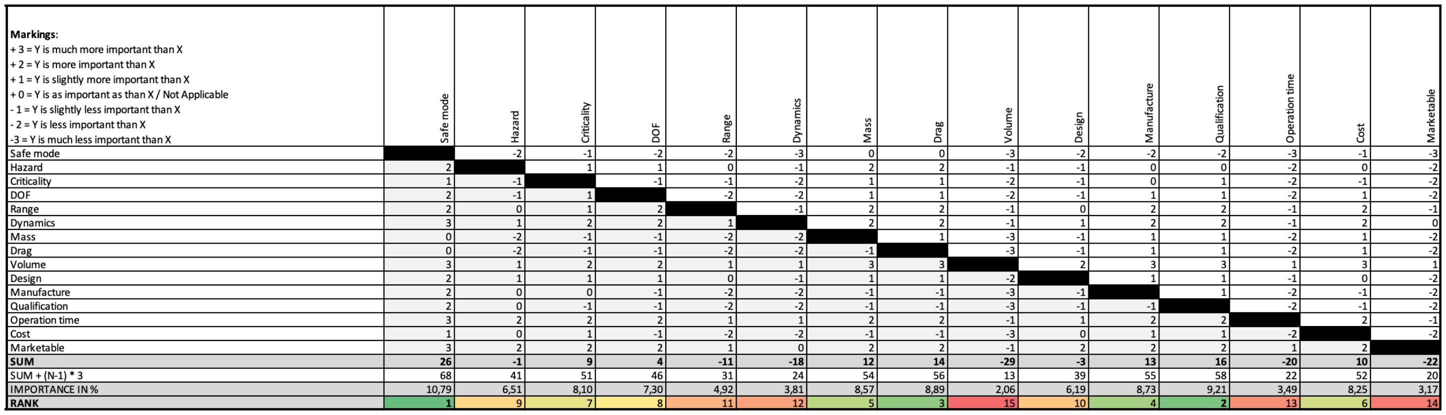
Appendix B

TVC trade-off



Group	Name	Description
Safety	Safe mode	How easy it is to implement a reliable safe mode
	Hazard	Does it rely on hazardous components/process (e.g. pyrotechny, dangerous operations, unstability) ?
	Criticality	Does it impact critical rocket's components/process (e.g. recovery conops, engine combustion, structural integrity) ?
Maneuverability	DOF	How many degree of freedom or parameters to control does the alternative brings
	Range	How much range these DOF/parameters, or their effect on the flight have
	Dynamics	How fast these DOF/parameters can be changed
Efficiency	Mass	Mass of the alternative
	Drag	How much extra drag (with athmosphere or engine gas) does it bring
	Volume	Volume of the alternative
Simplicity	Design	Number of parts, known materials and components, simple architecture
	Manufacture	Number of processes, simple tools and methods, possibility to do at home and iterate on design
	Qualification	Need test bench ? Special sensors ? Validated technologies with quicker qualification time,
Misc.	Operation time	Simple tools to integrate/operate, modularity
	Cost	Total cost of project
	Marketable	Can we use the alternative to promote project/association ?

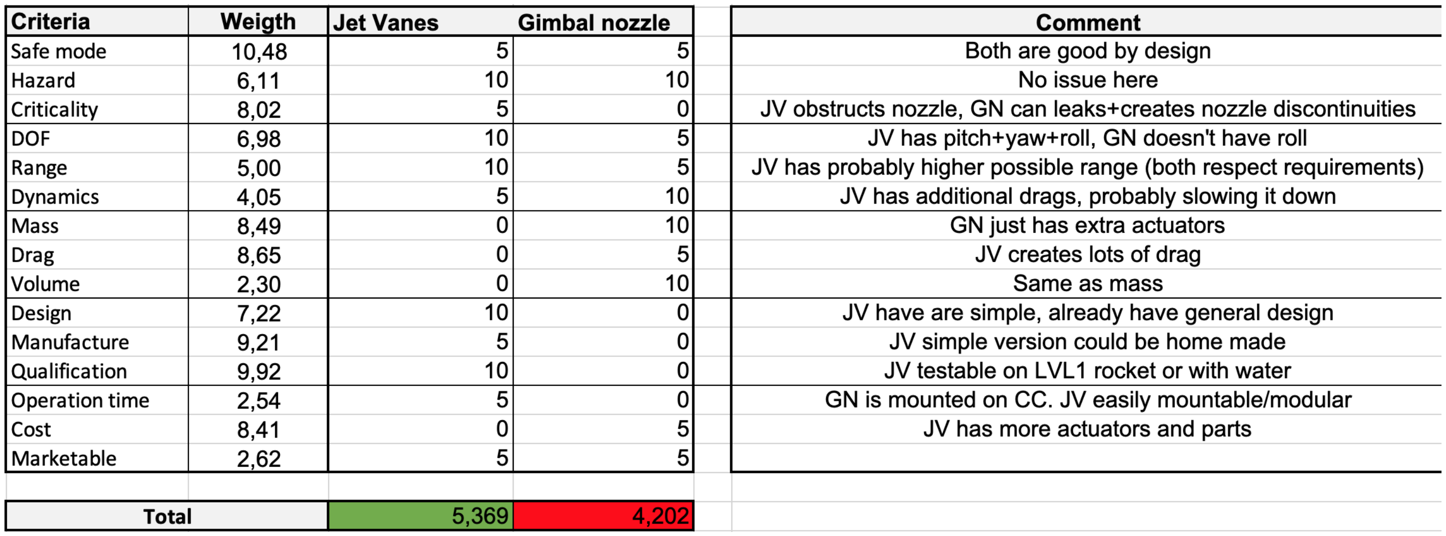
Figure B.1: Definition of the criteria



Markings:
+ 3 = Y is much more important than X
+ 2 = Y is more important than X
+ 1 = Y is slightly more important than X
+ 0 = Y is as important as than X / Not Applicable
- 1 = Y is slightly less important than X
- 2 = Y is less important than X
-3 = Y is much less important than X

	Safe mode	Hazard	Criticality	DOF	Range	Dynamics	Mass	Drag	Volume	Design	Manufacture	Qualification	Operation time	Cost	Marketable
Safe mode		-2	-1	-2	-2	-3	0	0	-3	-2	-2	-2	-3	-1	-3
Hazard	2		1	1	0	-1	2	2	-1	-1	0	0	-2	0	-2
Criticality	1	-1		-1	-1	-2	1	1	-2	-1	0	1	-2	-1	-2
DOF	2	-1	1		-2	-2	1	1	-2	-1	1	1	-2	1	-2
Range	2	0	1	2		-1	2	2	-1	0	2	2	-1	2	-1
Dynamics	3	1	2	2	1		2	2	-1	1	2	2	-1	2	0
Mass	0	-2	-1	-1	-2	-2		1	-3	-1	1	1	-2	1	-2
Drag	0	-2	-1	-1	-2	-2	-1		-3	-1	1	1	-2	1	-2
Volume	3	1	2	2	1	1	3	3		2	3	3	1	3	1
Design	2	1	1	1	0	-1	1	1	-2		1	1	-1	0	-2
Manufacture	2	0	0	-1	-2	-2	-1	-1	-3	-1		1	-2	-1	-2
Qualification	2	0	-1	-1	-2	-2	-1	-1	-3	-1	-1		-2	-1	-2
Operation time	3	2	2	2	1	1	2	2	-1	1	2	2		2	-1
Cost	1	0	1	-1	-2	-2	-1	-1	-3	0	1	1	-2		-2
Marketable	3	2	2	2	1	0	2	2	-1	2	2	2	1	2	
SUM	**26**	**-1**	**9**	**4**	**-11**	**-18**	**12**	**14**	**-29**	**-3**	**13**	**16**	**-20**	**10**	**-22**
SUM + (N-1) * 3	68	41	51	46	31	24	54	56	13	39	55	58	22	52	20
IMPORTANCE IN %	10,79	6,51	8,10	7,30	4,92	3,81	8,57	8,89	2,06	6,19	8,73	9,21	3,49	8,25	3,17
RANK	1	9	7	8	11	12	5	3	15	10	4	**2**	13	6	14

Figure B.2: Ranking of the criteria



Criteria	Weigth	Jet Vanes	Gimbal nozzle	Comment
Safe mode	10,48	5	5	Both are good by design
Hazard	6,11	10	10	No issue here
Criticality	8,02	5	0	JV obstructs nozzle, GN can leaks+creates nozzle discontinuities
DOF	6,98	10	5	JV has pitch+yaw+roll, GN doesn't have roll
Range	5,00	10	5	JV has probably higher possible range (both respect requirements)
Dynamics	4,05	5	10	JV has additional drags, probably slowing it down
Mass	8,49	0	10	GN just has extra actuators
Drag	8,65	0	5	JV creates lots of drag
Volume	2,30	0	10	Same as mass
Design	7,22	10	0	JV have are simple, already have general design
Manufacture	9,21	5	0	JV simple version could be home made
Qualification	9,92	10	0	JV testable on LVL1 rocket or with water
Operation time	2,54	5	0	GN is mounted on CC. JV easily mountable/modular
Cost	8,41	0	5	JV has more actuators and parts
Marketable	2,62	5	5	
Total		5,369	4,202	

Figure B.3: Ranking of the criteria

Appendix C

Drone test procedure

DRONE DOME TEST FLIGHT

Filename:	2021_TVC_TP_DroneDomeTestFlight_R01.docx
Prepared by:	Albéric de Lajarte
Checked by:	Raphaël Linsen
Approved by:	Peter Listov

REVISION HISTORY

Revision	Description	Date
Baseline	First complete draft	29/03/2021
Rev. 02	a. Minor formulation corrections	31/03/2021

Table of Contents

1 INTRODUCTION

This test is the first flight of the Thrust Vector Control team’s drone. It aims at testing the performance of the MPC controller in a safe and controlled environment developed especially for drone testing.

This test being a first for the team, it is necessary that the drone is first fully characterized, and the MPC controller tested extensively in both simulation and static test bench prior to drone dome testing.

1 REFERENCE DOCUMENTS

Table 2-1 Reference Documents

Ref	Description	Doc. Number	Issue
[RD01]	[RD02] Acceptance REDV test procedure.pdf	S3-D-SET-1-2	1.0

2 DEFINITIONS AND ABBREVIATIONS

RD	Reference Document
GNC	Guidance, Navigation and Control
MPC	Model predictive Control
TVC	Thrust Vector Control

2 VERIFICATION PROCESS

2.1 Verification objectives

The first goal of this test is to verify the ability of the controller to stabilize the drone in the vertical orientation.
The secondary objective is to track a constant 3D position required by the user.
The last objective is to assess the performance of the embedded navigation algorithm by comparing it to the reference tracking system.

2.2 Verification method

The attitude and position of the drone are measured by an optical tracking system that provides a reference to measure the stability of the attitude and the precision of its position.

2.3 Test procedures

The MPC controller is used on the drone V1, shown in figure 1.

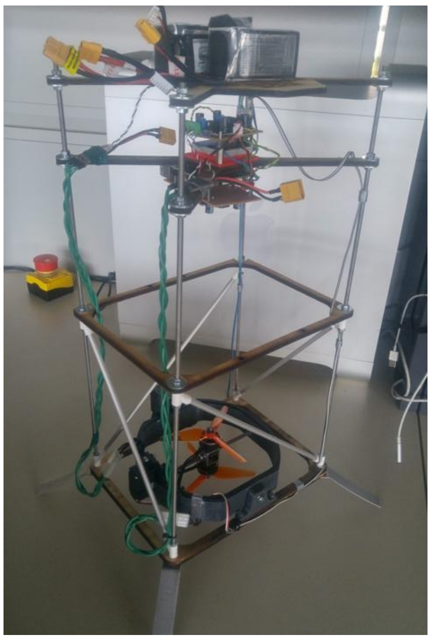


Figure 1: Version 1 of the test drone

The following components will be used:

Component	Reference
ESC battery (x1)	LiPo 6 cell, 1400mAh, 120C (Tattu R-line)
Board battery (x1)	LiPo 3 cell, 1000mAh
ESC (x2)	HGLRC ESC 35A 3-6S T-REX
Motor (x2)	Avenger 2306.5 V3, 2000KV
Propeller (x2)	5.1x3, pitch 4.2 (R42)

Servomotors (x2)	Futaba S3305
Sensor board	STEVAL-FCU001V1
Processing board	Raspberry pi 4 model B, 4GB

The following characteristics have been measured:

Parameter	Value
Drone total mass	1.533 kg
Inertia	0.0623, 0.0669, 0.0130 kg/m^2
Outer dimensions	25x19x53 cm
Center of mass height	20.5 cm

2.4 Test conditions

The test will be indoor, in a closed net of approximately 10x10 m^2 and 5m high.
The position of the drone will be limited by a tether to a 1 m radius horizontally and 3 m distance vertically. Each test can last a maximum of 2.5 minutes due the limited capacity of the battery.

2.5 Test setup and equipment

Safety setup:

A cord is attached at the top and bottom of the drone's structure. The cord forms a loop that passes through the roof's structure and a heavy table at the bottom. These two rigid points ensure the drone's movements are restrained horizontally, but don't restrain its vertical motion. One person is in charge of managing the tension of the loop, in order to block the drone in case of a controller failure.

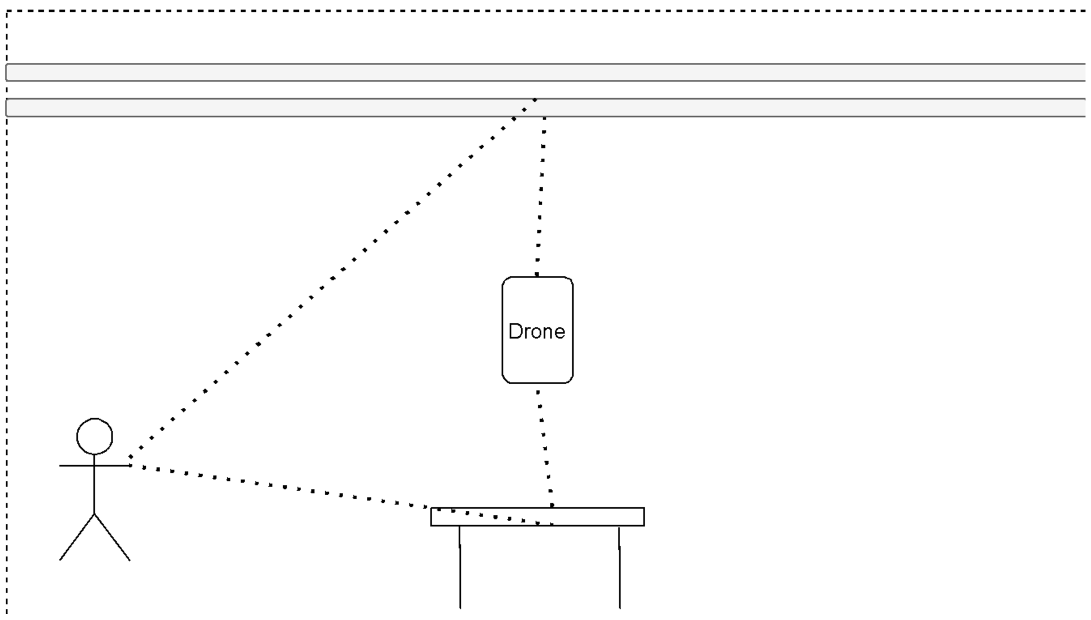


Figure 2: Safety setup. The outer rectangle is the large net of the drone dome

The person managing the cord inside the net needs to wear safety gloves and goggles.

Tracking setup:

The opti-track is used to track small reflective balls fixed on the drone. The balls and the camera are already calibrated, so no recalibration is necessary if the tracking precision is deemed sufficient.

Equipment:

- ❏ 20 m long cord
- ❏ scissors
- ❏ Safety goggles (x4)
- ❏ Safety gloves (x1)
- ❏ Ear plug (x4)
- ❏ Battery charger
- ❏ Backup ESC batteries (x2) and board batteries (x2)
- ❏ 5V Raspberry Pi charger
- ❏ Spanner M4 and M5
- ❏ Allen keys
- ❏ Electrical tape and zip-ties
- ❏ Multimeter
- ❏ Spare propellers
- ❏ Level

2.6 Step-by-step test procedure

1. Optitrack and ground station setup
   1.1. Setup optitrack on the local workstation
   1.2. Make the drone connect itself to the optitrack server's wi-fi
   1.3. Connect a laptop to the optitrack server's wi-fi and check that the SSH and ROS connections are working to the drone
   1.4. Launch optitrack on the local workstation
   1.5. Launch the ROS GUI on the laptop and check that the drone orientation and position are being received

2. Drone setup
   2.1. Install the safety cord on the drone and on the room's structure
   2.2. Position the drone on the ground with the sensor board frame in the same orientation as the Optitrack frame
   2.3. Power on the drone while minimizing vibrations on the sensor board (IMU calibration)
   2.4. Verify that the estimated drone orientation, position and velocity are coherent

3. Tethered test

3.1. Plug the ESC batteries and place the drone at its target position by hanging it from the cord
3.2. Start the flight controller from the ground station with **25% thrust limit**
3.3. Verify proper actuation of the servomotors and motors by balancing the drone like a pendulum in each axis of rotation

4. Stability test
   4.1. Plug the ESC batteries and place the drone at its target position by hanging it from the cord
   4.2. Start recording all flight data. Start filming the flight
   4.3. Start the flight controller from the ground station with **80% thrust limit**
   4.4. Prepare to stop the test in case of controller failure or empty battery
   4.5. Repeat this test until 3 stable flight as defined in 4.7 have been demonstrated

5. Altitude tracking
   5.1. Plug the ESC batteries and place the drone 2 meters away from its target altitude
   5.2. Start recording all flight data. Start filming the flight
   5.3. Start the flight controller from the ground station
   5.4. Prepare to stop the test in case of controller failure or empty battery
   5.5. Repeat this test until 3 precise flight as defined in 4.7 have been demonstrated

6. 3D position tracking
   6.1. Plug the ESC batteries and place the drones 2 meters away from its targeted position
   6.2. Start recording all flight data. Start filming the flight
   6.3. Start the flight controller from the ground station
   6.4. Prepare to stop the test in case of controller failure or empty battery
   6.5. Repeat this test until 3 precise flight as defined in 4.7 have been demonstrated

7. End of test
   7.1. Stop the flight controller.
   7.2. Unplug all batteries connected to the drone and store them
   7.3. Retrieve all flight data
   7.4. Tidy up the room and equipment

2.7 Pass-fail criteria

Test	Pass	Fail
Attitude stabilization	- The pitch and yaw are kept below 10° during the whole test - The roll angle is kept below 90° during the whole test - The angular rates are all below 20°/s - The drone's position doesn't deviate more than 0.5 m	- The orientation diverges from vertical and the drone loses stability. - The drone oscillates at high rate around the vertical orientation - The drone position diverges from the initial position

Altitude tracking	- The attitude stabilization works as defined above - The desired altitude is reached with a precision of 10 cm - The settling time is less than 5s	- The drone loses stability (as defined above) when moving - The steady state error is larger than 10cm, or grows over time - The drone oscillates indefinitely around its target
3D position tracking	- The attitude stabilization works as defined above - The desired position is reached with a precision of 10 cm - The settling time is less than 5s	- The drone loses stability (as defined above) when moving - The steady state error is larger than 10cm, or grows over time - The drone oscillates indefinitely around its target

References

[1] I. Sprenger, ‘Integration of a low-cost microprocessor for a sounding rocket’s avionics,’ Semester project, 2021 (cit. on pp. 8, 77).

[2] M. Quigley, B. Gerkey and K. Conley, ‘Ros: An open-source robot operating system,’ *ICRA Workshop on Open Source Software*, 2009 (cit. on pp. 10, 13).

[3] E. Brunner and E. Mingard, ‘Simulation avancée de la trajectoire d’une fusée et application à du contrôle actif,’ Semester project, 2018 (cit. on pp. 15, 20, 57).

[4] P. Zipfel, ‘Six-degrees-of-freedom simulation,’ in *Modeling and Simulation of Aerospace Vehicle Dynamics*. American Institute of Aeronautics and Astronautics, 2007, pp. 367–400 (cit. on p. 18).

[5] ——, ‘Inertial navigation system,’ in *Modeling and Simulation of Aerospace Vehicle Dynamics*. American Institute of Aeronautics and Astronautics, 2007, pp. 428–439 (cit. on p. 22).

[6] M. Franze, ‘Shefex ii - a first aerodynamic and atmospheric post-flight analysis,’ in *AIAA Atmospheric Flight Mechanics Conference*. DOI: `10.2514/6.2016-0786`. eprint: `https://arc.aiaa.org/doi/pdf/10.2514/6.2016-0786`. [Online]. Available: `https://arc.aiaa.org/doi/abs/10.2514/6.2016-0786` (cit. on p. 32).

[7] S. Niskanen, *Development of an open source model rocket simulation software*, master thesis, 2009 (cit. on p. 33).

[8] ‘Team 35 project technical report for the 2018 irec,’ Project report, 2018 (cit. on p. 34).

[9] ‘Team 54 project technical report for the 2019 irec,’ Project report, 2019 (cit. on p. 34).

[10] B. Braun, J. Barf and M. Markgraf, ‘Integrated navigation using mems-based inertial, gps and sun vector measurements aboard the spin-stabilized pmwe-1 sounding rocket,’ 8TH EUROPEAN CONFERENCE FOR AERONAUTICS and SPACE SCIENCES, 2019 (cit. on p. 36).

[11] U. KAYASAL, *Modeling and simulation of a navigation system with an imu and a magnetometer*, master thesis, 2007 (cit. on p. 36).

[12] S. M. Bezick, A. J. Pue and C. M. Patzelt, *Inertial Navigation for Guided Missile Systems*, 4. 2010, vol. 28, pp. 331–342 (cit. on p. 37).

[13] B. Graf, *Quaternions and dynamics*, 2008. arXiv: `0811.2889 [math.DS]` (cit. on p. 37).

[14] R. E. KALMAN, ‘A new approach to linear filtering and prediction problems,’ *Journal of Basic Engineering*, vol. 80D, pp. 33–45, 1960 (cit. on p. 39).

[15] C. S. Alexander, ‘Dynamic response of a carbon fiber - epoxy composite subject to planar impact,’ 15TH EUROPEAN CONFERENCE ON COMPOSITE MATERIALS, 2012 (cit. on p. 41).

[16] P. Listov and C. Jones, ‘Polympc: An efficient and extensible tool for real-time nonlinear model predictive tracking and path following for fast mechatronic systems,’ 2019 (cit. on p. 45).

[17] ‘Sequential quadratic programming,’ in *Numerical Optimization*. New York, NY: Springer New York, 2006, pp. 529–562, ISBN: 978-0-387-40065-5. DOI: 10.1007/978-0-387-40065-5_18 (cit. on p. 45).

[18] H. Tsien and R. C. Evans, ‘Optimum thrust programming for a sounding rocket,’ *Journal of the American ROCKET Society*, vol. 21, no. 5, pp. 99–107, 1951 (cit. on p. 45).

[19] A. Jnini and C. Chetcuti, ‘Developement d’une stratégie de controle pour les airbrakes de la fusée de la rocket team,’ Semester project, 2019 (cit. on p. 46).

[20] M. S. Francis, ‘Air vehicle management with integrated thrust-vector control,’ *AIAA Journal*, vol. 56, no. 12, pp. 4741–4751, 2018. DOI: 10.2514/1.J056768 (cit. on p. 56).

[21] T. Raziye, *Design, modeling, guidance and control of a vertical launch surface to air missile*, master thesis, 2010 (cit. on p. 56).

[22] L. Blackmore, ‘Autonomous precision landing of space rockets,’ *National Academy of Engineering 'The Bridge on Frontiers of Engineering'*, vol. 4, no. 46, pp. 15–20, 2016 (cit. on p. 56).

[23] T. Grosgurin, ‘Introduction to cfd for supersonic flight,’ Semester project, 2019 (cit. on p. 57).

[24] Tripoli Safety Codes and policies, *Tripoli rocketry association safe launch practices*, 2013 (cit. on p. 58).

[25] M. Sagliano, T. Tsukamoto and J. A. Macés-Hernández, ‘Guidance and control strategy for the callisto flight experiment,’ 8TH EUROPEAN CONFERENCE FOR AERONAUTICS and AEROSPACE SCIENCES (EUCASS), 2019 (cit. on p. 59).

[26] R. Linsen, ‘Thrust vector control of a small-scale rocket prototype,’ Semester project, 2021 (cit. on pp. 64, 66, 73).

[27] ‘Fundamentals of unconstrained optimization,’ in *Numerical Optimization*. New York, NY: Springer New York, 2006, pp. 26–27, ISBN: 978-0-387-40065-5. DOI: 10.1007/978-0-387-40065-5_18 (cit. on p. 64).

[28] B. RMILI, D. MONCHAUX and O. BOISNEAU, ‘Frog, a rocket for gnc demonstrations: Firsts flights attempts of the frog turbojet version and preparation of the future mono-propellant rocket engine,’ 8TH EUROPEAN CONFERENCE FOR AERONAUTICS and AEROSPACE SCIENCES (EUCASS), 2019 (cit. on p. 73).

[29] T. LaMarca, 'Jet vanes characterization and optimization,' Semester project, 2021 (cit. on p. 78).

[30] A. Muller, 'Tvc test bench semester project,' Semester project, 2021 (cit. on p. 79).

[31] U. Qayyum, 'Thermal cfd analysis of thrust vector control jet vanes & materials characterization,' Semester project, 2021 (cit. on p. 79).