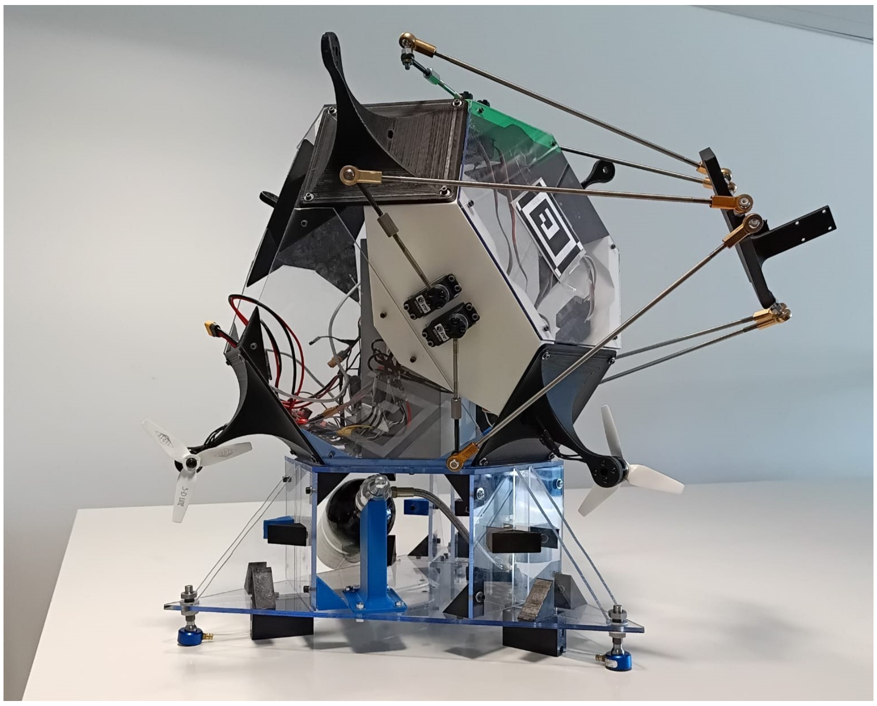


# Acrobat: A Free-flyer for In-Orbit Additive Manufacturing

## Group Number

| | |
|---|---|
| Alexandre Luís Rocha | - 95767 |
| Lucas Gonçalves | - 99519 |
| David Valente | - 103212 |
| Lourenço Gouveia Faria | - 103354 |
| Inês Lobato Mesquita | - 103645 |
| Afonso Lança | - 103838 |

Report elaborated for the Curricular Unit

## Project in Aerospace/Eletrotechnical Engineering

Supervisors: Prof. Rodrigo Ventura
Prof. Luís Caldas de Oliveira
Rafael Cordeiro

**June 2024**

## Executive Summary

Acrobat is a proof of concept designed to assess the viability of in-orbit additive manufacturing. In its zero gravity configuration designed for operation in aeriform environments, the robot has six propellers oriented at the vertices of an octahedron giving it six Degrees of Movement (DoM). We used a Stewart Platform with six Degrees of Freedom (DoF) mounted on the robot with a filament extruder attached as an end effector for 3D printing. On Earth, we tested the robot on an air-bearing base which allowed the robot to move over the test table with virtually no friction. It used only the three propellers whose axes lie parallel with the table top. Our intention was to use the robot's depth camera and externally mounted camera with accelerometer data in a Model Predictive Control (MPC) approach to control the robot's movement over the table and achieve position stabilization. However, after many setbacks we managed limited testing of the control system in an open loop configuration. From the data generated, we provide an analysis of the open loop's performance for different movement types and directions. We also implemented inverse kinematics and developed movement methodology for the parallel manipulator originally intended to demonstrate FDM 3D printing. Despite not meeting this final, ambitious goal, we still managed to make calibration algorithms, an adapted printing system, and a methodology for how the two systems would function together.

# Contents

# List of Tables

# List of Figures

# Abbreviations

| | |
|---|---|
| 3D | 3 Dimensional |
| CAD | Computer Aided Design |
| DoF | Degrees of Freedom |
| DoM | Degrees of Movement |
| ESA | European Space Agency |
| ESC | Electronic Speed Controller |
| FCU | Flight Controller Unit |
| FDM | Fused Deposition Modeling |
| FOV | Field of View |
| GPIO | General Purpose Input/Output |
| HAT | Hardware Attached on Top |
| IMU | Inertial Measurement Unit |
| ISR | Institute for Systems and Robotics |
| ISS | International Space Station |
| MIMO | Multiple Input Multiple Output |
| MIT | Massachussetts Institute of Technology |
| MOSFET | Metal-Oxide-Semiconductor Field-Effect Transistor |
| MPC | Model Predictive Controller |
| NASA | National Aeronautics and Space Administration |
| PID | Proportional Integral Derivative |
| PLA | Polylactic Acid |
| PM | Parallel Manipulator |
| PWM | Pulse-Width Modulation |
| ROS | Robot Operating System |
| RPS | Rotations per second |
| SO3 | Smart Orbiter V3 |
| SSH | Secure Shell |

# Nomenclature

| | |
|---|---|
| $A$ | Actuation matrix |
| $C_P$ | Coefficient of power |
| $C_T$ | Coefficient of thrust |
| $D$ | Propeller diameter |
| $F$ | Force |
| $I$ | Inertial momentum |
| $K$ | Constant for RPS-to-thrust conversion |
| $M$ | Momentum |
| $N$ | Controller horizon size |
| $R$ | Rotation matrix |
| $T$ | Torque |
| $\Delta t$ | Time interval |
| $\alpha$ | Angular velocity |
| $\omega$ | Angular velocity |
| $\rho$ | Air density |
| $\theta$ | Orientation |
| $a$ | acceleration |
| $d$ | Distance between geometric center and propellers |
| $kg$ | Kilograms |
| $m$ | Mass |
| $m$ | Meters |

|  |  |
|---|---|
| $p$ | Position |
| $s$ | Seconds |
| $u$ | Input vector |
| $v$ | Velocity |

# Introduction

Free-flyers are autonomous robotic systems designed to operate and navigate independently in microgravity environments such as the International Space Station (ISS) or in orbit. They are untethered and do not rely on external rails or supports for movement. This intrinsic flexibility and autonomy makes them appealing for complex tasks, like assisting astronauts and manipulating objects, or high-risk tasks, like maintenance and object deflection.

The aim of this project is to answer the question: Can we build in space? In-space construction has innumerous benefits and implications for both the space industry and industries on Earth. By providing a proof of concept of a free-flyer adapted to testing on Earth and performing additive manufacturing, we hope to demonstrate that this powerful prospect has real merit.

## 0.1 Motivation

Current space systems and infrastructure have to undergo the rocket launch environment which imposes mass, volume, and robustness stipulations. Even with the increasingly reduced cost of sending material into space, parts made to withstand launch forces will be significantly heavier than those made only to resist small in-orbit operational forces. Moreover, in space parts are no longer constrained by the launch bay. The ability to manufacture and assemble structures in orbit provides more efficient resource allocation and enables larger, theoretically infinite, structures to be built.

The implications are obvious for the space industry, but far more extensive for Earth. Enabling larger, better successors to instruments like the James-Webb Telescope, building larger space stations for research, or establishing refuelable supply depots in orbit could be revolutionary for the space industry, though not directly tangible on Earth. Emerging areas like tissue engineering benefit immensely from access to zero gravity environments. Assuming this area continues to advance, the ability to construct manufacturing facilities in space could alleviate organ supply issues. Space construction also helps other industries like semiconductors, pharmaceuticals, and materials, among others, essentially allowing manufacturers to maximize or minimize parameters where they cannot within Earth's gravity. Whether it be purity, strength, thinness, or weight, the potential effects for a wide variety of industries could have a larger impact for the world than the advent of reusable rockets brought on by SpaceX.

## 0.2 Objectives

Acrobat has two distinct parts for broad-limitless and fine-limited movements: the free-flyer and the manipulator, respectively. As such, our objectives are divided between the two parts as well as where they overlap.

### Free-flyer

- Move to a predetermined location with centimetric precision
- Avoid obstacles when navigating
- Anchor at a predetermined location

### Manipulator

- Calibrate using external and onboard information
- Move with sub-millimetric precision
- Track markers using an onboard camera

### Full System

- Reject perturbations caused by movement of the manipulator
- Print simple geometry such as a line or circle
- Follow and align towards a marker

## 0.3 Document Outline

Much like the objectives, the document is divided into three distinct parts covering the background, methodology, and results of the free-flyer and manipulator systems in parts one and two respectively. In part three, two sections cover methodology and results for the full robot.

# Part I

# Free-Flyer

# Chapter 1

# Background

Free-flyers have been theorized since the 1960s by visionaries such as Krafft Ehricke for tasks like maintenance or astronaut assistance. Throughout the 80s and 90s, various free-flyers were developed for testing simple docking, rendezvous, and maneuvering tasks. In the early 2000s, MIT ran the *MIT SPHERES* project that developed formation flying and navigation control systems [1]. Results from the project, due to its considerable complexity, laid the groundwork for performing sophisticated tasks.

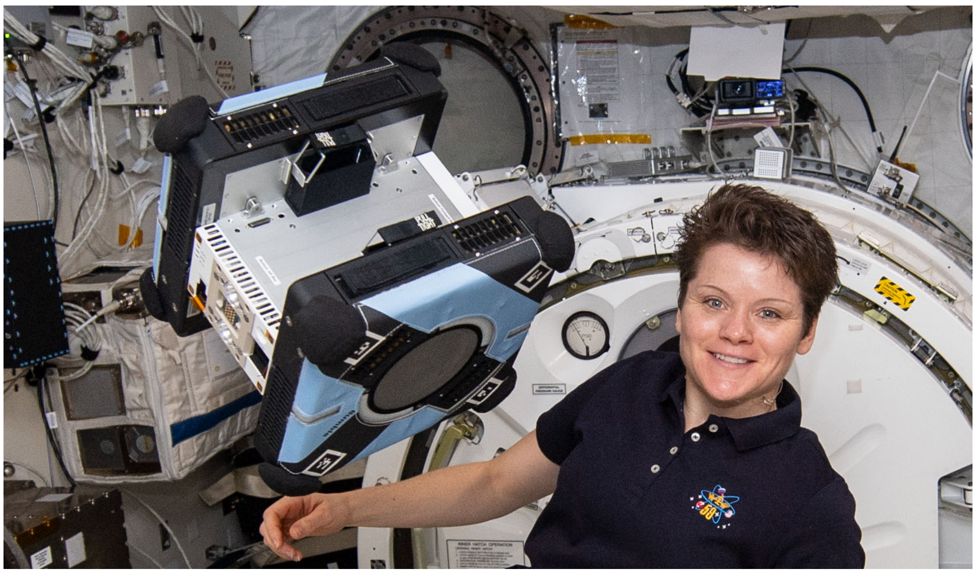


Figure 1.1: Astrobee next to astronaut Anne McClain

Through the culmination of previous free-flyer development, *Astrobee* was the first free-flyer to have a manipulator integrated into its design and used directly for the robot's function to manipulate objects and stabilize itself to preserve energy [2]. Subsequently, *Space Cobot*, Acrobat's sister project developed at ISR, improved on some attributes such as maneuverability and control methodologies still within the same guise of human assistance [3]. Rather than using an arm to stabilize itself, Space Cobot is currently developing a controller through a model predictive controller (MPC) approach to estimate not only its own inertia, but the inertia of whatever it is interacting with. This is crucial for free-flyers, as we discuss on section 2.2.4, to provide them with true autonomy and capability in environments where they may not have a larger structure to depend on.

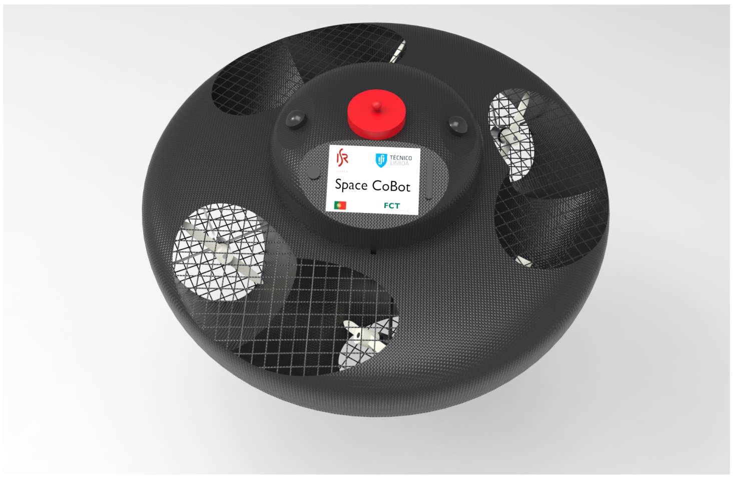


Figure 1.2: Rendering of Space Cobot

## Acrobat

Acrobat began its development with Prof. Rodrigo Ventura recruiting Alexandre Rocha to provide assistance with mechanical design, prototyping, and development work, and later on João Vale to do his thesis focused on multi-objective optimization and kinematics. All figures below show Acrobat versions in their space operation configurations.

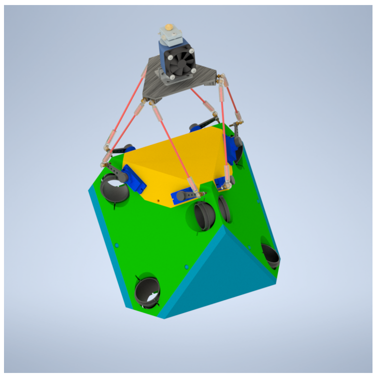


Figure 1.3: First version

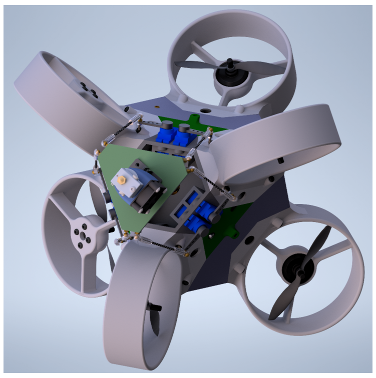


Figure 1.4: Second version made with João Vale

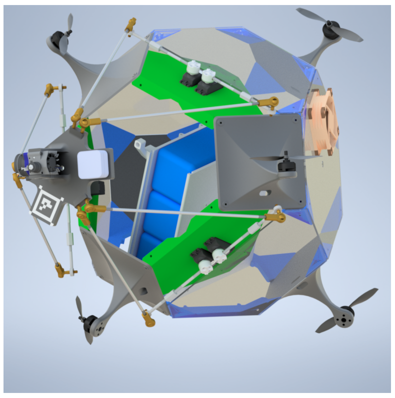


Figure 1.5: Third version for LEAer/LEEC capstone project

Figure 1.3 shows a first mock-up designed in *Autodesk Inventor* CAD software which, like all subsequent versions, was done to get an idea of how the geometry suggested by Prof. Ventura would be applied in practice. After realizing the first version's mechanical limitations and applying other optimization criteria, in collaboration with João Vale, we designed, partially constructed, and tested the second version shown in figure 1.4. These tests were more focused on the manipulator, which we discuss in part II. However, much of the dynamics characterization demonstrated in this part is similar to that found in João Vale's thesis[4]. Finally, figure 1.5 depicts the latest version of Acrobat which we discuss in the following section (2).

# Chapter 2

# Methodology

## 2.1 Mechanics and Dynamics

The geometry we selected for Acrobat's movement is designed to provide optimal force and moment generation for effective movement and stabilization in all directions. Since it is designed to operate in zero gravity environments, it is expected to actuate in bursts. This type of actuation is best suited to cold gas thrusters which release controlled bursts of pressurized gas at known thrusts. Despite this, when considering the complexity of designing a pneumatic system suitable for their operation, we chose to use bidirectional thrust propellers powered by brushless motors instead for their simplicity and widespread adoption in robotics when prototyping.

To make Acrobat easy to manufacture and assemble, a modular design approach was taken. As has been and will be seen throughout the document, Acrobat contains a lot of brightly colored parts. These are fabricated using Fused Deposition Modeling(FDM) 3D printing on a *Bambu Lab X1C* 3D printer. The vast majority of these parts contain brass threaded heat set inserts used instead of nuts to make strong, rigid connections.

The transparent parts are made of $3\,\mathrm{mm}$ polycarbonate sheets cut on a table saw. They are held together by several light blue brackets printed in PLA to form the frame on which other parts can be mounted. These include the black parts, printed in carbon fiber PLA for added strength and rigidity, which serve as mounts for the brushless motors as well as the green parts once again printed in PLA to serve as the base for the parallel manipulator later discussed in chapter 5.

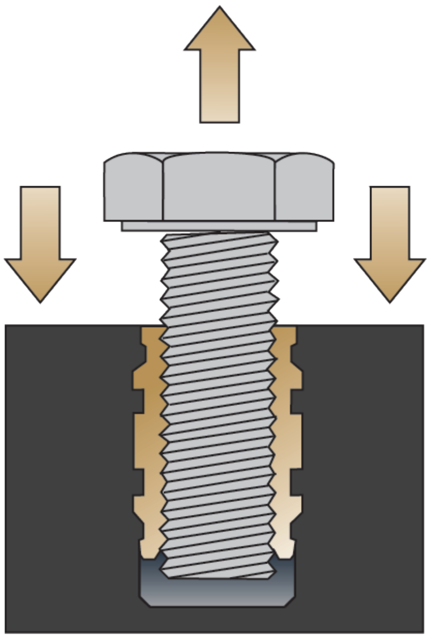


Figure 2.1: Illustration of a threaded heat set insert being used with a bolt

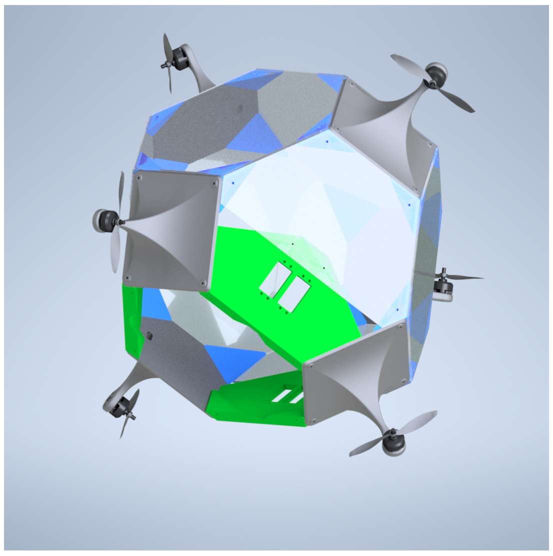


Figure 2.2: Rendering of Acrobat's free-flyer body

This voluminous construction gives use the freedom to configure and reconfigure the interior of the robot as necessary only having to replace a few parts in case we need to change electronics on the interior or change where the filament spool is held. During prototyping, in many cases it was as easy as drilling a few new holes. The only downside to this construction method is that the robot is only rigid when all the parts are mounted together and tightened. However, we feel remounting panels removed for access is a worthy trad-off as opposed to redesigning an entire frame from the ground up.

### 2.1.1 Space Configuration

The robot's six DoM are derived from bidirectional thrust axes situated at the vertices of an octahedron. Axis positions and orientations effectively give Acrobat equal movement in X, Y, and Z and equal orientation capabilities roll, pitch, and yaw. Their distance from the geometric center also allows them to quickly produce torques for resisting perturbations caused externally or from the robot's own operation.

We can characterize this system with matrices:

$$\begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{M} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a_1} & \dots & \boldsymbol{a_6} \end{bmatrix} \begin{bmatrix} q_1 \\ \vdots \\ q_6 \end{bmatrix} = \boldsymbol{Aq}, \tag{2.1}$$

where net body forces and torques are a result of actuation $\boldsymbol{a}$ and actuation signal $q$. Actuation matrix, $\boldsymbol{A}$, is a square matrix, meaning the robot is neither under nor over actuated, and $\boldsymbol{A}$ is full rank, meaning all propellers are non-redundant. Thus, $\boldsymbol{A}$ is invertible . This property has favorable implications for the robot's control[5]. However, this project's aim is to provide a practical proof of concept rather than a theoretical analysis. As such, the dynamics on which we focused our efforts are those adapted to ground testing.

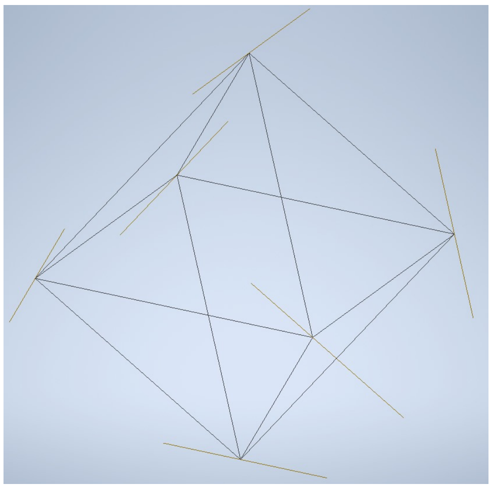


Figure 2.3: Octahedron with bi-directional thrust axes at vertices

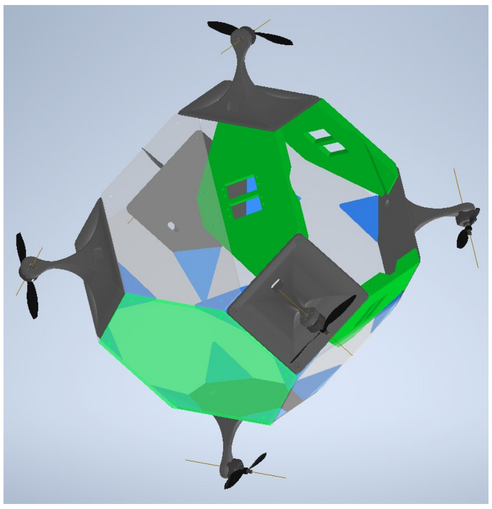


Figure 2.4: Space configuration of Acrobat

### 2.1.2 Ground Testing Configuration

It is apparent, when looking at Acrobat, that it cannot hold its own weight in flight when exposed to Earth's gravity. Thus, for ground testing, we use an air bearing base, constructed with $5\,\mathrm{mm}$ polycarbonate sheets and brackets with heat set, inserts that allows the robot to glide freely over a large test table.

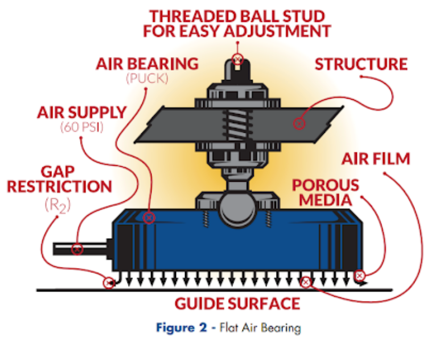


Figure 2.5: Illustration of air bearing in operation[6]

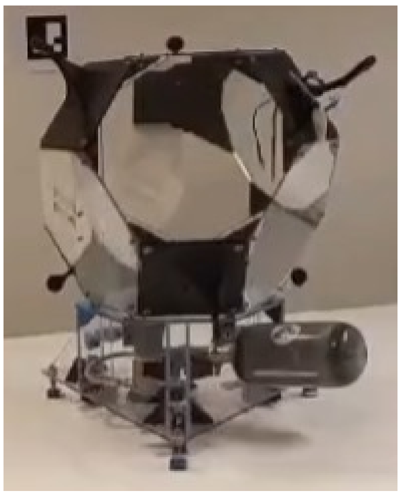


Figure 2.6: Acrobat on air bearing base

As shown in figure 2.5, air bearings work by pushing a thin layer of air, usually 5 to $15\,\mu\mathrm{m}$, between themselves and a smooth guide surface. This allows for frictionless motion of objects with substantial weight. Figure 2.6 depicts our setup, which uses a $1.1\,\mathrm{L}$ paintball tank holding air at $300\,\mathrm{bar}$ converted by two regulators to about $5\,\mathrm{bar}$ and expelled through three $25\,\mathrm{mm}$ flat, round air bearings. This base can last nearly one and a half hours of consecutive operation, allowing adequate time for testing.

Since the free-flyer glides over a table, in this configuration it only possesses 3 DoM: translation over the table plane (X and Y) and yaw (rotation about the Z axis). Conveniently, due to the way the axes have been positioned, there are three planes where the bi-directional thrust axes are parallel to the free-flyer's own body plane. By simply aligning one of these planes with the plane of the table, similar to how six propellers give full

6 DoM control, three propellers give full, direct control over two dimensional motion.

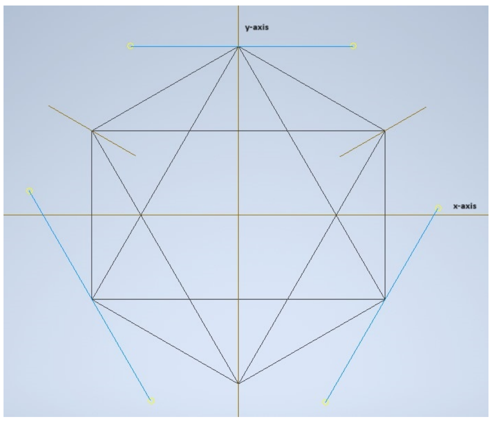


Figure 2.7: Top view of highlighted axes with base geometry aligned with the movement plane

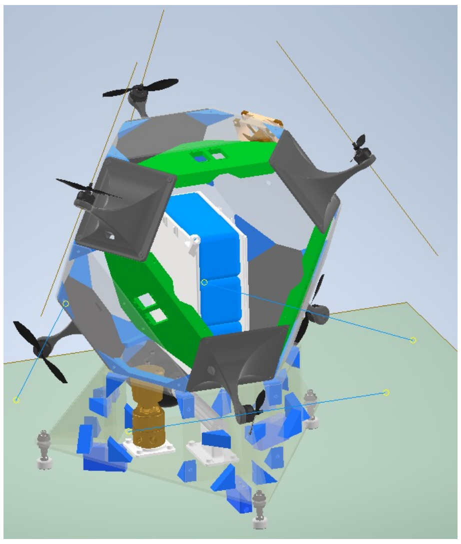


Figure 2.8: Aligned axes and table plane shown in modeled testing configuration

### 2.1.3 System Dynamics

Knowing how the system has been adapted for testing on Earth, we now characterize its dynamics. Figure 2.7 shows that the plane aligned with the test table is an equilateral triangle with angles $\alpha_i$ that represent the orientation of the propellers and their highlighted thrust axes, with respect to the body frame. Assuming the center of mass coincides with the geometric center of the free-flyer, we define the actuation matrix as follows:

$$\boldsymbol{A} = \begin{bmatrix} K_1 \sin \alpha_1 & K_1 \sin \alpha_2 & K_1 \sin \alpha_3 \\ K_1 \cos \alpha_1 & K_1 \cos \alpha_2 & K_1 \cos \alpha_3 \\ K_1 d & K_1 d & K_1 d \end{bmatrix} \tag{2.2}$$

where d is the distance between the geometric center and the center of each propeller. From the geometry discussed earlier, $\alpha_1 = 0\,\text{rad}$, $\alpha_2 = \frac{2\pi}{3}\text{rad}$ and $\alpha_3 = \frac{4\pi}{3}\text{rad}$ with respect to the body frame x-axis. $K_1$ provides a relation between the thrust generated by a propeller and the appropriate PWM signal.

To describe how this actuation translates to the test table, we now define the dynamic system. Let $p \in \mathbb{R}^2$ and $\theta \in \mathbb{R}^1$ be the position and orientation of the free-flyer, $v \in \mathbb{R}^2$ and $\omega \in \mathbb{R}^1$ be its linear and angular velocities, $R \in \mathbb{R}^{3\times3}$ be its attitude expressed as a rotation matrix, and $F \in \mathbb{R}^2$ and $M \in \mathbb{R}^1$ be the force and torque in the body frame. Again considering that the center of mass and geometric center are coincident, the dynamic system is as follows:

$$\begin{cases} m\dot{v} = R_F(\theta)A_F u(t) = R_F(\theta)F_b \\ I_{zz}\dot{\omega} = A_M u(t) \\ \dot{p} = v \\ \dot{\theta} = \omega. \end{cases} \tag{2.3}$$

Here $\boldsymbol{u}$ is a vector of dimension 3, and each argument is the PWM signal sent to its respective propeller.

As seen in equation 2.2, the actuation matrix can be separated into $A_F$ and $A_M$, the forces and momentum applied to the robot by each propeller. $F_b$ denotes the forces applied in the body frame.

The force that each propeller applies to the robot can be described in the following way [7]:

$$\overline{F}_i = f_i \hat{u}_i \text{ , with } f_i = K_1 u_i, \tag{2.4}$$

where $\hat{u}_i$ is a unit vector that separates the force into X and Y components.

The torque generated can be described as the cross product of distance between the center of the propeller and geometric center with force as well as a term considering gyroscopic effects induced by the propeller's rotation:

$$\overline{M}_i = \overline{r}_i \times \overline{F}_i - \tau_i \hat{u}_i \text{ , with } \tau_i = \omega_i K_2 u_i. \tag{2.5}$$

The $\times$ operator denotes a vector cross product. Because the thrust axis, the axis of rotation, is parallel to the plane of movement, a good approximation is that the gyroscopic effects are described by $\tau_i \approx 0$.

$\hat{u}_i$ is a unit vector with the following structure:

$$\hat{u}_i = \begin{bmatrix} \sin\alpha \\ \cos\alpha \end{bmatrix}$$

It is useful to deduce the actuation matrix presented in equation 2.2.

To convert the forces from body frame to test table reference, the following rotation matrix may be used:

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

Because the torque generated is the same in both referentials, the resultant rotation matrix for forces and momentum is:

$$\boldsymbol{R}_\theta = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.6}$$

Another way the system described in 2.3 can be represented is as follows:

$$\begin{cases} \begin{bmatrix} F_{2\times1} \\ M_{1\times1} \end{bmatrix} = R_{3\times3} A_{3\times3} u_{3\times1} \\ \begin{bmatrix} \dot{p}_{2\times1} \\ \dot{\theta}_{1\times1} \end{bmatrix} = \begin{bmatrix} v_{2\times1} \\ \omega_{1\times1} \end{bmatrix} \end{cases}. \tag{2.7}$$

### 2.1.4 Propeller Dynamics

The propeller we chose for testing was the *Graupner 3D 5x3.5x3* because, to our knowledge, it was the only 5 inch propeller with a fully symmetric profile and neutral geometry making it conducive to bidirectional thrust. In addition, it had claims of high thrust-to-RPS ratio [8] making it ideal for bursts of thrust and low velocity operation.

Initially, a few static tests were performed using the RCbenchmark© software. Among all the data collected, thrust, rotation speed and PWM signals are the most important to analyze in order to design a robust control system for the robot to later develop a dynamic model for the robot. Before we can develop it, however, we must first study the dynamics of the propellers themselves. Their behaviour can be mathematically modeled through the following coefficients:

$$C_T = \frac{T}{\rho n^2 D^4} \quad (2.8) \qquad C_P = \frac{P}{\rho n^3 D^5}. \quad (2.9)$$

Here, $\rho$ is the air density, $D$ the propeller diameter, $n$ revolutions per second, $C_T$ and $C_P$ the dimensionless blade coefficients of thrust and power. We may assume $\rho$ is constant in a controlled laboratory environment. $D$ is a fixed value. The coefficients $C_T$ and $C_P$ are important for the dynamic modeling of the robot because they simplify what are otherwise complex aerodynamics.

The thrust provided by each propeller can be described using a simple, linear function where variables $K_1$ and $K_2$ are defined as follows:

$$K_1 = \rho D^4 C_T \quad (2.10) \qquad K_2 = \frac{\rho D^5}{2\pi} C_P. \quad (2.11)$$

We believe this simplification is valid for our use case because thrust will be generated in short bursts. By substituting these variables into the propeller equations, a relation between thrust and $n^2$ of the propeller can be established:

$$T = C_T \rho n^2 D^4 = K_1 n^2 \quad (2.12) \qquad P = C_P \rho n^3 D^5 = 2\pi K_2 n^3. \quad (2.13)$$

To obtain the values of $C_t$ and $C_p$, a linear regression of thrust as a function of rotations per second squared can be performed. The result is a slope that describes the linear relations $\frac{T}{n^2}$ and $\frac{P}{n^3}$. Dividing the slopes by $\rho n^2$ and $\rho n^3$ respectively, we obtain the value of both coefficients.

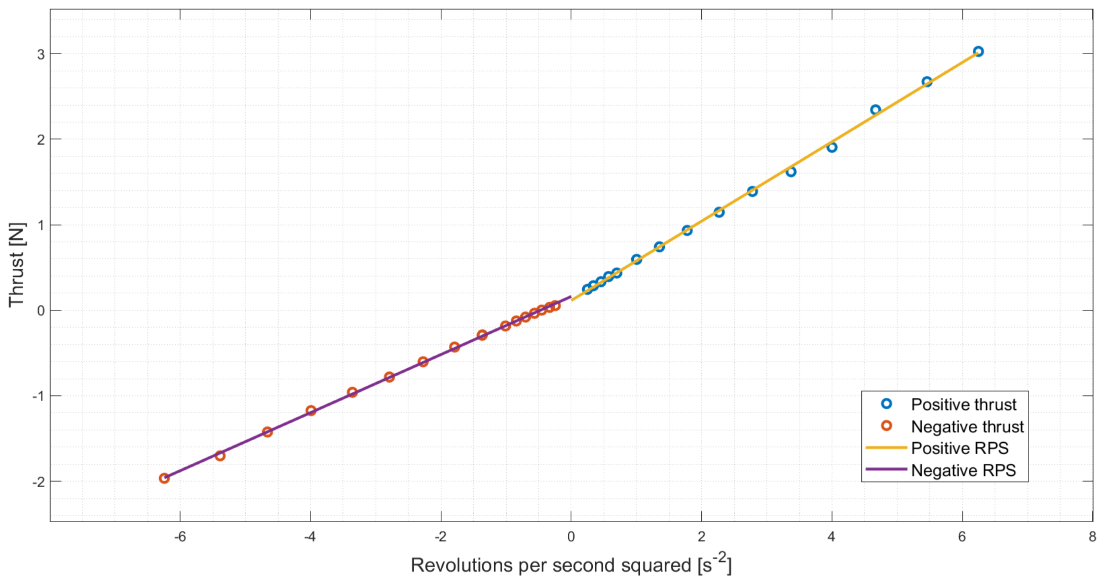


Figure 2.9: Thrust as a function of of rotating speed squared, obtained in MATLAB®

The plot in figure 2.10 was obtained by plotting the thrust provided by the propeller as a function of the revolutions per second squared of the propeller blades. As shown, there is a linear relation between these variables.

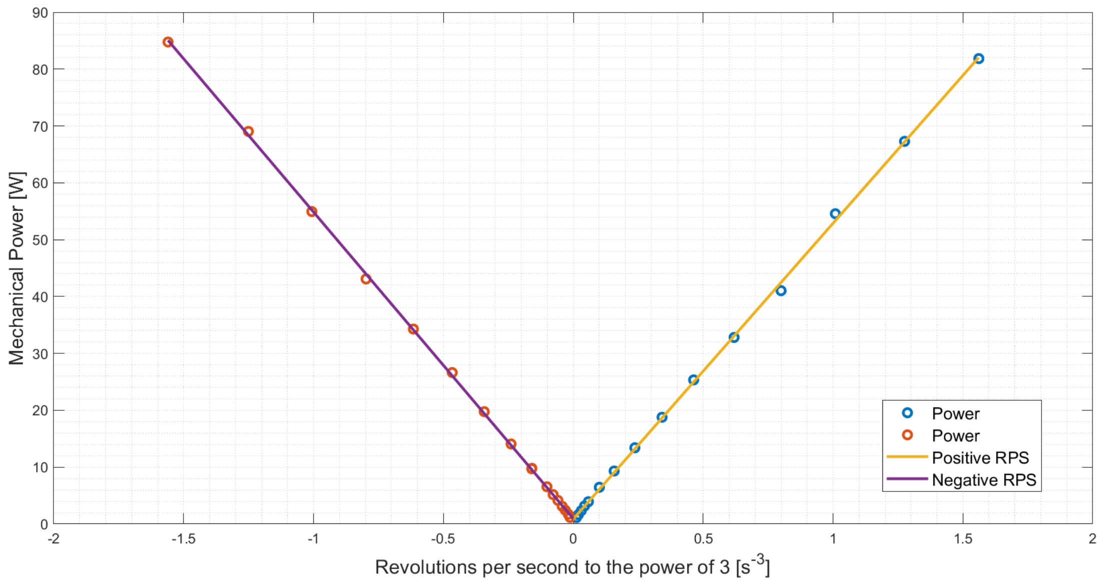


Figure 2.10: Power in function of square of rotating speed, obtained in MATLAB®

It is important to note that the blades generate different values of thrust for the same revolutions per second when spinning in opposite direction, meaning they are not perfectly bidirectional. By using the slopes obtained in the linear regression and applying formula 2.8, the coefficient of thrust for each rotation direction is obtained.

| $C_{T_{pos}}$ | $C_{T_{neg}}$ |
|---|---|
| $6.17$ | $-4.51$ |

Table 2.1: Coefficients of thrust for clockwise and anti-clockwise rotations

Here we assume the local atmosphere to be at 101.325 $kPa$ and 20 $°C$ and, thus, $\rho$ is 1.204 $kgm^{-3}$ following the standard atmosphere model. The propeller's diameter is 5 $cm$ and the constant $K_1$ for both clockwise and anti-clockwise rotations is obtained through equation 2.10.

| $K_{1_{pos}}$ | $K_{1_{neg}}$ |
|---|---|
| $4.64 \times 10^{-5}$ | $-3.40 \times 10^{-5}$ |

Table 2.2: $K_1$ for clockwise and anti-clockwise rotations

By analysing these values, we conclude that when the blades are rotating clockwise, there is a generation of $34\%$ more thrust in this direction when compared to the anti-clockwise direction for the same revolutions per second squared. The propellers will receive PWM signals and, therefore, if the dynamics of the system are designed to work with the thrust provided by each propeller, we deduce a function that converts these values to their respective pulse-width modulation signal. We estimated two polynomials up to order 4 for the positive and negative forces applied by the propeller. We made this separation due to the non-presence of a reasonably good bidirectional thrust for the same rotation velocity. See table 2.6.

| | Polynomial order | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| MSE | Clockwise rotation | 101.79 | 11.26 | 1.62 | 1.48 |
| | Anti-clockwise rotation | 87.61 | 19.63 | 2.29 | 1.34 |

Table 2.3: Mean squared error of the fitting polynomials up order 4 of PWM signals as a function of thrust provided by each propeller

It is important to note that MATLAB® indicates that the polynomial becomes badly conditioned when approximated to order 4. Therefore, the optimal order is order 3, where the mean squared error of the approximation is reasonably small.

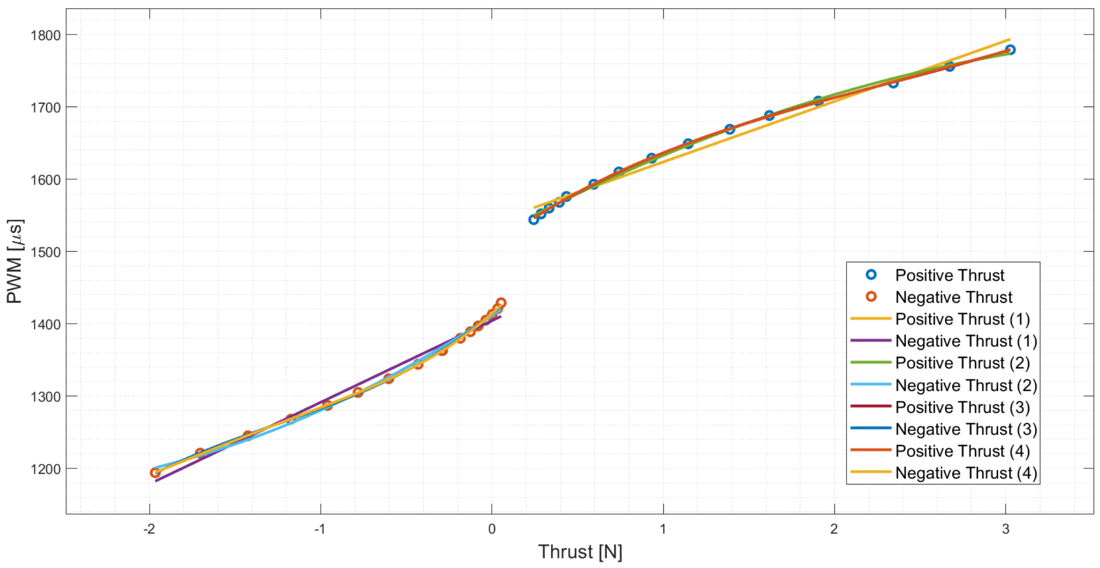


Figure 2.11: PWM signal as a function of thrust, obtained in MATLAB®

## 2.2 Control and Navigation

### 2.2.1 Navigation

The navigation protocol used was based on a mix between data acquisition from ArUco markers, using vision algorithms, and the *Pixhawk 6C Mini* systems. Position and orientation are extracted from a relation between the ArUcos that map the environment and an ArUco placed on the robot. This ArUco system will be described in great detail in a later section (5.6). Regarding the Pixhawk, acceleration is obtained through its accelerometer and velocity by an integration that is made by one of its navigation subsystems.

### 2.2.2 Vision

The system that consists in the ceiling camera reading 2 ArUco markers and determining the position of the one affixed on the robot in the reference frame created by the one placed on the table is calibrated the same way as the one explained in the 5.6. In order to recreate that system, the methodology described in that section should be followed.

### 2.2.3 Open-loop test

To test the actuation of the propellers in open-loop, the control vector $u(t)$ can be isolated as follows:

$$u_{3\times1}(t) = A_{3\times3}^{-1} R_{3\times3}(\theta)^{-1} \begin{bmatrix} F_{2\times1} \\ M_{1\times1} \end{bmatrix}.$$

The value of force (in N) or momentum (in Nm) can be requested independently, as well as in a combination of forces or force and momentum. This open-loop test allows us to verify if the robot accelerates in a desired direction and if it is able to correctly rotate along its geometric central axis.

### 2.2.4 Model predictive controller

The optimal controller for the propellers is an MPC because it can handle multi-input multi-output (MIMO) systems that have interactions between their inputs and outputs. Due to these interactions, it is often challenging to design MIMO systems using traditional controllers such as PID. However, MPC can simultaneously control all the outputs while taking into account input-output interactions. MPC can also handle constraints. Constraints are important, as violating them may lead to undesired consequences. MPC has preview capabilities (similar to feed-forward control). If set point changes are known in advance, the controller can better react to those changes and improve its performance. To develop an MPC for Acrobat, we opted to use *CasADi* open-source software as a tool for nonlinear optimization and algorithmic differentiation. The *opti* tool from this package came as very useful for this goal.

#### System definition

To establish a mathematical relation between the forces and momentum provided by the propellers to the change in position and orientation, we must use simple physics. The effect of the force in the position of the robot is described by the following simple kinematic formulas:

$$F_{world}(i) = m \times a(i),$$

$$a(i) = \frac{v(i) - v(i-1)}{\Delta t}, \text{ and}$$

$$v(i) = \frac{p(i) - p(i-1)}{\Delta t},$$

where $m$ corresponds to the mass of the robot and $\Delta t$ to the time interval between each iteration.

The same thing happens with the effect of the momentum provided by the propellers in the orientation of the robot:

$$M_{world} = J \times \alpha(i),$$

$$\alpha(i) = \frac{\omega(i) - \omega(i-1)}{\Delta t}, \text{ and}$$

$$\omega(i) = \frac{\theta(i) - \theta(i-1)}{\Delta t},$$

where $J$ is the inertial momentum of the robot in its vertical axis aligned with its geometrical center.

**Robot parameters and controller horizon**

The robot and environment parameters are defined as follows:

| m $[kg]$ | J $[kg\,m^2]$ | $\rho$ $[kg\,m^{-3}]$ | D $[m]$ |
|---|---|---|---|
| $16.341$ | $0.394$ | $1.204$ | $5 \times 10^{-5}$ |

Table 2.4: Robot and environment parameters for simulation

It it is important to keep in mind that the propellers do not generate perfect bidirectional thrust.

The actuation matrix 2.2 has the numerical value of:

$$A = \begin{bmatrix} 0 & 0.866 & -0.866 \\ -1 & 0.5 & 0.5 \\ -0.3 & -0.3 & -0.3 \end{bmatrix}$$

$K_1$ is implicitly defined as singular and only applied once the thrust provided by each propeller is obtained from the controller's solver optimal solution.

With all the numerical data needed to describe the robot's dynamics, the the CasADi parameters can now be defined. The controller's horizon was initialized in the following way:

| Variable | Rows | Columns |
|---|---|---|
| Position | 2 | N |
| Velocity | 2 | N |
| Total velocity | 1 | N |
| Acceleration | 2 | N |
| Orientation | 1 | N |
| Angular velocity | 1 | N |
| Angular acceleration | 1 | N |
| Forces (self) | 2 | N |
| Forces (world) | 2 | N |
| Momentum (self) | 1 | N |
| Momentum (world) | 1 | N |
| Dynamics (self) | 3 | N |
| Dynamics (world) | 3 | N |
| Rotation matrix | 3 | 3 |
| Propellers actuation | 3 | N |

Table 2.5: Horizon parameters and respective sizes

where $N$ denotes the number of iterations in the horizon.

### Controller steps

In each iteration of the MPC, the corresponding state of the robot will not match the prediction made by the controller perfectly due to unpredictable external perturbations and the non perfect design of the robot's dynamics and propeller behaviour. In order for the controller to work as a closed system, the first horizon state must be defined with the navigation data provided by the robot's sensors. The following variables are obtained through navigation data provided by the pixhawk and ArUcO markers camera algorithms:

- position (ArUcO)
- linear velocity (Pixhawk)
- linear acceleration (Pixhawk)
- orientation (ArUcO)
- angular velocity (Pixhawk)

To enable the MPC to achieve faster solutions, the remaining variables that cannot be observed by the navigation system can be estimated through the mathematical relations presented in the beginning of this section. To facilitate the solution optimization of the MPC, the other horizon steps can initially be predicted to be the value corresponding to $t+1$ from the previous horizon. In other words, the horizon shifts one coordinate to the left and the initial values are obtained and derived from navigation data. The last horizon value may be set to the penultimate value of the current horizon. Another method would be to calculate a bigger horizon before the robot starts taking action. Then, using a smaller horizon for active control of the robot, the variables from the initial horizon would be put into the last step of the real time horizon accordingly. This was not, however, the approach we chose.

### Boundaries/constraints

In order for the MPC to output realistic and usable data, a few constraints need to be assumed. Others could allow for a safer dislocation of the robot in the test table, however, soft constraints are usually more suitable and reliable. The most important boundary condition is the maximum input each propeller can take. As mentioned above, the input vector $u$ is described with the thrust provided by each propeller. The maximum value for the thrust in the benchmark test was 3 newtons in clockwise rotation and 2 newtons n anti-clockwise rotation. Therefore, the following constraint is added to the *opti* environment:

$$-2\,\mathrm{N} < u(t) < 3\,\mathrm{N}.$$

### Cost function

The cost function is the most important aspect of the controller. It can be manipulated to make the solution closer to the one desired within its horizon and in a faster manner, to reduce the amount of actuation of the propellers, and to correct undesired overshoots. To make the solver, discussed next, reach the reference values faster, the term $A_1$ may be applied. The coefficient $\rho_1$ weights this importance, as do $\rho_{1_1}$ and $\rho_{1_2}$ in position and orientation, respectively.

$$A_1(i) = \rho_1[\rho_{1_1}[(x(i) - x_{ref})^2 + (y(i) - y_{ref})^2] + \rho_{1_2}(\theta(i) - \theta_{ref})^2]$$

To achieve a smoother actuation, the input control vector can also be added to the cost function in square form, weighted by $\rho_2$.

$$A_2(i) = \rho_2[u_1^2(i) + u_2^2(i) + u_3^2(i)]$$

To minimize the occurrence of overshoots, the final positions of the horizon can be minimized in correspondence to the reference values. This condition is weighted by $\rho_3$ and we chose the square form for this.

$$A_3 = \rho_3[\rho_{3_1}(pos(N) - pos_{ref})^2 + \rho_{3_2}(pos(N-1) - pos_{ref})^2 + \rho_{3_3}(pos(N-2) - pos_{ref})^2]$$

Last but not least, to assure that the robot reaches static position by the end of the horizon, the square of the velocity can be added to the cost function, weighted by $\rho_4$.

$$A_4 = \rho_4[vel^2(N)]$$

All of these parameters can be added to the actual cost function which is minimized and later used by the solver.

$$J = \sum_1^N [A_1(i) + A_2(i)] + A_3 + A_4 \tag{2.14}$$

$ref$ denominates the desired values and $i$ the value that corresponds to each MPC-optimal solution iteration.

### Solver

The solver we chose for the optimization of the solutions was the *ipopt*. To achieve faster solutions, we defined a few parameters. The first is the solver tolerance, which we set to $1 \times 10^{-3}$. Additionally, we took out all default prints because these are very time-costly. The aim was to achieve a control system system that could be operated at 10Hz. With the current design, we achieved horizons of 10 iterations, 1 second in time, in 30ms, computationally. However, it is important to note that the upboard might not stay consistent over time due to heating and other tasks being performed by the processing unit, so rigorous testing is important to guarantee that the solver does not last more than 100ms.

### Simulation results

The cost function parameters were defined the following way:

| $\rho_1$ | $\rho_{1_1}$ | $\rho_{1_2}$ | $\rho_2$ | $\rho_3$ | $\rho_{3_1}$ | $\rho_{3_2}$ | $\rho_{3_3}$ | $\rho_4$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 299 | 19 | 100 | 1 | 999999 | 8 | 7 | 999 |

Table 2.6: Cost function parameters for simulation

For a reference of [1, 0.5] in position, 0 in orientation, and initial conditions [0, 0] meters and 1 radian, we made three plots to analyse the robot's trajectory, forces, and momentum applied, as well as the actuation of the three propellers. Note that we chose the horizon to be bigger than the one intended for practical use, with $N = 120$, equivalent to 12 seconds at 10Hz.

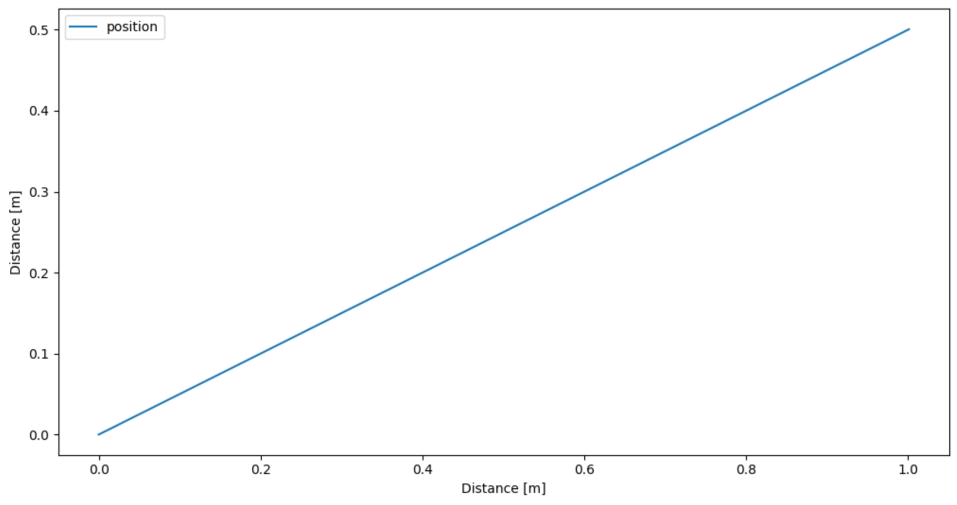


Figure 2.12: Simulation of robot's trajectory

As can be seen in figure 2.12, the MPC naturally assumes a linear trajectory between the origin and the *rendezvous* point.

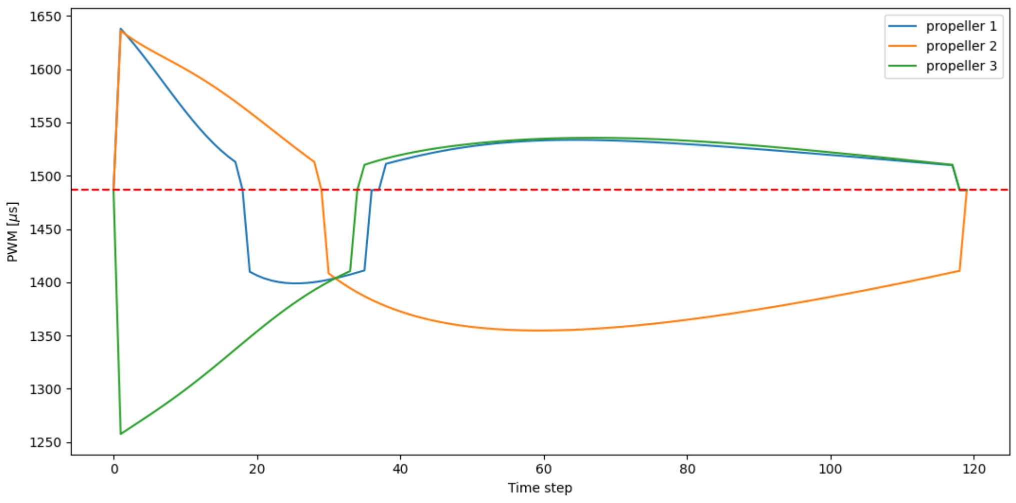


Figure 2.13: Simulation of PWM signals

The propeller actuation is very smooth and also greater in the first moments of new reference reception. The absolute value of each propeller is considerably smaller than the maximum value that the controller was modelled to perform at (between 1100 and 1800 $\mu s$ roughly) due to the parameter $\rho_2$ of the cost function.

Additionally, we note that a dead zone of operation for the propellers around a PWM signal of 1486.5 $\mu s$ is assumed. Also, as consequence of the different thrusts for the same revolutions in time in clockwise and anti-clockwise rotations, the controller tries to compensate by assuming a higher PWM signal when anti-clockwise rotation is used.

## Obstacle avoidance

The controller can be adapted to receive obstacle information and avoid a collision with it. This information is obtained from the camera that stays in the ceiling. For a simple test, boundary conditions were assumed to check the response of the robot. However, for a practical approach, it is best to heavily penalize the position of the robot once it approaches the obstacle in the cost function. Using the boundary approach, if for some reason the robot was to be found inside of the obstacle's hit box, the controller would find itself as having an unfeasible solution. The simplest way would be to set a hit-box around the object that was detected.

$$[x(i) - obstacle_x]^2 + [y(i) - obstacle_y]^2 > \text{hit-box radius}^2$$

Note that vision algorithms usually assume rectangle shaped boxes instead of circular ones. This information would be updated over time, assuring that the object's position is always up to date. In theory, the obstacle could be avoided if it were to be either in static or dynamic position. However, a simple kinematic model could be assumed for and obstacle in motion to better predict its position over time and guarantee that no collisions occur.

Below is the result by placing two obstacles at coordinates [0.75, 0.48] and [0.3, 0.15], with a position reference at [1, 0.5]. The horizon was made big enough so that it could include the desired position within its position parameters. However, in a realistic application of the MPC, the horizon would be made much smaller and most likely would not include the desired position.

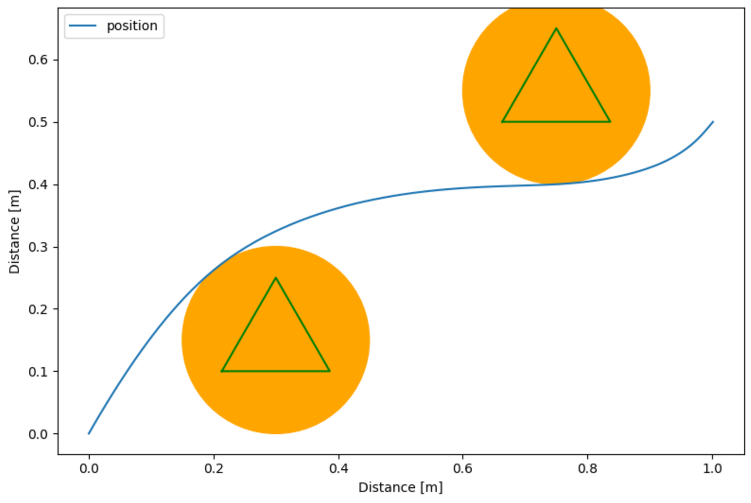


Figure 2.14: Obstacle avoidance trajectory simulation

The robot assumes a tangent trajectory to the obstacle's hit boxes. A minor slip in the navigation system would result in an unfeasible solution. Penalization in the cost function is surely the best approach to adopt.

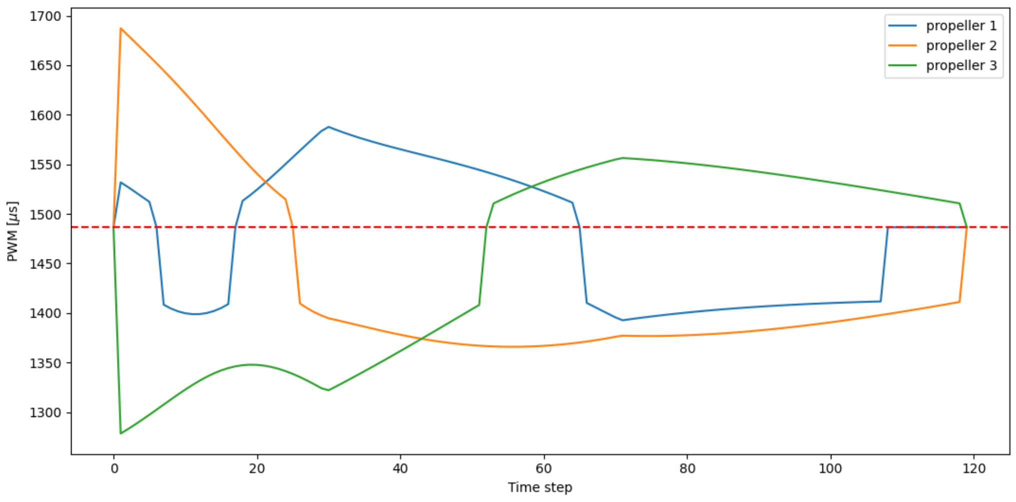


Figure 2.15: Actuation of the propellers in obstacle avoidance simulation

It is also interesting to observe the propeller behaviour in these condititons.

## 2.3 Programming and Architecture

### 2.3.1 Electronics

The electronics for the free-flyer component of Acrobat are rather simple comprising of an UpBoard serving as the onboard computer, a *Holybro PixHawk 6C Mini* serving as a flight controller unit, a high FOV external camera and three *COBRA 11A ESC W/2A LINEAR BEC* ESCs to control the *Cobra CM-2204/28 Multirotor Motor, KV=2300* brushless motors responsible for its actuation.

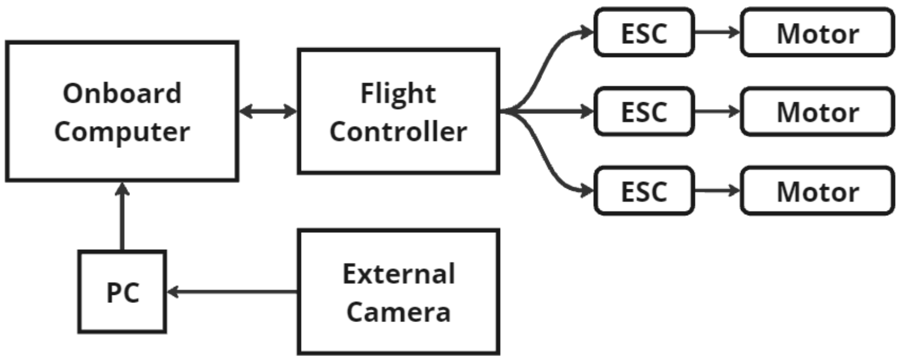


Figure 2.16: Free-flyer onboard electronics

The FCU was selected because of many reasons, being some of them the high-performance, low-noise IMUs on board, designed to be cost-effective while having IMU redundancy and a high performance STM32H743 processor. The Pixhawk series is normally selected in robotics projects because of the reliability of the software PX4, which is open-source and full of safety conditions as well as an integrated Extended Kalman Filters. The flight controller is responsible for sending PWM signals to the ESCs based on commands received by the onboard computer through SSH protocol communication with the user's computer. These commands define the amount and direction of thrust each motor provides. It also possesses an accelerometer and gyroscope which provide acceleration and orientation values to be used by the free-flyer's control system. Due to compatibility issues between the flight controller's software library and ROS2, in our final system we used the *Holybro PixHawk 6C Mini* purely as an IMU and an Arduino Uno to send PWM signals. These issues will be discussed in greater detail in subsequent sections.

The primary flight computer chosen is the *UpBoard* from Up Bridge the Gap. Its principal role is to execute control algorithms and transmit signals to all actuators. Given its compatibility with various operating systems, having integrated Intel Technology, high computational power, and user-friendly configuration and usage, it emerged as the optimal choice for the prototype. Analogous to the globally renowned Raspberry Pi, it features a 40 HAT pin with GPIO functionality. However, in our scenario, this feature was not utilized due to the lack of support from the manufacturer to the Linux distribution Ubuntu.

The brushless motors chosen, supplied by Cobra Motors USA, are recognized as some of the most reliable and efficient in the industry. The specific model used in Acrobat boasts a remarkable rate of 2300 rotations per minute per 1V difference, positioning it as one of the most powerful motors within its compact size category. It also allows for rotation in both direction, being that an important factor in the robot dynamics.

The connections in the system are represented below.

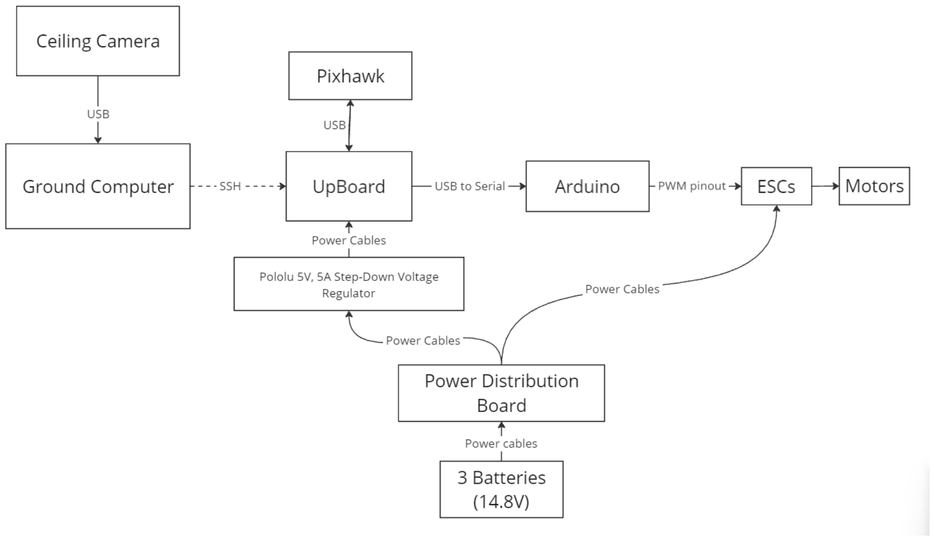


Figure 2.17: Eletrical connections

Since the UpBoard, when in the free-flyer, needs to be power by the batteries, it was needed to buy a step-down voltage. The one presented in the graph was selected because of all the fail-safe conditions already implemented.

### 2.3.2 ROS2 Architecture

ROS is a versatile framework designed to streamline the development of robotic applications. It offers a comprehensive suite of tools, libraries, and conventions that simplify the complexities of robot software development. ROS2 offers improved performance, enhanced real-time capabilities, and better support for various hardware platforms, when comparing with his predecessor, ROS. The motivation to use ROS was mainly due to its compatibility with industry-standard communication protocols and its emphasis on security and reliability.

The simplified ROS architecture used in this project is as follows:

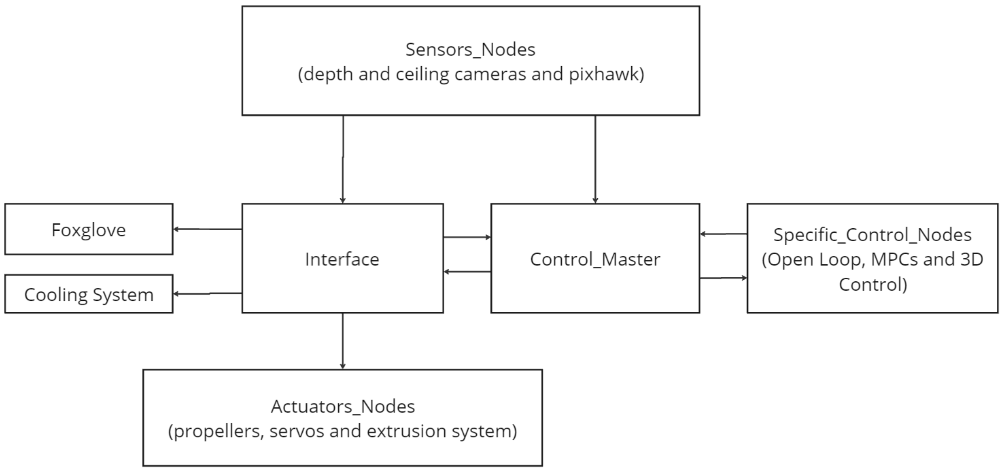

Sensors_Nodes
(depth and ceiling cameras and pixhawk)
Foxglove
Cooling System
Interface
Control_Master
Specific_Control_Nodes
(Open Loop, MPCs and 3D Control)
Actuators_Nodes
(propellers, servos and extrusion system)

Figure 2.18: ROS2 architecture flow diagram

The diagram illustrates a well-structured software architecture with four key groups of ROS2 nodes. The sensors nodes calculate positions and IMU from our sensors that are then processed in the control unit which consists of the Control Master and the Control Nodes. This processed data guides the generation of movement instructions for our actuator nodes. The interface plays a central role. It acts as a two-way communication hub: receiving user commands and transmitting them to the control unit. Importantly, the control unit transmits its calculated movement instructions to the interface, which then directly commands the actuator nodes, closing the control loop. Finally, the interface takes responsibility for managing the cooling system, turning it on and off as needed.

The ROS2 system isn't fully integrated into the on-board computer, although a majority of it operates there. However, the data from the ceiling camera is processed on the ground computer. ROS2 enables every computer connected to the same Wi-Fi network to subscribe to data being published by another machine.

Additionally, the interface utilizes Foxglove for data visualization and analysis, leveraging data directly from the sensors.

The following figure was achieved by running the command "$rqt\_graph$", a ROS command for visualization of all the active nodes, the oval shapes, and topics, the rectangular shapes.

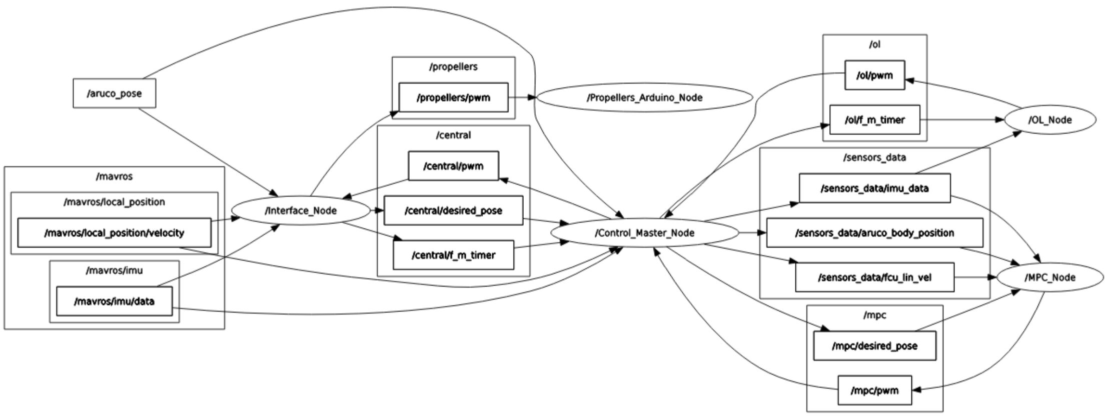


Figure 2.19: Communication between nodes generated by ROS2 command

The /mavros topics represent the processed data from our PixHawk and the /ArUcO_pose represent the processed data from the regular camera. Both the "MPC_Node" as well as the "OL_Node" are part of the Control Nodes mentioned earlier. The "/Propellers_Arduino_Node" is the node responsible to send the PWM values to the propellers' respective motors.

In spite of not incorporated in the architecture of the robot, it was developed an additional node to test the connections between the UpBoard computer and the propellers as well as their bidirectionality.

### Sensors Nodes

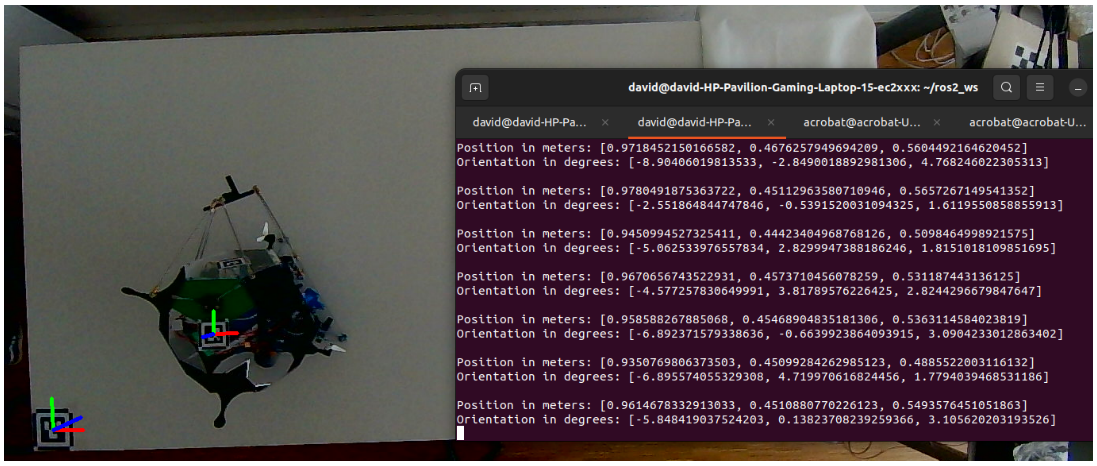
As mentioned in the electronics section, our sensor suite includes a regular camera, a depth camera, and a PixHawk (PX4) and while currently used just as a powerful Inertial Measurement Unit (IMU), the PixHawk's potential extends beyond this role and we aim to leverage its full capabilities in the near future.

Each of these sensors have dedicated nodes which calculate the data we use on our control unit.

The node responsible with contacting with the PixHawk, which is developed by the community and his named MAVROS, gives an immense quantity of data associated with all the algorithms present like the extended kalman filters and the controllers.The ones used are in particular the IMU data that includes orientations, linear accelerations and angular velocities, as well as linear velocity data, that results from the integration of the linear acceleration.

The ceiling-mounted regular camera node, developed by us, uses computer vision to calculate pose data (positions and orientations) by looking at our ArUcO Markers, one on top of the robot and another one on one of the corners of our table, and then calculating the relative pose of the robot to our world plane, which in this case is our table. The only pose data we use are planar positions $x$ and $y$ and the orientation yaw since in our case the robot "lives" on a 2D world which is the table.

In the next picture it is possible to visualize the perspective of the ceiling camera and some of the data, the "/ArUcO_pose" topic, being published.



```
Position in meters: [0.9718452150166582, 0.4676257949694209, 0.5604492164620452]
Orientation in degrees: [-8.90406019813533, -2.8490018892981306, 4.768246022305313]

Position in meters: [0.9780491875363722, 0.45112963580710946, 0.5657267149541352]
Orientation in degrees: [-2.551864844747846, -0.5391520031094325, 1.6119550858855913]

Position in meters: [0.9450994527325411, 0.44423404968768126, 0.5098464998921575]
Orientation in degrees: [-5.062533976557834, 2.8299947388186246, 1.8151018109851695]

Position in meters: [0.9670656743522931, 0.4573710456078259, 0.531187443136125]
Orientation in degrees: [-4.577257830649991, 3.81789576226425, 2.8244296679847647]

Position in meters: [0.958588267885068, 0.45468904835181306, 0.5363114584023819]
Orientation in degrees: [-6.892371579338636, -0.6639923864093915, 3.0904233012863402]

Position in meters: [0.9350769806373503, 0.45099284262985123, 0.4885522003116132]
Orientation in degrees: [-6.895574055329308, 4.719970616824456, 1.7794039468531186]

Position in meters: [0.9614678332913033, 0.4510880770226123, 0.5493576451051863]
Orientation in degrees: [-5.848419037524203, 0.13823708239259366, 3.105620203193526]
```

Figure 2.20: Ceiling camera perspective and the published data

Additionally, although not used yet, we own a depth camera to acquire the pose, this time, position $x$, $y$ and $z$ and roll, pitch and yaw of the Stewart Platform which also as an ArUcO Marker in it. This pose, will eventually be used by our 3D control node.

**Control Unit**

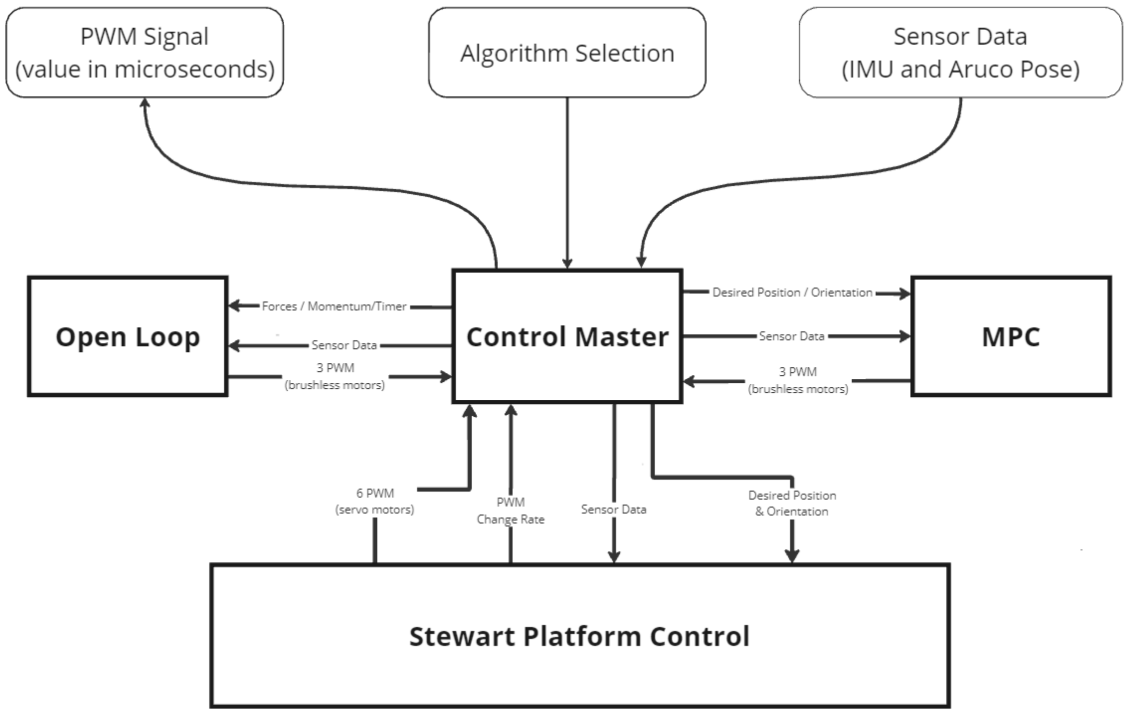


Figure 2.21: Schematic of the control unit in ROS2

Aforementioned the control unit is subdivided in two components, one of them being the control master which act as a intermediary between the user (interface) and the specific control nodes. Whenever it receives the user command from the interface node, it analises it and depending on what the user wants to achieve, send this data, as well as the sensor data it receives directly, to the necessary control node, after the calculations in the specific control nodes, it send back to the interface the data for the actuators such as PWM values for the propellers motors. The other component are the specific control nodes.

Firstly the open loop node which received an array of forces in the x and y axis as well as a torque around the z axis, and from there calculated the necessary PWM values for the robot to move accordingly to these instructions. The forces can be relative to the robot referential or the world referential, depending on what the user whats to achieve. In the case of the world referential, the node also received the orientation of the robot relative to the world, to then use on the rotation matrix. This node is useful to test our actuation matrix and conversion from thrust to PWM, to then use them in the MPC. The MPC node received all of the PixHawk data previously mentioned as well as the pose of the robot provided by the ceiling camera, besides a desired position and orientation for the robot to be in the world plane, the table. Then it proceeds to calculate the necessary PWM values to reach them.

### Interface

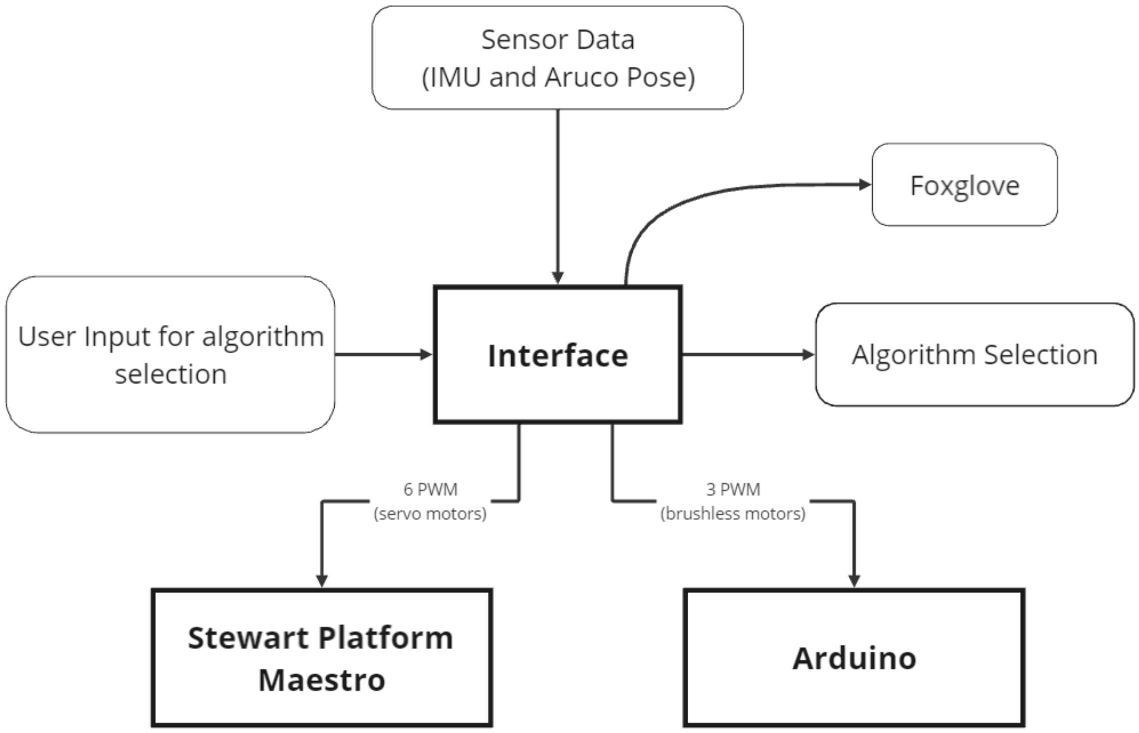
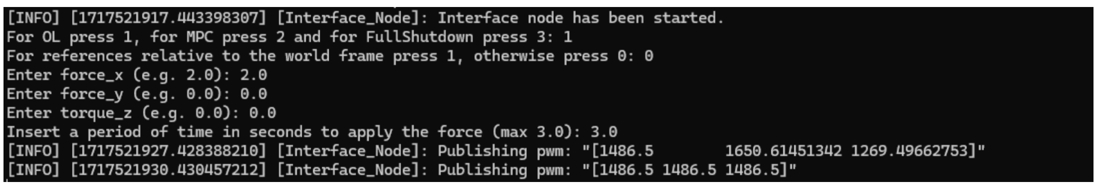


Figure 2.22: Interface Node Communication in ROS2

Currently, users interact with the robot through a text-based interface on a computer terminal. The interface first prompts the user to choose between control modes: open-loop control, MPC (Model Predictive Control), or full shutdown. If the user selects open-loop control, it then asks whether to apply forces relative to the robot's reference frame or the world reference frame. Subsequently, it prompts for the values of forces and torque, and a timer duration for applying them. In MPC mode, it only prompts for desired positions, along with a desired yaw angle.

As a safety measure, because it's the node that directly contacts the actuators, it has a full shutdown function by turning off all propellers. It achieves this by publishing a Pulse Width Modulation signal of 1486.5 microseconds, which corresponds to the neutral position for the propellers. Following this, it proceeds to shut down all other running nodes. Finally after receiving back from the control master node the actuation instructions, it redirects them to the actuation nodes, notifying the user of all the PWM values published.



```
[INFO] [1717521917.443398307] [Interface_Node]: Interface node has been started.
For OL press 1, for MPC press 2 and for FullShutdown press 3: 1
For references relative to the world frame press 1, otherwise press 0: 0
Enter force_x (e.g. 2.0): 2.0
Enter force_y (e.g. 0.0): 0.0
Enter torque_z (e.g. 0.0): 0.0
Insert a period of time in seconds to apply the force (max 3.0): 3.0
[INFO] [1717521927.428388210] [Interface_Node]: Publishing pwm: "[1486.5       1650.61451342 1269.49662753]"
[INFO] [1717521930.430457212] [Interface_Node]: Publishing pwm: "[1486.5 1486.5 1486.5]"
```

Figure 2.23: Interface for open loop

### Actuators Nodes

For the moment, the only actuation node incorporated in the ROS2 Architeture is a node that acts as an intermediary between the interface and the Arduino, which controls the propellers. Whenever this node receives an array of three PWM values, one for each propeller, it send the array through serial to the Arduino. As previously presented in the electronics subsection, MAVROS has the capability to transmit the PWM to the motors via the PixHawk. However, due to the unique geometry and propeller orientation, coupled with the inability to modify or override safety conditions, arming the Pixhawk was not feasible. These adaptations are slated for future implementation, given that there is no time constraint, which allows for a more comprehensive understanding of all the software in the FCU. In the near future we intend to add separate nodes for the actuation of the Stewart platform and the extrusion system.

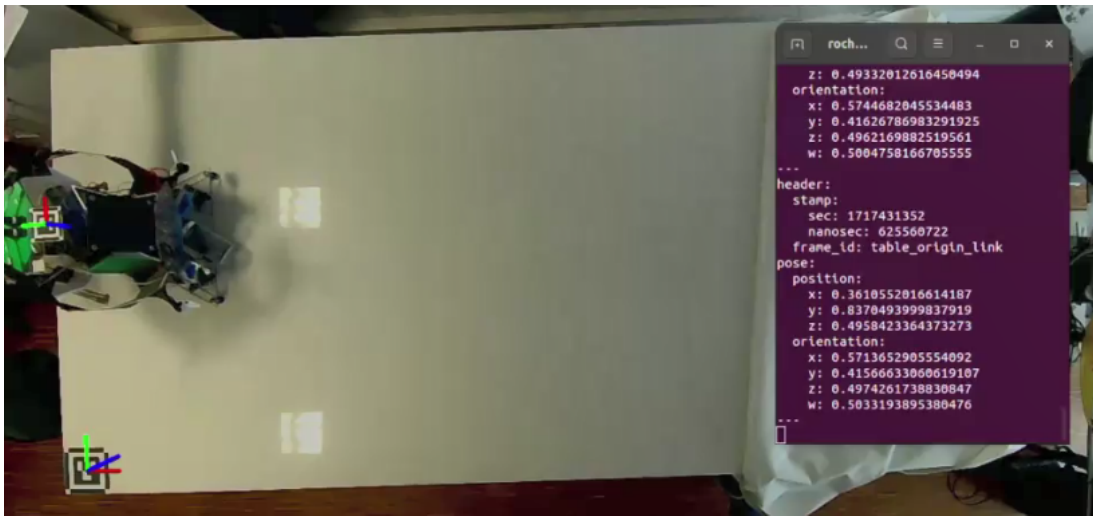
# Chapter 3

# Results

The complete fabrication of the prototype facilitated the execution of several tests on an air bearing table. These tests were designed to verify the accuracy of the robot model and to evaluate the performance of the controllers and the printing system. However, numerous delays and practical issues arose due to the malfunction of certain components. For instance, the air compressor, which was essential for filling the compressed gas cylinder that powered the pneumatic system, failed completely. Additionally, there were challenges in configuring the PX4 prior to the decision to switch to an Arduino. These setbacks limited the team's ability to conduct extensive tests and collect data for future analysis. Despite these challenges, we executed some open loop commands. Although the data from these tests was not preserved, the tests were filmed, enabling some qualitative analysis and discussion.

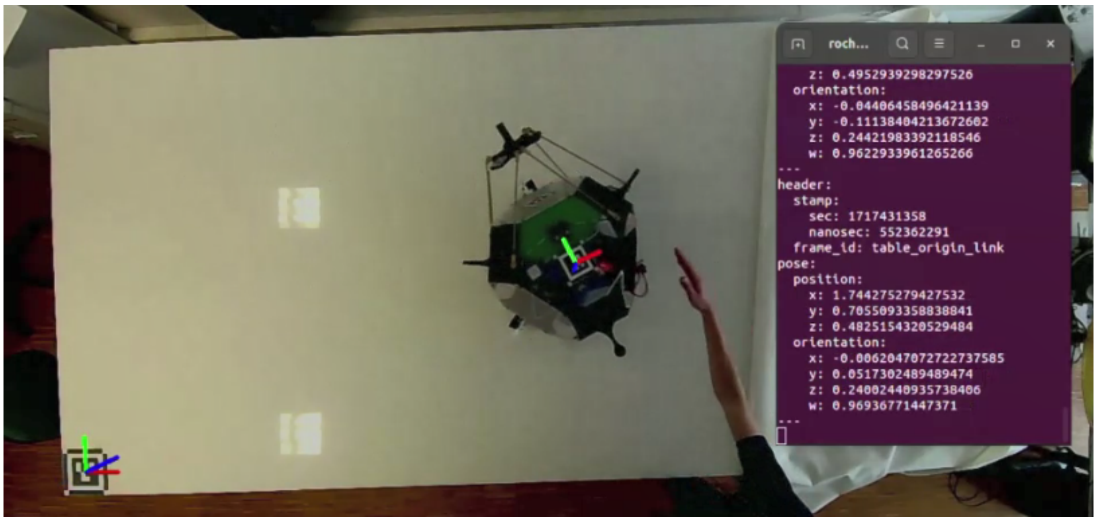
## 3.1 Open Loop testing

For the Open Loop Tests developed on the table, the objective was to test the validity of the equations developed in 2.1.3. Firstly, the equations have some approximations when compared to the real model. The geometrical center of the free-flyer doesn't coincide with the center of gravity of the robot which creates some deviations from the ideal actuation of the propellers, when the input is to develop a certain force along one the axis of the body.



```
    z: 0.49332012616450494
  orientation:
    x: 0.5744682045534483
    y: 0.41626786983291925
    z: 0.4962169882519561
    w: 0.5004758166705555
---
header:
  stamp:
    sec: 1717431352
    nanosec: 625560722
  frame_id: table_origin_link
pose:
  position:
    x: 0.3610552016614187
    y: 0.8370493999837919
    z: 0.4958423364373273
  orientation:
    x: 0.5713652905554092
    y: 0.41566633060619107
    z: 0.4974261738830847
    w: 0.5033193895380476
---
```

(a) Starting Position



```
    z: 0.4952939298297526
  orientation:
    x: -0.04406458496421139
    y: -0.11138404213672602
    z: 0.24421983392118546
    w: 0.9622933961265266
---
header:
  stamp:
    sec: 1717431358
    nanosec: 552362291
  frame_id: table_origin_link
pose:
  position:
    x: 1.744275279427532
    y: 0.7055093358838841
    z: 0.4825154320529484
  orientation:
    x: -0.006204707272273 7585
    y: 0.0517302489489474
    z: 0.24002440935738406
    w: 0.96936771447371
---
```

(b) End Position

Figure 3.1: Open Loop Test

As observed in the two images, although the majority of the robot's movement was along the $x$ axis,which was the axis corresponding to the open loop movement request, there was a significant change in orientation. This can be attributed to the influence of the Stewart platform and the compressed air cylinder on the robot's center of gravity. This observation indicates that the Model Predictive Control (MPC) will require a more rigorous tuning of the orientation variable, as it is the most susceptible to disturbances.

# Part II

# Parallel Manipulator

# Chapter 4

# Background

Parallel manipulators are a class of closed-loop kinematic mechanisms where the end-effector is connected in parallel to the base by multiple kinematic chains. Series manipulators, like robotic arms, suffer in terms of precision as the error of each actuator stacks on the next. This means that for a 6 DoF series manipulator the total error, assuming all actuators are the same, is six times the error of an individual actuator. Because all actuators of a parallel manipulator connect directly from the base to the end-effector, its total error, no matter how many actuators are used, is that of a single actuator. Parallel connection also provides greater stiffness and payload capacity at the expense of workspace volume.

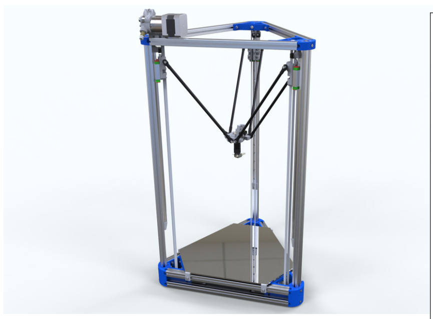


Figure 4.1: Example of a Delta Platform being used on a 3D printer[9]

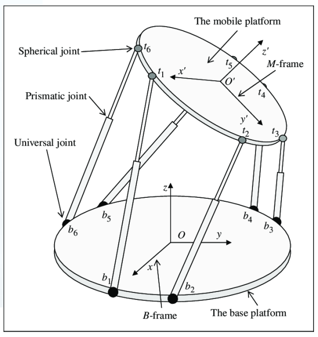


Figure 4.2: Illustration of a Stewart-Gouge platform[10]

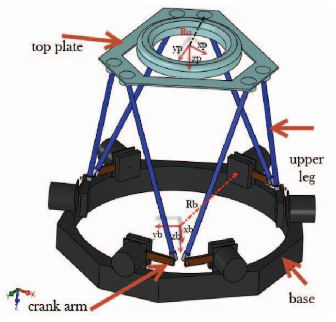


Figure 4.3: Schematic of a rotary Stewart platform[11]

The most common parallel manipulator is the Delta platform seen in a wide variety of 3D printers like the one shown above in figure 4.1. It possesses 3 legs of fixed lengths connected to guide rails where they independently move to provide 3 degrees of translational freedom. The Stewart-Gouge platform has 3 additional DoF as it possesses 6 linkages whose lengths vary to provide roll, pitch and yaw in addition to translation as seen in figure 4.2.

A common variant of the Stewart-Gouge platform in the field of robotics is the rotary Stewart platform seen in figure 4.3. Instead of the linear actuators seen in figure 4.2, through the rotation of six pairs of arm-leg segments, the total lengths from the base to the platform vary providing motion in 6 DoF.

## Acrobat

In its prior development, Acrobat adopted the rotary Stewart platform using 6 servos. Their wide adoption in the robotics industry makes them appealing for development in contrast to linear actuators used in the classic Stewart-Gouge platform configuration. The main focus of João Vale's thesis was on the rotary Stewart platform, its calibration and its movement potential. In it, he laid out the inverse kinematics and other useful tools such as workspace volume estimation. Using his work as a guide, as well as Alexandre Rocha's experience learned from the previous design seen below in figure 4.4, we developed our own rotary Stewart platform and the framework for making it function.

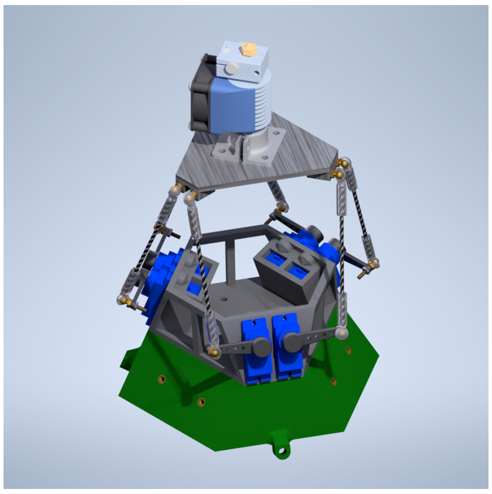


Figure 4.4: Previous Acrobat version rotary Stewart platform

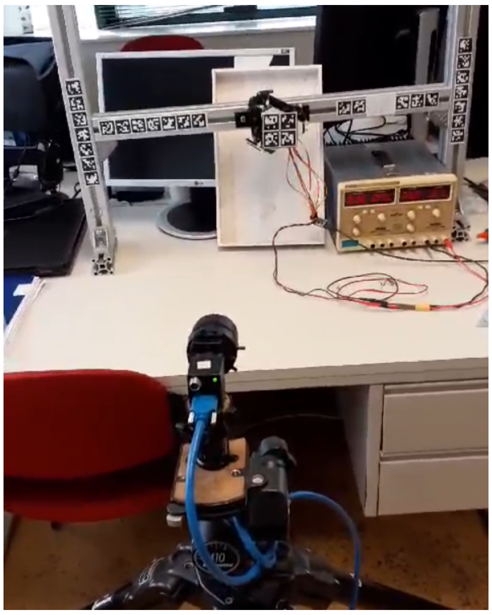


Figure 4.5: Old version manipulator being calibrated and tested

# Chapter 5

# Methodology

## 5.1 Mechanics

Below is the latest design of our rotary Stewart platform which provides six degrees of freedom through the use of six actuated arms-leg pairs that connect from a fixed base platform to a mobile platform. The arm is the shorter segment, directly actuated 5 inch (approximately $127\,\mathrm{mm}$) segment comprising of a 2 smaller segments connected by a coupling nut. One is a 2 inch rod end bolt whose end serves to mount a ball joint and the other is a 3 inch connecting rod using a 3D printed hub with a threaded heat set insert to mount to the aluminum servo horn using M3 bolts. The leg is the longer segment comprising of a 12 inch (approximately $305\,\mathrm{mm}$) connecting rod linking the end of the arm and mounting point of the platform with ball joints. All connections to the ball joints, the ball joints themselves, and the threaded insert in the servo hub use 10-32 thread possessing a diameter of $\frac{3}{16}$ of an inch (nearly $5\,\mathrm{mm}$). To reduce vibrations and the risk of connections becoming loose, all threaded connections except for those connected to the platform were treated with high strength thread locker.

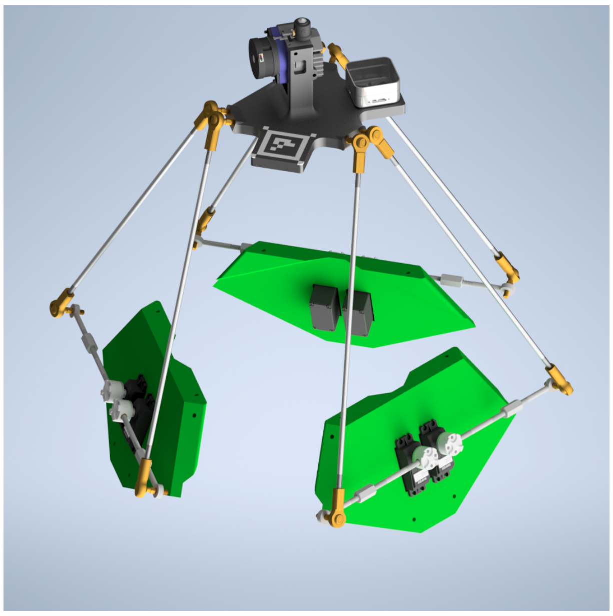


Figure 5.1: Latest rotary Stewart platform design

The ball joints have a maximum swivel range of $50^\circ$ while the base faces are angled inward by $19.47^\circ$. This means that if the ball joints on the platform were mounted parallel to the plane of the platform they would only have around $5^\circ$ to move in one of the tilting directions. To alleviate this, their mounting points on the platform are also angled upward by $19.47^\circ$ to allow for the full $25^\circ$ of tilt in either direction.

For its printing capability, the manipulator boasts a *Smart Orbiter V3* [12] FDM direct drive extruder equipped with a $1.8\,\mathrm{mm}$ *Bondtech CHT* nozzle. At first glance, considering that the manipulator will be mounted to a mobile free-flyer, a direct extruder may seem like a poor choice. After all, the stepper motor which drives the filament is connected directly to the hotend increasing the inertia of the end-effector. However, using a Bowden type extruder, where the filament driving stepper motor could be placed inside the robot's body, presents feeding concerns when considering how much plastic will be extruded from a $1.8\,\mathrm{mm}$ nozzle using $1.75\,\mathrm{mm}$ filament. This, along with the rotary Stewart platform's unusual twisting movements and longer path from the spool to the hotend than most conventional FDM printers makes filament skipping through the drive gear or breaking in the Bowden tube a likely scenario. Typical direct drive extruders weigh around $250\,\mathrm{g}$ whereas the *Smart Orbiter V3* weighs only $175\,\mathrm{g}$ and pushes filament with a force of $6.5\,\mathrm{kg}$[12] making it one of the lightest, strongest extruders of this type on the market. However, its most important feature, and ultimate deciding factor, is its heat break.

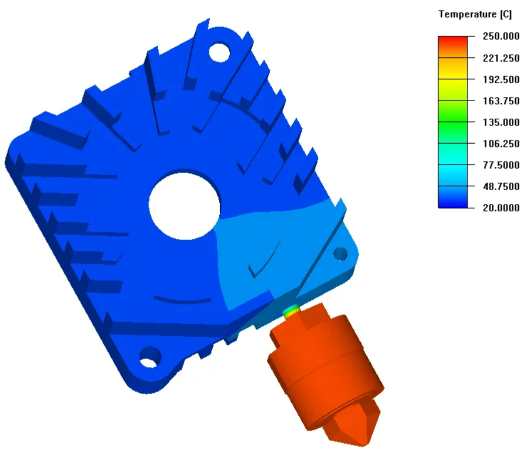


Figure 5.2: Thermal simulation results depicting heat break between the hotend and heatsink with fan turned off[12]

Seen above in figure 5.2, are the results of a thermal simulation of the hotend. They show that even with the heatsink fan off, its temperature stays below $50\,^\circ\mathrm{C}$. Since the heatsink serves as a mounting point, this means that it can be mounted on 3D printed parts. In fact, the platform as shown in figure 5.1, with the depicted ArUcO marker, was printed on a *Bambu Lab X1C* 3D printer in black carbon fiber PLA and white generic PLA. This property made prototyping vastly easier, and, as will be seen later on, the ArUcO was usable for calibration of the manipulator.

The final component mounted on the platform is an *Intel® RealSense™ Depth Camera D405* stereo camera. It is mounted such that the tip of the extruder is $7\,\mathrm{cm}$ away from the lenses at the beginning of the stereo camera's 7 to $50\,\mathrm{cm}$ range. The camera only weighs around $50\,\mathrm{g}$ making it a worthy trade-off for a camera that can provide sub-millimetric precision and real-time positioning and calibration.

## 5.2 Design Parameters

Before establishing the design parameters of the current iteration of the Stewart platform, it's important to highlight that the mathematical description adopted for the manipulator does not accurately portray its real-world counterpart. However, in the light of the inverse kinematics, only a simple translation of the desired pose is needed to adapt between the two. A similar simplification of the model might not always be an option depending on the design of the Stewart platform, for it might be impossible to adapt between them or it might prevent the modeling of certain constraints.

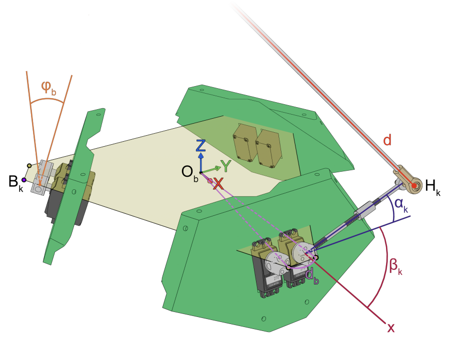


Figure 5.3: Design parameters of Stewart's base platform

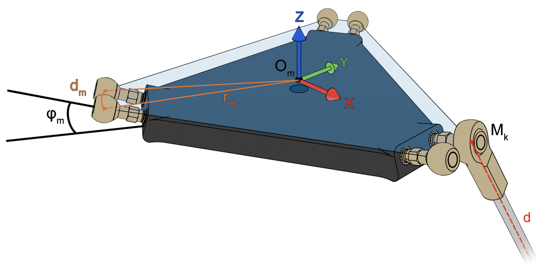


Figure 5.4: Design parameters of Stewart's mobile platform

Figures 5.3 and 5.4 depict the design parameters of the current iteration Acrobat's Stewart platform. The translucent polygons illustrate the base and mobile platforms described by the mathematical model. The base platform is defined by its origin, $O_b$, radius, $r_b$, anchor points, $\boldsymbol{B_k}$, and the angle between the revolute joint's orientation in space and the $xy$ plane, $\phi_{b_k}$. The mobile platform is defined by its origin, $O_m$, radius, $r_m$, anchor points, $\boldsymbol{M_k}$, and the angle between the ball joint's orientation in space and the $xy$ plane, $\phi_{m_k}$. Each Revolute-Spherical-Spherical arm is defined by its anchor point, $\boldsymbol{H_k}$, the length of the rigid link that connects $\boldsymbol{B_k}$ to $\boldsymbol{H_k}$, $h$, the angle between this link's orientation in space when in the neutral position and the $x$ axis, $\beta_k$, and the length of the rigid link that connects $\boldsymbol{H_k}$ to $\boldsymbol{M_k}$, $d$.

The anchor points of the manipulator's platforms and arms can be determined using the above parameters. The base platform's anchor points, $\boldsymbol{B_k}$, are calculated as:

$$\boldsymbol{B_k} = \begin{bmatrix} r_b cos(\theta_b) & r_b sin(\theta_b) & 0 \end{bmatrix}^T \qquad \theta_b = \frac{2\pi\lfloor\frac{k+1}{2}\rfloor}{3} + (-1)^k d_b \tag{5.1}$$

The mobile platform's anchor points, $\boldsymbol{M_k}$, are calculated as:

$$\boldsymbol{M_k} = \boldsymbol{T} + \boldsymbol{R}\begin{bmatrix} r_m cos(\theta_m) & r_b sin(\theta_m) & 0 \end{bmatrix}^T \qquad \theta_m = \frac{2\pi\lfloor\frac{k+1}{2}\rfloor}{3} + (-1)^k d_m \tag{5.2}$$

where $\boldsymbol{T} = (O_m - O_b)^T = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is the translation and $\boldsymbol{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix which define the mobile platform's pose $\boldsymbol{p}$.

The arms' anchor points, $\boldsymbol{H_k}$, are calculated as:

$$\boldsymbol{H_k} = \boldsymbol{B_k} + h\begin{bmatrix} sin(\beta_k)sin(\phi_{b_k})sin(\alpha_k) + cos(\beta_k)cos(\alpha_k) \\ -cos(\beta_k)sin(\phi_{b_k})sin(\alpha_k) + sin(\beta_k)cos(\alpha_k) \\ cos(\phi_{b_k})sin(\alpha_k) \end{bmatrix} \tag{5.3}$$

$$\phi_{b_k} = (-1)^{k+1}\phi_0 \qquad \beta_k = \frac{2\pi\lfloor\frac{k+1}{2}\rfloor}{3} + (-1)^k\beta_0 \tag{5.4}$$

where $\alpha_k$ is the angle of each revolute joint.

Table 5.1 presents the design parameters of the current iteration of Acrobat's Stewart platform, while figure 5.5 shows a 3D plot of the manipulator as described by its anchor points.

| Parameter | $r_b$ $[mm]$ | $r_m$ $[mm]$ | $d_b$ $[rad]$ | $d_m$ $[rad]$ | d [mm] | $h$ $[mm]$ | $\phi_{b_0}$ $[rad]$ | $\phi_{m_0}$ $[rad]$ | $\beta_0$ $[rad]$ |
|---|---|---|---|---|---|---|---|---|---|
| Value | 185.89 | 81.68 | 0.162 | 0.209 | 333.35 | 127.00 | 0.340 | 0.340 | 1.571 |

Table 5.1: Design parameters of ACROBAT's current manipulator

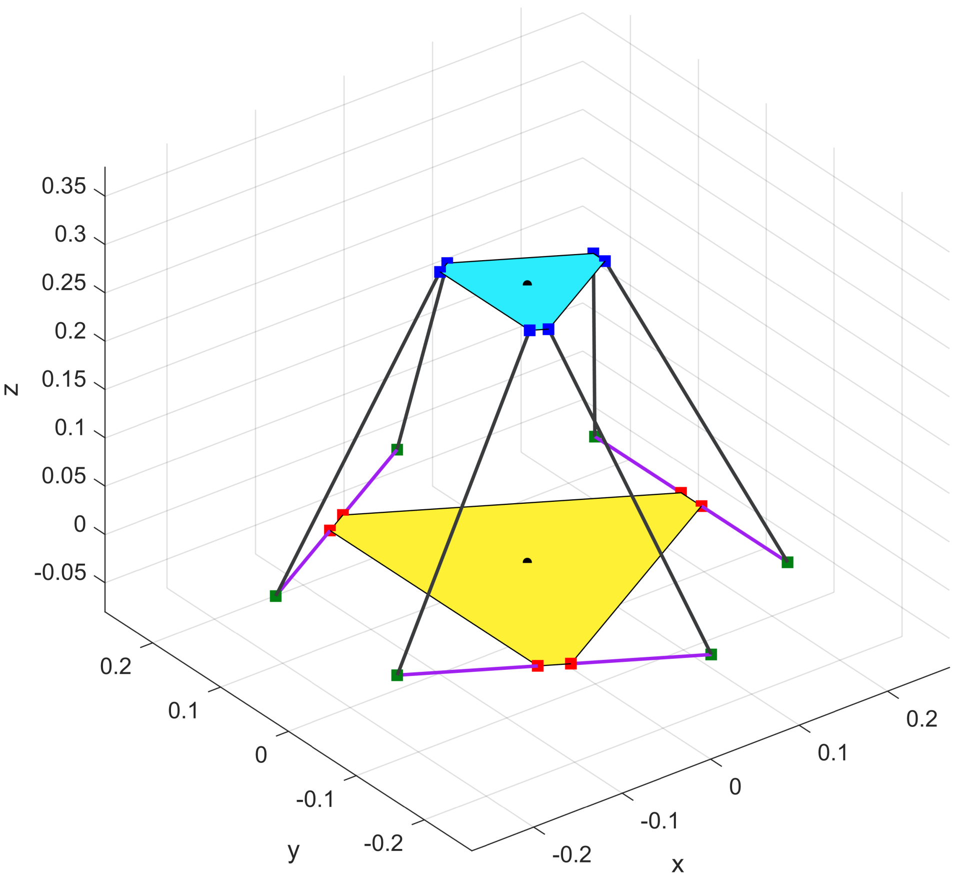


Figure 5.5: MATLAB 3D plot representing the manipulator used for inverse kinematics

## 5.3 Inverse Kinematics

The previous analysis has demonstrated that the pose of all the anchor points can be derived by knowing both the end effector pose $\boldsymbol{p}$ and the angle of each revolute joint $\alpha_k$. Given the configuration of the Stewart platform, inverse kinematics can be applied to compute the latter variables from the former [5]. The resulting inverse kinematics equation is:

$$\alpha_k = arcsin\left(\frac{\|\boldsymbol{i_k}\|^2-(d^2-h^2)}{\sqrt{a_k^2+b_k^2}}\right) - arctan2(b_k, a_k) \tag{5.5}$$

where

$$\boldsymbol{i_k} = \boldsymbol{M_k} - \boldsymbol{B_k} \tag{5.6}$$

$$a_k = sin(\beta_k)sin(\phi_{b_k}){i_k}^{(x)} - cos(\beta_k)sin(\phi_{b_k}){i_k}^{(y)} + cos(\phi_{b_k}){i_k}^{(z)} \tag{5.7}$$

$$b_k = cos(\beta_k){i_k}^{(x)} + sin(\beta_k){i_k}^{(y)} \tag{5.8}$$

## 5.4 Workspace

The workspace $\mathcal{W}$ of a manipulator can be defined as the set of $\boldsymbol{p}$ that the end effector can take while satisfying various constraints. Generally, three types of constraints restrict parallel manipulators:

1. Constraints imposed by the actuators;

2. Constraints imposed by the passive joints;

3. Constraints imposed by the mechanical interference of links.

The work presented in (Vale, 2021), which focused on the first type of constraints, is now expanded to model the other two, taking into account the current configuration of the manipulator.

As discussed in the mechanics section, each leg of length $d$ has a passive spherical joint at both ends, with a maximum ball swivel of $50°$. This means that the angle between the bearing's normal vector and the joint's orientation is restricted to a maximum of $25°$. However, the bearing's normal orientation can't be fully described, as there is a degree of freedom that cannot be accounted for. Specifically, this degree of freedom corresponds to a rotation of the leg rod along its longitudinal axis, which does not result in a variation of the end effector pose $\boldsymbol{p}$. This constraint is addressed by analyzing the angle between the leg vector and the joint's orientation. Considering the orthogonal relationship between each leg and its corresponding bearing's normal vector, the allowed range for this angle is determined to be $[65, 115]°$. To verify whether these constraints are satisfied for a given $\boldsymbol{p}$, the swivel angles at both the mobile platform's anchor points, $\theta_{m_k}$, and the arm's anchor points, $\theta_{h_k}$, must be computed:

$$\theta_{m_k} = \arccos\left(\frac{\boldsymbol{n_{m_k}} \cdot \boldsymbol{h_{m_k}}}{|\boldsymbol{n_{m_k}}||\boldsymbol{h_{m_k}}|}\right)^T \times \frac{180}{\pi} \tag{5.9}$$

$$\theta_{h_k} = \arccos\left(\frac{\boldsymbol{n_{h_k}} \cdot -\boldsymbol{h_{m_k}}}{|\boldsymbol{n_{h_k}}||-\boldsymbol{h_{m_k}}|}\right)^T \times \frac{180}{\pi} \tag{5.10}$$

where $\boldsymbol{n_{m_k}}$ and $\boldsymbol{n_{h_k}}$ are the orientations of each mobile platform's and arm's joints, respectively, and $\boldsymbol{h_{m_k}}$ is the leg vector, defined as follows:

$$\boldsymbol{n_{m_k}} = \left(R + \left[(-1)^{k-1}\sin(\beta_k(k)) \quad (-1)^k\cos(\beta_k(k)) \quad \sin(\phi_m)\right]^T\right)^T \tag{5.11}$$

$$\boldsymbol{n_{h_k}} = \left[(-1)^{k-1}\sin(\beta_k(k)) \quad (-1)^k\cos(\beta_k(k)) \quad \sin(\phi_b)\right] \tag{5.12}$$

$$\boldsymbol{h_{m_k}} = \boldsymbol{H_k} - \boldsymbol{M_k} \tag{5.13}$$

In the current configuration of the manipulator, the third type of constraint is only verified by the legs, which may intersect with the propellers in a small set of $\boldsymbol{p}$. A simple approach was adopted to model these constraints, where each propeller, indexed by $n \in 1, \ldots, 3$, is enveloped by a sphere centered at $S_n$ with radius $r_n$. Intersection analysis involves calculating the scalar projection of vector $\boldsymbol{p_{k_n}}$ connecting each anchor point $M_k$ to the sphere center $S_n$ onto the leg vector $\boldsymbol{h_{m_k}}$:

$$t = \frac{\boldsymbol{p_{k_n}} \cdot \boldsymbol{h_{m_k}}}{\|\boldsymbol{h_{m_k}}\|} \tag{5.14}$$

$$dist = \|M_k + t \cdot \boldsymbol{h_{m_k}} - S_n\| \tag{5.15}$$

An intersection is identified if the projected point lies within the leg vector's span $(1 \geq t \geq 0)$ and its distance to $S_n$ is less than or equal to $r_n$. This radius should not be less or equal to that of the propeller's radius, as it should also account for the radius of the leg rod.

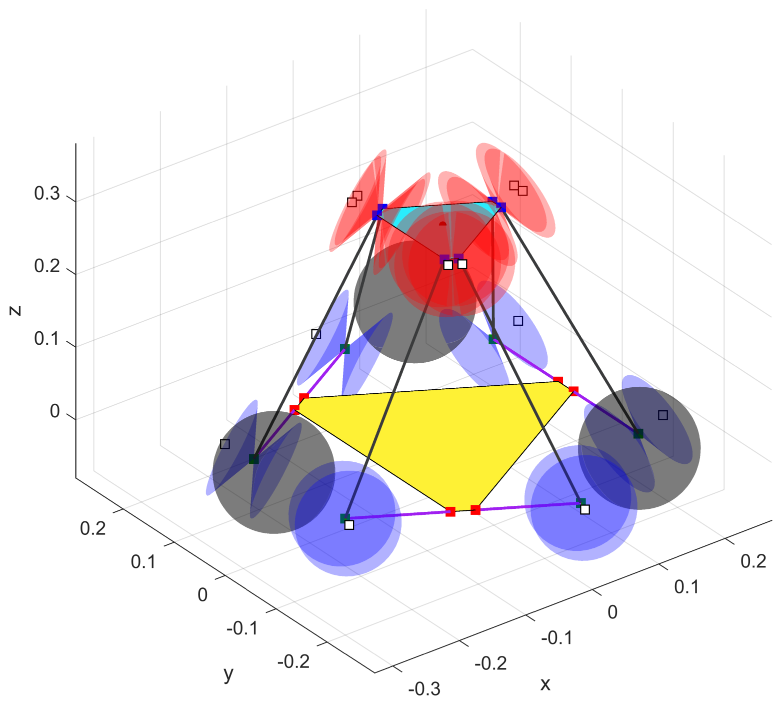


Figure 5.6: Rejection zones for the manipulator's legs

Figure 5.6 presents the rejection zones for the legs of the Stewart platform, being the red and blue cones illustrative of the constraints set by the rod end bearing joints and the black spheres illustrative of the constraints set by the mechanical interference with the propellers. With all the constraints modelled, the total workspace volume estimated is of $14.836\,\mathrm{dm}$.

## 5.5 Electronics

The manipulator's electronics are comprised of the same *UpBoard* onboard computer from section 2.3.1, a *Sony Playstation Eye* external camera, an *Intel® RealSense™ Depth Camera D405* onboard stereo camera, a *Smart Orbiter V3* FDM extruder, a *Pololu Micro Maestro* servo controller, and six *DSS-M15S 270°* servos motors to actuate its arms.

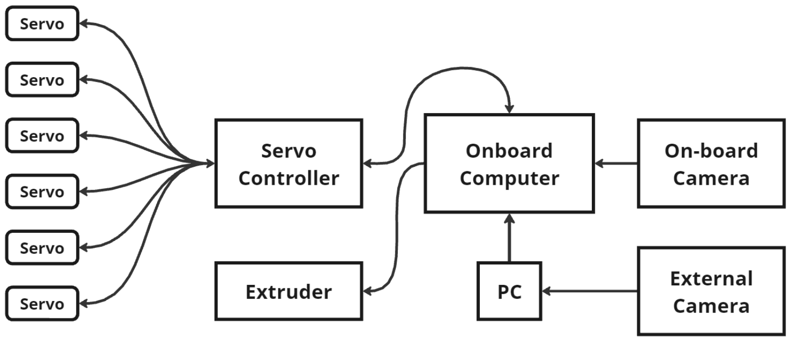


Figure 5.7: Manipulator electronics schematic

Communication with each servo is done via PWM where, according to *DSS-M15S* documentation, a PWM of $500\,\mu\mathrm{s}$ rotates the servo to $-\frac{3\pi}{4}$ radians and a PWM of $2500\,\mu\mathrm{s}$ to $\frac{3\pi}{4}$ radians. Ideally, one would actuate each servo to 0 by setting the PWM to $1500\,\mu\mathrm{s}$, and install the servo horn. Unfortunately, in practice, given the servo's geared system for installing the horn,an angle offset between the servo horn and the neutral $0\,\mathrm{rad}$ position will be introduced. To compensate for this, an offset PWM value, denoted as $q_k^0$, is determined for each servo k. This value, typically close to $1500\,\mu\mathrm{s}$, represents the PWM signal required to achieve a perceived neutral horn position. With that, given a desired actuator angle $\alpha_k$, the corresponding PWM input $q_k$ for the k-th servo is calculated as:

$$q_k = q_k^0 + (-1)^k \frac{\Delta q}{\Delta \alpha} \alpha_k \tag{5.16}$$

where $\frac{\Delta q}{\Delta \alpha}$ is a ratio of PWM µs per radian. In the case of the used *DSS-M15S*, $\frac{\Delta q}{\Delta \alpha} = \frac{2000}{\frac{3\pi}{2}}$.

Besides the ground, power and PWM pins found on typical servos, the *DSS-M15S* also possesses a feedback pin allowing us to compare the desired position sent and the actual position where it arrived.

The *Micro Maestro* controller is used as a communication layer between the on-board computer and servos, since directly controlling the six servos requires a large number of GPIO pins. The onboard computer can send messages through serial to communicate with the *Micro Maestro*, being the most notable messages the Set Target and Set Speed, both present in the Maestro documentation.

The manipulator's end effector is equipped with an *Intel® RealSense™ D405* stereo camera connected to the on-board computer. This camera allows for two functionalities:

1. Upon given the instruction to start printing, the initial position of the mobile platform is found and calibrated through the use of ArUco markers placed on the build plate. This process will ensure that the platform is parallel to the build plate and distanced from it by half the value of the nozzle diameter. Once the initial position is determined, the on-board computer can calculate the translation vector and rotation matrix required to transform each point describing the object to be printed to the desired position that the Stewart platform must take.

2. The camera's stereo lens allows for depth information to be retrieved, which can be used for obstacle avoidance. While translating, if the camera is pointing in the direction of the translation, the robot will be able to detect possible collisions and evade them by communicating with the MPC.

The *Sony Playstation Eye* is an external camera which will be used for a computer vision calibration system, details of which will be discussed in the next section.

### 5.5.1 Extruder Electronics

Finally, responsible for Acrobat's 3D printing capability, we will discuss the *Smart Orbiter V3* and its use in the project. This extruder is advertised as coming with its own tool board with several advantageous features like automotive inspired circuit protection, high-end stepper driver, sensor inputs and even an accelerometer[12]. Among these, the accelerometer was the most exciting as it could be used to make a robust control system using both vision and acceleration data quite similarly to how it is described in section 2.2.

To implement the *SO3* tool board, however, two issues arise with the first being that it simply is not yet available. Despite this, there are other tool boards in the Orbiter ecosystem with similar features which could be used instead. The second, more foreboding issue is that 3D printers are commanded by *.gcode* files which are, in almost every case, parsed by *Klipper*[13] or *Marlin*[14] firmware which tells the machine how to move its actuators. Besides the possible compatibility issues between these firmware options and the servo motors we selected, there aren't any readily available slicers capable of generating *.gcode* for 6 DoF systems.

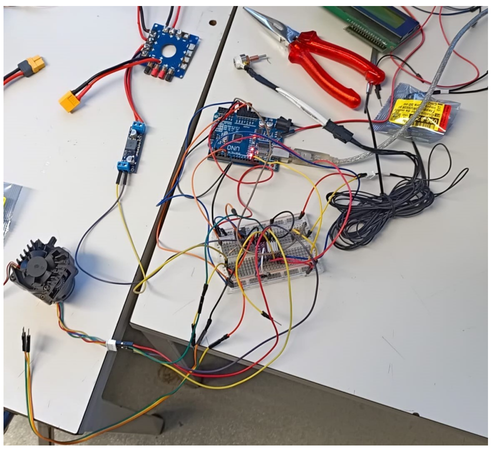


Figure 5.8: Prototype circuit for controlling the extruder

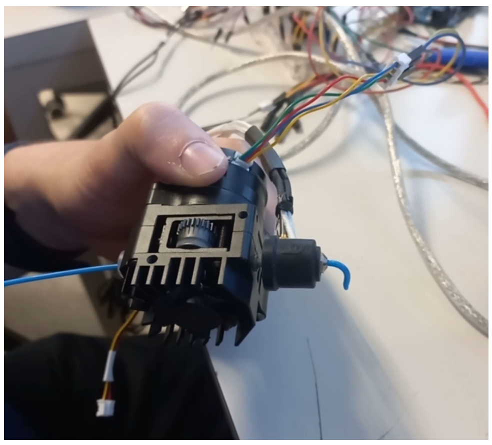


Figure 5.9: Smart Orbiter V3 extruding plastic from a $1.8\,\mathrm{mm}$ nozzle

Due to this, we made our own circuit to control the extruder using an *Arduino Uno*. There are three parts that need to be controlled on the extruder: the stepper motor, heating element, and heatsink cooling fan. Of these three, the last was easiest to control by negating to provide PWM and simply connecting the PWM pin to a constant $24\,\mathrm{V}$ power supply keeping it always on.

The stepper motor was also fairly easy to control using a DRV8825 stepper motor driver and the *AccelStepper* library in the in the Arduino code. Stepper motors are controlled using steps where coils with two pins each are powered on and off causing the motor to rotate. In our case, the stepper motor has 2 coils with 2 pins each. For a full step, the coils are powered sequentially with coil A fully energized followed by coil B being fully energized resulting in 2 steps. Operating in full steps, however, results in jittery motion. Using a stepper motor driver mitigates this by allowing for microstepping. For example, in quarter microstepping coil A starts powered to 100% with B at 0% then A moves to 75% with B at 25% and so on until A is at 0% and B at 100% signifying 1 full step in 4 smaller increments. The creator of the SO3 extruder was kind enough to provide direct ratios for one-sixteenth microstepping so this variation is what we used the most during development. To send commands, two pins, STEP and DIR, were connected from the DRV8825 to the Arduino Uno providing stepping and direction information respectively.

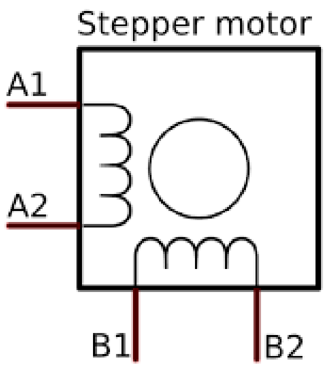


Figure 5.10: Two coil stepper motor diagram

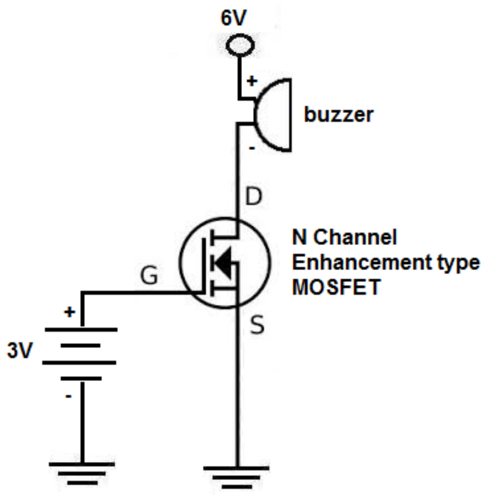


Figure 5.11: MOSFET use case example

By far, the hardest element to control was the heating element. While the stepper motor is powered by the stepper motor driver which is connected to the power supply, the $72\,\mathrm{W}$ heating element needs $24\,\mathrm{V}$ power directly. If we were to simply connect it directly with no control, the heating element would receive too much current, heat up too much and become a fire risk. To control it, we added a MOSFET to the circuit.

MOSFETs have 3 gates: source, gate and drain where gate activates or deactivates the connection between source and drain. In the example seen in figure 5.11, an N Channel enhancement type MOSFET, the same type as in our circuit, is used as a switch to activate or deactivate a buzzer. If the $3\,\mathrm{V}$ connection to gate is active, source and drain are connected providing ground and the buzzer turns on. If the connection is inactive, ground is disconnected and thus the buzzer turns off. In our case, the buzzer is a heating element connected to a $24\,\mathrm{V}$ power supply and the MOSFET gate is connected to one of the Arduino Uno's PWM pins.

Heeding to SO3 creator Róbert Lőrincz's warning to never fully power the heating element, we made sure to place a limit on the PWM duty cycle to never surpass 93%. We also programmed a simple PID controller such that a desired temperature could be set and maintained. In fact, to substitute *.gcode*, we created our own type of command for feed rate and temperature on the extruder. For example, if "V10T200" is sent to the Arduino over serial, the stepper motor will rotate such that it feeds enough plastic for a line printed at $10\,\mathrm{mm\,s^{-1}}$ and the temperature will be set to $200\,^{\circ}\mathrm{C}$ until the next command is sent. For more details, the Arduino code corresponding to this section can be found in appendix **??**.

It is important to mention that the temperature readings from the thermistor mounted on the hotend used for the PID controller could be inaccurate. To read them, we amplified its analog signal using a combination of resistors along with the Aruino's $5\,\mathrm{V}$ supply and fed it directly into one of its analog pins. However, because we intend to print in PLA which has a large interval of acceptable printing temperatures we deemed the readings likely sufficient.

## 5.6 Vision and Calibration

As seen in the previous section, the existence of the $q_k^0$ offset of each servo makes it necessary to have a calibration procedure for the manipulator, since the given manipulator pose $\boldsymbol{p}$ will not match the observed one $\boldsymbol{p'}$. To retrieve $\boldsymbol{p'}$, a single camera computer vision system is implemented in conjunction with the use of ArUco markers.

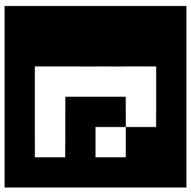


Figure 5.12: Standard ArUco Marker

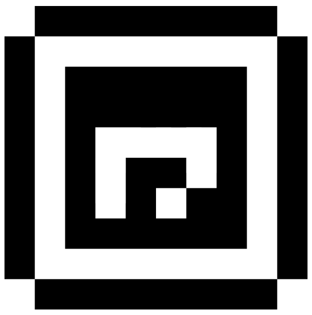


Figure 5.13: Proposed alternative of the ArUco marker

ArUco markers, characterized by their square shape and binary-coded black-and-white patterns, facilitate pose estimation through computer vision algorithms. These algorithms leverage the unique encoding of each marker to determine not only their presence but also their position and orientation in the camera's reference frame. To enhance positional accuracy, an alternative marker design proposed by (Bocco, 2021) was employed. This design integrates a standard ArUco marker within a chessboard-like pattern, enabling the use of corner subpixel algorithms for refined position estimation. Figure 5.12 shows the standard version of the marker, while figure 5.13 shows the enhanced marker. This approach for pose estimation requires for both the marker size and the camera intrinsic parameters to be known.

### 5.6.1 Camera Calibration

The process of determining the intrinsic parameters of the camera, such as the intrinsic matrix and distortion coefficients, was accomplished through the use of a ChArUco board and the OpenCV computer vision library. A ChArUco board is a calibration pattern that combines a chessboard and ArUco markers, providing a robust calibration target. Multiple images of the board were captured from various perspectives, and OpenCV functions were utilized to detect the board, extract chessboard corners, and identify ArUco markers. These data points were then processed through a mathematical optimization process to estimate the camera's intrinsic parameters, including focal length, principal point, and distortion coefficients.

### 5.6.2 Manipulator Calibration

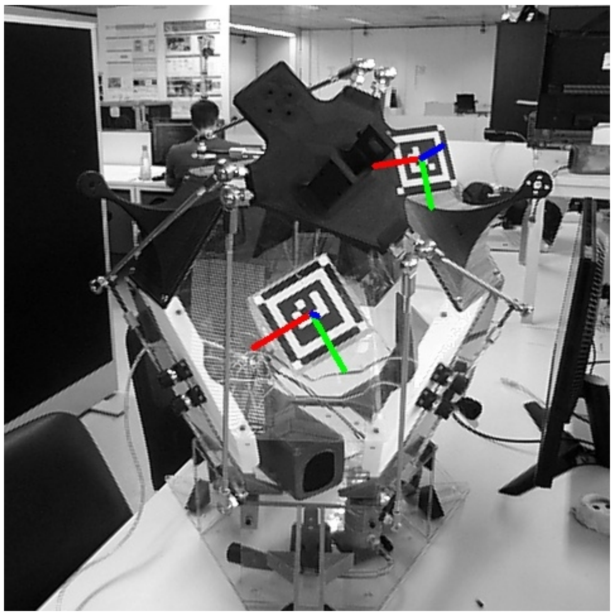


Figure 5.14: Detection of both Aruco Markers by the external camera

When using OpenCV, a rotation vector and a translation vector are retrieved for each detected marker. The rotation vector, expressed in Rodrigues form, represents the marker's orientation relative to the camera coordinate system and should be converted into a standard rotation matrix. The translation vector, also relative to the camera coordinate system, quantifies the marker's displacement from the camera's origin. As the pose $\boldsymbol{p}$ is defined relative to the manipulator's reference frame, two ArUco markers were required. One marker was positioned 3D printed onto the mobile platform, while the second was affixed to the face of the base body that is oriented parallel to the mobile platform's neutral position. While this means that, in certain positions, the camera might not be able detect the base marker, as it can be obstructed by the platform, the base marker's position will not change, meaning that at least one reading should suffice. This second marker establishes the origin and reference frame for determining the position and orientation of the first marker.

Let $\boldsymbol{T_0}$ and $\boldsymbol{R_0}$ denote the translation vector and rotation matrix, respectively, of the origin marker with respect to the camera frame. Similarly, let $\boldsymbol{T_1}$ and $\boldsymbol{R_1}$ represent the translation vector and rotation matrix of the mobile platform marker, also with respect to the camera frame. The translation vector $\boldsymbol{T_{0\rightarrow 1}}$ and rotation matrix $\boldsymbol{R_{0\to1}}$, which describe the pose of the mobile platform marker in the reference frame of the origin marker, can then be expressed as follows:

$$\boldsymbol{R_{0\to1}} = \boldsymbol{R_0}^T \cdot \boldsymbol{R_1} \qquad \boldsymbol{T_{0\to1}} = \boldsymbol{R_0}^T \cdot (\boldsymbol{T_1} - \boldsymbol{T_0}) \tag{5.17}$$

However, as clearly seen in figure 5.14, neither marker is placed of the origin of its respective platform, which means a translation vector is required for each marker to account for this offset. If $\boldsymbol{T_b}$ and $\boldsymbol{T_m}$ represent the displacement vectors from the base platform origin and the mobile platform origin to their respective markers, then the transformation of coordinates from the marker reference frame to the Stewart platform reference frame can then be expressed as follows:

$$\boldsymbol{p'} = \boldsymbol{T}_{0\to1} - \boldsymbol{T_b} - \boldsymbol{T_m} \tag{5.18}$$

This transformation enables the determination of the actual mobile platform's pose $\boldsymbol{p'}$, which can then be compared to the desired position, $\boldsymbol{p}$.

The calibration methodology, developed by João Vale, employs a two-stage approach. Initially, a dataset comprising $n$ distinct end effector poses $\boldsymbol{p}$ is generated. For each $\boldsymbol{p}$, the corresponding actuator angles, $\alpha_k$, are determined via inverse kinematics. PWM values corresponding to each $\alpha_k$ are then calculated and the end effector pose is set via serial. The actual end effector position, $\boldsymbol{p'}$, is measured using a single camera system. The second stage involves an optimization procedure. This procedure aims to minimize the discrepancy between the observed and estimated leg lengths of the Stewart platform. The optimization utilizes the acquired $\boldsymbol{p'}$ data and an initial estimate of the platform parameters and servo offsets, $q_k^0$, to refine these estimates, ultimately enhancing the system's positional accuracy.

# Chapter 6

# Results

With the Stewart platform built, two primary tests were conducted:

1. Translations Test: This test evaluated the platform's ability to perform slow translations between points, simulating movements required for additive manufacturing at typical printing speeds;

2. Position Error Evaluation After Manual Calibration: This test assessed the pose accuracy of the platform after a manual calibration procedure.

## 6.1 Translations Test

The translations test involved commanding the platform's mobile platform to move between two predefined points at speeds comparable to those encountered in FDM technologies, $[50\,\mathrm{mm}\backslash\mathrm{s}, 150\,\mathrm{mm}\backslash\mathrm{s}]$. A simple control scheme was implemented, directly specifying the desired initial and final $\boldsymbol{p}$ and the desired manipulator velocity. However, execution of this test at desired printing speeds resulted in significant oscillations of the end effector. This undesirable behavior highlighted limitations in the current Stewart platform configuration, primarily stemming from inherent deadband in the actuation system.

Two primary factors contribute to the overall deadband: Electrical Deadband of Servos and Mechanical Deadband.

#### Electrical Deadband

Each servo possesses an electrical deadband, $\epsilon_q$, which represents the minimum change in the PWM signal required to result in a change of the servo's position. This electrical deadband is usually implemented as a way to reduce the effect of noise present in the PWM signal, which could lead to a servo constantly trying to adjust it's position in order to achieve its ever changing target. The employed *DSS-M15S* servos exhibit an $\epsilon_q$ of $3\,\mu\mathrm{s}$. This corresponds to an approximate angular resolution, $\rho_{\text{servo}}$, calculated as:

$$\rho_{\text{servo}} = \epsilon_q \frac{\Delta\alpha}{\Delta q} \tag{6.1}$$

For the *DSS-M15S* servos, this resolution is approximately $0.405°$. Preliminary testing suggests that this resolution, even without considering mechanical deadband, is insufficient for achieving the precision required for 3D printing applications. For example, when starting from its neutral position, the mobile platform needs to be instructed to translate at least $4.48\,\mathrm{mm}$ in the $x$ direction for all servos react to the change in PWM. This distance is about 2.5 times the nozzle diameter. If the translation was of lower calibre, the resulting position would not be accurate, as some of the servos did not experience enough of a change in the PWM to react.

**Mechanical Deadband**

The mechanical components of the Stewart platform, specifically the linkages comprising each arm and the servo gearboxes, introduce additional deadband. Physical gaps and backlash within these components allow for a small range of motion in the mobile platform before any servo actuation occurs to stop said motion. The experienced mechanical deadband is of approximately $12\,\mathrm{mm}$ in all directions. While this value is influenced by factors beyond gearbox backlash, it appears to be the dominant contributor.

## 6.2 Position Error Evaluation After Manual Calibration

To perform the calibration of the Stewart platform, initial estimates for the manipulator's parameters must be provided to the optimization function. The initial estimates for the servos' calibrated PWM ($q_k^0$) were determined visually, by judging whether the servo horns where close to their neutral position. The remaining parameters were initialized using values retrieved from the CAD model.

The calibration process involved capturing the platform's pose at 200 randomly generated points within a defined workspace. The maximum allowed translations in each axis was of $80\,\mathrm{mm}$ and the maximum allowed rotations in each axis, using the $xyz$ extrinsic convention, was of $17.2°$. After acquiring the set of observed points, $p'$, the optimization problem was ran.

However, the calibration procedure failed to provide an accurate estimation of the desired parameters, for the precision error in the observed points was too great. Comparing the set of obtained poses $\boldsymbol{p'}$ against the desired poses $\boldsymbol{p}$ resulted in a root mean square error of approximately $20.7\,\mathrm{mm}$. While the deadband issues discussed previously contribute to this error, the accuracy of the vision-based pose estimation system also comes into play.

The employed system, utilizing a single camera and ArUco markers, exhibited acceptable accuracy for position estimation in simplified scenarios (approximately $0.2\,\mathrm{mm}$ error). However, orientation estimation proved significantly less reliable, with errors reaching $1°$. When more complex scenarios are ran, the accuracy of both tend to worsen. To further degrade the estimation precision, the camera had to be placed significantly far from the platform to allow the capture of the entire workspace. This increased distance resulted in the capture of images that were more prone to error when computer vision algorithms were applied.

# Part III

# Free-Flyer with Manipulator

# Chapter 7

# Methodology

The combination of the two systems explained previously appears as a state of the art idea that could revolutionize the usage of additive manufacturing in many industries.

The combination of the dynamics and kinematics of both systems allows to simplify the controller to a unique model that is focused on the desired position of the end-effector only. With the constraints design it is possible to achieve a system where both the free-flyer part and the Stewart platform act according to same reference values. When tested on the table, these two systems would deposit material on a build plate mounted perpendicularly to the plane of the table. This means that prints, likely with supports along the way, could expand horizontally potentially encompassing much of the nearly $6\,\mathrm{m}^2$ table allowing adequate space to test lattice patterns likely used in space for large structures. With this evolution, the idea of a complete robot with a system that can act in a limitless space and manufacture a given object by printing is achieved.

## 7.1 Control

The printing system of ACROBAT will follow the schematic below:

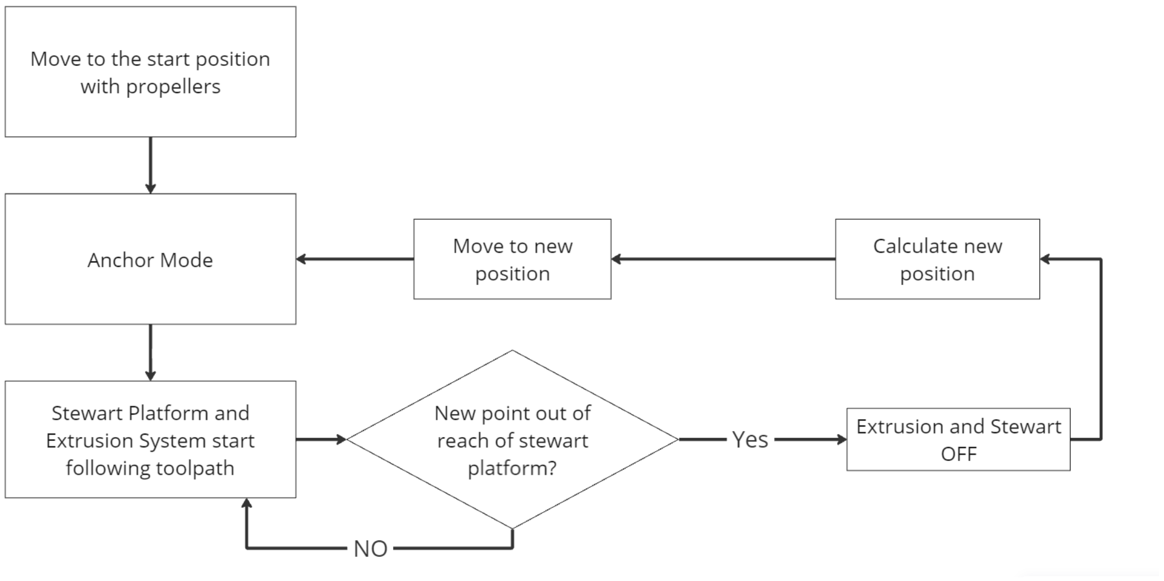

Move to the start position with propellers
Anchor Mode
Move to new position
Calculate new position
Stewart Platform and Extrusion System start following toolpath
New point out of reach of stewart platform?
Yes
Extrusion and Stewart OFF
NO

Figure 7.1: Printing System Schematic

This is an ideal case for the first iteration of this system, allowing to manufacture a 3 dimensional object on Earth with bigger dimensions than it is possible by any common 3D printer. In this version, there are two separate controllers active on the robot: one for the free-flyer and one for the Stewart platform. If the free-flyer controller has the ability to perfectly absorb all of the disturbances that the Stewart platform may generate through its own motion, then this system would be in itself ideal. However, this is not what is expected to happen in reality. A single controller which has both the free-flyer system and the Stewart platform modelled together could better guarantee that the end-effector stays in place so that the object being printed has minimal defects.

In the future, the main objective is to develop a control algorithm that follows a tool-path reducing the use of an anchoring system, allowing the free-flyer to be more active in the manufacture process and improving the velocity of the system. This is, nonetheless, much harder to perform. An initial step in this design development could be to neglect the Stewart platform's mass. This allows to simplify the kinematics of the overall system. After testing the implementation of this design, add these missing kinematics.

# Conclusions

To sum up, we made significant modifications and enhancements to the initial physical model of Acrobat based on insights we gained from previously developed systems. We completely re-engineered and built the free-flyer structure, including the test base with air bearings and the pneumatic system.

All the electronic components for the robot were procured and integrated. We made minor modifications to the original plan, such as the use of an Arduino instead of the PixHawk as a flight controller, for this system to ensure a functional Flat-Sat within the available time frame. This allowed us to finalize the electrical system in time to do tests to the free-flyer.

We developed the software entirely in ROS2 to facilitate communication among all components and to organize all the necessary functions, whether for user testing of the systems or for the control algorithms for the free-flyer and end-effector systems. Even though ROS is a complex tool, with an extremely steep learning curve, it simplified the work of the software flow and the communication between all the components needed for the functionality of the robot.

We developed and tested the Model Predictive Controller in a simulation environment using python and the library CasADi. Though it was not possible to develop intensive testing due to time constraints, we obtained some good results. The robot always followed the desired trajectory, or went to the position of reference, which proves that the system is well defined and controllable within the mathematical model presented in 2.1.3.

We redesigned the configuration of the Stewart Platform, which resulted in a threefold increase in the workspace. This improves massively the effectiveness of the system, as well as the printing capabilities of Acrobat. We reviewed and adapted all the work carried out in previous projects to fit the new model, allowing a fast implementation and calibration.

## 7.2 Obstacles

Acrobat is an involved, multifaceted project encompassing many disciplines. In the beginning, this led to the team quickly specializing and anxious to learn the tools they needed in order to complete the project. Whether it was programming control systems in Python and CasADi for the first time, diving into ROS2's difficult to approach architecture and hard to understand libraries, attempting and failing to use Gazebo to perform simulations, learning to use ArUcO vision markers and about camera parameters, tackling the inner workings of 3D printers, and so on, challenges were expected. Smaller issues like parts not fitting, difficult soldering, and lengthy waits for components were also expected. Yet, as long as these were the kinds of issues we were faced with, although ambitious, we were confident we could print something.

Ultimately, however, the large unforeseen issues are what led to us failing to fully meet our objectives. The two main issues were breaking the high pressure air compressor used to pressurize the air bearing system and failing to arm our flight controller. The former happened almost exactly halfway through the project when we were testing the air base and noticed that the tank would not pressurize. After sending it to be repaired, this shifted our priorities to trying to simulate the robot and control system in a Gazebo[16] rather than physically testing as originally planned. This drew focus away from developing ROS2 for a physical system and towards making it compatible with Gazebo, greatly delaying testing when a workaround was eventually found. It also took focus away from printing systems and the manipulator resulting in not enough time to make the systems truly collaborate to compensate for each of their limitations.

As for the flight controller, while the PixHawk has many advantages as a Flight Control Unit (FCU), it also comes with a number of limitations. Typically, this family of FCUs is used as a plug-and-play computer and comes with a software called PX4. This software has numerous safety conditions that are incompatible with the unique geometry of the robot. The absence of a manual kill-switch, GPS, and vertical actuation propellers prevented the system from being armed, which in turn blocked the PWM signal output from the Pixhawk. The only way to overcome the obstacle was to develop a document with all the geometry and configuration data of Acrobat, with some alterations in the source code. We decided to start using an Arduino, because the modifications would take a lot of time, because of the efforts that needed to be made to fully comprehend the software.

## 7.3 Future Work

Regarding improvements to the current free-flyer controller, we suggest that the implicit Euler method be adopted to estimate the kinematics of the skeleton. This is a more sophisticated approach than the simple derivation we applied. Another thing to have in mind is that the orientation references might not be followed in the most efficient way. Ideally, the robot should not rotate more than $180°$ for a given reference and the current controller does not have this constraint. Last but not least, having a real-time estimator for the parameters of the robot, specially when it comes to its own inertia, would be crucial since this robot is aimed to operate in a weightless environment. If the robot were to be made fully autonomous, then it should have the ability to dictate what is changing around him and in itself.

When it comes to electronics, during the test phase of the project the propellers were not as responsive as we expected them to be. On open-loop test, the propellers were fed with constant PWM signals so they should have rotated at constant speed. This was not, however, what happened.

Additionally, more RCbenchmark$^{©}$ tests should be performed to guarantee that the data related to the behaviour of the propellers which was extracted is not flawed. The Cobras may not be all equal from factory so this could also be taken into account and each motor would have its own separate polynomials for conversion between force and PWM signals.

For 3D printing to be achievable, careful consideration and investigation is needed on numerous aspects of the Stewart platform. A primary concern is the availability of affordable servo motors with sufficient resolution and minimal deadband to achieve the desired printing precision. Should commercially available servos prove inadequate, alternative actuation methods, such as stepper motors commonly employed in FDM printers, warrant exploration. Furthermore, increasing the accuracy of the calibration procedure should be a priority. Several paths can be taken to reduce the error of computer vision estimation, from utilizing a stereo camera system with higher image resolution to adopting more robust calibration pattern like the asymmetric circles pattern. If this approach proves inadequate for the desired precision, alternative calibration methods, such as laser-based methods, should also be considered.

Eventually firmware like those mentioned in 5.5.1 will be implemented calling for the development of a slicer that can create *.gcode* in 6 DoF. This project provides great potential for the mechanical engineering area of non-planar 3D printing, considering Acrobat's maneuverability. This research area is new and applied here could have tremendous benefits for prints giving them rigidity in all directions and eliminating de-lamination risks associated with planar printing. All in all, this project is not just state of the art in the area of robotics, but in many disciplines from control to mechanics and aerospace. We hope this technology will continue to develop and hopefully solve not just the logistics of space, but many of our pressing issues back home on Earth.

# Bibliography

[1] MIT. Mini MIT satellites rocketing to space station. https://news.mit.edu/2006/mini-satellites.

[2] NASA. Astrobee. https://www.nasa.gov/astrobee/.

[3] Intelligent Robot and Systems Group. Space Cobot. https://space-cobot.isr.tecnico.ulisboa.pt/.

[4] M. L. R. V. João Vale, Alexandre Rocha. A MULTI-OBJECTIVE OPTIMIZATION APPROACH TO THE DESIGN OF A FREE-FLYER SPACE ROBOT FOR IN-ORBIT MANUFACTURING AND ASSEMBLY. *AeroBest*, 137:517536, 2021. ID 40.

[5] J. Vale. Design of free-flyer robots for in-space cooperative additive manufacturing. M.Sc. Thesis in electrical and computer engineering, address = Lisboa, Portugal, month = November, note = , Instituto Superior Técnico, 2021.

[6] New Way air bearings. What is An Air Bearing? https://www.newwayairbearings.com/technology/design-basics/what-is-an-air-bearing/.

[7] R. Ventura and P. Roque. Space cobot: modular design of an holonomic aerial robot for indoor microgravity environments. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[8] M. Maier. Bidirectional thrust for multirotor mavs with fixed-pitch propellers. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8, 2018. doi:10.1109/IROS.2018.8593836.

[9] Bryan Morris. Kossel Delta Configuration 3D Printer. https://grabcad.com/library/kossel-delta-configuration-3d-printer-1.

[10] Z. W. Jianjun He, Hong Gu. Solving the forward kinematics problem of six-DOF Stewart platform using multi-task Gaussian process. *ResearchGate*, 2013. doi:10.1177/0954406212444508.

[11] M. Mamoon and Saifullah. Inverse kinematics and path planning of stewart platform using crank arm actuation system. In *Proceedings of 2014 11th International Bhurban Conference on Applied Sciences Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014*, pages 175–181, 2014. doi:10.1109/IBCAST.2014.6778142.

[12] Róbert Lőrincz. Smart Orbiter V3.0 Summary. https://www.orbiterprojects.com/so3/.

[13] klipper3d. Klipper Documentation. https://www.klipper3d.org/.

[14] Simulate Before You Build. https://gazebosim.org/home.

[15] T. M. Bocco. High accuracy pose estimation with computer vision. Master's thesis, Politecnico di Torino, 2021. URL https://webthesis.biblio.polito.it/17973/.

[16] Erik van der Zalm. Marlin Firmware. https://marlinfw.org/.