3-Axis Reaction Wheel Edge Balanced Cube

by

Gonzalo Belascuen

(31.58) Mechatronics Final Project

Submitted for obtaining the
Engineering Degree in Mechanical Engineering
at
INSTITUTO TECNOLÓGICO DE BUENOS AIRES
ESCUELA DE INGENIERÍA Y TECNOLOGÍA

Autor: Belascuen, Gonzalo
Legajo: 54414
Tutor: Ghersin, Alejandro

July 2021

Acknowledgements

To my mother, my father, my brother, my sister, my uncles and grandparents for supporting me.

To Alejandro Ghersin and Nicolas Nemirovsky for their guidance. To the company Skyloom Global Corp for letting me use their facilities to build the project.

Table of Contents

List of Figures

List of Tables

Acronyms and Definitions

Acronym	Description
RW	Reaction Wheel
CMG	Control Moment Gyro
IMU	Inertial Measurement Unit
PMSM	Permanent Magnet Synchronous Motor
BLDC	Brushless DC Motor
FOC	Field Oriented Control
EMF	Electromotive Force
ECS	Electronic Speed Controller
MCU	Microcontroller Unit
Li-Po	Lithium Polymer Battery
CG	Center of Gravity
SPI	Serial Peripheral Interface
IDE	Integrated Development Environment
ASCII	American Standard Code for Information Interchange

Table 1 – Acronym List.

1. Summary

This thesis presents the design and implementation of a 3-axis reaction wheel balanced cube capable of balancing on any of its 12 edges. Figure 1 shows a picture of the cube balancing on its edge. The first 3 sections present reaction wheel systems background, project motivation and objectives. This is followed by system electromechanical model and simulation description, which is used through the project.

Before embarking in the project, it was uncertain whether it was feasible to achieve wheel torque control within cost and time budgets. The high level design and feasibility addresses this question. This is followed by system definition and detailed design. Implementation , calibration and hardware problems encountered are described.

System identification was performed on the built system and the edge balancing control system was tuned using these results.

Vertex balancing and jumping from flat to edge functions were considered. However, these functions were left for future projects due to time constraints. To achieve vertex balancing with the build hardware, only the control algorithm is necessary. To achieve the jump up maneuver, a mechanical redesign to include a braking mechanism and higher inertia wheel is needed.

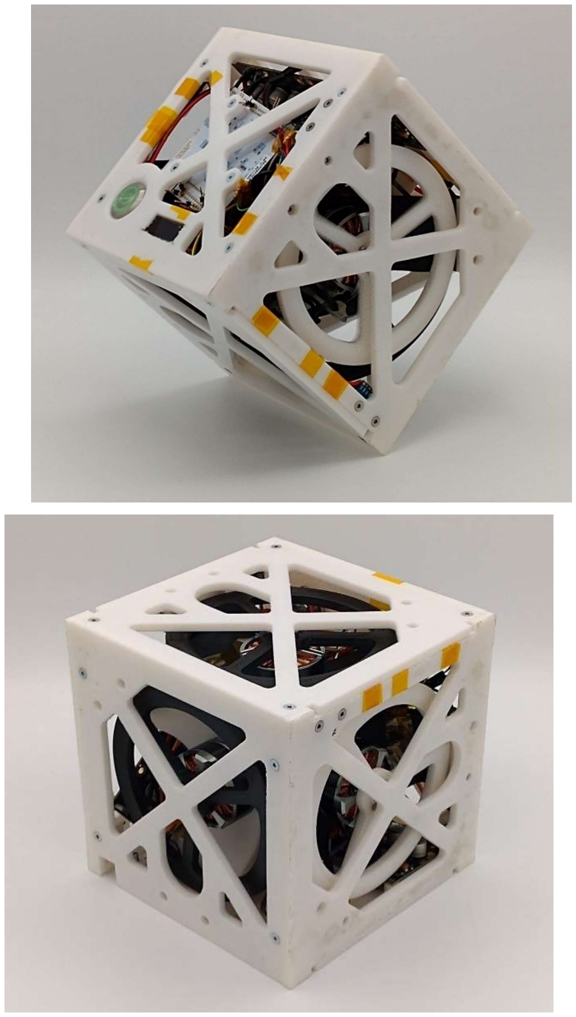


Figure 1 – Cube: Balancing on its edge (left). Flat on surface (right).

2. Introduction

2.1. Use of Flywheels

Flywheels are used to provide torques to a system without interacting with external objects. This is useful when external objects or the environment is unable to provide the torques required for the application.

Torque generation with flywheels can be divided into two main groups:

- Reaction Wheel (RW): Torque by changing the rotational speed of the flywheel. Generally used when refined torque control is needed.
- Control Moment Gyroscope (CMG): Torque by tilting the spin axis without necessarily changing the spin speed. Generally used when coarse (high force, low accuracy) torque control is needed.

Spacecraft Attitude Control Systems are a clear example of the use of flywheels. In these systems, flywheels are used as reaction wheels or as control moment gyros, depending on the requirements of each application. Figure 2 shows two examples of reaction wheels used in these systems. Figure 3 (right) shows an example of control moment gyros used in a large spacecraft.

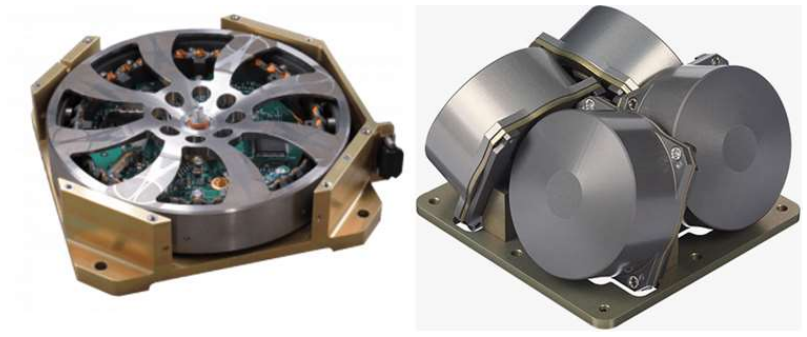


Figure 2 – Reaction Wheel Examples. CubeSat Reaction Wheel Attitude Control System. Single wheel (left). 4 RW assembly (right).

Another application example of flywheels are Boat or Ship Anti-rolling systems. In these applications, flywheels are generally used as control moment gyros given the capability of efficiently generate high torques. Figure 3 (left) shows a diagram of this system.

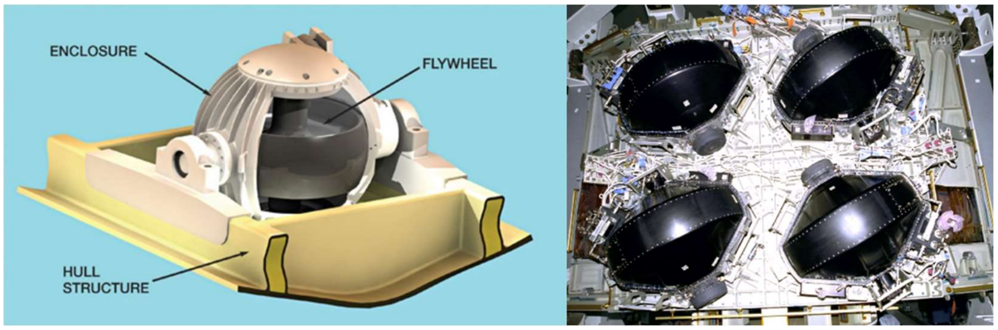


Figure 3 – Control Moment Gyro Examples. Ship Anti-rolling System (left). International Space Station Attitude Control System (right).

2.2. Motivation

This project was inspired by the *Cubli* project from ETH Zurich [1] (Figure 4). The main motivation for this project is to create a platform to allow mechatronics students at ITBA to develop reaction wheel attitude control systems, concepts that are applicable to the aerospace industry and mobile robotics in general. This project is the first iteration of a series of theses that will continue the work adding functionality.

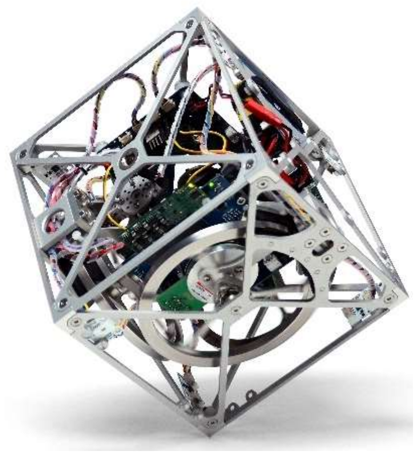


Figure 4 - Cubli (ETH Zurich).

3. Objectives

The objective of this project is to design, simulate and implement a 3-axis reaction wheel balanced cube with the following functions:

1. Edge Balancing: Ability to balance at any of the 12 cube edges if manually placed close to stable position.
2. Self-powered.
3. Simple User Interface.

These 3 high level requirements are flown down in Section 6.

The cube is also designed to perform the following functions, but the software and simulation to perform them is left for future work:

4. Vertex Balancing: Ability to balance at any of the 8 cube vertex if manually placed close to stable position.
5. Wireless Parameter Tunning & Control.
6. Wireless Data Transmission

The following two functions were considered and Section 5 describes a feasibility analysis of the first one, however it was decided not to add them to the final design due to time constraints and left for future work:

7. Jump from face flat to edge balancing position.
8. Jump from edge balancing position to vertex balancing position.

4. Models & Simulations

4.1. Edge Balancing

This section presents the dynamic model of a cube system during edge balancing. Models are physically derived, and Simulink simulation blocks are created.

The models are presented without value assignment to the parameters. Other sections show how these parameters are estimated or measured and they present the simulations results.

4.1.1. Dynamic Non-Linear Model

Figure 5 shows a schematic of the model for the edge balancing case.

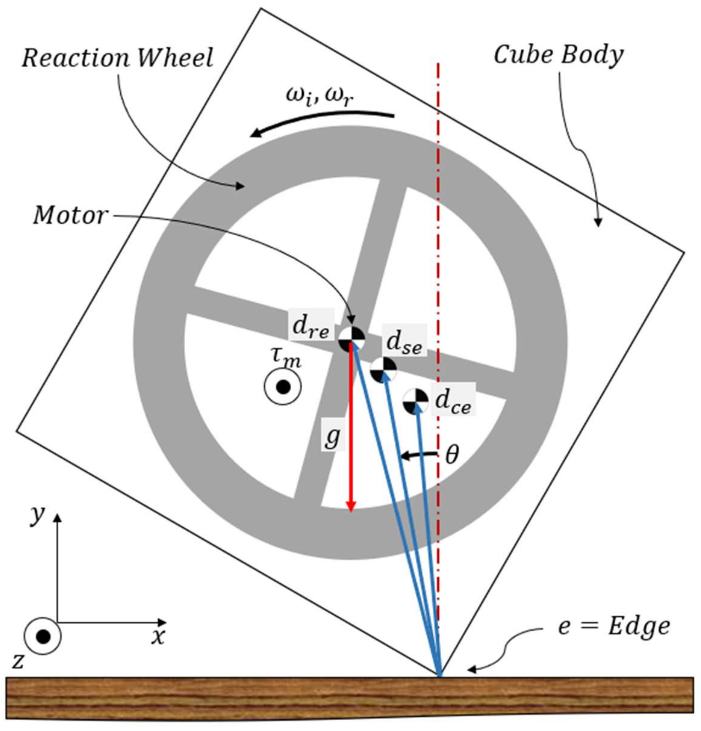


Figure 5 – Cube Model Schematic.

The system is modelled as two rigid bodies, reaction wheel and cube body, coupled by an electric motor, with the cube body rotating around the edge.

The cube body is characterized by a total inertia around the edge, a total mass, and a distance from that center of mass to the edge.

The reaction wheel is characterized by an inertia around its center of rotation.

The motor is modeled as a torque that generates an equal and opposite torque on these two bodies as shown in Figure 6. The motor mass and inertia are modelled by splitting it into its stator and rotor and attaching them to the cube body and reaction wheel, respectively.

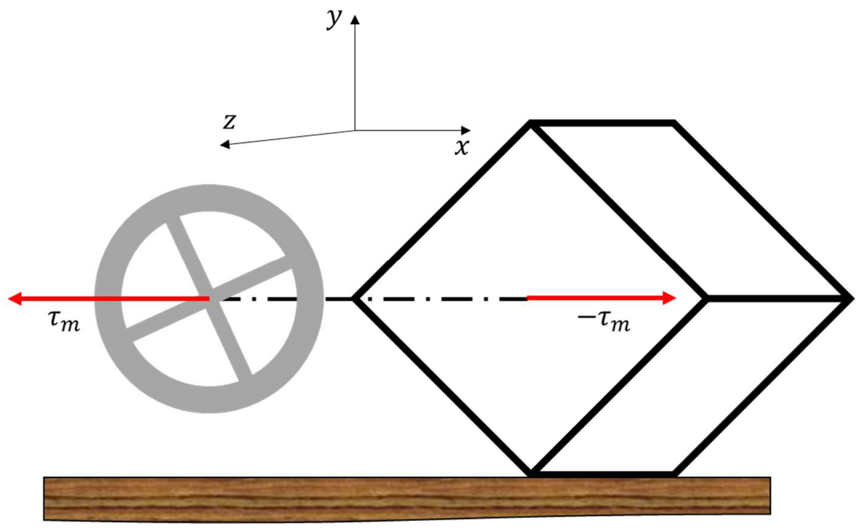


Figure 6 - Motor Torque Direction Definition. Torque action and reaction.

The system model is given by the following system of equations:

$$I_{se} \cdot \ddot{\theta} = k_{mgd} \cdot \sin\theta - \tau_m \tag{1}$$

$$I_r \cdot \dot{\omega}_i = \tau_m \tag{2}$$

$$\omega_r = \omega_i - \dot{\theta} \tag{3}$$

Equation (1) is the equation of motion of the cube body, where I_{se} is the system inertia with respect to the edge e derived in Equation (4), k_{mgd} is the gravity torque constant defined in equation (5), θ is the system center of mass angle with respect to the vertical, τ_m is the torque produced by the motor.

Equation (2) is the equation of motion of the reaction wheel, where I_r is the RW inertia around the axis of rotation and $\dot{\omega}_i$ is the inertial angular velocity.

Equation (3) is the relationship between the RW's inertial and relative angular velocities considering the cube angular velocity $\dot{\theta}$.

$$I_{se} = I_{ce} + I_{re} = I_c + M_c \cdot d_{ce}^2 + M_r \cdot d_{re}^2 \tag{4}$$

$$k_{mgd} = M_s \cdot d_{se} \cdot g \tag{5}$$

Table 2 shows the characteristic parameters of the system, the rest of the parameters are derived from these.

Item	**Parameter Name**	**Sym**	**Unit**
Cube Body	Mass	M_c	kg
	Rotational inertia with respect to its center of mass	I_c	$kg\ m^2$
	Center of mass distance to edge	d_{ce}	m
Reaction Wheel	Mass	M_r	kg
	Rotational inertia with respect to its center of mass	I_r	$kg\ m^2$
	Center of mass distance to edge	d_{re}	m
Motor	Torque Constant	k_t	$\frac{Nm}{A}$
	Resistance	R	Ω

Table 2 – Characteristic Parameters.

Table 3 shows the derived parameters calculated from the values in Table 2.

Item	**Parameter Name**	**Sym**	**Unit**
Cube	Total rotational inertia with respect to edge	I_{ce}	$kg\ m^2$
Reaction Wheel	Center of mass rotational inertia with respect to edge	I_{re}	$kg\ m^2$
System	Total rotational inertia with respect to edge	I_{se}	$kg\ m^2$
	Total mass	M_s	kg
	Center of mass distance to edge	d_{se}	m

Table 3 – Derived Parameters.

Table 4 shows the system variables.

Variable	**Unit**	**Name**
θ	rad	System center of mass angle with respect to vertical
ω_i	$\frac{rad}{s}$	Reaction wheel inertial angular velocity
ω_r	$\frac{rad}{s}$	Reaction wheel angular velocity relative to cube
τ_m	Nm	Motor Torque

Table 4 – Variables.

Figure 8 and Figure 9 show the Simulink models of equations (1) and (2) respectively.

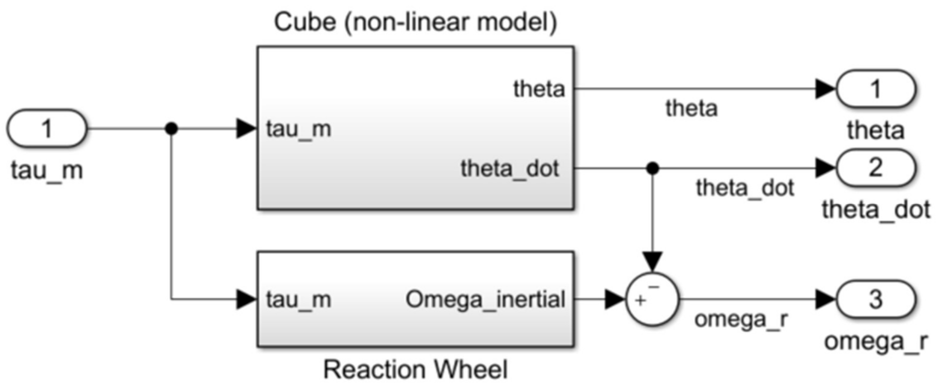


Figure 7 - Cube + Reaction Wheel. Simulink Model. Models Equations (1), (2) and (3).

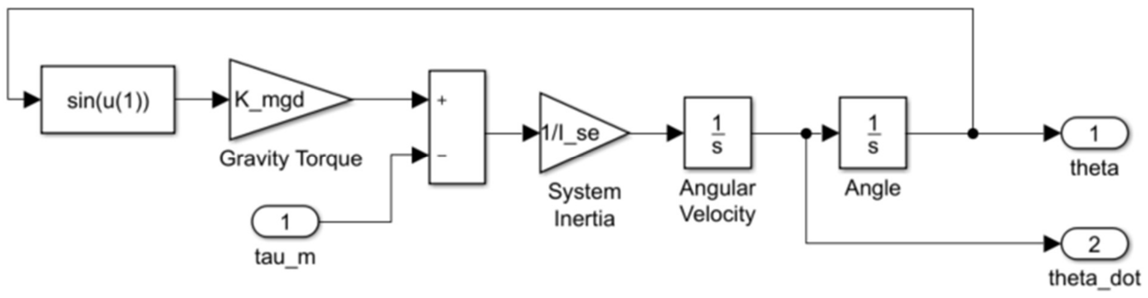


Figure 8 – Cube (non-linear model). Simulink Model. Models Equation (1).

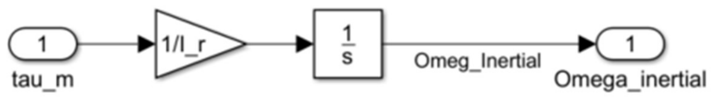


Figure 9 – Reaction Wheel. Simulink Model. Models Equation (2).

4.1.2. Linearized System

The system described in equations (1), (2) and (3) is linearized around the steady state point shown in equation (6).

$$\begin{cases} \theta^* = 0 \\ \omega_r^* = 0 \end{cases} \tag{6}$$

The only nonlinear term in these equations is the $\sin(\theta)$ present in Equation (1). The linear term of the Taylor expansion of $\sin(\theta)$ around $\theta^* = 0$ is θ. Therefore, Equation (1) is linearized by replacing $\sin(\theta)$ with θ.

Within the control loop, the required motor torque will be generated by commanding current using a motor controller. Equation (7) shows this relation.

$$\tau_m = k_t\ i \tag{7}$$

Equations (8) is the system of equations of the linearized system, including the motor.

$$\begin{cases} I_{se} \cdot \ddot{\theta} = k_{mgd} \cdot \theta - k_t\ i & \text{(8a)} \\ I_r \cdot \dot{\omega}_i = k_t\ i & \text{(8b)} \\ \quad \omega_r = \omega_i - \dot{\theta} & \text{(8c)} \end{cases}$$

4.1.3. State Space Representation

The system of equations that model the system presented in Equation (8) can be represented in state-space by defining the following states:

$$\begin{cases} x_1 = \theta \\ x_2 = \dot{\theta} \\ x_3 = \omega_r \end{cases} \tag{9}$$

These three turn out be related to the three energy storage mechanisms present in the system: potential energy with θ, cube angular kinetic energy with $\dot{\theta}$, reaction wheel angular kinetic energy with ω_r.

The general state-space representation of linear systems is given by [2]:

$$\begin{cases} \dot{\boldsymbol{X}}(t) = A\,\boldsymbol{X}(t) + B\,u(t) \\ \boldsymbol{Y}(t) = C\,\boldsymbol{X}(t) + D\,u(t) \end{cases} \tag{10}$$

where:

$\boldsymbol{X}(t)$ is the state vector:

$$\boldsymbol{X}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} \tag{11}$$

The output vector $\boldsymbol{Y}(t)$ includes the three states, given that they can be measured or estimated. ω_r is measured by the encoder on the wheel, $\dot{\theta}$ is measured by the gyroscope and θ is estimated by a complementary filter with measurements from the accelerometer and gyroscope; This algorithm is explained in Section 10.1.

$$\boldsymbol{Y}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} \tag{12}$$

$u(t)$ is the input vector:

$$u(t) = i \tag{13}$$

State space matrices A, B, C, D are found by expressing the system of equations in (8) in terms of the state variables defined in (9) and their first order derivatives.

$\dot{x}_1$ is found trivially by:

$$\dot{x}_1 = \dot{\theta} = x_2 \tag{14}$$

$\dot{x}_2$ is found by isolating $\ddot{\theta}$ from Equation (8a):

$$\dot{x}_2 = \ddot{\theta} = \frac{k_{mgd}}{I_{se}} x_1 - \frac{k_t}{I_{se}} i \tag{15}$$

$\dot{x}_3$ is found by substituting (8c) with $\dot{\omega}_r$ from Equation (8b) and $\ddot{\theta}$ from Equation (8a):

$$\dot{x}_3 = \dot{\omega}_r = \dot{\omega}_i - \ddot{\theta} = \frac{k_t}{I_r} i - \frac{k_{mgd}}{I_{se}} x_1 + \frac{k_t}{I_{se}} i \tag{16}$$

$$\dot{x}_3 = -\frac{k_{mgd}}{I_{se}} x_1 + \frac{I_r + I_{se}}{I_r \cdot I_{se}} k_t \, i \tag{17}$$

Therefore, the state spaces matrices are the following:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ \frac{k_{mgd}}{I_{se}} & 0 & 0 \\ -\frac{k_{mgd}}{I_{se}} & 0 & 0 \end{bmatrix} \tag{18}$$

$$B = \begin{bmatrix} 0 \\ -\frac{k_t}{I_{se}} \\ \frac{I_r + I_{se}}{I_r \cdot I_{se}} k_t \end{bmatrix} \tag{19}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{20}$$

$$D = [0] \tag{21}$$

4.1.4. Motor Driver and Motor Model

As it will be described in Section 5.1, the drive system selected consists of a PMSM motor (Permanent Magnet Synchronous Motor) driven by a three-phase field oriented control driver.

4.1.4.1. Field Oriented Control

The working principle of this method is to set the magnetic flux direction generated by the current at the coils to be perpendicular to the magnetic field generated by the permanent magnets. Figure 10 shows this condition. This makes it possible to generate a constant torque with a PMSM motor.

Setting the flux perpendicular to the magnet axis generates a constant torque at the motor shaft that is proportional to the total current in the windings, as shown in equation (22a).

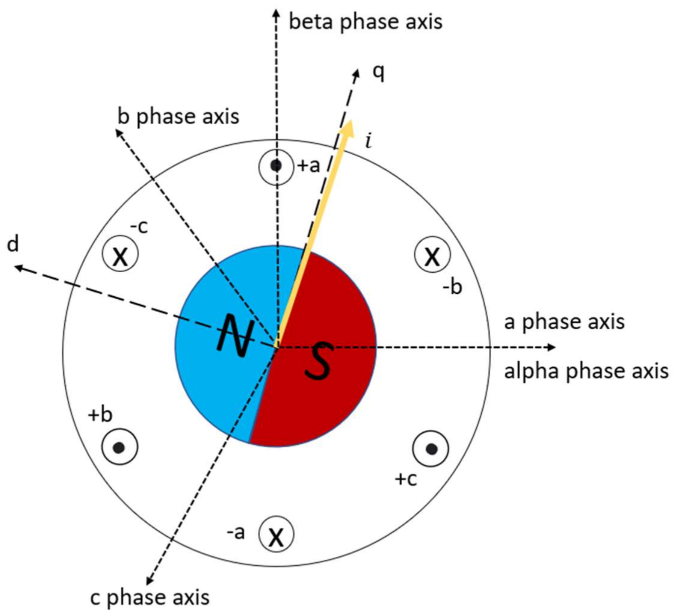


Figure 10 – PMSM schematic showing total current generated by the coils i being perpendicular to the magnet axis d (parallel to q).

This technique is achieved by a closed loop control as shown in Figure 11 where the permanent magnet position is measured by an encoder and flux by measuring the coil currents. The Clarke Transformation transforms the measured current at the three coils from the coil direction to alpha-beta cartesian directions. The Park Transformation rotates the current vector at the rotor speed measured by the encoder, in this frame, they shall be stationary and has two components, i_d aligned to the magnet and i_q perpendicular to the magnet. To achieve constant torque at the rotor, $i_{d\,ref}$ shall be kept at 0 given that it does not generate torque, and $i_{q\,ref} = \frac{\tau_m}{k_t}$ set by the desired torque.

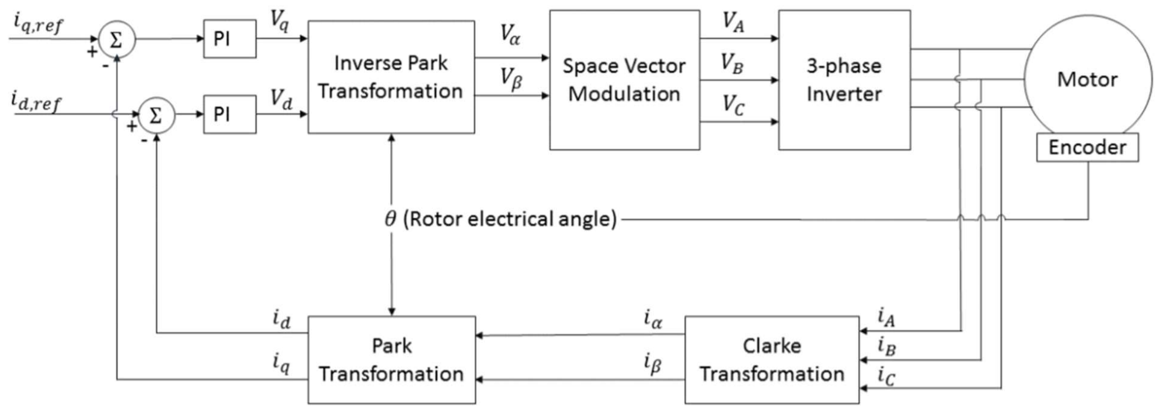


Figure 11 – PMSM field-oriented control block diagram.

For this project, it is assumed that the FOC controller will perform as described and the current commands sent to the module will generate such current perpendicular to the motor axis.

Therefore, the motor is modelled by [1]:

$$\begin{cases} \tau_m = k_t \cdot i & (22a) \\ i = (u - k_e \cdot \omega_r)/R & (22b) \end{cases}$$

Where, τ_m is the torque output, k_t is the motor torque constant, k_e is the motor voltage constant., i is the 3-phase current magnitude at the motor windings, ω_r is the motor angular velocity, u is the input 3-phase voltage magnitude at the motor windings.

The FOC driver is modelled by:

$$\begin{cases} i = i_{max}\ if\ \ i > i_{max} \\ i = -i_{max}\ if\ \ i < -i_{max} \\ \\ u_c = (i + k_e \cdot \omega_r)/R \\ u_c = u_{max}\ if\ \ i + k_e \cdot \omega_r > u_{max} \\ u_c = -u_{max}\ if\ \ i + k_e \cdot \omega_r < -u_{max} \end{cases} \tag{23}$$

where i_c is the reference control current, i is the real motor current, i_{max}, i_{min}, u_{max}, u_{min} are the maximum and minimum currents and voltages that the driver can generate.

Finally, the motor is modelled in Simulink as shown in Figure 12.

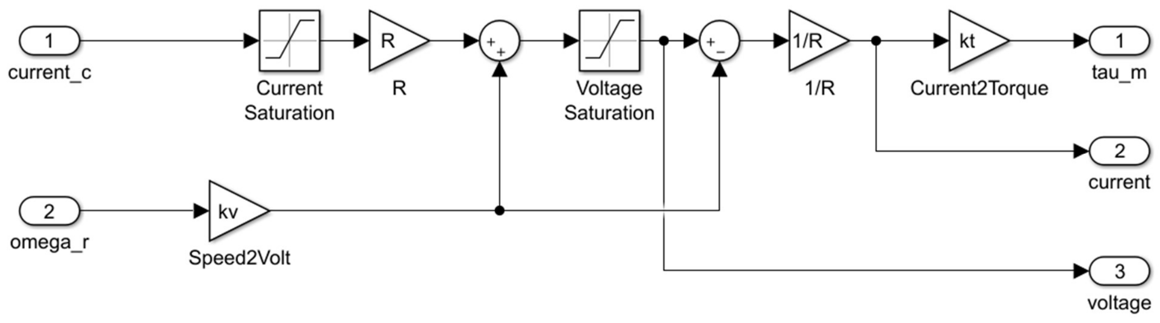


Figure 12 –Motor Driver and Motor. Simulink Model. Models Equations (22) and (23).

4.1.5. Control System

The system is controlled by a discrete time control system implemented in a microcontroller. This control system measures the system states $(\theta,\ \dot{\theta},\ \omega_r)$ and calculates the control signal τ_c to be sent to the motor controller.

To achieve this, the linear continuous state space model of the system is discretized with a sampling time T using a zero-order hold. Equation (24) shows the discrete state space model. G and H matrices are found by discretizing the A and B matrices using a zero-order hold sampling with period T, equations (25) and (26) show the discretization equations.

$$\bar{x}[k+1] = G\,\bar{x}[k] + H\,u[k] \tag{24}$$

$$G = e^{AT} \tag{25}$$

$$H = \int_0^T e^{A\sigma} d\sigma\, B \tag{26}$$

Using the discretized system model, a Linear Quadratic Regulator (LQR) feedback controller is designed. Equations (27) and (28) show the form of this controller:

$$u[k] = -\bar{K} \cdot \bar{x}[k] \tag{27}$$

$$u[k] = K_1\theta[k] + K_2\dot{\theta}[k] + K_3\omega_r[k] \tag{28}$$

Where $\bar{K}$ is the feedback gain vector $[K_1,\ K_2,\ K_3]$, $\bar{x}[k]$ is the state vector $\left[\theta[k],\ \dot{\theta}[k],\ \omega_r[k]\right]$, and $u[k]$ is the controller output, which is equal to the controller output torque τ_c.

The feedback gain vector $\overline{K}$ is found by the Optimal Linear Quadratic Regulator method, which minimizes the cost function (29) given the constraints from equations (24) and (27).

$$J = \sum_{n=1}^{\infty} \left[x^T[k]Qx[k] + u^T[k]Ru[k] \right] \tag{29}$$

Where:

Q is a positive semi-definite self-adjoint matrix which assigns weights to the states.

R is a positive definite self-adjoint matrix which assigns weight to the controller output.

MATLAB is used to compute the following: G and H matrices using the *c2d()* function, Optimal feedback gain $\overline{K}$ is with the *dlqr()* function.

Finally, the control algorithm is modelled in Simulink as shown in Figure 13.

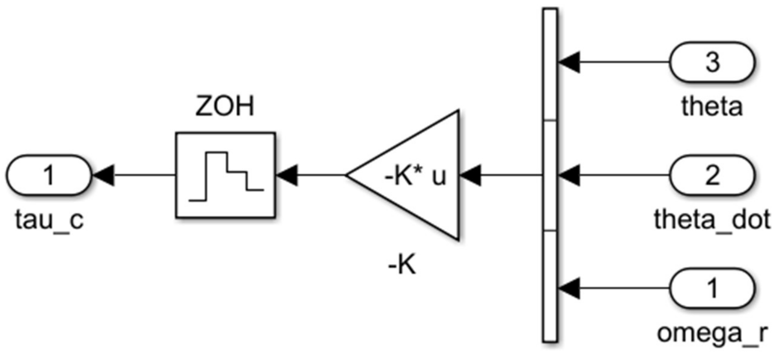


Figure 13 – Digital LQR Controller. Simulink Model. Models Equations (28).

4.1.6. Full System Model

Finally, Figure 14 shows the Simulink model of the edge balancing case, putting together all the models for the individual components presented previously in this section.

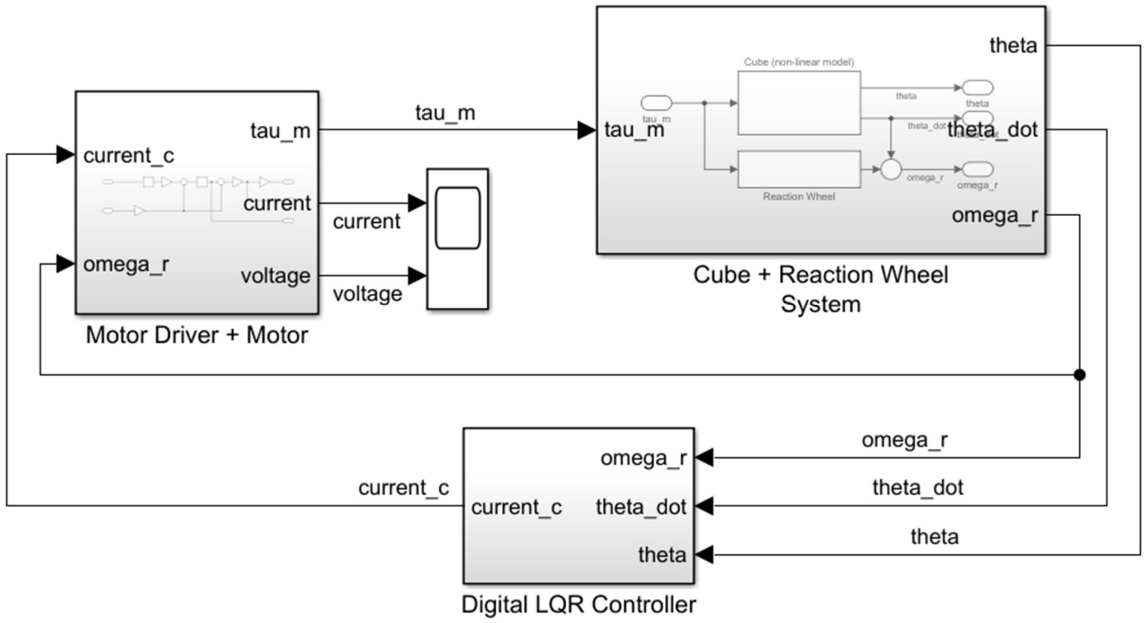


Figure 14 - Cube Edge Balancing Non-linear Model. Simulink Model.

In the next sections, this model will be used: firstly, to perform a feasibility analysis of the project and secondly, to model the implemented system.

5. High Level Design & Feasibility Analysis

Before embarking on the detailed design, component selection and implementation of the project, a feasibility analysis was performed to evaluate whether this project is feasible within the budget and time constraints available. To perform this feasibility analysis, a high level design with key component selection and considerations was done.

The functional requirements considered in this feasibility analyses are:

- Balancing on an edge.
- Jumping up from a flat on the table position to vertical on an edge, as shown in Figure 15.

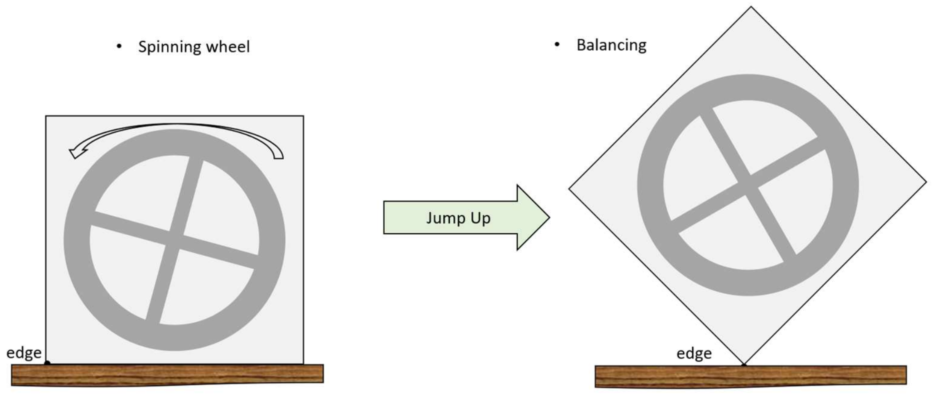


Figure 15 – Jump Up descriptive diagram.

The following parts are designed for this high-level design:

- Motor Driver Board Selection
    - The key enabler technology for the edge balancing functionality is a controller board capable of controlling the torque at any cube speed, that is also within the project budget.
- BLDC Motor Selection

    - In combination with the battery selection, this component is designed for the jump up functionality, to achieve the required maximum velocity to generate enough reaction wheel momentum to lift the cube.
- Battery Selection
    - Designed in combination with BLDC motor selection to set maximum speed for the jump up maneuver.
- General Mechanical Design
    - Designed for both edge balancing and jump up functionality.
- Wheel Holder & Braking Mechanism
    - Designed for jump up functionality to provide the required braking torque in a simple mechanical design.
- Reaction Wheel
    - Designed in combination with BLDC motor and battery to provide enough momentum for the jump up function. This is because it can be assumed that a wheel that can lift the cube, is also capable of balancing, but not vice versa.

5.1. High Level Design

5.1.1. Motor Driver Board Selection

This section discusses the way in which the key enabler for the edge balancing functionality is to procure a low-cost driver board that implements Field Oriented Control on the PMSM motor.

5.1.1.1. Field Oriented Control is Required for Torque Control

Field Oriented Control is the required motor driving method for this application given that it can precisely control the torque generated by the Permanent Magnet Synchronous Motors (PMSM) that shall be used given that they are the most convenient type of motor for this application.

To balance the cube, the electric motor shall be able to generate the torque commanded by balancing control loop, this is not a trivial task to achieve with PMSM motors, and it is harder to achieve at lower speeds, which is the desired speed of the motor when the cube is balanced. To achieve PMSM torque control, a specialized motor driver needs to be used that performs an algorithm known as Field Oriented Control (FOC), also known as vector control.

FOC is described in detailed in Section 4.1.4.1. The key characteristics of this method are that it can:

- Keep the windings magnetic field perpendicular to the permanent magnet field. This maximizes the efficiency of the motor.
- Control the magnitude of the windings magnetic field by controlling the current magnitude.

These two conditions together generate the torque desired at any speed. This is achieved by a closed loop control that measures the relative angle between the windings and the permanent magnet with a high-resolution encoder, and the current at the windings.

Conventional electronic speed controllers used in multirotor brushless dc motor control are not suitable for this application given that they do not actively control the winding magnetic field to set it perpendicular to the magnet's and therefore generating a smooth torque. Instead, these drivers use six-step commutation, which consists in dividing a full revolution in 6 60° sectors and applying a constant voltage while the magnet poles are at each sector. The location of the magnet pole is achieved either by hall effect sensors (only indicates the sector), or by measuring the back EMF at the coils. Controlling PMSM with this method will produce a rippling torque when the controller is set to generate a 'constant' torque, given that the winding and magnet fields misalignment will go from -30° to 30° when traversing each sector. Figure 16 shows the six-step commutation method [3] [4].

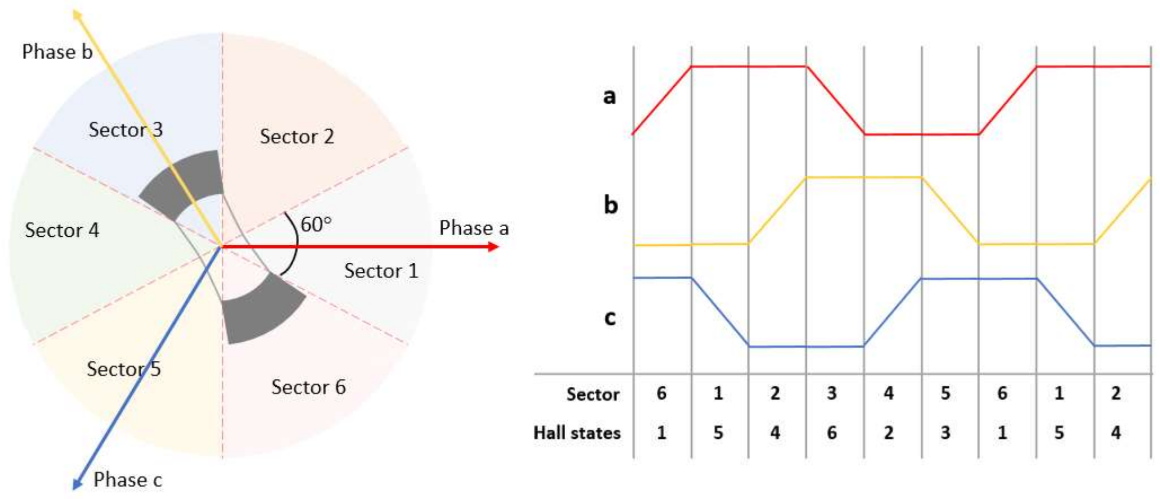


Figure 16 – Six-step commutation method. (right) voltage applied at each sector. (left) diagram showing sector locations relative to phases and magnet.

To conclude, procuring a motor driver board that implements Field Oriented Control is the key enabler for the feasibility of this project.

5.1.1.2. Field Oriented Control Driver Board Selection

Given the scope of this project, a ready to use, simple interface FOC driver board shall be procured rather than developed. Also, given the budget constraints, the driver board shall also have a low cost. Table 5 shows the comparison of the driver boards considered.

ODrive is selected for this project as it is the cheapest option when considering driving the 3 motors, each board is cheaper than the other options and given that each board can control two motors, only two boards are required. ODrive also provides the highest peak current of the considered options, which make the cube able to take bigger disturbances while balancing as shown in the simulations. Lastly, ODrive has an active online forum where implementation issues are discussed and can prove useful when implementing the system.

Supplier	Model	FOC	Price to drive 3 motors	Peak Current	Driver Complexity	Projects used
Maxon	EPOS4 24/1.5	Y	274x3 = 822	1.5A	Simple	Cubli
ODrive	ODrive V3.6	Y	120x2* = 240	120A	Simple	Stanford Doggo
Nanotec	CL3-E-2-0F	Y	147x3 = 441	6A	Complex	
Trinamic	TMCM -1640	Y	156x3 = 468	5A	Complex	

Table 5 – Motor Driver Board Options Comparison (*each board drives 2 motors).

5.1.2. BLDC Motor Selection

The main characteristics of BLDC motors and their selection criteria are as follows:

- **Motor Speed Constant:** Vendors specify the KV value which corresponds to the RPM generated per volt applied. We symbolize it as $k_v\left[\frac{V}{rad/s}\right]$ and the conversion from KV to k_v is shown in Equations (31) and (32).
  - **Selection Criteria:** Together with the battery voltage, they specify the maximum reaction wheel speed the system can use to jump up. The lower this parameter, higher wheel speed and greater momentum to transfer to the cube to reach the top position.
- **Motor Torque Constant:** Symbolized as $k_t\left[\frac{Nm}{A}\right]$ is the relation between the current at the windings and the torque produced. Without motor friction and other non-idealities, it is theoretically identical to k_v with a change in units from $V\,s$ to $\frac{Nm}{A}$ shown in Equation (30).
  - **Selection Criteria:** The motor torque constant together with the maximum designed current (commented below) specify the torque the control system will be able to produce. The higher this parameter, higher torque which leads

to a control system capable of stabilizing after bigger disturbances. The torque at zero speed (stall torque) is considered when evaluating this constant.

- **Electrical Inductance:** does not influence the motor selection given that the effect produced by these motor inductances is negligible.
- **Electrical Resistance:** Determines the current generated per volt applied, and given that current then determines the torque, together with the motor torque constant, we get the relationship between the battery voltage and the torque applied.
    - **Selection Criteria:** It will determine whether the stall torque of the motor is current limited by the driver or voltage limited by the battery voltage.
- **Mass:** Lower mass leads to greater margin in both jump up and edge balancing.
- **Price:** A significant consideration given the project tight budget.

$$V\,s = \frac{W}{A}\,s = \frac{J}{s\,A}s = \frac{J}{A} = \frac{Nm}{A} \tag{30}$$

$$\left[\frac{RPM}{V}\right] = \frac{\frac{2\pi\,rad}{60\,s}}{V} = 0.1047\left[\frac{rad}{s\,V}\right] \tag{31}$$

$$k_e[Vs] = \frac{1}{KV\left[\frac{RPM}{V}\right]} = \frac{1}{KV\cdot 0.1047}\left[\frac{sV}{rad}\right] \tag{32}$$

To summarize, regarding the cube jump up and edge balancing functions, the criteria is:

- **Jump Up:** Requires high reaction wheel velocity, therefore low motor speed constant.
- **Edge Balancing:** Requires high stall torque, therefore high motor torque constant, low resistance.

Table 6 shows survey of the BLDCs found in the market at the time. The maximum RPM and Stall torque are calculated for each motor assuming a $15V$ battery and a maximum allowable current of $50A$, which is half the rated current of both the motor driver and battery.

Power dissipation was also considered and is shown in the table. The case considered is applying 15V and holding the wheel for $100ms$ and the figure of merit is the change in temperature this would induce on the motor. The motor is thermally modelled as a lump of copper with half the motor mass, absorbing all the electrical thermal power generated. Equation (34) represents the thermal model of the motor.

$$\Delta T = \frac{P\ t}{c_t\ m} \tag{33}$$

Where $P = 15V * i_{stall}$ is the electrical power dissipated, m is the lump of copper mass which is considered to be half the motor mass, $c_t = 385 \frac{J}{kg\ C°}$ is the copper specific heat, and $t = 100ms$ is the time considered. From the table all options considered present a ΔT of less than 3.7°, which is a manageable amount.

Finally, the selected part was the *T-Motor MT4008 KV380*, mainly due to its low cost and in Section 5.2, it is shown that the maximum RPM and stall torque it provides are enough to achieve jump up and balancing. During assembly, one of the 3 purchased motors was damaged, and a replacement part was not available, therefore a *SunnySky V4008 KV380* was also purchased, which has the same KV value and very similar resistance and generates the same stall torque and maximum RPM as the T-Motor.

Brand	Model	KV	R	W	Kv	Kt	I stall	T Stall	RPM Max	P	ΔT	Price
		RPM/V	Ohm	g	mVs	mNm/A	A	mNm	RPM	W	°C	USD
T-Motor	MT4008 KV380	380	0.15	113	25	25	50	1257	5700	375	0.9	38
SunnySky	V4008 KV380	380	0.131	105	25	25	50	1257	5700	328	0.8	59
SunnySky	V4008 KV600	600	0.075	104	16	16	50	796	9000	188	0.5	59
SunnySky	V4004 KV300	300	0.448	51	32	32	33	1066	4500	502	2.6	55
SunnySky	V4004 KV400	400	0.288	51	24	24	50	1194	6000	720	3.7	55
SunnySky	V4006 KV320	320	0.23	66	30	30	50	1492	4800	575	2.3	55
SunnySky	V4006 KV380	380	0.17	66	25	25	50	1257	5700	425	1.7	55
SunnySky	V4006 KV740	740	0.047	68	13	13	50	645	11100	118	0.4	55
SunnySky	V5208 KV340	340	0.078	175	28	28	50	1405	5100	195	0.3	78
Maxon	EC 45 flat 30W	187	4.83	75	51	51	3	159	2805	46.6	0.2	80
Maxon	EC 45 flat 50W	201	2.83	110	48	48	5	252	3015	79.5	0.2	118
Maxon	EC 32 flat 15W	397	3.51	57	24	24	4	103	5955	64.1	0.3	79
Maxon	EC 32 flat 15W	195	13.8	57	49	49	1	53	2925	16.3	0.1	79
Maxon	EC 45 flat 50W	285	1.03	110	34	34	15	488	4275	218	0.5	118
Maxon	EC 45 flat 30W	374	1.2	75	26	26	13	319	5610	188	0.6	80
Maxon	EC 45 flat 50W	380	0.464	110	25	25	32	813	5700	485	1.1	118

Table 6 – Brushless Motor Selection Table.

5.1.3. Battery Selection

Lithium Polymer (Li-Po) is the most widely used battery technology in the Radio-Controlled hobby market and it is the battery technology selected for this project for the following advantages it provides:

- Wide range of low-cost battery options
- Wide range of low-cost battery charger options
- High current output

Li-Po batteries have the following characteristics of interest:

- S: this number represents the number of cells in series connected internally in the battery package. Given that each cell provides an average of 3.7V and connecting cells in series sums their voltage, this parameter specifies the nominal voltage of the battery. Increasing the number of cells also increases the size and weight of the battery, which negatively impacts performance. Finally, higher cell count come at a higher cost.
- Capacity (C): this is the amount of current that the battery can deliver, and it is specified in milliamp-hours [mAh]. Higher capacity batteries are heavier.
- Peak Output Current: is a dimensionless number that specifies the output current as a multiple of the battery capacity. To calculate the output current in Amperes, take the battery capacity C, divide by 1-hour and multiply by the specified dimensionless value.

A market survey was performed and is summarized in Table 7.

#	**S**	**V**	**Capacity**	**Mass**	**Discharge**	**Price**	**Size**			**Seller**	**Link**
			mAh	g	C	USD	mm	mm	mm		
1	3	11.1	450	41		28	56	30	12	Sypom	Link
2	4	14.8	430	51		30	49	30	17	MaxAmps	Link
3	4	14.8	800	76		42	68	24	25	MaxAmps	Link
4	4	14.8	450	54	75	15	45	27	24	Amazon	Link
5	4	14.8	450	76	150	16	56	31	25	HobbyKing	Link
6	4	14.8	1300	126	100	17	69	32	29	Amazon	Link
7	5	18.5	1050	175		64	68	35	35	MaxAmps	Link
8	6	22.2	1050	210		76	68	35	42	MaxAmps	Link
9	6	22.2	1000	212		27	75	35	43	HobbyKing	Link
10	6	22.2	1000	182		24	74	5	39	HobbyKing	Link
11	6	22.2	1000	172	100	24	76	36	35	Amazon	Link
12	6	22.2	550	109	80	20	31	31	61	Grayson	Link

Table 7 – Li-Po Battery Survey (Selected part marked in green).

Cell count S has the following impact on design: Increasing this number increases the maximum wheel speed (adds margin to cube jump) and the peak torque generated (adds margin to controller output saturation). On the other hand, it adds weight (decreases jumping and balancing margins).

The battery charger was also considered when selecting the battery. One parameter of these chargers is the number of cells it is capable of charging, which conditions the S number during battery selection. Another parameter is the charging current; it is recommended to charge Li-Po batteries at a 1C rate, this means that the charger provides a current equal to the capacity of the battery divided by 1 hour, which in turns ends up charging the battery in about 1 hour. During selection, the cheapest charger found was the Enegitech e430, which charges up to 4 cells and has a minimum charging current of 1A, therefore restricting the list from Table 7 to only 4S and capacity greater than 1000mAh (lower capacities will lead to charging at >1C which would lead to battery damage).

Capacity C is designed by performing an estimation on the amount of energy required to operate the cube. The figure of merit considered is the number of wheel accelerations from 0 to maximum speed that the battery can power. The energy required to accelerate the wheel once is estimated by the simulation shown in Figure 17, which simulates the wheel acceleration and integrates the electrical and mechanical power used from 0 to maximum velocity. The parameters such as the wheel inertia and motor constant are described through this section, Table 9 summarizes these values. Figure 18 shows the result of this simulation.

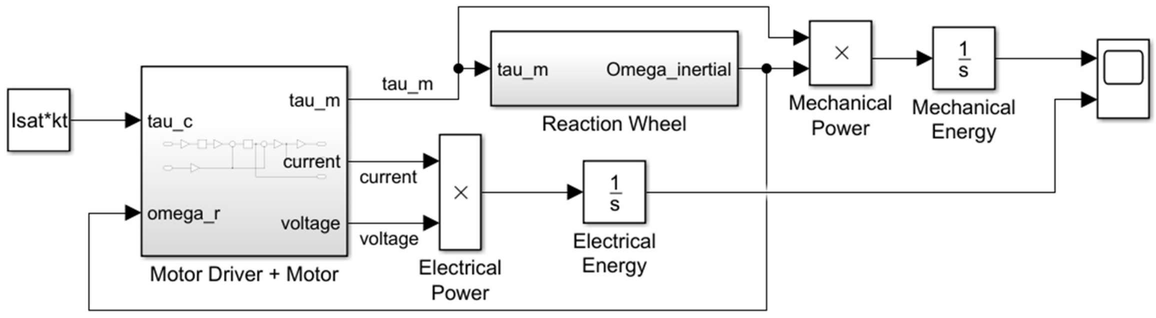


Figure 17 – Wheel Acceleration Simulation. Simulink Model.

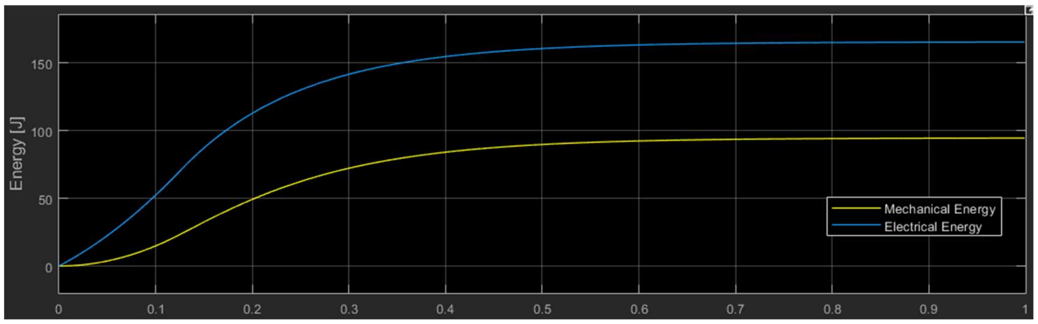


Figure 18 -Wheel Acceleration Energy. Simulation Result.

The simulation indicates that the energy required to accelerate the wheel to its maximum velocity is $165\,J$. After iterating with different battery options from Table 7, option #6 was selected (Figure 19). This battery has 14.8V and 1300mAh, storing $69kJ$. We can estimate the number of individual wheel accelerations this battery can achieve considering by roughly considering 50% energy transfer efficiency from the battery to the wheel, therefore $\frac{69000\,J}{165\,J}\cdot 50\% = 209$ wheel accelerations.

To conclude, 209 accelerations might be a large number for this application, and lighter batteries with less capacity as in options #4 or 5, however, this battery was selected mainly because of the charger availability.

Figure 19 – Selected Battery. Option #6. Ovonic 14.8V 1300mAh 100C 4S LiPo.

5.1.4. General Mechanical Design

This mechanical preliminary design is performed to get an estimate of the mass and inertia of the system and to assess the feasibility of a wheel braking mechanism. After geometrical iteration a side length of 150 mm was reached to fit the parts inside.

The design considers the following aspects:

- Cube structural panels are designed for 3D printing due to ease in manufacturability, light weight and allows to easily achieve a more complex geometry than cutting sheet metal would provide.
- Reaction wheels are made of steel, this follows from the jump up and balancing analysis which conclude that higher material density leads to better performance.
- The wheel holder ‘Arm’, which holds the motor, servo and braking mechanism is designed for 3D printing given its complex geometry.

Figure 20 shows an assembly view of the preliminary mechanical design.

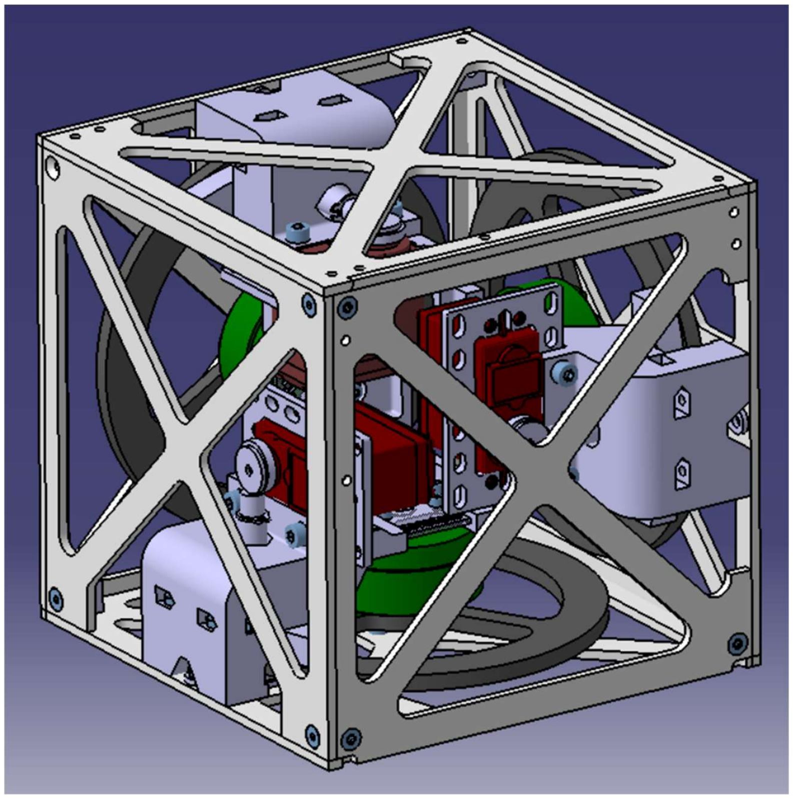


Figure 20 - Cube Preliminary Mechanical Design - Assembly View.

During mechanical design, it was decided that the braking system will not be included in the final design, therefore the progress of this version was stopped at Figure 20, not including the battery and electronic boards. However, the mass of the electronics and battery was considered for the jump up and balancing analysis.

5.1.5. Wheel Holder & Braking Mechanism

Figure 21 and Figure 23 show this assembly. The part called Arm holds the motor with 4 screws and the braking mechanism.

The braking mechanism consists of two bicycle brake pads placed in opposite sides of the wheel. One of the pads is held up by a spring and it is pushed down by a servo actuated cam-follower mechanism.

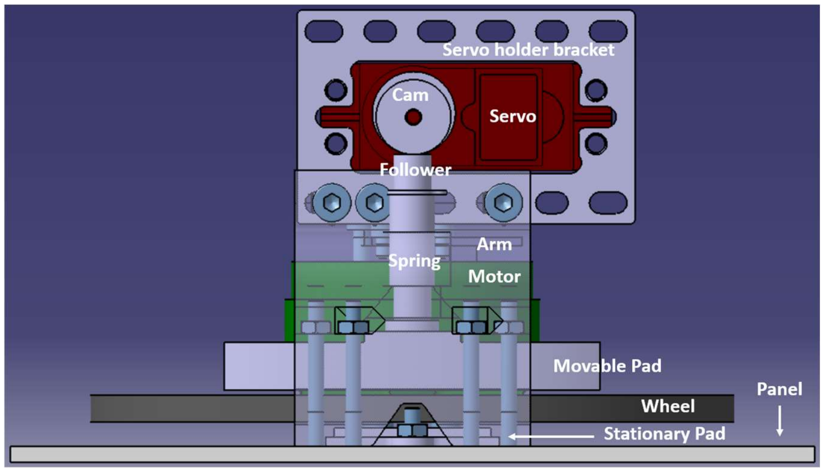


Figure 21 – Braking Mechanism Preliminary Design – Front View.

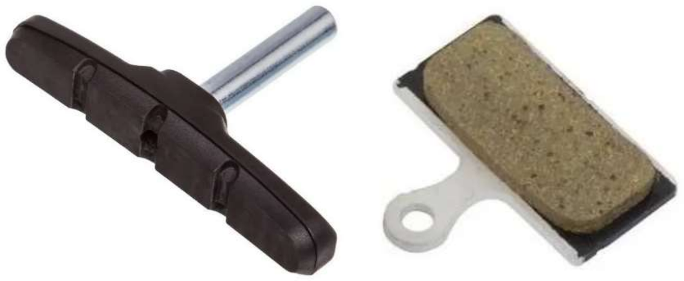


Figure 22 – Shimano M70t2 BR-M40 Cantilever brake pad (left). Shimano G02a brake Pad (right).

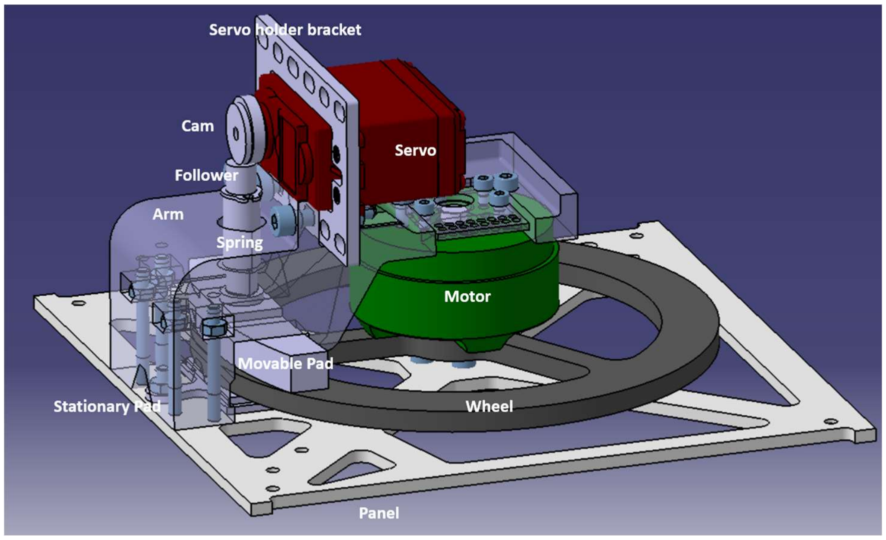


Figure 23 – Braking Mechanism Preliminary Design.

Cam-follower mechanism: the follower is the cylindrical end of the cantilever brake. For the cam, the challenging aspect is manufacturing an adapter to the 25-tooth spline that the servo actuator provides as drive shaft; this was solved by finding a part that already has the servo 25-tooth spline adapter and has enough extra material to machine the cam geometry. Figure 24 shows the servo actuator and the 25-tooth spline hub with enough material to machine the cam.

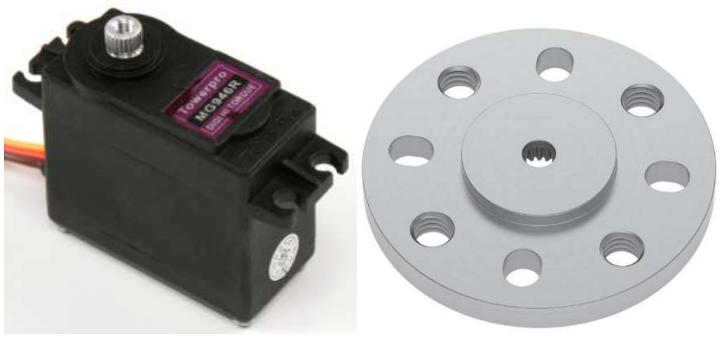


Figure 24 - MG946R Servo (left). Servo Hub 25 Tooth Spline (right).

The Servo Hub selected allowed to machine a 16mm diameter 1mm center offset circular cam, as shown in Figure 25.

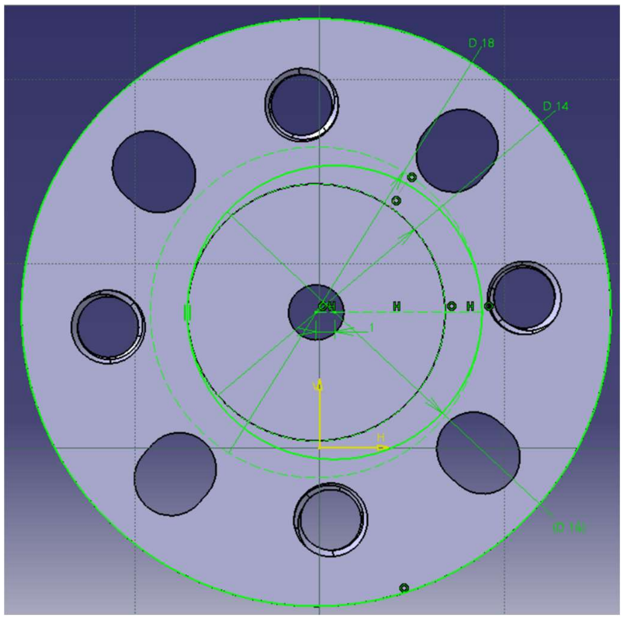


Figure 25 - Servo Hub Cam Machining.

This 1 mm offset generates the requirement that the sum of the upper and lower gaps shown in Figure 26 must be less than 1 mm such that when the servo is actuated, both pads touch the wheel. This constraint might be hard to achieve considering the arm and panel will be 3D printed, and any angle that the wheel has when mounted can also contribute to a displacement at the brake pads. A disadvantage of this design is that for the wheel to reach the stationary pad, the structure will have to deform the whole lower gap distance.

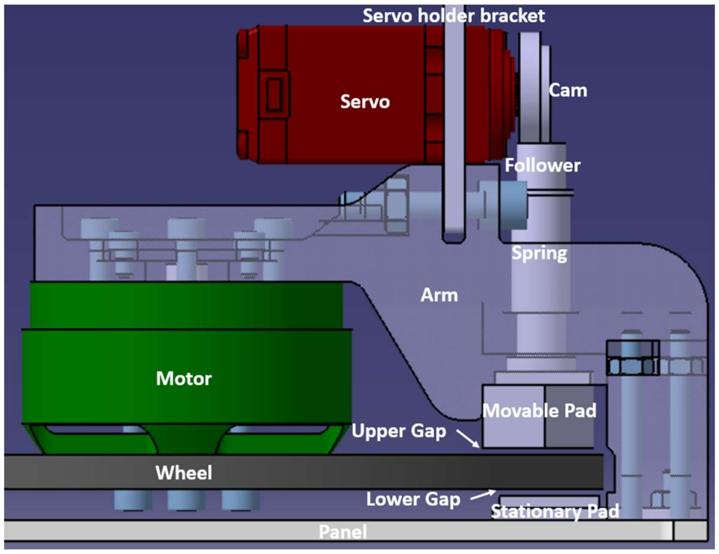


Figure 26 – Braking mechanism gaps.

5.2. Physical Feasibility Analysis

5.2.1. Mass and Inertia Estimation

For this estimation, only the most heavy parts were considered. These are the wheels, motors, braking mechanism and servo, battery, motor driver boards, microcontroller board and the structure.

Table 8 shows the items considered, the approximation performed and the result. The approximation considered is described by the following steps:

1. Perform mechanical design including motors, reaction wheels, braking mechanism, and structural panels.
2. Calculate mass and radius of gyration.

3. Add motor driver boards, battery, microcontroller masses to total mass.
4. Perform an estimation of the inertia including these items by assuming same radius of gyration but using the total mass calculated.

The formula used to calculate inertia I_{total} from mass M and radius of gyration r_{gy} is the following:

$$I_{total} = M \cdot r_{gy}^2 \tag{34}$$

Parameter	**Value**	**Eng Units**	**Comment**
Mass			
Motors+Wheels+Panels+Braking Mechanism+Structure	1.91	kg	From Catia Preliminary Mechanical Design
Driver Boards	30	g	Specification
Battery	125	g	Specification
Microcontroller	20	g	Specification
Total Mass	2.09	kg	
Inertia			
Radius of Gyration	113	mm	From Catia Preliminary Mechanical Design
Total Inertia at Edge Axis	266234	g cm^2	

Table 8 – Mass and Inertia Estimation Table.

5.2.2. Braking Torque Modeling

Figure 27 shows the position of minimum force transmitted from the cam torque to the follower, the braking torque estimation is performed at this position and will give a lower bound estimation, meaning that if this position provides enough braking torque, the other position will do so too.

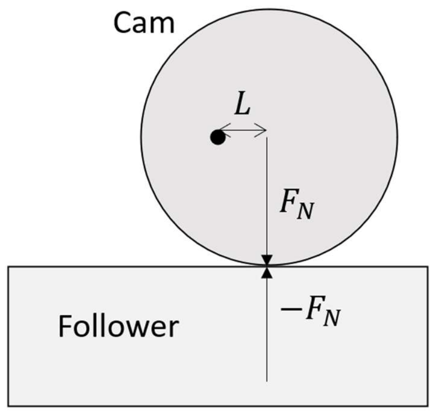


Figure 27 – Cam-lever free body diagram.

The servo generates a torque on the cam of the cam-follower mechanism. The rated torque for the MG946R servo is $\tau_{servo} = 13\ kg \cdot cm$. At the position shown in Figure 27, the lever arm corresponds to the designed $L = 1mm$ shown in 5.1.5. Therefore, the force on the follower, considering a 50% margin (rough estimate), is:

$$F_N = \frac{\tau_{servo}}{L} \cdot 50\% \ = 650N \tag{35}$$

The lower and upper braking pads will generate the same tangential force equal to:

$$F_T = \mu_{rubber-metal} \cdot F_N \ = 325N \tag{36}$$

Where $\mu_{rubber-metal} = 0.5$ [5].

To calculate the total braking force on the wheel rim, the tangential force generated by both braking pads surfaces must be considered, therefore:

$$F_{Brake} = 2\ F_t \ = 600N \tag{37}$$

The geometric center of the braking surface is estimated to be at $R_{Brake} = 55mm$, therefore the braking torque is:

$$\tau_{Brake} = F_{Brake}\ R_{Brake} \ = 35.75Nm \tag{38}$$

5.2.3. Jump Up Feasibility Assessment

This section will show an analytic approach to assess the feasibility of the jump up maneuver, next section will do so by Simulink simulation.

The jump up maneuver starts with the cube lying flat on the table and spinning the wheel at maximum speed, then the braking mechanism is actuated, and the braking torque generated while the wheel is slowing down serves to push the cube to the upright position. When the cube is near the equilibrium point, the edge balancing control system is engaged, and the cube will stay balancing at that equilibrium point.

This analytic approach divides the jump into 3 states with 2 processes as shown in Figure 28. The process starts by spinning the wheel, when the wheel reaches maximum velocity, braking starts.

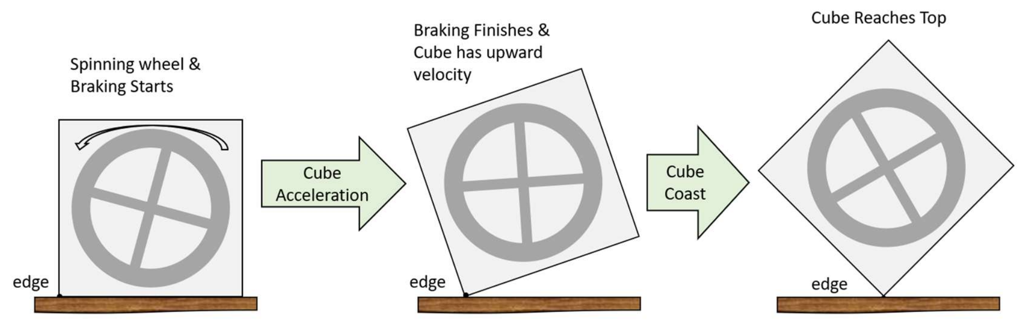


Figure 28 – Jump up maneuver decomposition.

During braking, the cube is modelled as a rigid body rotating with inertia around an edge with two torques being applied, braking torque and gravity torque. The braking torque shall be in opposite direction to the gravity torque.

$$I_{se} \cdot \ddot{\theta} = -\tau_G + \tau_{Brake} \tag{39}$$

Where I_{se} is the cube inertia around the edge, presented in Table 8.

To calculate the gravity torque τ_G, the diagram shown in Figure 29 is considered. This diagram shows the weight force F_W generating a torque with radius $R_W = \frac{L_{cube}}{2} = \frac{150mm}{2} = 75mm$.

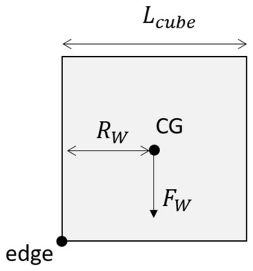


Figure 29 – Gravity torque on the cube diagram.

Considering the total cube mass M_{cube} shown in Table 8, the gravity torque is:

$$\tau_g = F_W \cdot R_W = M_{cube} \cdot g \cdot R_W = 1.53Nm \tag{40}$$

This is lower than the $\tau_{Brake} = 35.75Nm$ calculated in Equation (38), and therefore the cube will start moving upwards.

In the real case, the gravity torque will start decreasing as the cube starts moving upwards due to the weight lever arm R_W decreasing. For this analysis, we consider that the gravity torque will remain constant while the wheel is being slowed down, this is easier to calculate and it provides an upper bound model to the maneuver, therefore we consider the cube will accelerate following Equation (39) until the wheel reaches the current velocity of the cube, at that point, there will be no relative velocity between the wheel and cube and the braking torque will disappear.

To estimate the cube velocity and speed at the instant braking stops, the braking time shall be calculated. Considering the equation of motion of the wheel:

$$I_r \cdot \dot{\omega}_i = -\tau_{Brake} \tag{41}$$

Where I_r is the wheel inertia. The wheel was iteratively designed until this analysis concluded that the cube would jump up and reach the top. This is the wheel shown in Figure 20 and Figure 23.

$$I_r = 4988\ g\ cm^2 \tag{42}$$

Integrating the previous equation with initial condition ω_{i_0} and finding we get:

$$\omega_i(t) = \omega_{i_0} - \frac{\tau_{Brake}}{I_r}\ t \tag{43}$$

$$\dot{\theta}(t_{Brake}) = \omega_{i_0} - \frac{\tau_{Brake}}{I_r}\ t_{Brake} \tag{44}$$

$$t_{Brake} = \frac{\omega_{i_0} - \dot{\theta}(t_{Brake})}{\tau_{Brake}/I_r} \tag{45}$$

Integrating Equation (39) we get:

$$\dot{\theta}(t) = \frac{-\tau_g + \tau_{Brake}}{I_{se}} t \tag{46}$$

From battery selection Section 5.1.3 and motor selection Section 5.1.2, the initial wheel velocity is equal to:

$$\omega_{i_0} = \frac{V_{bat}}{k_v} = \frac{15\ V}{0.025\ \left[V\frac{s}{rad}\right]} = 600\frac{rad}{s} = 5732RPM \tag{47}$$

With equation (45) and (46) we get a braking time of:

$$t_{Brake} = 8.2\ ms \tag{48}$$

We can estimate how far the cube travelled this time by integrating Equation (46):

$$\theta(t) = \frac{\tau_G - \tau_{Brake}}{I_{se}}\frac{t^2}{2} \tag{49}$$

$$\theta(t_{brake}) = 2.46° \tag{50}$$

And at what speed the cube is when braking finishes with Equation (46):

$$\dot{\theta}(t_{brake}) = 10.5\ rad/s \tag{51}$$

Now with the cube angular velocity $\dot{\theta}$, we can conclude if the cube can reach the top by comparing the angular kinetic energy at this stage to the gravitational potential energy at the top. The cube total angular kinetic energy is given by:

$$E_\omega = \frac{1}{2} I_{se}\, \dot{\theta}^2 = 1.47\, J \tag{52}$$

To calculate the potential energy, we estimate the height left to travel from the $2.46°$ already traveled to the top, as shown in Figure 30.

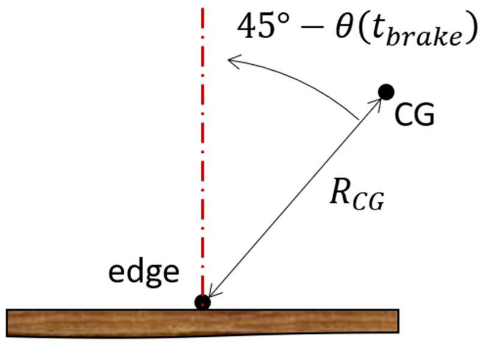


Figure 30 – Center of mass angle with respect to vertical.

$$E_G = M_{cube}\, g\, R_{CG}\left(1 - \cos\left(45° - \theta(t_{brake})\right)\right) = 0.57\, J \tag{53}$$

It can be concluded that the cube will reach the top given that the kinetic energy at the end of the wheel braking is 2.57 (1.47 J vs 0.57 J) times greater than the potential energy required to reach the top.

Braking using the wheel motor was also considered. The motor should provide more torque than the torque generated by gravity which is $\tau_g = 1.53$ as shown in equation (40). Considering the torque constant of the selected motor $k_t = 25\, mNm/A$, the current required to start lifting the cube is $I = \tau_g/k_t = 61.2\, A$, which is half of the maximum allowable current of the motor driver (120A) and what the battery can provide (130A). This shows the motor can start lifting the cube, however a simulation is needed to evaluate if it can make it reach the top.

5.2.4. Jump Up & Edge Balancing Simulation

This simulation is performed using the Simulink Models for each component presented in Section 4.1. A small change is made to add the braking function to the system model, as shown in Figure 31, the braking is modeled by switching the input torque to the wheel from a constant braking torque to the control system torque.

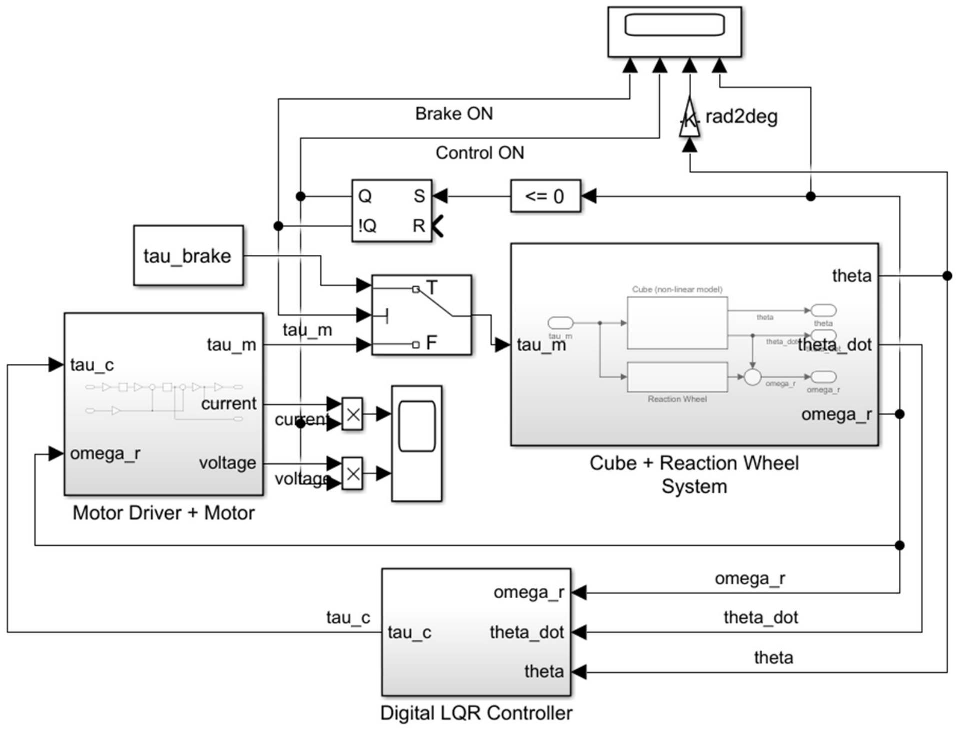


Figure 31 – Cube Jump Up and Edge Balance Non-linear model. Simulink.

This section presents two simulations:

1. Jump up without turning on the control system
2. Jump up and turning on the control system.

The parameters used for these simulations are the ones presented through this Section 5 and are summarized in Table 9.

Item	Parameter Name	Sym	Unit
Cube Body	Mass	M_c	$2.09\,kg$
	Center of mass inertia	I_c	$0.0267\,kg\,m^2$
	Center of mass distance to edge	d_{ce}	$0.106\,m$
Reaction Wheel	Mass	M_r	$0.219\,kg$
	Center of mass inertia	I_r	$5.25 \cdot 10^{-4}\,kg\,m^2$
	Center of mass distance to edge	d_{re}	$0.106\,m$
Motor	Torque Constant	k_t	$0.025\,\frac{Nm}{A}$
	Resistance	R	$0.15\,\Omega$
Braking	Initial Wheel Velocity	ω_{r_0}	$600\,rad/s$
	Initial Angle	θ_0	$-45°$
	Braking Torque	τ_{Brake}	$10\,Nm$
Battery	Voltage	V_{sat}	$15\,V$
Motor Driver	Maximum Current	i_{sat}	$50\,A$

Table 9 – Jump up and Balancing Model Parameters.

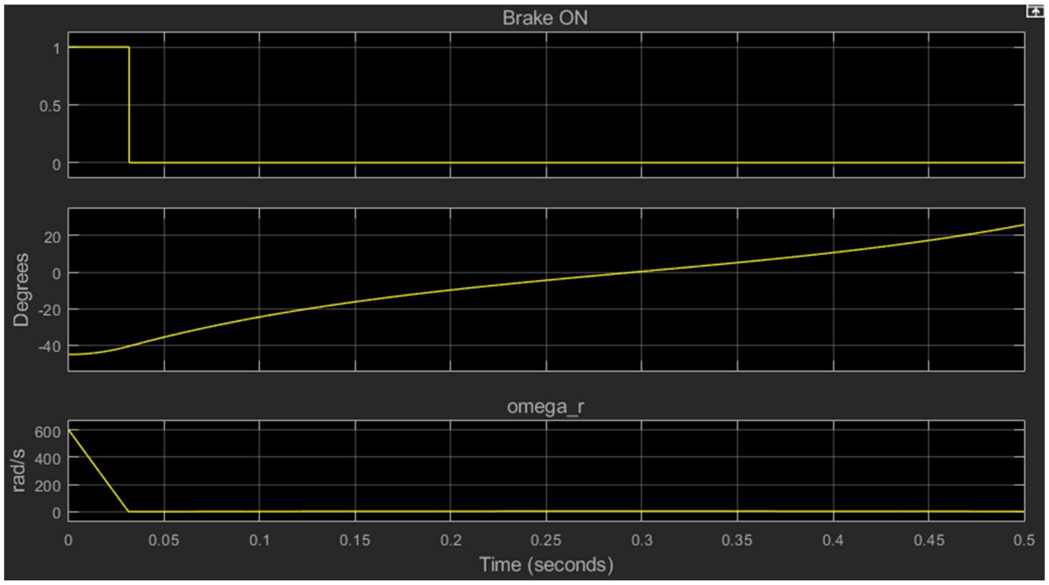


Figure 32 – Jump Up without turning on control system. Simulation Results.

Figure 32 shows the jump up without turning on control system simulation results. The simulation time is 0.5 seconds. The first plot shows a Boolean variable indicating if the braking is enabled. The third plot shows the reaction wheel relative speed starting at $600\frac{rad}{s}$ and being slowed down by the brake until it reaches zero. The second plot shows the angle of the cube, it starts at $-45°$ and begins to increase as the braking torque accelerates the cube upwards; the key point to note is that this angle crosses zero, which corresponds to the vertical position, indicating the jump up maneuver was successful.

Figure 33 shows the jump up and turning on balancing control system simulation results. The simulation time is 0.5 seconds. The first two plots show Boolean values indicating if the brake or control system are one. As in the previous simulation, the fourth plot shows the reaction wheel starting at maximum speed and being slowed down by the brake until it reaches zero. The third plot shows the cube angle starting at $-45°$ and asymptotically reaching zero as the control system stabilizes the cube at that position. Plots 6 and 7 show the current and voltage on the motor respectively, it can be seen the moment the control system is turned on and these variables start actuating and stabilizing the cube.

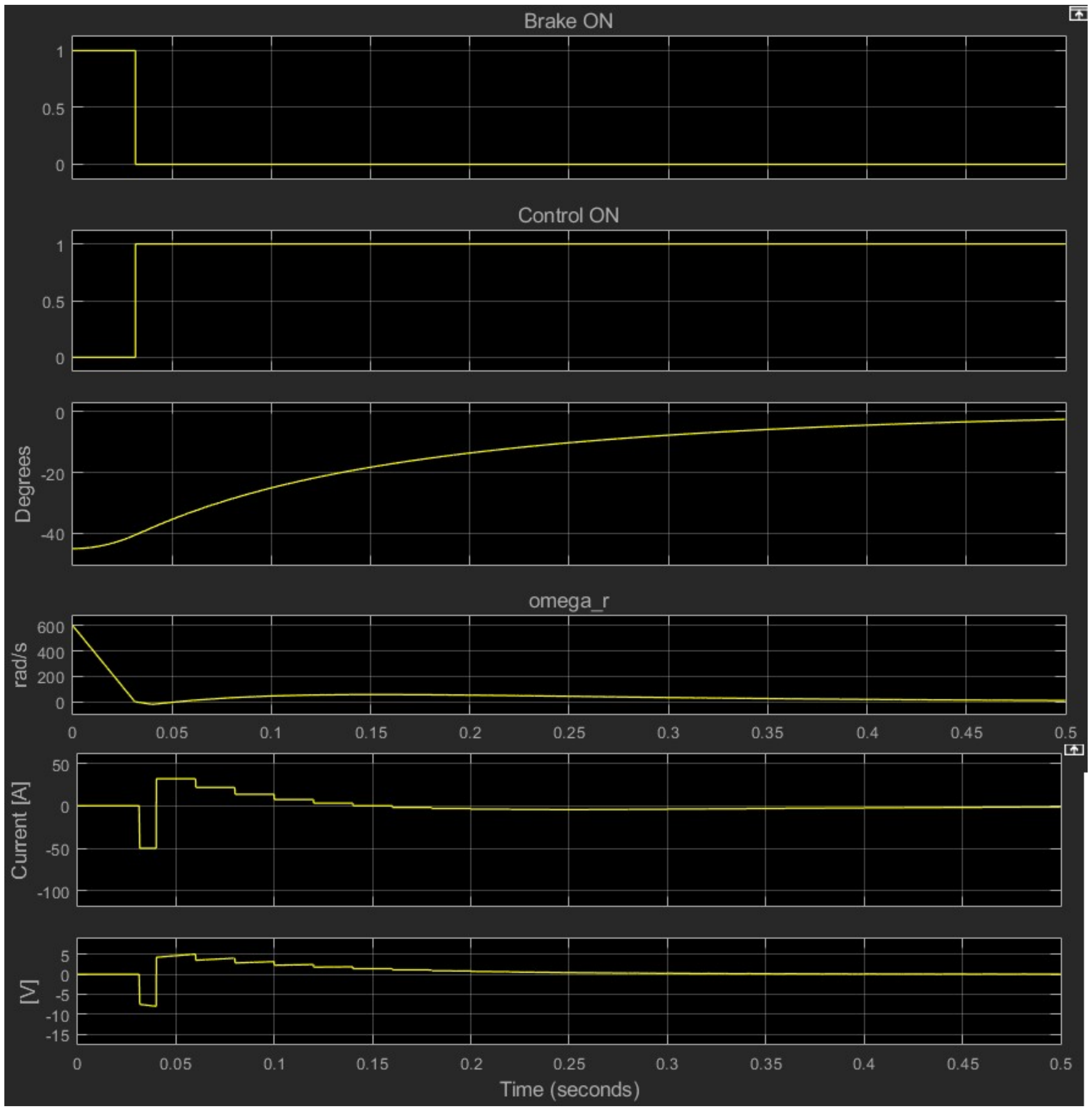


Figure 33 – Jump Up and turning on balancing control system. Simulation Results.

Section 4.1.5 shows the model for the LQR control system used. The plant poles are $[0\ Hz, 1.07\ Hz, -1.07\ Hz]$, considering this, the selected controller frequency is $50\ Hz$.

For the LQR, after iteration, the following Q and R matrices were used:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{54}$$

$$R = 10000 \tag{55}$$

Finally, Figure 34 shows the jump up maneuver using only the electric motor by spinning it at maximum velocity followed by accelerating the wheel in the opposite direction at the maximum allowable current of 120A and maximum voltage of 15V.

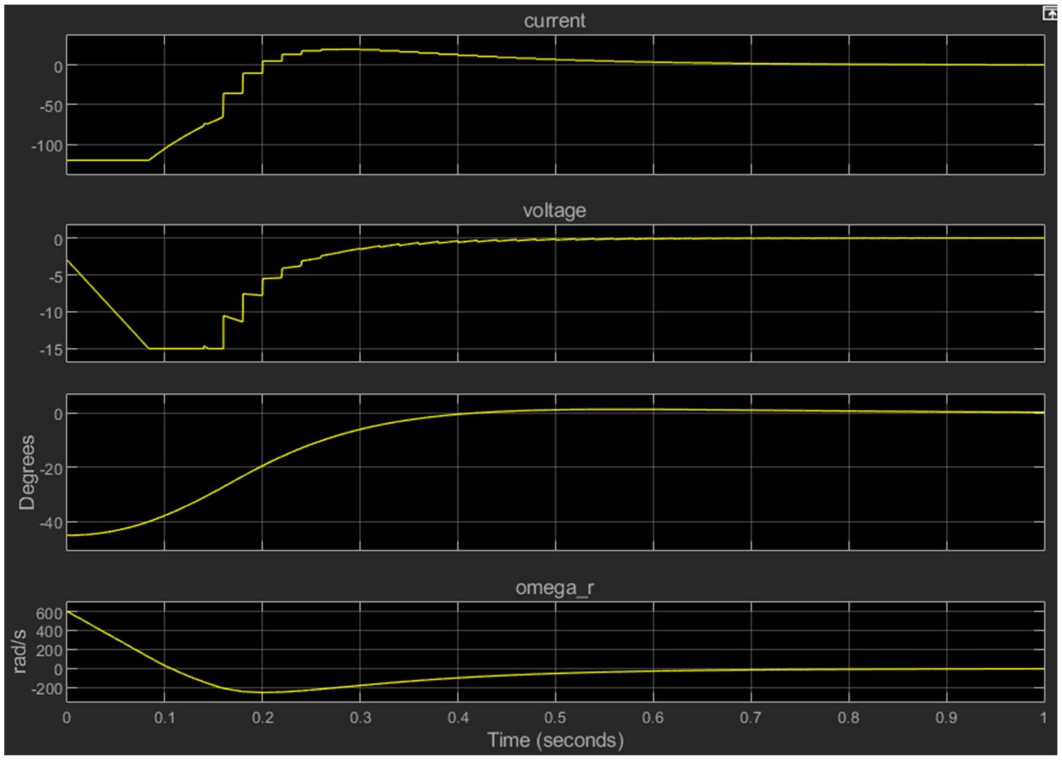


Figure 34 - Jump Up using electric motor. Simulation Results.

It can be seen that the cube is able to jump up and stabilize using the electric motor as brake. However, this method was not chosen given that a steel reaction wheel is required (as seen in Figure 35), and it was unfeasible to manufacture such a wheel for this project. Another reason for not using this method is that it requires to stress the board up to its highest rated current, which risks ruining hardware.

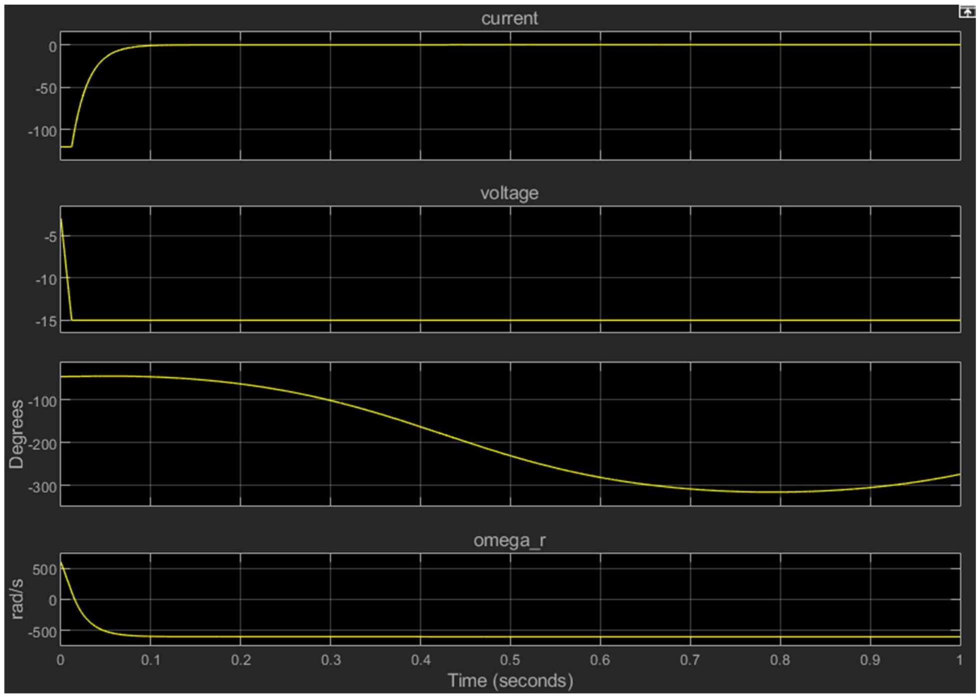


Figure 35 – Jump up using motor and plastic reaction wheel. Does not reach the top.

5.3. Economic Feasibility

Table 10 presents a preliminary part list with their corresponding quantity and cost. Some of the selected items were specified in the former sections. The cost of other parts was approximated.

		Item	Part Number	Provider	Unit Price	Q	Total Cost
Motor Panel 1,2,3	Actuator	Motor	T-Motor MT4008 KV380	Amazon	38	3	114
		Servo	MG946R	Todo Micro	10	3	30
	Electronics	Encoder	AS5047P	Digikey	17	3	50
		IMU	MPU6050	AliExpress	3	4	10
	Mechanical	Panel	PLA Plastic	3D Printed	0	3	0
		Reaction Wheel	Steel Round Bar	Aceros Cas	6	3	17
		Arm	PLA Plastic	3D Printed	0	3	0
		Servo Holder	1801-0040-0001	gobilda	3	3	9
		Servo Cam	1906-0025-0032	gobilda	3	3	9
		Cantilever Brake	Shimano M70t2 Br-m40	Mercado Libre	7	3	20
		Spring	Do 8.5, Di 7.1, L 18	Ferr/ Bulmars	2	3	6
		Retaining Ring		GATA	1	3	3
		Brake Pad		Mercado Libre	4	3	11
Electronics Panel 4,5,6	Electronics	Microcontroller	STM32	Mouser	10	1	10
		Motor Driver	ODrive V3.6	ODrive	138	2	276
		Communications	HM-10	Amazon	10	1	10
	Power Supply System	Battery	14.8V 1300mAh 100C 4S LiPo	Amazon	17	1	17
		Charger	Lipo Battery Charger	Amazon	30	1	30
		Electronics Step Down	Dc-Dc Lm2596	Amazon	3	1	3
		Servo Step Down	Dc-Dc Lm2596	Amazon	3	3	9
	Mechanical	Panel	PLA Plastic	3D Printed	0	3	0
Cube	Mechanical	Fasteners Estimate		GATA	15	1	15
			Shipping Estimate for all purchases		40	1	40
					Total USD		**688**

Table 10 – Preliminary Part List and Cost Estimation.

The total estimated cost is within the available budget.

5.4. Conclusion and Transition to Final Design

This chapter presented the preliminary design of a 3-axis balanced cube and it was shown by simulation that this design can jump up and balance on its edge.

When transitioning to the final design, it was decided to reduce the scope in functionality given that it would require more hours of work than intended for the project. It would help reduce the already stretched budget. Section 6 presents the full requirement specification for the final design.

The eliminated function is the jump up maneuver given that it would save a significant amount of time due to the following reasons:

- Requires steel reaction wheel which is expensive to manufacture.
- Braking mechanism tolerances. Might require iteration on the mechanical fabrication.
- High braking forces. Might damage parts, requiring iteration.
- Braking mechanism takes significant space inside the structure, might require iteration when integrating all parts.
- More electrical design and integration.
- More software developing and testing.
- More system testing.

The selected motor, motor driver board and battery presented in this section are preserved for the final design presented in the following section.

6. System Definition

Section 3 defined the high-level functional requirements for the cube. This section will flow down each high-level requirement to lower level requirements and design decision.

Figure 36 shows the building blocks used in the requirement flow-downs shown in this section.

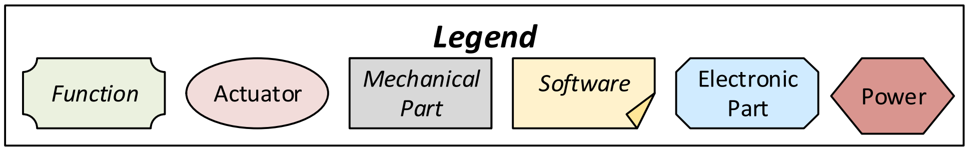


Figure 36 - Diagrams Legend.

6.1. Edge Balancing

This requirement specifies the ability to balance the cube at any of its 12 edges. Figure 37 shows the requirement flow down starting with the high level requirement of edge balancing and ending at the sensors, actuators, electronics boards, power systems and software required to achieve the high level function.

To achieve edge balancing we need three main sub functions:

- Torque Generation
- Attitude Sensing
- Control System

Torque generation is performed by a reaction wheel attached to a torque controlled electric motor. Torque control drive is a key function required for this application and the feasibility of such a system is discussed in Section 5.1.1 . A torque controlled motor requires a motor controller board and an encoder attached to the motor's axis. The requirement to balance the cube in any of its 12 edges derives in requiring 3 torque controlled reaction wheels in the 3 cube orthogonal axis, each wheel will be able to balance the cube in the 4 edges parallel to the wheel's axis.

Attitude sensing is performed by Inertial Measurement Units (IMUs), which consist of an accelerometer and a gyroscope integrated into the same chip, sharing the same mechanical axis and spatial position. Attitude estimation from acceleration and angular velocity can be done with the following algorithms:

- Assuming acceleration measurement is equal to the gravity vector.
- Acceleration and Gyroscope sensor fusion.
    - Weighted average
    - Kalman filtering

It is advantageous to place the IMUs close to the balancing edge, given that this reduces the radial distance from the edge to the sensor and therefore reducing the centrifugal and tangential accelerations induced at the sensor, which will introduce interference in the measurement of the gravity acceleration direction. If one IMU is placed close to a cube vertex, this will be the IMU of choice when balancing around the 3 edges that form the vertex, considering this, 4 IMUs placed in a tetrahedral configuration in 4 of the 8 vertices is enough to always provide an IMU with minimum distance to any edge.

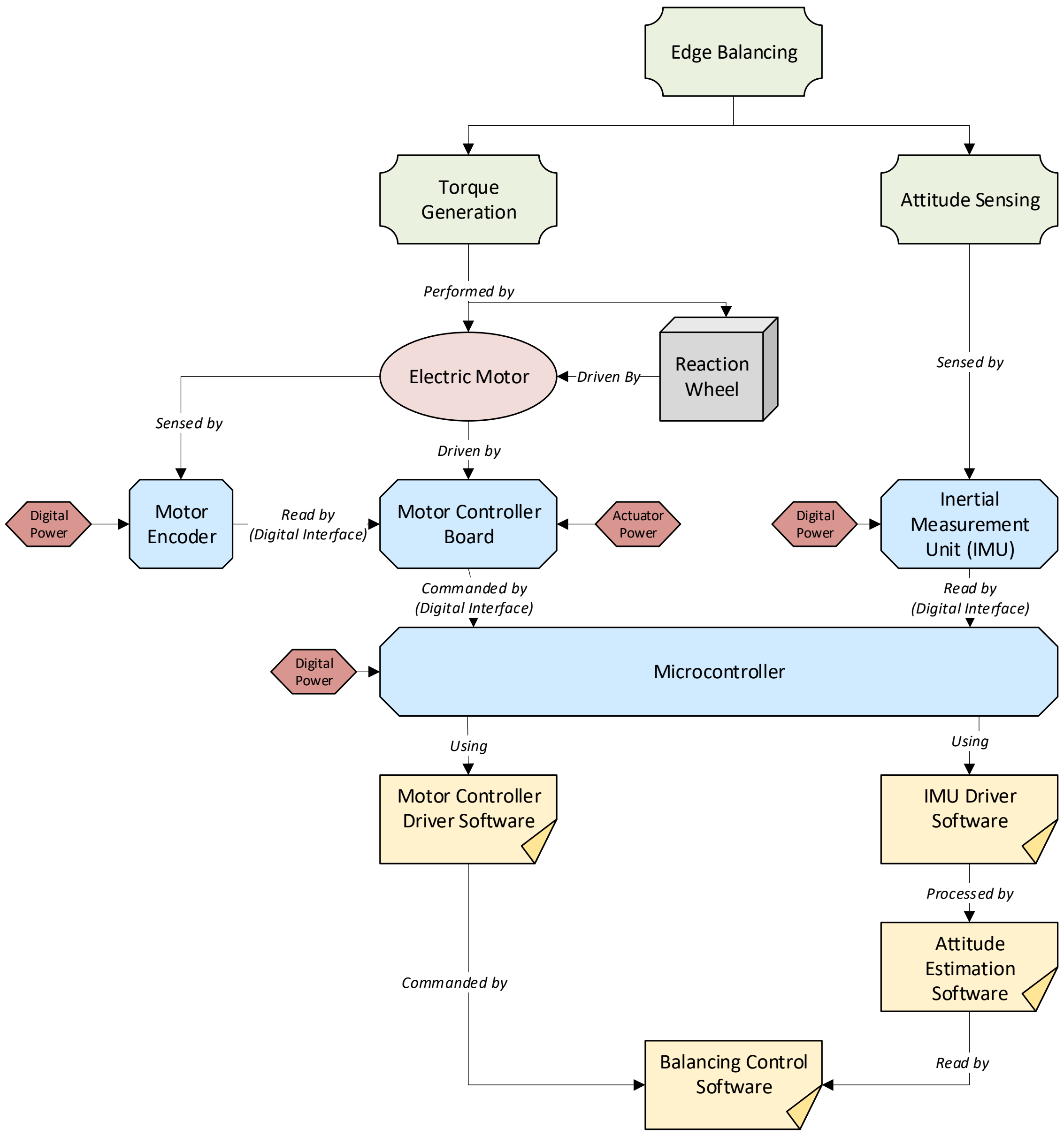


Figure 37 - Balancing Function System Design.

6.2. Self-Powered

Figure 38 shows the requirement flow down for this function. A rechargeable battery is used to self-power the system. The charging system is external to the cube to reduce weight given that the cube will not be operated during charging. To simplify the design, system operation and in order to reduce part number, one battery to power the whole system is used. This requires direct battery connection to the motor drivers and a step down regulator for the low voltage electronics.

Low voltage battery protection is performed by software with two methods:

Shut down motor drivers.

- Turn on a buzzer to signal operator to turn off the device.

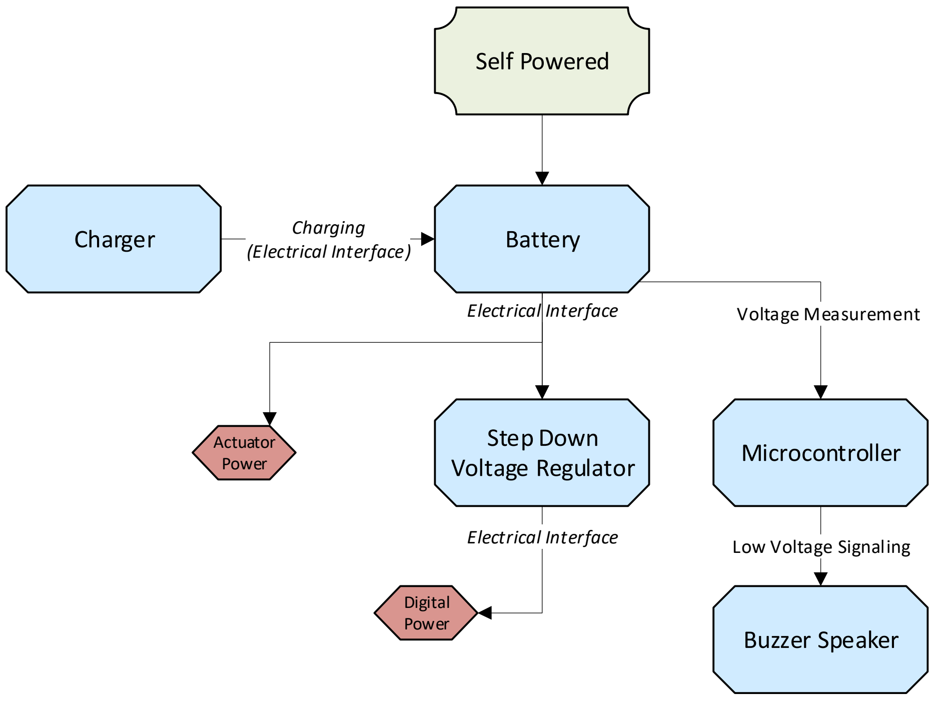


Figure 38 - Self Power Function System Design.

6.3. Simple User Interface

Figure 39 shows the three components of the simple user interface: Power button, Digital Button, Microcontroller Serial Interface.

- Power button connects and disconnects the battery from the device.
- Digital button provides an interface to the device software.
- Microcontroller serial interface is used for software debugging, data output and complex commands.

During operation, the user interface is given by the Power Button and the Digital Button. The next section describes the states of the system and how they relate to this interface.

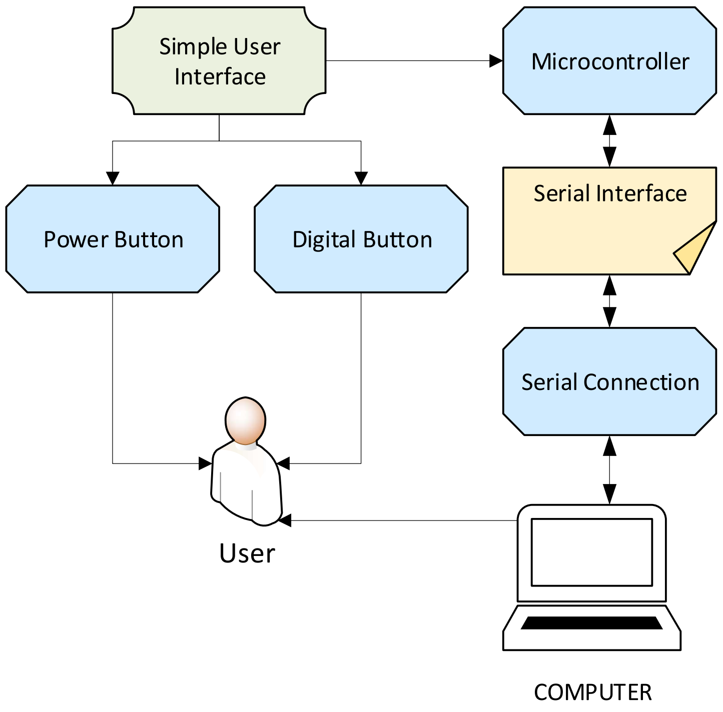


Figure 39 - Simple User Interface.

6.4. System States

As mentioned in the previous section, the user will interface with the system via two buttons:

- Power Button: this is a toggle switch with ON and OFF states.

- Digital Button: this is a push button that generates a ‘pressed’ event when pressed.

Figure 40 shows a finite state machine describing the relation between the inputs and states.

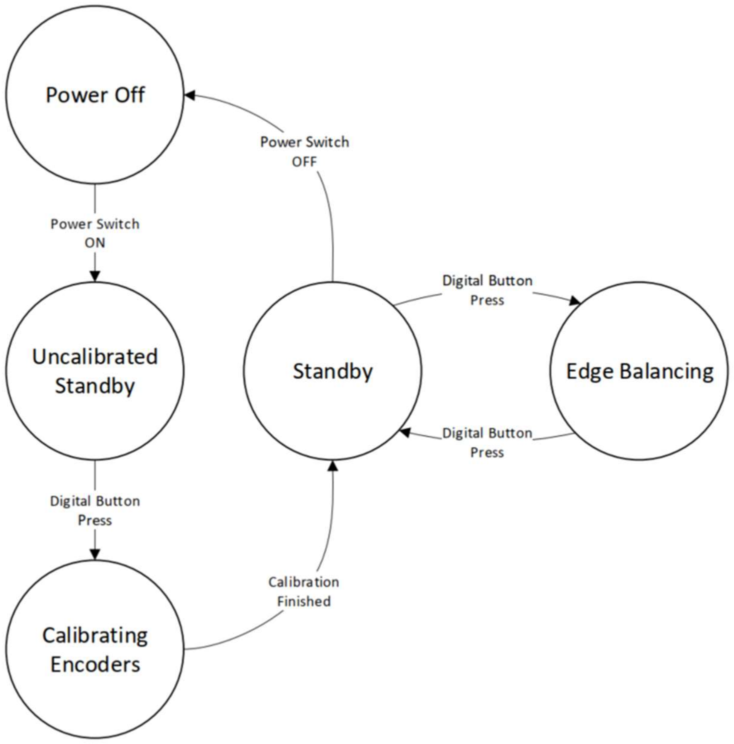


Figure 40 – System Finite State Machine.

The system states are the following:

- Power Off: when Power button is set to off, any previous state will transition to this one.
- Uncalibrated Standby: this is the state entered when the cube is powered on. The name refers to the encoder being uncalibrated.
- Calibrating Encoders: during this state, the wheels are slowly rotated to find the zero position of the encoders. This process lasts for 10 seconds and then transitions to the next state.
- Standby: the cube is ready to get activated for edge balancing

- Edge Balancing: during this state, if the cube is manually placed within 10 degrees of the stable position over any of the 12 edges, the control system will activate, and it will stabilize the cube around that edge.

7. Detailed Design

7.1. Microcontroller Board Selection

To select a microcontroller board, several characteristics need to be taken into account. Considering the stage of the design, these features and characteristics were divided in two: Requirements (Known and/or quantifiable) and Desirable Characteristics (Preliminary and/or hard to quantify).

7.1.1. Requirements

These features or characteristics are already known and/or are numerically quantifiable at this stage. These are:

	Requirement	**Rationale/Comment**
1	The MCU shall fit within geometrical constraints	Smaller than 150mm x 150mm
2	The MCU shall satisfy the Interface & Peripheral Requirements specified in Table 14.	

Table 11 – Microcontroller Board Requirements.

7.1.2. Desirable Characteristics

	Characteristic	**Rationale/Comment**
1	High Flash Memory	Final software size currently unknown.
2	High Clock Speed	Software processing power requirement currently unknown.
3	Has Floating Point Unit (FPU)	Desirable to avoid working with fixed point numbers.
4	Easy to use development software backed with video tutorials	
5	Low Price	Given the budget constraints.

Table 12 – Microcontroller Board Desirable Characteristics.

7.1.3. Interface and Peripheral Requirement Identification

Table 13 shows the interface and peripheral requirement identification based on the system architecture.

Device	Quantity	Interface
Motor Driver	2	UART
IMU	3	SPI
Communications Module	1	SPI
Battery Voltage	1	ADC
Status LED	3	GPIO

Table 13 – Microcontroller Interface & Peripheral Requirements Identification.

Peripheral/Interface	Quantity
UART	2
SPI	4
ADC	1
GPIO	3

Table 14 – Microcontroller Interface & Peripheral Type and Quantity Requirements.

7.1.4. Selection

The final selection is broken down in two: selection of embedded board family and then selection of board within that family.

Table 15 shows the two families considered. The table was completed after browsing board options. Considering this table, the STM32 Nucleo family was selected.

Embedded Board Family	**Development Environment**	**Peripherals**	**FPU**	**Memory**	**Clock**
Arduino	Arduino	Low Count	no	low	slow
STM32 NUCLEO	Arduino Mbed STM32Cube MATLAB/Simulink	High Count	Yes	high	Fast
STM32 Blue Pill	Arduino	High Count	No	medium	medium

Table 15 – Microcontroller Family Options Comparison.

Now within the STM32 family, a list of options was compiled in Table 16.

#	**P/N**	**Pins**	**Core**	**FPU**	**Clock [MHz]**	**Flash [KB]**	**SRAM [KB]**	**UART**	**SPI**	**CAN**	**ADC**	**$**
1	NUCLEO-F446RE	64	M4	Y	180	512	128	6	4	2	3	14
2	NUCLEO-F411RE	64	M4	Y	100	512	128	3	5	0	1	14
3	NUCLEO-L476RG	64	M4	N	80	1024	128	5	3	1	3	14
4	NUCLEO-L432KC	32	M4	N	80	256	64	2	2	-	1	10
5	STM32 Blue Pill	44	M3	N	72	64	20	3	2	1	2	6

Table 16 – STM32 NUCLEO Boards Options Comparison.

From these options, 1 and 2 are desirable with the following rationale:

- Satisfies Peripheral Requirements
- Has FPU
- High Clock speed
- Fits within geometrical constraints and has mounting holes
- High Flash and SRAM Memory
- No considerable price differences.

At purchase time, 1 was unavailable therefore 2 was chosen. Figure 41 shows the board.

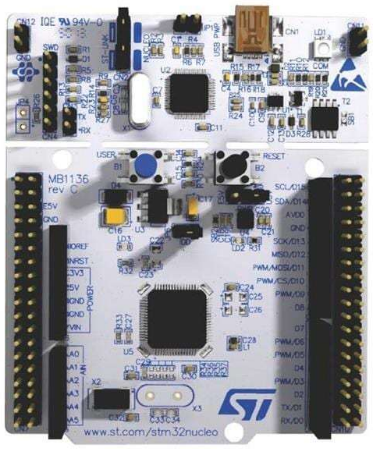


Figure 41 – STM32 NUCLEO-F411RE Board.

7.2. Inertial Measurement Unit (IMU) Selection

The MP6050 was selected due to its low cost and wide availability in the Arduino environment.

It can be read at 40Hz, which is 40 times greater than the estimated plant poles frequency in Section 5.2.4.

Figure 42 – MPU6050 Inertial Measurement Unit.

7.3. Power and Charging System Design

To design this system, the following requirements were considered:

- The system shall have only one battery.
- The battery shall be charged by an external charger.
- The external charger shall connect to the cube through chassis connectors.
- A toggle switch shall control charging or internal power state.

As presented in Section 5.1.3: the cube is powered by an Ovonic 14.8V 1300mAh 100C 4S LiPo Battery. The battery is charged by a Enegitech e430 Charger 4S LiPo.

The charging interface consists of a T-Plug connector for charging and a JST-XH 4-pin connector for cell balancing.

The cube electrical system requires three voltage levels:

- Vbat (16.8 V to 12 V): To power the ODrive motor driver boards.
- 5V: To power the microcontroller's Low-dropout Regulator at the Nucleo board.
- 3.3V: To power IMU, Display, Bluetooth Module. Note: Encoders are powered by the motor driver board's own 3.3V source.

To simplify the design, all digital voltages are derived from the single LiPo battery in the system. Figure 43 shows the power and charging system diagram and how these voltages are generated. The power switch connects the positive lead of the battery either to the internal circuit or to the charging interface.

Figure 44 shows a picture of the charging interface showing the charging connectors, balancing connectors, and power switch.

Figure 45 shows a picture of the cube being charged using mains power.

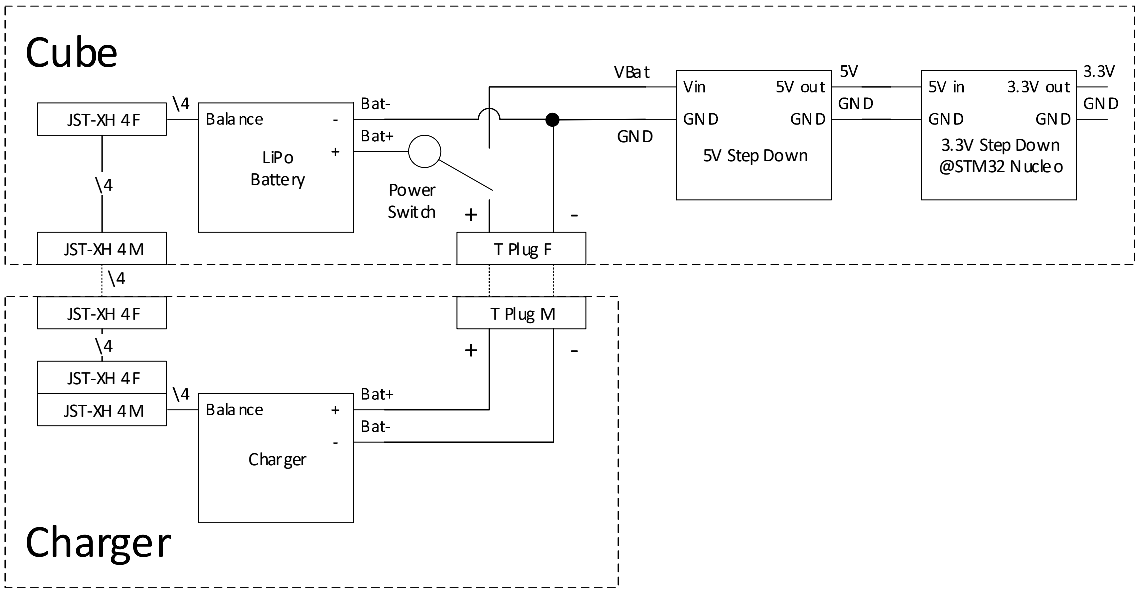


Figure 43 – Power and Charging System Electrical Schematic.

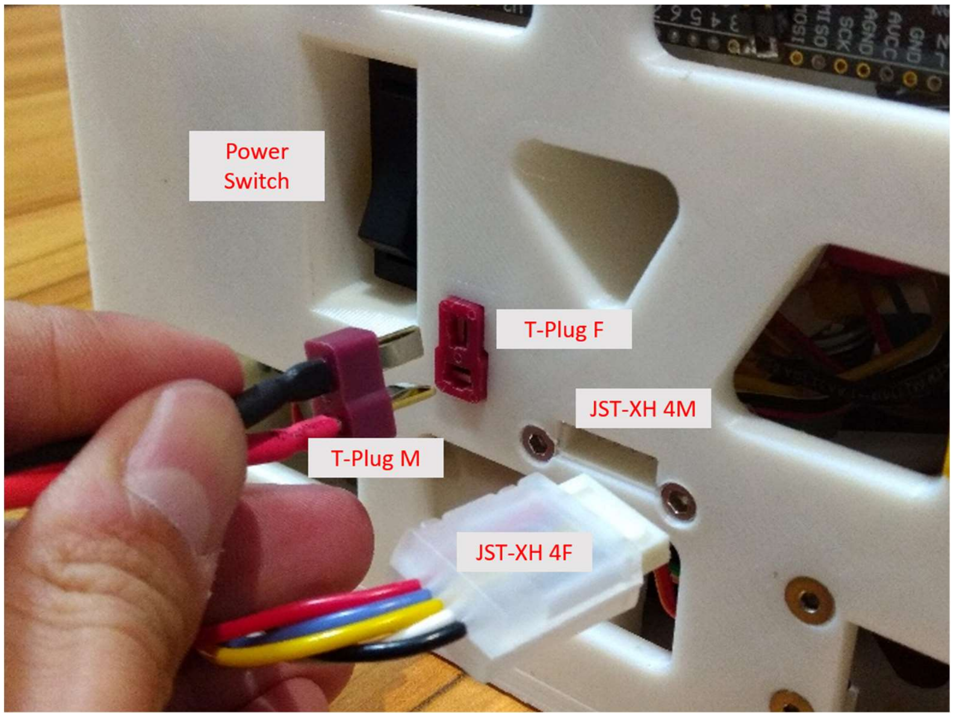


Figure 44 – Picture of charging interface.

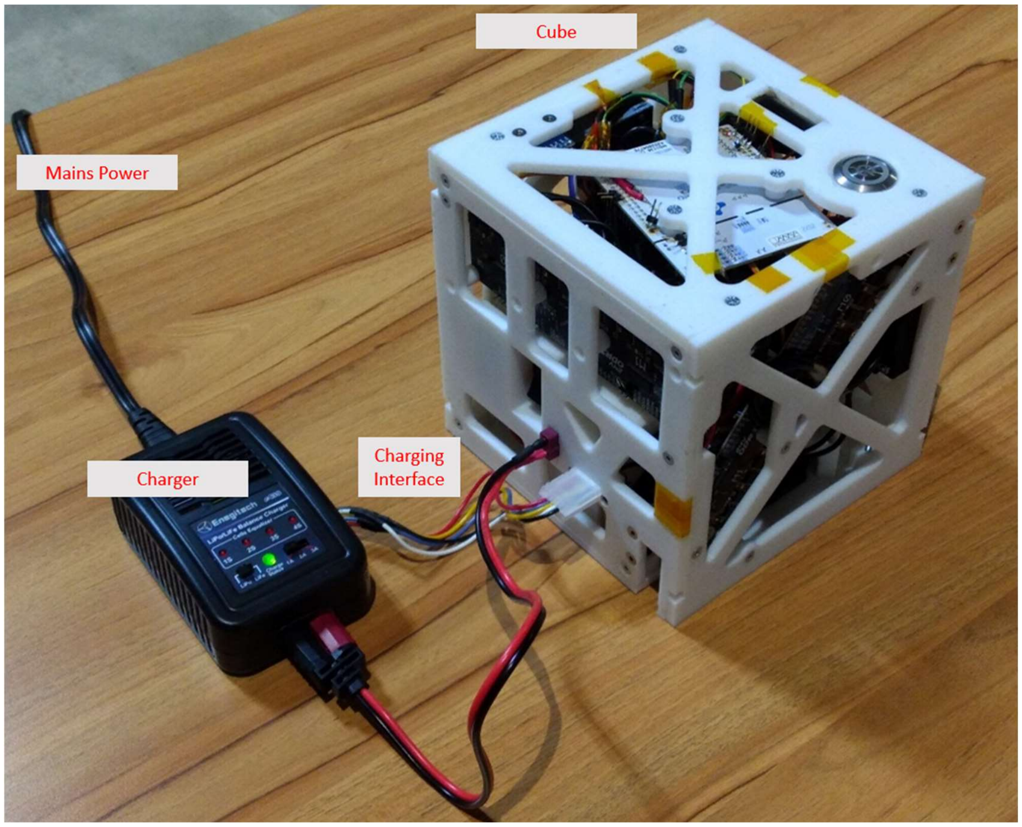


Figure 45 – Picture of cube being charged.

The current load on the STM32 Nucleo LDO is evaluated. Table 17 shows a budget of the current required by all the connected modules, which results in an estimated current of 190 mA. The low-dropout regulator included in this board is a LD39050PU33R with 3.3V and 500mA output and therefore can handle the load.

Part	Description	Voltage [V]	Current [mA]	Q	Total Current [mA]
MPU6050	IMU	3.3	3.9	4	15.6
HM-10	Bluetooth	3.3	50	1	50
SSD1306	Display	3.3	15	1	15
NUCLEO-446RE	Microcontroller	5	110	1	110
			Total USD		190.6

Table 17 – LDO Current Load Budget.

7.4. Electrical Design

This section presents the electrical schematics of all components of the system except the battery system which was presented in the previous section.

Figure 46 shows the reaction wheel system, which consists of the STM32 NUCLEO board, two ODrive boards and three BLDC motors with their corresponding encoders. The STM32 interfaces with the ODrive boards through UART. Each ODrive board interfaces with each motor though the U V W three-phase driving signals, and via an ABI[1] interface with the encoders. The encoders are powered from the ODrive board given that it provides this possibility. The encoders interface with the STM32 microcontroller through SPI, although this latter interface was not used, and the encoder speed is measured from the ODrive.

Figure 47 shows the IMU's connections. The MPU6050 provides two selectable I2C addresses by setting the AD0 to high or low, therefore two I2C peripherals were used to interface with the four IMUs by alternating the AD0 pin.

[1] ABI is also known as the incremental encoder interface. It consists of three wires, A B and I. As the encoder spins, the A and B are square waves with 90° phase difference that cycle a specific number of times per revolution. The I signal is a pulse for each whole revolution.

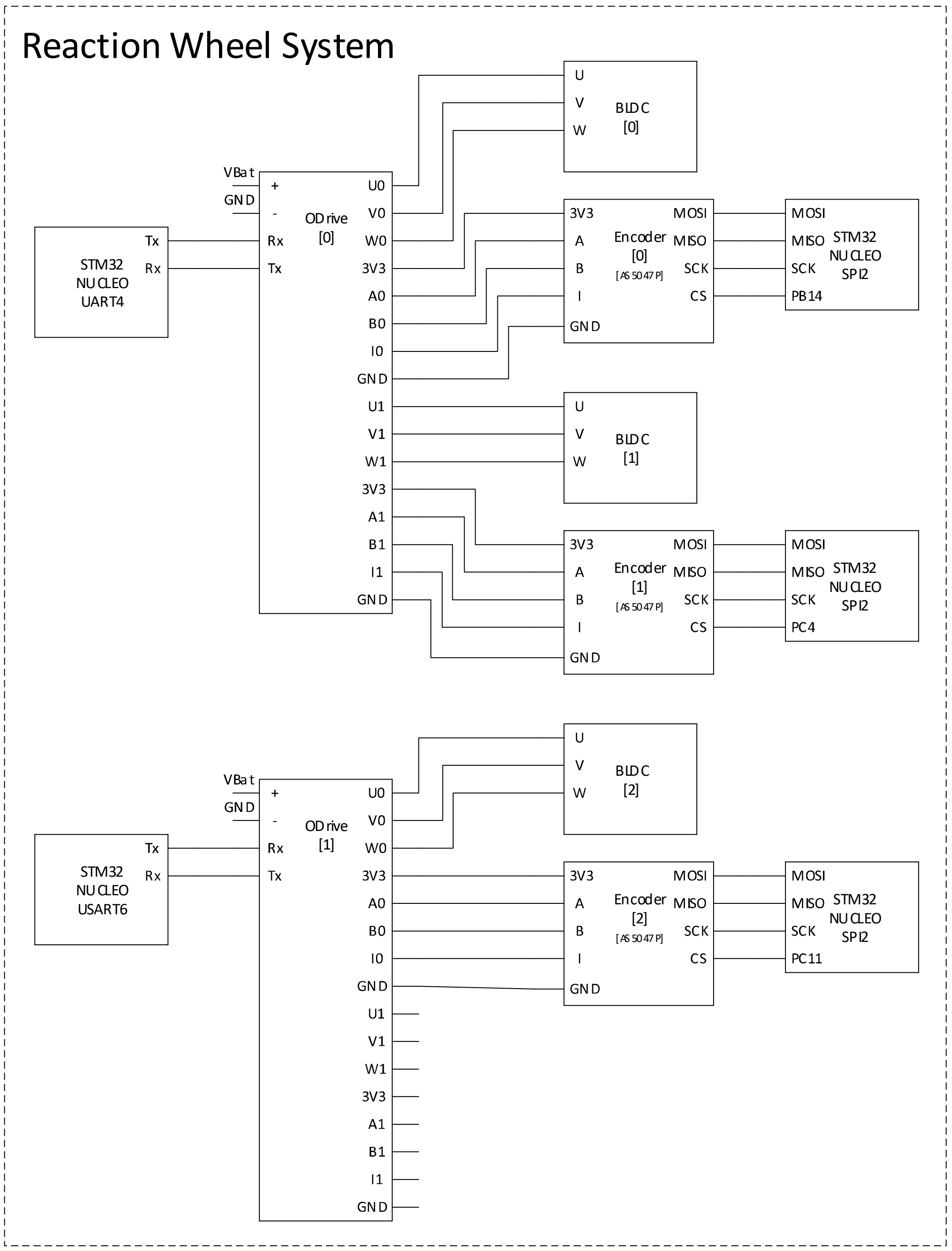


Figure 46 – Reaction Wheel System Electrical Schematic.

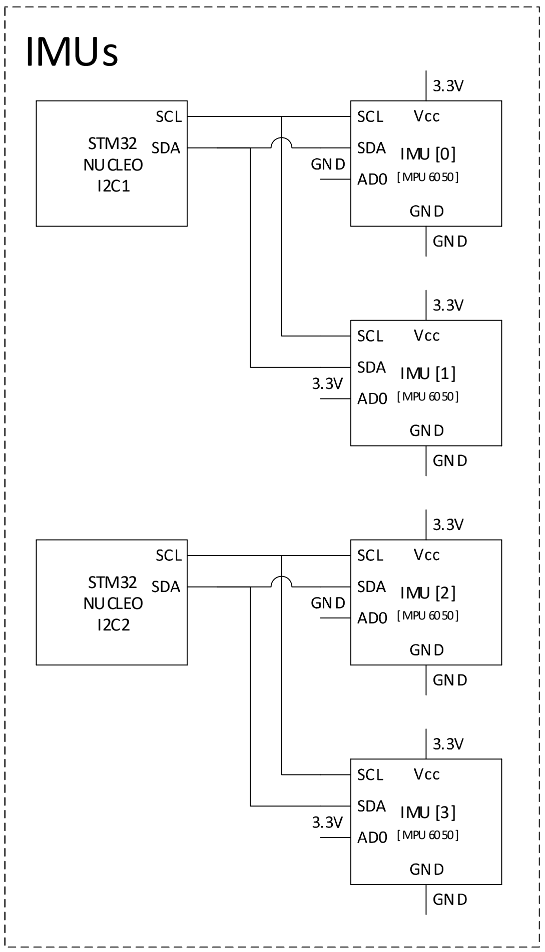


Figure 47 – Inertial Measurement Unit Array Electrical Schematic.

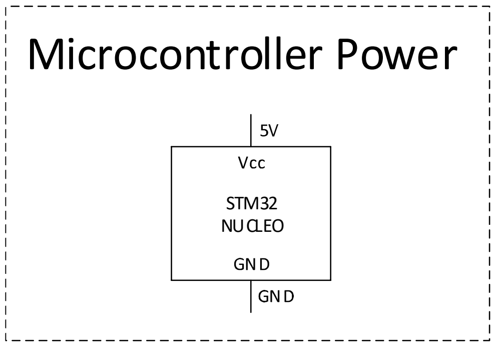


Figure 48 – Microcontroller Power Electrical Schematic.

Figure 48 shows the STM32 Nucleo board being powered by the 5V step down from the power supply system. Figure 50 shows the interface to the 612-SV7F23SS-6G1 pushbutton with LEDs. The pushbutton is connected to the PC8 interrupt enabled PIN from the Nucleo board. The LEDs are connected to GPIO outputs PA6 and PA7. The microcontroller is rated to output or sink up to ±20mA, therefore a 220 Ω resistor was connected in series to limit the current.

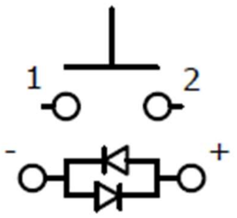


Figure 49 - 612-SV7F23SS-6G1 Schematic.

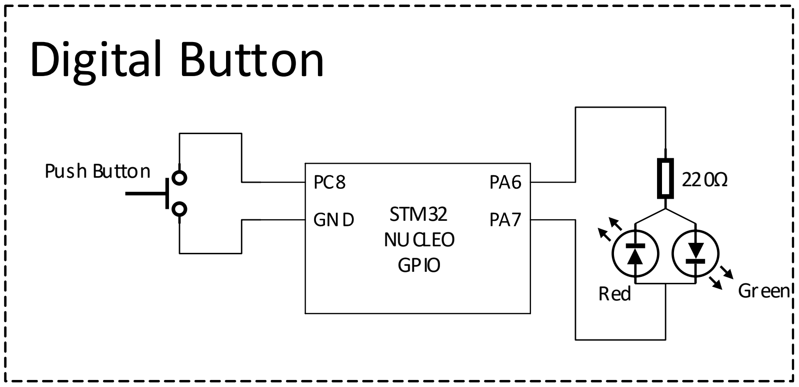


Figure 50 – Digital Button Electrical Schematic.

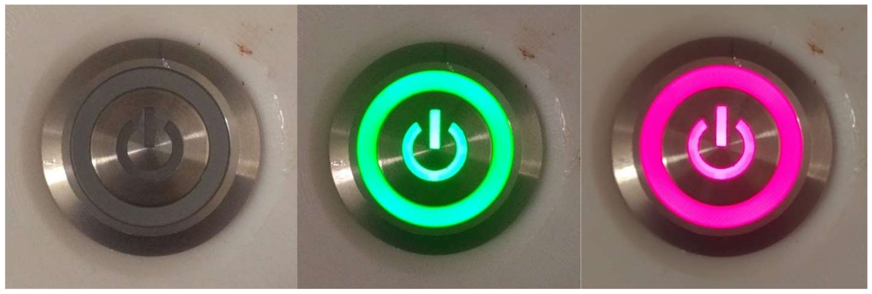


Figure 51 – Digital Button LED States. From left to right: OFF, Green, Red.

Figure 52 shows the Bluetooth module connected through UART to the STM32 microcontroller. This module is included for future projects in case remote control is required.

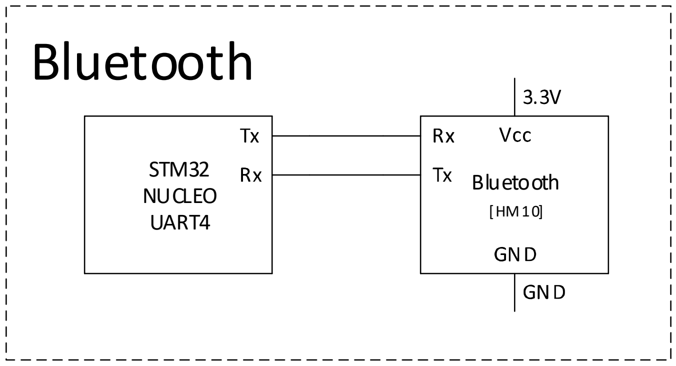


Figure 52 – Bluetooth Electrical Schematic.

7.4.1. ODrive Electrical Integration

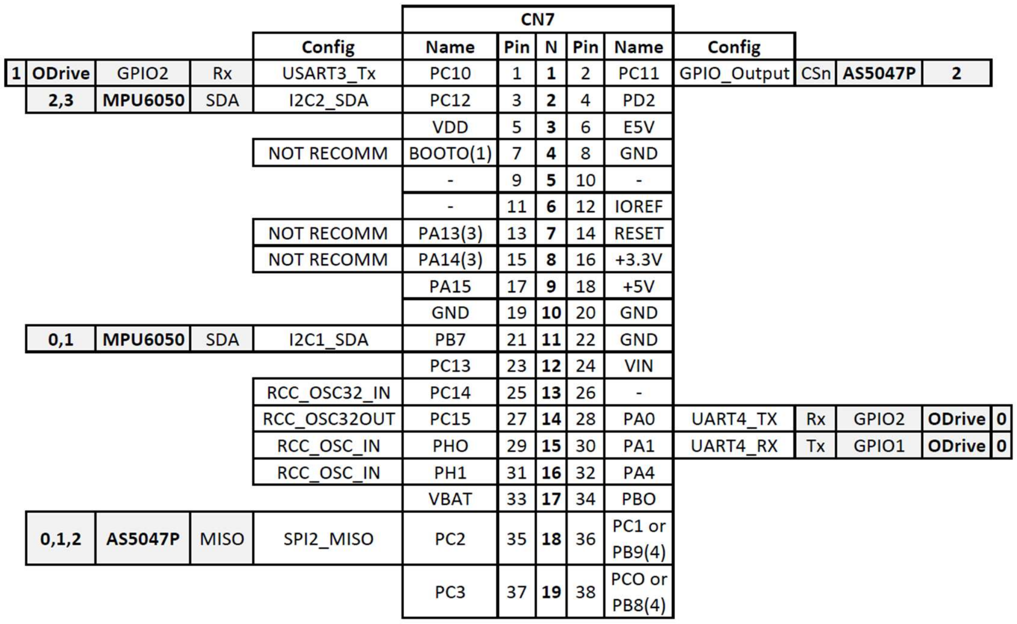
All information used to interface the ODrive boards to the system can be found here or at 07 - Software\3 - Components Datasheets\ODrive.

7.5. Microcontroller Pin Assignment

This section shows all the interfaces to the STM32 Nucleo board. Figure 53 and Figure 54 show the microcontroller interfaces at the CN7 and CN10 headers respectively. The port name is shown in the "Name" column and the mode it will operate in, in in the "Config" column. The connected peripheral is represented in grey, showing the pin function, peripheral port name, peripheral name and peripheral identification number.

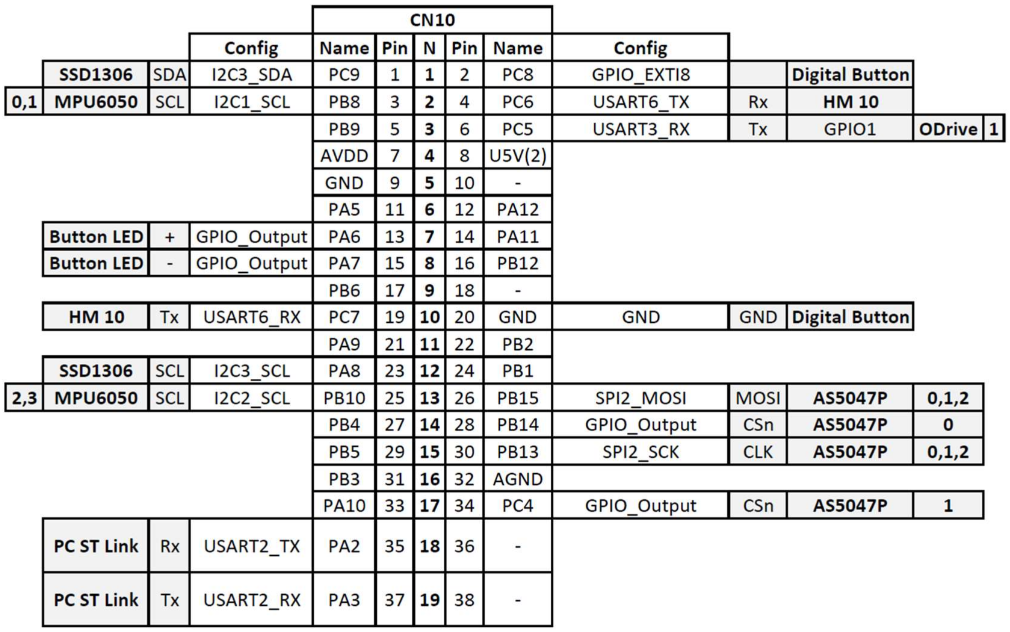
Figure 55 shows a summary of all the microcontroller peripherals with their corresponding modules.

Figure 56 shows a screenshot of the configuration tool used to manage the pins in the IDE (integrated development environment) software. This tool is called STM32cubeMX and is included in the STM32Cube IDE. It allows pins to be configured graphically and then auto-generates the initialization code for all peripherals configured.



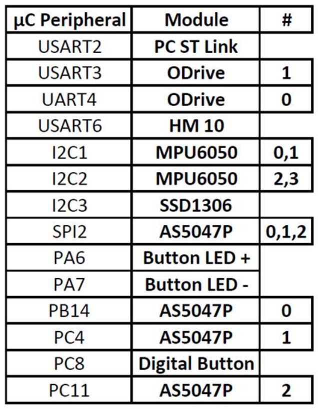
					CN7									
				Config	Name	Pin	N	Pin	Name	Config				
1	ODrive	GPIO2	Rx	USART3_Tx	PC10	1	**1**	2	PC11	GPIO_Output	CSn	**AS5047P**	2	
	2,3	**MPU6050**	SDA	I2C2_SDA	PC12	3	**2**	4	PD2					
					VDD	5	**3**	6	E5V					
				NOT RECOMM	BOOTO(1)	7	**4**	8	GND					
					-	9	**5**	10	-					
					-	11	**6**	12	IOREF					
				NOT RECOMM	PA13(3)	13	**7**	14	RESET					
				NOT RECOMM	PA14(3)	15	**8**	16	+3.3V					
					PA15	17	**9**	18	+5V					
					GND	19	**10**	20	GND					
	0,1	**MPU6050**	SDA	I2C1_SDA	PB7	21	**11**	22	GND					
					PC13	23	**12**	24	VIN					
				RCC_OSC32_IN	PC14	25	**13**	26	-					
				RCC_OSC32OUT	PC15	27	**14**	28	PA0	UART4_TX	Rx	GPIO2	**ODrive**	**0**
				RCC_OSC_IN	PHO	29	**15**	30	PA1	UART4_RX	Tx	GPIO1	**ODrive**	**0**
				RCC_OSC_IN	PH1	31	**16**	32	PA4					
					VBAT	33	**17**	34	PBO					
	0,1,2	**AS5047P**	MISO	SPI2_MISO	PC2	35	**18**	36	PC1 or PB9(4)					
					PC3	37	**19**	38	PCO or PB8(4)					

Figure 53 – Microcontroller Pin Assignment. CN7 Port.



				CN10									
			Config	Name	Pin	N	Pin	Name	Config				
	SSD1306	SDA	I2C3_SDA	PC9	1	**1**	2	PC8	GPIO_EXTI8		**Digital Button**		
0,1	**MPU6050**	SCL	I2C1_SCL	PB8	3	**2**	4	PC6	USART6_TX	Rx	**HM 10**		
				PB9	5	**3**	6	PC5	USART3_RX	Tx	GPIO1	**ODrive**	**1**
				AVDD	7	**4**	8	U5V(2)					
				GND	9	**5**	10	-					
				PA5	11	**6**	12	PA12					
	Button LED	+	GPIO_Output	PA6	13	**7**	14	PA11					
	Button LED	-	GPIO_Output	PA7	15	**8**	16	PB12					
				PB6	17	**9**	18	-					
	HM 10	Tx	USART6_RX	PC7	19	**10**	20	GND	GND	GND	**Digital Button**		
				PA9	21	**11**	22	PB2					
	SSD1306	SCL	I2C3_SCL	PA8	23	**12**	24	PB1					
2,3	**MPU6050**	SCL	I2C2_SCL	PB10	25	**13**	26	PB15	SPI2_MOSI	MOSI	**AS5047P**	**0,1,2**	
				PB4	27	**14**	28	PB14	GPIO_Output	CSn	**AS5047P**	**0**	
				PB5	29	**15**	30	PB13	SPI2_SCK	CLK	**AS5047P**	**0,1,2**	
				PB3	31	**16**	32	AGND					
				PA10	33	**17**	34	PC4	GPIO_Output	CSn	**AS5047P**	**1**	
	PC ST Link	Rx	USART2_TX	PA2	35	**18**	36	-					
	PC ST Link	Tx	USART2_RX	PA3	37	**19**	38	-					

Figure 54 – Microcontroller Pin Assignment. CN10 Port.



µC Peripheral	Module	#
USART2	**PC ST Link**	
USART3	**ODrive**	**1**
UART4	**ODrive**	**0**
USART6	**HM 10**	
I2C1	**MPU6050**	**0,1**
I2C2	**MPU6050**	**2,3**
I2C3	**SSD1306**	
SPI2	**AS5047P**	**0,1,2**
PA6	**Button LED +**	
PA7	**Button LED -**	
PB14	**AS5047P**	**0**
PC4	**AS5047P**	**1**
PC8	**Digital Button**	
PC11	**AS5047P**	**2**

Figure 55 – STM32 Microcontroller Interface Summary.

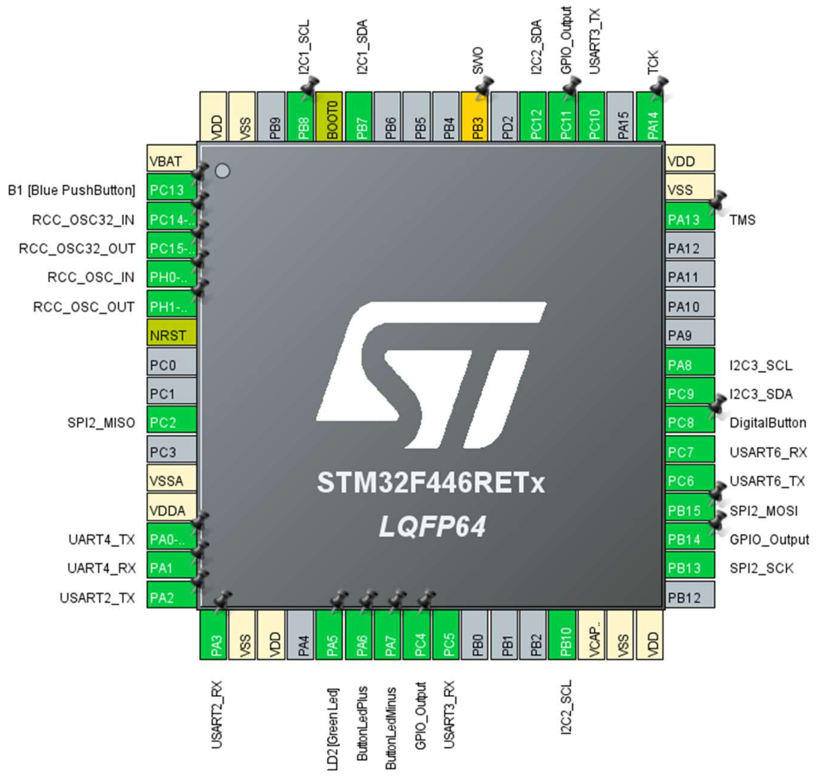


Figure 56 - STM32cubeMX configuration tool. Microcontroller Pinout Assignment.

7.6. Mechanical Design

This section describes the mechanical design of the system. Figure 57 the cube can be seen from all vertices.

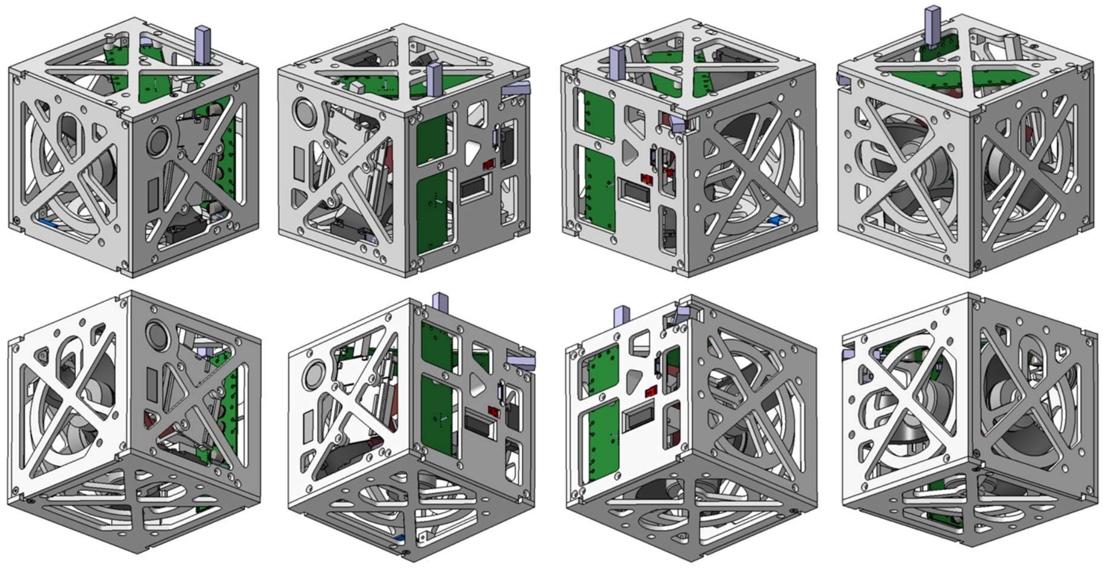


Figure 57 – Cube Mechanical Design. Views from all vertices.

7.6.1. Vertex Design

Figure 58 shows the 8 vertices of the cube, the pictures are arranged in the same manner as in Figure 57. The panels were design such that the exterior envelope of the system assembly is a cube.

Next section describes how panels outer contour was designed to generate these vertices.

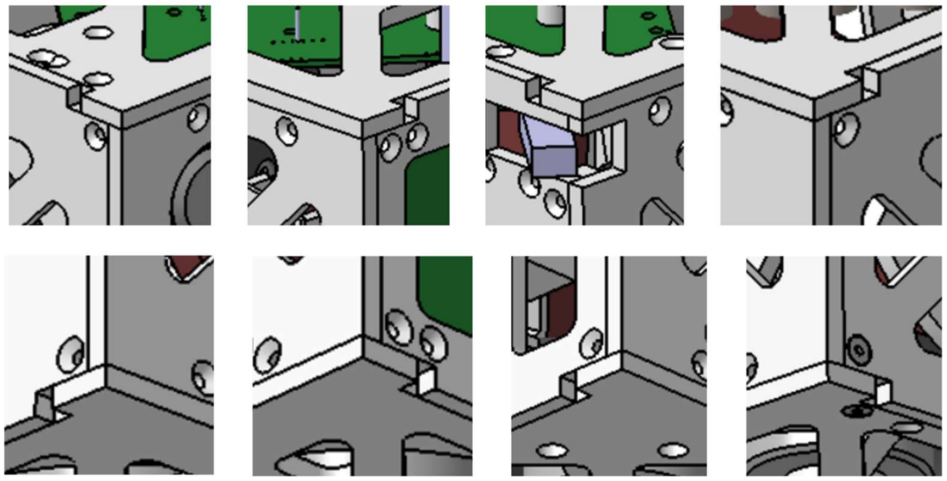


Figure 58 – 8 cube vertices.

7.6.2. Cube Panel Assembly Design

Figure 59 shows the 6 cube panels are named: 1A, 2A, 3A, 1B, 2B, 3B. The numbers correspond to the panel outer shape type and the letter indicates the instance. These panels were designed to be 3D printed. All 8 vertices of the cube are generated by the 1 panel type. Give that each 1 panel has 4 vertex, panels 1A and 1B sum the 8 vertices of the cube. Figure 60 shows the screw cubes used to screw the panels together. Given that the panels are 3D printed, it is easy to generate the cubes as part of the panel.

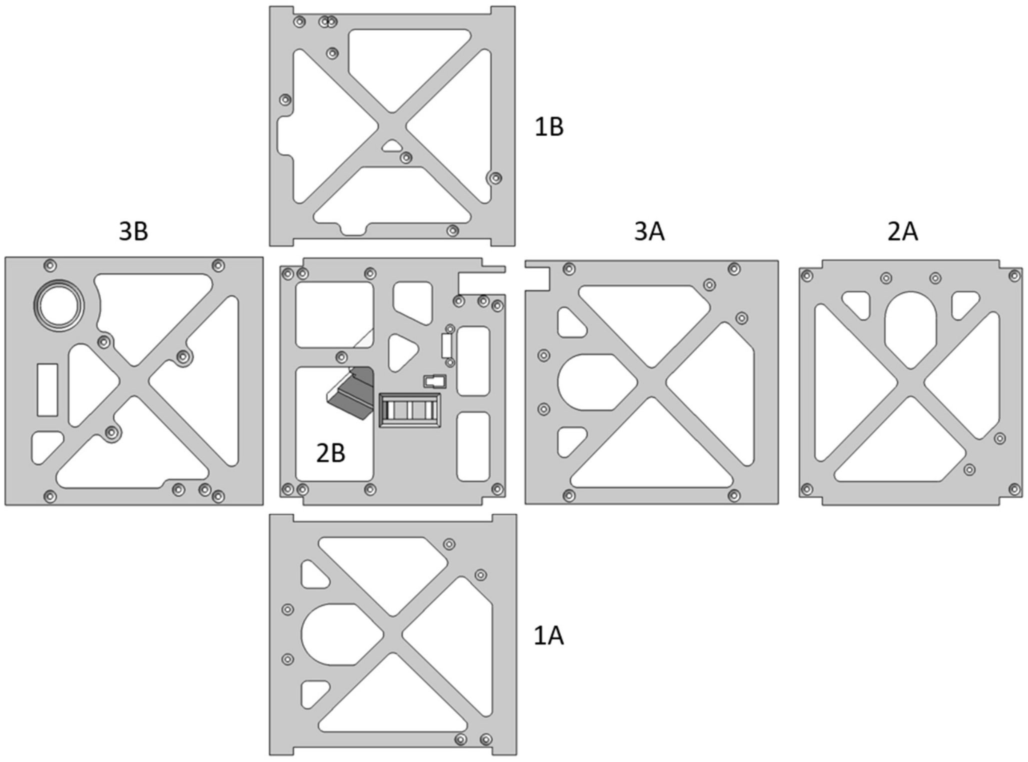


Figure 59 – Cube Panels. Outer face view.

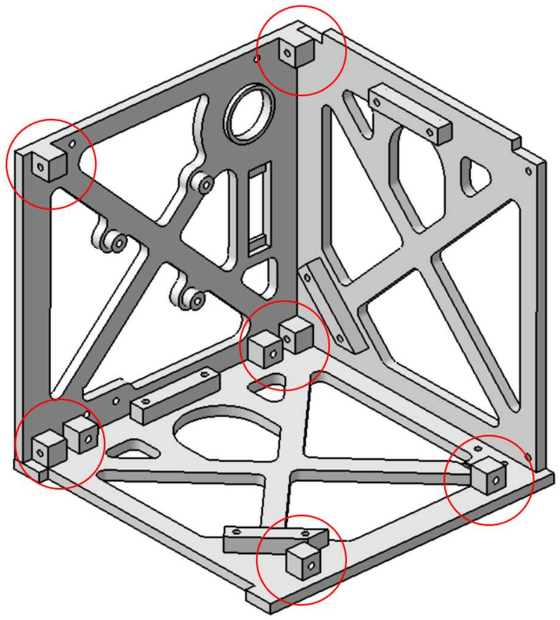


Figure 60 – Cube panel assembly screw cubes.

7.6.3. Panel Assemblies

This section presents a detailed description of each panel assembly. These assemblies are built before assembling all of them together to form the cube.

7.6.3.1. 1A

This panel holds:

- Motor 0
- Encoder 0
- IMU 3
- Arm 0

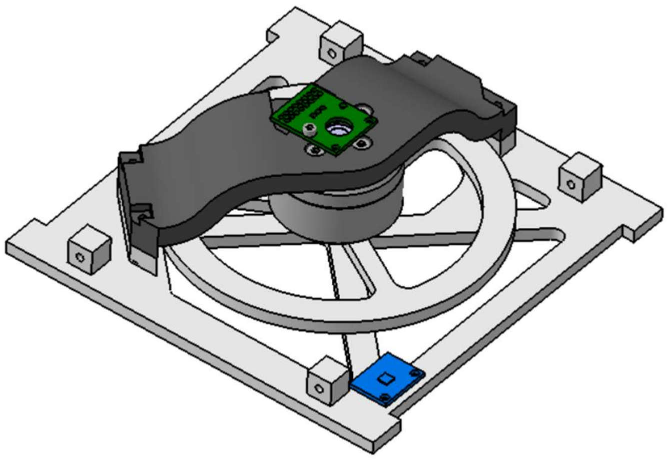


Figure 61 – Panel 1A.

7.6.3.2. 2A

This panel holds:

- Motor 1
- Encoder 1
- Arm 1

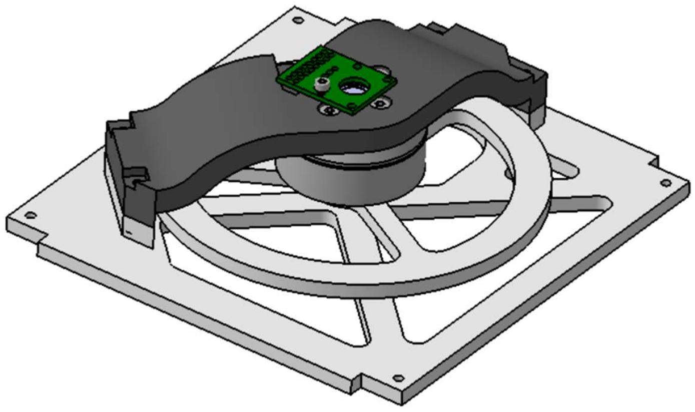


Figure 62 – Panel 2A.

7.6.3.3. 3A

This panel holds:

- Motor 2
- Encoder 2
- Arm 2

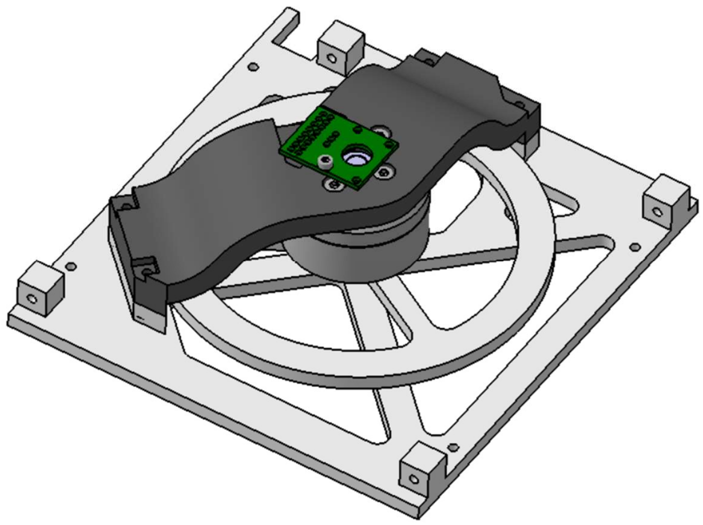


Figure 63 – Panel 3A.

7.6.3.4. 1B

This panel holds:

- ODrive 1
- IMU 2

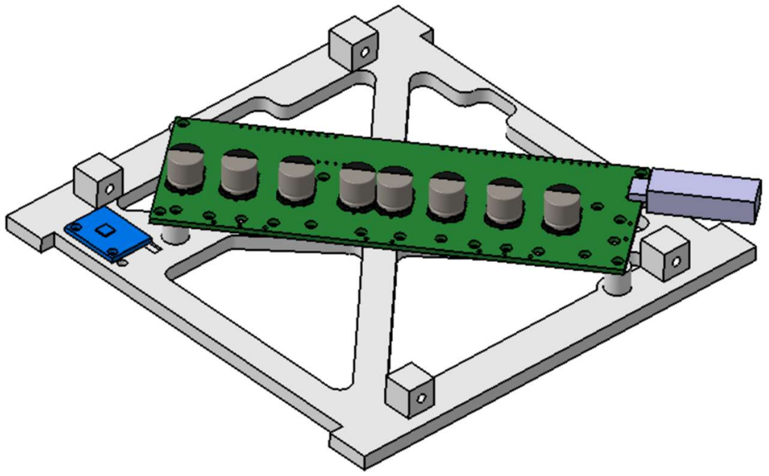


Figure 64 – Panel 1B.

7.6.3.5. 2B

This panel holds:

- Battery
- IMU 1
- Bluetooth Module
- Battery Charger T-Plug Connector
- Battery Balancer JST-XH male connector
- Power Switch

3D printing this part allows for the complex geometry. A key part of this panel is the battery mount. The battery positioning was optimized to move the center of mass of the battery as close as possible to the center of mass of the cube without interfering with the rest of the parts inside the cube. Another key part with complex geometry is the power button slot, as shown in Figure 66.

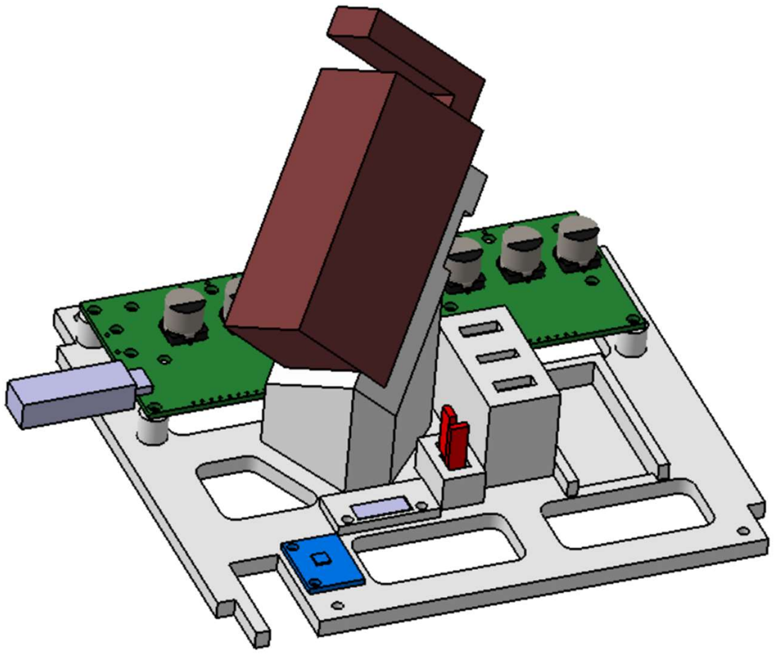


Figure 65 – Panel 2B.

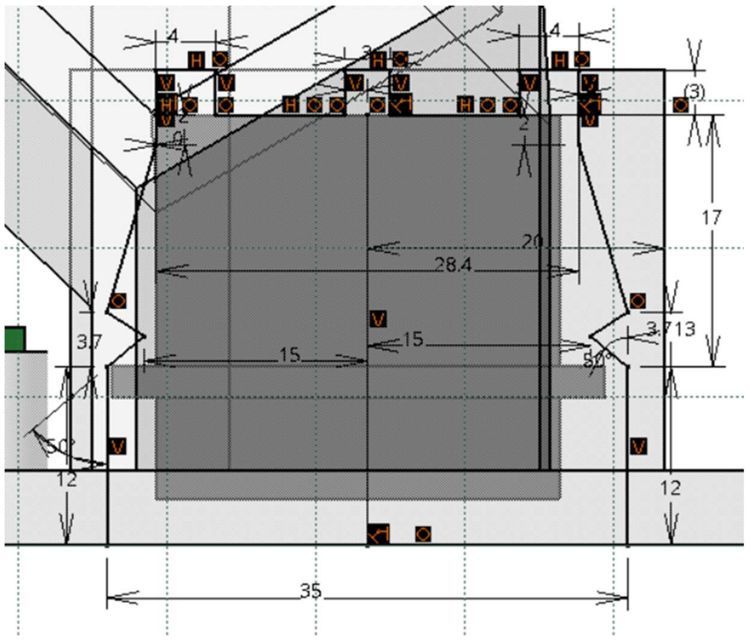


Figure 66 – Power Switch Slot Design.

7.6.3.6. 3B

This panel holds:

- STM32 NUCLEO-F411RE
- IMU 0
- Digital Button
- OLED Display

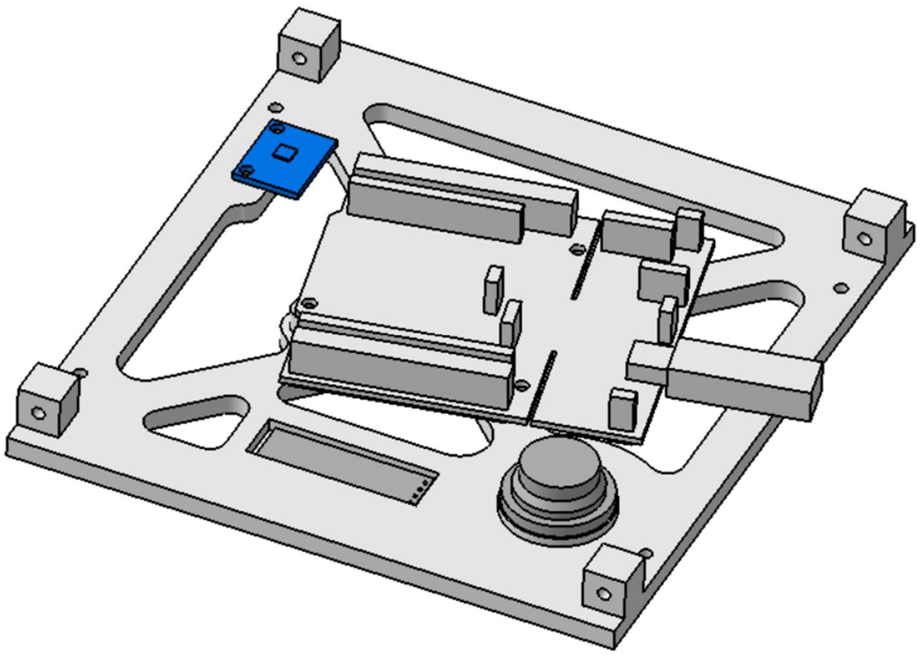


Figure 67 – Panel 3B.

7.6.4. External Interfaces

Figure 68 shows the Micro USB interface to the two ODrive boards.

Figure 69 shows the Mini USB interface to the STM32 Nucleo-F411RE

Figure 70 shows the battery charging interface.

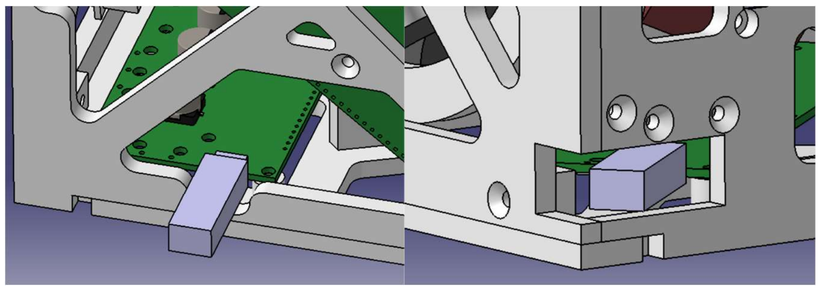


Figure 68 – (Left) ODrive 1 interface. (Right) ODrive 2 interface.

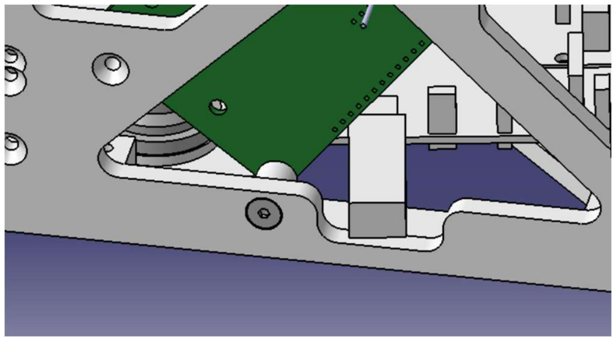


Figure 69 – STM32 NUCLEO-F411RE interface.

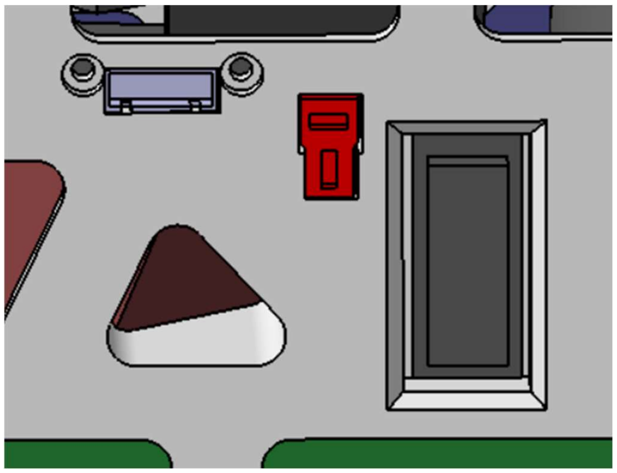


Figure 70 – from left to right: battery balancer connector, battery charging connector, power switch.

8. Implementation

8.1. Manufacturing

All parts were manufactured by 3D printing Polylactic acid (PLA), including the reaction wheels. 3D printers work by executing a G-Code file; the software used to transform .stl files to G-Code is called the slicing software. Ultimaker Cura 4.6 was used to slice the 3D models. For support material, it was found that the *tree* method gave the best results.

Figure 71 shows the result of 3D printing the Motor Arm, including the tree support material.

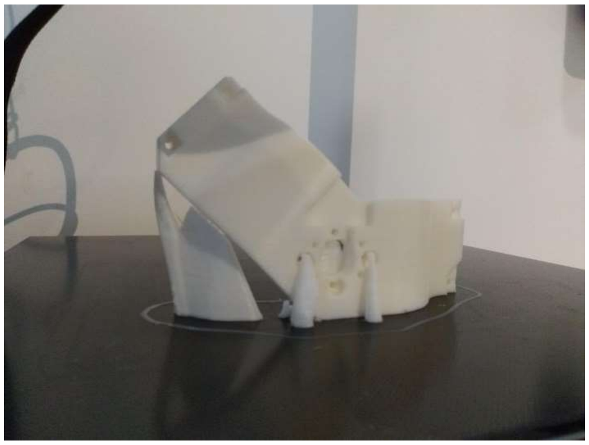


Figure 71 – 3D printing Motor Arm.

No balancing of the reaction wheels was required given that the 3D print quality was good enough, this was verified by mounting the reaction wheels on the motor and not detecting considerable vibration when spinning at maximum velocity.

8.2. Electrical

Figure 72 shows a picture of all components of the system wired as described in Sections 7.3, 7.4 and 7.4.1. The following were the assembly criteria for the wiring, mostly oriented to save development time:

- Almost cables were soldered in place, instead of using connectors. This will make disassembly more time consuming, but it makes assembly faster.
- Cable’s length was chosen to be the minimum length to connect the components in the flat configuration shown in Figure 72.
- Cables were left mostly loose inside the cube and held with tape when needed during assembly, instead of running the cables through the structure in a planned layout. This could produce interference between signals and it could be a possible reason for the I2C bus problem encountered, described later.

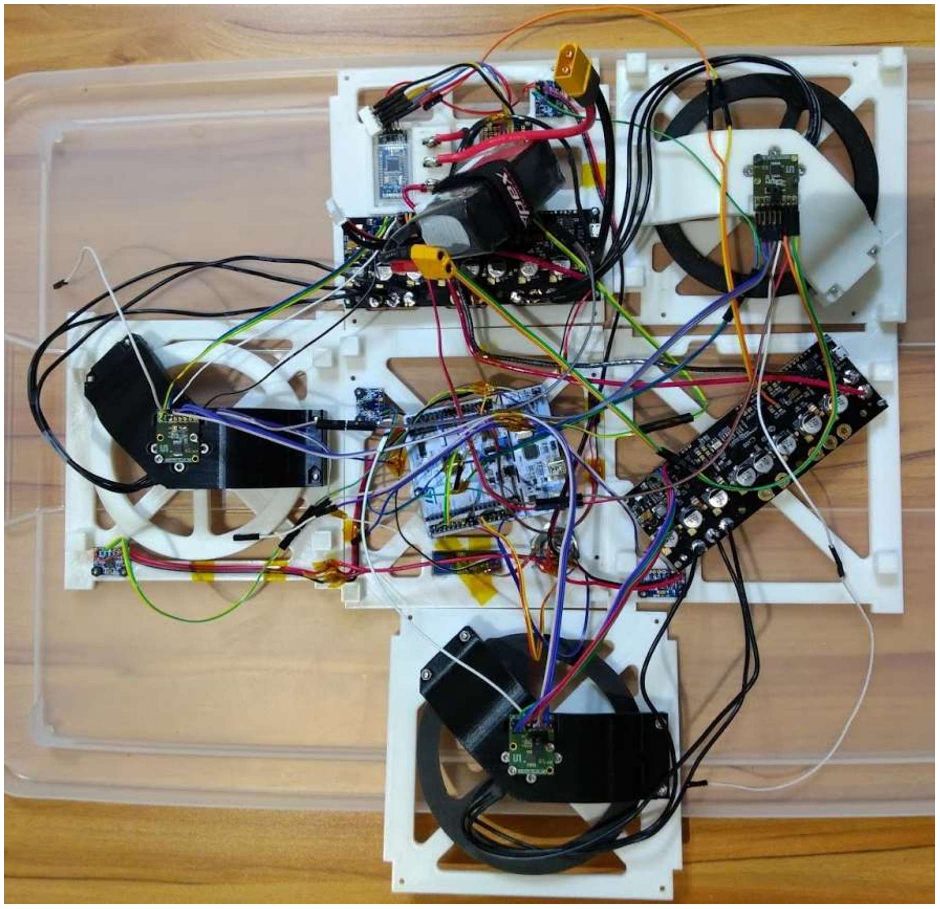


Figure 72 – All Panels Assembled. All components Connected.

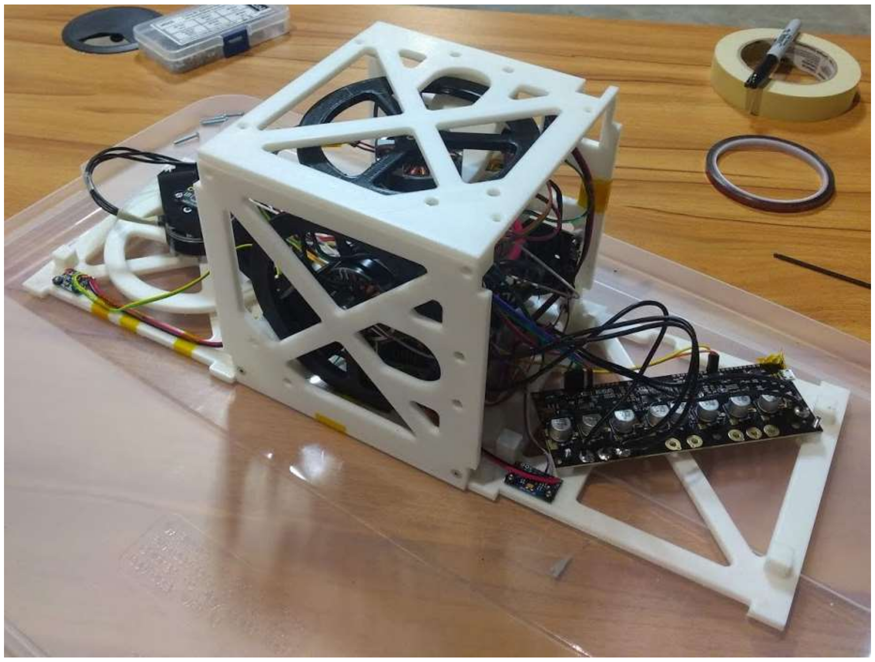


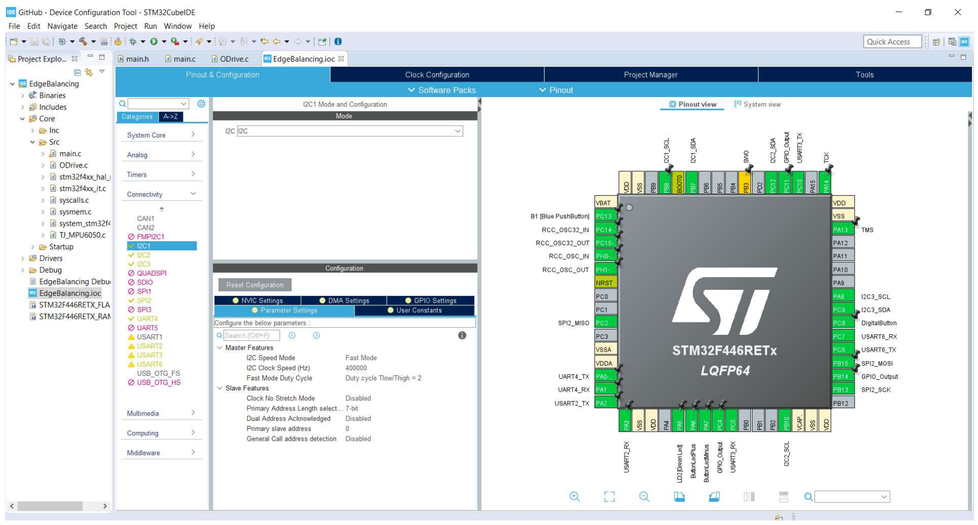
Figure 73 – Cube assembling process, two pannels left.

8.3. Software

8.3.1. Development Environment

For the STM32 Nucleo-F411RE embedded software, the STM32CubeIDE environment was used. This environment provides easy project creation and live debugging functionality. It also includes the STM32CubeMX, which is a graphical pin configuration tool that automatically generates all code needed to configure the peripherals as needed. The code generated is written in a Hardware Abstraction Layer (HAL) which allows for more intuitive interaction with peripherals, avoiding the need to read the microcontroller datasheet and understand all registers.

Figure 74 shows a screenshot of the development environment in the pin configuration tab. As an example, it shows the configuration interface of the I2C1 peripheral.



I2C1 Mode and Configuration

Mode

I2C I2C

Configuration

Configure the below parameters :

```
Master Features
    I2C Speed Mode                     Fast Mode
    I2C Clock Speed (Hz)               400000
    Fast Mode Duty Cycle               Duty cycle Tlow/Thigh = 2
Slave Features
    Clock No Stretch Mode              Disabled
    Primary Address Length select...   7-bit
    Dual Address Acknowledged          Disabled
    Primary slave address              0
    General Call address detection     Disabled
```

Figure 74 - STM32CubeIDE screenshot showing the STM32CubeMX pin configuration tool.

To successfully use these tools, the following learning resources were used:

- Video tutorials from ST
    - MOOC - STM32CubeIDE basics
    - MOOC - STM32CubeMX and STM32Cube HAL basics
- Video tutorials from Digikey
    - Getting Started with STM32 and Nucleo
- User Manuals from ST:
    - UM1725: Description of STM32F4 HAL and low-layer drivers.
    - UM1724: STM32 Nucleo-64 boards (MB1136)
    - STM32F446RE Datasheet

8.3.2. Software Description

8.3.2.1. Main Function

This function initializes all peripheral and variables used thought the program. After this, it enters a while loop that reacts to three flags: digital button pressed, digital button released and new UART command from PC.

When the digital button is pressed for the first time, it triggers ODrive calibration. If pressed again during calibration, nothing is done. If pressed after calibration, edge balancing control loop starts.

When new UART command flag is risen, the received character is retransmitted and switched through several possible values which will trigger an action. The next subsection presents the command list and the functions executed for each command. The PC UART peripheral is configured at a baud rate of 921600.

8.3.2.2. Command List

Sending the ‘h’ command will show this command list.

v : ODrive_GetVoltage

- Returns current battery voltage.

c : Configure and Calibrate all Axes

- Configures and calibrates both axes. The cube beeps and spins wheels in both directions.

l : EdgeBalancingControlLoop()

- Starts edge balancing control loop. Requires ‘c’ command to be executed before. Cube will balance over any edge if manually placed near any edge.

L : EdgeBalancingControlLoopTest()

- Same as ‘l’ but used to test variations in the control loop code.

1 : Select ODrive_Ax[0], ODrive 0 M0

- Select this axis to respond to the ODrive control commands.

2 : Select ODrive_Ax[1], ODrive 0 M1

- Select this axis to respond to the ODrive control commands.

3 : Select ODrive_Ax[2], ODrive 1 M0

- Select this axis to respond to the ODrive control commands.

r : Start Torque Control on Selected ODrive Axis

- Starts torque control with 0A as reference on selected axis.

i : Stop Torque Control on Selected ODrive Axis

- Stops torque control

q : Set Current to 1A

- Sets current control reference current.

w : Set Current to -1A

- Sets current control reference current.

Q :Set Current to 10A

- Sets current control reference current.

W : Set Current to -10A

- Sets current control reference current.

space : WheelAccelerationTest()

- Performs wheel acceleration test described in Section 9.2. Data is recorded and then sent to the terminal as comma separated values with the following columns: iterationNumber, motorCurrent[A], omegaWheel[counts/s], voltage[V]

m : HangingTest()

- Performs Cube inertia estimation test as described in Section 9.3. Data is recorded and then sent to the terminal as comma separated values with the following columns: Cube angular velocity [deg/s], wheel speed [counts/s], x acceleration [mG], y acceleration [mG].

o : ODrive_isCalibrationOk

- Prints a first line indicating if encoder is ready and a second line indicating if motor is calibrated.

f : ODrive_getFeedback

- Prints position and velocity feedback in [counts] and [counts/s] respectively.

S : ODrive_SaveConfiguration

- Saves current configuration in selected ODrive nonvolatile memory.

k : ODrive_Reboot

- Reboots selected ODrive.

a : IMU_ReadingLoop()

- Continuously reads and prints all 4 IMUs measurements. The print format is in 4 columns, one per IMU, separated by '|||'. Each column prints the XYZ Accelerometer values [mG], '|' separator and the XYZ Gyroscope values [deg/s].

8.3.2.3. Digital Button

The goal of this function is to raise a flag when the button is pressed or raise another flag when the button is released. These flags will be used by any loop to take an action. This function is implemented in the main.c file.

The algorithm is presented in Figure 75 and is as follows:

1. Generate an interruption when with either a rising or falling edge is detected on the pin.
2. Wait 50ms.
3. Read button state and record as new button state.
4. If previous button state is released and new state is pressed, raise 'Button Pressed' flag.
5. If previous button state is pressed and new state is released, raise 'Button Released' flag.
6. If previous and new button states are equal, do nothing.

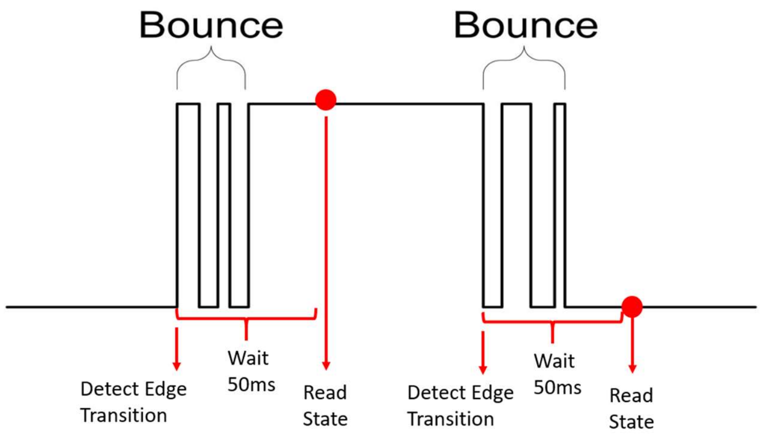


Figure 75 - Digital Button Debouncing Algorithm.

The pin used for this button is PC8, configured as pull up. For edge detection, the pin is configured as EXTI and shall generate an interrupt with either falling or rising edges.

For the waiting stage, a custom `DigitalButton_DecreaseCounter()` function was subscribed to the `SysTick_Handler()` function, which is called periodically at 1ms.

8.3.2.4. ODrive Motor Controller

The ODrive boards are commanded via a UART interface at 115200 baud rate. The peripheral used is specified in the microcontroller interface diagram. All commands are ASCII characters, and the documentation can be found here. All this documentation is downloaded to this project directory, the ASCII protocol is located at ODrive-fw-v0.4.11\docs\ascii-protocol.md .

The driver for this board was developed as part of this project following the provided ODrive documentation.

All functions in this library receive an ODrive Axis instance. Each ODrive board has two axes: 0 and 1. Each instance has the following parameters: UART handler for the board of that axis, axis number and encoder counts. Each instance is created in the main.c function. Read and write functions where implemented which send and receive ascii arrays from the UART interface, all messages are also relayed to the PC UART interface for debugging.

Each axis must be configured with several parameters before calibration, this is done with the `ODrive_ConfigureParameters` function.

Each axis needs to be calibrated before use; this is done by the `ODrive_FullCalibrationSequence` function. After calibration, the axes can be set to several states with the `ODrive_RequestState` function. All possible states are described in ODrive-fw-v0.4.11\Arduino\ODriveArduino\ODriveArduino.h . The two states of interest for this project are AXIS_STATE_FULL_CALIBRATION_SEQUENCE which is used by the `ODrive_FullCalibrationSequence` function, and AXIS_STATE_CLOSED_LOOP_CONTROL which starts the current field oriented control used in the edge balancing control loop.

During the edge balancing control loop, the wheel speed feedback is read with `ODrive_getFeedback` and the current reference is updated by `ODrive_UpdateCurrent`.

Finally, some utility functions are `ODrive_GetVoltage` which reads voltage and relays to PC interface, `ODrive_isCalibrationOk` checks if the axis is already calibrated to avoid re-calibration, `ODrive_Reboot` is for rebooting the board and `ODrive_SaveConfiguration` is for saving the last configuration in the board nonvolatile memory.

Note: the two ODrive boards present in the cube have a slightly different firmware version. ODrive labeled as 0 has version 0.4.11, its documentation can be found here or at ODrive-fw-v0.4.11. ODrive labeled as 1 has version 0.4.12, its documentation can be found here or at ODrive-fw-v0.4.12. No difference in functionality was observed between these two versions and the boards are controlled with the same driver.

8.3.2.5. MPU6050 Accelerometer and Gyroscope Library

Each IMU is defined as a configuration structure. This structure captures the I2C handler used to communicate with it, the I2C address (either 0x68 or 0x69, configured by setting pin AD0 to 0 or 1 respectively), peripheral configuration such as accelerometer and gyro dynamic range, and clocks. Three other parameters are defined for this project. These are the calibration offset for each axis for the accelerometer and gyro measurements, the rotation matrix such that the outputs when reading the accelerometers are aligned to the reference frame of the cube, and the XYZ position in the cube frame of reference. Section 8.5 shows how all these values are obtained.

All IMUs and their corresponding parameters are defined during code initialization. The main functions of these library are: `MPU6050_Config(MPU_ConfigTypeDef *MPU_ID)` which configures each IMU with most of the configuration parameters, `MPU6050_Set_rotMatrix` which writes the rotation matrix, and `MPU6050_Get_InertialData` which reads XYZ accelerometer and gyro values for a particular IMU and outputs it using the IMU data structure also defined in this library.

8.3.2.6. Edge Balancing Control Loop

This function gets called by the main loop with the 'l' command or by pressing the digital button after calibration is done. This is the only function that turns the digital button LED green, therefore indicating when the loop is on.

This function requires to execute a control algorithm every 30ms. This is achieved by raising a flag at this period using TIMER 6 while the control loop is polling such flag. This timer is connected to the APB1 clock bus at 90MHz, therefore by setting a prescaler of 8999, a counter period of 299 and enabling interruptions, we get a call to `HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)` every 30ms, where the flag is raised and can be read by the control loop.

The detailed description of the control algorithm is presented in Section 10.

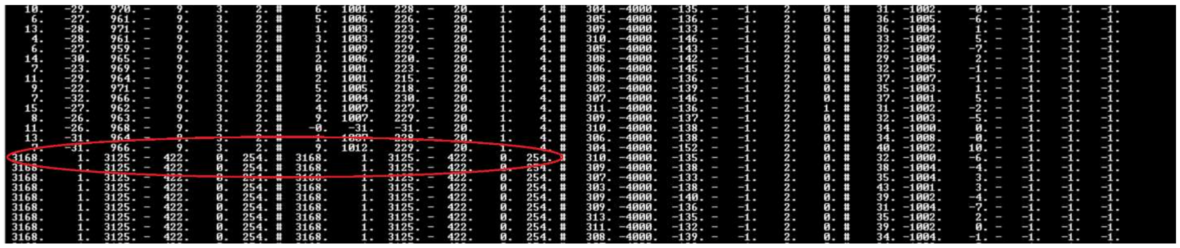
8.4. IMU I^{2}C Bus Crashing

As it can be seen in Figure 47, the 4 IMUs are connected to 2 I^{2}C buses. IMUs 0 and 1 are connected to I2C1 bus while IMUs 2 and 3 are connected to I2C2 bus. IMUs that share the same I^{2}C bus are set to a different I^{2}C addresses by setting AD0 pin to 3.3V (sets address to 0x68) or 0V (sets address to 0x69).

This bug was seen as follows: while reading all IMUs values, if any ODrive motor torque control loop was turned on, suddenly the two IMUs that share the same I2C bus started sending the same constant value.

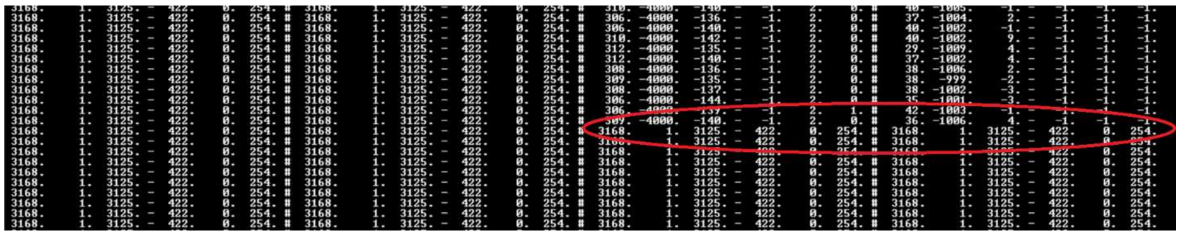
Figure 76 shows the moment the I2C1 bus crashes highlighted in red, this figure shows a console output of all 4 IMUs 3 accelerometer and 3 gyro values, they are ordered as follows: AX1, AY1, AZ1 - GX1, GY1, GZ1 # AX2, AY2, AZ2 - GX2, GY2, GZ2 # AX3, AY3, AZ3

- GX3, GY3, GZ3. Figure 77 shows the moment the I2C2 bus crashes too, while the ODrive control loop is running.



```
  10.   -29.   970. -    9.    3.    2. #    6.  1001.   228. -   20.    1.    4. #   304. -4000.  -135. -   -1.    2.    0. #   31. -1002.   -0. -   -1.   -1.   -1.
   6.   -27.   961. -    9.    3.    2. #    5.  1006.   226. -   20.    1.    4. #   305. -4000.  -136. -   -1.    2.    0. #   36. -1005.   -6. -   -1.   -1.   -1.
  13.   -28.   971. -    9.    3.    2. #    1.  1003.   223. -   20.    1.    4. #   309. -4000.  -133. -   -1.    2.    0. #   36. -1004.    1. -   -1.   -1.   -1.
   4.   -28.   961. -    9.    3.    2. #    3.  1003.   229. -   20.    1.    4. #   310. -4000.  -146. -   -1.    2.    0. #   33. -1002.    5. -   -1.   -1.   -1.
   6.   -27.   959. -    9.    3.    2. #    1.  1009.   229. -   20.    1.    4. #   305. -4000.  -143. -   -1.    2.    0. #   32. -1009.   -7. -   -1.   -1.   -1.
  14.   -30.   965. -    9.    3.    2. #    2.  1006.   220. -   20.    1.    4. #   308. -4000.  -142. -   -1.    2.    0. #   29. -1004.    2. -   -1.   -1.   -1.
   7.   -23.   969. -    9.    3.    2. #    0.  1001.   223. -   20.    1.    4. #   306. -4000.  -145. -   -1.    2.    0. #   32. -1005.   -1. -   -1.   -1.   -1.
  11.   -29.   964. -    9.    3.    2. #    2.  1001.   215. -   20.    1.    4. #   308. -4000.  -136. -   -1.    2.    0. #   37. -1007.   -1. -   -1.   -1.   -1.
   9.   -22.   971. -    9.    3.    2. #    5.  1005.   218. -   20.    1.    4. #   302. -4000.  -139. -   -1.    2.    0. #   35. -1003.    1. -   -1.   -1.   -1.
   7.   -32.   966. -    9.    3.    2. #    2.  1004.   230. -   20.    1.    4. #   307. -4000.  -146. -   -1.    2.    0. #   37. -1001.    5. -   -1.   -1.   -1.
  15.   -27.   962. -    9.    3.    2. #    4.  1007.   227. -   20.    1.    4. #   311. -4000.  -136. -   -1.    2.    1. #   31. -1002.   -2. -   -1.   -1.   -1.
   8.   -26.   963. -    9.    3.    2. #    9.  1007.   229. -   20.    1.    4. #   309. -4000.  -137. -   -1.    2.    0. #   32. -1003.   -5. -   -1.   -1.   -1.
  11.   -26.   968. -    9.    3.    2. #   -0.   -31.   -31. -   20.    1.    4. #   310. -4000.  -138. -   -1.    2.    0. #   34. -1000.    0. -   -1.   -1.   -1.
  13.   -31.   964. -    9.    3.    2. #    1.  1007.   228. -   20.    1.    4. #   306. -4000.  -138. -   -1.    2.    0. #   34. -1008.   -0. -   -1.   -1.   -1.
   7.   -31.   966. -    9.    3.    2. #    9.  1012.   229. -   20.    1.    4. #   304. -4000.  -152. -   -1.    2.    0. #   40. -1002.   10. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   310. -4000.  -135. -   -1.    2.    0. #   32. -1000.   -6. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   309. -4000.  -138. -   -1.    2.    0. #   38. -1004.   -4. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   307. -4000.  -133. -   -1.    2.    0. #   35. -1004.    3. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   303. -4000.  -138. -   -1.    2.    0. #   43. -1001.    3. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   309. -4000.  -140. -   -1.    2.    0. #   39. -1002.   -4. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   309. -4000.  -136. -   -1.    2.    0. #   31. -1004.   -7. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   313. -4000.  -135. -   -1.    2.    0. #   35. -1002.    2. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   311. -4000.  -132. -   -1.    2.    0. #   39. -1002.    0. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   308. -4000.  -139. -   -1.    2.    0. #   34. -1004.   -1. -   -1.   -1.   -1.
```

Figure 76 – I2C1 bus crash, seen as the two IMUs reading a constant value.



```
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   310. -4000.  -140. -   -1.    2.    0. #   40. -1005.   -1. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   306. -4000.  -136. -   -1.    2.    0. #   37. -1004.    2. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   306. -4000.  -140. -   -1.    2.    0. #   40. -1002.   -1. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   310. -4000.  -142. -   -1.    2.    0. #   40. -1002.    9. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   312. -4000.  -135. -   -1.    2.    0. #   29. -1009.    4. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   312. -4000.  -140. -   -1.    2.    0. #   37. -1002.    4. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   308. -4000.  -136. -   -1.    2.    0. #   38. -1006.    2. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   309. -4000.  -135. -   -1.    2.    0. #   38.  -999.   -2. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   308. -4000.  -137. -   -1.    2.    0. #   38. -1002.   -3. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   306. -4000.  -144. -   -1.    2.    0. #   35. -1001.   -3. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   306. -4000.  -137. -   -1.    2.    1. #   42. -1003.   -1. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. #   307. -4000.  -140. -   -1.    2.    0. #   36. -1006.    4. -   -1.   -1.   -1.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254. # 3168.     1.  3125. -  422.    0.  254.
```

Figure 77 - I2C2 bus crash, seen as the two IMUs reading a constant value.

The cause of this error is shown in Figure 78, where the SDA pin is stuck low during communication. This was caused by one of the IMUs holding this pin low without releasing it, this can happen because the I2C any device can pull the SDA pin down, but no device can pull it back up as shown in Figure 79.

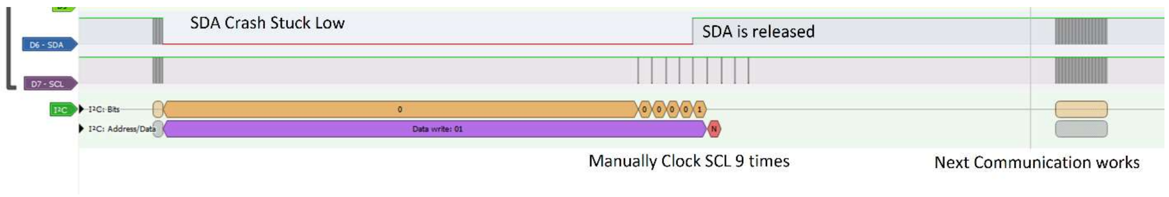


Figure 78 – I2C Logic Analyzer plot. Bus crash and fix.

The logic analyzer debugging was performed using a HiLetgo USB Logic Analyzer using PulseView software.

The solution was found in *Section 3.1.16 – Bus Clear* from UM10204 I2C-bus specification and user manual. This section mentions that "If the data line (SDA) is stuck LOW, the master should send nine clock pulses. The device that held the bus LOW should release it sometime within those nine clocks.”. This solution can be seen in Figure 78.

The software implementation of this solution is as follows:

1. Try sending I2C address. If error callback is called, retry this line.
2. Try receiving I2C data. If error callback is called. Retry starting from previous line.

The error callback performs the following steps to unblock the I2C bus:

1. De-initialize the I2C bus.
2. Initialize SCL pin to GPIO.
3. Manually clock the pin 9 times.
4. De-initialize SCL pin from GPIO.
5. Initialize the I2C bus.

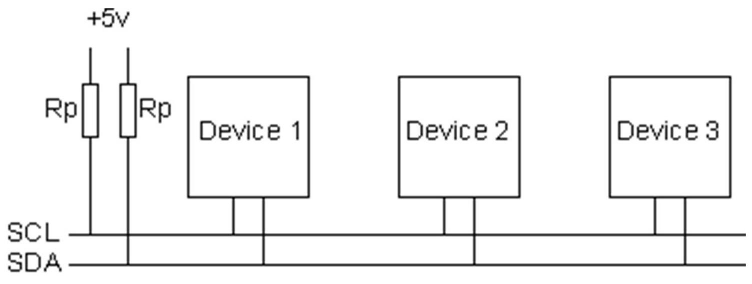


Figure 79 – I2C Bus Topology.

Given that this bus crash occurs while the ODrive control loop is running, the source of this error is probably interference generated by the ODrive, either at the power lines or signal lines. Possible hardware solutions for this problem include:

- Selecting an IMU with a more robust interface such as SPI.
- Twisting signal lines with ground.
- Signal cable shielding.
- Fix ground loops.
- Signal isolation with capacitive couplers.

8.5. IMU Calibration

The MPU6050 presents a constant bias in the acceleration and angular velocity measurements. These biases are present in each measurement axis and are a constant value added to the measurement signal.

Sensor calibration is performed at a stationary condition. In this condition, gravity is present in the accelerometer measurement while zero values are expected for the angular velocity measurement. This leads to two different approaches to calibrate each measurement type, shown in the following two sections.

8.5.1. Accelerometer Calibration Method

To calibrate each axis, two measurements with different gravity direction are used. One with gravity in the positive direction of the axis and the other in the negative.

The following equations show the example for the X axis. Each value in Table 18 corresponds to the average of 1000 measurements with the sensor stationary.

Variable	Value	Unit
$\boldsymbol{A_{x\,Uncal}(g_+)}$	964	mG
$\boldsymbol{A_{x\,Uncal}(g_-)}$	1052	mG

Table 18 - Uncalibrated Measurements X Axis.

Equation (56) shows method for calculating bias value.

$$AccBias_x = \frac{\boldsymbol{A_{x\,uncal}(g_+) + A_{x\,uncal}(g_-)}}{2} \tag{56}$$

Variable	Value	Unit
$\boldsymbol{AccBias_x}$	-44	mG

Table 19 – Calibration Offset.

Equation (57) shows the method used to subtract the bias value.

$$A_{x\,Calibrated} = A_{x\,Uncal} - AccBias_x \tag{57}$$

8.5.2. Gyroscope Calibration Method

Given that the nominal output of the sensor in stationary condition is zero, calibration of this measurement consists of averaging 1000 measurements with the sensor stationary.

$$GyroBias_x = \omega_{x\,Uncal}(stationary) \tag{58}$$

Equation (59) shows the method used to subtract the bias value.

$$\omega_{x\,Calibrated} = \omega_{x\,Uncal} - GyroBias_x \tag{59}$$

8.5.3. All IMUs Accelerometer and Gyroscope Simultaneous Calibration Method

The cube's 4 IMUs were calibrated simultaneously mounted in the assembled cube.

- Measurements taken:
  - 6 measurements, each measurement with a different cube face laying on a flat surface as shown in Table 20. The raw data files are then named X+, X-, Y+, Y-, Z+, Z-.
- Data recorded in each measurement:
  - All IMU outputs of the 4 IMUs (XYZ acceleration and angular velocity)
  - Number of samples: 1000.

Data processing code found in Appendix A, this code arranges the data in a 4 dimensional matrix with ID, value measured, test gravity coordinate, test gravity direction. With this, the calculations shown in 8.5.1 and 8.5.2 are performed to obtain calibrations of the accelerometer and gyro respectively. Finally, it generates a calibration C code that is ready to paste into the source code and will load the corresponding calibration values to the IMUs following the library format.

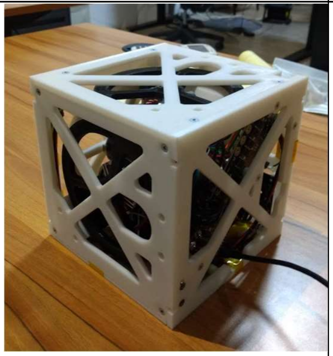
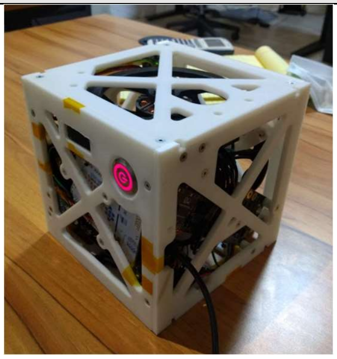
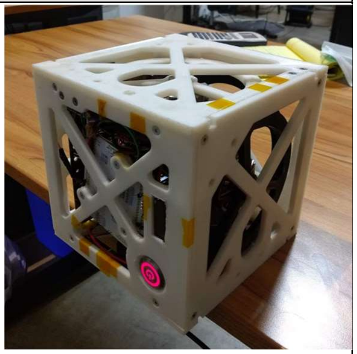
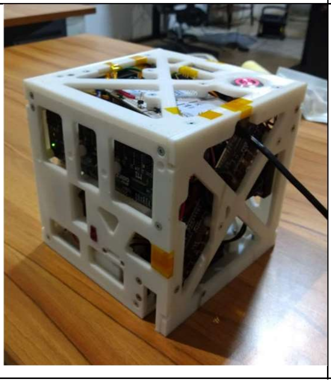
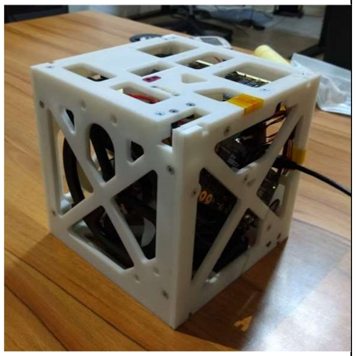
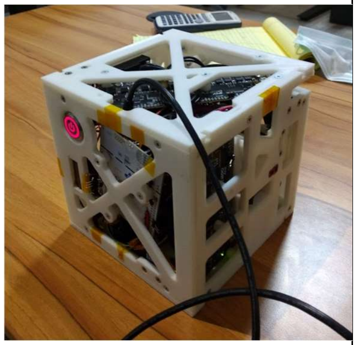
	X	Y	Z
+			
-			

Table 20 - Cube IMU Calibration Measurement Positions.

8.5.4. IMU Rotation Matrices

The IMU rotation matrices were found by first configuring all rotations matrices to the identity matrix, setting the cube to a known orientation, record the measured gravity direction and then adjust the matrix coefficients such that the output matches the expected direction. This process was done for all axes.

8.5.5. IMU Coordinates and Numbering

The IMU coordinates are defined in the cube reference frame as defined in the CATIA model, the numbering is defined as shown in Figure 80. Table 21 shows the coordinates obtained from the CATIA model. Appendix B shows the MATLAB code that generates the calibration C code from this excel table.

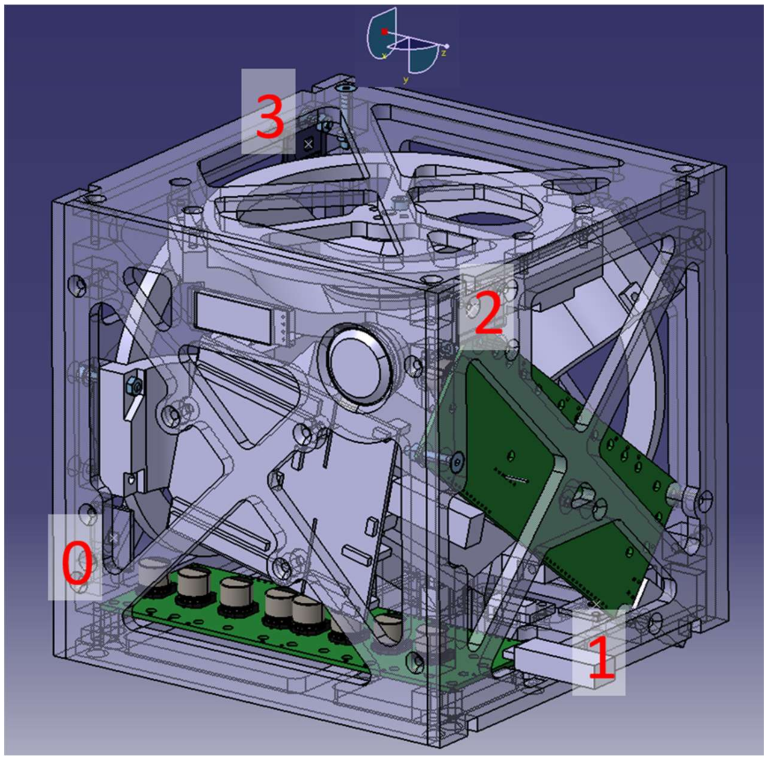


Figure 80 – Cube Coordinate System and IMU Numbering.

ID	X [mm]	Y [mm]	Z [mm]
0	148.1	112.94	19.2
1	27.84	148.1	122.8
2	125.95	13.95	148.1
3	26.06	15.2	6.9

Table 21 – IMU Coordinates.

9. System Model Identification

The goal of this system identification procedure is to find the state space matrices of the linear model of the system presented in Section 4.1.3. The state space model is given by:

$$\begin{cases} \dot{\boldsymbol{X}}(t) = A\,\boldsymbol{X}(t) + B\,u(t) \\ \boldsymbol{Y}(t) = C\,\boldsymbol{X}(t) + D\,u(t) \end{cases} \tag{60}$$

Where the input, output and state vectors are:

$$\boldsymbol{X}(t) = \boldsymbol{Y}(t) = \begin{bmatrix} \theta \\ \dot{\theta} \\ \omega_r \end{bmatrix}, u(t) = i \tag{61}$$

And state space matrices:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ \frac{k_{mgd}}{I_{se}} & 0 & 0 \\ -\frac{k_{mgd}}{I_{se}} & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ -\frac{k_t}{I_{se}} \\ \frac{I_r + I_{se}}{I_r \cdot I_{se}} k_t \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad D = [0] \tag{62}$$

The following sections obtain values for the state space matrices.

- Section 9.1: k_t
- Section 9.2: I_r
- Section 9.3: I_{se}, k_{mgd}

9.1. Motor Characterization

The motor characteristic parameters are its resistance R and motor constants k_v. The resistance is measured by the ODrive motor driver to be $R = 0.1\ \Omega$. To measure the motor constant k_v, a simultaneous measurement of the voltage amplitude generated at a given speed shall be performed while the motor is freely spinning.

$$k_v = \frac{V_{free}}{\omega_{free}} \tag{63}$$

The simultaneous measurement of V and ω is performed by measuring the voltage across any two motor leads with an oscilloscope with the following procedure:

1. Set motor driver to maximum speed
2. Turn off motor driver and set outputs to high impedance
3. Capture the instant the driver is turned off and the first freely spinning electrical oscillation.

Figure 81 shows the result of this experiment, the oscillation voltage amplitude is $V_{free} = 11.9V$ and the period is $T_{electrical} = 1.2ms$.

ω_{free} is calculated by considering that the motor has 12 pole pairs, therefore one mechanical period generates 12 electrical periods at the motor terminals, therefore:

$$\omega_{free} = \frac{2\,\pi}{T_{electrical} \cdot 12} = 436\frac{rad}{s} \tag{64}$$

The motor speed constant $k_v[Vs]$ and motor torque constant $k_t\left[\frac{Nm}{A}\right]$ are theoretically equal given that $V\,s = \frac{W}{A}\,s = \frac{J}{s\,A}s = \frac{J}{A} = \frac{Nm}{A}$. Therefore k_v and k_t are:

$$k_v = \frac{11.9\,V}{436\frac{rad}{s}} = 0.027\ [Vs], \qquad k_t = 0.027\left[\frac{Nm}{A}\right] \tag{65}$$

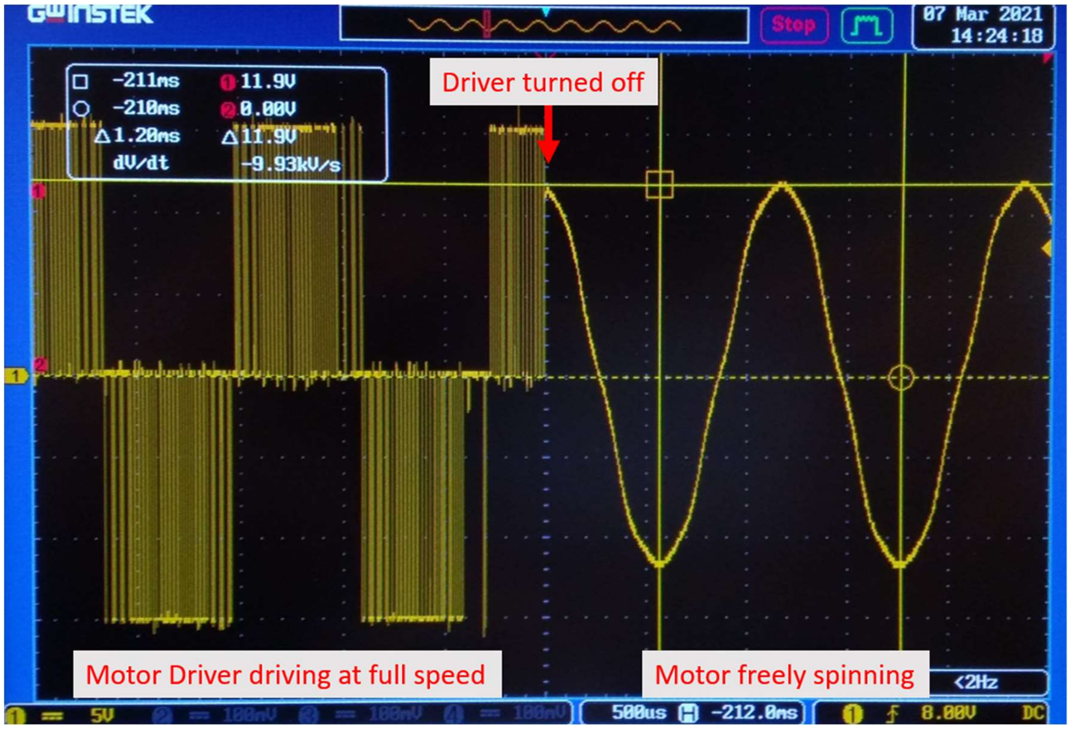


Figure 81 – Motor Constant Measurement.

9.2. Reaction Wheel Inertia Measurement

The reaction wheel inertia is measured by accelerating the wheel using the motor at a constant current and measuring the acceleration produced. The equation of motion of this experiment is:

$$I_r \dot{\omega}_i = k_t \, i \tag{66}$$

Where I_r is the wheel inertia, $\dot{\omega}_i$ wheel acceleration, k_t motor torque constant and i is the motor current. The measurable parameters are the wheel acceleration $\dot{\omega}_i$ and the motor current i. Both variables are read by the ODrive motor driver. With this measurement, the motor constant to wheel inertia quotient is calculated as follows:

$$\frac{\dot{\omega}_i}{i} = \frac{k_t}{I_r} \tag{67}$$

9.2.1. Method

The wheel acceleration was measured for a series of currents as shown in Table 22. At 1 second, the current is set to 1A to take the wheel to maximum velocity and start the characterization measurement currents.

The current setpoints are alternating between negative and positive to take the wheel from negative maximum velocity to positive maximum velocity (or vice versa) to measure the wheel acceleration for a given current both during wheel deceleration and wheel acceleration.

Time [s]	Current Setpoint [A]
0	0
1	1
5	-0.3
25	0.5
35	-0.7
45	1
50	-2
53	3
56	-5
58	10
60	0

Table 22 – Wheel acceleration experiment current setpoints.

Figure 82 shows the unprocessed data acquired in the experiment, wheel speed and motor current. After applying each setpoint, enough time is given for the wheel to fully decelerate, fully accelerate and stabilize at maximum velocity. The slope of the curve changes noticeably while going from deceleration to acceleration (zero velocity crossing) for each current setpoint.

Figure 83 shows wheel acceleration and motor current. The acceleration difference between acceleration and deceleration is also visible.

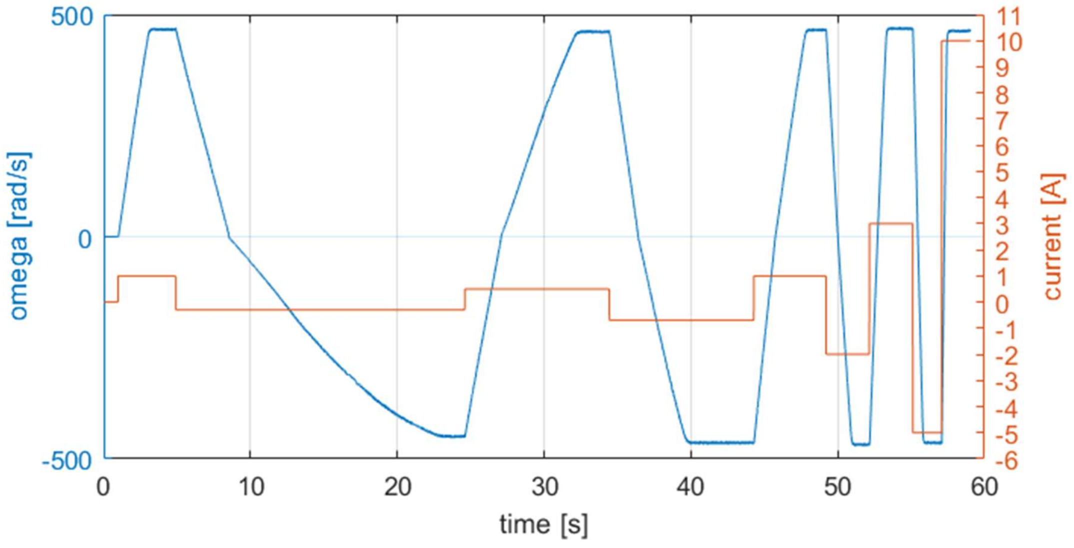


Figure 82 – Wheel Speed and Motor current vs time.

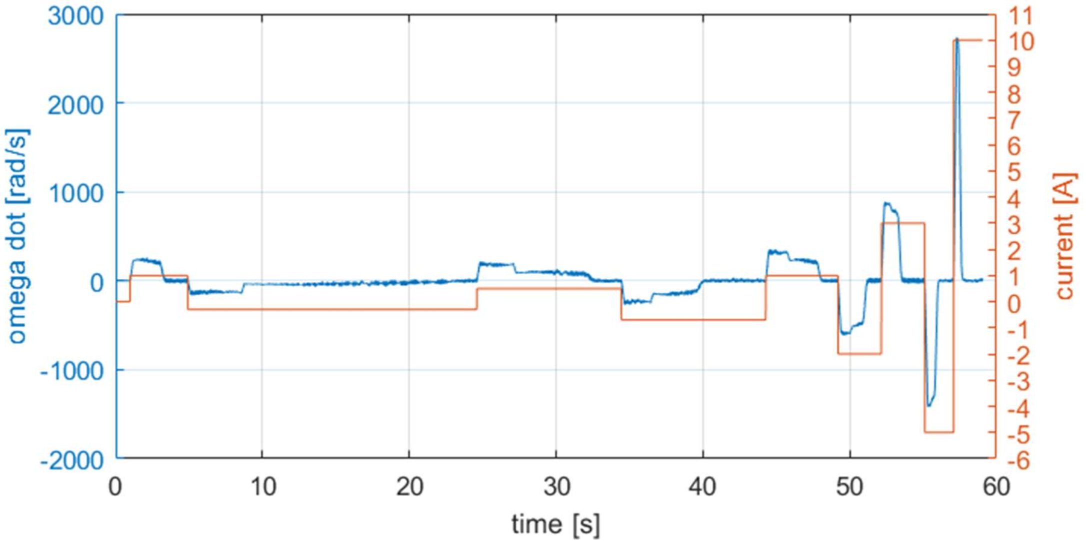


Figure 83 – Wheel Acceleration and Motor current vs time.

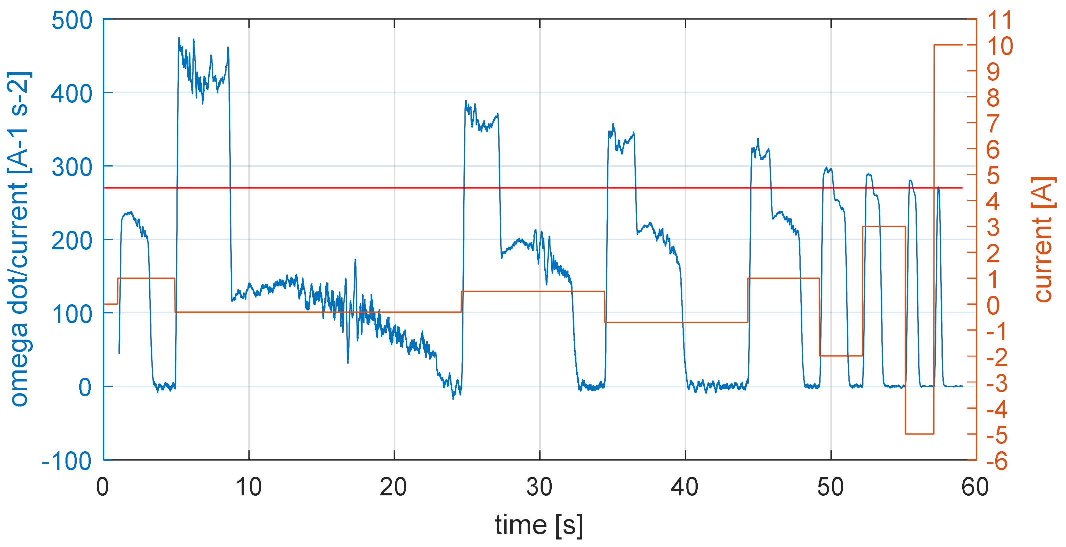


Figure 84 – Wheel acceleration to motor current quotient vs time.

Figure 84 shows the wheel acceleration to motor current quotient calculated for every point in time. The calculated quotient is valid while the wheel is accelerating or decelerating. When the motor reaches maximum velocity, this quotient is zero and it is not valid for this characterization. This quotient is larger during deceleration and smaller during acceleration. For larger currents, the deceleration quotient decreases, and the acceleration quotient increases, decreasing the difference between them. The values appear to converge to a value shown with a red line, and for 10A, there is no measurable difference between both cases. Therefore, the parameter is estimated to converge at:

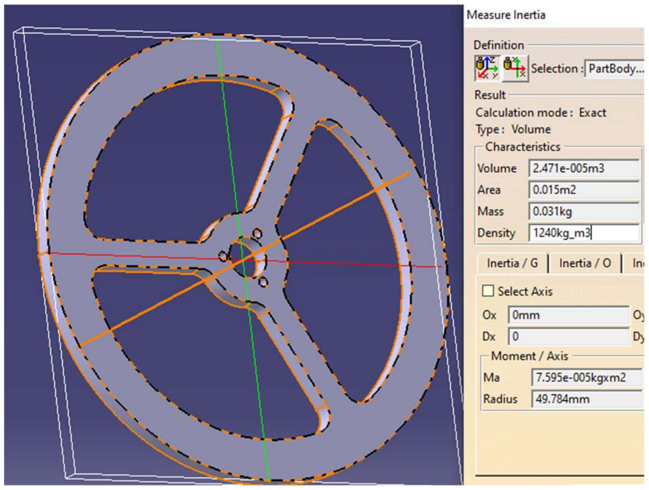
$$\frac{\dot{\omega}_i}{i} = \frac{k_t}{I_r} = 270 \frac{1}{A\, s^2} \tag{68}$$

Finally, the reaction wheel inertia is estimated as:

$$I_r = k_t \frac{i}{\dot{\omega}_i} = \frac{0.027 \frac{Nm}{A}}{270 \frac{1}{A\, s^2}} = 1.01 \times 10^{-4}\, kg\, m^2 \tag{69}$$

To check if this measurement is within an expected value, the inertia of the reaction wheel alone is calculated in CATIA, considering PLA density, we get an inertia of

$7.59 \times 10^{-5} kg\, m^2$, which is lower than measured. This can be explained considering the measurement also includes the inertia of the motor rotor.



Measure Inertia

Definition

Selection : PartBody...

Result

Calculation mode : Exact

Type : Volume

Characteristics

Volume	2.471e-005m3
Area	0.015m2
Mass	0.031kg
Density	1240kg_m3

Inertia / G | Inertia / O | In

Select Axis

Ox	0mm
Dx	0

Moment / Axis

Ma	7.595e-005kgxm2
Radius	49.784mm

Table 23 – CATIA Reaction wheel inertia estimation.

Other methods of measuring inertia can also be used such as using a torsional pendulum.

9.3. Cube Inertia Estimation

The goal of this test is to estimate the cube inertia around an edge of interest with 3 measurements:

1. Hang the cube and measure oscillation frequency
2. Mass
3. Center of mas distance to edge

9.3.1. Model

The equation of motion of the cube hanging is described by Equation (29), which is the same equation of motion of the inverted cube with a sign flip to the gravity toque term.

$$I_{se} \cdot \ddot{\theta} = -M_s\, g\, d_{se} \cdot \theta \tag{70}$$

Where I_{se} is the cube inertia, θ cube angle, M_s cube mass, d_{se} cube center of mass distance to edge and g gravitational acceleration. The solution to this equation is given by:

$$\theta(t) = A \sin(2\pi f\ t) \tag{71}$$

where A is the amplitude, f is the oscillation frequency and t time.

Substituting this solution into Equation (70) and solving for I_{se} we get:

$$I_{se} = \frac{M_s\ g\ d_{se}}{(2\pi f)^2} \tag{72}$$

9.3.2. Method

9.3.2.1. Oscillation Frequency

The cube is hung by a hook to a V profile beam which acts as a pivot between to support positions, as shown in Figure 85.

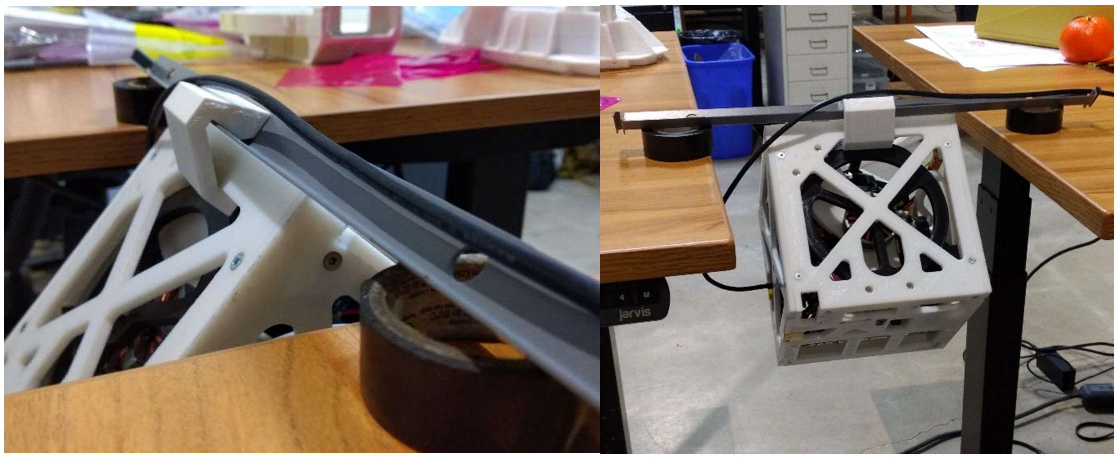


Figure 85 – Cube Hanging Test Method.

The cube is manually set to an inclined position and let go. The angular velocity over time during oscillation is recorded. The oscillation frequency is estimated from this measurement.

Figure 86 shows the recorded angular velocity of the cube over the duration of the experiment.

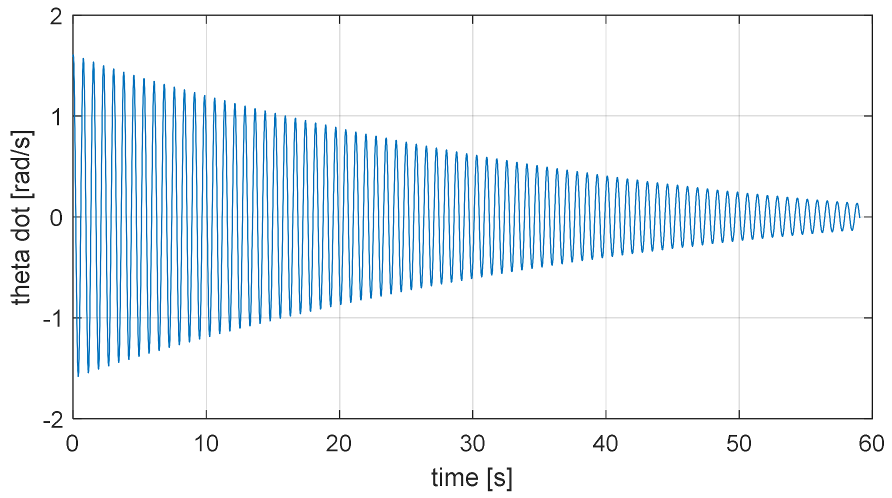


Figure 86 – Hanging Cube Test Recorded Data.

Figure 87 shows the amplitude spectrum of the recorded data. The peak of this plot is used to measure the oscillation frequency.

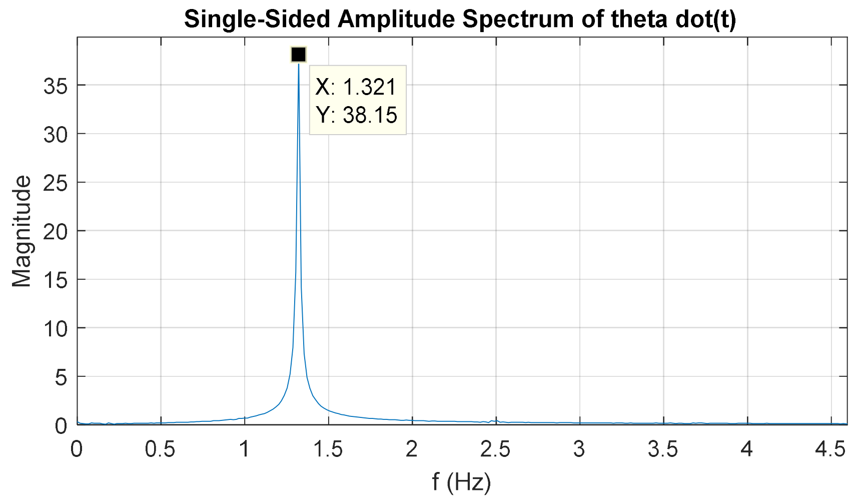


Figure 87 – Hanging Test Amplitude Spectrum.

An oscillation frequency of $1.32\ Hz$ was measured, giving an estimation of this parameter as shown in Equation (73).

$$\frac{k_{mgd}}{I_{se}} = (2\pi\ 1.32\ Hz)^2 = 68.8\frac{1}{s^2} \tag{73}$$

9.3.2.2. Mass

The cube is weighed in a digital scale, showing a mass of $1.337\ kg$.

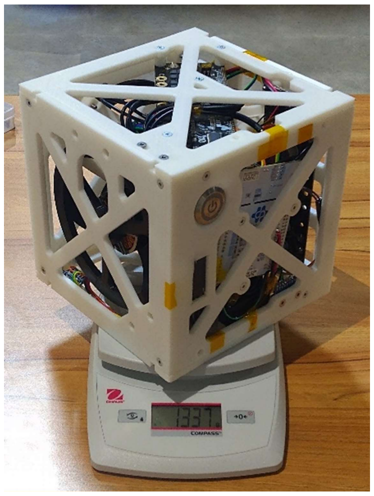


Figure 88 - Cube Weight.

9.3.2.3. Center of Mass Distance to Edge

This distance is measured by measuring dx and dy as shown in Figure 89. These distances are measuring by manually finding the balancing position of the cube around a line.

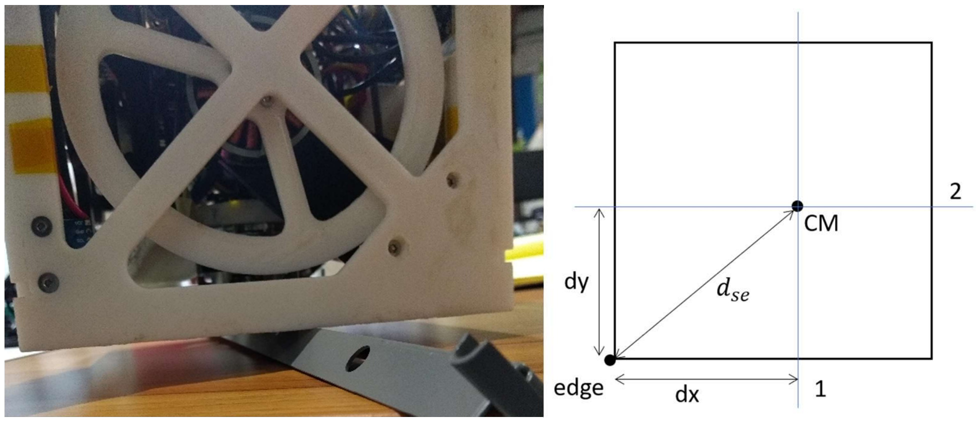


Figure 89 – Center of Mass Location Estimation.

The estimated dx and dy where $79.4mm$ and $75.4mm$ respectively, giving an estimation for d_{se} of $109.5mm$.

9.3.3. Result

From the three measurements and Equation (72), the inertia is estimated as:

$$I_{se} = 0.02086\,kg\,m^2 \tag{74}$$

The gravity torque constant is estimated as:

$$k_{mgd} = 1.434\,Nm \tag{75}$$

10. Edge Balancing Control System Implementation & Tunning

10.1. Single Edge Balancing

This section describes the controller algorithm to balance the cube on a single edge and the methods used to measure the required feedback variables.

As described in Section 4.1.5, the balancing control system is a discrete time linear quadratic regulator. During the first tests, Equation (28) was implemented and it was observed that the cube stabilized with a steady state constant wheel speed, when a zero speed is desirable. As shown in [1], this is because of a constant offset present in the θ measurement. To account for this, the term $\theta[k]$ in the feedback equation was changed to $\theta_m[k] - \theta_o[k]$ to estimate and subtract this offset, where θ_m is the estimated angle.

$$u[k] = K_1(\theta_m[k] - \theta_o[k]) + K_2\dot{\theta}[k] + K_3\omega_r[k] \tag{76}$$

This variable has the following dynamics:

$$\theta_o[k] = (1 - \alpha_o)\theta_o[k-1] + \alpha_o\,\theta_m[k] \tag{77}$$

After iteration with the real system, $\alpha_o = 0.001$ was selected, giving a stable system.

It can be shown that θ_o converges to the constant offset in the steady state. Consider the measured θ consisting of a real θ_r and a constant offset d as shown here:

$$\theta_m = \theta + d \tag{78}$$

The steady state equations are:

$$\bar{x}_{ss} = G\,\bar{x}_{ss} - H\,K\,\bar{x}_{ss} - H\,K_1(d - \theta_{o\,ss}) \tag{79}$$

$$\theta_{o\,ss} = (1 - \alpha_o)\theta_{o\,ss} + \alpha_o\,(\theta_{ss} + d) \tag{80}$$

Solving for $\bar{x}_{ss}$ and $\theta_{o\,ss}$ we get:

$$\bar{x}_{ss} = -(I_3 - G + HK)^{-1} HK_1 (d - \theta_{o\,ss})$$

$$\begin{pmatrix} \theta_{ss} \\ \dot{\theta}_{ss} \\ \omega_{r\,ss} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ h \end{pmatrix} (d - \theta_{o\,ss}) \tag{81}$$

$$\theta_{o\,ss} = \theta_{ss} + d \tag{82}$$

Where $h \neq 0$. Therefore, we get that:

$$\begin{pmatrix} \theta_{ss} \\ \dot{\theta}_{ss} \\ \omega_{r\,ss} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \qquad \theta_{o\,ss} = d \tag{83}$$

The discrete control period is set to $T = 29.55ms$ and is verified by measuring the communication signals using a logic analyzer as shown in Figure 90.

The controller feedback constants are obtained by first discretizing the continuous state space model and obtaining the discrete state space matrices using the *c2c()* MATLAB function and the zero order hold method.

$$\bar{x}[k+1] = G\,\bar{x}[k] + H\,u[k] \tag{84}$$

$$G = \begin{bmatrix} 1.0302 & 0.0298 & 0 \\ 2.0528 & 1.0302 & 0 \\ -2.0528 & -0.0302 & 1 \end{bmatrix}, \qquad H = \begin{bmatrix} -0.0006 \\ -0.0386 \\ 7.9381 \end{bmatrix} \tag{85}$$

Finally, the controller feedback constants are obtained by using the *dlqr()* MATLAB function with the discrete system space model and the following weight matrices:

$$Q = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad R = 10000 \tag{86}$$

$$K = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix} = \begin{bmatrix} -120.6415\text{ A/rad} \\ -14.5543\text{ A/(rad/s)} \\ -0.0075231\text{ A/(rad/s)} \end{bmatrix} \tag{87}$$

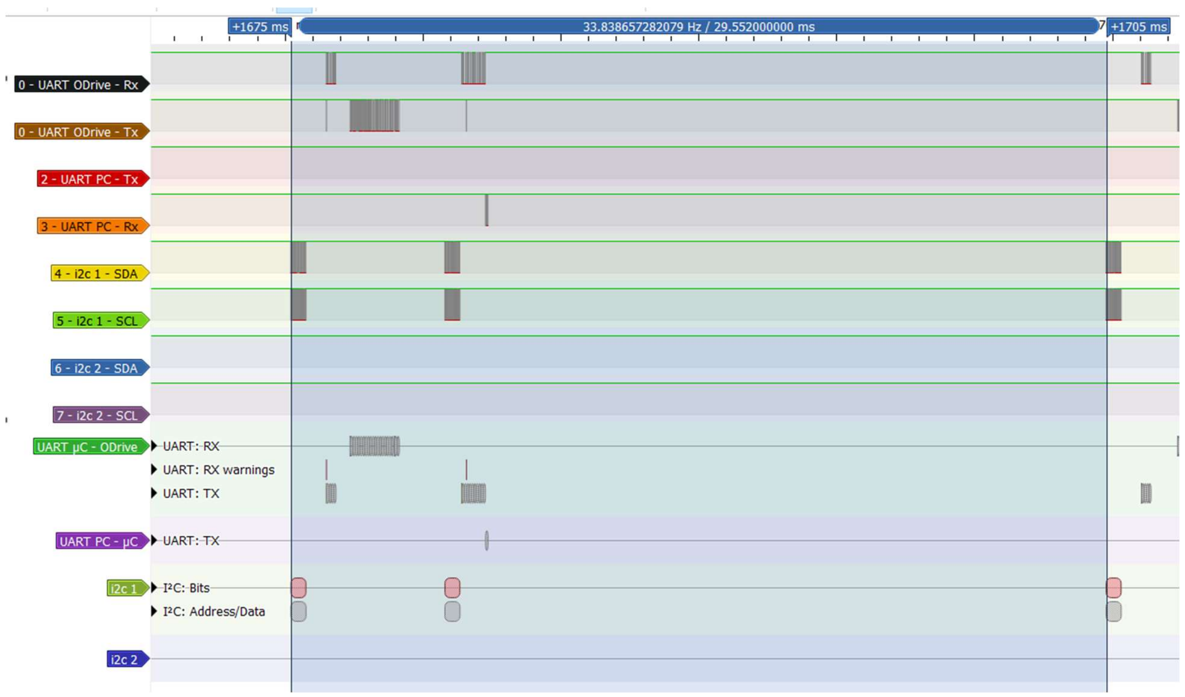


Figure 90 – Digital Control Loop Period Measurement using logic analyzer on communication signals.

The wheel speed state ω_r is measured by the ODrive board and is read by the microcontroller using the UART interface. The STM32 microcontroller sends the request feedback command `f %d\n`, where `%d` is the axis used either 0 or 1, and receives two floating point numbers corresponding to the wheel position and speed respectively. The units received are in encoder counts and are converted to rad/s by dividing by the encoder resolution and multiplying by 2π.

The cube angular velocity state $\dot{\theta}$ is measured by the MPU6050 and read by the microcontroller using the I2C interface. The received value is a digital number corresponding to the angular velocity multiplied by a scaling factor configured during peripheral initialization. With this scaling factor, the digital number is converted to rad/s.

The cube angle state θ is estimated by the combining the acceleration and gyroscope measurements from the MPU6050 using the following complementary filter:

$$\theta_m[k] = \alpha\,\theta_{acc}[k] + (1-\alpha)\,\theta_{gyro}[k] \tag{88}$$

Where θ_m is the estimated angle used by the feedback loop, θ_{acc} is the estimated angle using accelerometer measurements, θ_{gyro} is the estimated angle using gyro measurements and $\alpha \in (0, 1)$ is the term weight parameter.

The sensor fusion algorithm is used to compensate the following undesirable effects on the measurement of the gravity vector: inherent sensor noise, vibrations from reaction wheel spinning and induced accelerations due to motion of the cube. Estimating the angle by integrating gyroscope measurements is immune to these problems at the cost of integrating the sensor bias, which will result in accumulating error over time. Therefore, the sensor fusion algorithm consists in combining the accelerometer and gyroscope estimations of angle, keeping the low frequencies of the accelerometer and the high frequencies of the gyroscope.

θ_{gyro} is calculated by numerically integrating the last measured angular velocity $\dot{\theta}$ starting from the last estimation of the cube angle θ_m using the controller period T, as follows:

$$\theta_{gyro}[k] = \theta_m[k-1] + \dot{\theta}[k] \cdot T \tag{89}$$

Figure 92 shows how θ_{acc} is calculated by calculating the gravity direction in IMU coordinates θ_{abs}, and then subtracting that angle from the gravity direction in the stable position θ_{stable}, shown as follows:

$$\theta_{abs}[k] = atan2\left(g_{y_{IMU}}[k], g_{x_{IMU}}[k]\right) \tag{90}$$

$$\theta_{acc}[k] = \ \theta_{stable} - \theta_{abs}[k] \tag{91}$$

Figure 91 shows the comparison of these methods. The high noise of the accelerometer measurements and the bias integration of the gyro can be seen. The sensor fusion accounts for these two problems to generate a better approximation of the cube angle.

The filter constant α is selected such that the filter will converge to 95% of the acceleration value for 0 gyroscope input after 1 second. The equation for this case is:

$$\theta_m[k] = \alpha\, \theta_{acc}[k] + (1-\alpha)\, \theta_m[k-1] \tag{92}$$

Considering an initial value for $\theta_m[0] = 1$ and $\theta_{acc}[k] = 0,\ \forall\, k$, the kth value for θ_m is:

$$\theta_m[k] = (1-\alpha)^k \tag{93}$$

1 second at $T = 29.55ms$ takes 34 samples, therefore $\theta_m[34] = 0.05$, we then get:

$$(1-\alpha) = 0.05^{\left(\frac{1}{34}\right)} = 0.9782, \qquad \alpha = 0.0218 \tag{94}$$

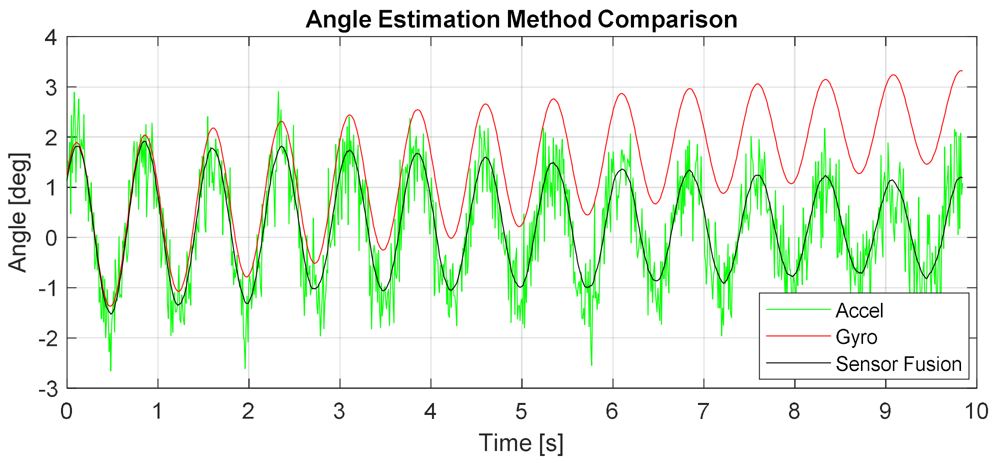


Figure 91 – Angle Estimation Method Comparison.

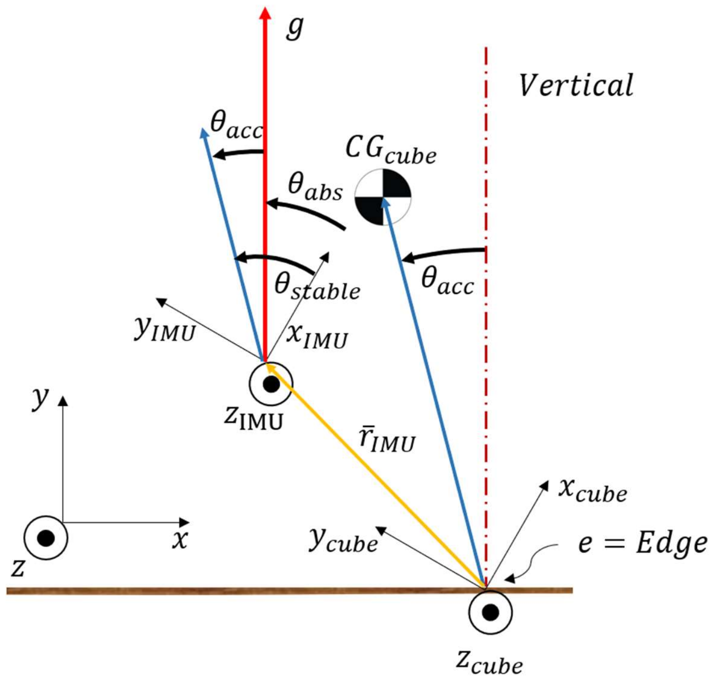


Figure 92 – Coordinate system and vector definitions for IMU θ_{acc} estimation.

Finally, the calculated controller current output $u[k]$ is sent to the ODrive current control loop as a reference value using the UART interface. The microcontroller sends update current command `c %d %.3f\n`, where `%d` is the axis used either 0 or 1 and `%.3f` is the new current value sent as a floating point value.

10.2. Any Edge Balancing

This section presents the steps required to balance the cube on any edge. A high level description of the steps is presented first, followed by a detailed description of each. The steps are:

1. Identify to which edge is gravity closest.
2. Identify the axis corresponding to that edge.
3. Select the IMU closest to that edge as the source of acceleration and gyro measurements. This is to reduce rotational accelerations induced on the accelerometer measurement.
4. θ Measurement: Calculate theta on the plane perpendicular to that axis and subtract from the stable direction corresponding to that edge.
5. $\dot{\theta}$ Measurement: Get this value from the selected IMU and the corresponding axis.
6. ω Measurement: Select the ODrive Axis corresponding to that axis.
7. Current Command: Select the ODrive Axis corresponding to that axis.

10.2.1. Identify Edge Closest to Gravity and Corresponding Axis

Each of the 12 edges is uniquely characterized by the two following properties:

1. X,Y,Z: Axis to which they are parallel.
2. 0,1,2,3: Quadrant (on plane perpendicular to the parallel axis) in which they are located.

Also, for each edge, there is a gravity acceleration direction in which the cube center of mass will be above the edge. These directions are called stable gravity acceleration directions (SD), and point in the opposite direction to the location of the edge.

Figure 93 shows the edge naming for the 4 edges parallel to the cube Z axis. The figure also shows the stable gravity acceleration directions (SD) for each edge, and the IMU locations as seen from this axis.

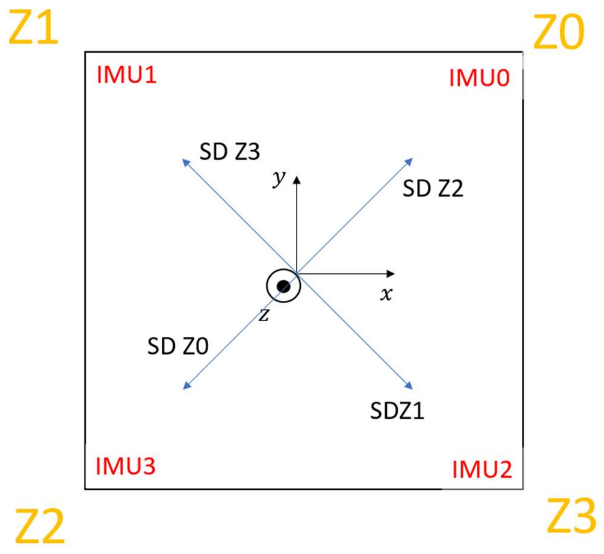


Figure 93 – Edge Nomenclature Example for Z direction.

Figure 94 shows the unique identification of all the cube edges following this rule.

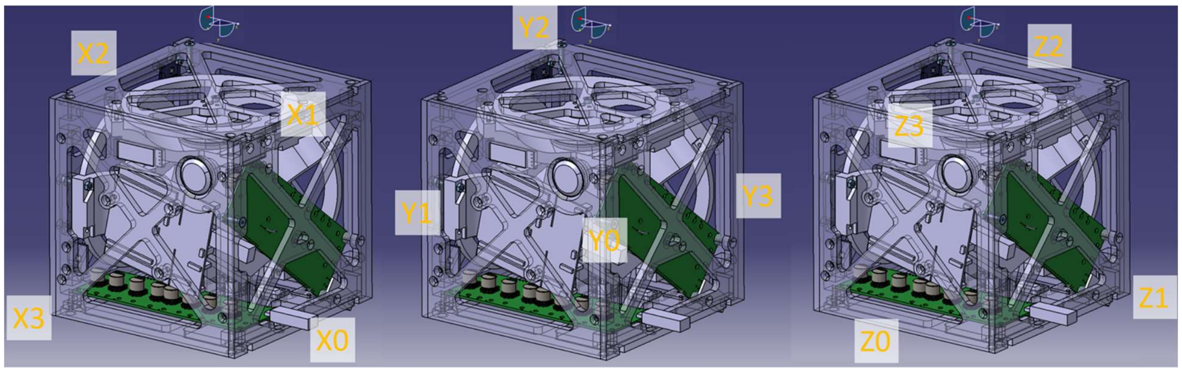


Figure 94 - Edge Identification.

Therefore, in each control loop iteration, the edge closest to the balancing position is found by measuring the angle between the gravity direction at that instant and the 12 stable directions for each edge. When an angle of less than 10° is measured, that edge is selected with its corresponding axis. If no stable direction is within this range, the controller outputs 0A as current reference. Appendix C shows the C code implementation of this algorithm.

The matrix `axis_quadrant_to_stableDirectionVector` is generated with the MATLAB code presented in Appendix D.

10.2.2. Select IMU for corresponding edge

As shown in the C code in Appendix C, the IMU is selected from the `axis_quadrant_to_IMU_ID` matrix. This matrix is generated by the MATLAB code in Appendix E. This code takes as input the coordinates of each IMU and assigns it to the closest edge.

Figure 95 shows a graphical representation of the algorithm in Appendix E. Each IMU is connected to the 3 edges to which they correspond with a line of two sections, one section of the line is the radial distance perpendicular to the axis and the other section is the longitudinal distance parallel to the axis.

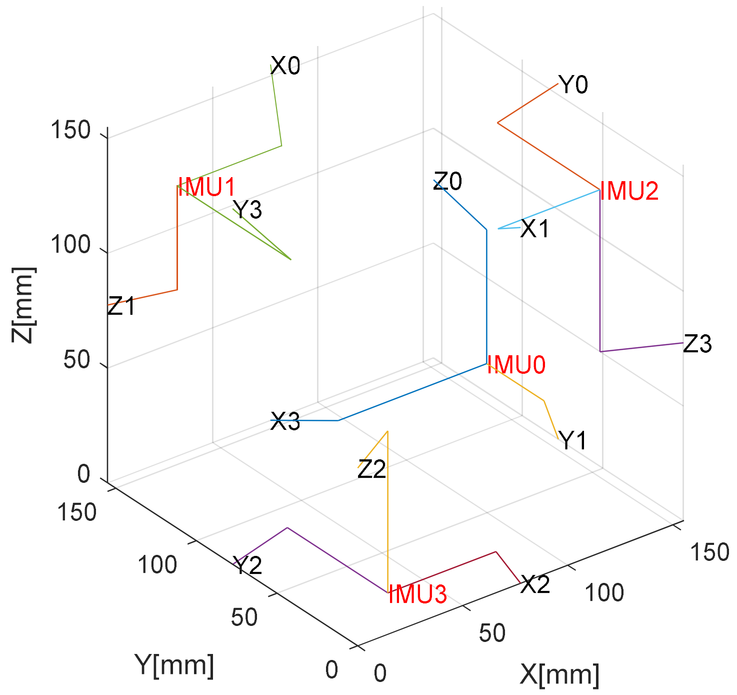


Figure 95 – IMU Selection for each corresponding edge.

11. Edge Balancing Test Results

The edge balancing was tested under two conditions: Steady state balancing and impulse disturbances. These tests are described in the following sections.

11.1. Steady State Balancing

Figure 96 shows the recording of the states of the system during this test, where the cube is balancing over its edge in steady state with no external torques being applied for 11 minutes.

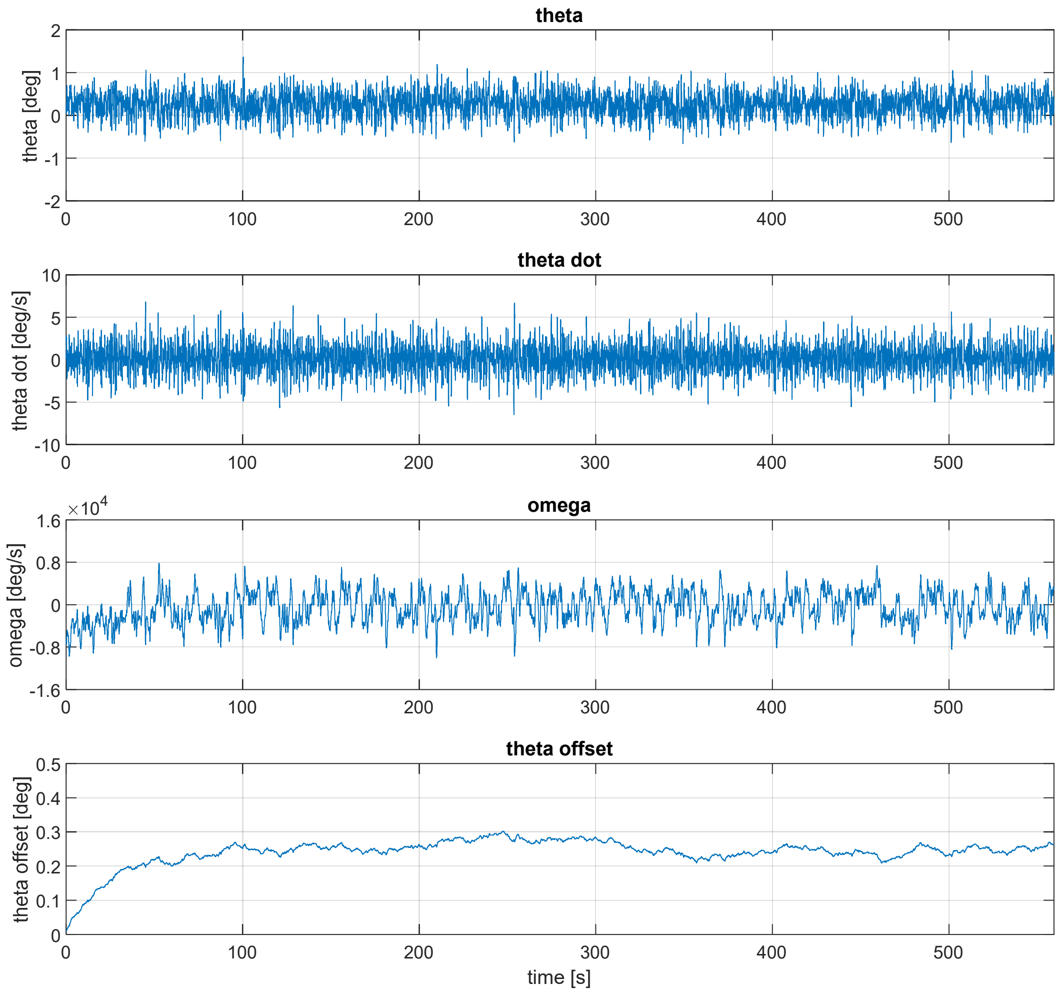


Figure 96 – Steady State Edge Balancing. States Recording over time.

This test is performed as follows: with the control system turned off, manually take the cube as close as possible to the vertical position, then turn on the control system and let the cube go, record the cube states.

The cube successfully balanced for 11 minutes. In the theta offset plot from Figure 96, it can be seen how the filter starts at 0 and converges to the offset value after about 90 seconds.

The angle of the cube θ moves randomly around the offset position. The standard deviation of this motion is measured at 0.25°. This motion is mainly attributed to the static friction and torque ripple from the motor.

11.2. Impulse Disturbance

This test is performed as follows: with the control system turned off, manually take the cube as close as possible to the vertical position, then turn on the control system and let the cube go. Wait for theta offset to stabilize, hit the cube to create a torque impulse, wait for the cube to recover and hit again. Repeat 7 times.

Figure 97 shows the results of this test. Each hit is marked with a red arrow. The test shows the cube can stabilize after small hits. Larger hits will tip the cube away from the stable condition. It can also be seen that the theta offset reacts to the hits and follows theta as it is stabilized.

The third hit, marked by the third arrow from the left, is compared to the response of the linear system in Figure 98. The comparison starts at the start time of the impulse in the test data. The states of the cube at this instant in time are fed as initial state conditions for the linear system and then the simulation is ran. The linear system follows closely the real system for the first 250ms and then they start to deviate. This deviation is mainly attributed to the motor static friction and torque ripple. This effect can be seen in Figure 84 from Section 9.2, where the motor and wheel equation $I_r\,\dot{\omega} = k_t\,i$ does not hold and varies depending on the wheel speed and direction.

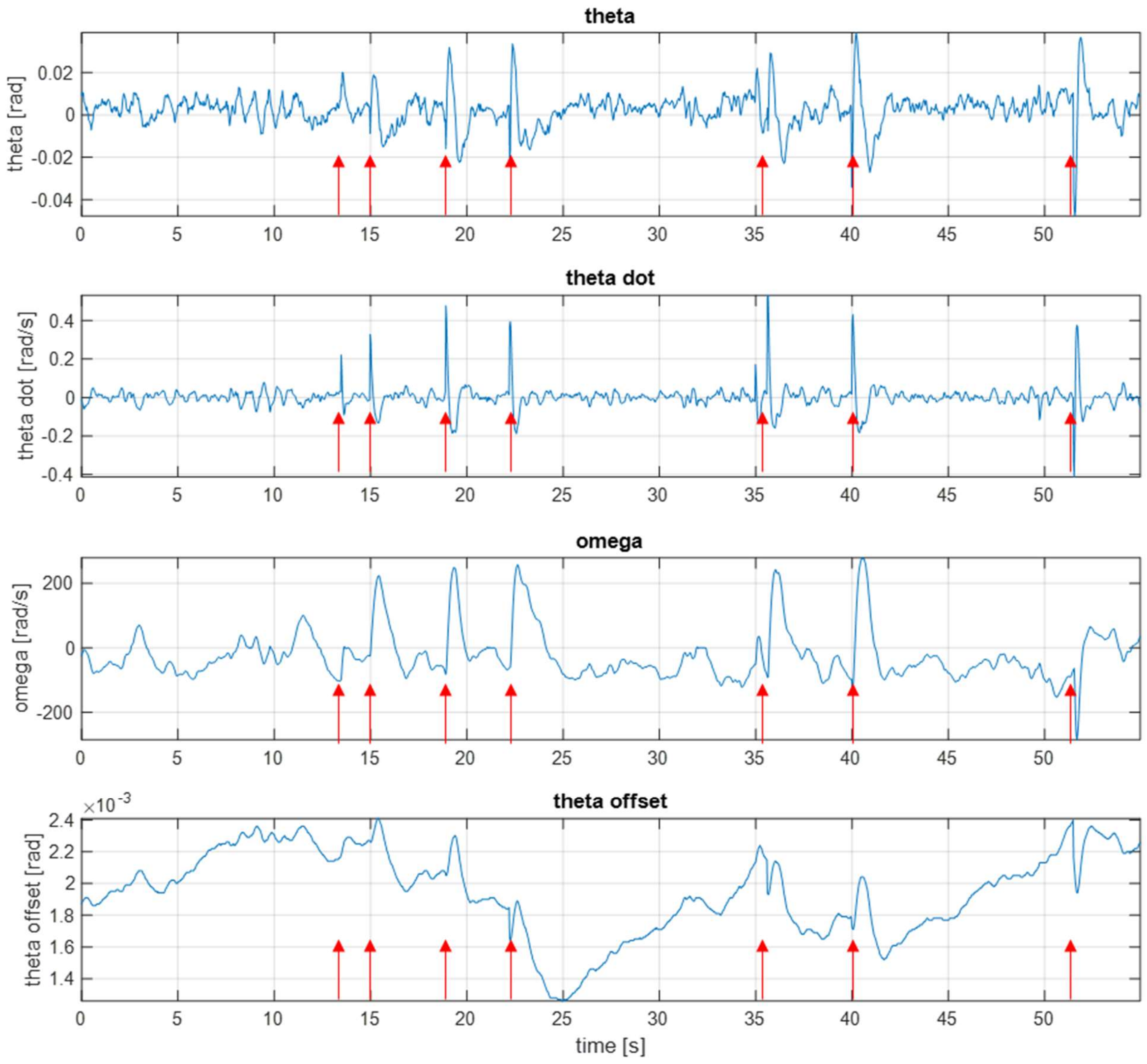


Figure 97 – Edge balancing with external impulse torques applied.

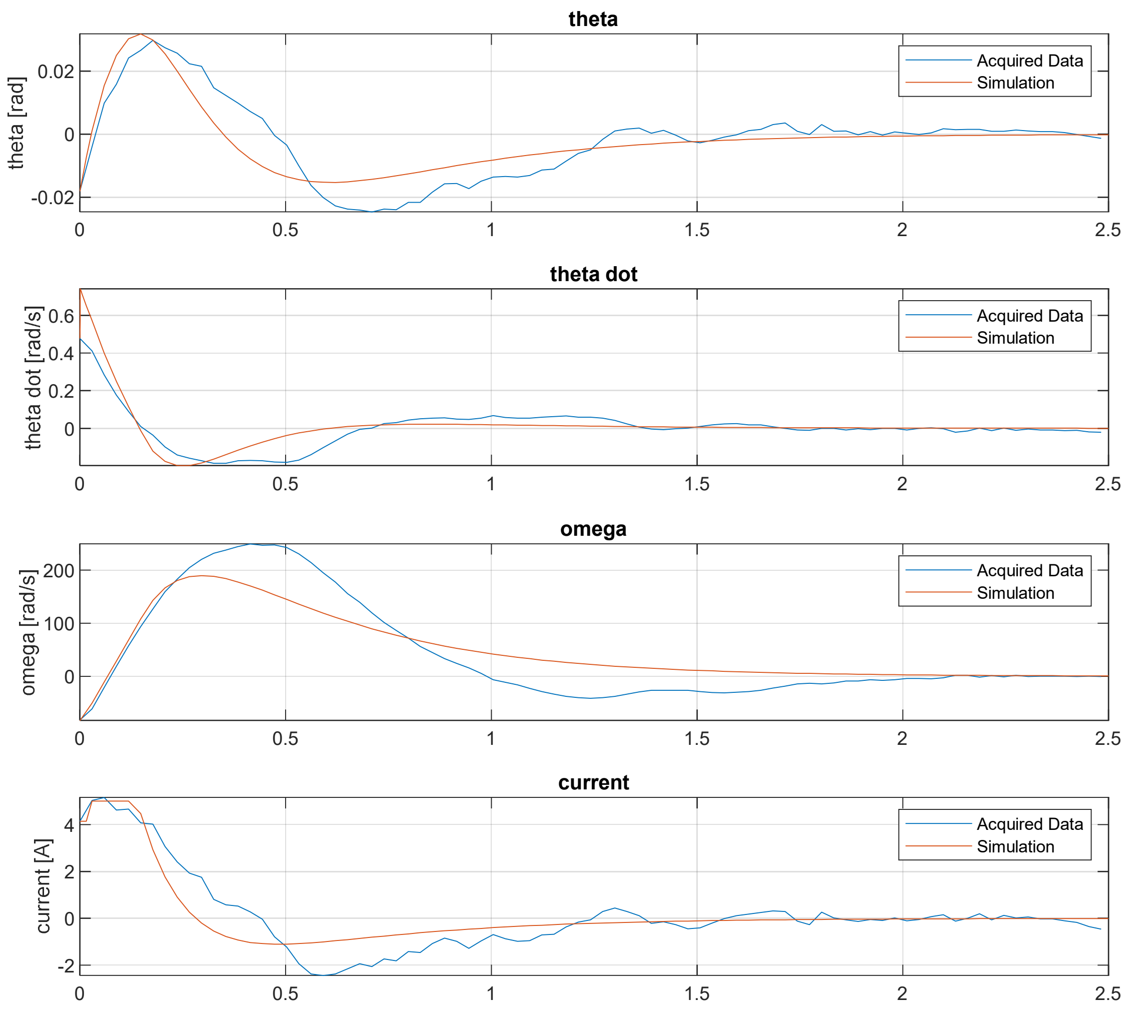


Figure 98 – Impulse Response Comparison, Test data vs Linear System Simulation.

11.3. Battery Duration Time Test

Starting with the battery fully charged, the cube was able to stabilize on its edge for 5 hours 10 minutes before battery went below 14V threshold, considered as discharged. Considering a battery capacity of 1300 mAh, the average current drawn was 252 mA, and considering and average voltage of 15V, the average power was 3.78 W.

12. Final Product Part List

Table 24 lists all the parts present in the final cube assembly.

	Name	Q	Part Number	Provider	U/P	Total
Motor Drive System	Motor 0,1	2	T-MOTOR MT4008 KV380	Amazon	37.99	75.98
	Motor 2	1	SunnySky V4008 KV380	SunnySky	59	59
	Encoder 0,1	2	AS5047P	Digikey	16.76	33.52
	Encoder 2	1	AS5147P	Mouser	15.75	15.75
	Motor Driver Board	2	ODrive V3.6	ODrive	137.9	275.8
Inertial Sensing	IMU	4	MPU6050	AliExpress	2.6	10.4
Digital System	Microcontroller	1	STM32 NUCLEO-F446RE	Mouser	15	15
	Breadboard Wires	1	Breadboard Jumper Wires Ribbon Cables	Amazon	8	8
	Pushbutton LED	1	612-SV7F23SS-6G1	Mouser	13.74	13.74
Battery System	Battery	1	Ovonic 14.8V 1300mAh 100C 4S LiPo Battery	Amazon	18	18
	Battery Straps	1	iFlight 5pcs RC LiPo Battery Straps 10x100mm	Amazon	7	7
	Battery Charger	1	Enegitech Battery Charger for 14.8V 4S LiPo	Amazon	25.99	25.99
	Step Down	1	Buck Converter 6-24V to 5V 1.5A Step-Down	Amazon	10	10
	Connectors	1	4pcs XT60 Plug Male Female Connector	Amazon	9	9
	Connectors	1	5 Pairs T Plug Connector Female and Male	Amazon	10	10
	Connectors	1	Vanka JST-XH 4S Connector Adapter	Amazon	10	10
	Rocker Switch SPDT	1	611-CM101J12S205QA	Mouser	1.05	1.05
Mechanical Assembly	Panels	6	3D Printed			
	Reaction Wheel	3	3D Printed			
	Arm	3	3D Printed			
	Screw	30	Countersunk M3X20			
	Screw	8	Countersunk M3X10			
	Screw	12	Cap M3X20			
	Screw	12	Cap M3X8			
	Screw	24	Cap M2.5X8			
	Nut	27	M3 Steel			
	Nut	23	M3 Nylon			
					Total:	598.23

Table 24 – Final Product Part List.

13. User Manual

13.1. Operation

Always keep fingers away from wheels.

1. Turn on battery switch. (See Figure 99).
2. Check if digital switch red LED is on (see Figure 100).
   a. If red LED does not turn on, the cube needs battery charging.
3. CALIBRATE: Press and release the digital button once, this will start encoder calibration procedure. Digital switch LED will stay red.
4. Wait for completion of the automatic calibration procedure, the cube will automatically:
   a. Perform a beep sound.
   b. Slowly spin wheels in one direction for about one revolution.
   c. Slowly spin wheels in the other direction for about one revolution.
   d. Wheels stop. Calibration procedure finished.
5. TURN ON BALANCING CONTROL SYSTEM: Press and release the digital button. The LED should change to green, if LED stays red, turn off battery switch and start again.
6. Cube is now ready to balance on any edge.
7. Place any edge on the table and manually position the cube close to the vertical balancing position
8. When the cube is close to the stable position, the control system will turn on and start balancing.
9. If the cube is far away from any stable position, the control system is turned on and all wheels spin down to zero.
10. Turn off cube after usage by turning the battery switch off.

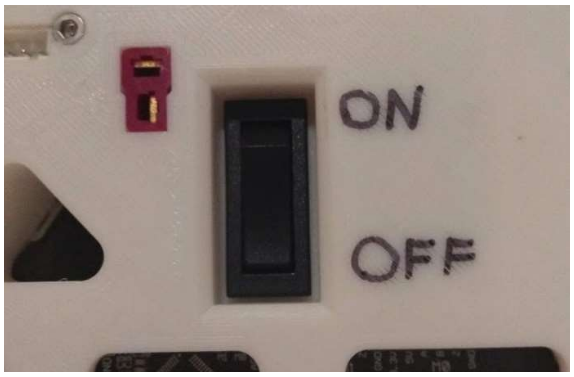


Figure 99 – Battery Switch.

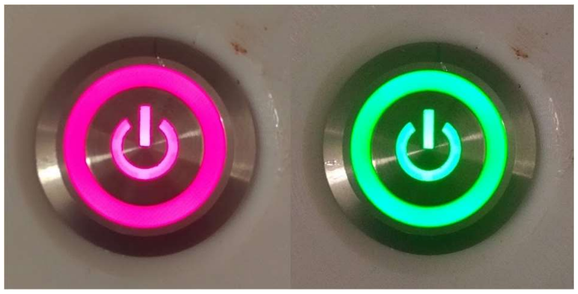


Figure 100 - Digital Switch – LED Modes.

13.2. Battery Charging

1. Set Energitech e430 battery charger to LiPo and 1A. (see Figure 101)
2. Connect the female-female JST-XH 5-pin cable to the Energitech e430 and the cube male ports.
3. Connect the dual-banana to T-plug cable to the Energitech e430 and the cube.
4. Figure 103 shows how these two cables should be connected.
5. Connect the Energitech e430 to mains voltage. (110V-240V)
6. Charge Status LED should turn solid red, this indicates battery is charging.
7. Wait for full charge. Charge Status LED turns green when charging complete.
8. Disconnect Energitech e430 from mains voltage.
9. Disconnect all cables from cube.
10. Cube is fully charged and ready to use.

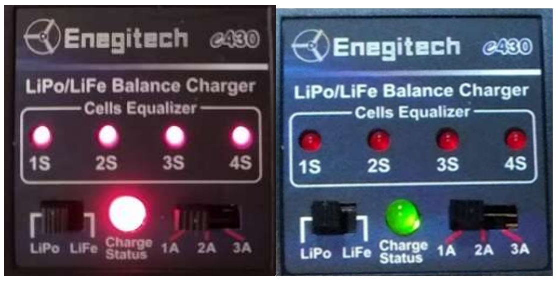


Figure 101 - Energitech e430. Charging (left). Charged (right).

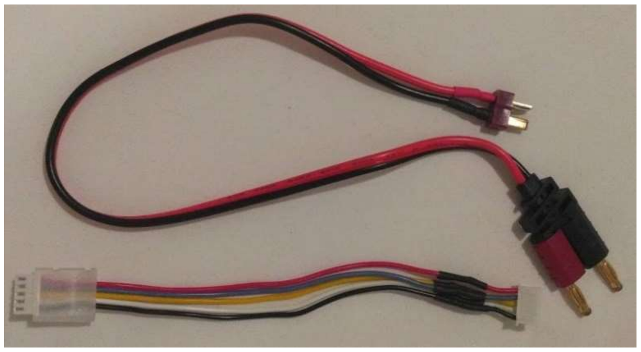


Figure 102 - Dual-Banana to T-plug (top). Female-Female JST-XH 5-pin (bottom).

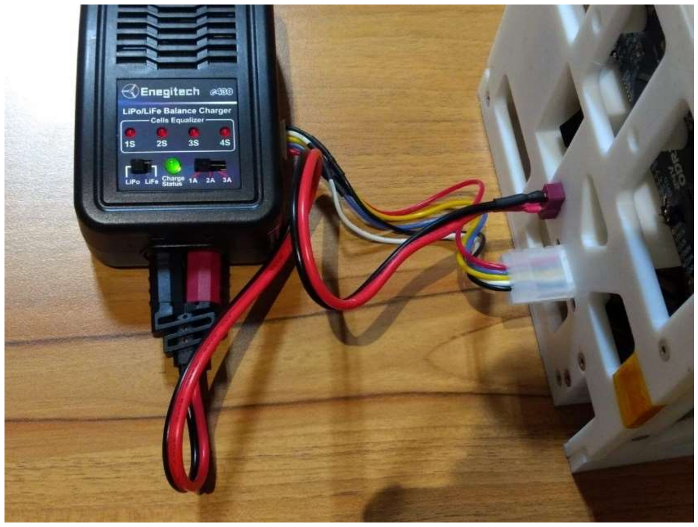


Figure 103 – Cube Charging Cable Connection.

Appendix A. – IMU Calibration MATLAB Code

```
%% Load Calibration Data
coordLetter = ['x' 'y' 'z'];
sign = ['+' '-'];
A = cell(3,2); % coord, sign
for i = 1:length(coordLetter)
    for j = 1:length(sign)
        disp(['A{' num2str(i) '}{' num2str(j) '} = load(''' coordLetter(i) sign(j) '.log'');' ])
        eval(['A{' num2str(i) '}{' num2str(j) '} = load(''' coordLetter(i) sign(j) '.log'');' ])
    end
end
%% Parse Data
IMU = cell(4,6,3,2); % ID, value, coord, sign
for coord = 1:3
    for sgn = 1:2
        data = A{coord}{sgn};
        for ID = 1:4
            for value = 1:6
                idx = (ID-1)*6+value;
                IMU{ID}{value}{coord}{sgn} = data(:,idx);
            end
        end
    end
end
%% Accelerometer Calibration
Cal = cell(4,6);
for ID = 1:4
    for value = 1:3
        pos = mean(IMU{ID}{value}{value}{1});
        neg = mean(IMU{ID}{value}{value}{2});
        Cal{ID}{value} = (pos+neg)/2;
    end
end
%% Gyroscope Calibration
gyroOffset = zeros(3,2);
for ID = 1:4
    for value = 4:6
        for coord = 1:3
            for sgn = 1:2
                gyroOffset(coord,sgn) = mean(IMU{ID}{value}{coord}{sgn});
            end
        end
        Cal{ID}{value} = mean(mean(gyroOffset));
    end
end
%% Print Calibration C code
disp('Calibration C Code:')
for ID = 1:4
    for value = 1:3
        disp(['IMU[' num2str(ID-1) '].Cal.A[' num2str(value-1) '] = ' num2str(Cal{ID}{value})
';'])
    end
    for value = 4:6
        disp(['IMU[' num2str(ID-1) '].Cal.W[' num2str(value-4) '] = ' num2str(Cal{ID}{value})
';'])
    end
    disp(' ')
end
```

Appendix B. – IMU Coordinates C Code Generation

```
% Load Table
IMU_Coordinates = readtable('IMU_Coord.xlsx')
IMU = table2cell(IMU_Coordinates);
% Generate C Code
for ID = 1:4
    for coord = 1:3
        disp(['IMU[' num2str(ID-1) '].r[' num2str(coord-1) '] = ' num2str(IMU{ID,coord+1}) ';']);
    end
end
```

Appendix C. – Stable Edge Selection C Code.

```
// Set control mode to zero, check if any axis is within control range, if so, set
control mode to 1 and save that edge
control_mode = 0;
// for every axis
for(int axis_i=0;axis_i<3;axis_i++){
        // for quadrant of the axis
        for(int quadrant_i=0;quadrant_i<4;quadrant_i++){
                // measure the angle with gravity of each stable direction
                Vector_angle_between(axis_quadrant_to_stableDirectionVec-
tor[axis_i][quadrant_i],g3,&angleWithGravity);
                // if angle within range
                if((angleWithGravity>0)&&(angleWithGravity<ControlAngleRange)){
                        // set control mode
                        control_mode = 1;

                        // Save edge number
                        quadrant = quadrant_i;
                        axis = axis_i;

                        // Select IMU ID
                        IMU_ID = axis_quadrant_to_IMU_ID[axis][quadrant];

                        // Force Loops Exit
                        quadrant_i=4;
                        axis_i=3;
                }
        }
}
```

Appendix D. – Stable Directions C Code Generation

```
%% Load IMU Location Table
IMU_Coor = readtable('IMU_Coord.xlsx');

%% Edges Names and Stable Directions
l = 155;%mm
edges_centered = zeros(12,3);
edgeID = cell(12,1);

axisName = ['Z' 'X' 'Y'];

plane_versors = [
    1,2;
    2,3;
    3,1];

quadrant_directions = [
    1,1;
    -1,1;
    -1,-1;
    1,-1];

for axis_i = 1:3
    versors = plane_versors(axis_i,:);
    for quad = 1:4
        edge = quad+4*(axis_i-1);
        % Name Edge
        edgeID{edge} = [axisName(axis_i) num2str(quad-1)];
        % Edge Stable Directions
        edges_centered(edge,versors) = quadrant_directions(quad,:);
    end
end

% Shift zero centered coordinates to CATIA model coordinates
edges = (edges_centered+ones(size(edges_centered)))*l/2;

%% Stable Edges Directions
count = 1;
for axis_i = 1:3
    for quad = 1:4
        for coord = 1:3
            disp(['axis_quadrant_to_stableDirectionVector[' num2str(axis_i-1) '][' num2str(quad-
1) '][' num2str(coord-1) '] = ' num2str(edges_centered(count,coord)) ';'])
        end
        count = count + 1;
    end
end
disp(' ');disp(' ')
```

Appendix E. – IMU Selection C Code Generation

```
%% IMU Select
d_vec = zeros(4,2);
d = zeros(4,1);
selectedIMU = zeros(12,6); %plane, quadrant, IMU, x,y,z
selectedIMUrot = zeros(12,5); %plane, quadrant, IMU, x,y

for axis_i = 1:3
    idx = plane_versors(axis_i,:);
    for quad = 1:4
        edge = quad+4*(axis_i-1);
        for ID = 1:4
            d_vec(ID,:)=IMU_Coor{ID,idx+1}-edges(edge,idx);
        end
        [d,min_idx] = min(sqrt(sum(d_vec.^2,2)));
        d3_vec = zeros(1,3);
        d3_vec(1,idx) = d_vec(min_idx,:);
        selectedIMU(edge,:) = [axis_i-1 quad-1 min_idx-1 d3_vec];
        selectedIMUrot(edge,1:3) = [axis_i-1 quad-1 min_idx-1];
        switch lower(axis_i)
            case 1
                temp = rotz(180-90*(quad-1))*d3_vec';
            case 2
                temp = rotx(180-90*(quad-1))*d3_vec';
            case 3
                temp = roty(180-90*(quad-1))*d3_vec';
        end
        selectedIMUrot(edge,4:5) = temp(idx);
    end
end

%% Edge IMU Select - C Code
M = selectedIMUrot;
for i=1:length(selectedIMUrot)
    disp(['axis_quadrant_to_IMU_ID[' num2str(M(i,1)) '][' num2str(M(i,2)) '].ID = '
num2str(M(i,3)) ';'])
end

for i=1:length(selectedIMUrot)
    disp(['axis_quadrant_to_IMU_ID[' num2str(M(i,1)) '][' num2str(M(i,2)) '].r[0] = '
num2str(M(i,4)) ';'])
    disp(['axis_quadrant_to_IMU_ID[' num2str(M(i,1)) '][' num2str(M(i,2)) '].r[1] = '
num2str(M(i,5)) ';'])
end
```

14. Bibliography

[1] M. Gajamohan, M. Merz, I. Thommen and R. D'Andrea, "The Cubli: A cube that can jump up and balance," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, 2012.

[2] M. F. Golnaraghi and B. C. Kuo, Automatic Control Systems, Wiley, 2009.

[3] T. Instruments, "Demystifying BLDC motor commutation: Trap, Sine, & FOC," 4 7 2021. [Online]. Available: https://www.ti.com/lit/ml/slyp711/slyp711.pdf.

[4] C. Z. a. D. Bian, "A PWM Control Algorithm for Eliminating Torque Ripple Caused by Stator Magnetic Field Jump of Brushless DC Motors," in *7th World Congress on Intelligent Control and Automation*, pp. 6547-6549, doi: , 2008.

[5] "Coefficient of friction, Rolling resistance and Aerodynamics," [Online]. Available: https://www.tribology-abc.com/abc/cof.htm. [Accessed 8 2 2021].

[6] K. Ogata, Discrete-Time Control Systems, Englewood Cliffs, N.J: Prentice Hall, 1995.

[7] G. Belascuen and N. Aguilar, "Design, Modeling and Control of a Reaction Wheel Balanced Inverted Pendulum," in *IEEE Biennial Congress of Argentina (ARGENCON)*, San Miguel de Tucumán, Argentina, 2018.