POLYTECH Nice Sophia – Département Robotique et Systèmes Autonomes

Arm control from a Matlab interface

Double Propeller Arm

Work supervised by Frédérique JUAN

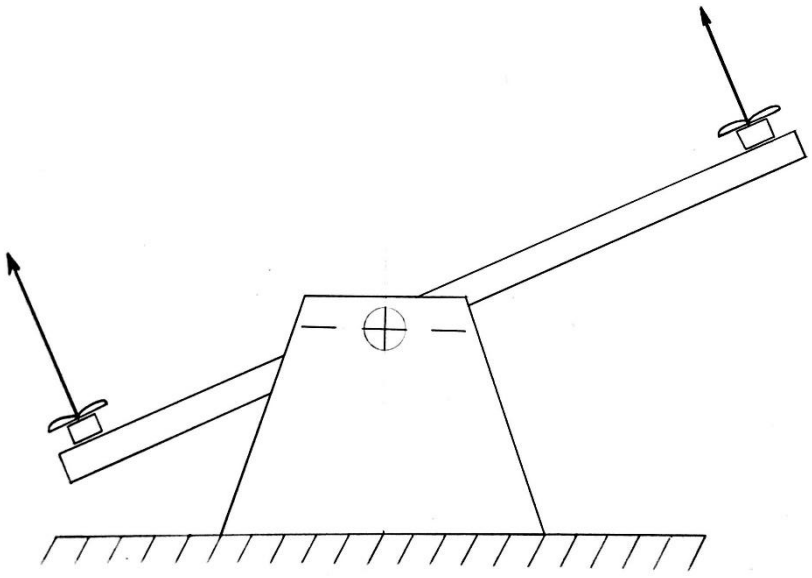


Table des matières

I-) Introduction and Project Overview 4
I.1 Context and Objectives 4
I.2 Project Contribution: MATLAB Interface 4
I.3 Project Overview 4
I.3.1 System Architecture 4
I.3.2 Challenges of Estimation 5
II-) Mathematical Modeling and Parameter Identification 6
II.1 Importance of Mathematical Modeling for Kalman Filter 6
II.2 Dynamic Analysis 6
II.2.1 System Description 6
II.2.2 Force Analysis 7
II.2.3 Fundamental Principle of Dynamics 7
II.3 State-Space Representation 7
II.4 Parameter Identification Using Linear Regression 8
II.4.1 Reformulation for Parameter Estimation 8
II.4.2 Matrix Formulation 8
II.4.3 Least Squares Solution 8
II.4.4 Some examples of code implementation 9
III Hardware Configuration and Development Workflow 11
III.1 Development Approach and Timeline 11
III.1.1 MATLAB Interface Development Status 11
III.1.2 STM32 Configuration and System Control 12
III.2 Challenges and Issues Encountered 13

I-) Introduction and Project Overview

I.1 Context and Objectives

This project focuses on the stabilization of a mechanical arm equipped with two propellers, a test bench initially designed by Mr. Ducard. The system consists of an articulated arm mounted on a pivot, actuated by two brushless motors and equipped with an inertial measurement unit (IMU) comprising a gyroscope and an accelerometer. The primary objective is to maintain the arm in a horizontal position using advanced control strategies coupled with state estimation algorithms.
This work builds upon and deepens developments carried out by Mr. Julien, who implemented several filters (complementary and Kalman) and controllers (PID, cascade, LQR) on an STM32F411 board. Our contribution aims to **improve and take ownership of this system** by developing a communication interface with MATLAB. This allows us to move beyond exclusive reliance on the C language and facilitates testing, tuning, and performance analysis.

I.2 Project Contribution: MATLAB Interface

To make the system more accessible and flexible, we want to develop a communication interface between the STM32 board and MATLAB. This interface enables:

- Real-time data visualization and logging,
- On-the-fly modification of filter and controller parameters,
- Result export for in-depth analysis.

This evolution will complement the existing Rust graphical interface and will provide a flexible alternative for algorithm prototyping and validation.

I.3 Project Overview

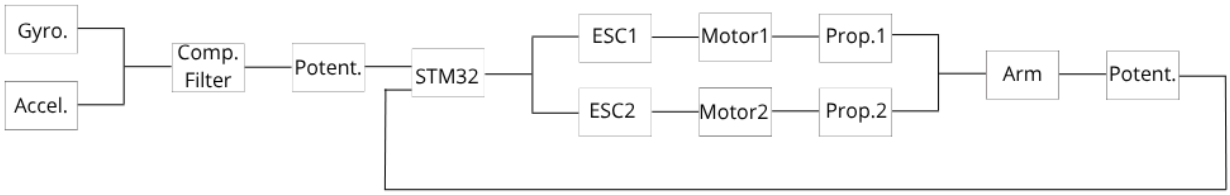
I.3.1 System Architecture

The test bench is composed of the following elements:

- A mechanical arm with free rotation around a single axis,
- Two brushless motors **XMA2830A-14C** equipped with propellers, controlled by **ESCs** (Electronic Speed Controllers) **30A**,
- An **LSM303DLHC** inertial measurement unit (accelerometer + gyroscope) at the beginning and **BNO055** at the end,

- An STM32F411 board then **STM32 F407 Discovery** board handling data acquisition and actuator control.

The system's functional diagram is shown below:



I.3.2 Challenges of Estimation

Raw measurements from the accelerometer and gyroscope are noisy. Sensor fusion is necessary to obtain a precise and stable estimate of the angle. Two approaches were compared:

- The **Complementary Filter**, simple and computationally efficient.
- The **Kalman Filter**, more complex but offering better precision and estimation of the gyroscope bias.

This initial phase of the project thus involved understanding, reproducing, and extending the previous work, with a emphasis on integrating an interactive development environment with MATLAB.

II-) Mathematical Modeling and Parameter Identification

II.1 Importance of Mathematical Modeling for Kalman Filter

A precise mathematical model is fundamental to the performance of the Kalman filter. While the Kalman filter is theoretically optimal for linear systems with Gaussian noise, its practical effectiveness heavily depends on the accuracy of the system model. An inaccurate model can lead to:

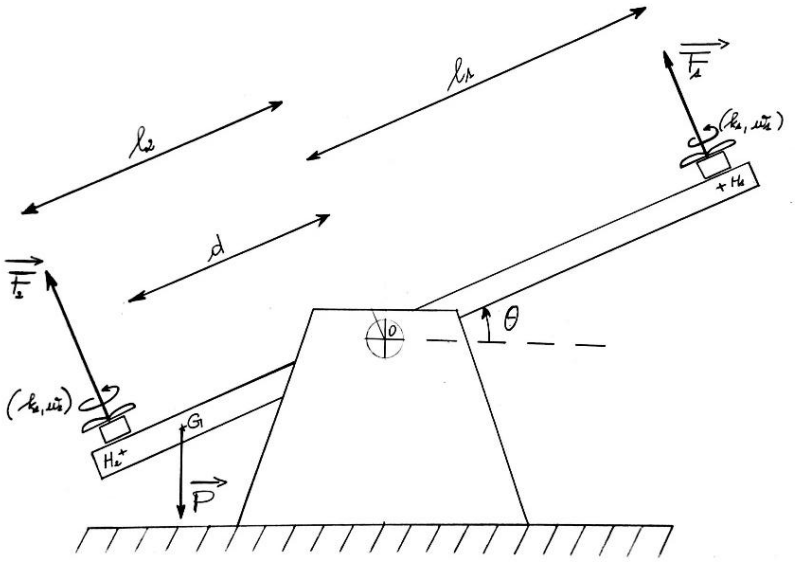
- Biased estimates due to unmodeled dynamics
- Poor noise rejection from incorrect process noise characterization
- Filter divergence when the model fails to capture true system behavior
- Suboptimal performance despite theoretical optimality

For our double-propeller arm system, developing a comprehensive dynamic model ensures that the Kalman filter can effectively fuse sensor data and provide reliable state estimates for control purposes.

II.2 Dynamic Analysis

II.2.1 System Description

The system under study comprises the arm, motors, and propellers, with total mass m and center of mass located at point G. The analysis is performed in a Galilean reference frame (O, x_b, y_b, z_b)..



$J_z \equiv System\ moment\ of\ inertia$

$g \equiv Gravity$

$f_r \equiv$ friction coefficient

II.2.2 Force Analysis

The forces acting on the system are:

- Weight force: $\vec{P}_G = mg(\cos\ \theta\, \vec{x}_b - \sin\ \theta\, \vec{y}_b)$
- Thrust from motor 1: $\vec{F}_{H_1} = F_{H_1}\, \vec{y}_b$
- Thrust from motor 2: $\vec{F}_{H_2} = F_{H_2}\, \vec{y}_b$
- Viscous friction torque: $\vec{F}_r = -f_r\dot{\theta}\, \vec{z}_b$

II.2.3 Fundamental Principle of Dynamics

Applying the rotational form of the Fundamental Principle of Dynamics:

$$\sum_i (\vec{M}_O)(F_i) = J_z\ddot{\theta}\, \vec{z}_b$$

Substituting the moments of each force gives:

$$-mgd\sin\ \theta + F_{H_1}l_1 - F_{H_2}l_2 - f_r\dot{\theta} = J_z\ddot{\theta}$$

Thus, the nonlinear equation governing the system dynamics is:

$$\boxed{\ddot{\theta} = \frac{1}{J_z}(-mgd\sin\ \theta + F_{H_1}l_1 - F_{H_2}l_2 - f_r\dot{\theta})}$$

II.3 State-Space Representation

To design estimators and controllers, the system is linearized around an operating point $(\theta_{op}, \dot{\theta}_{op}, F_{H1,op}, F_{H2,op})$.

We define the state and input variations: $X = \begin{pmatrix}\Delta\theta \\ \Delta\dot{\theta}\end{pmatrix}, U = \begin{pmatrix}\Delta F_{H_1} \\ \Delta F_{H_2}\end{pmatrix}$

The linearized dynamics becomes:

$$\dot{X} = \underbrace{\begin{pmatrix} 0 & 1 \\ \dfrac{-\text{mgdcos}\,\theta_{op}}{J_z} & -\dfrac{f_r}{J_z}\end{pmatrix}}_{\text{F}} X + \underbrace{\begin{pmatrix} 0 & 0 \\ \dfrac{l_1}{J_z} & -\dfrac{l_2}{J_z}\end{pmatrix}}_{\text{G}} U$$

This linear state-space model is essential for Kalman filter implementation.

II.4 Parameter Identification Using Linear Regression

II.4.1 Reformulation for Parameter Estimation

The nonlinear dynamic equation can be rearranged as:

$$J_z\ddot{\theta} = -mgd\sin\,\theta + F_{H_1}l_1 - F_{H_2}l_2 - f_r\dot{\theta}$$

We want to estimate the parameter vector: $\hat{\pi} = \begin{pmatrix} J_z \\ f_r \\ md \end{pmatrix}$

II.4.2 Matrix Formulation

For n collected samples of angle, velocity, acceleration, and thrusts, we construct the regression matrix:

$$J_z\begin{pmatrix}\ddot{\theta}_1\\ \vdots \\ \ddot{\theta}_n\end{pmatrix} = l_1 \cdot \begin{pmatrix}F_{H1,1}\\ \vdots \\ F_{H1,n}\end{pmatrix} - l_2 \cdot \begin{pmatrix}F_{H2,1}\\ \vdots \\ F_{H2,n}\end{pmatrix} - f_r \cdot \begin{pmatrix}\dot{\theta}_1\\ \vdots \\ \dot{\theta}_n\end{pmatrix} - md\begin{pmatrix}g\cdot\sin(\theta_1)\\ \vdots \\ g\cdot\sin(\theta_n)\end{pmatrix}$$

This can be written in compact matrix form:

$$\Rightarrow \underbrace{\begin{pmatrix}\ddot{\theta}_1 & \dot{\theta}_1 & g\cdot\sin(\theta_1)\\ \vdots & \vdots & \vdots \\ \ddot{\theta}_n & \dot{\theta}_n & g\cdot\sin(\theta_n)\end{pmatrix}}_{=\emptyset} \cdot \underbrace{\begin{pmatrix}J_z\\ f_r\\ md\end{pmatrix}}_{=\hat{\pi}} = \underbrace{\begin{pmatrix}F_{H1,1} & F_{H2,1}\\ \vdots & \vdots \\ F_{H1,n} & F_{H2,n}\end{pmatrix}}_{=U} \cdot \underbrace{\begin{pmatrix}l_1\\ -l_2\end{pmatrix}}_{=l}$$

II.4.3 Least Squares Solution

The parameter estimates are obtained using the least squares method:

$$\Phi \cdot \hat{\pi} = U \cdot l \Rightarrow \boxed{\hat{\pi} = (\Phi^T \cdot \Phi)^{-1} \cdot \Phi^T \cdot U \cdot l}$$

To implement this identification method, we must:

1. **Collect n samples** of the input force signals F_{H1} and F_{H2}
2. **Measure corresponding angles** θ, angular velocities $\dot{\theta}$, and angular accelerations $\ddot{\theta}$
3. **Construct the regression matrix** Φ and input matrix U
4. **Compute the parameter estimates** using the least squares formula

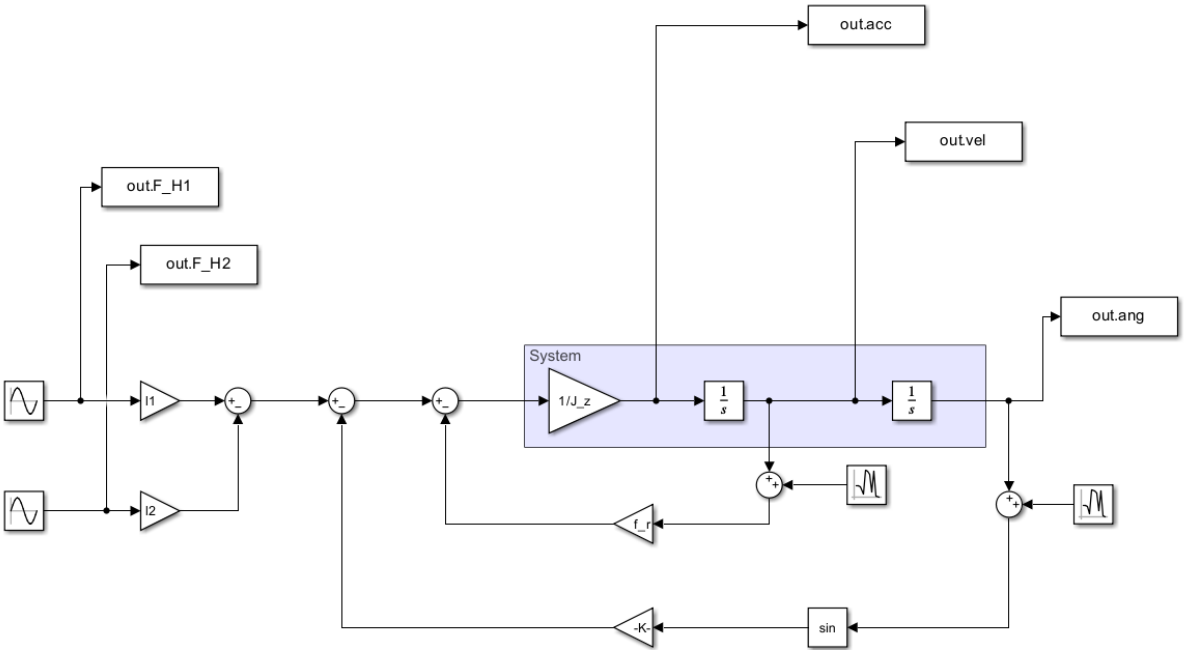
All calculations and derivations presented in this section were personally developed and verified to ensure a comprehensive understanding of the system dynamics and to provide a solid foundation for the Kalman filter implementation.

This parameter identification approach provides accurate estimates for J_z, f_r, and md, which are crucial for both simulation accuracy and filter design performance.

II.4.4 Some examples of code implementation

II.4.4.1 Implementation of the Linear Regression Model

Synchronized samples from each signal are collected, and the previously presented regression is applied. The important parameters are obtained at the end."



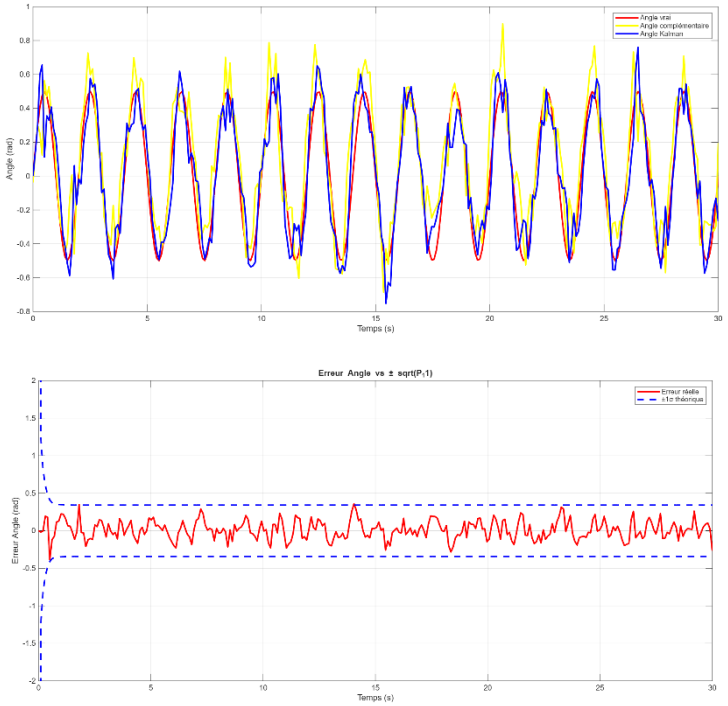
The estimated parameters are consistent, since values very close to those implemented in the model are recovered. Configurations with and without noise were also tested, and the results are reported in the following table.

True values $\hat{\pi} = \begin{pmatrix} J_z \\ f_r \\ md \end{pmatrix}$	**Without noise in the sensors**	**noisy sensors: Gaussian noise with mean equal to 1**	**Gaussian noise with zero mean**
$\begin{pmatrix} 0.0576\ m^2kg \\ 0.0480\ s^{-1}kg \\ 0.2112\ kgm \end{pmatrix}$	$\begin{pmatrix} 0.0576\ m^2kg \\ 0.0480\ s^{-1}kg \\ 0.2112\ kgm \end{pmatrix}$	$\begin{pmatrix} 0.0193\ m^2kg \\ 0.0531\ s^{-1}kg \\ 0.0386\ kgm \end{pmatrix}$	$\begin{pmatrix} 0.0283\ m^2kg \\ 0.0449\ s^{-1}kg \\ 0.0788\ kgm \end{pmatrix}$

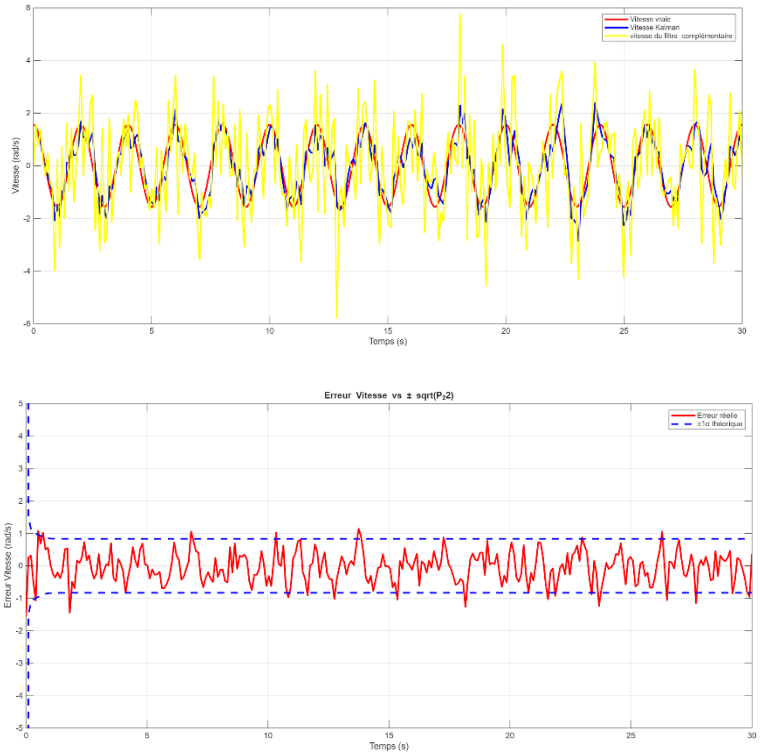
Result of the ***Least_square_double_propeller_arm.m*** Code.

II.4.4.2 Comparison of Estimators

Then, the estimated output obtained using a complementary filter is compared with that obtained using a Kalman filter, still in simulation, in order to determine which method provides better results.



In terms of angle estimation, both filters provide comparable performance. However, when estimating velocity, the Kalman filter shows superior results, mainly because it is more robust to noise-induced drift.



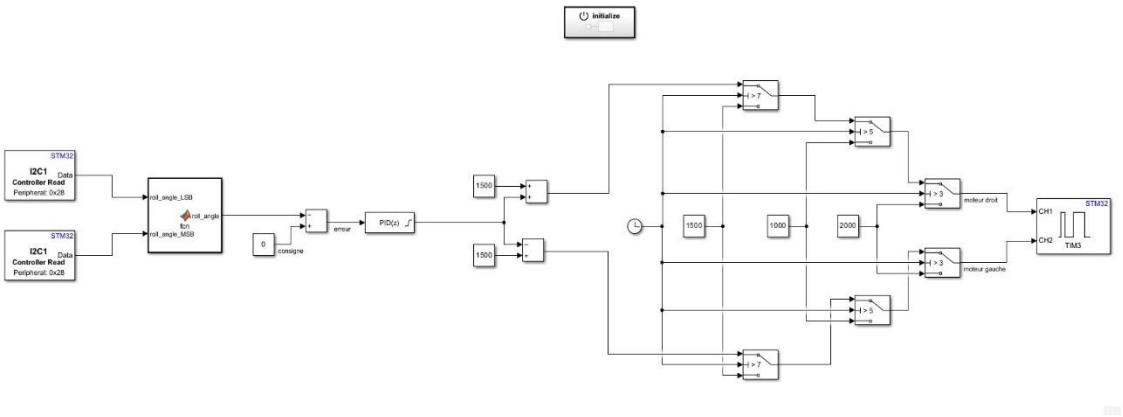
III Hardware Configuration and Development Workflow

III.1 Development Approach and Timeline

III.1.1 MATLAB Interface Development Status

The MATLAB interface for real-time data acquisition and parameter tuning has been successfully implemented. The system architecture was complete, and all core functionalities were fully operational. The interface was demonstrated during the final project presentation at the end of the semester.

The system is modeled in Simulink and consists of the following functional blocks:



- **Roll angle acquisition via I2C:**
  Two Simulink blocks are configured to read the LSB and MSB registers of the inertial measurement unit (IMU) connected via I2C at address 0x28. These data are combined to reconstruct the roll angle of the arm:

$$\text{roll\_angle} = \text{LSB} + 256 \cdot \text{MSB}$$

- **Error calculation:**
  The reference is set to 0° (arm perfectly horizontal). The error is computed as the difference between the reference and the measured angle:

$$\text{error} = Reference - \text{estimated\_angle}$$

- **Discrete PID control:**
  The error signal is processed by a digital PID controller (PID(z) block), which generates proportional, integral, and derivative commands. This controller dynamically corrects the arm's position based on the measured deviations.

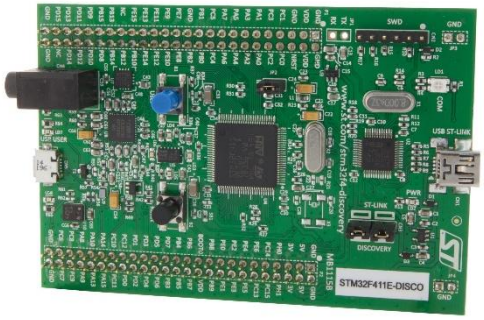
- **PWM signal generation:**
  The PID output is compared to thresholds and routed through Switch blocks to generate PWM signals ranging from 1000 µs to 2000 µs, centered at 1500 µs

(neutral position). These signals are then sent to the motors via channels CH1 and CH2 of the TIM3 timer on the STM32.

- **Motor control:**
  The right and left motor blocks receive the PWM signals and drive the arm actuators. The goal is to compensate for angle variations and maintain horizontal alignment.

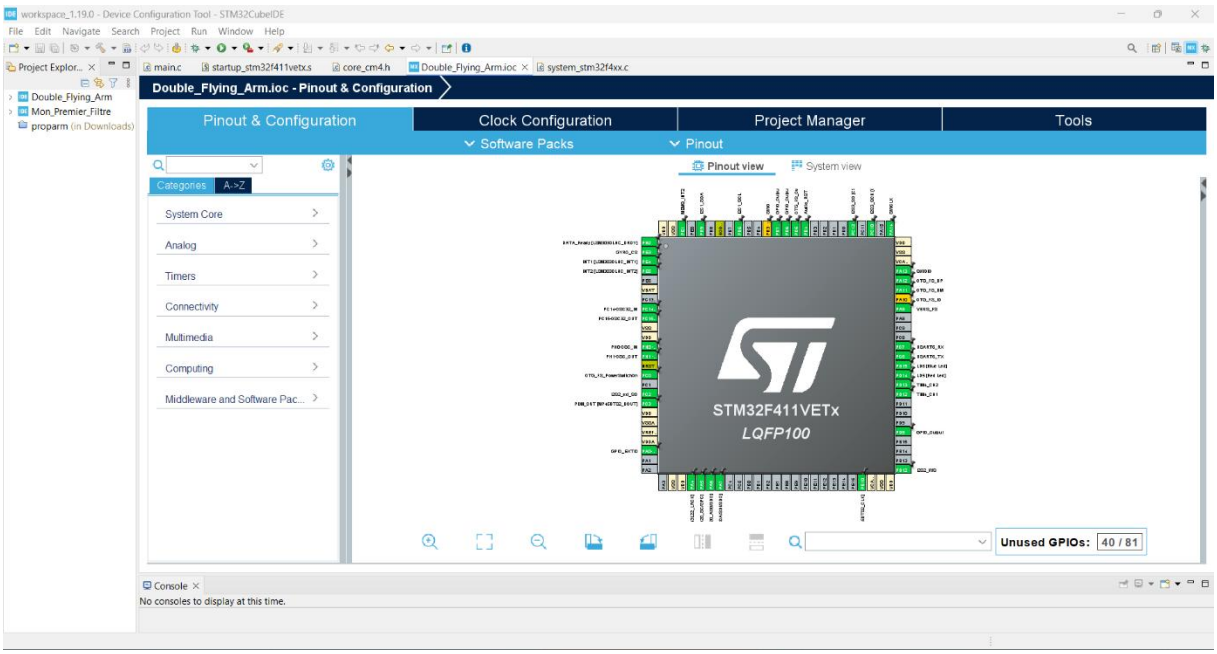
III.1.2 STM32 Configuration and System Control

To maintain project momentum, we are currently using STM32CubeIDE's direct development environment for system control and monitoring. This approach allows us to:



- Validate algorithms in real-time
- Collect experimental data for offline analysis
- Test control strategies without external tool dependencies
- Debug and optimize system performance efficiently

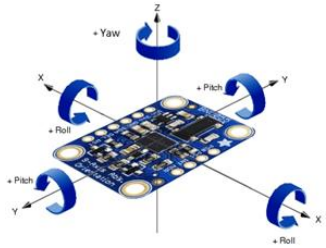
Although the final system architecture will be decentralized and primarily controlled via MATLAB, an initial configuration of the STM32 board through STM32CubeIDE is still required. In fact, we switched from the STM32F4011 to the STM32F407 Discovery board, as we encountered significant difficulties interfacing the former, even for simple tasks such as blinking an LED.



III.2 Challenges and Issues Encountered

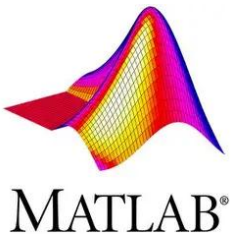
During the project, several challenges were encountered:

- **Hardware changes:** The original STM32F4011 board was replaced with the STM32F407 Discovery board. However, the new board did not include a gyroscope, so we had to integrate an independent BNO055 IMU for orientation measurements. Additionally, one of our ESCs malfunctioned and had to be replaced at the last minute.



- **Use of IMU internal estimation:**
  Unlike the embedded solution initially considered, the BNO055 IMU directly provides an angle estimation through its internal sensor fusion algorithm. Consequently, in the final stage of the project, this built-in angle estimation was directly used, without relying on the estimators (complementary and Kalman filters) previously developed.

- **Software and debugging challenges:** There was no team member specialized in both MATLAB and STM32 simultaneously. This was particularly problematic because MATLAB does not provide direct logs for C code errors. As a result, debugging took significant time and effort, slowing down progress.



- **Organizational limitations:** The lack of direct support for embedded C debugging in MATLAB, combined with the absence of a dedicated specialist, made it difficult to quickly resolve issues and validate system behavior.