

NGUYỄN ĐỨC NAM BÌNH	<p style="text-align: center;"><b>BỘ CÔNG THƯƠNG</b></p> <p style="text-align: center;"><b>TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI</b></p> <p style="text-align: center;">_____</p> <p style="text-align: center;"> _____</p>
	<p style="text-align: center;"><b>ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC CNKT ĐIỆN TỬ VIỄN THÔNG</b></p>
	<p style="text-align: center;"><b>ĐỀ TÀI: THIẾT KẾ MÔ HÌNH MÁY BAY KHÔNG NGƯỜI LÁI SỬ DỤNG STM32</b></p>
CNKT ĐIỆN TỬ - VIỄN THÔNG	<p style="text-align: right;"><b>CBHD: Ts. Phạm Xuân Thành</b></p> <p style="text-align: right;"><b>SVTH : Nguyễn Đức Nam Bình</b></p> <p style="text-align: right;"><b>Mã sinh viên: 2018600020</b></p> <p style="text-align: right;"><b>Lớp: 2018DHDTT01</b></p>
	<p style="text-align: right;"><b>Hà Nội – Năm 2022</b></p>



## MỤC LỤC

<b>MỤC LỤC .....</b>	<b>I</b>
<b>DANH MỤC TỪ VIẾT TẮT .....</b>	<b>III</b>
<b>DANH MỤC HÌNH ẢNH .....</b>	<b>IV</b>
<b>LỜI CAM ĐOAN .....</b>	<b>VI</b>
<b>LỜI CẢM ƠN .....</b>	<b>VII</b>
<b>LỜI NÓI ĐẦU .....</b>	<b>VIII</b>
Lí do chọn đề tài .....	viii
Đối tượng nghiên cứu.....	viii
Mục đích nghiên cứu .....	viii
<b>CHƯƠNG 1. TỔNG QUAN VỀ THIẾT KẾ MÔ HÌNH MÁY BAY KHÔNG NGƯỜI LÁI SỬ DỤNG STM32 .....</b>	<b>1</b>
1.1. Giới thiệu chung về đề tài .....	1
1.1.1. Đặt vấn đề .....	1
1.1.2. Đối tượng nghiên cứu – mục tiêu nghiên cứu .....	1
1.2. Nguyên lí hoạt động của UAV .....	2
1.2.1. Động lực học [1].....	2
1.2.2. Động học [1].....	3
1.2.3. Lý thuyết bộ điều khiển PID.....	4
<b>CHƯƠNG 2. THIẾT KẾ MÔ HÌNH.....</b>	<b>6</b>
2.1. Phần cứng .....	6
2.1.1. Yêu cầu thiết kế .....	6
2.1.2. Pin lipo.....	6
2.1.3. Động cơ BLDC.....	7
2.1.4. Bộ điều khiển ESC .....	7
2.1.5. Cánh quạt .....	8
2.1.6. Bộ tay cầm điều khiển MC6C và rx MC7RB.....	8
2.1.7. Khung uav.....	9
2.1.8. Cảm biến MPU6050 .....	10
2.1.9. Kit STM32F103C8 [2] .....	11

2.1.10. Mạch nguồn hạ áp DC LM2596 [3] .....	13
2.1.11. Sơ đồ khói ghép nối phần cứng .....	14
2.1.12. Ghép nối phần cứng theo sơ đồ khói .....	15
2.2. Phần mềm .....	15
2.2.1. Các phần mềm sử dụng trong đê tài .....	15
2.2.2. Thiết kế chương trình điều khiển.....	17
<b>CHƯƠNG 3. KẾT QUẢ .....</b>	<b>28</b>
3.1. Kết quả đạt được.....	28
3.2. Kết quả chưa đạt được và các vấn đề gấp phải .....	28
3.3. Hướng phát triển của đê tài .....	29
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>30</b>
<b>PHỤ LỤC .....</b>	<b>31</b>
Code đồ án.....	31

## **DANH MỤC TỪ VIỆT TẮT**

Từ viết tắt	Từ đầy đủ	Ý nghĩa
BLDC	Brushless DC motor	Động cơ không chổi than
ESC	Electronic Speed Controller	Bộ điều khiển tốc độ động cơ (không chổi than)
PID	proportional–integral–derivative controller	Bộ điều khiển tỉ lệ, tích phân, vi phân
PWM	Pulse-width modulation	Điều chế độ rộng xung
UAV, uav	unmanned aerial vehicle	Thiết bị bay không người lái

## DANH MỤC HÌNH ẢNH

Hình ảnh 1.1: Quadcopter.....	1
Hình ảnh 1.2: Mô hình động lực học.....	2
Hình ảnh 1.3: Hướng chuyển động phụ thuộc chiều quay và tốc độ của từng cặp động cơ	3
Hình ảnh 1.4: Động học của uav .....	3
Hình ảnh 2.1: Pin Li-Po 3s 11.1V .....	6
Hình ảnh 2.2: Động cơ không chổi than BLDC.....	7
Hình ảnh 2.3: ESC điều khiển BLDC .....	7
Hình ảnh 2.4: Cánh quạt.....	8
Hình ảnh 2.5: Tay điều khiển và bộ thu tín hiệu đi kèm .....	9
Hình ảnh 2.6: Khung quacopter loại F450 .....	9
Hình ảnh 2.7: Module cảm biến GY-87 tích hợp MPU6050 .....	11
Hình ảnh 2.8: Kit STM32F103C8T6.....	11
Hình ảnh 2.9: Sơ đồ chân STM32F103C8 .....	12
Hình ảnh 2.10: Sơ đồ khói vi điều khiển STM32F103C8T6 .....	13
Hình ảnh 2.11: Mạch nguồn hạ áp DC LM2596 .....	13
Hình ảnh 2.12: Sơ đồ khói phần cứng .....	14
Hình ảnh 2.13: UAV sau khi hoàn tất lắp ráp phần cứng .....	15
Hình ảnh 2.14: Giao diện phần mềm STM32CubeIDE .....	16
Hình ảnh 2.15: Giao diện phần mềm STMStudio .....	16
Hình ảnh 2.17: Sử dụng 4 kênh điều khiển từ tay điều khiển tới module Rx .....	17
Hình ảnh 2.18: Thông số cài đặt đồng hồ cho STM32.....	18
Hình ảnh 2.19: Cài đặt chế độ input capture .....	19
Hình ảnh 2.20: Khai báo biến để chụp tín hiệu điều khiển .....	19
Hình ảnh 2.22: Cài đặt thông số cho I2C .....	20
Hình ảnh 2.23: Khai báo địa chỉ của thanh ghi trên cảm biến [4] [5] .....	20
Hình ảnh 2.25: Thông số của bộ lọc thông thấp tích hợp theo từng cấp độ [5] .....	21
Hình ảnh 2.26: Các giá trị gia tốc và tốc độ góc của từng cấp độ lọc tương ứng [6].....	21
Hình ảnh 2.27: Hàm đọc giá trị cảm biến.....	22
Hình ảnh 2.29: Bộ lọc thông thấp Butterworth với tần số cắt [7] .....	23
Hình ảnh 2.30: Mã code đọc giá trị góc lệch so với gốc .....	24
Hình ảnh 2.31: Thuật toán PID.....	26

Hình ảnh 2.32: Câu hình chân điều khiển động cơ qua chế độ PWM Generation.....	27
Hình ảnh 3.1: Đọc giá trị các biến và biểu đồ theo thời gian thực giá trị điều khiển động cơ .....	28
Hình ảnh 3.2: Mô phỏng khí động học và vận tốc dài các điểm trên cánh [8] .....	29
Hình ảnh 3.3: Gia tốc đo được không qua bộ lọc Kalman và đã qua bộ lọc Kalman .....	29

## LỜI CAM ĐOAN

Để hoàn thành đề tài này em có tham khảo các tài liệu có liên quan đến cảm biến mpu6050, các tài liệu liên quan tới chủ đề thiết bị không người lái.

Em xin cam đoan đề tài này là do em thực hiện, các số liệu và kết quả của đề tài này là trung thực. Mọi sự giúp đỡ cho báo cáo này đã được cảm ơn và các thông tin trích dẫn trong bài báo cáo đã được ghi rõ nguồn gốc.

Hà Nội, tháng 12 năm 2022

**Sinh viên**

**Nguyễn Đức Nam Bình**

## LỜI CẢM ƠN

Để hoàn thành đồ án tốt nghiệp này, em xin chân thành cảm ơn tới sự chỉ dẫn tận tình của thầy Phạm Xuân Thành, đồng thời cũng là giảng viên hướng dẫn cho đồ án tốt nghiệp của em. Em xin cảm ơn những góp ý chuyên môn và sự giúp đỡ về mặt công cụ của các thầy trong khoa Điện Tử.

Em xin chân thành cảm ơn!

Hà Nội, ngày 25 tháng 12 năm 2022

Sinh viên

Bình

Nguyễn Đức Nam Bình

## LỜI NÓI ĐẦU

### Lí do chọn đề tài

Hiện nay máy bay không người lái đang trở lên phổ biến trong nhiều lĩnh vực của cuộc sống. Chẳng hạn như: chụp ảnh trên không, giám sát trên cao, bảo vệ rừng, cứu nạn, cứu hộ, thăm dò,... Ngoài ra các thiết bị không người lái còn được sử dụng trong quân sự. Không nằm ngoài xu thế đó, nhu cầu sử dụng thiết bị bay không người lái trong nước cũng tăng cao, đi kèm với đó là nhu cầu nghiên cứu và phát triển, làm chủ được công nghệ chế tạo và điều khiển.

Dựa trên lí do trên, em trọng đề tài: “Thiết kế mô hình máy bay không người lái sử dụng STM32“

### Đối tượng nghiên cứu

- Cảm biến MPU 6050
- Máy bay không người lái và thuật toán điều khiển

### Mục đích nghiên cứu

- Thiết kế mô hình máy bay không người lái
- Sử dụng thuật toán PID để cân bằng mô hình
- Đọc dữ liệu và xử lý nhiễu từ cảm biến bằng các bộ lọc

Với đề tài: “Thiết kế mô hình máy bay không người lái sử dụng stm32” có bối cảnh như sau:

Chương 1: Tổng quan về thiết kế mô hình máy bay không người lái sử dụng stm32.

Chương 2: Thiết kế mô hình.

Chương 3: Kết quả.

## **CHƯƠNG 1. TỔNG QUAN VỀ THIẾT KẾ MÔ HÌNH MÁY BAY KHÔNG NGƯỜI LÁI SỬ DỤNG STM32**

### **1.1. Giới thiệu chung về đề tài**

#### **1.1.1. Đặt vấn đề**

Việc tiếp cận địa điểm mà các phương tiện cơ giới hoặc con người khó có thể đến được do điều kiện và tính chất môi trường để thực hiện các nhiệm vụ cứu nạn, cứu hộ, dân dụng hoặc quân sự (theo dõi mục tiêu, tấn công...). Từ tình hình trên, các thiết bị bay không người lái dần ra đời mang theo nhiều tính năng ngày càng ưu việt.

#### **1.1.2. Đối tượng nghiên cứu – mục tiêu nghiên cứu**

Mô hình của đề tài này được thiết kế dựa theo Quadcopter với 4 cánh quạt ở 4 góc của khung dạng dấu cộng (để ngắn gọn thì từ đây, “Quadcopter” sẽ được gọi chung là “UAV” hoặc “uav”, tên gọi chỉ có tác dụng thay thế trong phạm vi báo cáo này).

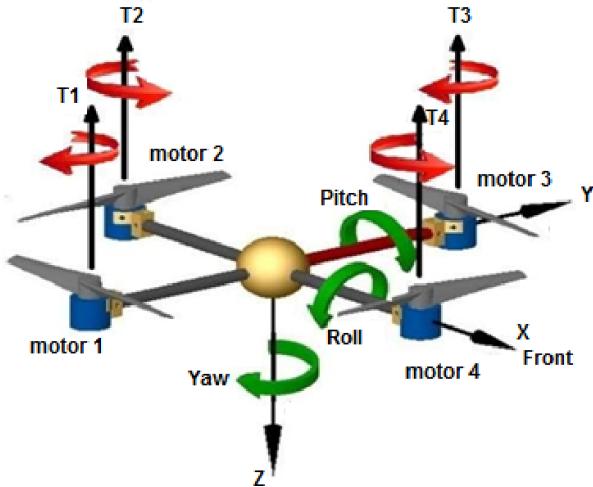


**Hình ảnh 1.1: Quadcopter**

Mô hình bay hoạt động dựa trên nguyên lý cân bằng góc nghiêng của từng cặp động cơ đặt đối diện nhau. Vấn đề đặt ra là làm thế nào để điều khiển bốn động cơ giúp cho máy bay có thể cân bằng từng trực, kết hợp cân bằng các trực với nhau, triệt tiêu quán tính xoay tròn và điều khiển UAV di chuyển ổn định. Các yếu tố quan trọng để xử lý vấn đề đặt ra là việc đọc các giá trị từ cảm biến phải chính xác, hạn chế sai số do nhiễu và thuật toán cân bằng sử dụng PID.

## 1.2. Nguyên lí hoạt động của UAV

### 1.2.1. Động lực học [1]



**Hình ảnh 1.2: Mô hình động lực học**

**Throttle/Hover:** chuyển động lên, xuống của uav theo độ cao

- Nếu cả 4 động cơ đẩy đều chạy với tốc độ bình thường thì uav sẽ di chuyển xuống
- Nếu cả bốn động cơ đẩy chạy với tốc độ cao hơn, thì uav sẽ di chuyển lên

**Pitch:** chuyển động của uav quanh trục ngang (tiến hoặc lùi)

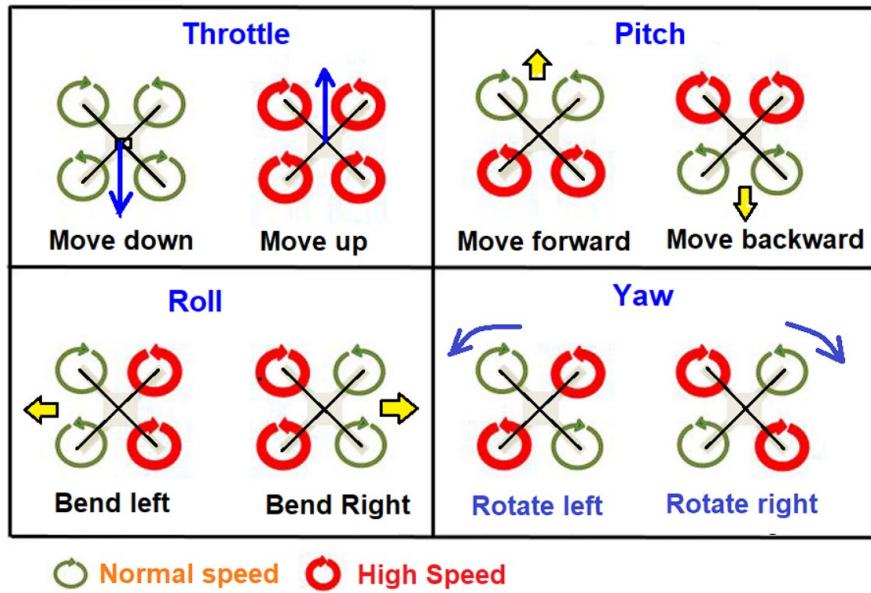
- Nếu 2 cánh quạt phía sau chạy với tốc độ cao thì drone sẽ di chuyển theo hướng tiến về trước
- Nếu hai cánh quạt phía trước chạy với tốc độ cao, thì máy bay không người lái sẽ di chuyển theo hướng lùi về sau

**Roll :** chuyển động của uav quanh trục dọc

- Nếu hai cánh quạt bên phải chạy với tốc độ cao, thì uav sẽ di chuyển theo hướng trái
- Nếu 2 cánh quạt bên trái chạy với tốc độ cao thì uav sẽ di chuyển theo hướng ngược lại

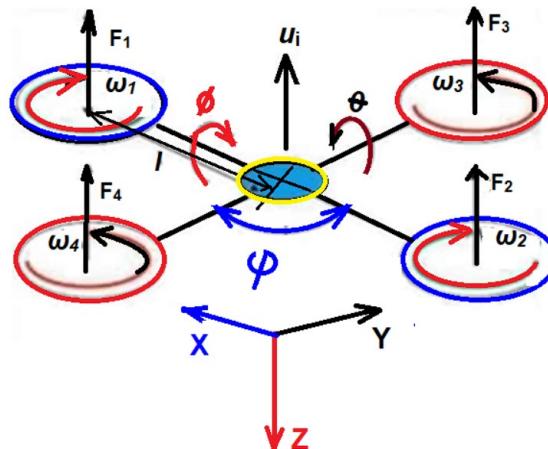
**Yaw:** chuyển động tự quay quanh trục z (trái hoặc phải)

- Nếu hai cánh quạt của một đường chéo bên phải chạy ở tốc độ cao, thì uav sẽ quay theo hướng ngược chiều kim đồng hồ
- Nếu hai cánh quạt của đường chéo bên trái chạy với tốc độ cao, thì uav sẽ quay theo chiều kim đồng hồ



Hình ảnh 1.3: Hướng chuyển động phụ thuộc chiều quay và tốc độ của từng cặp động cơ

### 1.2.2. Động học [1]



Hình ảnh 1.4: Động học của uav

- Giữ cân bằng (Hover)

Điều kiện giữ cân bằng:

$$\forall \text{ moment} = 0,$$



$$mg = F_1 + F_2 + F_3 + F_4$$

Phương trình chuyển động:

$$\begin{aligned} m &= F_1 + F_2 + F_3 + F_4 - mg \\ m &= 0 \end{aligned}$$

- Chuyển động tăng giảm độ cao (Throttle)

Điều kiện chuyển động lên cao:

$$\forall \text{ moment} = 0,$$

$$mg < F_1 + F_2 + F_3 + F_4$$

Điều kiện chuyển động xuống thấp:

$$\forall \text{ moment} = 0,$$

$$mg > F_1 + F_2 + F_3 + F_4$$

Phương trình chuyển động:

$$m = F_1 + F_2 + F_3 + F_4 - mg$$

$$m > 0$$

- Chuyển động quay (Yaw)

Điều kiện chuyển động:

$$mg = F_1 + F_2 + F_3 + F_4$$

$$\forall \text{ moment} \neq 0$$

Phương trình chuyển động:

$$m^*a_{tt} = F_1 + F_2 + F_3 + F_4 - mg, a_{tt} \text{ là gia tốc tuyén tính}$$

$$I_{zz}^*a_{gz} = M_1 + M_2 + M_3 + M_4$$

- Chuyển động tiến-lùi, trái-phải

Điều kiện chuyển động:

$$mg = F_1 + F_2 + F_3 + F_4$$

$$\forall \text{ moment} \neq 0$$

Phương trình chuyển động:

$$m^*a_{tt} = F_1 + F_2 + F_3 + F_4 - mg, a_{tt} \text{ là gia tốc tuyén tính}$$

$$I_{xx}^*a_{gx} = (M_3 + M_4)^*L$$

### 1.2.3. Lý thuyết bộ điều khiển PID

Bộ điều khiển PID được đặt tên theo 3 khâu hiệu chỉnh đó là khâu tỉ lệ, khâu tích phân và khâu vi phân, ngõ ra điều khiển tốc độ của 4 động cơ là tổng của 3 khâu này.

Ta có:

$$\text{Output}(t) = P_{\text{out}} + I_{\text{out}} + D_{\text{out}}$$

Trong đó

$P_{\text{out}}$  là thành phần đầu ra khâu tỉ lệ.

$I_{\text{out}}$  là thành phần đầu ra khâu tích phân.

Dout là thành phần đầu ra khâu vi phân.

## CHƯƠNG 2. THIẾT KẾ MÔ HÌNH

### 2.1. Phần cứng

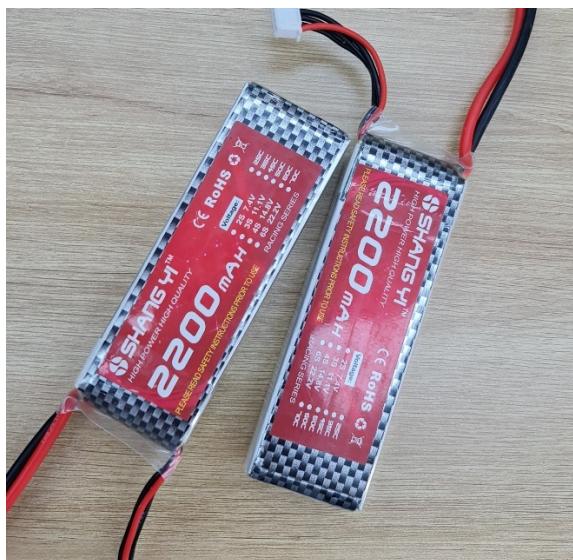
#### 2.1.1. Yêu cầu thiết kế

- Khung uav phải chắc chắn.
- Khối lượng phân bố đều theo tổng thể của uav.
- Giới hạn khối lượng tổng thể: <2kg.

#### 2.1.2. Pin lipo

Pin Li-Po là viết tắt của Lithium Polymer, là loại pin có thể sạc lại dùng công nghệ lithium-ion, sử dụng chất điện phân polymer thay vì chất điện phân lỏng. Pin Li-po được sử dụng rộng rãi trong các thiết bị di động, uav, ... Pin Li-Po có cấu tạo gồm 2 cực âm và dương và xung quanh nó là môi trường chứa các chất dung môi đặc biệt giúp lưu trữ điện năng một cách dễ dàng. Đặc điểm khiến pin Li-Po được ứng dụng nhiều là:

- Pin có thể sạc.
- Năng lượng lưu trữ lớn so với các loại pin khác có cùng khối lượng.
- Dòng xả lớn.



Hình ảnh 2.1: Pin Li-Po 3s 11.1V

### 2.1.3. Động cơ BLDC

Động cơ BLDC có nhiều ưu điểm so với động cơ có chổi than thông thường như tốc độ cao, nhỏ và nhẹ, quán tính thấp, độ ồn thấp, tuổi thọ cao.

Thông số kỹ thuật của động cơ BLDC A2212 loại 1000KV:

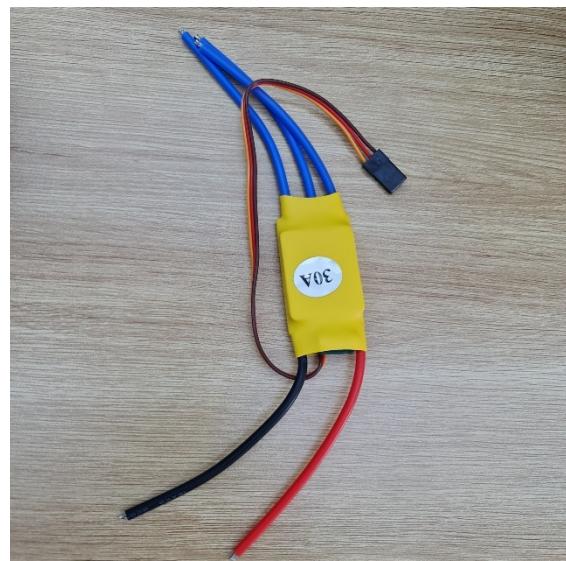
- + Tốc độ quay 1000KV = 1000 RPM/V, tốc độ thực tế khi sử dụng với cánh 1047 ở 10V là 6530 RPM.
- + Điện áp hoạt động: ~11.1V (pin Li-Po 3s)
- + Lực đẩy tối đa (khi sử dụng với cánh 1047): 820g (10V, 14A)



**Hình ảnh 2.2: Động cơ không chổi than BLDC**

### 2.1.4. Bộ điều khiển ESC

Là mạch công suất điều khiển động cơ BLDC, nhờ ESC mà ta có thể điều khiển động cơ BLDC chỉ bằng xung PWM.



**Hình ảnh 2.3: ESC điều khiển BLDC**

### 2.1.5. Cánh quạt

Tạo lực đẩy cho uav với hình dáng, kích thước và chiều quay phù hợp cho từng kích thước khung của uav.



**Hình ảnh 2.4: Cánh quạt**

### 2.1.6. Bộ tay cầm điều khiển MC6C và rx MC7RB

Bộ tay cầm điều khiển MC6C và rx MC7RB có các tính năng sau:

- + 6 kênh điều khiển.
- + Chế độ mix kênh, đảo kênh.
- + Khoảng cách truyền 400-800m.
- + Tần số bộ phát và thu: 2.4 GHz đến 2.483 GHz
- + Công suất  $\leq 100\text{mW}$
- + Dòng hoạt động  $\leq 120\text{mA}$
- + Nguồn cung cấp : 4.8V – 8V (Sử dụng 4 PIN AA)



**Hình ảnh 2.5: Tay điều khiển và bộ thu tín hiệu đi kèm**

### 2.1.7. Khung uav

Là thành phần cơ khí quan trọng, nơi gắn các thiết bị phần cứng, đảm bảo sự chắc chắn, tính đối xứng và phân bố đều về khối lượng, góp phần điều khiển uav được dễ dàng hơn.



**Hình ảnh 2.6: Khung quacopter loại F450**

### 2.1.8. Cảm biến MPU6050

MPU6050 là cảm biến tích hợp 2 cảm biến vận tốc góc (gyroscope) và gia tốc kế (accelerometer).

Thông số kỹ thuật:

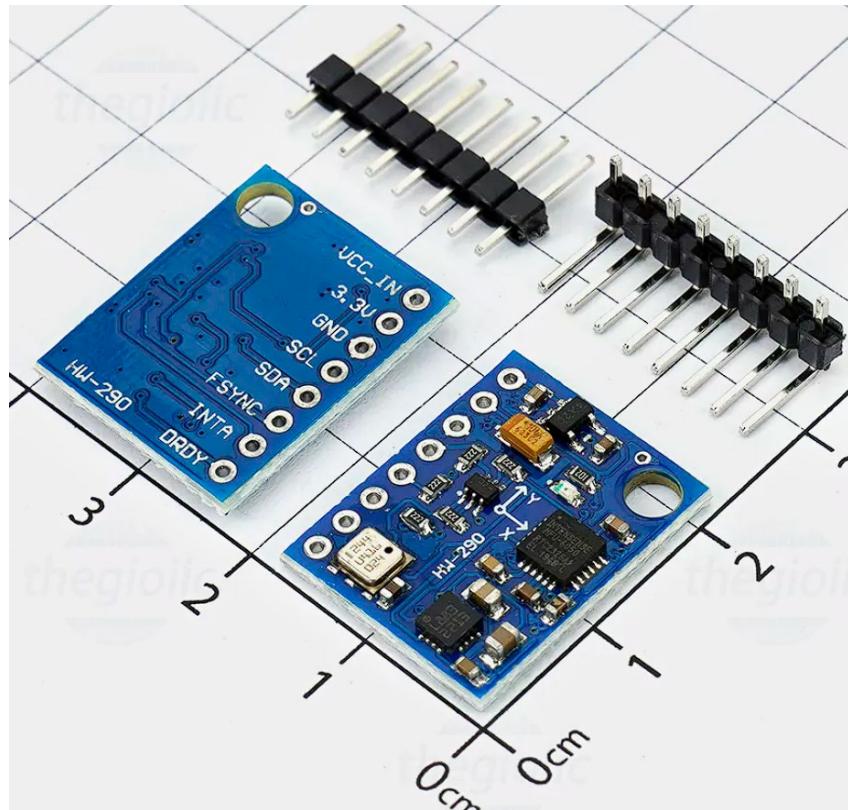
- + Đo vận tốc góc full-scale range: +/- 250 500 1000 2000 dps
- + Đo gia tốc trong các khoảng: +/- 2g, +/- 4g, +/- 8g, +/- 16g
- + Dữ liệu được lưu trong thanh ghi ADC 16bits.
- + Sử dụng giao tiếp I2C

- Đặc tính kỹ thuật của cảm biến vận tốc góc trên MPU6050:

Vấn đề thường gặp nhất của cảm biến vận tốc góc là bị trôi tích lũy dần theo thời gian do các rung động tác động lên cảm biến nên sau một thời gian sử dụng thì giá trị đo được không còn chính xác. Tuy nhiên, cảm biến vận tốc góc lại ít bị nhiễu hơn so với gia tốc kế nên giá trị tức thời của nó đáng tin cậy.

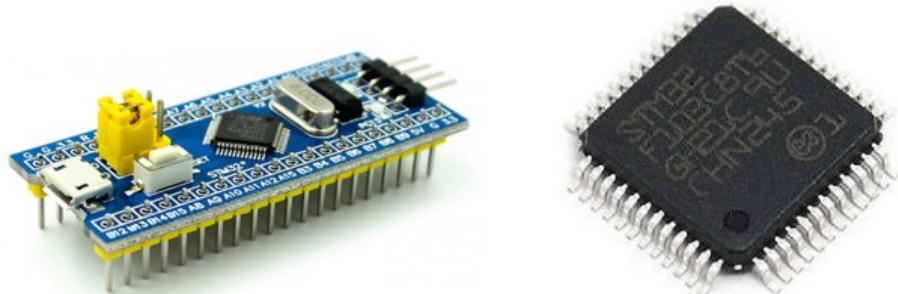
- Đặc tính kỹ thuật của gia tốc kế trên MPU6050:

Do ảnh hưởng của G mà mỗi trực của gia tốc kế luôn có sai số nhất định dẫn tới giá trị đo được thường bị lệch so với thực tế. Ngoài ra, gia tốc kế rất nhạy với rung động dù là rất nhỏ khiến các kết quả đo được tức thời trở lên không đáng tin cậy, do đó để sử dụng được giá trị đo thì chúng ta phải cho qua các bộ lọc. Mặt khác, gia tốc kế không bị trôi như cảm biến vận tốc góc nên sẽ được sử dụng để hiệu chỉnh lại cảm biến vận tốc góc thông qua thuật toán.



**Hình ảnh 2.7: Module cảm biến GY-87 tích hợp MPU6050**

### 2.1.9. Kit STM32F103C8 [2]



**Hình ảnh 2.8: Kit STM32F103C8T6**

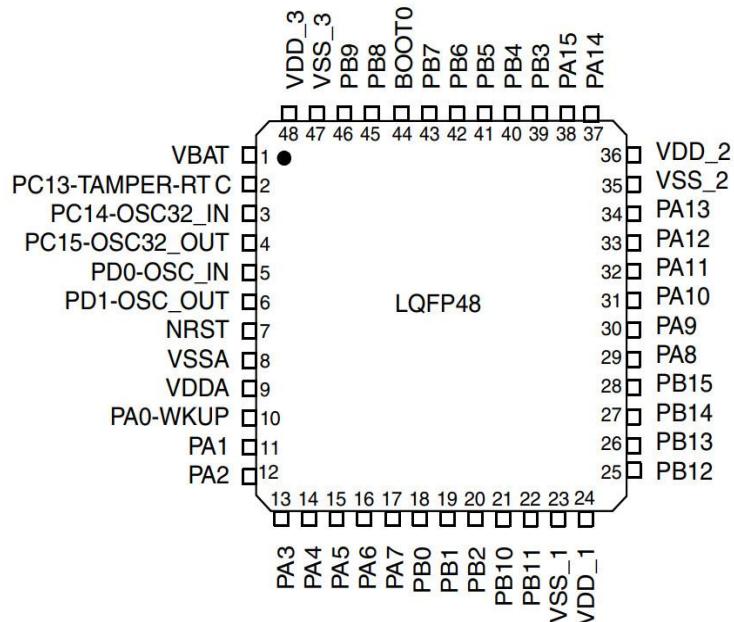
Có tích hợp các ngoại vi hay chuẩn giao tiếp như (GPIO, I2C, SPI, ADC, USB, Ethernet, ...), ST cung cấp cho chúng ta đầy đủ các thư viện cho mỗi dòng, chúng ta chỉ đơn giản là cấu hình, định nghĩa và khai báo để sử dụng.

STM32F103C8T6 hiệu suất cao và tiêu thụ năng lượng thấp.

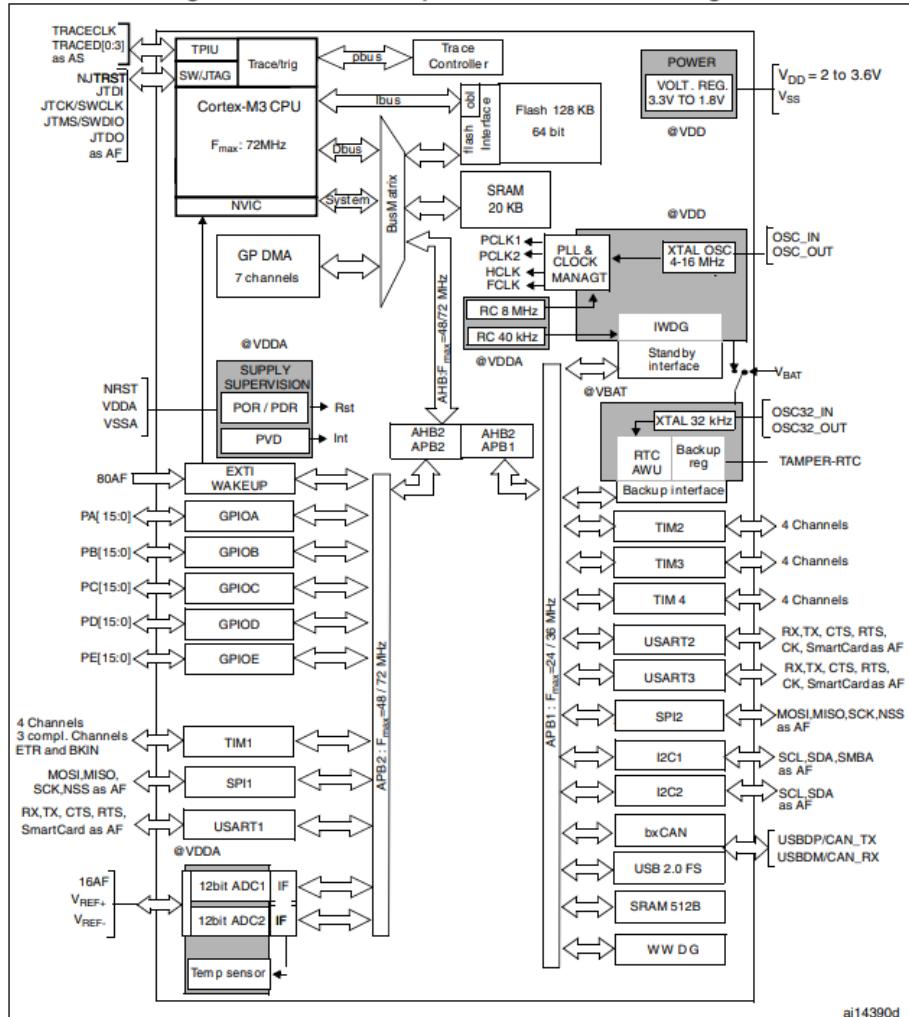
Đây là vi điều khiển 32 bit cung cấp những tính năng sau:

- Lõi ARM Cortex M3.

- 64K bytes bộ nhớ chương trình dạng Flash và 20K Byte SRAM.
- Độ rộng bus dữ liệu là 32 bit.
- Tần số đồng hồ tối đa là 72MHz.
- Độ phân giải ADC 12 bit.
- Số lượng kênh ADC là 10 kênh.
- Số lượng chân I/O là 37 I/O.
- Điện áp cấp vận hành từ 2V đến 3,6V.
- Nhiệt độ làm việc tối thiểu - 40°C
- Nhiệt độ làm việc tối đa + 85°C
- Loại giao tiếp CAN I2C SPI USART USB.
- Số bộ hẹn giờ timer là 4 bộ.



Hình ảnh 2.9: Sơ đồ chân STM32F103C8



Hình ảnh 2.10: Sơ đồ khối vi điều khiển STM32F103C8T6

### 2.1.10. Mạch nguồn hạ áp DC LM2596 [3]



Hình ảnh 2.11: Mạch nguồn hạ áp DC LM2596

Đầu vào: DC 3V đến 40V (điện áp đầu vào phải cao hơn điện áp đầu ra)

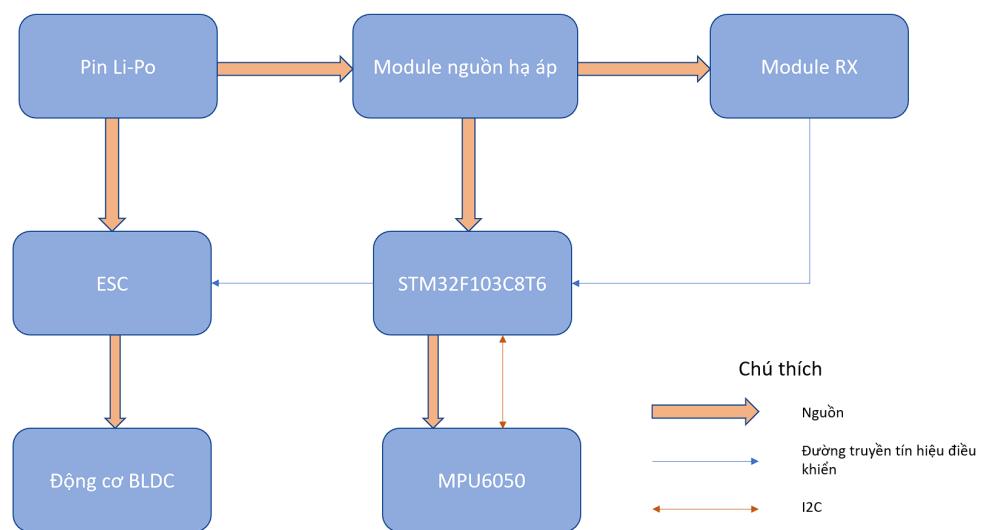
Đầu ra: DC 1.5V đến 35V điện áp điều chỉnh, dòng 3A.

Kích thước: 45( l)\* 20( w)\* 14( h) mm

Mạch được sử dụng để cung cấp nguồn điện cho kit stm32, cảm biến và module RX nhận tín hiệu từ tay điều khiển với nguồn vào sử dụng từ pin Li-Po 3s hạ xuống 5V.

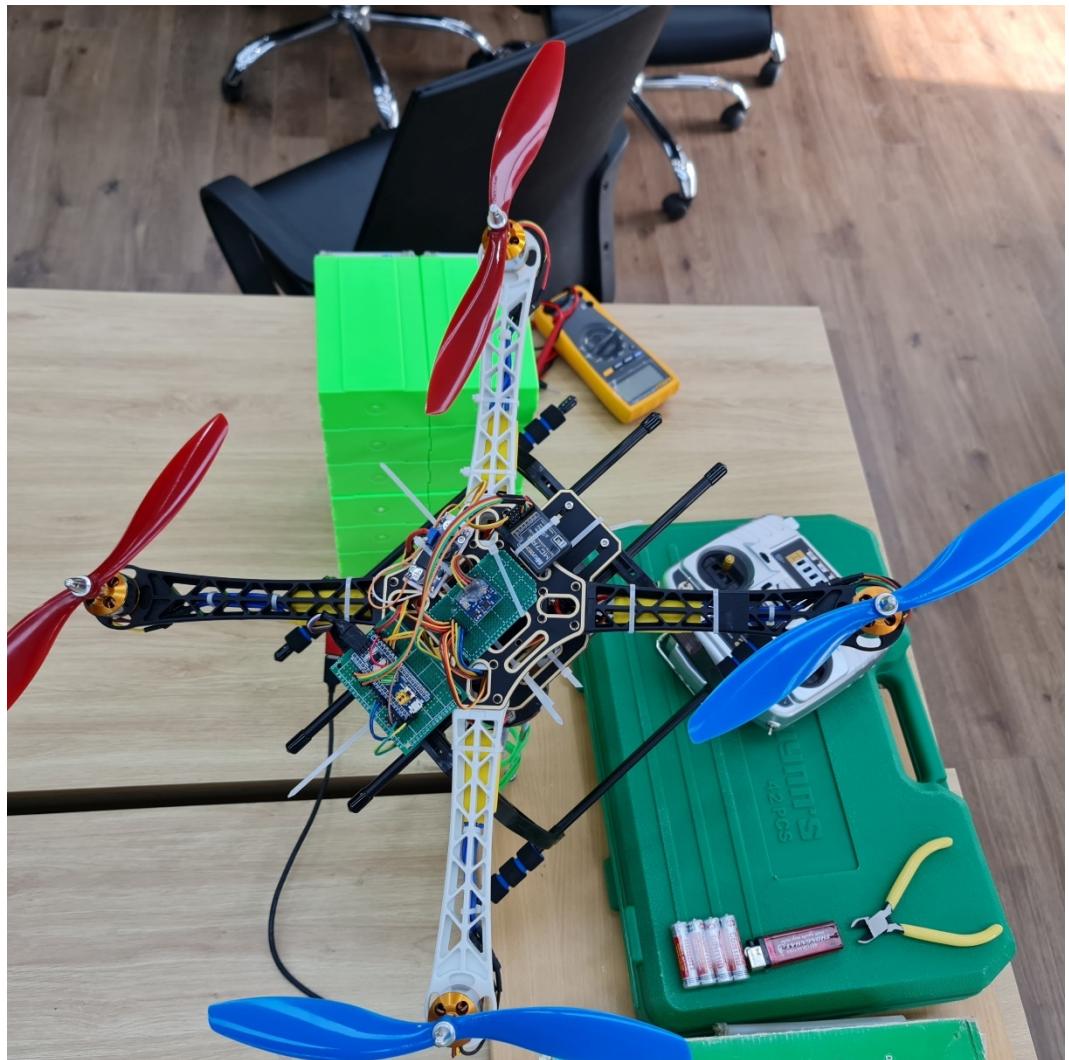
### **2.1.11. Sơ đồ khối ghép nối phần cứng**

Việc ghép nối được thực hiện đơn giản giữa các module với nhau thông qua các jack kết nối theo sơ đồ khối tổng quát dưới đây.



**Hình ảnh 2.12: Sơ đồ khối phần cứng**

### 2.1.12. Ghép nối phần cứng theo sơ đồ khối



**Hình ảnh 2.13: UAV sau khi hoàn tất lắp ráp phần cứng**

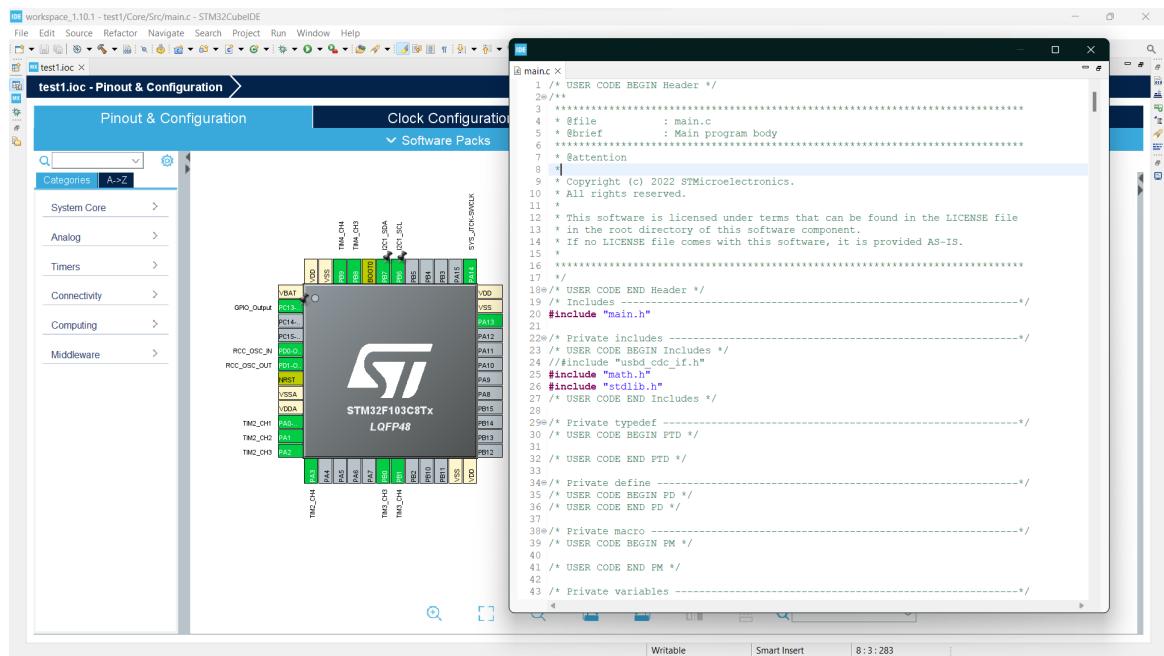
## 2.2. Phần mềm

### 2.2.1. Các phần mềm sử dụng trong đề tài

#### 2.2.1.1. STM32CubeIDE (ver 1.10.1)

STM32CubeIDE là một nền tảng phát triển C / C ++ tiên tiến với các tính năng cấu hình ngoại vi, tạo mã, biên dịch mã và gỡ lỗi dành riêng cho vi điều khiển và vi xử lý STM32.

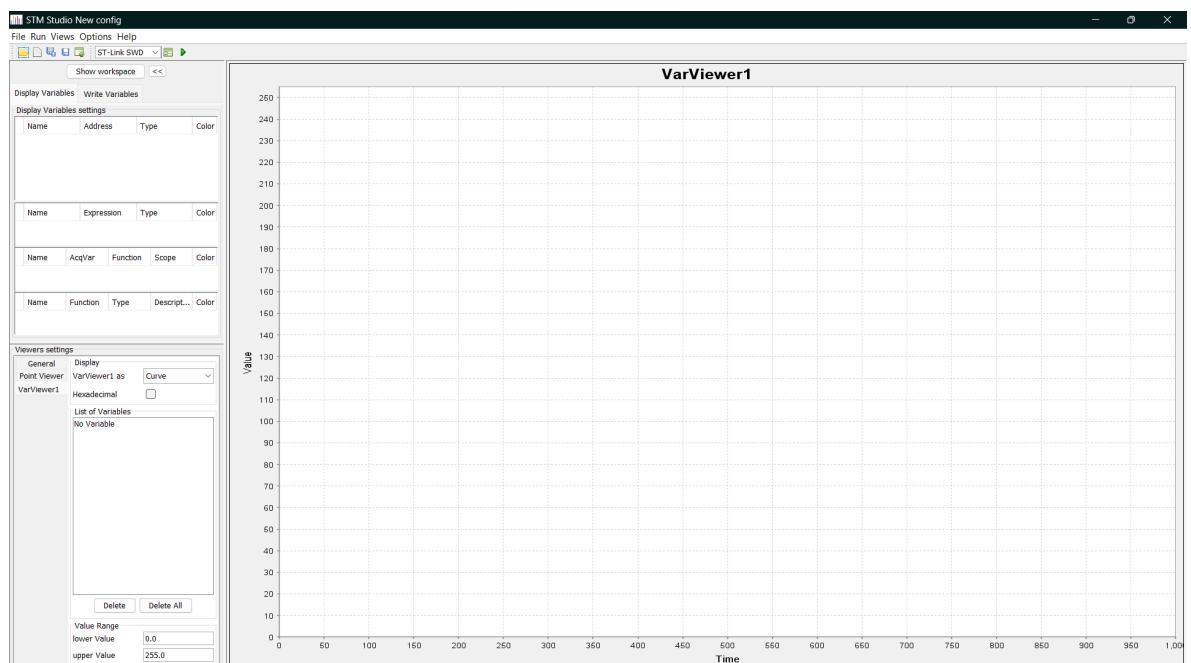
Giao diện sử dụng của phần mềm là sự kết hợp giữa phần mềm STM32CubeMX và KeilC mang tới sự tiện dụng tất cả trong một.



Hình ảnh 2.14: Giao diện phần mềm STM32CubeIDE

### 2.2.1.2. STMStudio

STM Studio là một phần mềm giúp gỡ lỗi và chẩn đoán các ứng dụng sử dụng STM8 và STM32 khi chúng đang chạy bằng cách đọc và hiển thị các biến của chúng trong thời gian thực. Nó rất phù hợp để gỡ lỗi các ứng dụng không thể dừng lại, chẳng hạn như các ứng dụng điều khiển động cơ.



**Hình ảnh 2.15: Giao diện phần mềm STMStudio**

## 2.2.2. Thiết kế chương trình điều khiển

### 2.2.2.1. Tổng quan chương trình điều khiển

Chúng ta cần xây dựng ba tính năng chính cho chương trình điều khiển:

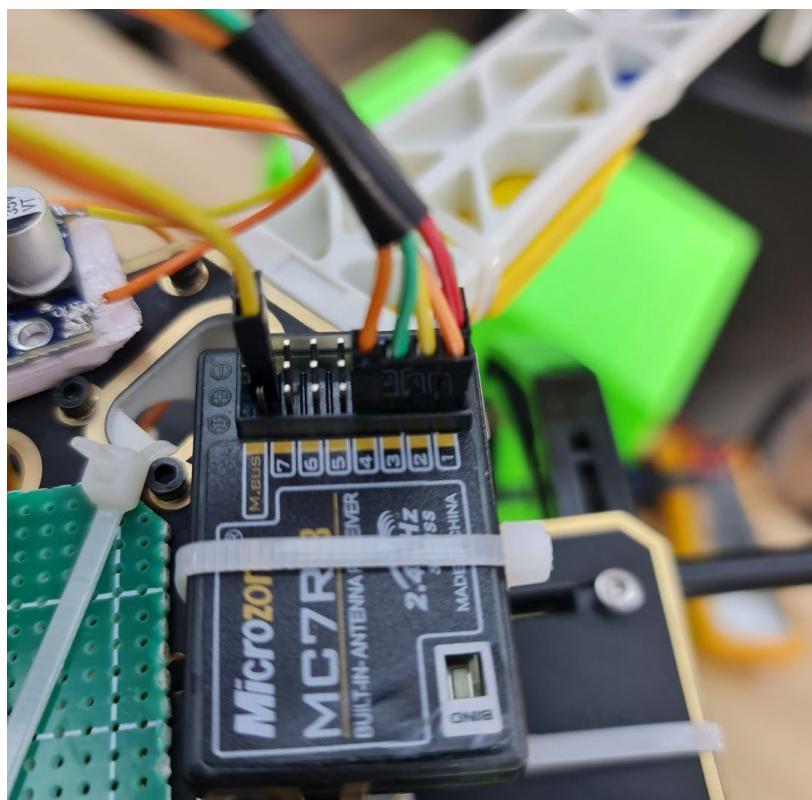
- + Đọc tín hiệu từ bộ thu được điều khiển bởi tay điều khiển từ xa.
- + Đọc dữ liệu từ cảm biến MPU6050.
- + Cấp xung PWM cho ESC để điều khiển động cơ.

Kết hợp các tính năng trên và thuật toán PID để được chương trình điều khiển bay cơ bản cho uav.

### 2.2.2.2. Đọc dữ liệu từ bộ thu RX

Đề tài sử dụng 4 kênh điều khiển:

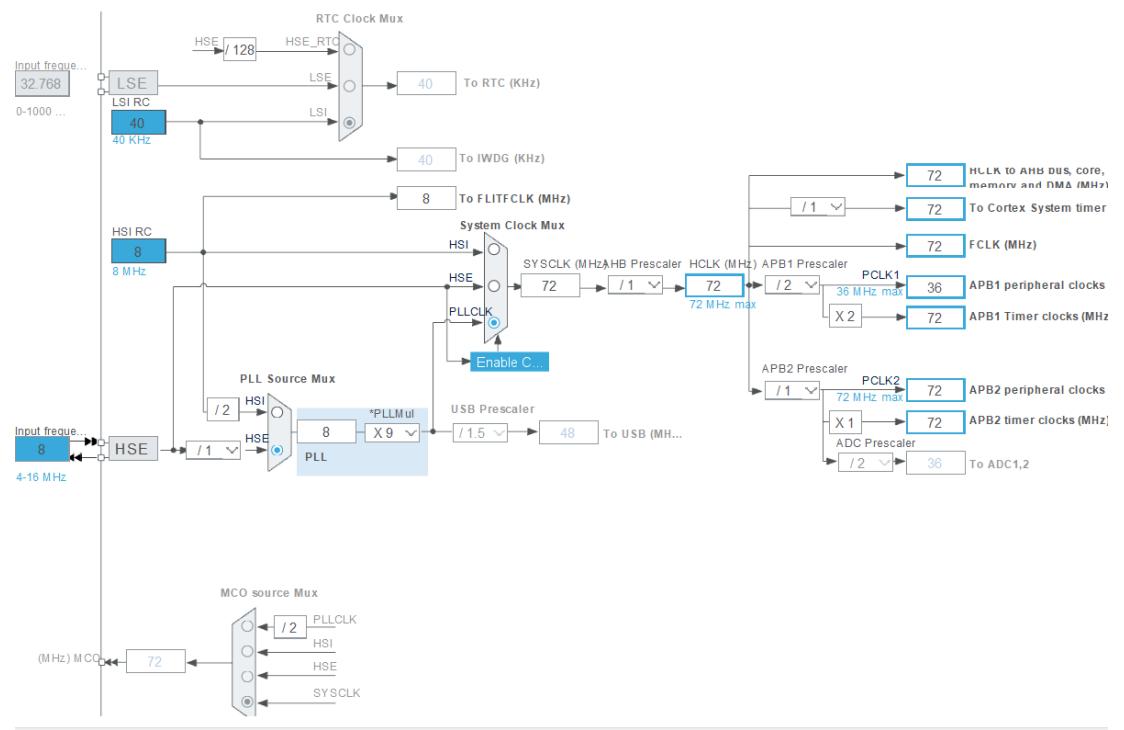
- + Kênh 1: điều khiển chuyển động roll.
- + Kênh 2: điều khiển chuyển động pitch.
- + Kênh 3: điều khiển chuyển động throttle.
- + Kênh 4: điều khiển chuyển động yaw.



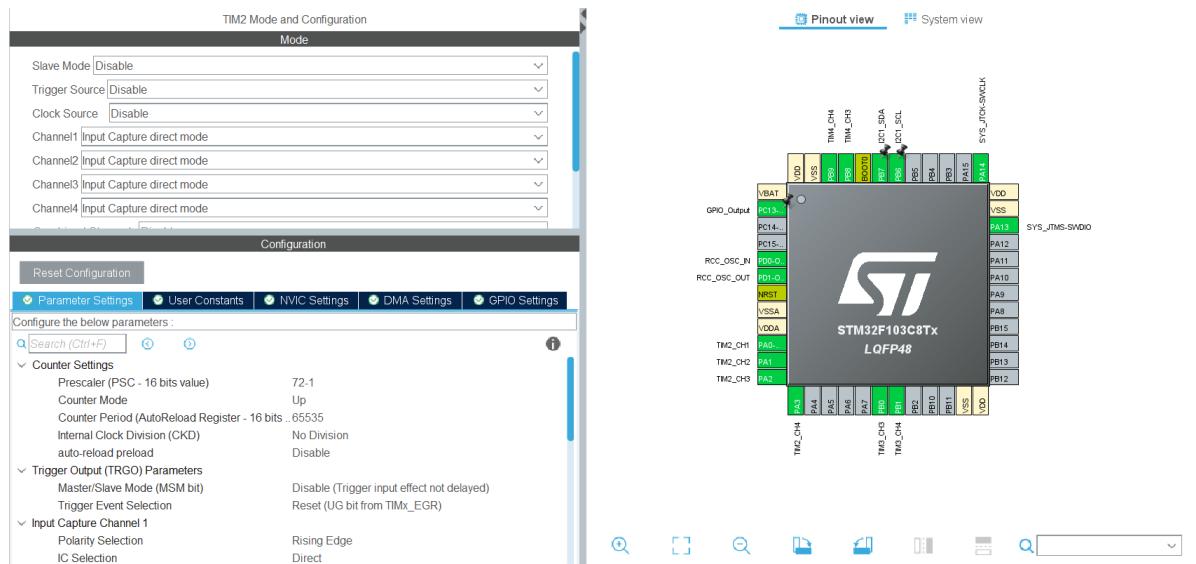
**Hình ảnh 2.16: Sử dụng 4 kênh điều khiển từ tay điều khiển tới module Rx**

Tín hiệu nhận được từ tay điều khiển từ xa thông qua bộ thu có dải độ rộng xung trong khoảng 1000us – 2000us (tương ứng với giá trị điều khiển tốc độ động cơ min và max của ESC).

Để đọc tín hiệu điều khiển từ tay cầm, chúng ta sử dụng tính năng Input Capture (chụp đầu vào) của STM32. Cấu hình tần số hoạt động của các TIMER được thể hiện như hình dưới.



**Hình ảnh 2.17: Thông số cài đặt đồng hồ cho STM32**



**Hình ảnh 2.18: Cài đặt chế độ input capture**

TIM2 được sử dụng để chụp đầu vào cho 4 kênh tín hiệu điều khiển, tương ứng lần lượt với các chân PA0, PA1, PA2, PA3. Cài đặt thông qua Parameter Settings với từng channel được để ở chế độ chụp sườn dương (Direct – Rising Edge).

Khai báo các biến lưu trữ và khai báo hàm ngắt TIM2 khi chụp được thay đổi tín hiệu vào.

```
uint32_t channel_1_start, channel_1;
uint32_t channel_2_start, channel_2;
uint32_t channel_3_start, channel_3;
uint32_t channel_4_start, channel_4;

volatile uint32_t diff1 = 0, diff2 = 0, diff3 = 0, diff4 = 0;

uint8_t firt_capture1 = 0, firt_capture2 = 0, firt_capture3 = 0, firt_capture4 = 0; //fist_capture? 0-no 1-yes

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
```

**Hình ảnh 2.19: Khai báo biến để chụp tín hiệu điều khiển**

Trong hàm xử lý ngắt, chúng ta kiểm tra đầu vào là từ channel nào thông qua lệnh:

```
if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
```

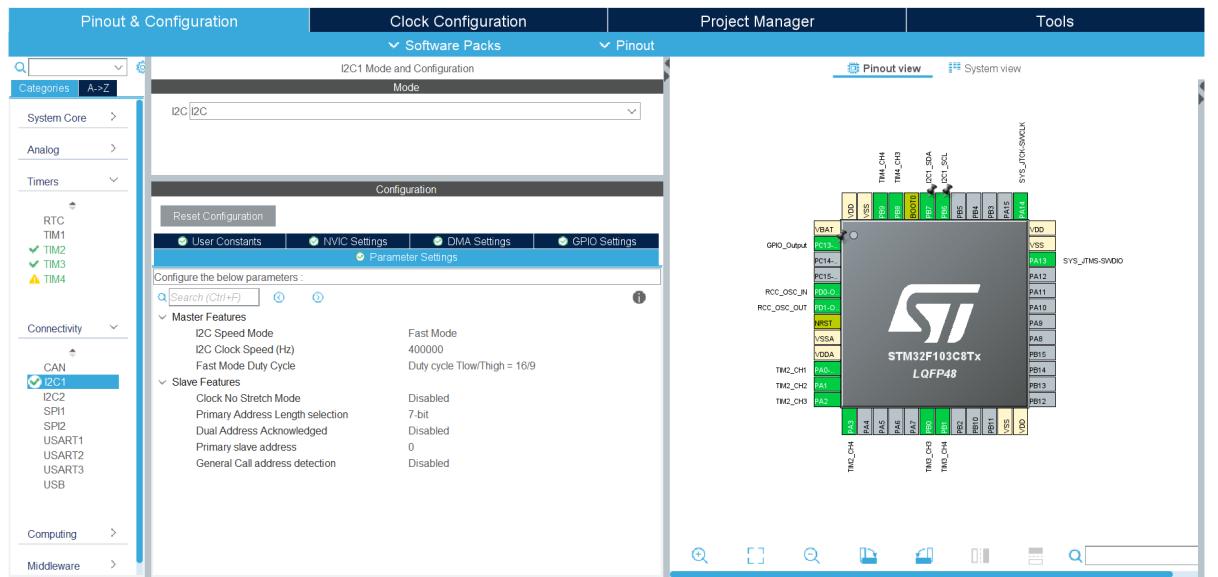
Khi đã xác nhận kênh đầu vào tiến tới xác nhận lần chụp tín hiệu vào là sườn dương hay âm (lần chụp đầu tiên là sườn dương). Sau khi chụp được tín hiệu sườn dương sẽ đổi sang chế độ chụp sườn âm (do cài đặt này không khả dụng qua setup bằng phần mềm đối với stm32f103) bằng lệnh

[HAL\\_TIM\\_SET\\_CAPTUREPOLARITY](#)

Độ rộng xung sau đó được lưu trữ trong biến diff1 tương ứng channel 1 của TIM2 – channel 1 của tay điều khiển. Tương tự cho các channel khác.

### 2.2.2.3. Đọc dữ liệu từ MPU6050

Thiết lập giao tiếp I2C với MPU6050 với chế độ Fast Mode với Duty cycle Tlow/high = 16/9 thông qua phần mềm.



Hình ảnh 2.20: Cài đặt thông số cho I2C

Sau đó tiến hành khai báo địa chỉ thanh ghi của MPU6050 (để cài đặt cảm biến và đọc dữ liệu)

```
#define MPU6050_ADDR 0xD0
#define SMPLRT_DIV_REG 0x19
#define GYRO_CONFIG_REG 0x1B
#define ACCEL_CONFIG_REG 0x1C
#define ACCEL_XOUT_H_REG 0x3B
#define TEMP_OUT_H_REG 0x41
#define GYRO_XOUT_H_REG 0x43
#define PWR_MGMT_1_REG 0x6B
#define WHO_AM_I_REG 0x75

int16_t Accel_X_RAW = 0, Accel_Y_RAW = 0, Accel_Z_RAW = 0, Gyro_X_RAW = 0,
       Gyro_Y_RAW = 0, Gyro_Z_RAW = 0;
int16_t temp_raw = 0;
float Ax, Ay, Az, Gx, Gy, Gz, temp;
```

Hình ảnh 2.21: Khai báo địa chỉ của thanh ghi trên cảm biến [4] [5]

Ngoài cài đặt thông số giá trị cơ bản trả về thông qua thanh ghi GYRO\_CONFIG\_REG và ACCEL\_CONFIG\_REG, chương trình còn sử dụng bộ

lọc thông thấp tích hợp trên cảm biến để cải thiện giá trị đo theo địa chỉ 0X1A và giá trị bộ lọc theo bảng dưới đây.

DLPF_CFG	Accelerometer (Fs = 1kHz)		Gyroscope		
	Bandwidth (Hz)	Delay (ms)	Bandwidth (Hz)	Delay (ms)	Fs (kHz)
0	260	0	256	0.98	8
1	184	2.0	188	1.9	1
2	94	3.0	98	2.8	1
3	44	4.9	42	4.8	1
4	21	8.5	20	8.3	1
5	10	13.8	10	13.4	1
6	5	19.0	5	18.6	1
7	RESERVED		RESERVED		8

Hình ảnh 2.22: Thông số của bộ lọc thông thấp tích hợp theo từng cấp độ [5]

DLPF Setting	minX	minY	minZ	maxX	maxY	maxZ
0	-1408	912	7362	-906	1574	8044
1	-1408	1023	7370	-994	1407	7970
2	-1408	1152	7552	-1041	1407	7938
3	-1388	1152	7552	-1290	1407	7807
4	-1403	1154	7552	-1154	1407	7895
5	-1360	1345	7558	-1316	1390	7807
6	-1354	1355	7566	-1322	1383	7619

Hình ảnh 2.23: Các giá trị gia tốc và tốc độ góc của từng cấp độ lọc tương ứng [6]

Do việc sử dụng bộ lọc gây ra thời gian trễ, nên để phù hợp, đề tài này sử dụng DLPF\_CFG = 3, tần số đọc cảm biến là 250Hz (đồng thời là tần số hoạt động của cả chương trình điều khiển).

Hàm đọc dữ liệu cảm biến trả về và hàm tính trung bình giá trị cảm biến khi ở vị trí gốc – giá trị offset (đây là vị trí cân bằng ban đầu ở dưới đất, và cũng là vị trí gốc để đo độ lệch khi vị trí, tốc độ của uav thay đổi)

```

void MPU6050_Read(void) {
    uint8_t Rec_Data[14];

    HAL_I2C_Mem_Read(&hi2c1, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data, 14,
                      1000);

    Accel_X_RAW = (int16_t) Rec_Data[0] << 8 | Rec_Data[1];
    Accel_Y_RAW = (int16_t) Rec_Data[2] << 8 | Rec_Data[3];
    Accel_Z_RAW = (int16_t) Rec_Data[4] << 8 | Rec_Data[5];
    temp_raw = (int16_t) Rec_Data[6] << 8 | Rec_Data[7];
    Gyro_X_RAW = (int16_t) Rec_Data[8] << 8 | Rec_Data[9];
    Gyro_Y_RAW = (int16_t) Rec_Data[10] << 8 | Rec_Data[11];
    Gyro_Z_RAW = (int16_t) Rec_Data[12] << 8 | Rec_Data[13];

    Gyro_Y_RAW *= -1;
    Gyro_Z_RAW *= -1;

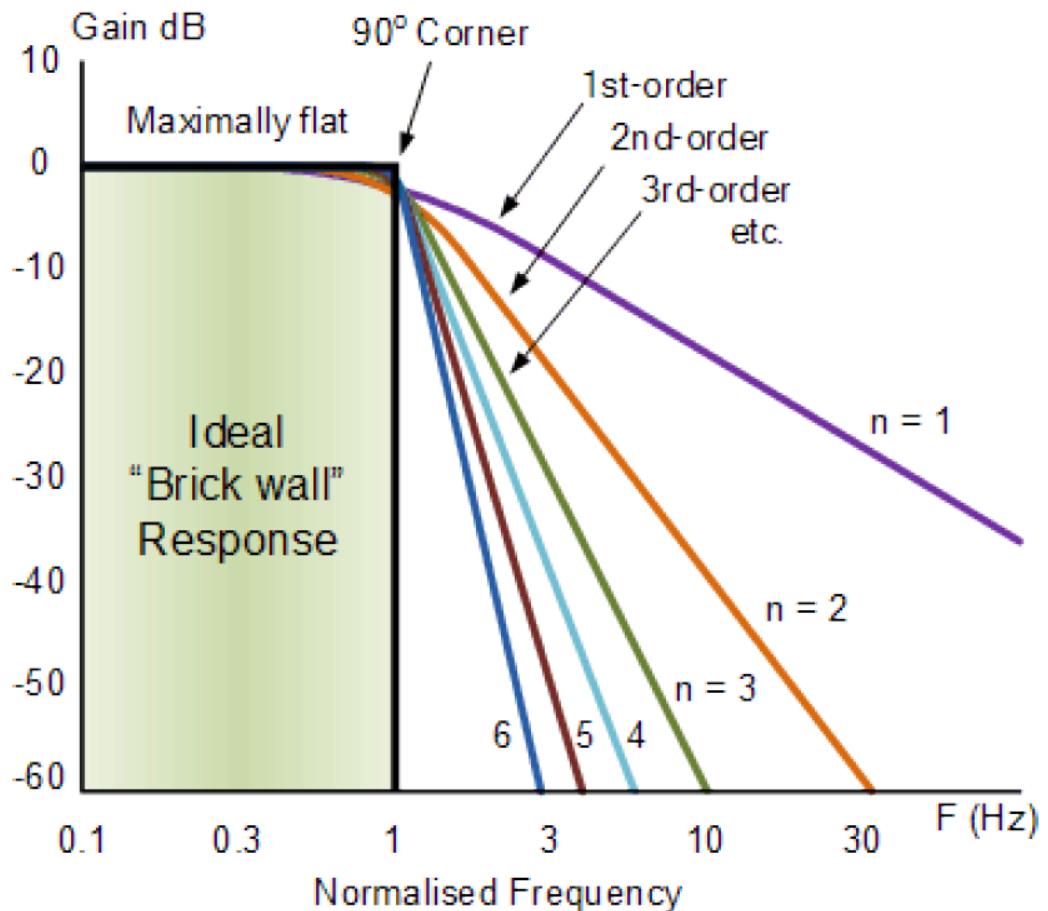
    Accel_X_RAW -= os_Ax;
    Accel_Y_RAW -= os_Ay;
    Accel_Z_RAW -= os_Az;
    Gyro_X_RAW -= os_Gx;
    Gyro_Y_RAW -= os_Gy;
    Gyro_Z_RAW -= os_Gz;
}

```

### Hình ảnh 2.24: Hàm đọc giá trị cảm biến

Sau khi lấy được các giá trị gốc (độ lệch so với gốc), chúng ta chuyển đổi các giá trị nhận được về gia tốc và góc để làm thông số vào cho bộ điều khiển PID sau này.

Để sử dụng được tín hiệu góc vào PID ổn định hơn, em sử dụng bộ lọc thông thấp Butterworth. Bộ lọc Butterworth là một loại bộ lọc xử lý tín hiệu được thiết kế để có đáp ứng tần số phẳng nhất có thể (không có gợn sóng) trong dải thông và phản hồi “0” trong dải dừng. Bộ lọc Butterworth là một trong những bộ lọc kỹ thuật số được sử dụng phổ biến nhất trong phân tích chuyển động và trong các mạch âm thanh. Chúng nhanh và đơn giản để sử dụng. Để tài sử dụng tần số ngắt 10Hz.



Hình ảnh 2.25: Bộ lọc thông thấp Butterworth với tần số cắt [7]

```
// Apply low-pass filter (10Hz cutoff frequency)
angular_motions[roll] = 0.7 * angular_motions[roll] + ((float) Gyro_X_RAW * 0.3 / 65.5); //Gyro pid input is deg/sec.
angular_motions[pitch] = 0.7 * angular_motions[pitch] + ((float) Gyro_Y_RAW * 0.3 / 65.5);
angular_motions[yaw] = 0.7 * angular_motions[yaw] + ((float) Gyro_Z_RAW * 0.3 / 65.5);
```

Đọc góc đo và hiệu chỉnh sự trôi góc dựa theo cảm biến gia tốc góc.

```

// Angle calculation using integration
gyro_angle[roll] += (float)(Gyro_X_RAW / (250 * 65.5));
gyro_angle[pitch] += (float)(Gyro_Y_RAW / (250 * 65.5));

// Transfer roll to pitch if IMU has yawed
gyro_angle[pitch] -= gyro_angle[roll] * sin((float)Gyro_Z_RAW * (M_PI / (250 * 65.5 * 180)));
gyro_angle[roll] += gyro_angle[pitch] * sin((float)Gyro_Z_RAW * (M_PI / (250 * 65.5 * 180)));

//Accelerometer angle calculations
//acc_total_vector = sqrt(pow(Accel_X_RAW, 2) + pow(Accel_Y_RAW, 2) + pow(Accel_Z_RAW, 2));
acc_total_vector = sqrt(pow(os_Ax, 2) + pow(os_Ay, 2) + pow(os_Az, 2));
if (Accel_X_RAW > acc_total_vector)
    Accel_X_RAW = acc_total_vector; //Limit the maximum accelerometer value.
if (Accel_X_RAW < -acc_total_vector)
    Accel_X_RAW = -acc_total_vector;
if (Accel_Y_RAW > acc_total_vector)
    Accel_Y_RAW = acc_total_vector; //Limit the maximum accelerometer value.
if (Accel_Y_RAW < -acc_total_vector)
    Accel_Y_RAW = -acc_total_vector;

acc_angle[roll] = asin((float)Accel_X_RAW / acc_total_vector) * (-180/M_PI); //Calculate the pitch angle.
acc_angle[pitch] = asin((float)Accel_Y_RAW / acc_total_vector) * (180/M_PI); //Calculate the roll angle.

gyro_angle[roll] = gyro_angle[roll] * 0.995 + acc_angle[roll] * 0.005;
gyro_angle[pitch] = gyro_angle[pitch] * 0.995 + acc_angle[pitch] * 0.005;

// To dampen the pitch and roll angles a complementary filter is used
measures[roll] = measures[roll] * 0.8 + gyro_angle[roll] * 0.2;
measures[pitch] = measures[pitch] * 0.8 + gyro_angle[pitch] * 0.2;
measures[yaw] = Gyro_Z_RAW / 65.5; //Store the angular motion for this axis

```

**Hình ảnh 2.26: Mã code đọc giá trị góc lệch so với gốc**

Kết quả đo là độ lệch góc theo vị trí lấy gốc ban đầu được lưu trữ bởi biến measures[] tương ứng với độ lệch của từng trục.

#### 2.2.2.4. Bộ điều khiển PID

- Bộ điều khiển PID tổng quát ở miền liên tục:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de(t)}{dt}$$

Bộ điều khiển PID ở miền rời rạc với khởi tạo  $e(0) = 0$ .

$$u(k) = k_p e(k) + k_i \sum_{i=1}^k e(k)T + k_d \frac{e(k) - e(k-1)}{T}$$

Trong đó T là chu kỳ thực thi bộ điều khiển PID, trong trường hợp của đề tài này chính là thời gian lấy mẫu của cảm biến MPU6050. Vậy, với  $f = 250\text{Hz} \Rightarrow T = 4\text{ms}$ .

P (Proportional): là khâu điều chỉnh tỉ lệ chính, giúp tạo ra tín hiệu điều chỉnh tỉ lệ với sai lệch đầu vào theo thời gian lấy mẫu.

I (Integral): là khâu tích phân của sai lệch theo thời gian lấy mẫu (sai lệch ở đề tài này là góc lệch so với vị trí cân bằng gốc). Theo dõi tín hiệu đặt (setpoint) và tín hiệu hồi tiếp (từ cảm biến) sau đó điều chỉnh để độ lệch giảm dần về 0.

D (Derivative): là khâu vi phân của sai lệch, được dùng để hạn chế sự đáp ứng quá nhanh gây hiện tượng vọt lố mà khâu P gây ra.

Phương pháp được sử dụng trong đề tài để tìm các hệ số Kp, Ki, Kd là chỉnh định bằng tay dựa theo đặc điểm của từng khâu.

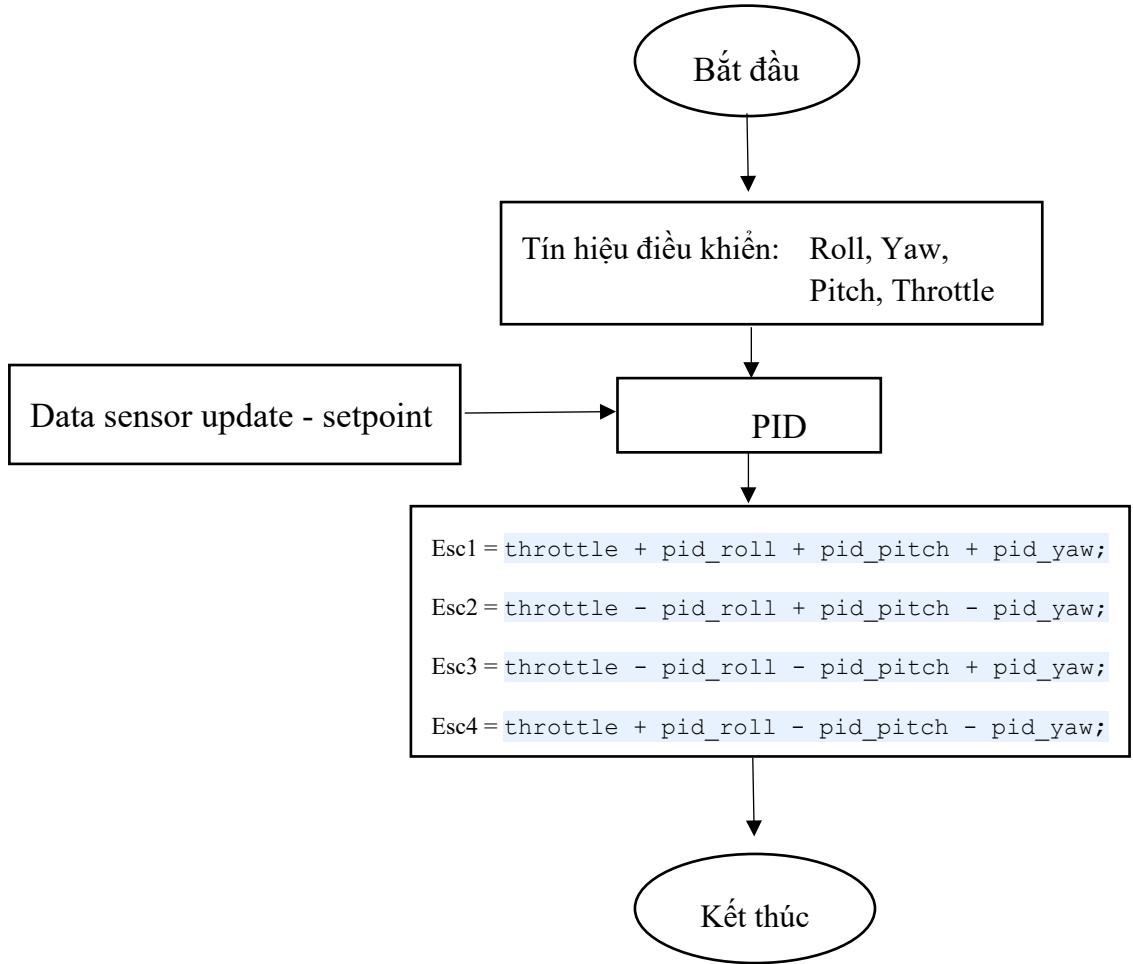
Các bước của phương pháp chỉnh định như sau:

Bước 1: đặt  $Ki = Kd = 0$ , tăng dần hệ số Kp cho đến khi hệ dao động ổn định.

Bước 2: giảm hệ số Kp vừa tìm được 50%, tăng hệ số Kd tới khi hệ dao động nhỏ và nhanh.

Bước 3: giảm hệ số Kd vừa tìm được 50%, tăng hệ số Ki tới khi hệ dao động, sau đó giảm dần để tìm mức thay đổi phù hợp.

## Giải thuật cân bằng các trục



Code thực thi:

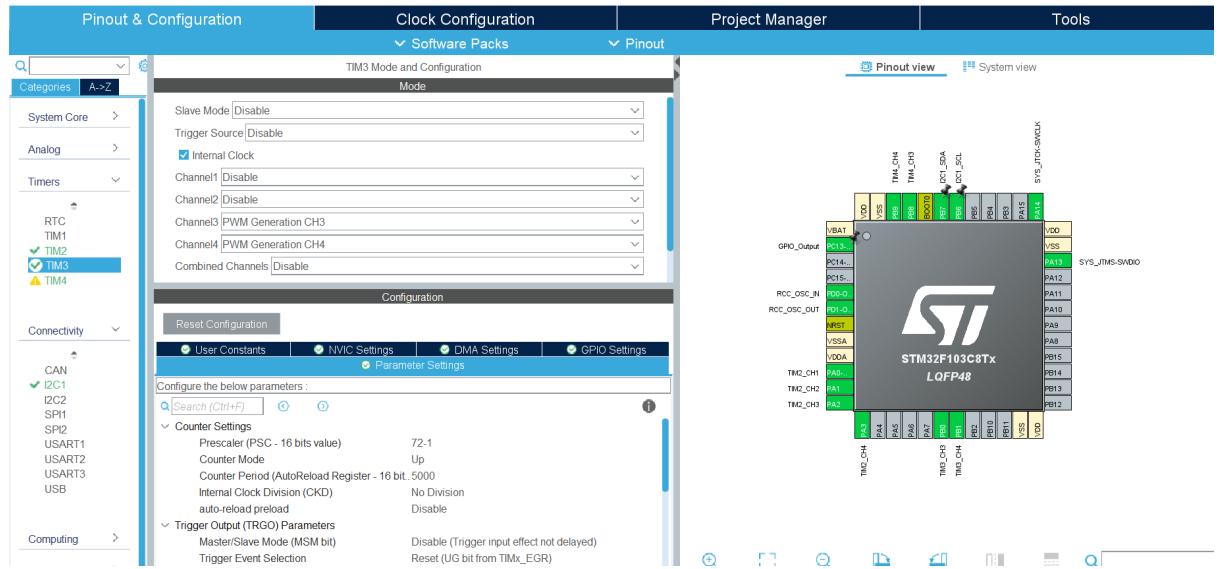
```

// PID = e.Kp + ∫e.Ki + Δe.Kd
pid_roll = (errors[roll] * Kp[roll]) + (error_sum[roll] * Ki[roll]) * 0.004 + (delta_err[roll] * Kd[roll])/0.004;
pid_pitch = (errors[pitch] * Kp[pitch]) + (error_sum[pitch] * Ki[pitch]) * 0.004 + (delta_err[pitch] * Kd[pitch])/0.004;
pid_yaw = (errors[yaw] * Kp[yaw]) + (error_sum[yaw] * Ki[yaw]) * 0.004 + (delta_err[yaw] * Kd[yaw])/0.004;

if(pid_roll > pid_max[roll]) pid_roll = pid_max[roll];
if(pid_roll < (-pid_max[roll])) pid_roll = -pid_max[roll];
if(pid_pitch > pid_max[pitch]) pid_pitch = pid_max[pitch];
if(pid_pitch < (-pid_max[pitch])) pid_pitch = -pid_max[pitch];
if(pid_yaw > pid_max[yaw]) pid_yaw = pid_max[yaw];
if(pid_yaw < (-pid_max[yaw])) pid_yaw = -pid_max[yaw];
  
```

**Hình ảnh 2.27: Thuật toán PID**

Cấu hình chân output PWM trên STM32 điều khiển BLDC, sử dụng channel 3, 4 của TIM3 và channel 3, 4 của TIM4.

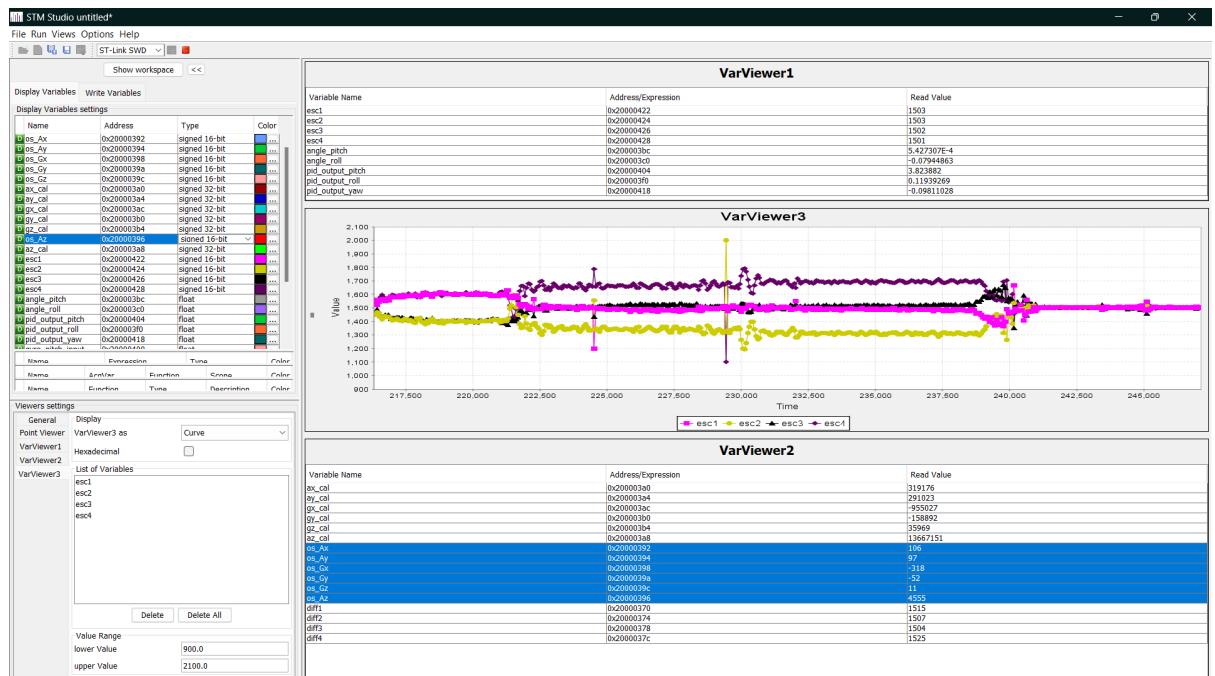


**Hình ảnh 2.28: Cấu hình chân điều khiển động cơ qua chế độ PWM Generation**

## CHƯƠNG 3. KẾT QUẢ

### 3.1. Kết quả đạt được

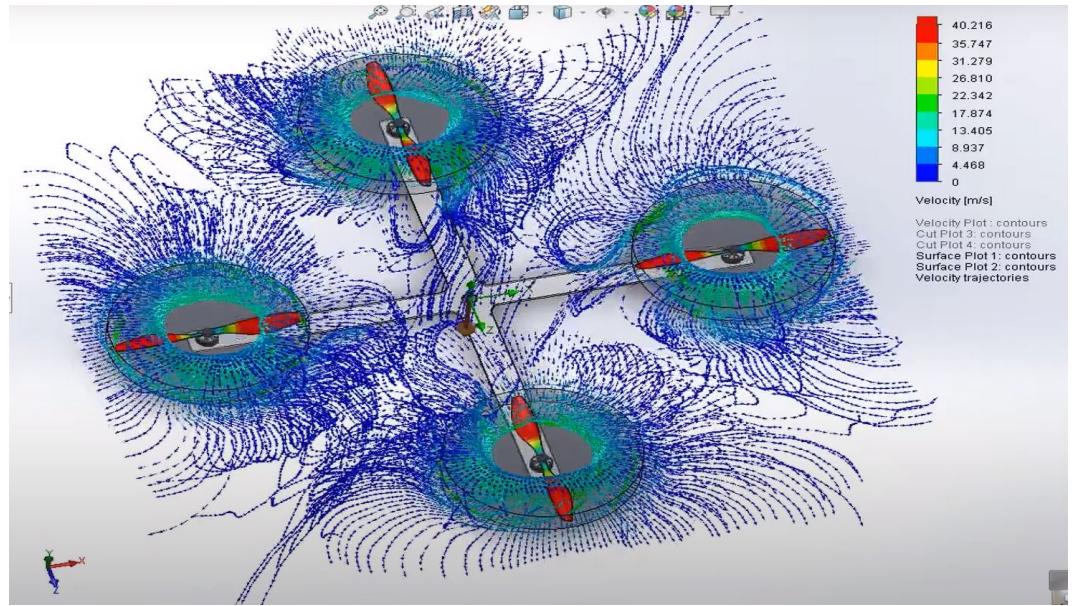
- Hoàn thành ghép nối, cấu hình phần cứng cho uav.
- Đọc dữ liệu từ cảm biến dưới dạng sai số góc lệch.
- Khắc phục hiện tượng trôi góc của cảm biến góc.
- Áp dụng bộ lọc thông thấp tích hợp và các bộ lọc kỹ thuật số để xử lý nhiễu của cảm biến.
- Xây dựng thuật toán điều khiển sử dụng PID, kết hợp tay điều khiển để điều khiển tốc độ động cơ BLDC theo độ nghiêng giữa các trục.



Hình ảnh 3.1: Đọc giá trị các biến và biểu đồ theo thời gian thực giá trị điều khiển động cơ

### 3.2. Kết quả chưa đạt được và các vấn đề gặp phải

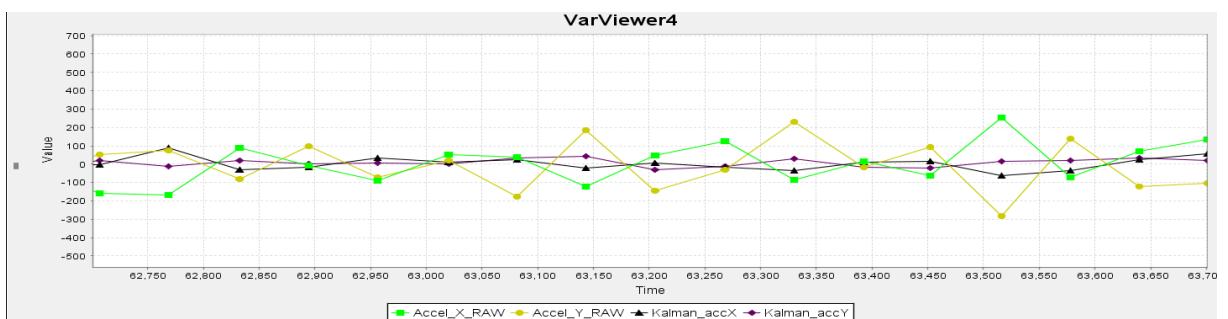
- Thuật toán PID với các hệ số của các khâu điều khiển chưa tối ưu.
- Lỗi treo vi điều khiển và cảm biến => cài đặt các giá trị gốc ban đầu bị sai lệch.
- Rung lắc mạnh do không tìm được cánh phù hợp => dẫn tới cảm biến đọc sai lệch.
- Cánh mềm dẻo => làm giảm lực nâng khiến uav không bay được.



**Hình ảnh 3.2: Mô phỏng khí động học và vận tốc dài các điểm trên cánh [8]**

### 3.3. Hướng phát triển của đề tài

- Cải thiện thuật toán PID, kết hợp các phương pháp tìm thông số điều khiển khác phù hợp hơn (mô phỏng).
- Thay cánh phù hợp với động cơ.
- Sử dụng led để báo trạng thái hoạt động và trạng thái lỗi.
- Áp dụng bộ lọc Kalman để cải thiện chất lượng tín hiệu đầu vào.



**Hình ảnh 3.3: Gia tốc đo được không qua bộ lọc Kalman và đã qua bộ lọc Kalman**

## TÀI LIỆU THAM KHẢO

- [1] cfdflowengineering@gmail.com, 2022. [Online]. Available: [https://cfdflowengineering.com/working-principle-and-components-of-drone/#Quadcopter\\_Dynamics](https://cfdflowengineering.com/working-principle-and-components-of-drone/#Quadcopter_Dynamics).
- [2] STMicroelectronics, 2022. [Online]. Available: <https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html>.
- [3] linhkienvietnam.vn. [Online]. Available: [https://linhkienvietnam.vn/module-nguon-lm2596-b3h13?variantId=73232552&gclid=CjwKCAiAwc-dBhA7EiwAxPRyIC7pf-AL5GbrBvT9vjjJUSQi5jTwN-rb6oqale6n7oXJ2jtym4egOxoCh7YQAvD\\_BwE](https://linhkienvietnam.vn/module-nguon-lm2596-b3h13?variantId=73232552&gclid=CjwKCAiAwc-dBhA7EiwAxPRyIC7pf-AL5GbrBvT9vjjJUSQi5jTwN-rb6oqale6n7oXJ2jtym4egOxoCh7YQAvD_BwE).
- [4] jarzebski, 31 Jul 2017. [Online]. Available: <https://github.com/jarzebski/Arduino-MPU6050>.
- [5] I. Inc., Writer, MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2. [Performance]. 2013.
- [6] U. Buschbaum, 18 January 2015. [Online]. Available: <https://ulrichbuschbaum.wordpress.com/2015/01/18/using-the-mpu6050s-dlpf/>.
- [7] V. Kim, in How to Design 10 kHz filter (Using Butterworth filter design).
- [8] L. V. Đức, 21 12 2021. [Online]. Available: <https://www.youtube.com/watch?v=nod5tmuz1uI>.

## PHỤ LỤC

### Code đồ án

```

/* USER CODE END Header */
/* Includes -----*/
/*
#include "main.h"

/* Private includes -----*/
/*
/* USER CODE BEGIN Includes */
//#include "usbd_cdc_if.h"
#include "math.h"
#include "stdlib.h"
/* USER CODE END Includes */

/* Private typedef -----*/
/*
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define -----*/
/*
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -----*/
/*
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables -----*/
I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim4;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----*/
/*
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_I2C1_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code -----*/
/*
/* USER CODE BEGIN 0 */
//Input capture callback
uint32_t channel_1_start, channel_1;

```

```

uint32_t channel_2_start, channel_2;
uint32_t channel_3_start, channel_3;
uint32_t channel_4_start, channel_4;

volatile uint32_t diff1 = 0, diff2 = 0, diff3 = 0, diff4 = 0;

uint8_t firt_capture1 = 0, firt_capture2 = 0, firt_capture3 = 0,
firt_capture4 =
    0; //fist_capture? 0-no 1-yes

void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim) {
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1) {
        if (firt_capture1 == 0) {
            channel_1_start = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_1);
            firt_capture1 = 1;
            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1,
                TIM_INPUTCHANNELPOLARITY_FALLING);
        } else if (firt_capture1 == 1) {
            channel_1 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_1);
            if (channel_1 > channel_1_start)
                diff1 = channel_1 - channel_1_start;
            else if (channel_1 < channel_1_start)
                diff1 = (0xffff - channel_1_start) + channel_1;

            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_1,
                TIM_INPUTCHANNELPOLARITY_RISING);
            firt_capture1 = 0;
        }
    }
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2) {
        if (firt_capture2 == 0) { //read
            channel_2_start = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_2);
            firt_capture2 = 1;
            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_2,
                TIM_INPUTCHANNELPOLARITY_FALLING);
        } else if (firt_capture2 == 1) {
            channel_2 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_2);
            if (channel_2 > channel_2_start)
                diff2 = channel_2 - channel_2_start;
            else if (channel_2 < channel_2_start)
                diff2 = (0xffff - channel_2_start) + channel_2;

            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_2,
                TIM_INPUTCHANNELPOLARITY_RISING);
            firt_capture2 = 0;
        }
    }
    if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3) {
        if (firt_capture3 == 0) { //read
            channel_3_start = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_3);
            firt_capture3 = 1;
            //Change the input capture mode to the falling edge of the
pulse
            __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_3,
                TIM_INPUTCHANNELPOLARITY_FALLING);
        } else if (firt_capture3 == 1) {
            channel_3 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_3);
            if (channel_3 > channel_3_start)
                diff3 = channel_3 - channel_3_start;
            else if (channel_3 < channel_3_start)
                diff3 = (0xffff - channel_3_start) + channel_3;
        }
    }
}

```

```

diff3 = (0xffff - channel_3_start) + channel_3;

    __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_3,
                                  TIM_INPUTCHANNELPOLARITY_RISING);
    firt_capture3 = 0;
}
}

if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4) {
    if (firt_capture4 == 0) { //read
        channel_4_start = HAL_TIM_ReadCapturedValue(htim,
TIM_CHANNEL_4);
        firt_capture4 = 1;
        //Change the input capture mode to the falling edge of the
pulse
        __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4,
                                      TIM_INPUTCHANNELPOLARITY_FALLING);
    } else if (firt_capture4 == 1) {
        channel_4 = HAL_TIM_ReadCapturedValue(htim, TIM_CHANNEL_4);
        if (channel_4 > channel_4_start)
            diff4 = channel_4 - channel_4_start;
        else if (channel_4 < channel_4_start)
            diff4 = (0xffff - channel_4_start) + channel_4;

        __HAL_TIM_SET_CAPTUREPOLARITY(htim, TIM_CHANNEL_4,
                                      TIM_INPUTCHANNELPOLARITY_RISING);
        firt_capture4 = 0;
    }
}
}

//MPU6050
///////////////////////////////
///////////////////////////////
#define MPU6050_ADDR 0xD0
#define SMPLRT_DIV_REG 0x19
#define GYRO_CONFIG_REG 0x1B
#define ACCEL_CONFIG_REG 0x1C
#define ACCEL_XOUT_H_REG 0x3B
#define TEMP_OUT_H_REG 0x41
#define GYRO_XOUT_H_REG 0x43
#define PWR_MGMT_1_REG 0x6B
#define WHO_AM_I_REG 0x75

int16_t Accel_X_RAW = 0, Accel_Y_RAW = 0, Accel_Z_RAW = 0, Gyro_X_RAW = 0,
       Gyro_Y_RAW = 0, Gyro_Z_RAW = 0;
int16_t temp_raw = 0;
float Ax, Ay, Az, Gx, Gy, Gz, temp;

//the gyro offset
//int16_t os_Ax = 71, os_Ay = 96, os_Az = 4456, os_Gx = -299, os_Gy = -43,
os_Gz = 13; //1
//int16_t os_Ax = 79, os_Ay = 90, os_Az = 4463, os_Gx = -301, os_Gy = -43,
os_Gz = 13; //2
//int16_t os_Ax = -50, os_Ay = 115, os_Az = 4569, os_Gx = -302, os_Gy = -43,
os_Gz = 13; //3
int16_t os_Ax = 114, os_Ay = 56, os_Az = 4566, os_Gx = -300, os_Gy = -52,
       os_Gz = 11; //4
//int16_t os_Ax = 0, os_Ay = 0, os_Az = 0, os_Gx = 0, os_Gy = 0, os_Gz = 0;

int32_t ax_cal = 0, ay_cal = 0, az_cal = 0, gx_cal = 0, gy_cal = 0, gz_cal =
0;

void MPU6050_Init(void) {
    uint8_t check, Data;
}

```

```

// check device ID WHO_AM_I

HAL_I2C_Mem_Read(&hi2c1, MPU6050_ADDR, WHO_AM_I_REG, 1, &check, 1,
1000);

    if (check == 104) // 0x68 will be returned by the sensor if everything
goes well
    {
        // power management register 0X6B we should write all 0's to wake
the sensor up
        Data = 0;
        HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, PWR_MGMT_1_REG, 1, &Data,
1,
                           1000);           //done

        // Set Gyroscopic configuration in GYRO_CONFIG Register
        // XG_ST=0,YG_ST=0,ZG_ST=0, FS_SEL=1 -> ± 500 °/s
        Data = 0x08;
        HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, GYRO_CONFIG_REG, 1,
&Data, 1,
                           1000);
        // Set accelerometer configuration in ACCEL_CONFIG Register
        // XA_ST=0,YA_ST=0,ZA_ST=0, FS_SEL=3 -> ± 8g
        Data = 0x10;
        HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, ACCEL_CONFIG_REG, 1,
&Data, 1,
                           1000);

        Data = 0x03;
        HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, 0x1A, 1, &Data, 1, 1000);
    }
}

void MPU6050_Read(void) {
    uint8_t Rec_Data[14];

    HAL_I2C_Mem_Read(&hi2c1, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data,
14,
                           1000);

    Accel_X_RAW = (int16_t) Rec_Data[0] << 8 | Rec_Data[1];
    Accel_Y_RAW = (int16_t) Rec_Data[2] << 8 | Rec_Data[3];
    Accel_Z_RAW = (int16_t) Rec_Data[4] << 8 | Rec_Data[5];
    temp_raw = (int16_t) Rec_Data[6] << 8 | Rec_Data[7];
    Gyro_X_RAW = (int16_t) Rec_Data[8] << 8 | Rec_Data[9];
    Gyro_Y_RAW = (int16_t) Rec_Data[10] << 8 | Rec_Data[11];
    Gyro_Z_RAW = (int16_t) Rec_Data[12] << 8 | Rec_Data[13];

    Gyro_Y_RAW *= -1;
    Gyro_Z_RAW *= -1;

    Accel_X_RAW -= os_Ax;
    Accel_Y_RAW -= os_Ay;
    Accel_Z_RAW -= os_Az;
    Gyro_X_RAW -= os_Gx;
    Gyro_Y_RAW -= os_Gy;
    Gyro_Z_RAW -= os_Gz;
}

void offset_calculate() {
    for (int i = 0; i < 500; i++) {
        MPU6050_Read();
}

```

```

        HAL_Delay(4);
    }
    for (int j = 0; j < 3000; j++) {
        MPU6050_Read();
        ax_cal += Accel_X_RAW;
        ay_cal += Accel_Y_RAW;
        az_cal += Accel_Z_RAW;
        gx_cal += Gyro_X_RAW;
        gy_cal += Gyro_Y_RAW;
        gz_cal += Gyro_Z_RAW;
        HAL_Delay(4);
    }
    os_Ax = ax_cal / 3000;
    os_Ay = ay_cal / 3000;
    os_Az = az_cal / 3000;
    os_Gx = gx_cal / 3000;
    os_Gy = gy_cal / 3000;
    os_Gz = gz_cal / 3000;
}

//PID gain and limit settings
#define roll      0
#define pitch     1
#define yaw       2
// ----- Global variables used for PID controller -----
-
float pid_set_points[3] = { 0, 0, 0 }; // roll, pitch, yaw
int pid_max[3] = { 180, 180, 20 };           // roll, pitch, yaw
// Errors
float errors[3]; // Measured errors (compared to instructions) : [roll,
pitch, yaw]
float delta_err[3] = { 0, 0, 0 }; // Error deltas in that order : roll,
pitch, yaw
float error_sum[3] = { 0, 0, 0 }; // Error sums (used for integral component)
: [roll, pitch, yaw]
float previous_error[3] = { 0, 0, 0 }; // Last errors (used for derivative
component) : [roll, pitch, yaw]
// PID coefficients
float Kp[3] = { 0.4, 0.4, 0.4 };           // P: roll, pitch, yaw
//float Ki[3] = { 0.15, 0.15, 0.02};          // I: roll, pitch, yaw
float Ki[3] = { 0.0, 0.0, 0.0 };
float Kd[3] = { 0.01, 0.01, 0 };           // D: roll, pitch, yaw

float pid_roll = 0;
float pid_pitch = 0;
float pid_yaw = 0;
// -----
-
// ----- MPU variables -----
-
// The RAW values got from gyro (in °/sec) in that order: X, Y, Z
int gyro_raw[3] = { 0, 0, 0 };
// Average gyro offsets of each axis in that order: X, Y, Z
long gyro_offset[3] = { 0, 0, 0 };
// Calculated angles from gyro's values in that order: X, Y, Z
float gyro_angle[3] = { 0, 0, 0 };
// The RAW values got from accelerometer (in m/sec2) in that order: X, Y, Z
int acc_raw[3] = { 0, 0, 0 };
// Calculated angles from accelerometer's values in that order: X, Y, Z
float acc_angle[3] = { 0, 0, 0 };
// Total 3D acceleration vector in m/s2
long acc_total_vector;

```

```

// Calculated angular motion on each axis: roll, pitch, yaw
float angular_motions[3] = { 0, 0, 0 };
float pre_angular_motions[3] = { 0, 0, 0 };
float measures[3] = { 0, 0, 0 };

uint8_t active, first_angle = 0;
int16_t esc1, esc2, esc3, esc4, throttle;
uint32_t looptimer;
float pitch_adjust, roll_adjust, pitch_angle, roll_angle;

//KALMAN-----
-----

float Kalman_angle_pitch_acc, Kalman_angle_roll_acc, Kalman_Gx, Kalman_Gy,
    Kalman_Gz;

float _err_measure;
float _err_estimate;
float _q;
float _current_estimate;
float _last_estimate = 0;
float _kalman_gain;
void KalmanFilterX(float mea_e, float est_e, float q) {
    _err_measure = mea_e;
    _err_estimate = est_e;
    _q = q;
}
float updateEstimateX(float mea) {
    _kalman_gain = _err_estimate / (_err_estimate + _err_measure);
    _current_estimate = _last_estimate + _kalman_gain * (mea -
    _last_estimate);
    _err_estimate = (1.0 - _kalman_gain) * _err_estimate
        + fabs(_last_estimate - _current_estimate) * _q;
    _last_estimate = _current_estimate;
    return _current_estimate;
}

float _err_measure1;
float _err_estimate1;
float _q1;
float _current_estimate1;
float _last_estimate1 = 0;
float _kalman_gain1;
void KalmanFilterY(float mea_e1, float est_e1, float q1) {
    _err_measure1 = mea_e1;
    _err_estimate1 = est_e1;
    _q1 = q1;
}
float updateEstimateY(float meal) {
    _kalman_gain1 = _err_estimate1 / (_err_estimate1 + _err_measure1);
    _current_estimate1 = _last_estimate1
        + _kalman_gain1 * (meal - _last_estimate1);
    _err_estimate1 = (1.0 - _kalman_gain1) * _err_estimate1
        + fabs(_last_estimate1 - _current_estimate1) * _q1;
    _last_estimate1 = _current_estimate1;
    return _current_estimate1;
}

float _err_measure2;
float _err_estimate2;
float _q2;
float _current_estimate2;
float _last_estimate2 = 0;
float _kalman_gain2;

```

```

void KalmanFilterGX(float mea_e2, float est_e2, float q2) {
    _err_measure2 = mea_e2;
    _err_estimate2 = est_e2;
    _q2 = q2;
}
float updateEstimateGX(float mea2) {
    _kalman_gain2 = _err_estimate2 / (_err_estimate2 + _err_measure2);
    _current_estimate2 = _last_estimate2
        + _kalman_gain2 * (mea2 - _last_estimate2);
    _err_estimate2 = (1.0 - _kalman_gain2) * _err_estimate2
        + fabs(_last_estimate2 - _current_estimate2) * _q2;
    _last_estimate2 = _current_estimate2;
    return _current_estimate2;
}

float _err_measure3;
float _err_estimate3;
float _q3;
float _current_estimate3;
float _last_estimate3 = 0;
float _kalman_gain3;
void KalmanFilterGY(float mea_e3, float est_e3, float q3) {
    _err_measure3 = mea_e3;
    _err_estimate3 = est_e3;
    _q3 = q3;
}
float updateEstimateGY(float mea3) {
    _kalman_gain3 = _err_estimate3 / (_err_estimate3 + _err_measure3);
    _current_estimate3 = _last_estimate3
        + _kalman_gain3 * (mea3 - _last_estimate3);
    _err_estimate3 = (1.0 - _kalman_gain3) * _err_estimate3
        + fabs(_last_estimate3 - _current_estimate3) * _q3;
    _last_estimate3 = _current_estimate3;
    return _current_estimate3;
}

float _err_measure4;
float _err_estimate4;
float _q4;
float _current_estimate4;
float _last_estimate4 = 0;
float _kalman_gain4;
void KalmanFilterGZ(float mea_e4, float est_e4, float q4) {
    _err_measure4 = mea_e4;
    _err_estimate4 = est_e4;
    _q4 = q4;
}
float updateEstimateGZ(float mea4) {
    _kalman_gain4 = _err_estimate4 / (_err_estimate4 + _err_measure4);
    _current_estimate4 = _last_estimate4
        + _kalman_gain4 * (mea4 - _last_estimate4);
    _err_estimate4 = (1.0 - _kalman_gain4) * _err_estimate4
        + fabs(_last_estimate4 - _current_estimate4) * _q4;
    _last_estimate4 = _current_estimate4;
    return _current_estimate4;
}

float _err_measureEscl;
float _err_estimateEscl;
float _qEscl;
float _current_estimateEscl;
float _last_estimateEscl = 0;
float _kalman_gainEscl;

```

```

void Kalman_esc1(float mea_eEsc1, float est_eEsc1, float qEsc1) {
    _err_measureEsc1 = mea_eEsc1;
    _err_estimateEsc1 = est_eEsc1;
    _qEsc1 = qEsc1;
}
float update_esc1(float meaEsc1) {
    _kalman_gainEsc1 = _err_estimateEsc1
        / (_err_estimateEsc1 + _err_measureEsc1);
    _current_estimateEsc1 = _last_estimateEsc1
        + _kalman_gainEsc1 * (meaEsc1 - _last_estimateEsc1);
    _err_estimateEsc1 = (1.0 - _kalman_gainEsc1) * _err_estimateEsc1
        + fabs(_last_estimateEsc1 - _current_estimateEsc1) *
    _qEsc1;
    _last_estimateEsc1 = _current_estimateEsc1;
    return _current_estimateEsc1;
}

float _err_measureEsc2;
float _err_estimateEsc2;
float _qEsc2;
float _current_estimateEsc2;
float _last_estimateEsc2 = 0;
float _kalman_gainEsc2;
void Kalman_esc2(float mea_eEsc2, float est_eEsc2, float qEsc2) {
    _err_measureEsc2 = mea_eEsc2;
    _err_estimateEsc2 = est_eEsc2;
    _qEsc2 = qEsc2;
}
float update_esc2(float meaEsc2) {
    _kalman_gainEsc2 = _err_estimateEsc2
        / (_err_estimateEsc2 + _err_measureEsc2);
    _current_estimateEsc2 = _last_estimateEsc2
        + _kalman_gainEsc2 * (meaEsc2 - _last_estimateEsc2);
    _err_estimateEsc2 = (1.0 - _kalman_gainEsc2) * _err_estimateEsc2
        + fabs(_last_estimateEsc2 - _current_estimateEsc2) *
    _qEsc2;
    _last_estimateEsc2 = _current_estimateEsc2;
    return _current_estimateEsc2;
}

float _err_measureEsc3;
float _err_estimateEsc3;
float _qEsc3;
float _current_estimateEsc3;
float _last_estimateEsc3 = 0;
float _kalman_gainEsc3;
void Kalman_esc3(float mea_eEsc3, float est_eEsc3, float qEsc3) {
    _err_measureEsc3 = mea_eEsc3;
    _err_estimateEsc3 = est_eEsc3;
    _qEsc3 = qEsc3;
}
float update_esc3(float meaEsc3) {
    _kalman_gainEsc3 = _err_estimateEsc3
        / (_err_estimateEsc3 + _err_measureEsc3);
    _current_estimateEsc3 = _last_estimateEsc3
        + _kalman_gainEsc3 * (meaEsc3 - _last_estimateEsc3);
    _err_estimateEsc3 = (1.0 - _kalman_gainEsc3) * _err_estimateEsc3
        + fabs(_last_estimateEsc3 - _current_estimateEsc3) *
    _qEsc3;
    _last_estimateEsc3 = _current_estimateEsc3;
    return _current_estimateEsc3;
}

```

```

float _err_measureEsc4;
float _err_estimateEsc4;
float _qEsc4;
float _current_estimateEsc4;
float _last_estimateEsc4 = 0;
float _kalman_gainEsc4;
void Kalman_esc4(float mea_eEsc4, float est_eEsc4, float qEsc4) {
    _err_measureEsc4 = mea_eEsc4;
    _err_estimateEsc4 = est_eEsc4;
    _qEsc4 = qEsc4;
}
float update_esc4(float meaEsc4) {
    _kalman_gainEsc4 = _err_estimateEsc4
        / (_err_estimateEsc4 + _err_measureEsc4);
    _current_estimateEsc4 = _last_estimateEsc4
        + _kalman_gainEsc4 * (meaEsc4 - _last_estimateEsc4);
    _err_estimateEsc4 = (1.0 - _kalman_gainEsc4) * _err_estimateEsc4
        + fabs(_last_estimateEsc4 - _current_estimateEsc4) *
    _qEsc4;
    _last_estimateEsc4 = _current_estimateEsc4;
    return _current_estimateEsc4;
}
/////////////////////////////////////////////////////////////////
/* USER CODE END 0 */

/***
 * @brief  The application entry point.
 * @retval int
 */
int main(void) {
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/
    /* Reset of all peripherals, Initializes the Flash interface and the
    Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_I2C1_Init();
    MX_TIM2_Init();
    MX_TIM3_Init();
    MX_TIM4_Init();
    /* USER CODE BEGIN 2 */
    MPU6050_Init();

    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
}

```

```

//offset_calculate();

for (int i = 0; i < 20; i++) {
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
    HAL_Delay(100);
}

HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_1);
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_2);
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_3);
HAL_TIM_IC_Start_IT(&htim2, TIM_CHANNEL_4);
//wait for connection tx-rx
while (diff1 < 990 || diff2 < 990 || diff3 < 990 || diff4 < 990) {
}
while (diff3 < 990 || diff3 > 1050) {
}

HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);

//-----
//esc calibration
__HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 2000); //fl
__HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 2000); //fr
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, 2000); //rr
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 2000); //rl

HAL_Delay(50);

__HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 1000); //fl
__HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 1000); //fr
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, 1000); //rr
__HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 1000); //rl

HAL_Delay(300);
//-----

HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET);

for (int i = 0; i < 20; i++) {
    HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13);
    HAL_Delay(100);
}

looptimer = HAL_GetTick();

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1) {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    MPU6050_Read(); //ok
    // Apply low-pass filter (10Hz cutoff frequency)
    angular_motions[roll] = 0.7 * angular_motions[roll]
        + ((float) Gyro_X_RAW * 0.3 / 65.5); //Gyro pid input
    angular_motions[pitch] = 0.7 * angular_motions[pitch]
        + ((float) Gyro_Y_RAW * 0.3 / 65.5);
}

```

```

angular_motions[yaw] = 0.7 * angular_motions[yaw]
                      + ((float) Gyro_Z_RAW * 0.3 / 65.5);

// Angle calculation using integration
gyro_angle[roll] += (float) (Gyro_X_RAW / (250 * 65.5));
gyro_angle[pitch] += (float) (Gyro_Y_RAW / (250 * 65.5));

// Transfer roll to pitch if IMU has yawed
gyro_angle[pitch] -= gyro_angle[roll]
                     * sin((float) Gyro_Z_RAW * (M_PI / (250 * 65.5 *
180)));
gyro_angle[roll] += gyro_angle[pitch]
                     * sin((float) Gyro_Z_RAW * (M_PI / (250 * 65.5 *
180)));

//Accelerometer angle calculations
//acc_total_vector = sqrt(pow(Accel_X_RAW, 2) + pow(Accel_Y_RAW,
2) + pow(Accel_Z_RAW, 2));
acc_total_vector = sqrt(pow(os_Ax, 2) + pow(os_Ay, 2) +
pow(os_Az, 2));
if (Accel_X_RAW > acc_total_vector)
    Accel_X_RAW = acc_total_vector; //Limit the maximum
accelerometer value.
if (Accel_X_RAW < -acc_total_vector)
    Accel_X_RAW = -acc_total_vector;
if (Accel_Y_RAW > acc_total_vector)
    Accel_Y_RAW = acc_total_vector; //Limit the maximum
accelerometer value.
if (Accel_Y_RAW < -acc_total_vector)
    Accel_Y_RAW = -acc_total_vector;

acc_angle[roll] = asin((float) Accel_X_RAW / acc_total_vector)
                  * (-180 / M_PI); //Calculate the pitch angle.
acc_angle[pitch] = asin((float) Accel_Y_RAW / acc_total_vector)
                  * (180 / M_PI); //Calculate the roll angle.

gyro_angle[roll] = gyro_angle[roll] * 0.995 + acc_angle[roll] *
0.005;
gyro_angle[pitch] = gyro_angle[pitch] * 0.995
                    + acc_angle[pitch] * 0.005;

// To dampen the pitch and roll angles a complementary filter is
used
measures[roll] = measures[roll] * 0.8 + gyro_angle[roll] * 0.2;
measures[pitch] = measures[pitch] * 0.8 + gyro_angle[pitch] *
0.2;
measures[yaw] = Gyro_Z_RAW / 65.5; //Store the angular motion for
this axis

pitch_adjust = measures[pitch] * 15;
roll_adjust = measures[roll] * 15;

pitch_angle = measures[pitch];
roll_angle = measures[roll];

//active
if (diff3 < 1050 && diff4 < 1050)
    active = 1;
if (active == 1 && diff3 < 1050 && diff4 > 1450) {
    active = 2;
    //When this is the first time.
    gyro_angle[roll] = acc_angle[roll];
    gyro_angle[pitch] = acc_angle[pitch];
}

```

```

//Reset the PID controllers for a bumpless start.
errors[0] = 0;
errors[1] = 0;
errors[2] = 0;
delta_err[0] = 0;
delta_err[1] = 0;
delta_err[2] = 0;
error_sum[0] = 0;
error_sum[1] = 0;
error_sum[2] = 0;
previous_error[0] = 0;
previous_error[1] = 0;
previous_error[2] = 0;
pid_roll = 0;
pid_pitch = 0;
pid_yaw = 0;
}
//Stopping the motors: throttle low and yaw right.
if (active == 2 && diff3 < 1050 && diff4 > 1950)
    active = 0;

pid_set_points[roll] = 0;
pid_set_points[pitch] = 0;
pid_set_points[yaw] = 0;

if (diff1 > 1525)
    pid_set_points[roll] = diff1 - 1525;
else if (diff1 < 1490)
    pid_set_points[roll] = diff1 - 1490;
pid_set_points[roll] = (pid_set_points[roll] - roll_adjust) /
3.0;

if (diff2 > 1525)
    pid_set_points[pitch] = diff2 - 1525;
else if (diff2 < 1490)
    pid_set_points[pitch] = diff2 - 1490;
pid_set_points[pitch] = (pid_set_points[pitch] - pitch_adjust) /
3.0;

if (diff3 > 1150) { //Do not yaw when turning off the motors.
    if (diff4 > 1550)
        pid_set_points[yaw] = (diff4 - 1550) / 3.0;
    else if (diff4 < 1350)
        pid_set_points[yaw] = (diff4 - 1350) / 3.0;
}

// Calculate current errors
errors[roll] = angular_motions[roll] - pid_set_points[roll];
errors[pitch] = angular_motions[pitch] - pid_set_points[pitch];
errors[yaw] = angular_motions[yaw] - pid_set_points[yaw];

// Calculate sum of errors : Integral coefficients
error_sum[roll] += errors[roll];
error_sum[pitch] += errors[pitch];
error_sum[yaw] += errors[yaw];

// Keep values in acceptable range
if (error_sum[roll] < (-180 / Ki[roll]))
    error_sum[roll] = -180 / Ki[roll];
if (error_sum[roll] > 180 / Ki[roll])
    error_sum[roll] = 180 / Ki[roll];
if (error_sum[pitch] < (-180 / Ki[pitch]))

```

```

        error_sum[pitch] = -180 / Ki[pitch];
    if (error_sum[pitch] > 180 / Ki[pitch])
        error_sum[pitch] = 180 / Ki[pitch];
    if (error_sum[yaw] < (-180 / Ki[yaw]))
        error_sum[yaw] = -180 / Ki[yaw];
    if (error_sum[yaw] > 180 / Ki[yaw])
        error_sum[yaw] = 180 / Ki[yaw];

    // Calculate error delta : Derivative coefficients
    delta_err[roll] = angular_motions[roll] -
pre_angular_motions[roll];
    delta_err[pitch] = angular_motions[pitch] -
pre_angular_motions[pitch];
    delta_err[yaw] = angular_motions[yaw] - pre_angular_motions[yaw];

    // Save current error as previous_error for next time
    pre_angular_motions[roll] = angular_motions[roll];
    pre_angular_motions[pitch] = angular_motions[pitch];
    pre_angular_motions[yaw] = angular_motions[yaw];

    throttle = diff3;

    if (throttle > 1050) {
        // PID = e.Kp + ∫e.Ki + Δe.Kd
        pid_roll = (errors[roll] * Kp[roll])
            + (error_sum[roll] * Ki[roll]) * 0.004
            + (delta_err[roll] * Kd[roll]) / 0.004;
        pid_pitch = (errors[pitch] * Kp[pitch])
            + (error_sum[pitch] * Ki[pitch]) * 0.004
            + (delta_err[pitch] * Kd[pitch]) / 0.004;
        pid_yaw = (errors[yaw] * Kp[yaw])
            + (error_sum[yaw] * Ki[yaw]) * 0.004
            + (delta_err[yaw] * Kd[yaw]) / 0.004;

        if (pid_roll > pid_max[roll])
            pid_roll = pid_max[roll];
        if (pid_roll < (-pid_max[roll]))
            pid_roll = -pid_max[roll];
        if (pid_pitch > pid_max[pitch])
            pid_pitch = pid_max[pitch];
        if (pid_pitch < (-pid_max[pitch]))
            pid_pitch = -pid_max[pitch];
        if (pid_yaw > pid_max[yaw])
            pid_yaw = pid_max[yaw];
        if (pid_yaw < (-pid_max[yaw]))
            pid_yaw = -pid_max[yaw];
    }

    if (active == 2) {
        if (throttle > 1800)
            throttle = 1800;

        esc1 = throttle + pid_roll + pid_pitch + pid_yaw;
//Calculate the pulse for esc 1 (front-right - CCW).
        esc2 = throttle - pid_roll + pid_pitch - pid_yaw;
//Calculate the pulse for esc 2 (rear-right - CW).
        esc3 = throttle - pid_roll - pid_pitch + pid_yaw;
//Calculate the pulse for esc 3 (rear-left - CCW).
        esc4 = throttle + pid_roll - pid_pitch - pid_yaw;
//Calculate the pulse for esc 4 (front-left - CW).

        //
        esc1 = update_esc1(esc1);
        esc2 = update_esc1(esc2);
    }
}

```

```

//          esc3 = update_esc1(esc3);
//          esc4 = update_esc1(esc4);

          if (esc1 < 1200)
              esc1 = 1200; //Keep the motors running.
          if (esc2 < 1200)
              esc2 = 1200; //Keep the motors running.
          if (esc3 < 1200)
              esc3 = 1200; //Keep the motors running.
          if (esc4 < 1200)
              esc4 = 1200; //Keep the motors running.

          if (esc1 > 2000)
              esc1 = 2000; //Limit the esc-1 pulse to 2000us.
          if (esc2 > 2000)
              esc2 = 2000; //Limit the esc-2 pulse to 2000us.
          if (esc3 > 2000)
              esc3 = 2000; //Limit the esc-3 pulse to 2000us.
          if (esc4 > 2000)
              esc4 = 2000; //Limit the esc-4 pulse to 2000us.

} else {
    esc1 = 1000;
    esc2 = 1000;
    esc3 = 1000;
    esc4 = 1000;
}

//fr __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, esc1);
//fl __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, esc4);
//br __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, esc2);
//bl __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, esc3);

if (HAL_GetTick() - looptimer > 4) {
    HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_3, 1000);
//fr __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 1000);
//fl __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, 1000);
//br __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 1000);
//bl
}
while (HAL_GetTick() - looptimer < 4) {
}
looptimer = HAL_GetTick();
}

/* USER CODE END 3 */
}

```