BỘ GIÁO DỤC VÀ ĐÀO TẠO BỘ GIAO THÔNG VẬN TẢI

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP.HCM

---------o0o---------

NGUYỄN VĂN VŨ

ĐIỀU KHIỂN MÔ HÌNH BAY BỐN CÁNH TỰ CÂN BẰNG VÀ ỔN ĐỊNH BẰNG PHƯƠNG PHÁP ĐIỀU KHIỂN PID KẾT HỢP VỚI PHƯƠNG PHÁP LQR VÀ BỘ LỌC KALMAN

NGÀNH: KỸ THUẬT ĐIỀU KHIỂN VÀ TỰ ĐỘNG HOÁ

MÃ SỐ: 60520205

LUẬN VĂN THẠC SĨ KỸ THUẬT

NGƯỜI HƯỚNG DẪN KHOA HỌC:
GVC.TS. HOÀNG MINH TRÍ

TP. HCM Năm 2016

LUẬN VĂN ĐƯỢC HOÀN THÀNH TẠI

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI THÀNH PHỐ HỒ CHÍ MINH

Cán bộ hướng dẫn khoa học : GVC TS. HOÀNG MINH TRÍ

Cán bộ chấm nhận xét 1 : TS NGÔ MẠNH DŨNG

Cán bộ chấm nhận xét 2 : TS ĐẶNG XUÂN KIÊN

Luận văn thạc sĩ được bảo vệ tại Trường Đại học Giao thông vận tải Tp. HCM ngày 07 tháng 07 năm 2016

Thành phần Hội đồng đánh giá luận văn thạc sĩ gồm:

(Ghi rõ họ, tên, học hàm, học vị của Hội đồng chấm bảo vệ luận văn thạc sĩ)

1.	PGS TS NGUYỄN HỮU KHƯƠNG	Chủ tịch Hội đồng;
2.	TS NGÔ MẠNH DŨNG	Ủy viên, phản biện;
3.	TS ĐẶNG XUÂN KIÊN	Ủy viên, phản biện;
4.	PGS TS NGUYỄN THANH PHƯƠNG	Ủy viên, thư ký;
5.	TS VÕ CÔNG PHƯƠNG	Ủy viên.

Xác nhận của Chủ tịch Hội đồng đánh giá luận văn và Trưởng Khoa quản lý chuyên ngành sau khi luận văn đã được sửa chữa.

CHỦ TỊCH HỘI ĐỒNG **TRƯỞNG KHOA** Điện - ĐT VT

TS. Võ Công Phương

LỜI CẢM ƠN

Bản luận văn này được hoàn thành trên cơ sở những kết quả nghiên cứu của tôi, dưới sự hướng dẫn tận tình của GVC.TS. Hoàng Minh Trí, Đại Học Bách Khoa Thành phố Hồ Chí Minh. Tôi xin được bày tỏ lòng biết ơn sâu sắc đối với thầy hướng dẫn. Thầy đã tin tưởng, quan tâm, giúp đỡ và tạo mọi điều kiện cho tôi hoàn thành luận văn này – một cơ hội lớn trong cuộc đời.

Tôi xin chân thành cảm ơn TS. Võ Công Phương - Trưởng Khoa cùng với các Thầy, Cô thuộc bộ môn điều khiển tự động, Khoa Điện-Điện Tử Viễn Thông, Đại học Giao Thông Vận Tải TP.Hồ Chí Minh, về những ý kiến đóng góp và sự ủng hộ nhiệt tình.

Trân trọng cảm ơn Viện Đào tạo sau đại học, Đại học Giao Thông Vận Tải TP.Hồ Chí Minh, đã tạo mọi điều kiện thuận lợi nhất cho tôi thực hiện luận văn.

Tôi xin cảm ơn các thầy, cô giáo, bạn bè và đồng nghiệp luôn giúp đỡ, động viên và chia sẻ khó khăn để tôi hoàn thành tốt nhất luận văn.

Tác giả luận văn

Nguyễn Văn Vũ

MỤC LỤC

DANH MỤC CÁC HÌNH

DANH MỤC BẢNG

KÍ HIỆU KHOA HỌC

F_1	: Lực tạo ra của motor thứ 1
F_2	: Lực tạo ra của motor thứ 2
F_3	: Lực tạo ra của motor thứ 3
F_4	: Lực tạo ra của motor thứ 4
F	: Lực tổng hợp của bốn động cơ
I_x	: Moment quán tính trên phương trục x
I_y	: Moment quán tính trên phương trục y
I_y	: Moment quán tính trên phương trục z
PWM_1	: Tín hiệu điều khiển PWM của motor thứ 1
PWM_2	: Tín hiệu điều khiển PWM của motor thứ 2
PWM_3	: Tín hiệu điều khiển PWM của motor thứ 3
PWM_4	: Tín hiệu điều khiển PWM của motor thứ 4
$T_\emptyset$	: Moment trên phương trục x
T_θ	: Moment trên phương trục y
T_ψ	: Moment trên phương trục z
R	: Ma trận xoay góc Euler (ma trận cosine)
$(x,y,z)_b$	: Hệ tọa độ trên thân máy bay
$(x,y,x)_E$	: Hệ tọa độ tham chiếu mặt đất
u	: Vận tóc máy bay theo phương x
v	: Vận tóc máy bay theo phương y
w	: Vận tóc máy bay theo phương z
p	: Vận tốc góc trên trục x
q	: Vận tốc góc trên trục y
r	: Vận tốc góc trên trục z
$\emptyset$	: Góc Roll
θ	: Góc pitch
ψ	: Góc yaw

DANH MỤC CÁC TỪ VIẾT TẮT

UAS	Uninhabited Aerial System
PID	Proportional Integral Derivative
LQR	Linear- Quadratic Regulator
ESC	Electronic Speed Controller
PWM	Pulse Width Modulation
BLDC	Brushless DC motor
IMU	Inertial Measurement Unit.

Chương 1
TỔNG QUAN

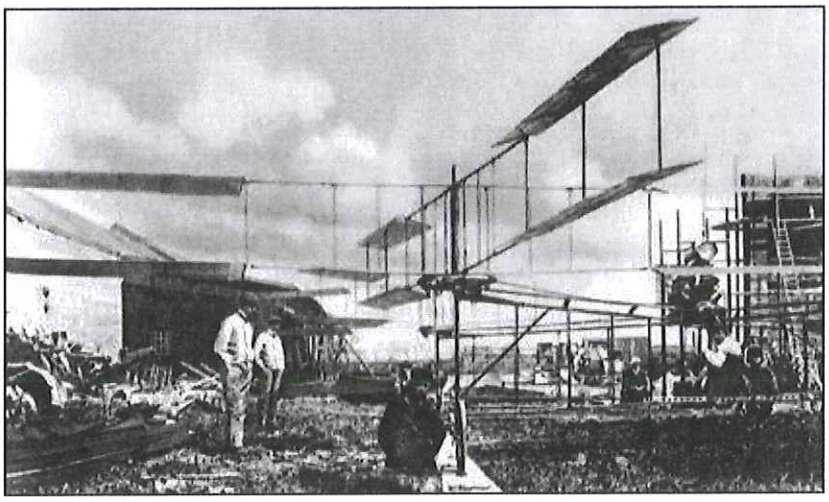
1.1 Tổng quan về lĩnh vực nghiên cứu

Ngày nay, thiết bị bay không người lái (Uninhabited Aerial System: UAS) ngày càng phát triển mạnh trong lĩnh vực nghiên cứu khoa học, trong đó Quadrotor là một trong những UAS được quan tâm nghiên cứu. Phương tiện này có thể được điều khiển từ xa bởi người điều hành trên mặt đất hoặc bay tự động thông qua một đường bay được lập trình từ trước. Mô hình Quadrotor được lắp ráp có cấu trúc cứng, nhẹ và gọn, được thiết kế trong một khối khép kín, giá trị của nó được khẳng định qua nhiều ứng dụng thực tế hiện nay như:

- Dùng tìm kiếm cứu nạn trong môi trường đồi núi, nơi nguy hiểm cho con người.
- Dùng trong khảo sát, thiết lập bản đồ.
- Dùng trong lĩnh vực an ninh, quân sự
- Dùng trong lĩnh vực giải trí, quảng cáo…

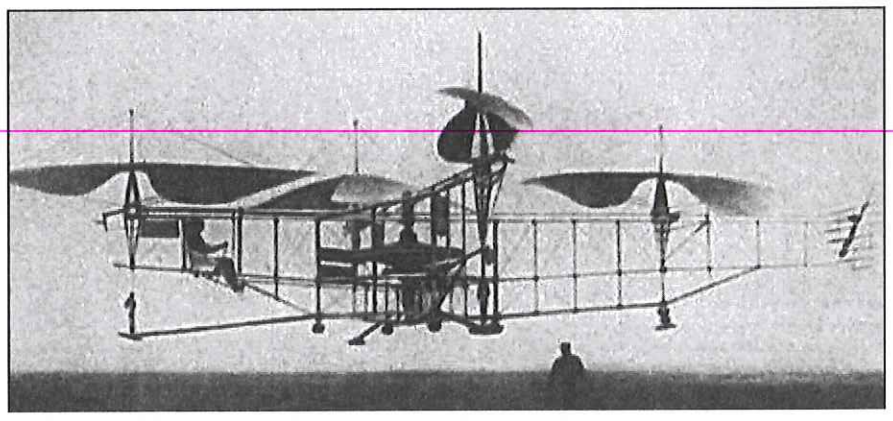
1.2 Lịch sử phát triển

UAS được nghiên cứu từ rất lâu. Chiếc Quadrotor đầu tiên trên thế giới ra đời vào năm 1907 do hai anh em nhà khoa học người pháp Charles Richet và Charles Breguet chế tạo, có tên là " Breguet-Richet Gyroplane No.1". Nó chỉ có một động cơ đốt trong 40/45 HP (1HP = 750W) và được dùng điều khiển các cánh quạt của Quadrotor thông qua hệ thống truyền động bằng dây đai và pu-li. Bộ khung của Quadrotor này được làm bằng ống thép, có tổng trọng lượng khoảng 500 kg. Breguet-Richet Gyroplane No.1 được thực hiện bay thử đầu tiên vào tháng 9 năm 1907 tại Douai-Pháp, lần này nó chỉ bay được vài phút ở độ cao 1,5 mét [12].



Hình 1-1: Breguet-Richet Gyroplane No.1

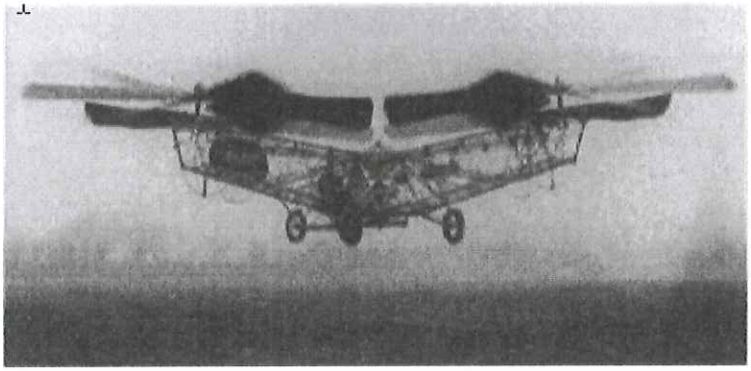
Vào năm 1920, một kỹ sư người Pháp tên Etienne Oemichen cũng đã nghiên cứu và cho ra đời một Quadrotor mang tên ông, Oemichen No.2 như hình 1-2. Nó gồm có bốn động cơ và được bố trí cân bằng tại bốn góc của một khung hình chữ thập. Ban đầu nó được gắn thêm một khí cầu để nâng và giữ ổn định cho cổ máy này. Oemichen No.2 có kết quả khả quan hơn Breguet-Richet Gyroplane No.1 trước đó. Oemichen No.2 đã được thử nghiệm rất nhiều lần và có khả năng bay trong vòng một dặm (01 dặm = 1.6 Km), có thể đạt được 300 mét độ cao [12].



Hình 1-2: Oemichen No.2

Năm 1922, Georges De Bothezat và Evan Jerome thành công khi thiết kế chiếc Quadrotor khổng lồ phục vụ cho quân đội Mỹ. Cỗ máy này đã được điều khiển bằng cách thay đổi đơn lẻ cùng lúc các góc xoắn của cánh quạt. Ngoài ra

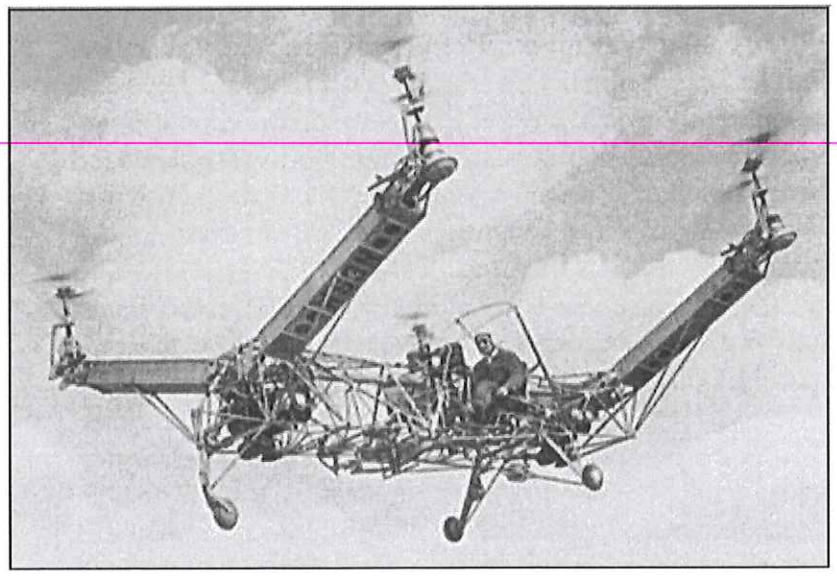
nó còn được gắn thêm 4 cánh quạt nhỏ để điều khiển. Chiếc Quadrotor này được đặt tên là "Bạch tuộc bay".



Hình 1-3: Bạch tuộc bay

Từ đó, Quadrotor ít được chú ý hơn trước. Cho đến những năm 1980, nó được chú ý trở lại với kết cấu được thiết kế đơn giản, linh động, khả năng mang tải cao và giá thành thấp.

Ngày nay, một số công ty ở nước ngoài như công ty Parrot, công ty Draganfty Innovations, chuyên nghiên cứu và phát triển Quadrotor nhằm đáp ứng các nhu cầu của xã hội: phục vụ trong thám hiểm, tìm kiếm cứu nạn, khảo sát địa hình, giải trí, quảng cáo… Sau đây là một số hình ảnh của Quadrotor trên thế giới đã được sản xuất trong những năm qua [14]:



Hình 1-4: Convertawings Model A, 1955/1956

Hình 1-5: Curtiss-Wright VZ-7, 1958

Hình 1-6: AR.Drone của hãng Parrot

Hình 1-7: Quadrotor MD4-200 của Microdrone

Hình 1-8: X-Pro của hãng Draganfly Innovations

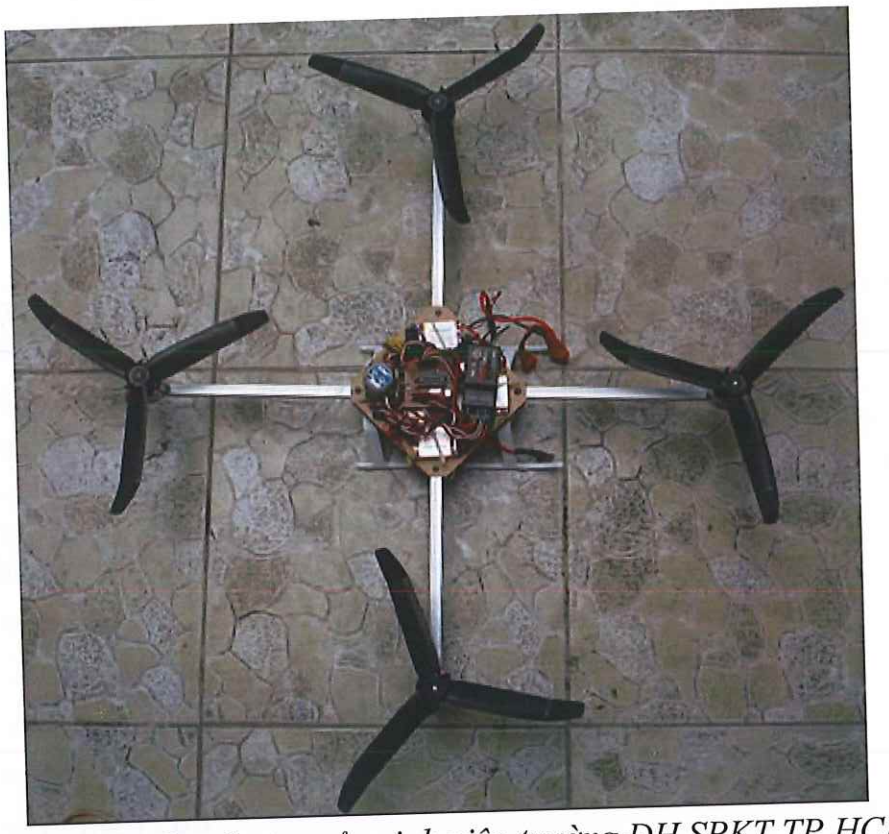
Công việc nghiên cứu về Quadrotor đã được quan tâm từ lâu bởi các nhóm nghiên cứu khác nhau trên thế giới. Và thực tế đã chứng tỏ rằng Quadrotor là một lựa chọn cho các ứng dụng của UAS so với máy bay trực thăng thông thường. Nó đã mang lại cho các nhà khoa học thêm nhiều lựa chọn cho công việc nghiên cứu.

Quadrotor phát triển mạnh song song với sự phát triển vượt bậc của công nghệ vi điện tử. Sự phát triển này đã mang lại nhiều lợi ích cho xã hội, giá thành luôn được giảm xuống trong khi chất lượng lại được nâng lên, một bài toán muôn thuở tưởng chừng không thể giải quyết được. Ngày nay, sinh viên trong các trường đại học cũng rất quan tâm, nhiệt tình nghiên cứu trong lĩnh vực này.

Trong những năm gần đây, sinh viên các trường đại học trong nước cũng đã quan tâm nghiên cứu Quadrotor điều khiển từ xa, bằng sóng RF như: Đại học Sư phạm kỹ thuật TP HCM, đại học Bách khoa TP HCM, Đại học Bách khoa Đà Nẵng, Đại học Giao thông vận tải TP HCM…Sau đây là một số hình ảnh Quadrotor do sinh viên Việt Nam nghiên cứu:

Hình 1-9. Mô hình bay Quadrotor điều khiển từ xa (nhóm BKIT4U) – Ngày hội sáng tạo trẻ ĐHQG Tp.HCM

Hình 1-10. Quadrotor điều khiển từ xa của sinh viên trường ĐH GTVT.



Hình 1-11. Quadrotor của sinh viên trường ĐH SPKT TP.HCM

1.3 Mục đích nghiên cứu

Khảo sát mô hình vật lý, kết hợp với việc xây dựng mô hình và giải thuật điều khiển từ đó đưa ra cơ sở lý thuyết phục vụ cho việc tính toán, thiết kế phần cứng và phần mềm điều khiển.

Thực hiện mô phỏng mô hình Quadrotor, đánh giá khả năng tự cân bằng của mô hình khi dùng phương pháp điều khiển PID kết hợp với phương pháp tuyến tính hóa LQR (Linear quadratic regulator) và bộ lọc Kalman.

1.4 Phương pháp nghiên cứu

Nghiên cứu các tài liệu và các thiết kế có sẵn trong và ngoài nước. Tính toán thiết kế mô hình hóa và mô phỏng, sau đó thiết kế mô hình tổng hợp hoàn chỉnh dựa trên mô hình ảo, sau đó thử nghiệm, đánh giá và hiệu chỉnh, cụ thể như sau:

- Nghiên cứu các loại cảm biến như: Gyroscope (cảm biến góc quay), accelerometer (cảm biến gia tốc), magnetometer (cảm biến từ trường), cảm biến siêu âm…và các ứng dụng trong thiết kế Quadrotor.
- Dựa vào các thành phần như động cơ, cánh quạt, vị trí bố trí động cơ, trọng lượng, kích thước của hệ thống từ đó phân tích các yếu tố, các thông số ảnh hưởng đến sự chuyển động của Quadrotor và rút ra các tham số cần thiết của hệ thống.
- Sử dụng các kiến thức toán học để xây dựng các phương trình toán tượng trưng có thể coi như mô hình thực và có thể dùng trong môi trường mô phỏng.
- Kết hợp phương pháp điều khiển đã chọn là PID và LQR, lọc Kalman, các thuật toán này có nhiệm vụ nhận và xử lý các thông tin nhận được từ các cảm biến gia tốc, cảm biến góc quay, cảm biến từ trường, cảm biến độ cao, từ đó tính toán, ước lượng kết quả của hệ thống và đưa ra tác vụ điều khiển tích hợp nhằm thực hiện điều khiển được tốt nhất, tối ưu nhất.

Các phần mềm hỗ trợ trong nghiên cứu như sau:

- Matlab Simulink
- IDE Arduino
- Microsoft Visual Studio

1.5 Phạm vi nghiên cứu

Nghiên cứu về Quadrotor là một đề tài khó, đòi hỏi kiến thức tổng hợp ở nhiều lĩnh vực như: thiết kế cơ khí, động lực học, các phương pháp điều khiển, mạch điện tử, truyền thông và lập trình điều khiển…

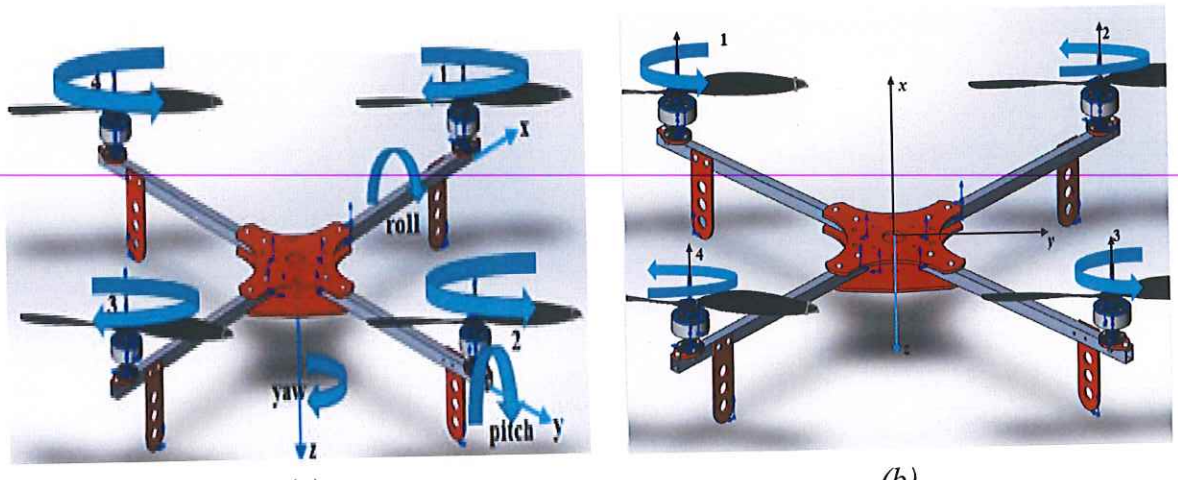
Trong nước, tình hình nghiên cứu cũng chưa đạt kết quả cao, do việc tính toán để đạt sự ổn định và cân bằng trên không trung rất khó. Vì vậy trong luận văn này chỉ trọng tâm đi sâu vào nghiên cứu Quadrotor với các điều kiện sau:

- Chỉ nghiên cứu Quadrotor bốn cánh công suất nhỏ.
- Thời gian bay của mô hình <= 5 phút.
- Chỉ tập trung vào bài toán điều khiển cân bằng.
- Không trực tiếp thiết kế các mạch điều khiển, tác giả dùng các vật tư có sẵn trên thị trường như: dùng board Arduino cho mạch điều khiển trung tâm, 4 ESC (Electronic speed controller), board IMU GY-86, khung cơ khí, động cơ, pin.

Chương 2
CƠ SỞ LÝ THUYẾT VÀ GIẢI PHÁP

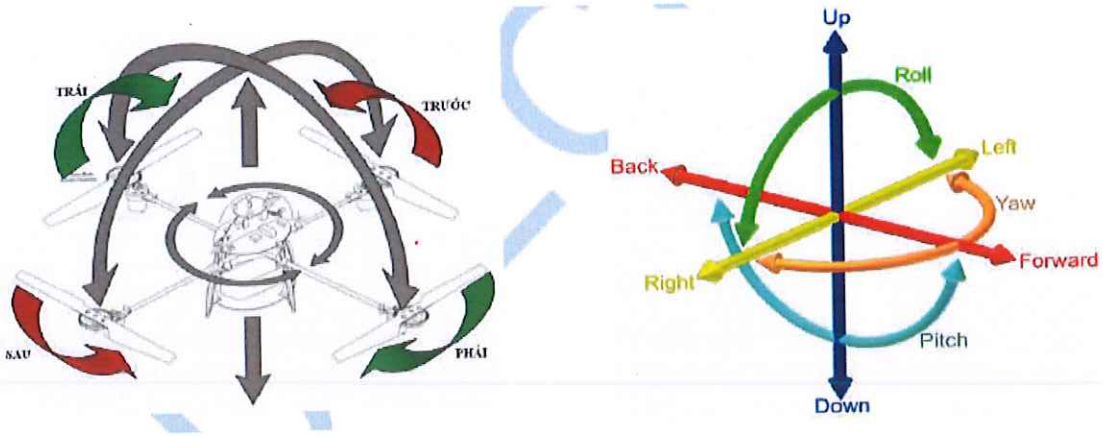
2.1 Nguyên lý hoạt động của Quadrotor

Quadrotor có thể được định nghĩa là một phương tiện bay có bốn cánh quạt gắn lên phần cuối cùng của một khung hình chữ thập, là một trong những AUS linh hoạt nhất trong di chuyển. Nó có thể cất hay hạ cánh theo phương thẳng đứng nên nó không cần đường dẫn để lấy đà như các loại máy bay cánh bằng khác. Nó có thể quay trái, quay phải, lên, xuống ở bất kỳ góc quay nào khi người điều khiển yêu cầu. Bốn cánh quạt được điều khiển bởi bốn động cơ không chổi than (BLDC). Trong bốn cánh quạy này sẽ được chia làm 2 cặp đối nhau. Cặp cánh quạt phía trước và phía sau quay cùng chiều kim đồng hồ, trong khi đó cặp cánh quạt bên phải và bên trái quay ngược chiều kim đồng hồ nhằm cân bằng moment xoay được tạo ra từ các cánh quạt trên khung. Cả bốn cánh quạt sinh ra một lực đẩy bằng nhau khi cất cánh nếu quay cùng một tốc độ. Hệ thống được mô tả tổng quát trong hình 2-1 và sơ đồ nguyên lý được mô tả trong hình 2-2 [5], [6].



(a) *(b)*

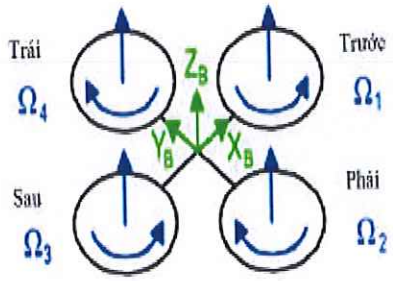
Hình 2-1: Mô tả tổng quát của Quadrotor



Hình 2-2: Sơ đồ nguyên lý của Quadrotor

2.1.1 Trạng thái lơ lửng

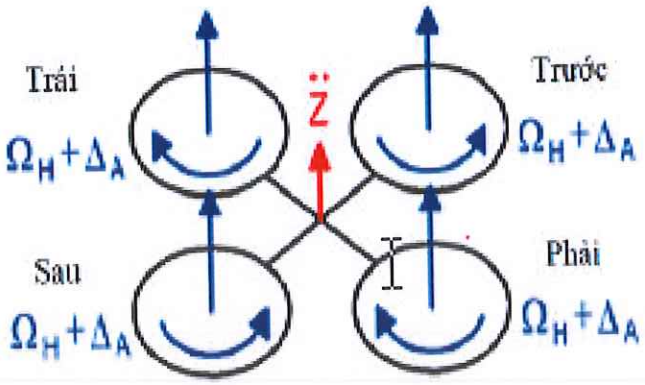
Quadrotor được thiết kế có thể bay lơ lửng trong không trung. Ở trạng thái này, tất cả các cánh quạt quay với một tốc độ không đổi ($\Omega_1=\Omega_2=\Omega_3=\Omega_4$).



Hình 2-3: Trạng thái lơ lửng

2.1.2 Trạng thái bay lên, bay xuống

Quadrotor được thiết kế có thể bay lên hoặc xuống theo phương thẳng đứng. Khi muốn điều khiển bay lên theo phương thẳng đứng ta sẽ cùng tăng như nhau về tốc độ của 4 động cơ. Khi muốn giảm độ cao theo phương thẳng đứng, ta sẽ cùng giảm dần tốc độ của 4 đông cơ như nhau.



Hình 2-4: Trạng thái bay lên, xuống

Trong đó:

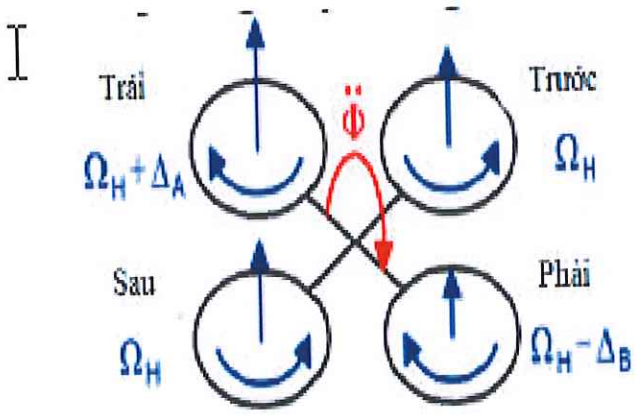
$\ddot{z}$: là gia tốc theo phương thẳng đứng

Ω_H: là vận tốc góc của cánh quạt

Δ_A: là lượng tăng hoặc giảm ω để Quadrotor bay lên hay hạ xuống

2.1.3 Trạng thái nghiêng trái, nghiêng phải (Roll)

Quadrotor được thiết kế có thể bay nghiệng trái, nghiêng phải. Để bay nghiêng sang phải, giữ nguyên tốc độ của của hai cánh quạt phía trước và phía sau, tăng tốc độ của cánh quạt bên trái và giảm tốc độ của cánh quạt bên phải. Tương tự, để nghiêng sang trái, ta vẫn giữ nguyên tốc độ của hai cánh quạt phía trước và phía sau, tăng tốc độ của cánh quạt bên phải và giảm tốc độ của cánh quạt bên trái.



Hình 2-5: Trạng thái nghiêng trái, nghiêng phải

Trong đó:

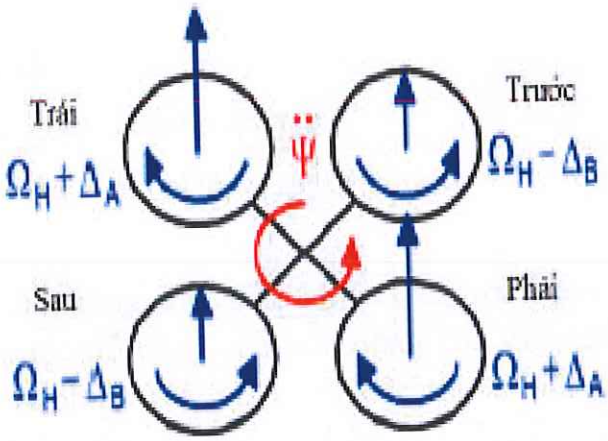
$\ddot{\Phi}$: là gia tốc theo phương nghiêng

Ω_H: là vận tốc góc của cánh quạt

Δ_A, Δ_B: là lượng tăng hoặc giảm ω để Quadrotor nghiêng tái, nghiêng phải

2.1.4 Trạng thái xoay trái, xoay phải (Yaw)

Quadrotor được thiết kế có thể xoay sang trái, xoay sang phải theo trục thẳng đứng. Để xoay sang phải, ta giảm tốc độ cặp cánh quạt phía trước và phía sau, tăng tốc độ cặp cánh quạt bên trái và bên phải. Để xoay sang trái, ta làm ngược lại theo cách trên.



Hình 2-6: Trạng thái xoay trái, xoay phải

Trong đó:

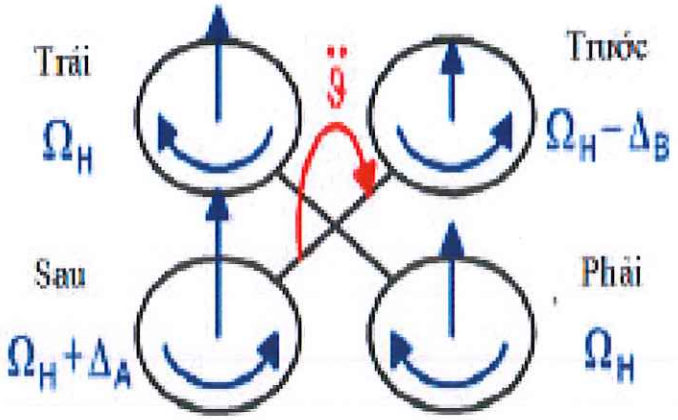
$\ddot{\Psi}$: là gia tốc xoay

Ω_H: là vận tốc góc của cánh quạt

Δ_A, Δ_B: là lượng tăng hoặc giảm ω để Quadrotor xoay trái, xoay phải

2.1.5 Trạng thái lật trước, lật sau (Pitch)

Tương tự như trạng thái nghiêng trái, nghiêng phải, hai cánh quạt trái và phải giữ nguyên tốc độ như nhau. Để lật trước hay lật sau, ta tăng giảm tốc độ của hai cánh quạt phía trước và phía sau.



Hình 2-7: Trạng thái lật trước, lật sau

Trong đó:

$\ddot{\theta}$: là gia tốc lật

Ω_H : là vận tốc góc của cánh quạt

Δ_A, Δ_B : là lượng tăng hoặc giảm ω để Quadrotor xoay trái, xoay phải

Chúng ta có thể thấy rằng, việc điều khiển Quadrotor bay như thế nào thực chất là việc điều khiển tốc độ quay của bốn động cơ. Việc điều khiển các cặp động cơ thật đơn giản và chính xác, vì vậy Quadrotor sẽ linh động hơn các thiết bị bay khác rất nhiều. Đây cũng là một ưu điểm của Quadrotor để các nhà khoa học quan tâm nghiên cứu.

2.2 Cấu tạo của Quadrotor

2.2.1 Động cơ không chổi than (Brushless DC motor: BLDC)

Động cơ BLDC là một động cơ không chổi than, nên không bị ăn mòn hay không bị phóng tia lửa điện làm hư hỏng thiết bị. BLDC là một loại động cơ đồng bộ nam châm vĩnh cửu, có tốc độ cao, monent lớn, khối lượng nhỏ rất phù hợp cho việc điều khiển Quadrotor. BLDC trong thực tế như hình 2-8 và hình 2-9.

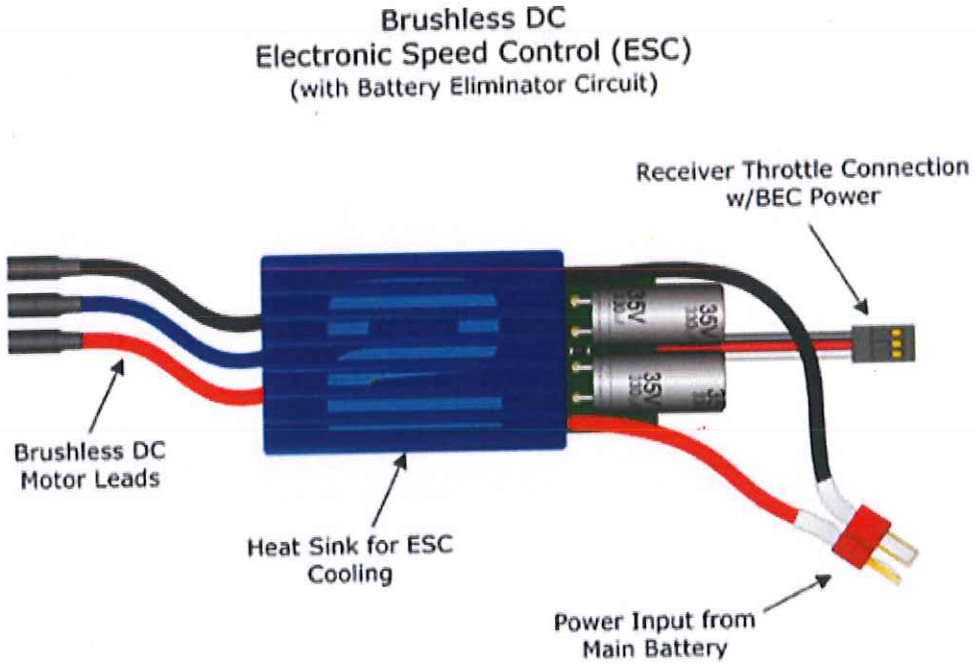
Hình 2-8: Động cơ BLDC

Hình 2-9: Động cơ BLDC Max-CF 2822

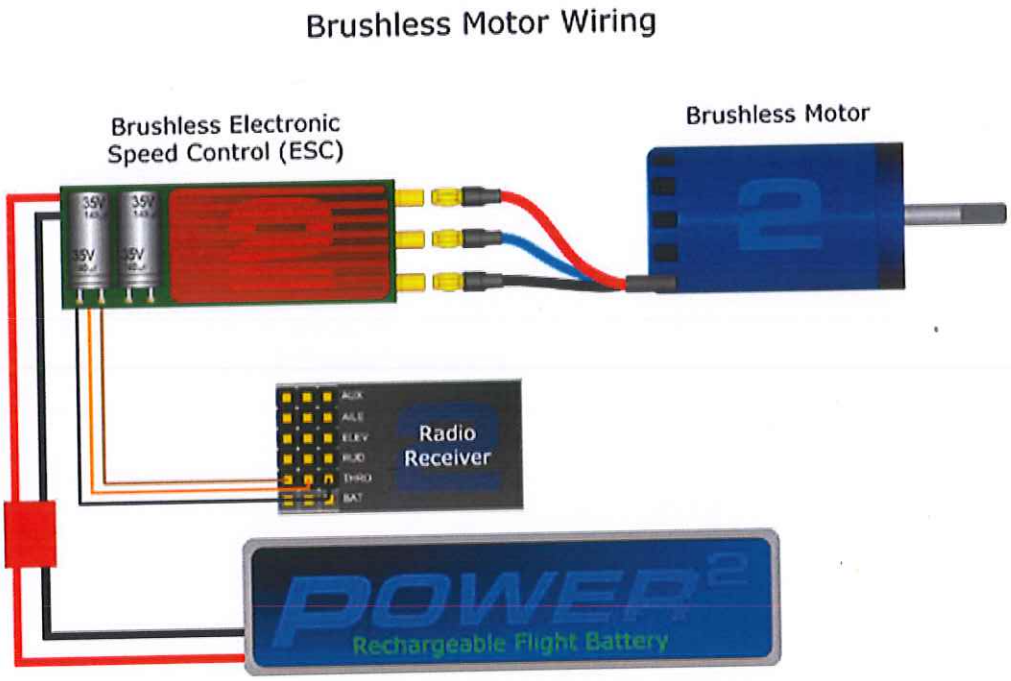
2.2.2 Mạch điều khiển động cơ BLDC

Trong luận văn này, người thực hiện không trực tiếp nghiên cứu mạch điều khiển động cơ BLDC mà sử dụng board mạch điều khiển có sẵn, đó là mạch ESC (Electronic Speed Controller). Mạch ESC dùng để thay đổi tốc độ của động cơ BLDC dựa trên nguyên tắc cung cấp xung tuần tự cho các cuộn dây để tạo ra từ trường quay cho động cơ. Trong trường hợp này, các ESC đóng vai trò như các biến tần, dùng để đổi điện áp một chiều sang điện áp 3 pha xoay chiều, có

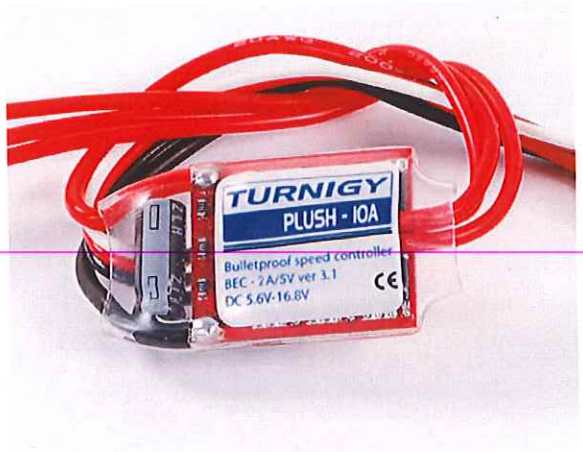
tần số thay đổi được. Để đổi chiều quay cho động cơ, ta chỉ cần thay đổi vị trí cấp nguồn của 2 trong 3 cuộn dây [15]. Sơ đồ cấu tạo của một ESC như hình 2-10, cách đấu nối ESC để điều khiển động cơ BLDC được thể hiện trên hình 2-11, hình ảnh một ESC thực tế được thể hiện trong hình 2-12, chi tiết về sơ đồ nguyên lý của một ESC được thể hiện trong phụ lục A.

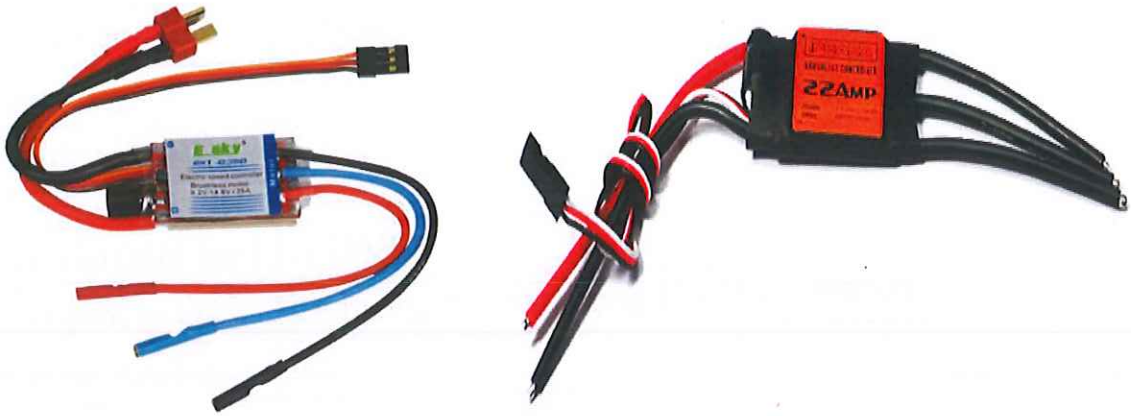


Hình 2-10: Cấu tạo của một ESC



Hình 2-11: Cách kết nối một ESC

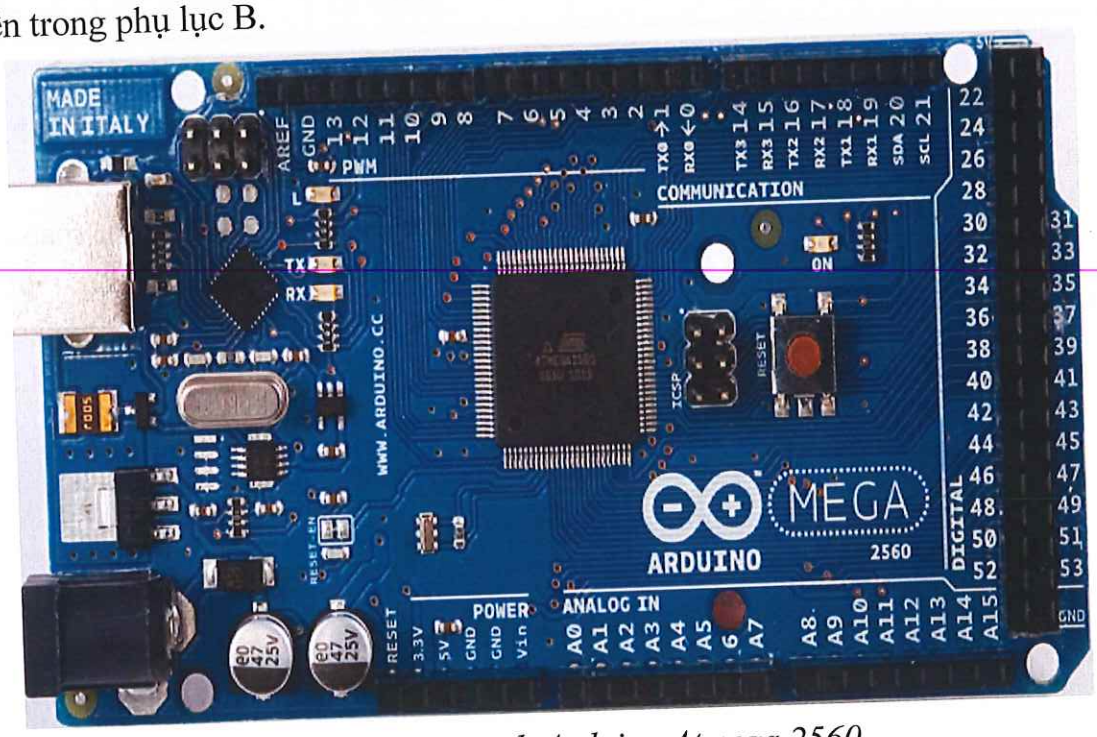




Hình 2-12: ESC thực tế

2.2.3 Khối điều khiển trung tâm

Trong luận văn này, người thực hiện không trực tiếp thiết kế Board điều khiển trung tâm mà sử dụng Board có sẵn trên thị trường, đó là Board Arduino Atmega 2560 như hình 2-13, các thông số kỹ thuật của Board Arduino Atmega 2560 được trình bày trong bảng 2-1, các thông số chi tiết của Board được thể hiện trong phụ lục B.



Hình 2-13: Board Arduino Atmega 2560

Bảng 2-1: Các thông số của Board Arduino Atmega 2560 [16]

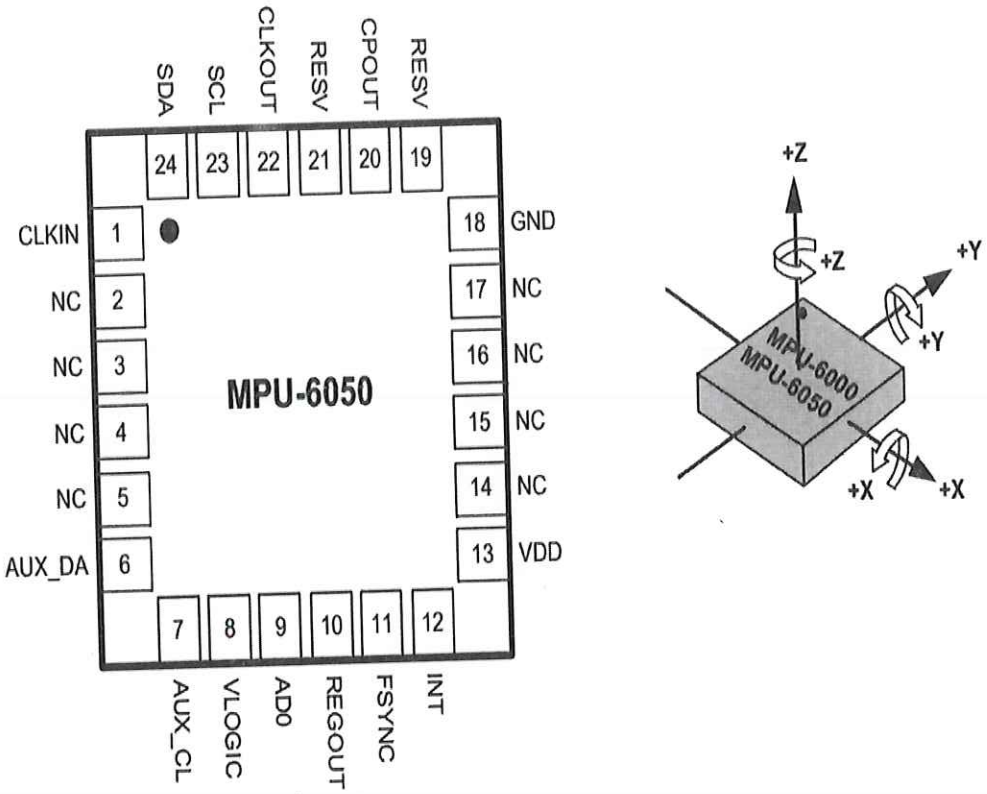
Vi điều khiển	ATmega2560
Áp hoạt động	3.3 V
Áp vào đề nghị (đề nghị)	7-12V
Áp ngõ vào (giới hạn)	6-20V
Chân Digital I / O	54 (trong đó có 12 cung cấp đầu ra PWM)
Chân input Analog	16
Dòng DC I / O Pin	130 mA
Dòng DC 3.3V Pin	800 mA
Dòng DC chân 5V	800mA
Flash Memory	256KB trong đó tất cả đều được sử dụng bởi bộ nạp ứng dụng.
RAM	8 KB
Tần số xung clock	16 MHz

2.2.4 Khối cảm biến

Trong luận văn này, người thực hiện sử dụng Board tích hợp sẵn nhiều loại cảm biến trên cùng một mạch in, đó là Board IMU GY86. Các loại cảm biến được tích hợp trong Board này như MPU 6050 tích hợp 3 trục gia tốc, 3 trục góc gyro, HMC5883L, MS5611, các dữ liệu được truyền qua cổng nối tiếp chuẩn I_2C. Hình ảnh của Board IMU GY86 được trình bày trong hình 2-14.

Hình 2-14: Board IMU GY86

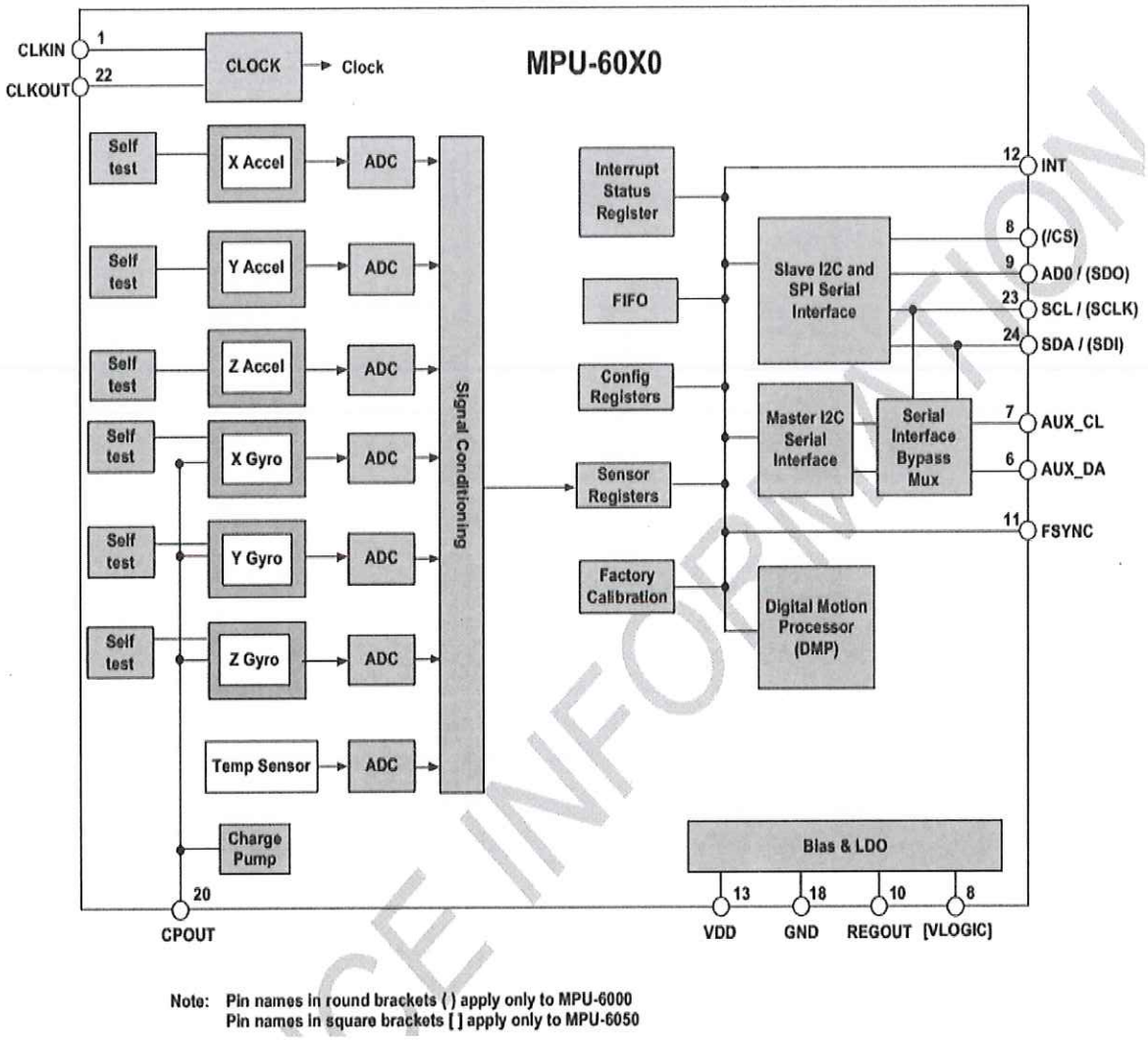
MPU - 6050 là loại thiết bị Motion Tracking 6 trục tích hợp đầu tiên trên thế giới, nó là sự kết hợp của 3 trục gyroscope, 3 trục Accelerometer và một bộ xử lý chuyển động số (Digital Motion Processor). MPU - 6050 bao gồm 3 bộ ADC 16 bit để số hóa đầu ra của gyroscope và 3 bộ ADC dành cho Accelerometer. Để có sự tính toán chính xác với cả sự nhanh và chậm, nó cho phép người sử dụng lập trình cho gyroscope với các tầm đo ±250, ±500, ±1000, và ±2000°/sec (dps), và cho Accelerometer với các tầm đo ±2g, ±4g, ±8g, và ±16g. Sơ đồ chân MPU 6050 được trình bày trong hình 2-15, chức năng các chân của MPU – 6050 được trình bày trong bảng 2-2, sơ đồ khối của MPU- 6050 được trình bày trong hình 2-16.



Hình 2-15: Sơ đồ chân của MPU- 6050

Bảng 2-2: Chức năng các chân MPU 6050

Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
1	Y	Y	CLKIN	Optional external reference clock input. Connect to GND if unused.
6	Y	Y	AUX_DA	I^2C master serial data, for connecting to external sensors
7	Y	Y	AUX_CL	I^2C Master serial clock, for connecting to external sensors
8	Y		/CS	SPI chip select (0=SPI mode)
8		Y	VLOGIC	Digital I/O supply voltage
9	Y		AD0 / SDO	I^2C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I^2C Slave Address LSB (AD0)
10	Y	Y	REGOUT	Regulator filter capacitor connection
11	Y	Y	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	Y	Y	INT	Interrupt digital output (totem pole or open-drain)
13	Y	Y	VDD	Power supply voltage and Digital I/O supply voltage
18	Y	Y	GND	Power supply ground
19, 21	Y	Y	RESV	Reserved. Do not connect.
20	Y	Y	CPOUT	Charge pump capacitor connection
22	Y	Y	CLKOUT	System clock output
23	Y		SCL / SCLK	I^2C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I^2C serial clock (SCL)
24	Y		SDA / SDI	I^2C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I^2C serial data (SDA)
2, 3, 4, 5, 14, 15, 16, 17	Y	Y	NC	Not internally connected. May be used for PCB trace routing.



Hình 2-16: Sơ đồ khối MPU 6050

2.2.4.1 Cảm biến trường HMC5883L

Cảm biến trường HMC5883L là loại cảm biến được thiết kế dùng để cảm biến từ ở mức thấp thông qua một giao tiếp số được sử dụng trong các ứng dụng như la bàn và đo từ trường với chi phí thấp. HMC5883L bao gồm những công nghệ tiên tiến, độ phân giải cao của dòng cảm biến HMC118X, dòng cảm biến magneto-resistive, thêm vào đó là một ASIC chứa đựng độ khuếch đại, tự động khử từ, khử offset và một bộ ADC-12 bit cho phép sai số hướng chuyển động từ.

HMC5883L sử dụng công nghệ Anisotropic Magnetoresistive (AMR) của Honeywel tạo nên một lợi thế so với các công nghệ cảm biến từ khác. Cảm biến Low Noise, AMR có thể đạt tới độ phân giải là 5 milli-gauss trong trường từ ±8 Gauss. Tốc độ tối đa dữ liệu đầu ra là 160 Hz. I_2C sử dụng giao tiếp ở mode Standard và Fast mode (100 và 400 khz).

2.2.4.2 Cảm biến độ cao MS5611

Cảm biến độ cao MS5611-01BA là một cảm biến đo độ cao thế hệ mới với độ phân giải cao (10cm) sử dụng chuẩn giao tiếp SPI và I_2C. Modul cảm biến bao gồm một cảm biến áp suất tuyến tính cao và một bộ ADC 24 bit tiêu thụ năng lượng cực thấp với hệ số calibrated bên trong. MS5611 cung cấp giá trị chính xác 24 bit, giá trị nhiệt độ và một số mode hoạt động khác, nó cho phép người sử dụng chuyển đổi tốc độ và năng lượng tiêu thụ.

2.3 Phương pháp điều khiển

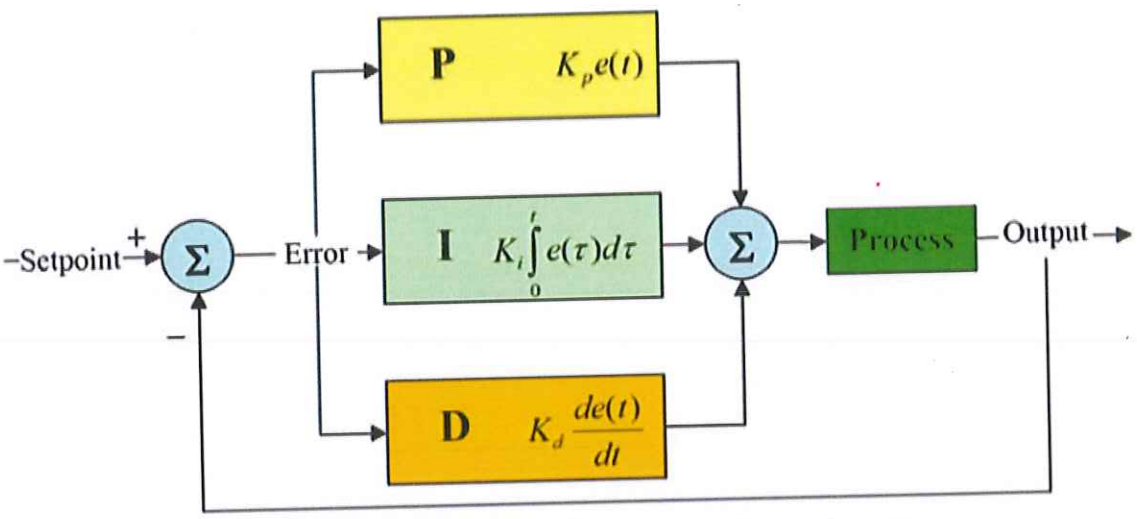
2.3.1 Giải thuật điều khiển PID

2.3.1.1 Lý thuyết điều khiển PID

Bộ điều khiển PID (Proportional Integral Derivative) là một bộ điều chỉnh có phản hồi nhằm làm giá trị sai lệch của một tín hiệu đang được điều khiển bằng 0. Bộ PID có ba thành phần:

- Proportional: Tích phân
- Integral: Vi phân
- Derivative: Đạo hàm

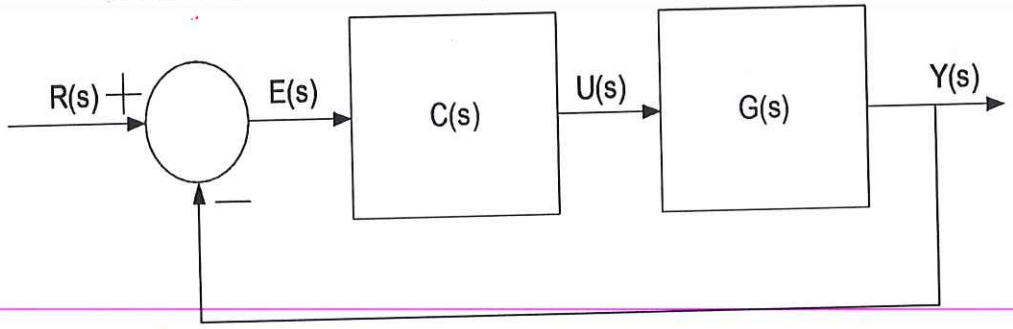
Ba thành phần này đều có vai trò đưa tín hiệu sai lệch về 0, nhưng mỗi thành phần đều có tính chất riêng. Tín hiệu phản hồi (feedback signal) thường là tín hiệu đo bằng cảm biến. Giá trị sai lệch bằng giá trị của tín hiệu đặt (setpoint) trừ cho giá trị của tín hiệu phản hồi.



Hình 2-17: Sơ đồ khối của bộ điều khiển PID

Bộ điều khiển PID được đặt theo tên của 3 khâu hiệu chỉnh P, I và D. Người ta có thể chỉ áp dụng bộ điều khiển P, PI, PID.

Mô hình điều khiển PID được trình bày trong hình 2-18.



Hình 2-18: Mô hình điều khiển PID

Công thức toán học của bộ điều khiển PID trên miền Laplace như công thức 2.1

$$C(s) = K_p + \frac{K_i}{s} + K_d s = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \quad (2.1)$$

Trong đó:

- K_p: Độ lợi tỉ lệ

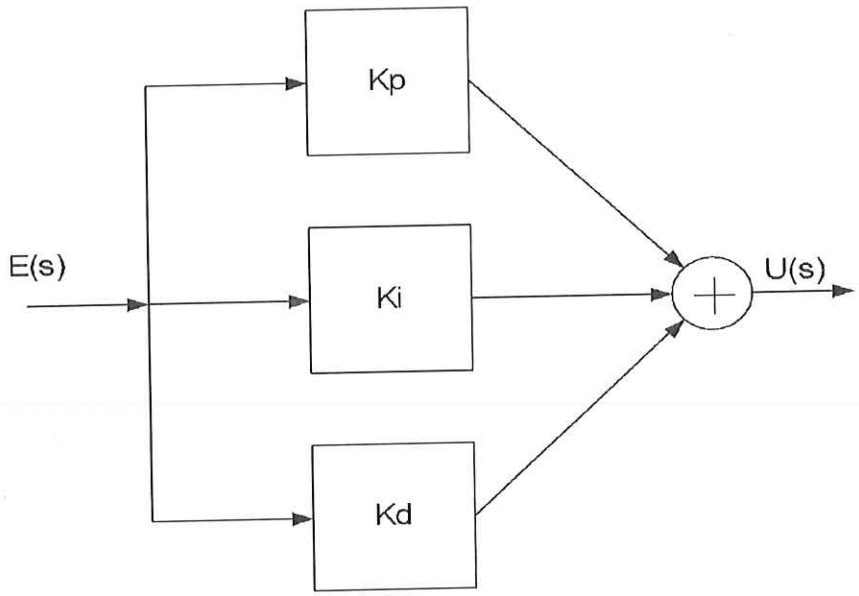
- K_i: Độ lợi tích phân
- K_d: Độ lợi vi phân
- $T_i = K_p/K_i$: Thời gian khâu vi phân
- T_d: Thời gian khâu tích phân

➢ Điều chỉnh tỉ lệ P: là phương pháp điều chỉnh tạo ra tín hiệu điều chỉnh tỉ lệ với sai lệch đầu vào.

➢ Điều chỉnh tích phân I: là phương pháp điều chỉnh tỉ lệ để lại một độ lệch (offset) sau điều chỉnh rất lớn. Để khắc phục ta kết hợp điều chỉnh tỉ lệ và điều khiển tích phân. Điều chỉnh tích phân là phương pháp điều chỉnh tạo ra tín hiệu điều chỉnh sao cho độ lệch giảm về 0. Thời gian càng nhỏ thể hiện tác động điều chỉnh tích phân càng mạnh, ứng với độ lệch càng bé.

➢ Điều chỉnh vi phân D: Khi hằng số thời gian hoặc thời gian chết của hệ thống rất lớn, điều chỉnh theo P hoặc PI có đáp ứng quá chậm thì ta sử dụng kết hợp với sử dụng vi phân. Điều chỉnh vi phân tạo ra tín hiệu điều chỉnh sao cho tỉ lệ với tốc độ thay đổi sai lệch đầu vào. Thời gian càng lớn thì điều chỉnh vi phân càng mạnh, ứng với bộ điều chỉnh với đáp ứng thay đổi đầu vào càng nhanh.

Cấu trúc của bộ điều khiển PID được trình bày trong hình 2-19



Hình 2-19: Cấu trúc bộ điều khiển PID

Bộ điều khiển PID có tín hiện ngõ ra tỉ lệ tuyến tính với tính hiệu ngõ vào, với vi phân và tích phân tính hiệu vào theo công thức 2.2

$$U(s) = K_p E(s) + K_I . \frac{E(s)}{S} + K_d . S . E(s) \qquad (2.2)$$

Cả 3 bộ điều chỉnh tỉ lệ (P), tích phân (I), vi phân (D) được cộng lại với nhau để tính toán đầu ra cho bộ điều khiển PID.

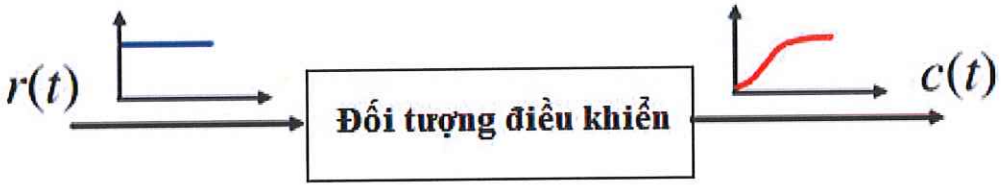
- Giá trị độ lợi tỉ lệ K_p càng lớn thì đáp ứng càng nhanh, do đó sai số càng lớn, bù khâu tỉ lệ càng lớn. Khi giá trị độ lợi tỉ lệ K_p quá lớn sẽ dẫn tới quá trình mất ổn định và dao động.
- Giá trị độ lợi tích phân K_i càng lớn, kéo theo sai số ổn định bị khử càng cao. Đổi lại là độ vọt lố càng lớn thì bất kỳ sai số âm nào được tích phân trong suốt đáp ứng quá độ phải được triệt tiêu tích phân bằng sai số dương trước khi tiến tới trạng thái ổn định.
- Giá trị độ lợi vi phân K_d càng lớn thì làm giảm độ vọt lố, nhưng làm chậm thời gian đáp ứng quá độ và có thể dẫn đến mất ổn định do khuếch đại nhiễu tín hiệu trong phép vi phân sai số.

2.3.1.2 Phương pháp điều chỉnh bộ PID

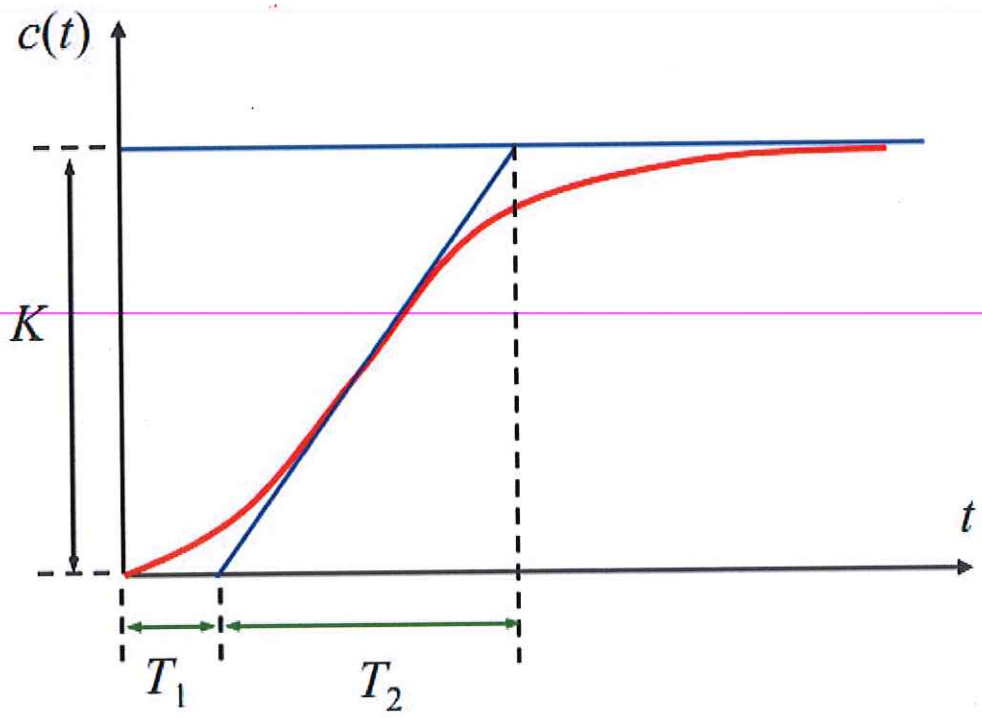
Ziegler và Nichols đưa ra hai phương pháp thực nghiệm để xác định tham số bộ điều khiển PID. Phương pháp thứ nhất dùng mô hình quán tính bậc nhất của đối tượng điều khiển. Phương pháp thứ hai không cần đến mô hình toán học của đối tượng nhưng chỉ áp dụng cho một số lớp đối tượng nhất định.

- **Phương pháp Ziegler - Nichols thứ nhất**

Xác định thông số của bộ điều khiển dựa vào đáp ứng của hệ hở [2], [3]. Ta cho tín hiệu *r(t)* tác động ở ngõ vào của mô hình điều khiển và đo lại đáp ứng ở ngõ ra như hình 2-20 và hình 2-21.

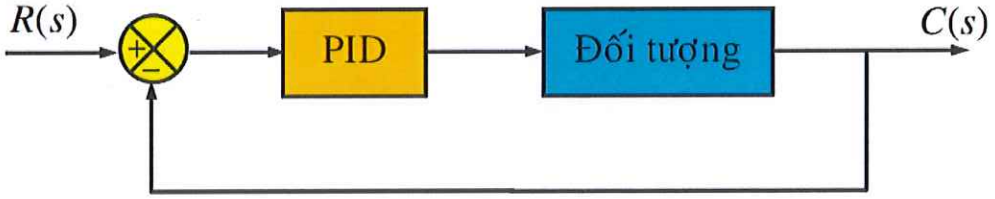


Hình 2-20: Sơ đồ khối của hệ hở



Hình 2-21: Đáp ứng của hệ hở

Khi đó ta có sơ đồ điều khiển dùng PID như hình 2-22 [2], [3], ngõ ra của hệ thống được hồi tiếp về để tham chiếu với ngõ vào mong muốn.



Hình 2-22: Sơ đồ khối của hệ kín có bộ PID

Cuối cùng ta có phương trình cho bộ điều khiển PID như công thức (2.3)

$$G_C(s) = K_P\left(1+\frac{1}{T_I s}+T_D s\right) \tag{2.3}$$

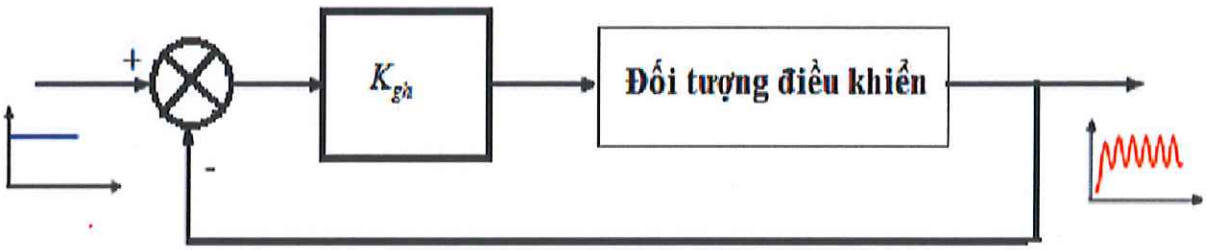
Bằng phương pháp thực nghiệm, Ziegler-Nichols đã tổng hợp lại như trong bảng 2-3.

Bảng 2-3: Bảng tính các thông số PID theo Z-N1

Thông số bộ điều khiển	K_P	K_I	K_D
P	$\frac{T_2}{T_1 K}$	∞	0
PI	$0.9\frac{T_2}{T_1 K}$	$T_1/0.3$	0
PID	$1.2\frac{T_2}{T_1 K}$	$2T_1$	$0.5T_1$

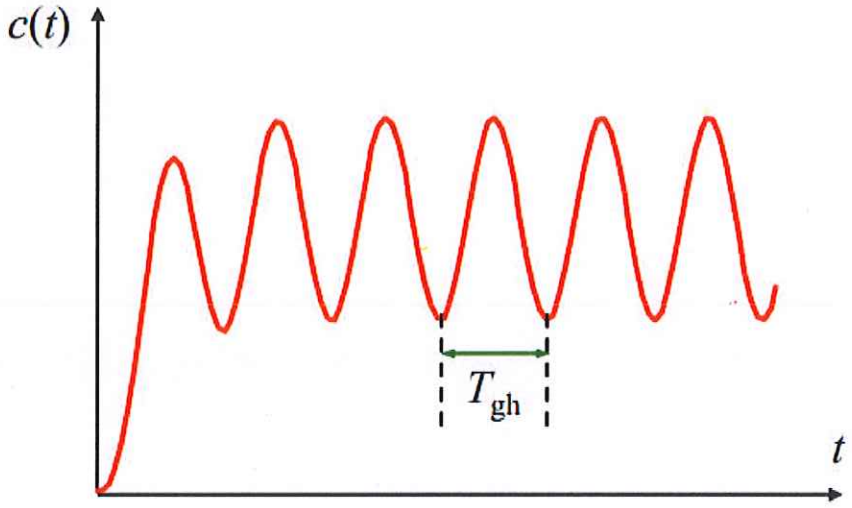
- **Phương pháp Ziegler-Nichols thứ hai**

Đối với phương pháp này, Ziegler-Nichols thay bộ PID thành bộ khuếch đại có hệ số khuếch đại K_{gh} như hình 2-23 [2], [3].



Hình 2-23: Sơ đồ khối của hệ kín có bộ tỉ lệ P

Phương pháp này thay bộ điều khiển PID trong hệ kín bằng bộ khuếch đại, sau đó tăng K cho đến khi hệ nằm ở biên giới ổn định, tức là hệ kín trở thành khâu dao động điều hoà như trong hình 2-24.



Hình 2-24: Đáp ứng của hệ kín

Lúc này ta có K_{gh} và chu kỳ của dao động đó là T_{gh}. Tham số cho bộ điều khiển PID chọn theo bảng 2-4.

Bảng 2-4: Bảng tính các thông số PID theo Z-N2

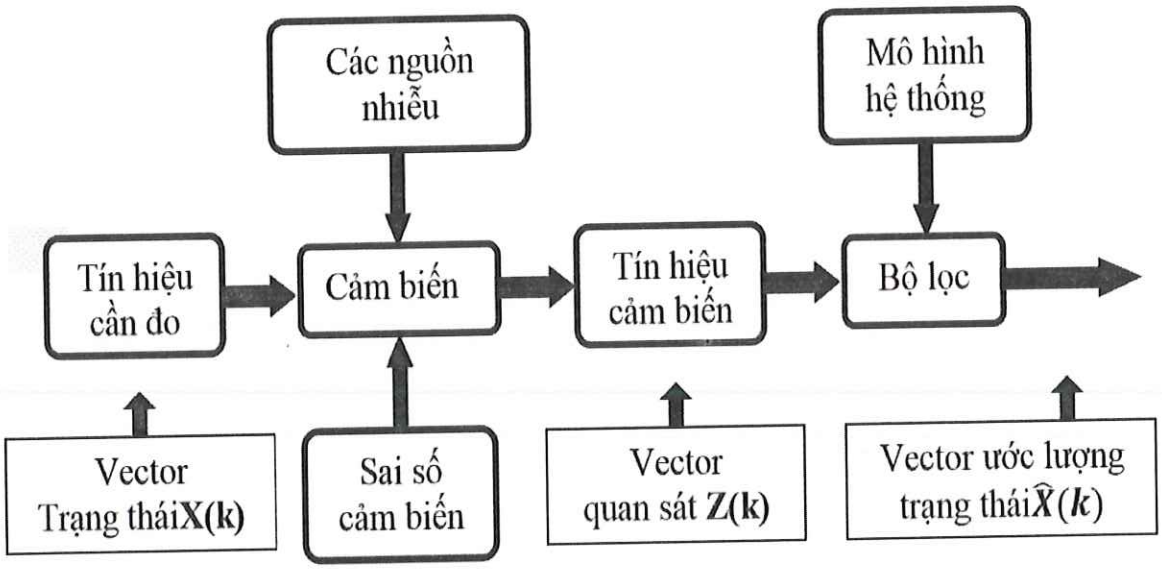
Bộ điều khiển	K_P	T_I	T_D
P	$0.5*K_{gh}$	∞	0
PI	$0.45*K_{gh}$	$\frac{1}{1.2}*T_{gh}$	0
PID	$0.6*K_{gh}$	$0,5*T_{gh}$	$0,125*T_{gh}$

2.3.2 Lý thuyết bộ lọc Kalman mở rộng

Bộ lọc là quá trình xử lý nhằm loại bỏ những gì không có giá trị hoặc không quan tâm đến và giữ lại những gì có giá trị sử dụng.

Bộ lọc Kalman là một tập hợp các phương trình toán học mô tả một phương pháp tính toán truy hồi hiệu quả cho phép ước đoán trạng thái của một quá trình (process) sao cho trung bình phương sai của độ lệch (giữa giá trị thực và giá trị ước đoán) là nhỏ nhất. Bộ lọc Kalman rất hiệu quả trong việc ước đoán các trạng thái trong quá khứ, hiện tại và tương lai, thậm chí ngay cả tính chính xác của hệ thống mô phỏng không được khẳng định.

Bộ lọc Kalman được sử dụng để lọc tín hiệu trả về từ cảm biến. Ngõ ra của bộ lọc Kalman không phải là tín hiệu cùng loại với tín hiệu của cảm biến (sau khi đã loại nhiễu) như những bộ lọc khác mà ngõ ra của nó là ước lượng (dự đoán) tối ưu của vector trạng thái. Sơ đồ khối của bộ lọc Kalman trong hình 2-25.



Hình 2-25: Sơ đồ khối của bộ lọc Kalman

Hệ thống rời rạc tuyến tính được trình bày trong phương trình (2.4) [6], [8].

$$\begin{aligned} x_k &= Ax_{k-1} + Bu_{k-1} + \omega_k \\ z_k &= Hu_k + v_k \end{aligned} \tag{2.4}$$

Trong đó:

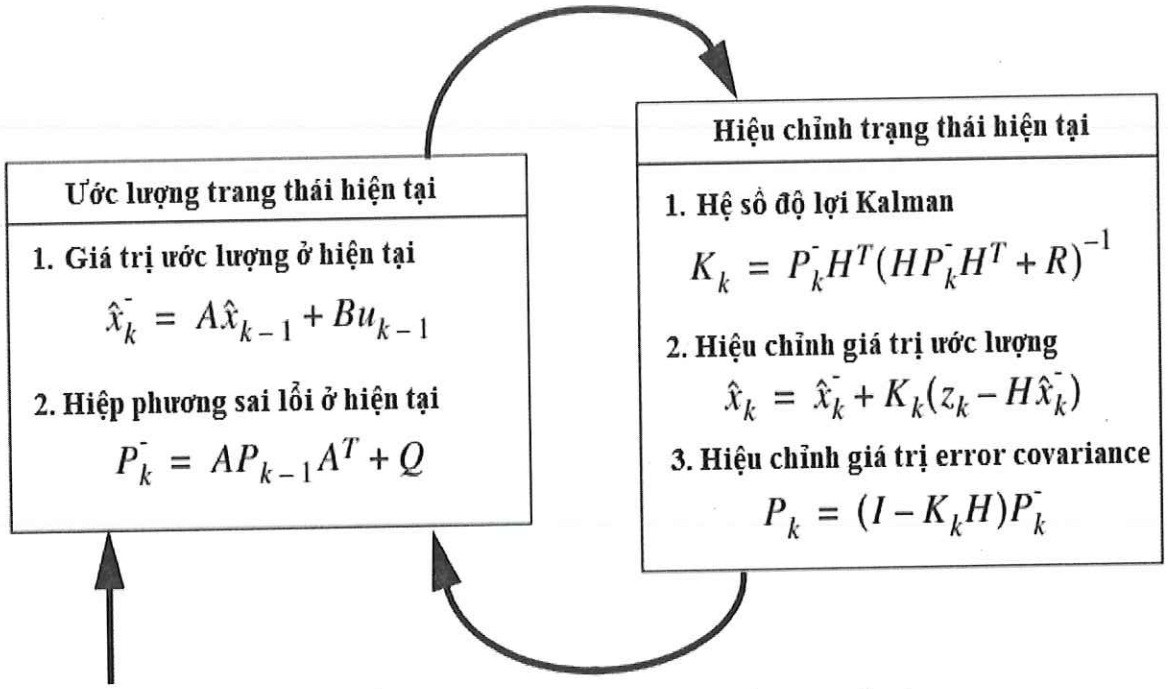
- A, B, H: là các ma trận không gian trạng thái
- x_k : là vector trạng thái tại thời điểm k
- ω_k, v_k : là các can nhiễu phép đo của hệ thống

Thuật toán Kalman có thể được mô tả qua hai giai đoạn: giai đoạn dự báo giá trị trạng thái của mô hình và giai đoạn hiệu chỉnh giá trị cho trạng thái của mô hình.

Giai đoạn dự báo: Dựa vào giá trị ước lượng của tín hiện vector ngõ vào trước đó và giá trị đo được từ cảm biến ở thời điểm hiện tại U_k, Kalman sẽ thực hiện dự báo giá trị của trạng thái tiếp theo và tính được ma trận hiệp phương sai lỗi (covariance error).

Giai đoạn hiệu chỉnh giá trị mô hình: Trước tiên sẽ tính toán độ lợi bộ lọc Kalman K_k sau đó hiệu chỉnh giá trị ước lượng ở hiện tại và ma trận hiệp phương sai P_k.

Sau đó quay về giai đoạn dự báo. Lưu đồ giải thuật của bộ lọc Kalman mở rộng được trình bày trong hình 2-26 [8].



Hình 2-26: Lưu đồ giải thuật của bộ lọc Kalman mở rộng

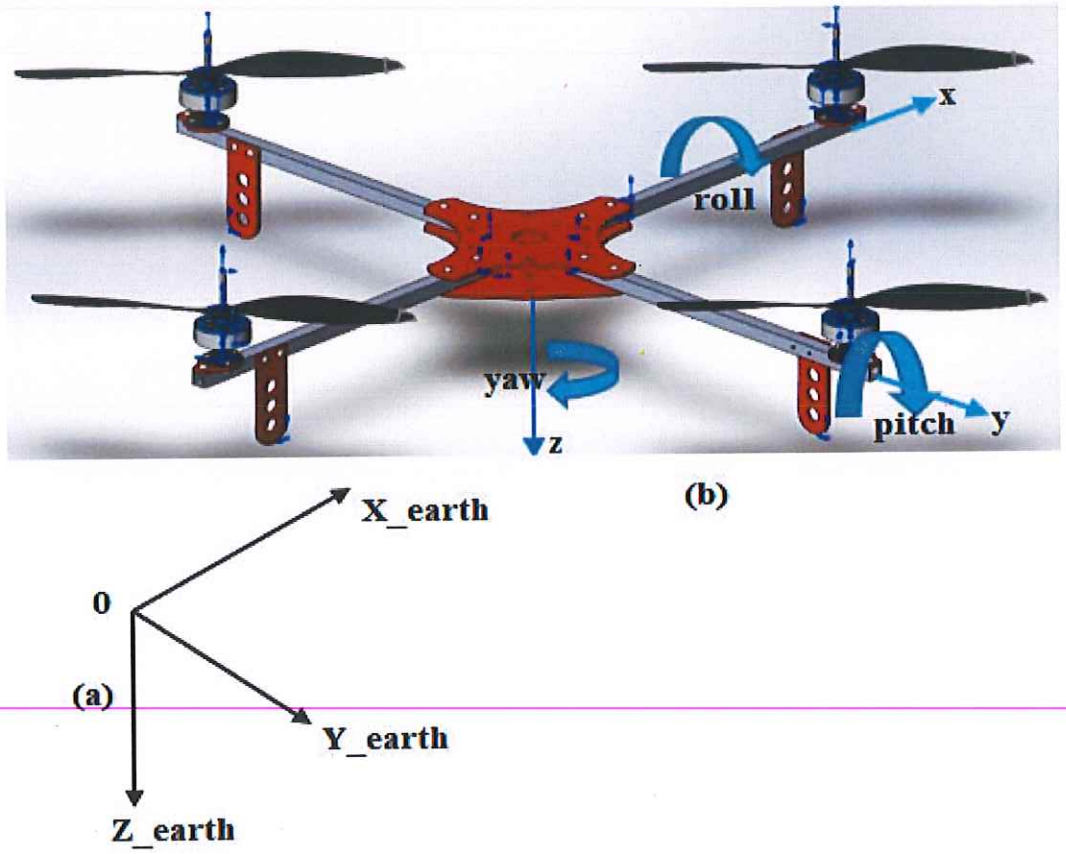
Trong luận văn này, tác giả dùng bộ lọc Kalman để lọc tín hiệu trả về từ cảm biến IMU, có nhiệm vụ tính toán các thông số nhận về từ cảm biến tại các thời điểm khác nhau, ước lượng giá trị tối ưu cần thiết cho hệ thống điều khiển.

Chương 3
MÔ HÌNH HÓA VÀ MÔ PHỎNG

3.1 Mô hình toán học

3.1.1 Mô phỏng theo góc Euler-Lagrange

Các trạng thái di chuyển của Quadrotor trong không gian được mô tả trong hình 3-1.



Hình 3-1: Các trạng thái trên hai hệ tọa độ: (a) hệ tọa độ mặt đất, (b) hệ tọa độ trên thân máy bay

Quadrotor là một hệ thống 6 bậc tự do và nó được định nghĩa bằng 12 thông số: ba góc roll, pitch, yaw ($\emptyset$, θ, Ψ), (p, q, r) là ba vận tốc góc quanh các trục x, y, z trên hệ tọa độ thân máy bay, ba thông số còn lại là (x, y, z) chỉ định vị trí của Quadrotor trên 3 trục x, y, z tham chiếu giữa hai hệ tọa độ mặt đất và hệ

tọa độ thân máy bay, (*u, v, w*) là các vận tốc tương ứng trên *x, y, z*. Vector trạng thái thể hiện đầy đủ các thông số cần thiết trong mô phỏng như sau:

$$V = \{x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}\}$$

3.1.2 Ma trận xoay góc_Euler

Từ không gian hình học và kết quả tham chiếu từ hai hệ toạ độ trên, ta có ma trận xoay xung quanh ba trục x,y,z tương ứng R_x, R_y, R_z [6], [7].

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \quad (3.1)$$

$$R_z = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.2)$$

$$R_z = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.3)$$

Ta có ma trận truyền tổng hợp:

$R = R_x . R_y . R_z$

$$R = \begin{pmatrix} c\psi c\theta & s\psi c\theta & -s\theta \\ c\psi s\theta s\phi - c\phi s\psi & s\psi s\theta s\phi + c\psi c\phi & s\phi c\theta \\ c\psi s\theta c\phi + s\psi s\phi & s\psi s\theta c\phi - s\phi c\psi & c\theta c\phi \end{pmatrix} \quad (3.4)$$

Trong đó:

$$c\phi = \cos\phi \qquad s\phi = \sin\phi$$

$$c\theta = \cos\theta, s\theta = \sin\theta$$

$$c\psi = \cos\psi, s\psi = \sin\psi$$

Để tìm vận tốc góc u, v,w theo phương x, y, z ta tìm theo đạo hàm của x, y và z.

Ta có:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = R \begin{pmatrix} u \\ v \\ w \end{pmatrix} \quad (3.5)$$

Suy ra:

$$\dot{x} = a + b + c \quad (3.6)$$

Với: $a = cos\Psi.cos\theta.u$, $b = sin\Psi.\ cos\theta.v$, $c = -sin\theta.w$

$$\dot{y} = d + e + f \quad (3.7)$$

Với: d = (cosΨ.sinθ.sinØ-cosØ,sinΨ)u,
e =(sinΨ.sinθ.sinØ+cosΨ.cosØ)v;
f= sinØ.cosθ.w

$$\dot{z} = g + h + k \quad (3.8)$$

Với: g = (cosΨ.sinθ.cosØ+sinΨ.sinØ)u,
h = (sinΨ.sinθ.cosØ-sinØ.cosΨ)v,
k = cosθ.cosØ.w

Để tìm vận tốc góc quay p, q, r trên ba trục x, y, z. Ta lấy đạo hàm góc quay ∅, θ, và Ψ.

Ta có:

$$\frac{d}{dt}\begin{pmatrix}\phi\\ \theta\\ \psi\end{pmatrix} = T\begin{pmatrix}p\\ q\\ r\end{pmatrix} \tag{3.9}$$

Với:

$$T = \begin{pmatrix}1 & tan\theta.sin\emptyset & tan\theta.cos\emptyset\\ 0 & cos\emptyset & -sin\emptyset\\ 0 & sin\emptyset/cos\theta & cos\phi/cos\theta\end{pmatrix} \tag{3.10}$$

Suy ra:

$$\dot{\phi} = p + q\tan\theta\sin\phi + r\tan\theta\cos\phi \tag{3.11}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \tag{3.12}$$

$$\dot{\psi} = \frac{q\sin\phi}{\cos\theta} + \frac{r\cos\phi}{\cos\theta} \tag{3.13}$$

3.1.3 Phương trình động học

Theo định luật Newton thì hợp lực F = ma, với a là gia tốc, m là khối lượng vật thể [6], [7].

$$\frac{F+G}{m}=\begin{pmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{pmatrix}-\begin{pmatrix}p\\ q\\ r\end{pmatrix}\begin{pmatrix}u & v & w\end{pmatrix} \tag{3.14}$$

Suy ra:

$$\begin{pmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{pmatrix}=\frac{1}{m}\begin{pmatrix}F_x\\ F_y\\ F_z\end{pmatrix}=R\begin{pmatrix}0\\ 0\\ g\end{pmatrix}-\begin{pmatrix}p\\ q\\ r\end{pmatrix}\begin{pmatrix}u & v & w\end{pmatrix} \tag{3.15}$$

Với vector gia tốc lực hấp dẫn:

$$R\begin{pmatrix}0\\ 0\\ g\end{pmatrix}=g\begin{pmatrix}-\sin\theta\\ \sin\phi\cos\theta\\ \cos\theta\cos\phi\end{pmatrix} \tag{3.16}$$

Vậy, vận tốc của Quadrotor có thể tính như công thức sau:

$$\begin{pmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{pmatrix}=\frac{1}{m}\begin{pmatrix}F_x\\ F_y\\ F_z\end{pmatrix}+g\begin{pmatrix}-\sin\theta\\ \sin\phi\cos\theta\\ \cos\theta\cos\phi\end{pmatrix}-\begin{pmatrix}qw-rv\\ ru-pw\\ pv-qu\end{pmatrix} \tag{3.17}$$

Ta được công thức tính gia tốc như sau:

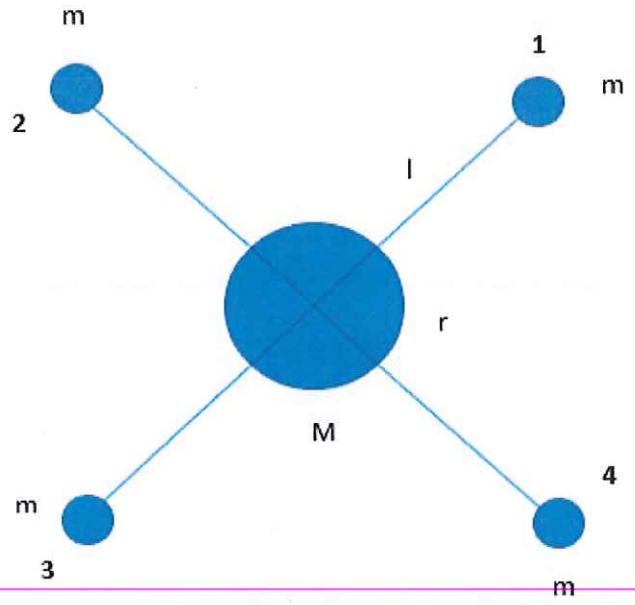
$$\dot{u}=\frac{1}{m}F_x-g\sin\theta-qw+rv \tag{3.18}$$

$$\dot{v} = \frac{1}{m} F_y + g \sin\phi \cos\theta - \text{ur} + pw \tag{3.19}$$

$$\dot{w} = \frac{1}{m} F_z + g \cos\theta \cos\phi - pv + qu \tag{3.20}$$

3.1.4 Khí động lực và moment hệ thống

Giả sử Quadrotor được sử dụng cho mô hình gồm: 04 động cơ có khối lượng m, chiều dài cánh tay l, tại tâm khối điều khiển có trọng lượng M và bán kính r như hình 3-2.



Hình 3-2: Sơ đồ bố trí phần cứng Quadrotor

Quadrotor là một khối đối xứng nên moment xoắn của hệ thống được sinh ra cũng mang tính đối xứng, do đó ta có ma trận của moment quán tính hệ thống như sau:

$$I = \begin{pmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{pmatrix} \tag{3.21}$$

Với:

$$I_x = I_y = I_z = \frac{2Mr^2}{5} + 2l^2ma \tag{3.22}$$

Và tại tâm lực của hệ thống sẽ trùng với tâm của mô hình, nên theo định luật Neuton – Euler ta có:

Lực tác động:

$$F = Ma = M\dot{v} \tag{3.23}$$

Với M: Khối lượng tổng của Quadrotor

Moment:

$$T = I\dot{\omega} + \omega.I\omega \tag{3.24}$$

Với: $\dot{\omega}$: Vận tốc góc

ω: Góc quay quanh trục.

Từ đó ta có thể viết dạng Newton-Euler cho mô hình như sau:

$$\begin{pmatrix} F_i \\ T_i \end{pmatrix} = \begin{pmatrix} MI & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \dot{v}_i \\ \dot{w}_i \end{pmatrix} + \begin{pmatrix} \omega \mathrm{Mv} \\ \omega.\mathrm{I}\omega \end{pmatrix} \tag{3.25}$$

Trong đó: F_i: Lực tác động lên phương i, i = x,y,z.

w_i: vận tốc Góc quay quanh trục i, i = x,y,z.

$T_i = T\phi, T\theta, T\Psi$: Moment xoắn

Hay

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} = -\frac{1}{M}\begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} + g\begin{bmatrix} -\sin\theta \\ \cos\theta\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} - \begin{pmatrix} \dot{\psi}\dot{v}_y - \theta\dot{v}_z \\ \dot{\phi}\dot{v}_z - \psi\dot{v}_x \\ \dot{\theta}\dot{v}_x - \phi\dot{v}_y \end{pmatrix} \quad (3.26)$$

Mô hình lực nâng của Quadrotor:

$$F = F_1 + F_2 + F_3 + F_4 \quad (3.27)$$

Mômen xoắn quanh góc Roll:

$$T\phi = -l(-F_2 + F_4) \quad (3.28)$$

Moment xoắn quanh góc pitch:

$$T\theta = l(F_1 - F_3) \quad (3.29)$$

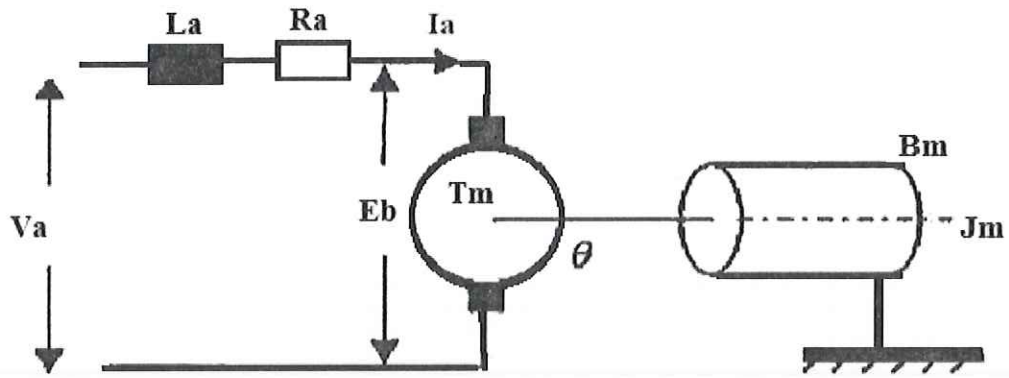
Moment xoắn quanh góc Yaw:

$$T\Psi = l(F_1 - F_2 + F_3 - F_4) \quad (3.30)$$

3.1.5 Mô hình toán động cơ BLDC

BLDC (brushless DC motor) sử dụng trong đề tài này là một động cơ một chiều DC nhưng nguyên lý hoạt động như một động cơ xoay chiều 3 pha.Vì khi điều khiển ta dùng mức áp DC để điều khiển hay dùng kỹ thuật PWM gióng như điều khiển động cơ DC thuần túy.

Vấn đề chuyển đổi từ mức áp DC nhận được qua tín hiệu PWM ở ngõ vào qua điện áp 3 pha cho động cơ là do khối driver ESC đảm nhiệm. Khối này sẽ nhận tín hiệu PWM ở ngõ vào và cho ra ở ngõ ra điện áp 3 pha lệch nhau 120^0 để cấp cho 3 pha của BLDC. Do đó trong trường hợp này ta có thể sử dụng mô hình toán của động cơ DC để thực hiện cho mô phỏng. Mô hình toán của động cơ DC như hình 3-3 [16].



Hình 3-3: Sơ đồ tương đương của động cơ BLDC

Ta có:

$$V_a(t) = R_a i_a(t) + L_a \frac{di_a(t)}{dt} + E_b(t) \tag{3.31}$$

$$E_b(t) = K_b \omega(t) \tag{3.32}$$

$$T_m(t) = K_t i_a(t) \tag{3.33}$$

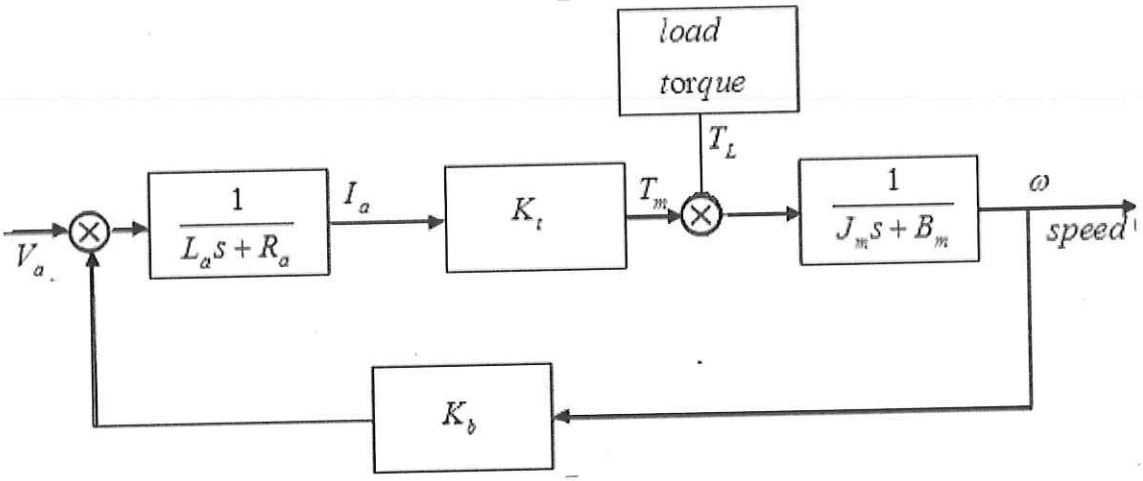
$$T_m(t) - T_L(t) = J_m B_m \omega(t) \tag{3.34}$$

Trong đó:

- V_a : điện áp phần ứng (v)
- R_a : trở kháng phần ứng (Ω)
- L_a : cuộn dây phần ứng (H)
- I_a : dòng phần ứng (A)
- E_b : Suất điện động (v)
- ω : tốc độ gốc (rad/s)
- T_m : động cơ mô-men xoắn (N.m)
- T_L : momen cản (N.m)
- θ : vị trí góc của trục rotor (rad)
- J_m = rotor quán tính (kgm^2)

- B_m = hệ số ma sát nhớt (NMS / rad).
- K_t = mô-men xoắn không đổi (Nm / A).
- K_b = hằng số sức phản điện động (Vs / rad).

Ta có mô hình động cơ BLDC như hình 3-4.



Hình 3-4: Sơ đồ khối của mô hình động cơ BLDC

V_a: điện áp cung cấp ngõ vào,

T_L: momen cản

ω : tốc độ gốc.

Thế (3.32) vào (3.31) và (3.33) vào (3.34) ta có:

$$V_a(t) = R_a i_a(t) + L_a \frac{di_a(t)}{dt} + K_b \omega(t) \tag{3.35}$$

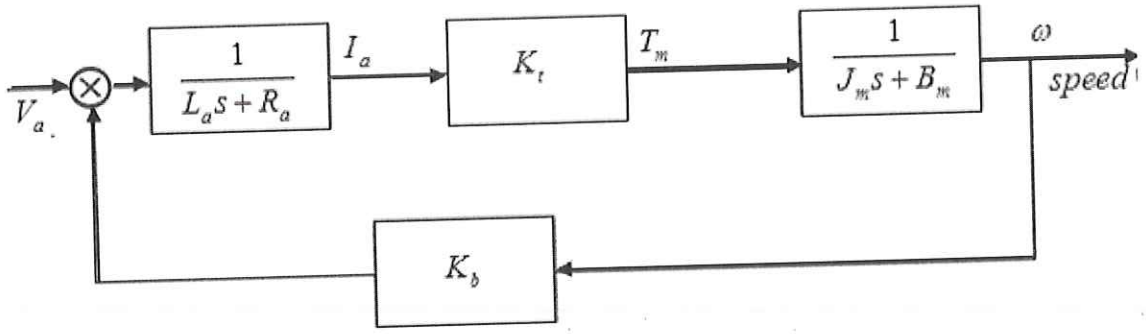
$$K_t i_a(t) = J_m \frac{d\omega(t)}{dt} + B_m \omega(t) \tag{3.36}$$

Biến đổi Laplace phương trình (2.38) và (2.39) ta có:

$$V_a(s) = R_a i_a(s) + sL_a i_a(s) + K_b \omega(s) \tag{3.37}$$

$$K_t i_a(s) = sJ_m\omega(s) + B_m\omega(s) \tag{3.38}$$

Khi T_L=0, ta có mô hình động cơ BLDC như hình 3-5.



Hình 3-5: Sơ đồ khối động cơ BLDC khi $T_L = 0$

Hình 3-5 cho thấy động cơ BLDC chạy với điều kiện không tải (lý tưởng)

Hàm truyền đạt giữa đầu vào điện áp và ngõ ra tốc độ động cơ như công thức (3.39).

$$\frac{\omega(s)}{V_a(s)} = \frac{K_t}{L_a J_m s^2 + (R_a J_m + L_a B_m)s + (R_a B_m + K_b K_t)} \tag{3.39}$$

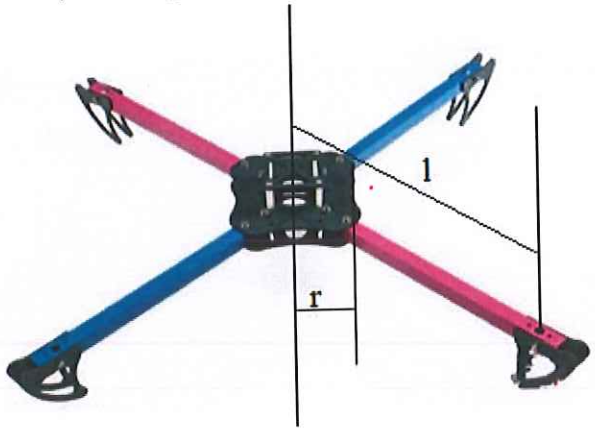
Trong đó:

$\omega(S)$: Vận tốc góc

$V_a(S)$: Điện áp đặt vào motor tương ứng giá trị PWM trong điều khiển

3.1.6 Khung mô hình:

Khung mô hình là một khung nhôm, kết cấu như hình 3-6.



Hình 3-6: Khung mô hình vật lý

Kích thước và khối lượng khung khác nhau sẽ có các giá trị quán tính của moment khác nhau. Đó là các giá trị của $I = [I_x, I_y, I_z]$, trong đó I_x, I_y, I_z là các giá trị moment quán tính trên trục x, y, và z. Như theo một số tài liệu tham khảo, giá trị này được tính toán như công thức (3.40) và (3.41) [9] :

$$I_x = I_y = \frac{Mr^2}{2} + \frac{Mh^2}{6} + 2Ml^2 + \frac{m_m r_m^2}{3} \quad (3.40)$$

$$I_z = \frac{m_m r_m^2}{2} + 4Ml^2 \quad (3.41)$$

Trong đó:

M: là khối lượng của Quadrotor

r: bán kính vòng tròn tâm của mô hình

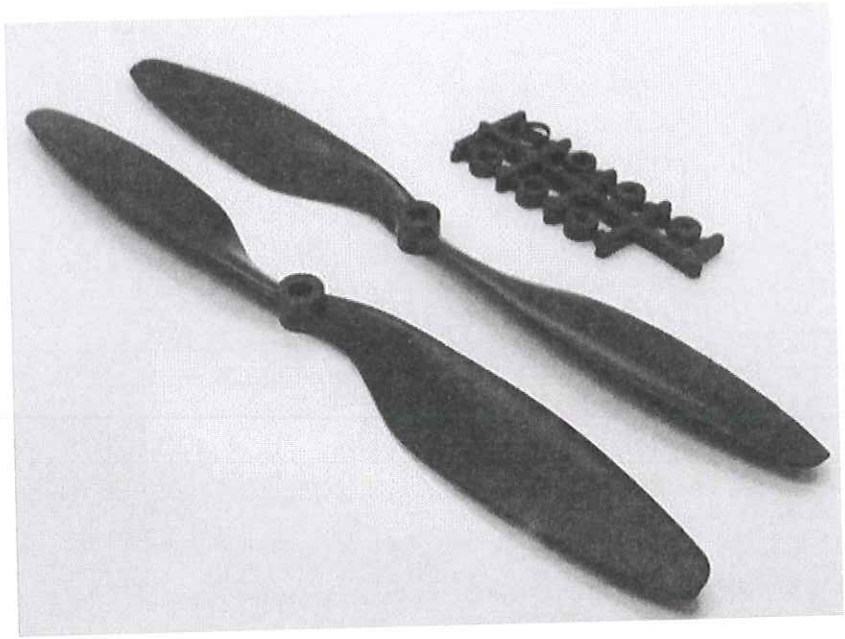
m_m: khối lượng của motor

r_m : Bán kính trục của motor

I: chiều dài cánh tay của Quadrotor

3.1.7 Cánh quạt

Trong đề tài này, cánh quạt sử dụng loại hai lá, có bán kính r_{pro}= 12,75 cm, như hình 3-7.



Hình 3-7: Cánh quạt hai lá

Để đơn giản hóa, phần ma sát của không khí tác động lên cánh quạt và góc nghiêng của cánh quạt có thể bỏ qua. Lúc này theo lý thuyết về động lượng tạo ra của cánh quạt sẽ gồm hai thành phần: lực F_{pro} và moment M_{pro} sẽ tỉ lệ với bình phương của vận tốc góc quay ω và được tính bằng hai công thức sau:

$$F_{pro} = \rho \pi r_{pro}^2 C_T \omega^2 = K_T \omega^2 \tag{3.42}$$

$$M_{pro} = \rho \pi r_{pro}^5 C_p \omega^2 = K_q \omega^2 \tag{3.43}$$

Trong đó:

C_T: Hằng số của lực đẩy ngang

C_p: Hằng số của lực nâng

ρ : Mật độ không khí

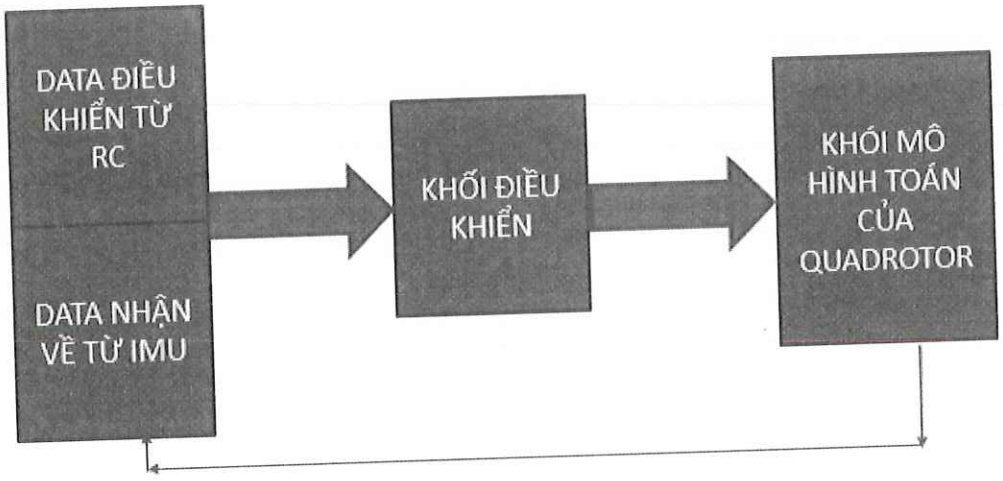
r_{pro}: Bán kính của cánh quạt

K_r: Hệ số quan hệ giữa lực đẩy ngang và vận tốc quay của cánh quạt

K_q: Hệ số quan hệ giữa lực nâng và vận tốc quay của cánh quạt

3.2 Mô hình mô phỏng

Mô hình Quadrotor thực tế có thể chia ra làm nhiều modun con như: khối ngõ vào, khối xử lý và khối cơ cấu chấp hành. Ta có sơ điều điều khiển như hình 3-8:



Hình 3-8: Sơ đồ điều khiển Quadrotor

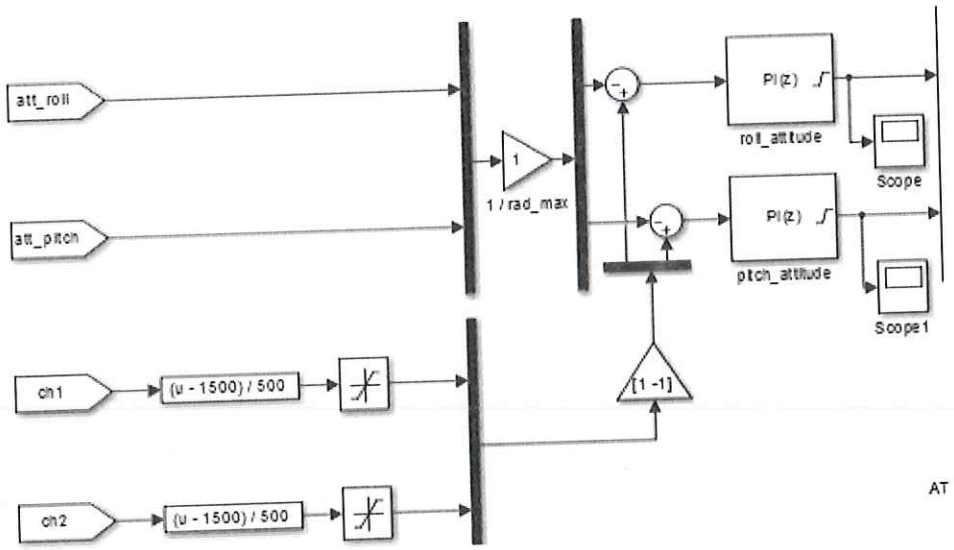
3.2.1 Khối tín hiệu ngõ vào

Tín hiệu ở ngõ vào được chia làm hai loại:

Tín hiệu từ bộ điều khiển RC, giá trị của các góc Roll, pitch, yaw và lực tạo ra của khối này đóng vai trò là giá trị mong muốn, cung cấp ngõ vào mong muốn cho bộ điều khiển PI.

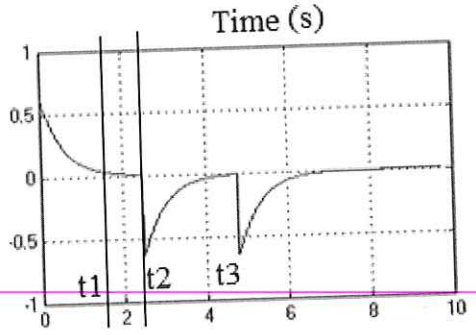
Tín hiệu roll, pitch, yaw nhận được từ cảm biến, thì đây là giá trị trạng thái hiện tại của máy bay, giá trị này là ngõ vào thứ hai của bộ điều khiển PI. Khối PI có nhiệm vụ điều chỉnh sau cho giá trị trả về từ IMU phải sắp sĩ bằng với góc mong muốn điều khiển, lúc đó giá trị sai số *e* xem như bằng không.

Trong matlab Simulink, ta thực hiện kết nối như hình 3-9.



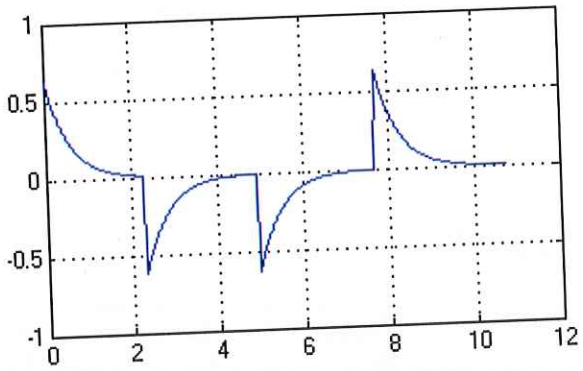
Hình 3-9: Khối điều khiển cân bằng

Kết quả đáp ứng của góc roll, pitch sau bộ PI được thể hiện như hình 3-10 và hình 3-11.



Hình 3-10:Đáp ứng góc Roll sau bộ PI

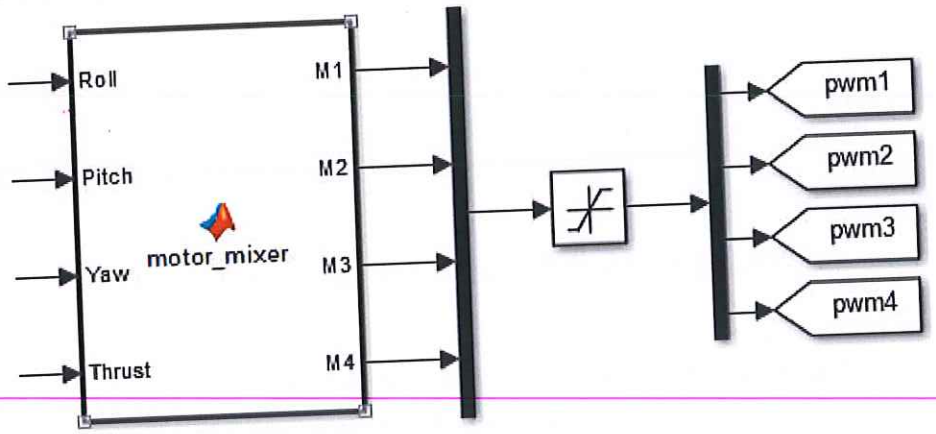
Theo hình 3-10, trục nằm ngang là trục thời gian t, đơn vị tính là giây (s), trục thẳng đứng thể hiện độ lớn của sai số *e*, đơn vị là (độ/s). Ta thấy rằng, tại thời điểm ban đầu t=0 khi ta đặt kênh CH1 ở mức 1, thì sau khoảng 1.5s, tại thời điểm t2 thì thân quadrotor đã nghiên đúng góc mong muốn điều khiển. Tại thời điểm t3=1.5s, ta cho kênh CH1 ở mức -1, sau thời gian 1.5s thì máy bay lại trả về trạng thái như mong muốn và tương tự cho các trường hợp tiếp theo.



Hình 3-11: Đáp ứng góc Pich sau bộ PI

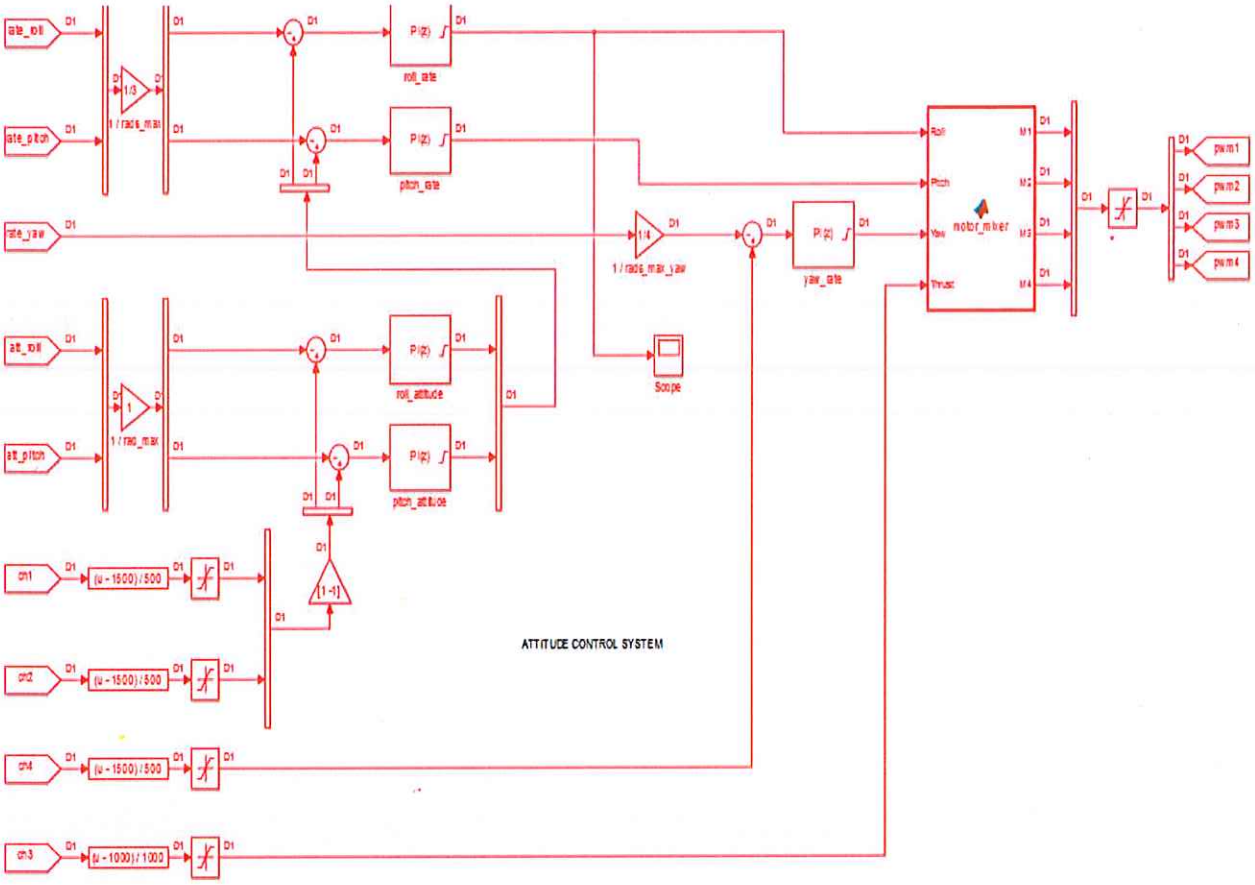
3.2.2 Khối tính toán, ước lượng điều khiển (control system)

Từ các công thức lực và moment của mô hình trong mục 3.1.4, ta xây dựng được hàm ước lượng giá trị điều khiển PWM cho bốn động cơ như hình 3-12 (Chi tiết xin xem phụ lục C).



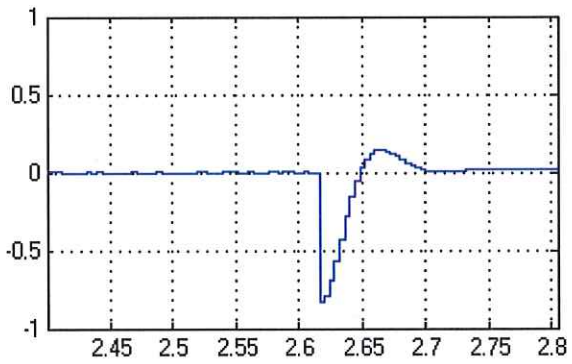
Hình 3-12: Mô hình ước lượng giá trị điều khiển motor

Từ các bước trên ta kết hợp lại và tạo thành khối điều khiển cho Quadrotor như hình 3-13.



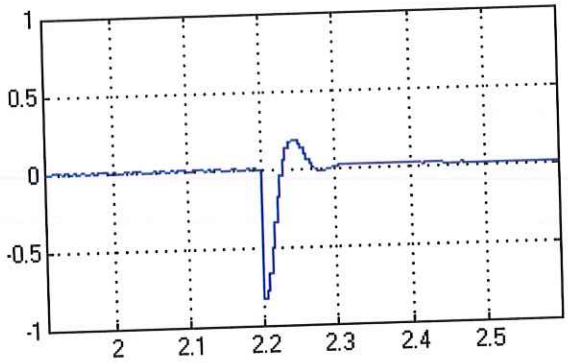
Hình 3-13: Khối mô phỏng điều khiển Quadrotor

Chạy mô phỏng trên Matlab Simulink ta được kết quả như hình 3-14, 3-15, 3-16 là gia tốc góc của góc roll, pitch và yaw.



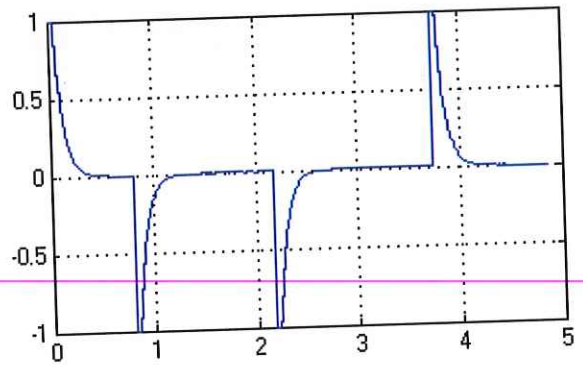
Hình 3-14: Đáp ứng góc Acc roll sau bộ PI

Nhận xét: Theo kết quả hình 3-14, ta thấy hệ thống đáp ứng khá nhanh với t≈0,8 giây, thì hệ thống đạt được gia tốc mong muốn.



Hình 3-15:Đáp ứng góc Acc pitch sau bộ PI

Nhận xét: Theo hình 3-15, thì sau thời gian t≈0,1 giây thì hệ thống đã đạt được giá trị Acc pitch mong muốn.

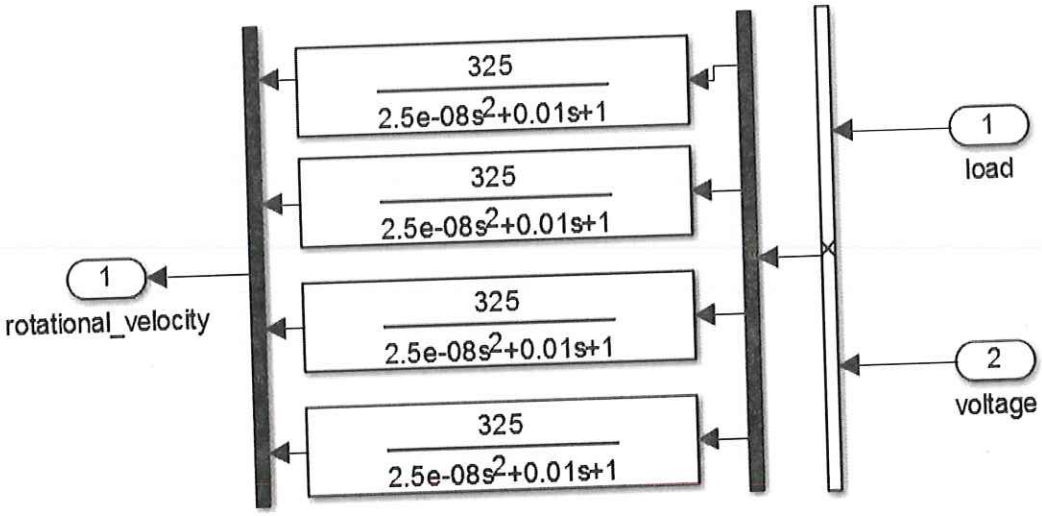


Hình 3-16: Đáp ứng gia tốc góc yaw sau bộ PI

Nhận xét: Dựa vào hình 3-14 đến 3-16, ta thấy sau khi qua bộ điều khiển PI, thì ngõ ra của nó nhanh chống đạt đến trạng thái mong muốn sau thời gian khá ngắn, lúc này sai số ngõ vào $e_{i,i=\emptyset,\theta,\Psi} = 0$. Tín hiệu ngõ ra của các khối PI, được đưa đến khối tính lực cần thiết để cấp xung PWM đến điều khiển từng động cơ.

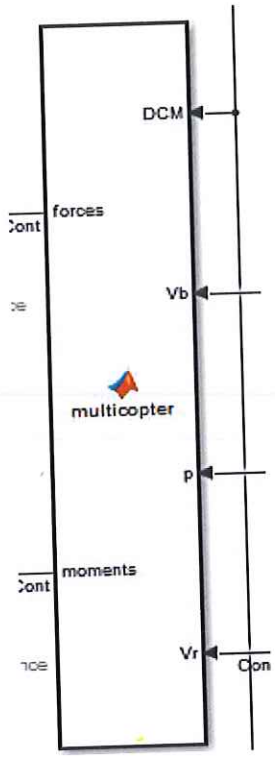
3.2.3 Khối mô hình động lực Quadrotor

Từ phương trình toán của động cơ DC (công thức 3.39), và các thông số kỹ thuật của động cơ BLDC EMAX XA2212, ta tính được phương trình toán cho bốn động cơ và thực hiện mô phỏng trong Matlab như hình 3-17.



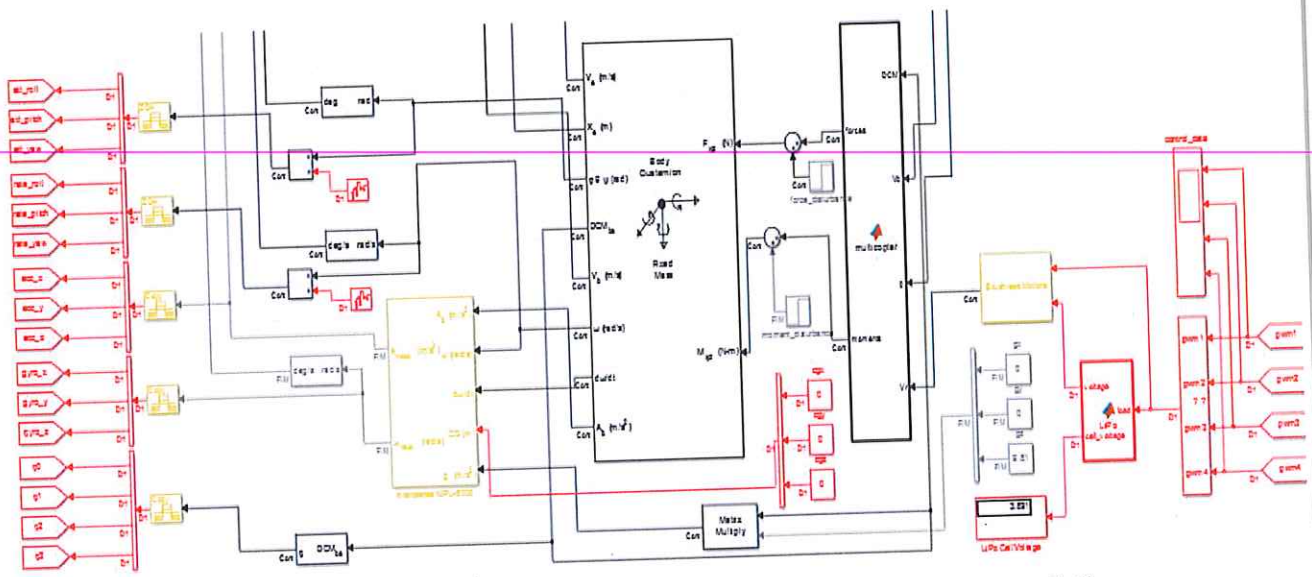
Hình 3-17: Phương trình toán động cơ BLDC

Theo các công thức toán về ma trận cosine và các công thức tính lực và moment của hệ thống (từ công thức 3.4, 3.27, 3.30) ta đã xây dựng được hàm mô phỏng lực nâng như hình 3-18, (hàm matlab được thể hiện trong phụ lục D).



Hình 3-18:Khối thực hiện hàm mô phỏng tính toán lực và moment của hệ thống

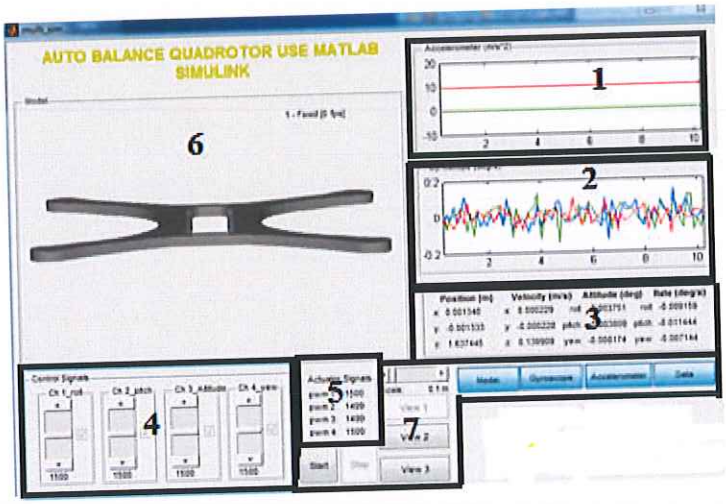
Từ các kết quả trên kết hợp với thư viện của cảm biến IMU ta có mô hình động lực của Quadrotor như hình 3-19.



Hình 3-19: Mô hình động lực Quadrotor (Dynamic model)

3.2.4 Khối giao diện điều khiển

Để thực hiện truyền dữ liệu từ bộ điều khiển trong mô phỏng, ta dùng giao diện GUI để thiết lập các giá trị CH1, CH2, CH3, CH4. Chi tiết giao diện điều khiển được trình bày như hình 3-20.



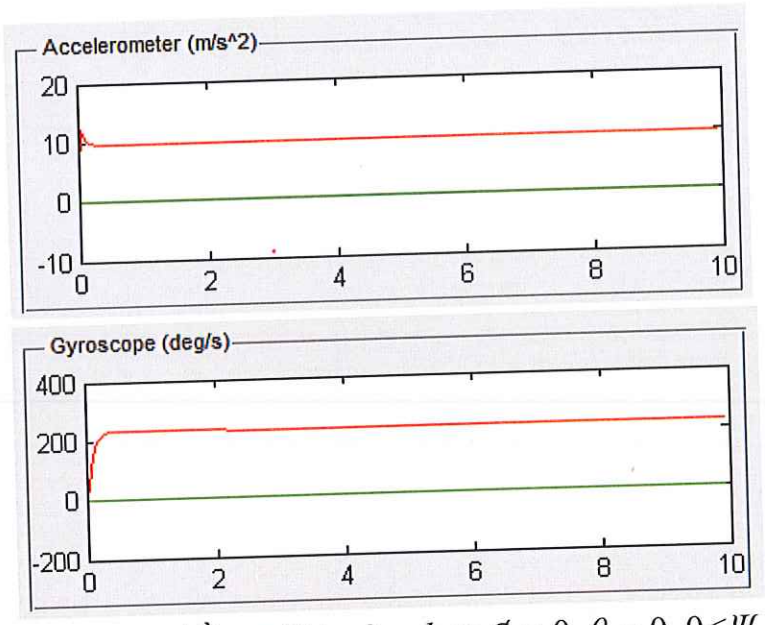
Hình 3-20: Giao diện điều khiển Quadrotor trong mô phỏng

Trong đó:

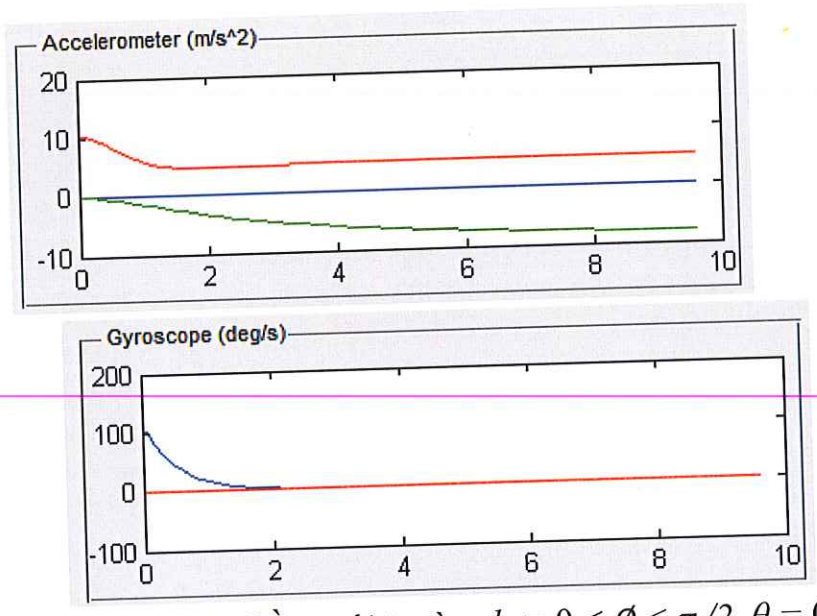
1. Đồ thị gia tốc góc roll, pitch và yaw
2. Đồ thị góc quay góc roll, pitch, yaw
3. Giá trị hiện tại của các trạng thái của Quadrotor
4. Khối tín hiệu điều khiển Radio controller.
5. Giá trị PWM sau khi tính toán, cung cấp đến ESC điều khiển động cơ.
6. Khối hiển thị mô hình Cad của quadrotor
7. Khối nút lệch (chi tiết được thể hiện tại phụ lục E)

3.3 Kết quả mô phỏng Quadrotor

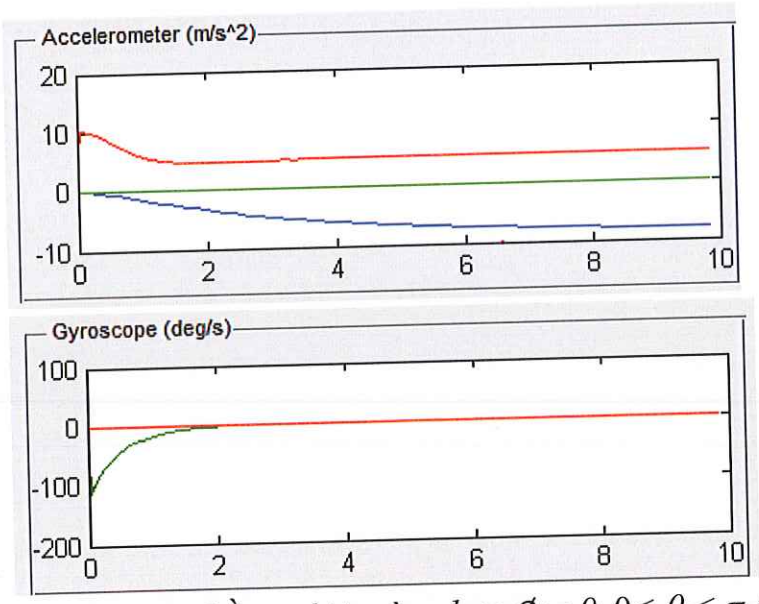
Sau khi thực hiện mô phỏng Quadrotor trên Matlab Simulink, ta đạt kết quả như sau:



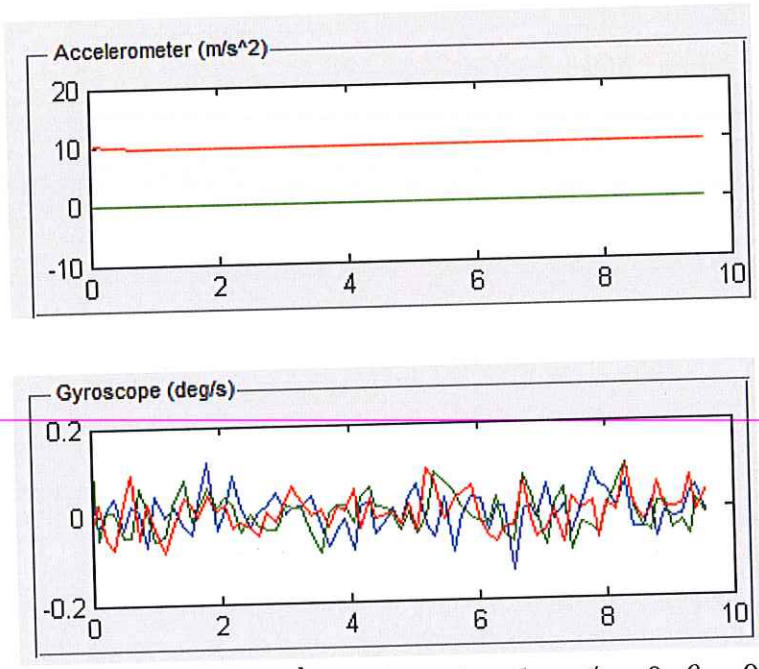
Hình 3-21: Đáp ứng cân bằng với trường hợp $\emptyset = 0$, $\theta = 0$, $0 < \Psi < \pi/2$



Hình 3-22: Đáp ứng cân bằng với trường hợp $0 < \emptyset < \pi/2$, $\theta = 0$, $\Psi = 0$



Hình 3-23: Đáp ứng cân bằng với trường hợp $\emptyset = 0$, $0 < \theta < \pi/2$, $\Psi = 0$



Hình 3-24: Đáp ứng cân bằng với trường hợp $\emptyset = 0$, $\theta = 0$, $\Psi = 0$

3.4 Nhận xét

Từ kết quả mô phỏng trên, ta thấy hệ thống đạt được độ ổn định cao. Dựa vào đồ thị hình 3-21 đến hình 3-24 ta thấy sai số của góc Roll, Pitch, Yaw là khá nhỏ $-0.2 < \emptyset, \theta, \Psi < 0.2$ do đó hệ thống đạt trạng thái cân bằng khá ổn định.

Chương 4
MÔ HÌNH VẬT LÝ VÀ BAY THỰC NGHIỆM

Trong chương này, tác giả đề tài triển khai trên mô hình vật lý và nhúng thuật toán điều khiển vào board Arduino 2560 và đã cho bay thử để kiểm chứng lại kết quả.

4.1 Mô hình vật lý

4.1.1 Kung mô hình

Khung mô hình gồm bốn tay nhôm có chiều dài l=1,5 cm, được gắn cố định đối xứng vào bốn góc của một đĩa hình vuông. Tại bốn đỉnh của thanh nhôm ta gắn bốn động cơ BLDC như hình 4-1.

Hình 4.1: Khung mô hình Quadrotor

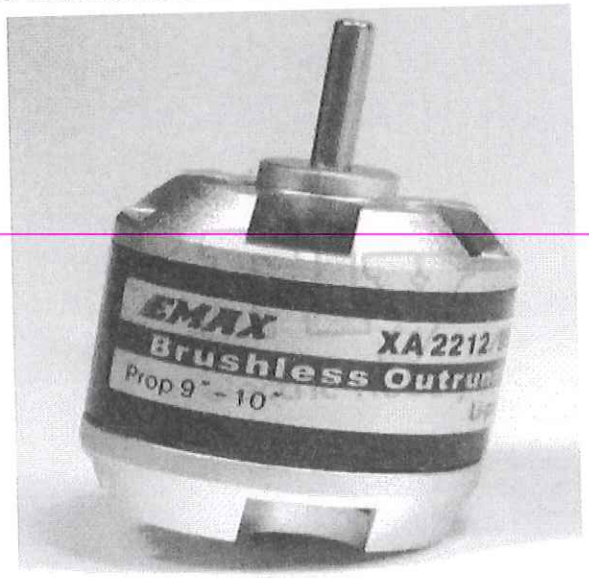
4.1.2 Động cơ:

Động cơ BLDC được sử dụng trong đề tài có các thông số kỹ thuật như bảng 4-1.

Bảng 4-1: Bảng thông số của động cơ A2212 sử dụng trong mô phỏng

Số vòng/volt (KV)	980
Trọng lượng (m)	52g
Đường kính trục	3mm
Chiều dài	43.5
Đường kính rotor	28,5mm
Điện trở R	0.09Ω
Vcc	12V
Hiệu suất tối đa	80%

Hình dạng thực tế của motor như hình 4-2.



Hình 4-2: Động cơ không chổi than (BLDC) A212

Trên lý thuyết thì bốn động cơ có các thông số kỹ thuật như nhau, hệ số KV như nhau nhưng trên thực tế thì đáp ứng của mỗi động cơ điều khác nhau. Do đó, để tăng thêm độ chính xác trong điều khiển ta cần khảo sát đáp ứng của từng motor.

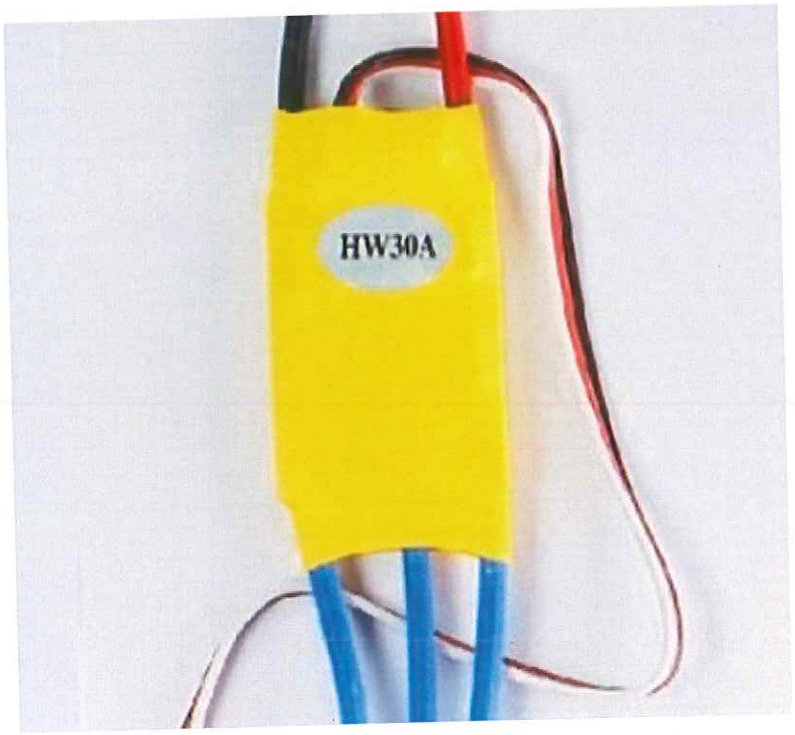
4.1.3 Mạch công suất ESC

Mạch ESC là một mạch điện tử công suất, được dùng để điều khiển điện áp ba pha ngõ ra để cấp cho động cơ BLDC. Mạch ESC dùng để điều khiển tốc độ của động cơ BLDC, ESC sử dụng trong đề tài có thông số kỹ thuật như bảng 4-2 .

Bảng 4-2: Thông số kỹ thuật của ESC

Vcc	*6-12V*
Imax	*30A*
Trọng lượng	*26g*

Mạch ESC điều khiển động cơ BLDC như hình 4-3.



Hình 4-3: Mạch ESC

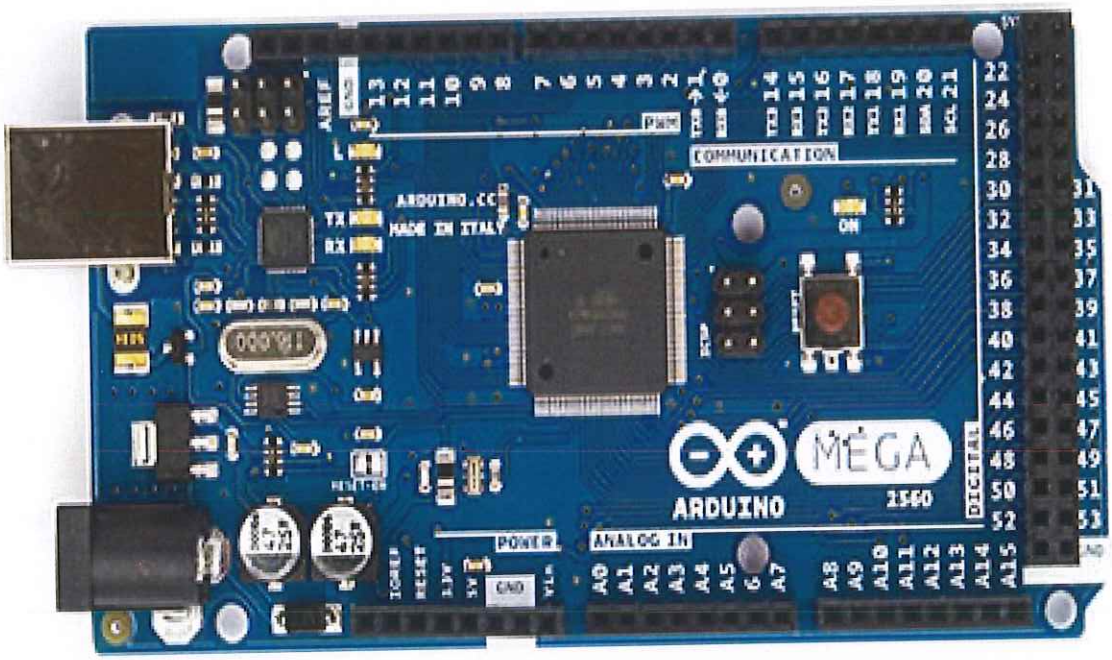
Trên thực tế thì đáp ứng của từng ESC cũng khác nhau khi cấp một tín hiệu PWM ở đầu vào. Khi ESC hoạt động thì tín hiệu PWM phải nằm trong khoảng [PWM_{min} , PWM_{max}], sau khi đo kiểm tra ta thu được bảng thông số như bảng 4-3.

Bảng 4-3: Giới hạn và độ lớn của từng PWM cho từng ESC

	PWM_{min}	PWM_{max}
ESC1	0,94 ms	0,193 ms
ESC2	0,95 ms	0,192 ms
ESC3	0,92 ms	0,193 ms
ESC4	0,95 ms	0,195 ms

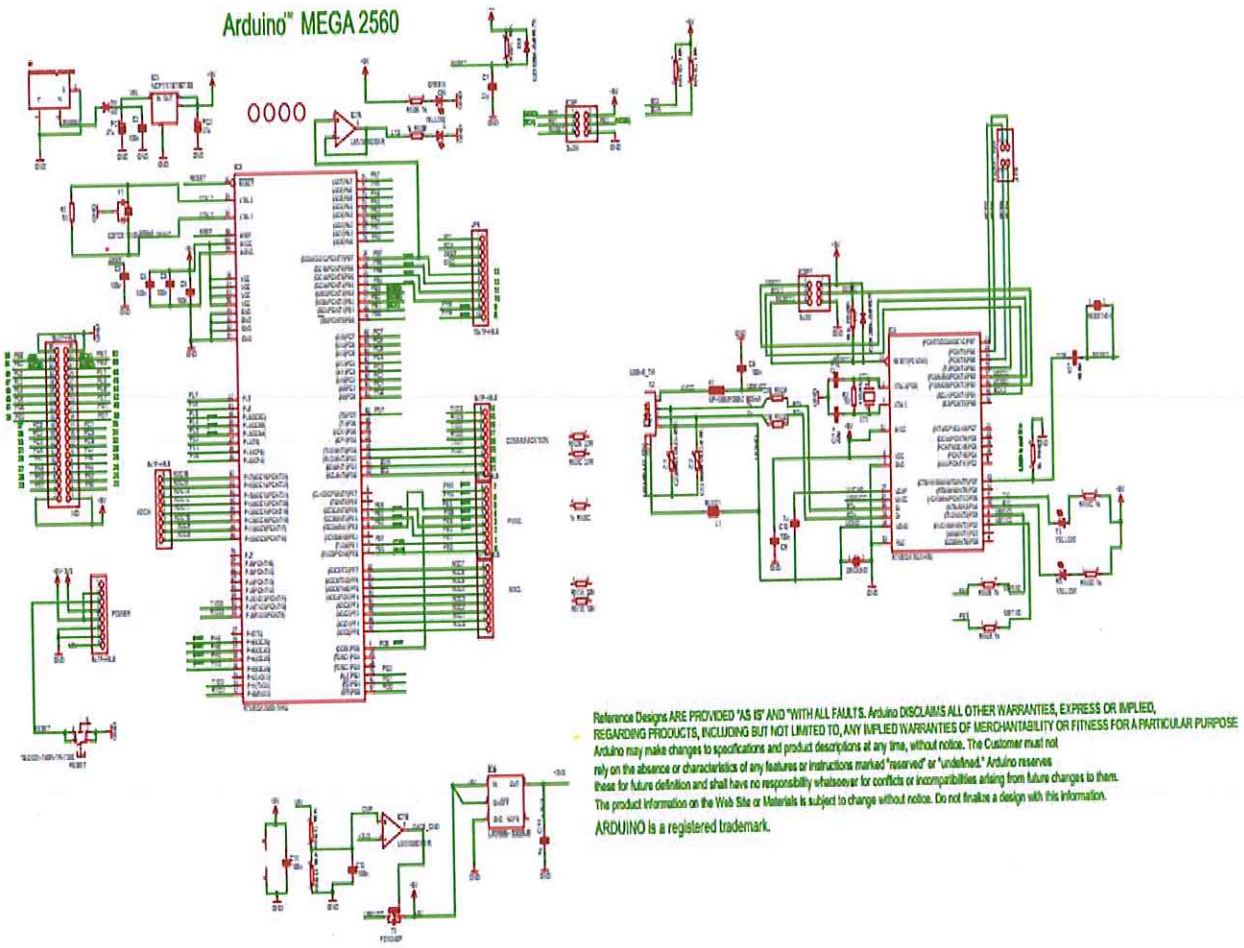
4.1.4 Mạch điều khiển trung tâm

Khối điều khiển trung tâm là một board Arduino 2560. Các thông số chi tiết của board như đã được giới thiệu trong mục 3.2.3. Hình dạng board như trong hình 4-4.



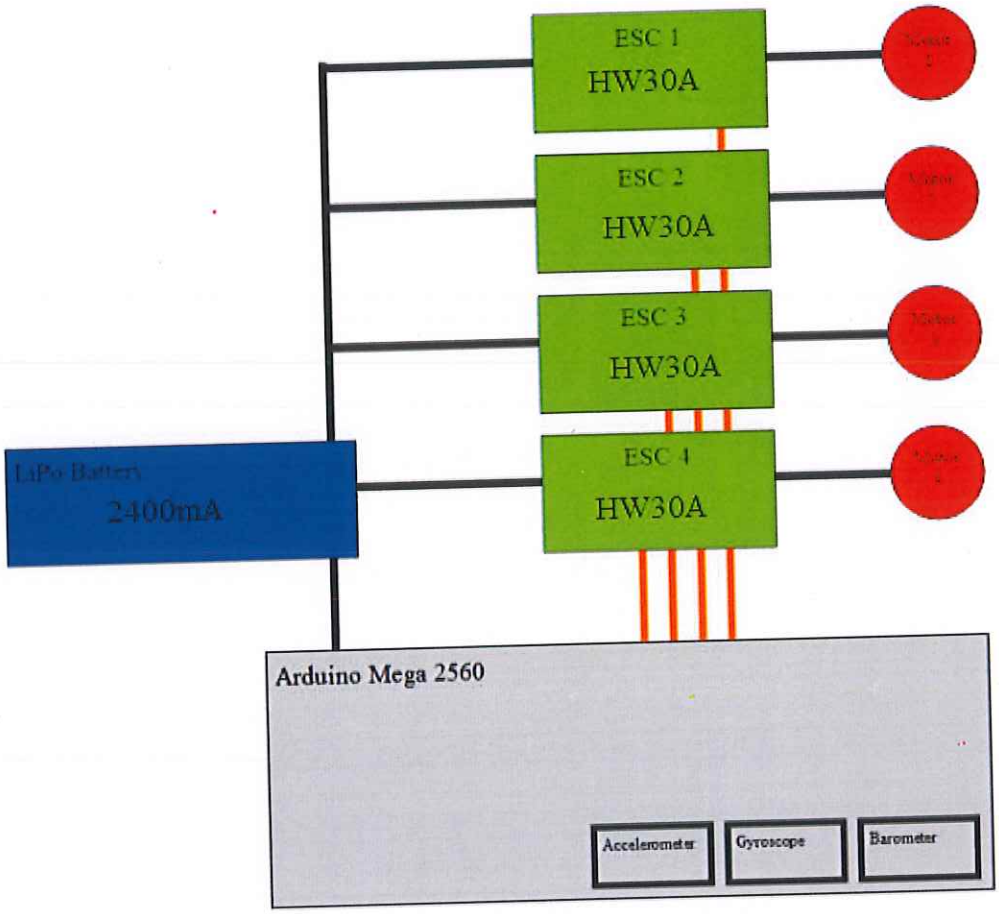
Hình 4-4: Khối điều khiển trung tâm dùng Board Arduino 2560

Sơ đồ nguyên lý của Board Arduino 2560 như hình 4-5 [16].



Hình 4-5: Sơ đồ nguyên lý của Board Arduino 2560

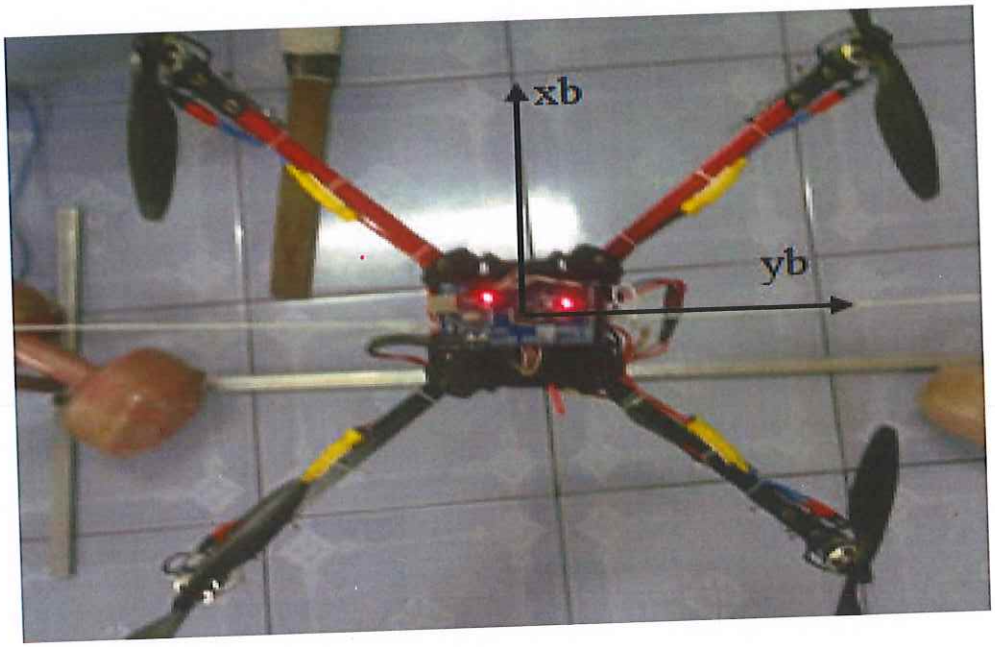
Cuối cùng ta thực hiện kết nối phần cứng như hình 4-6.



Hình 4-6: Sơ đồ kết nối phần cứng của Quadrotor

4.1.5 Mô hình vật lý hoàn chỉnh

Quadrotor sau khi gắn đầy đủ các thành phần cần thiết và có hình dáng như hình 4-7. Mô hình Quadrotor này có tâm của trọng lượng trùng với tâm của dĩa hình vuông, và điểm này sẽ được dùng để thực hiện các phép quy đổi giữa hai hệ tọa độ $(x, y, z)_B$ và $(x, y, z)_E$.



Hình 4-7: Mô hình Quadrotor hoàn chỉnh

4.2 Bay thực nghiệm

Để thực hiện bước nhúng thuật toán trên Matlab Simulink, trước hết cần chuẩn bị môi trường làm việc giữa Matlab Simulink và Arduino, các bước thực hiện như sau:

Bước 1: Cài đặt môi trường code cho arduino, link cài đăt: www. Arduino.com

Bước 2: Cài đặt tool hỗ trợ của Matlab cho board Arduino [13], [16].

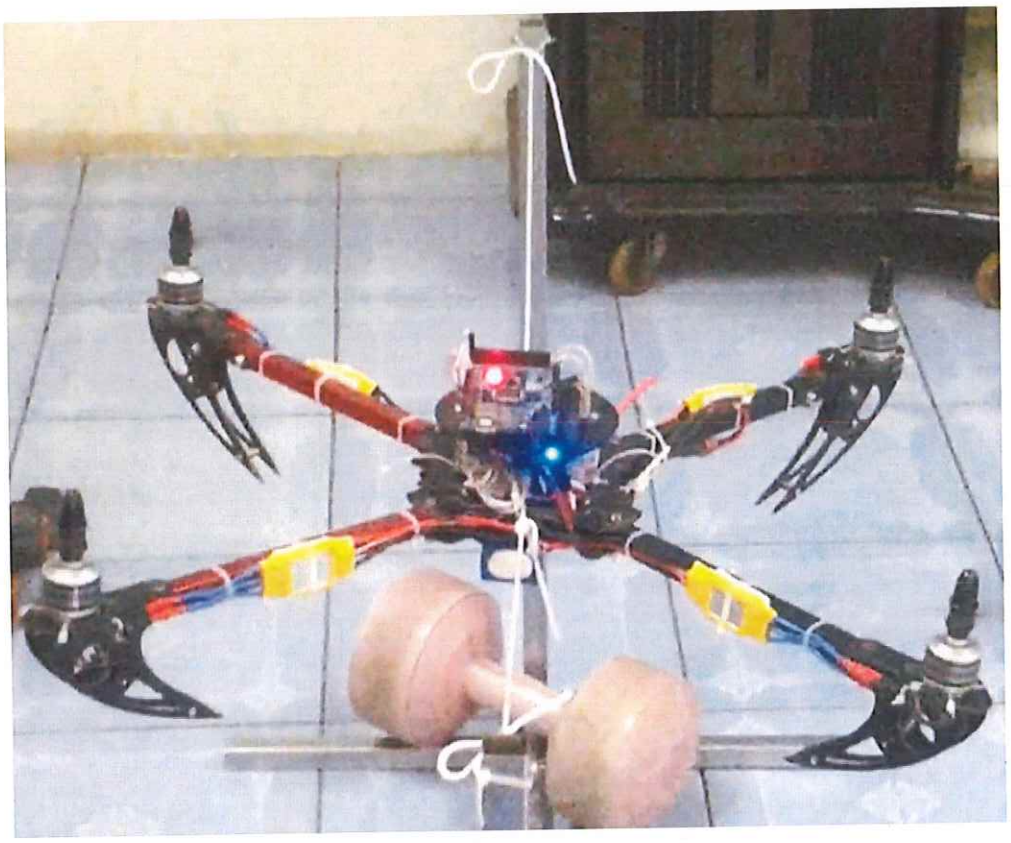
Bước 3: Kết nối board Arduino với máy tính qua cổng USB và lúc này ta có thể thực hiện nhúng thuật toán vào board arduino 2560.

4.2.1 Kiểm tra đáp ứng cân bằng

Đầu tiên để thử kiểm chứng lại thuật toán, ta tiến hành kiểm tra đáp ứng cân bằng của Quadrotor trên từng trục x, và trục y. Ta cố định theo phương trục y trên giá đỡ, và phương trục x ta để tự do, sau cho nó có thể chuyển động lên xuống quanh trục y như hình 4-7. Trường hợp này ta kiểm chứng đáp ứng của góc pitch trên mô hình.

Khi mô hình ở trạng thái cân bằng quanh trục y như hình 4-7, để kiểm chứng đáp ứng của góc pitch, ta tác động 1 lực lên cánh tay mô hình, lúc này mô

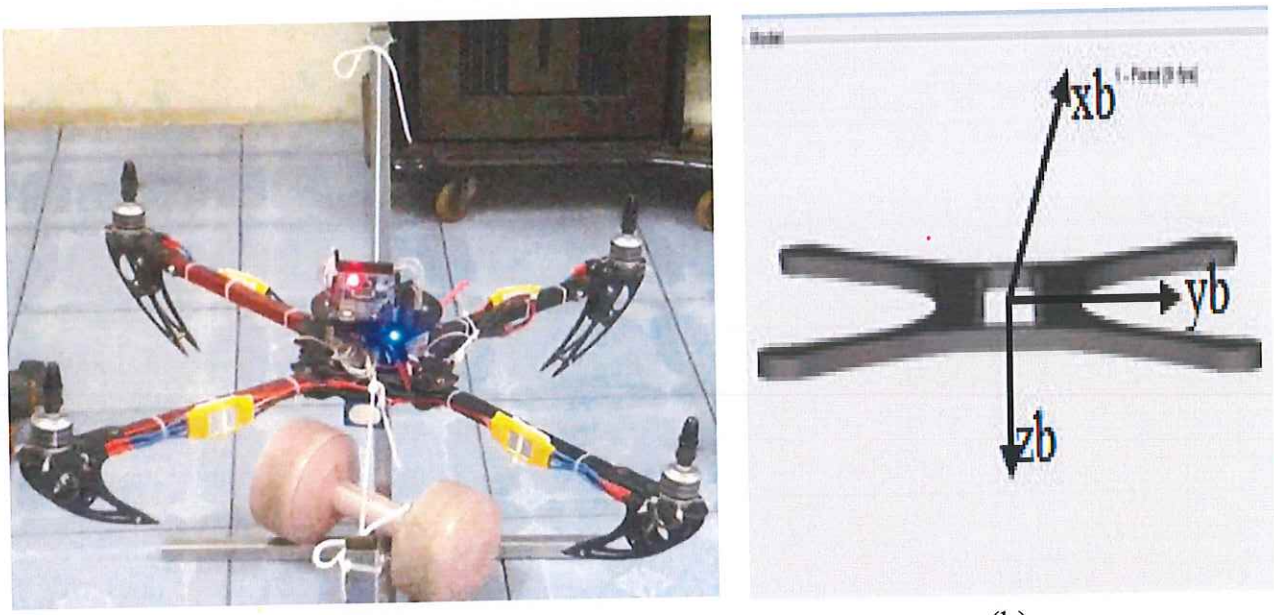
hình sẽ quay quanh trục x, làm lệch một góc Ø, $e_{\emptyset} \neq 0$, khi thôi tác dụng lực thì ta thấy mô hình tự động trở về vị trí cân bằng ban đầu. Tương tự cho trường hợp góc roll, ta cố định trên phương trục x trên giá đỡ, và trên phương trục y để dao động tự do, và kết quả ta thu được tương tự cân bằng trên góc Roll như hình 4-8.



Hình 4-8: Kiểm tra đáp ứng cân bằng trên trục x

Để dễ dàng so sánh giữa mô phỏng và thực tế, ta lần lược điều khiển ở từng điều kiện trên hai mô hình. Từ đó, ta đã thu được các trường hợp như sau:

Khi mô hình ở trạng thái cân bằng với $\emptyset = 0$, $\theta = 0$, $\Psi = 0$, ta được hình 4-9.

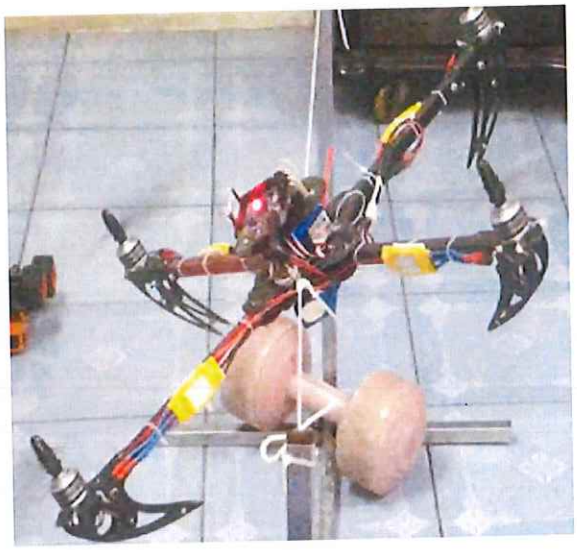


(a) (b)

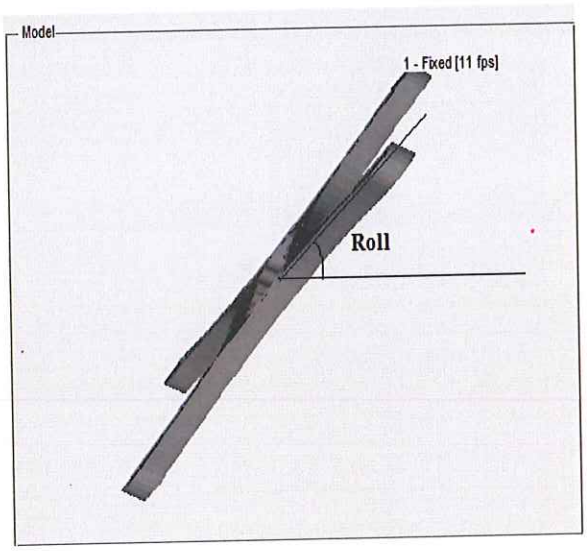
Hình 4-9: Quadrotor ở trạng thái cân bằng quanh trục, (a) mô hình vật lý, (b) mô hình mô phỏng

Trong mô phỏng, khi ta thay đổi các giá trị thanh trượt của bốn kênh CH1, CH2, CH3 và CH4 thì mô hình CAD sẽ thay đổi tương ứng theo giá trị của kênh đó. Trong mô hình thực thì ta dùng bộ điều khiển RC bốn kênh để điều khiển. Trong thực nghiệm, khi ta thay đổi trạng thái của tay điều khiển thì mô hình Quadrotor lập tức thay đổi trạng thái tương ứng với góc điều khiển của tay game.

Khi ta thay đổi góc dương $\emptyset > 0$, $\theta > 0$ thì mô hình dịch chuyển thay đổi góc nghiên như hình 4-10, tương tự như mô hình CAD trong mô phỏng.



(a)

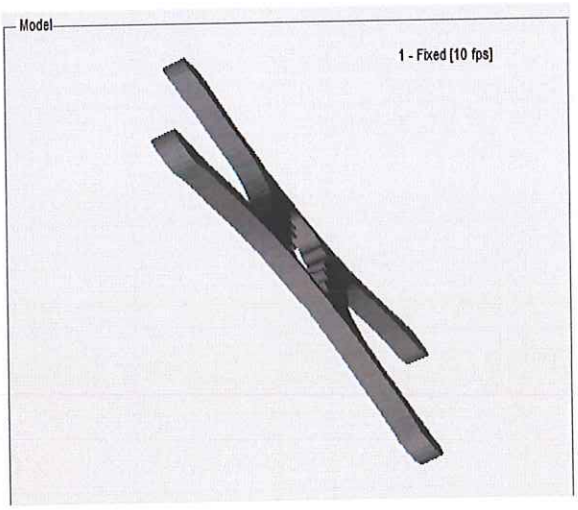


(b)

Hình 4-10: Trạng thái mô hình khi $\emptyset > 0$, $\theta > 0$, (a) mô hình thực, (b) mô hình mô phỏng

Tương tự khi ta thay đổi góc âm $\emptyset < 0$, $\theta < 0$, thì mô hình dịch chuyển thay đổi góc nghiên như hình 4-11, tương tự như kết quả mô phỏng. Tuy nhiên giá trị góc nghiên hoàn toàn khác nhau. Đây là sự khác biệt giữa mô phỏng và thực tế.

(a)



(b)

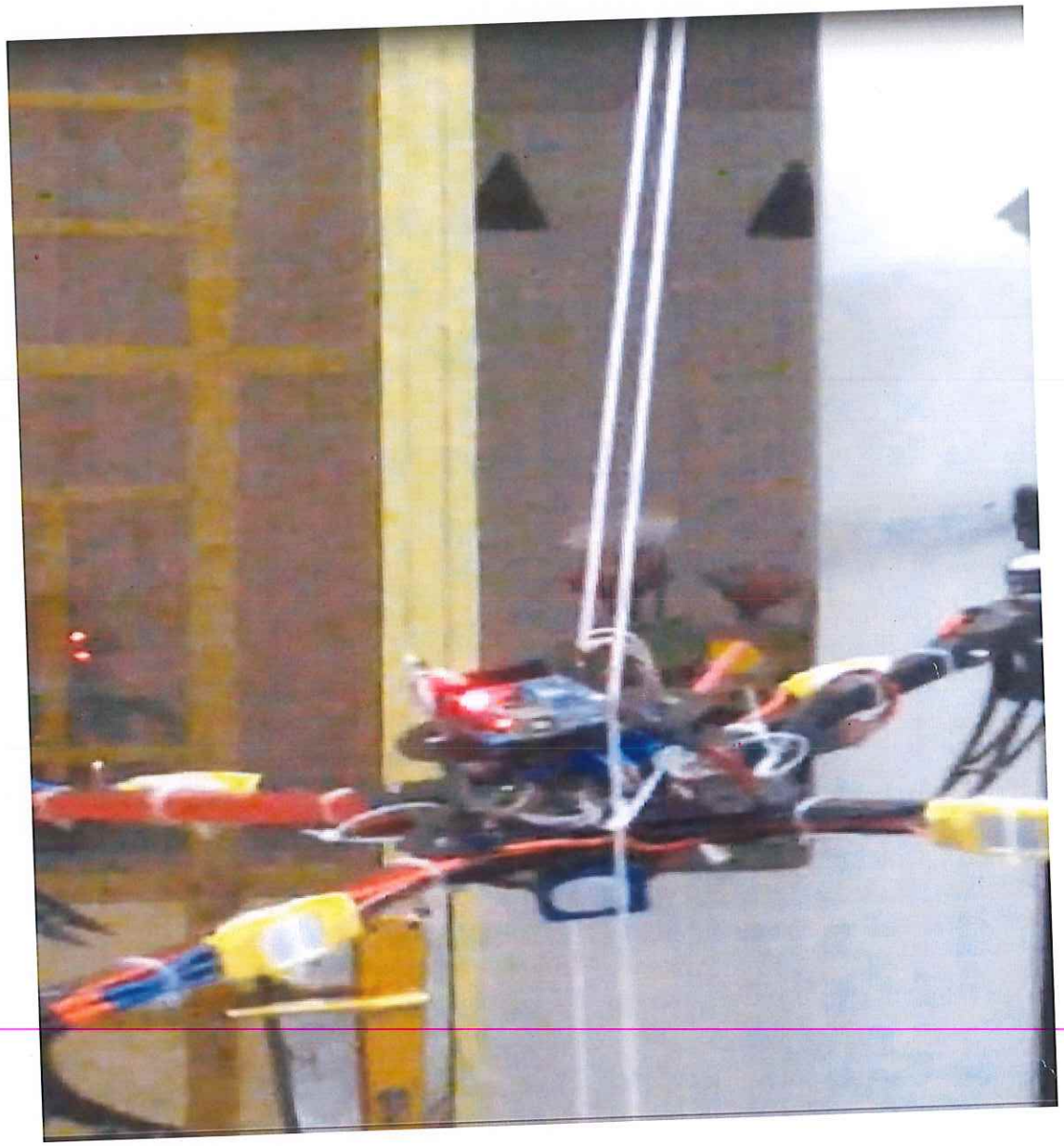
Hình 4-11: Trạng thái mô hình khi Ø < 0, θ < 0, (a) mô hình thực, (b) mô hình mô phỏng

Nhận xét: Từ mô hình thực và mô hình mô phỏng như trong các hình 4-9, 4-10, và 4-11, ta có nhận xét như sau:

- Phương điều khiển trong thực tế và mô phỏng là như nhau.
- Giá trị góc lệch giữa thực tế và mô phỏng khác nhau, góc lệch trong mô phỏng lớn hơn góc lệch trên mô hình thực tế.

4.2.2 Kiểm tra đáp ứng cân bằng khi có lực nâng

Để kiểm tra được lực nâng của mô hình vật lý, ta thiết lập hai dây song song từ trần nhà xuống, sao cho mô hình vật lý trượt được trên hai dây khi có tác động của lực nâng. Mô hình thể hiện như hình 4-12.



Hình 4-12: Kiểm tra lực nâng của mô hình

Qua nhiều lần tiến hành bay thử, ta đã thấy rằng hệ thống dao động khá nhiều khi bay, sự tự ổn định của hệ thống vẫn chưa đạt. Biên độ ổn định rất nhỏ, dẫn đến hệ thống dễ mất cân bằng do các yếu tố ngoại lực bên ngoài tác động.

KẾT LUẬN

Trong quá trình phân tích mô hình máy bay bốn cánh, chúng ta có thể cho rằng Quadrotor là một hệ dao động sáu bật tự do. Từ những thông số làm thay đổi trạng thái mô hình, chúng ta đã tìm ra được các mô hình toán tương ứng của các thành phần cấu thành của mô hình. Dựa vào các mô hình toán học, chúng ta dùng Matlab Simulink phiên bản 2014a để tiến hành mô phỏng và đã đạt được một số kết quả như đã được trình bày trong chương 4.

Từ những kết quả đạt được từ mô phỏng, chúng ta tiến hành cho xây dựng mô hình thực và quan sát các kết quả thực tế của nó như trong chương 4. Từ hai kết quả này chúng ta thấy chúng có những kết quả tương đồng như nhau nhưng chúng không đồng nhất với nhau về góc lệch. Chẳng hạn như chúng ta quan sát hình 4-10, khi tín hiệu điều khiển góc roll trên mô phỏng và trên mô hình thực là như nhau (2000s) thì góc lệch đo được của mô hình thực là $\varnothing$ gần bằng 32 độ, trong khi đó góc lệch trong mô hình CAD thì lớn hơn, tương tự cho các góc lệch khác, góc lệch trên mô hình mô phỏng luôn lớn hơn góc lệch trên mô hình thực tế.

Từ những sai biệt về giá trị của góc lệch, nó đã làm thay đổi về về khả năng tự cân bằng của mô hình thực. Nguyên nhân của sự chênh lệch kết quả giữa mô phỏng và thực tế này là do một số nguyên nhân tác động lên mô hình thực tế mà ta không có đề cập đến trong quá trình mô phỏng như:

- Mật độ không khí của môi trường
- Ảnh hưởng của các tiếng ồn xung quanh, tiếng ồn do bốn động cơ đã gây ra nhiễu trên cảm biến IMU.
- Những sai số của mô hình cơ khí, trọng lượng phân bố trên mô hình thực không đồng điều tuyệt đối.
- Những sai số do các mạch điện tử tạo ra.

Đây là một số yếu tố làm hạn chế khả năng tự cân bằng trên mô hình thực tế.

Kết quả đạt được:

- Đã hoàn thiện mô hình mô phỏng trên Matlab Simulink dùng bộ điều khiển PID.
- Đã thi công phần cứng, chương trình điều khiển trên mô hình vật lý được viết trên môi trường Arduino IDE.

Hạn chế của đề tài:

- Mô hình còn dao động quanh điểm cân bằng khá lớn.
- Chưa thiết kế được bộ điều khiển LQR + Bộ lọc Kalman.
- Khả năng tự cân bằng chưa thể hiện được bằng thực nghiệm
- Chưa thực hiện kết nối được giữa matlab Simulink và arduino, nên đã làm tăng thêm sự khác biệt về kết quả giữa hai mô hình.

Hướng phát triển:

Ngoài việc khắc phục những mặt hạn chế của đề tài, Chúng ta có thể kết hợp phương pháp nhận dạng tối ưu (Identification and Optimization) để tự cặp nhật các thông số trong điều khiển tại từng thời điểm của mô hình nhằm hạn chế biên độ dao động của mô hình. Tiếp theo ta có thể tích hợp thêm các modun khác như GPS, camera trên mô hình để phục vụ được một số nhu cầu thực tế như: giám sát từ xa, hỗ trợ trong cứu hộ, cứu nạn, quay phim trên không, và phục vụ cho nhu cầu giải trí.v.v…

TÀI LIỆU THAM KHẢO

[1] Ngô Kim Long, *Thiết kế và thi công mô hình bay bốn cánh tự cân bằng*, Trường ĐH sư phạm kỹ thuật tp.HCM, 2014.

[2] Phạm Quốc Phương, *Nghiên cứu ứng dụng mô hình Quadrotor trong giám sát và cứu hộ*, Trường ĐH kỹ thuật công nghệ tp.HCM, 2012.

[3] Nguyễn Hải Đăng Tâm, Nguyễn Lê Nhật Thắng, *Nghiên cứu và chế tạo mô hình máy bay Quadrocopter*, Trường ĐH SPKT tp.HCM, 2011.

[4] Dexin Xu, L.W., Guangchun Li, Lidong Guo, *Modeling and Trajectory TrackingControl of a Quad-rotor UAV*. 2012: p. 4.

[5] Mohammed Naseeruddin, A.M.P., *Analysis of the Speedcontrol of BLDC motor on Matlab/Simulink*, in 978-93-81583-72-2. 2012. p. 5

[6] Ying Luo, L.D., Jinlu Han, Haiyang Chao, YangQuan Chen, *VTOL UAV Altitude Flight Control Using Fractional Order Controller,*. 2010: p. pages 6.

[7] Barve, A.S.a.P.A., *Controlling of Quad-rotor UAV Using PID Controller and Fuzzy Logic Controller,* 2012: p. 4.

[8] Vilamoura, A., Portugal, *ViCoMoR 2012 2nd Workshop on Visual Control of Mobile Robots (ViCoMoR) in IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2012.

[9] Michael David Schmidt, *Simulation and control of a Quadrotor unmanned aerial Vehicle*, thesis of University Kentucky, 2011.

[10] Mahony, R., T. Hamel, and J.-M. Pflimlin. *Nonlinear complementary filters on the special orthogonal group*. Volume 53 n-5, pp. 1203–1218. IEEE Transactions on Automatic Control, 2008

[11] Aditya Sreekuar, P. Hithesan, M. Krishna Anand, *Design and implementation of the closed loop control of a Quad rotor UAV for stability*, Bachelor of Technology in Amrita school, 2011.

[12] Severino M. O. Raposo. *System and process of vector propulsion with independent control of three translation and three rotation axis*. Intellectual Property, 2010

[13] http:/www.arduino.cc

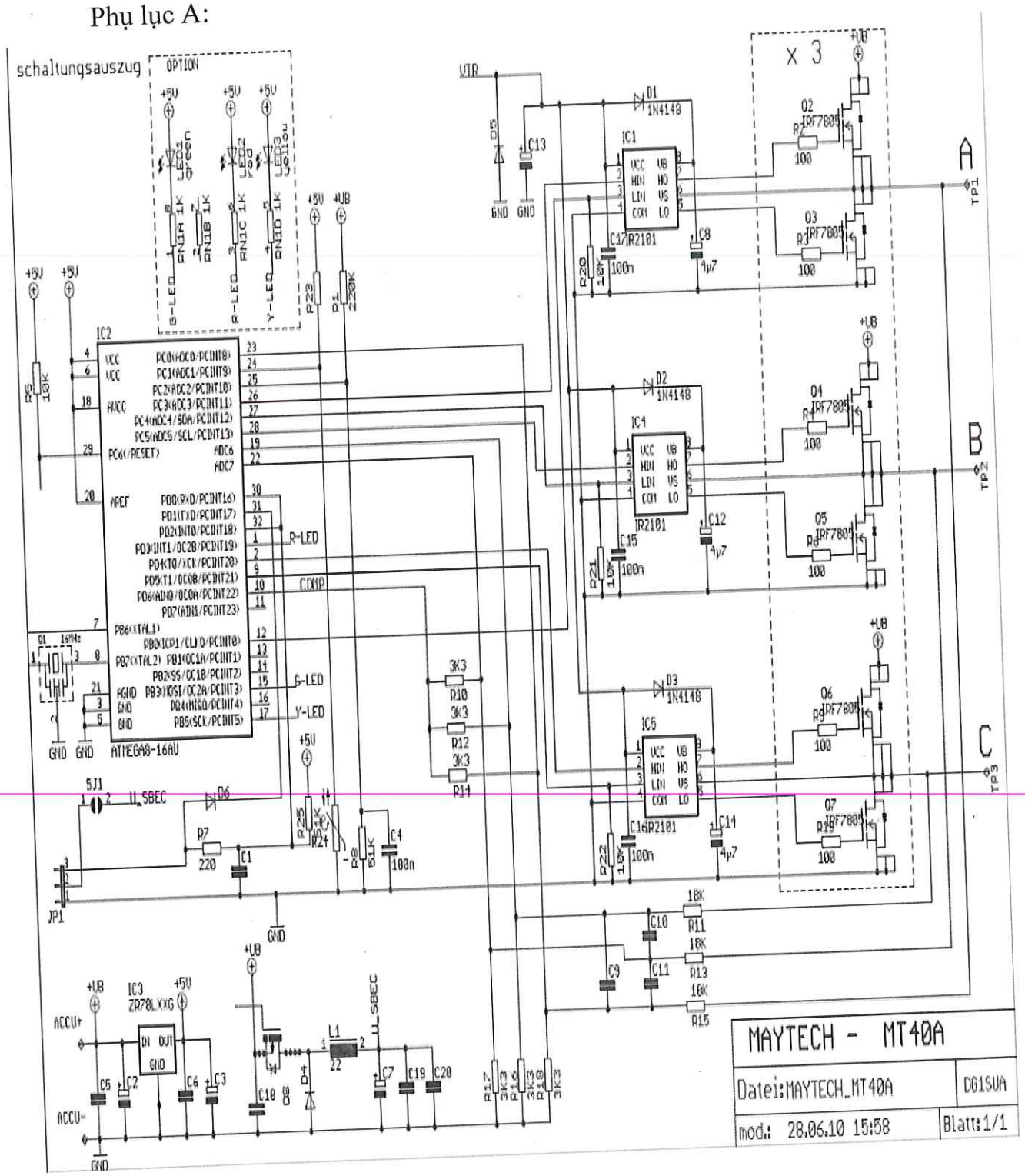
[14] http://www.draganfly.com/

[15] http:/www.icviet.vn

[16] www. matlab.com

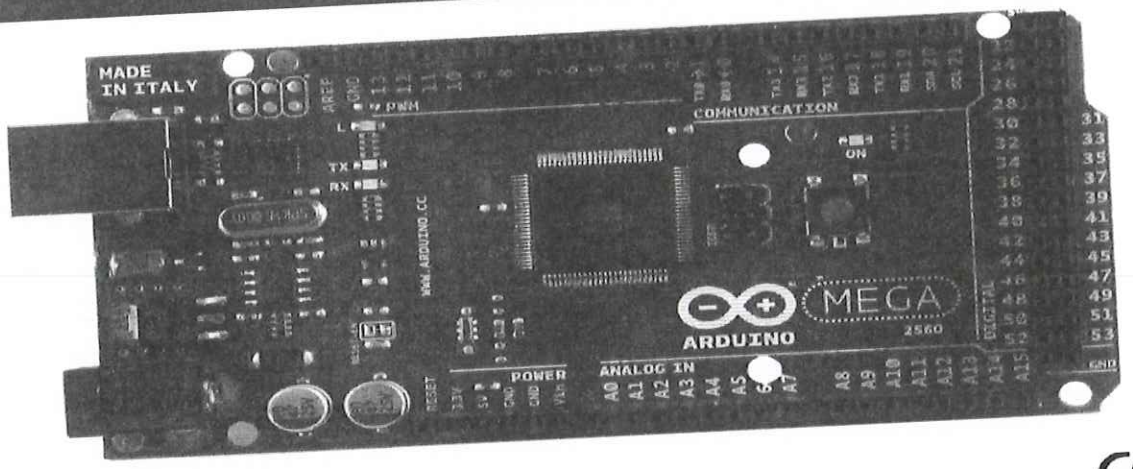
PHỤ LỤC

Phụ lục A:



Phụ lục B

Arduino MEGA 2560



CE

Product Overview

The Arduino Mega 2560 is a microcontroller board based on the ATmega2560 (datasheet). It has 54 digital input/output pins (of which 14 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega is compatible with most shields designed for the Arduino Duemilanove or Diecimila.

Index

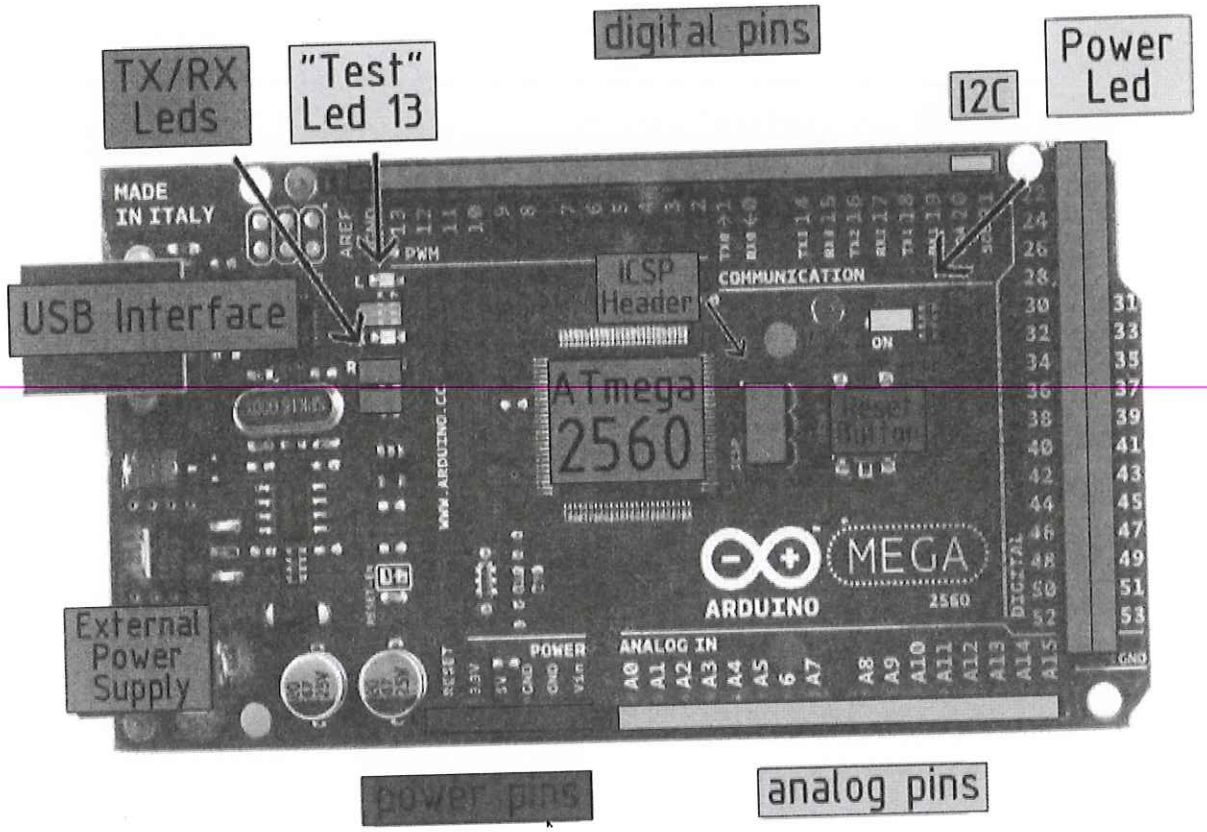
Technical Specification

EAGLE files: arduino-mega2560-reference-design.zip Schematic: arduino-mega2560-schematic.pdf

Summary

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

the board



Power

The Arduino Mega2560 can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.

The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

The power pins are as follows:

- **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.
- **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
- **3V3.** A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
- **GND.** Ground pins.

Memory

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

Input and Output

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.
- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Duemilanove and Diecimila.
- **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- **I²C: 20 (SDA) and 21 (SCL).** Support I²C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I²C pins on the Duemilanove.

The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

- **AREF.** Reference voltage for the analog inputs. Used with analogReference().
- **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

The Arduino Mega2560 has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega2560 provides four hardware UARTs for TTL (5V) serial communication. An ATmega8U2 on the board channels one of these over USB and provides a virtual com port to software on the computer (Windows machines will need a .inf file, but OSX and Linux machines will recognize the board as a COM port automatically. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the ATmega8U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows for serial communication on any of the Mega's digital pins.

The ATmega2560 also supports I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus; see the documentation on the Wiring website for details. To use the SPI communication, please see the ATmega2560 datasheet.

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

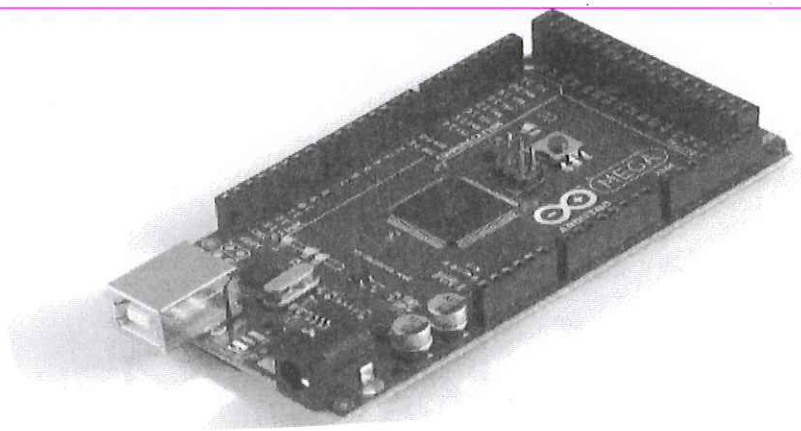
The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

Programming

The Arduino Mega2560 can be programmed with the Arduino software (download). For details, see the reference and tutorials.

The Atmega2560 on the Arduino Mega comes preburned with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.



Automatic (Software) Reset

Rather then requiring a physical press of the reset button before an upload, the Arduino Mega2560 is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2 is connected to the reset line of the ATmega2560 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino software uses this capability to allow you to upload code by simply pressing the upload button in the Arduino environment. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Mega2560 is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Mega2560. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Mega contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

USB Overcurrent Protection

The Arduino Mega has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Physical Characteristics and Shield Compatibility

The maximum length and width of the Mega PCB are 4 and 2.1 inches respectively, with the USB connector and power jack extending beyond the former dimension. Three screw holes allow the board to be attached to a surface or case. Note that the distance between digital pins 7 and 8 is 160 mil (0.16"), not an even multiple of the 100 mil spacing of the other pins.

The Mega is designed to be compatible with most shields designed for the Diecimila or Duemilanove. Digital pins 0 to 13 (and the adjacent AREF and GND pins), analog inputs 0 to 5, the power header, and ICSP header are all in equivalent locations. Further the main UART (serial port) is located on the same pins (0 and 1), as are external interrupts 0 and 1 (pins 2 and 3 respectively). SPI is available through the ICSP header on both the Mega and Duemilanove / Diecimila. **Please note that I^2C is not located on the same pins on the Mega (20 and 21) as the Duemilanove / Diecimila (analog inputs 4 and 5).**

How to use Arduino

Arduino can sense the environment by receiving input from a variety of sensors and can affect i surroundings by controlling lights, motors, and other actuators. The microcontroller on the board programmed using the Arduino programming language (based on Wiring) and the Arduir development environment (based on Processing). Arduino projects can be stand-alone or they ca communicate with software on running on a computer (e.g. Flash, Processing, MaxMSP).

Arduino is a cross-platoform program. You'll have to follow different instructions for your person OS. Check on the Arduino site for the latest instructions. *http://arduino.cc/en/Guide/HomePage*

Linux Install

Windows Install

Mac Install

Once you have downloaded/unzipped the arduino IDE, you can Plug the Arduino to your PC via USB cable

Blink led

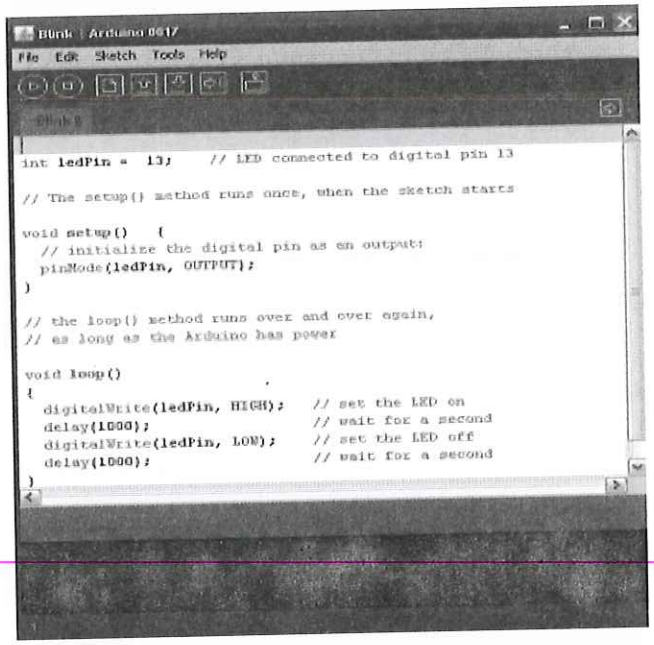
Now you're actually ready to "burn" your first program on the arduino board. To select "blink led", the physical translation of the well known programming "hello world", select

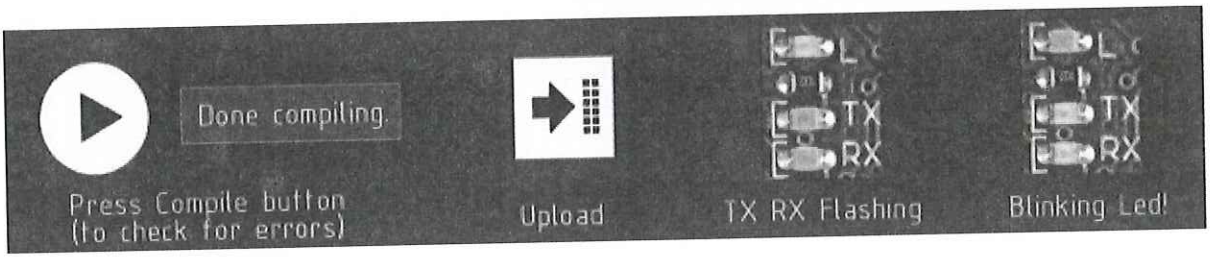
File>Sketchbook>
Arduino-0017>Examples>
Digital>Blink

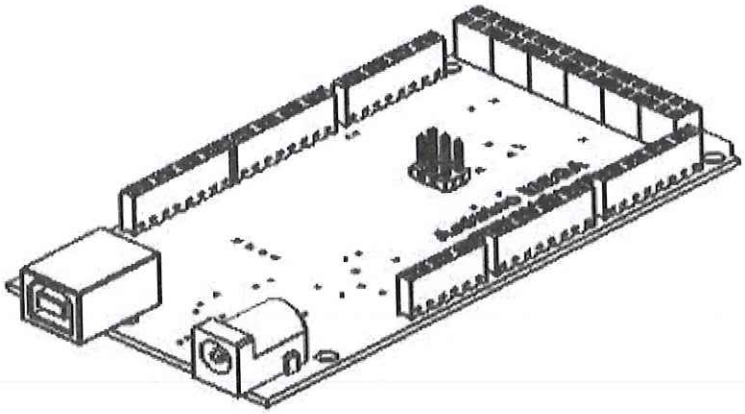
Once you have your skecth you'll see something very close to the screenshot on the right.

In **Tools>Board** select MEGA

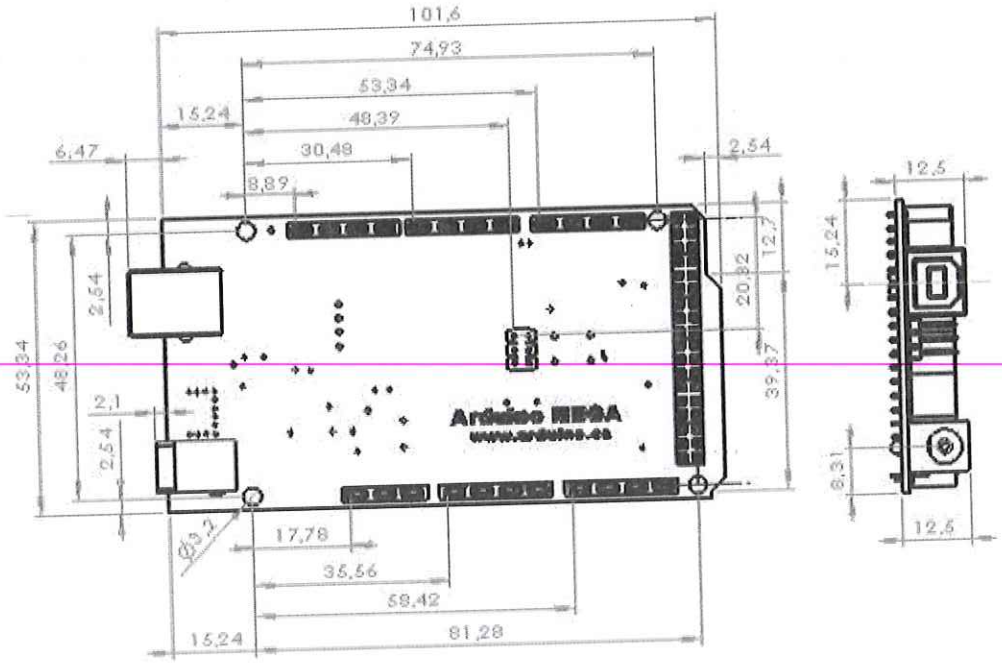
Now you have to go to
Tools>SerialPort
and select the right serial port, the one arduino is attached to.








10,14


101,6
74,93
53,34
15,24
48,39
30,48
6,47
8,89
2,54
12,5
15,24
12,7
20,32
39,37
2,64
53,34
48,26
2,1
2,54
Arduino MEGA
www.arduino.cc
8,31
12,5
Ø3,2
17,78
35,56
50,42
15,24
81,28

Terms & Conditions

1. Warranties

1.1 The producer warrants that its products will conform to the Specifications. This warranty lasts for one (1) years from the date of the sale. The producer shall not be liable for any defects that are caused by neglect, misuse or mistreatment by the Customer, including improper installation or testing, or for any products that have been altered or modified in any way by a Customer. Moreover, The producer shall not be liable for any defects that result from Customer's design, specifications or instructions for such products. Testing and other quality control techniques are used to the extent the producer deems necessary.

1.2 If any products fail to conform to the warranty set forth above, the producer's sole liability shall be to replace such products. The producer's liability shall be limited to products that are determined by the producer not to conform to such warranty. If the producer elects to replace such products, the producer shall have a reasonable time to replacements. Replaced products shall be warranted for a new full warranty period.

1.3 EXCEPT AS SET FORTH ABOVE, PRODUCTS ARE PROVIDED "AS IS" AND "WITH ALL FAULTS." THE PRODUCER DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, REGARDING PRODUCTS, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE

1.4 Customer agrees that prior to using any systems that include the producer products, Customer will test such systems and the functionality of the products as used in such systems. The producer may provide technical, applications or design advice, quality characterization, reliability data or other services. Customer acknowledges and agrees that providing these services shall not expand or otherwise alter the producer's warranties, as set forth above, and no additional obligations or liabilities shall arise from the producer providing such services.

1.5 The Arduino™ products are not authorized for use in safety-critical applications where a failure of the product would reasonably be expected to cause severe personal injury or death. Safety-Critical Applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino™ products are neither designed nor intended for use in military or aerospace applications or environments and for automotive applications or environment. Customer acknowledges and agrees that any such use of Arduino™ products which is solely at the Customer's risk, and that Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

1.6 Customer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products and any use of Arduino™ products in Customer's applications, notwithstanding any applications-related information or support that may be provided by the producer.

2. Indemnification

The Customer acknowledges and agrees to defend, indemnify and hold harmless the producer from and against any and all third-party losses, damages, liabilities and expenses it incurs to the extent directly caused by: (i) an actual breach by a Customer of the representation and warranties made under this terms and conditions or (ii) the gross negligence or willful misconduct by the Customer.

3. Consequential Damages Waiver

In no event the producer shall be liable to the Customer or any third parties for any special, collateral, indirect, punitive, incidental, consequential or exemplary damages in connection with or arising out of the products provided hereunder, regardless of whether the producer has been advised of the possibility of such damages. This section will survive the termination of the warranty period.

4. Changes to specifications

The producer may make changes to specifications and product descriptions at any time, without notice. The Customer must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." The producer reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The product information on the Web Site or Materials is subject to change without notice. Do not finalize a design with this information.

Enviromental Policies

The producer of Arduino™ has joined the Impatto Zero® policy of LifeGate.it. For each Arduino board produced is created / looked after half squared Km of Costa Rica's forest's.

Phụ lục C

```
function [M1, M2, M3, M4] = motor_mixer(Roll, Pitch, Yaw,
Thrust)

idle_PWM = 1000;

M1 = ((Pitch - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
M2 = ((-Roll + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
M3 = ((-Pitch - Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
M4 = ((Roll + Yaw) * Thrust / 2 + Thrust) * 1000 + idle_PWM;
```

Phụ lục D

```
function [forces, moments] = multicopter(DCM, Vb, p, Vr)
%==================================Constants====================
coefficient_drag = 15; % coefficient of drag
prop_diameter = 0.15; % Diameter of the propelle(met)
aircraft_mass = 0.812; % Aircraft mass 0.26(kg)
aircraft_diameter = 0.6; % Motor to motor distance (met)

%============================Actuator Mixer================

MIX = [ 0 1 -1 ; % QUAD [+]
-1 0 1 ;
0 -1 -1 ;
1 0 1 ];
%================================Forces====================
A = [ aircraft_diameter^2 / 10;… % body cross sectional area(X)
aircraft_diameter^2 / 10;… % body cross sectional area (Y)
sum(abs(MIX(:, 3))) * pi/4 * prop_diameter^2 ]; % body cross
sectional area (Z)

Fg = aircraft_mass .* (DCM * [0; 0; 9.81]); % gravitational
force
Fd = -Vb .* coefficient_drag .* A; % drag force
%-----------------------------Thrust Model----------------

Fr = abs(MIX(:, 3)) .* (1*1E-3) .* Vr; % rotor thrust <==
%---------------------------------------------------------
forces = Fg + Fd;
forces(3) = forces(3) - sum(Fr);
%================================Moments====================

mx = Fr .* MIX(:,1) .* (aircraft_diameter / 2)*sind(45); % x
moment
my = Fr .* MIX(:,2) .* (aircraft_diameter / 2)*sind(45); % y
moment
%-----------------------------Torque Model----------------

mz = MIX(:,3) .* (1.7E-5) .* Vr; % rotor torque <==
%---------------------------------------------------------

moments = [sum(mx); sum(my); sum(mz)];
%=========================================================
```

Phụ lục E

```
function varargout = multi_sim(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name', mfilename, ...
'gui_Singleton', gui_Singleton, ...
'gui_OpeningFcn', @multi_sim_OpeningFcn, ...
'gui_OutputFcn', @multi_sim_OutputFcn, ...
'gui_LayoutFcn', [] , ...
'gui_Callback', []);
if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
[varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
gui_mainfcn(gui_State, varargin{:});
end
%============================Initialization==================
function multi_sim_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;
global v f v_r index accelerometer gyroscope plot_length
update_rate camera camera1_scale;
load_system('multi_model');
set_param('multi_model/ch1_input', 'Value', '1500');
set_param('multi_model/ch2_input', 'Value', '1500');
set_param('multi_model/ch3_input', 'Value', '1500');
set_param('multi_model/ch4_input', 'Value', '1500');
camera = 1;
camera1_scale = 0.1;
update_rate = 24;
plot_length = 10;
index = update_rate * plot_length;
gyroscope = zeros(4, index);
accelerometer = zeros(4, index);
axes(handles.model_plot);
axis('off');
axes(handles.gyroscope_plot);
axis('off');
axes(handles.accelerometer_plot);
axis('off');
fid = fopen('model.stl', 'r');
ftitle = fread(fid,80,'uchar=>schar');
numFaces = fread(fid,1,'int32');
```

```
T = fread(fid,inf,'uint8=>uint8');
fclose(fid);
trilist = 1:48;
ind = reshape(repmat(50*(0:(numFaces-
1)),[48,1]),[1,48*numFaces])+repmat(trilist,[1,numFaces]);
Tri = reshape(typecast(T(ind),'single'),[3,4,numFaces]);
v = Tri(:,2:4,:);
v = reshape(v,[3,3*numFaces]);
v = double(v)';
f = reshape(1:3*numFaces,[3,numFaces])';
v_r = v * [1, 0, 0; 0, 1, 0; 0, 0, 1];
handles.timer = timer('ExecutionMode', 'fixedSpacing',
'Period', round(1000 / update_rate) / 1000, 'TimerFcn',
{@update_data, handles});
guidata(hObject, handles);
function varargout = multi_sim_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;
function ch4_pwm_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end
function ch3_pwm_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end
function ch2_pwm_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end
function ch1_pwm_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end
function slider_scale_CreateFcn(hObject, eventdata, handles)
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end
%==================================Buttons====================
function camera_1_Callback(hObject, eventdata, handles)
```

```
global camera;
camera = 1;
set(handles.camera_1, 'Enable', 'off');
set(handles.camera_2, 'Enable', 'on');
set(handles.camera_3, 'Enable', 'on');
%set(handles.start_button, 'Enable', 'off');
function camera_2_Callback(hObject, eventdata, handles)
global camera;
camera = 2;
set(handles.camera_1, 'Enable', 'on');
set(handles.camera_2, 'Enable', 'off');
set(handles.camera_3, 'Enable', 'on');
function camera_3_Callback(hObject, eventdata, handles)
global camera;
camera = 3;
set(handles.camera_1, 'Enable', 'on');
set(handles.camera_2, 'Enable', 'on');
set(handles.camera_3, 'Enable', 'off');
function slider_scale_Callback(hObject, eventdata, handles)
global camera1_scale;
camera1_scale = get(hObject,'Value');
set(handles.scale,'String', round(10*get(hObject,'Value'))/10);
function model_button_Callback(hObject, eventdata, handles)
if get(hObject,'value') ~= 1
axes(handles.model_plot);
cla;
axis('off')
end
function gyroscope_button_Callback(hObject, eventdata, handles)
global update_rate plot_length gyroscope;
if get(hObject,'value') ~= 1
gyroscope = zeros(4, update_rate * plot_length);
axes(handles.gyroscope_plot);
cla;
axis('off');
end
function accelerometer_button_Callback(hObject, eventdata,
handles)
global update_rate plot_length accelerometer;
if get(hObject,'value') ~= 1
accelerometer = zeros(4, update_rate * plot_length);
axes(handles.accelerometer_plot);
cla;
axis('off');
end
```

```
function measurements_button_Callback(hObject, eventdata,
handles)
if get(hObject,'value') ~= 1
set(handles.x_position,'String', '');
set(handles.y_position,'String', '');
set(handles.z_position,'String', '');
set(handles.x_velocity,'String', '');
set(handles.y_velocity,'String', '');
set(handles.z_velocity,'String', '');
set(handles.roll_attitude,'String', '');
set(handles.pitch_attitude,'String', '');
set(handles.yaw_attitude,'String', '');
set(handles.roll_rate,'String', '');
set(handles.pitch_rate,'String', '');
set(handles.yaw_rate,'String', '');
set(handles.out1,'String', '');
set(handles.out2,'String', '');
set(handles.out3,'String', '');
set(handles.out4,'String', '');
end
function start_button_Callback(hObject, eventdata, handles)
global index gyroscope accelerometer;
set(handles.start_button, 'Enable', 'off');
set(handles.stop_button, 'Enable', 'on');
gyroscope = zeros(4, index);
accelerometer = zeros(4, index);
set_param('multi_model','SimulationCommand','Start');
start(handles.timer);
function stop_button_Callback(hObject, eventdata, handles)
set(handles.start_button, 'Enable', 'on');
set(handles.stop_button, 'Enable', 'off');
set_param('multi_model','SimulationCommand','Stop');
stop(handles.timer);
function ch4_center_Callback(hObject, eventdata, handles)
if get(hObject,'value') == 1
set(handles.ch4_center, 'Enable', 'off');
end
set(handles.pwm4,'String', '1500');
set(handles.ch4_pwm,'Value',1500);
set_param('multi_model/ch4_input', 'Value', '1500');
function ch3_center_Callback(hObject, eventdata, handles)
if get(hObject,'value') == 1
set(handles.ch3_center, 'Enable', 'off');
end
set(handles.pwm3,'String', '1500');
```

```
set(handles.ch3_pwm,'Value',1500);
set_param('multi_model/ch3_input', 'Value', '1500');
function ch2_center_Callback(hObject, eventdata, handles)
if get(hObject,'value') == 1
set(handles.ch2_center, 'Enable', 'off');
end
set(handles.pwm2,'String', '1500');
set(handles.ch2_pwm,'Value',1500);
set_param('multi_model/ch2_input', 'Value', '1500');
function ch1_center_Callback(hObject, eventdata, handles)
if get(hObject,'value') == 1
set(handles.ch1_center, 'Enable', 'off');
end
set(handles.pwm1,'String', '1500');
set(handles.ch1_pwm,'Value',1500);
set_param('multi_model/ch1_input', 'Value', '1500');
function ch4_pwm_Callback(hObject, eventdata, handles)
set(handles.pwm4,'String', sprintf('%f',
get(hObject,'value')));
if get(hObject,'value') ~= 1500
set(handles.ch4_center, 'Value', 0);
set(handles.ch4_center, 'Enable', 'on');
elseif get(hObject,'value') == 1500
set(handles.ch4_center, 'Value', 1);
set(handles.ch4_center, 'Enable', 'off');
end
set_param('multi_model/ch4_input', 'Value', sprintf('%f',
get(hObject,'value')));
function ch3_pwm_Callback(hObject, eventdata, handles)
set(handles.pwm3,'String', sprintf('%f',
get(hObject,'value')));
if get(hObject,'value') ~= 1500
set(handles.ch3_center, 'Value', 0);
set(handles.ch3_center, 'Enable', 'on');
elseif get(hObject,'value') == 1500
set(handles.ch3_center, 'Value', 1);
set(handles.ch3_center, 'Enable', 'off');
end
set_param('multi_model/ch3_input', 'Value', sprintf('%f',
get(hObject,'value')));
function ch2_pwm_Callback(hObject, eventdata, handles)
set(handles.pwm2,'String', sprintf('%f',
get(hObject,'value')));
if get(hObject,'value') ~= 1500
set(handles.ch2_center, 'Value', 0);
```

```
set(handles.ch2_center, 'Enable', 'on');
elseif get(hObject,'value') == 1500
set(handles.ch2_center, 'Value', 1);
set(handles.ch2_center, 'Enable', 'off');
end
set_param('multi_model/ch2_input', 'Value', sprintf('%f',
get(hObject,'value')));
function ch1_pwm_Callback(hObject, eventdata, handles)
set(handles.pwm1,'String', sprintf('%f',
get(hObject,'value')));
if get(hObject,'value') ~= 1500
set(handles.ch1_center, 'Value', 0);
set(handles.ch1_center, 'Enable', 'on');
elseif get(hObject,'value') == 1500
set(handles.ch1_center, 'Value', 1);
set(handles.ch1_center, 'Enable', 'off');
end
set_param('multi_model/ch1_input', 'Value', sprintf('%f',
get(hObject,'value')));
%==================================Graphics====================
function update_data(hObject, eventdata, handles)
tic;
global v f v_r index accelerometer gyroscope plot_length camera
camera1_scale;
camera2_scale = camera1_scale * 1;
camera3_scale = camera1_scale * 1;
simulation_data =
get_param('multi_model/dynamic_model/simulation_data',
'RuntimeObject');
simulation_time = simulation_data.InputPort(19).Data;
%-----------------------------Plot Gyroscope-----------------
if get(handles.gyroscope_button, 'Value') == 1
gyroscope = circshift(gyroscope, [4 -1]);
axes(handles.gyroscope_plot);
gyroscope(1, index) = simulation_data.InputPort(16).Data;
gyroscope(2, index) = simulation_data.InputPort(17).Data;
gyroscope(3, index) = simulation_data.InputPort(18).Data;
gyroscope(4, index) = simulation_time;
plot(gyroscope(4,:), gyroscope(1,:), gyroscope(4,:),
gyroscope(2,:), gyroscope(4,:), gyroscope(3,:));
ylim('auto');
if simulation_time < plot_length
xlim([0 plot_length]);
else
xlim([simulation_time - plot_length simulation_time]);
```

```
end
end
%----------------------------Plot Accelerometer----------------
if get(handles.accelerometer_button, 'Value') == 1
accelerometer = circshift(accelerometer, [4 -1]);
axes(handles.accelerometer_plot);
accelerometer(1, index) = simulation_data.InputPort(13).Data;
accelerometer(2, index) = simulation_data.InputPort(14).Data;
accelerometer(3, index) = -simulation_data.InputPort(15).Data;
accelerometer(4, index) = simulation_time;
plot(accelerometer(4,:), accelerometer(1,:),
accelerometer(4,:), accelerometer(2,:), accelerometer(4,:),
accelerometer(3,:));
ylim('auto');
if simulation_time < plot_length
xlim([0 plot_length]);
else
xlim([simulation_time - plot_length simulation_time]);
end
end
%-------------------------------Update Data--------------------

if get(handles.measurements_button, 'Value') == 1
set(handles.x_position,'String', sprintf('%f',
simulation_data.InputPort(4).Data));
set(handles.y_position,'String', sprintf('%f',
simulation_data.InputPort(5).Data));
set(handles.z_position,'String', sprintf('%f', -
simulation_data.InputPort(6).Data));
set(handles.x_velocity,'String', sprintf('%f',
simulation_data.InputPort(1).Data));
set(handles.y_velocity,'String', sprintf('%f',
simulation_data.InputPort(2).Data));
set(handles.z_velocity,'String', sprintf('%f', -
simulation_data.InputPort(3).Data));
set(handles.roll_attitude,'String', sprintf('%f',
simulation_data.InputPort(7).Data));
set(handles.pitch_attitude,'String', sprintf('%f',
simulation_data.InputPort(8).Data));
set(handles.yaw_attitude,'String', sprintf('%f',
simulation_data.InputPort(9).Data));
set(handles.roll_rate,'String', sprintf('%f',
simulation_data.InputPort(10).Data));
set(handles.pitch_rate,'String', sprintf('%f',
simulation_data.InputPort(11).Data));
```

```
set(handles.yaw_rate,'String', sprintf('%f',
simulation_data.InputPort(12).Data));
control_data =
get_param('multi_model/dynamic_model/control_data',
'RuntimeObject');
set(handles.out1,'String', sprintf('%f',
control_data.InputPort(1).Data(1)));
set(handles.out2,'String', sprintf('%f',
control_data.InputPort(2).Data(1)));
set(handles.out3,'String', sprintf('%f',
control_data.InputPort(3).Data(1)));
set(handles.out4,'String', sprintf('%f',
control_data.InputPort(4).Data(1)));
end
%--------------------------------Update Model------------------

if get(handles.model_button, 'Value') == 1
roll_attitude = -simulation_data.InputPort(7).Data;
pitch_attitude = -simulation_data.InputPort(8).Data;
yaw_attitude = simulation_data.InputPort(9).Data;
roll_matrix = [1 0 0; 0 cosd(pitch_attitude) -
sind(pitch_attitude); 0 sind(pitch_attitude)
cosd(pitch_attitude)];
pitch_matrix = [cosd(roll_attitude) 0 sind(roll_attitude); 0 1
0; -sind(roll_attitude) 0 cosd(roll_attitude)];
yaw_matrix = [cosd(yaw_attitude) -sind(yaw_attitude) 0;
sind(yaw_attitude) cosd(yaw_attitude) 0; 0 0 1];
v = v_r * pitch_matrix * roll_matrix * yaw_matrix;
axes(handles.model_plot);
cla(handles.model_plot);
patch('Faces',f,'Vertices',v, 'FaceColor', [0.5 0.5 0.5],
'EdgeColor', 'none', 'FaceLighting', 'gouraud',
'AmbientStrength', 0.01);
camlight('headlight');
material('dull');
axis('off');
if camera == 1
xlim([-camera1_scale camera1_scale]);
ylim([-camera1_scale camera1_scale]);
zlim([-camera1_scale camera1_scale]);
camproj('perspective');
view([0 15]);
elseif camera == 2
y_focus = -simulation_data.InputPort(4).Data;
x_focus = -simulation_data.InputPort(5).Data;
```

```
z_focus = simulation_data.InputPort(6).Data;
ylim([y_focus-camera2_scale y_focus+camera2_scale]);
xlim([x_focus-camera2_scale x_focus+camera2_scale]);
zlim([z_focus-camera2_scale z_focus+camera2_scale]);
camproj('perspective');
view([0, -1, 0.5]);
elseif camera == 3
y_focus = -simulation_data.InputPort(4).Data;
x_focus = -simulation_data.InputPort(5).Data;
z_focus = simulation_data.InputPort(6).Data;
ylim([y_focus-camera3_scale y_focus+camera3_scale]);
xlim([x_focus-camera3_scale x_focus+camera3_scale]);
zlim([z_focus-camera3_scale z_focus+camera3_scale]);
camproj('perspective');
view([0, 0, camera3_scale]);
end
end
drawnow;
cycles = 1 / toc;
if camera == 1
set(handles.update_rate,'String', ['1 - Fixed ['
num2str(round(cycles)) ' fps]']);
elseif camera == 2
set(handles.update_rate,'String', ['2 - Rear ['
num2str(round(cycles)) ' fps]']);
elseif camera == 3
set(handles.update_rate,'String', ['3 - Top ['
num2str(round(cycles)) ' fps]']);
end
%==========================================================
```

Phụ lục F

Code cho board arduino

```
void setup()
{
// port directions: 1=input, 0=output
pinMode(13,OUTPUT);
pinMode(2,INPUT);
pinMode(3,INPUT);
pinMode(4,INPUT);
pinMode(5,INPUT);
pinMode(6,INPUT);
}
void pwmgen(float x)
{
digitalWrite(13,HIGH);
delay(x);
digitalWrite(13,HIGH);
delay(20-x);
}
void loop()
{
RC1=0;
RC2=1;
RC3=0;
int inmain=digitalRead(2);
int in1=digitalRead(3);
int in2=digitalRead(4);
int in3=digitalRead(5);
int in4=digitalRead(6);
digitalWrite(13,LOW);
if(inmain==1)
{
if(in1==0)
{
if(in2==0)
{
if(in3==0)
{
if(in4==0)
{
pwmgen(0.92);
```

```
}
else
{
pwmgen(0.9825);
}}
else
{
if(in4==0)
{
pwmgen(1.045);
}
else
{
pwmgen(1.1075);
}}}
else
{
if(in3==0)
{
if(in4==0)
{
pwmgen(1.17);
}
else
{
pwmgen(1.2325);
}}
else
{
66
if(in4==0)
{
pwmgen(1.295);
}
else
{
pwmgen(1.3575);
}
}
}
}
```

```
else
{
if(in2==0)
{
if(in3==0)
{
if(in4==0)
{
pwmgen(1.42);
}
else
{
pwmgen(1.4825);
}
}
else
{
if(in4==0)
{
pwmgen(1.545);
}
else
{
pwmgen(1.6075);
}
}
}
else
{
if(in3==0)
{
if(in4==0)
{
pwmgen(1.67);
}
else
{
pwmgen(1.7325);
}
}
else
```

```
{
if(in4==0)
{
pwmgen(1.795);
}
else
{
pwmgen(1.92);
}
}
}

//////////////////////////Tạo thời gian delay //////////////////////
#include<iostream.h>
#include<math.h>
#include<conio.h>
main()
{
//bien ban dau
intbits,count=0,ton[100];
float samples=1,i,res,j,toff;
unsigned long inttoffc;
clrscr();
cout<<"Enter the no of bits involved : ";
cin>>bits;
for(i=1;i<=bits;i++)
{
samples=2*samples;
}
cout<<"\n\nThe no of samples are "<<samples<<"\n";
samples-=1;
getch();
res=1/samples;
cout<<"\n\nThe resolution is "<<res<<"\n\n";
getch();
cout<<"TABULATING THE RESULTS OF THE CALCULATION FOR THE VALUE OF CYCLES
REQUIRED FOR DESIRED PWM - TON AND TOFF \n\nS.no\tTON\t\tTON
CYCLES\tTOFF\t\tTOFF
CYCLES\n";
i=1.000;
for(j=0;j<=samples;j++)
```

```
{
ton[j]=i*5000;
toffc=100000-ton[j];
toff=20-i;
cout<<j+1<<"\t"<<i<<"\t"<<ton[j]<<"\t\t"<<toff<<"\t"<<toffc<<"\t\n";
i+=res;
}
getch();
}

////////////////////////////// Hàm tạo PWM ////////////////
function PWM=PWM (u)
b=1;
d=1
a1=1;a2=1;a3=1;a4=1;
b1=1;b2=1;b3=1;b4=1;
c=[1 1 1 1 ; 0 0 1 -1;1 -1 0 0 ; d/b d/b -d/b -d/b]
A=[a1;a2;a3;a4]
B=[b1 0 0 0 ; 0 b2 0 0 ;0 0 b3 0; 0 0 0 b4]
PWM=(B^-1)*((c^-1)*u-A)
```

Phụ lục 4: Hàm động lực của motor

```
function F=FORCE (PWM)
b=1;
d=1
a1=1;a2=1;a3=1;a4=1;
b1=1;b2=1;b3=1;b4=1;
c=[1 1 1 1 ; 0 0 1 -1;1 -1 0 0 ; d/b d/b -d/b -d/b]
A=[a1;a2;a3;a4]
B=[b1 0 0 0 ; 0 b2 0 0 ;0 0 b3 0; 0 0 0 b4]
F=A+B*PWM
-------------------------------------------------
function phi(phi)
global cp
global sp
cp=cos(phi);
sp=sin(phi);
-------------------------------------------------
global ct
global st
ct=cos(theta);
st=sin(theta);
```

```
---------------------------------------------------
function si(si)
global cs
global ss
cs=cos(si);
ss=sin(si);
function angle (phi,theta,si)
cp=cos(phi);
ct=cos(theta);
cs=cos(si);
sp=sin(phi);
st=sin(theta);
% khoi tao gia tri ban dau
Ixx=0.017247
Iyy=0.017674
Izz=0.030082
b=1.5500e-006
d=b/10
g=9.81;
L=.3;
m=0.9;
Kiroll=-0.0070
Kdroll=0.0987
Kproll=0.0763
Kipitch=-0.0070
Kdpitch=0.0987
Kppitch=0.0763
Kiyaw=-0.0070
Kdyaw=0.0987
Kpyaw=0.0763
Kialtitude=-0.0099
Kdaltitude=-0.0831
Kpaltitude=-0.0195
Kix=-0.0099
Kdx=-0.0831
Kpx=-0.0195
Kiy=-0.0099
Kdy=-0.0831
Kpy=-0.0195
b=1;
d=1
```

```
a1=1;a2=1;a3=1;a4=1;
b1=1;b2=1;b3=1;b4=1;
c=[1 1 1 1 ; 0 0 1 -1;1 -1 0 0 ; d/b d/b -d/b -d/b]
A=[a1;a2;a3;a4]
B=[b1 0 0 0 ; 0 b2 0 0 ;0 0 b3 0; 0 0 0 b4]
global cs
global ss
global ct
global st
global cp
global sp
cs=0
ss=0
ct=0
st=0
cp=0
sp=0
////////////////////////// Điều khiển Quadrotor//////////////////
#include <inttypes.h>
#include <math.h>
#include <Wire.h> // For magnetometer readings
// QuadCopter PID GAINS
#define KP_QUAD_ROLL 1.8 // 2.2 //1.75
#define KD_QUAD_ROLL 0.42 // 0.54 //0.45
#define KI_QUAD_ROLL 0.5 // 0.45 //0.5
#define KP_QUAD_PITCH 1.8 // 2.2 //1.75
#define KD_QUAD_PITCH 0.42 // 0.54 //0.45
#define KI_QUAD_PITCH 0.5 // 0.45 //0.5
#define KP_QUAD_YAW 3.8 // 4.6 //3.2 //2.6
#define KD_QUAD_YAW 1.3 // 0.7 //0.8 //0.4
#define KI_QUAD_YAW 0.15 // 0.2 //0.15
#define KD_QUAD_COMMAND_PART 18.0 // for special KD implementation (in two
parts)
#define acc_offset_x 508
#define acc_offset_y 504
#define acc_offset_z 501
#define gyro_offset_roll 370
#define gyro_offset_pitch 373
#define gyro_offset_yaw 380
#define MAX_CHANNELS 4 // Bo RC 4 kenh
#define MIN_THROTTLE 1037
```

```
#define CHANN_CENTER 1500
#define SPEKTRUM 1
#define GRAVITY 101 //this equivalent to 1G in the raw data coming from the accelerometer
#define Accel_Scale(x) x*(GRAVITY/9.81)
#define ToRad(x) (x*0.01745329252) // *pi/180
#define ToDeg(x) (x*57.2957795131) // *180/pi
#define Gyro_Gain_X 0.92 //X axis Gyro gain
#define Gyro_Gain_Y 0.92 //Y axis Gyro gain
#define Gyro_Gain_Z 0.94 //Z axis Gyro gain
#define Gyro_Scaled_X(x) x*ToRad(Gyro_Gain_X)
#define Gyro_Scaled_Y(x) x*ToRad(Gyro_Gain_Y)
#define Gyro_Scaled_Z(x) x*ToRad(Gyro_Gain_Z)
#define Kp_ROLLPITCH 0.0125
#define Ki_ROLLPITCH 0.000010
#define Kp_YAW 1.2
#define Ki_YAW 0.00005
#define SPEEDFILT 3
#define OUTPUTMODE 1
//Sensor: GYROX, GYROY, GYROZ, ACCELX, ACCELY, ACCELZ
int SENSOR_SIGN[]={1,-1,-1,1,-1,1,-1,-1,-1}; //{1,-1,-1,-1,1,-1}
float AN[6]; //array that store the 6 ADC filtered data
float AN_OFFSET[6]; //Array luu tru gia tri gyro
float G_Dt=0.02;
float Accel_Vector[3]= {0,0,0}; //luu vector gia toc
float Accel_Vector_unfiltered[3]= {0,0,0};
float Accel_magnitude;
float Accel_weight;
float Gyro_Vector[3]= {0,0,0};
float Omega_P[3]= {0,0,0};
float Omega_I[3]= {0,0,0}
float Omega[3]= {0,0,0};
float errorRollPitch[3]= {0,0,0};
float errorYaw[3]= {0,0,0};
float errorCourse=0;
float COGX=0;
float COGY=1;
float roll=0;
float pitch=0;
float yaw=0;
----------
```

```
void Attitude_control(){
//dieu khien goc roll
err_roll_ant = err_roll;
if (AP_mode==2)
err_roll = comando_rx_roll - ToDeg(roll);
else
err_roll = 0;
err_roll = constrain(err_roll,-30,30);
roll_I += err_roll*G_Dt;
roll_I = constrain(roll_I,-50,50);
roll_D = comando_rx_roll_diff*KD_QUAD_COMMAND_PART - ToDeg(Omega[0]);
control_roll = KP_QUAD_ROLL*err_roll + KD_QUAD_ROLL*roll_D +
KI_QUAD_ROLL*roll_I;
//dieu khiên goc pitch
err_pitch_ant = err_pitch;
if (AP_mode==2)
88
err_pitch = comando_rx_pitch - ToDeg(pitch);
else
err_pitch = 0;
err_pitch = constrain(err_pitch,-30,30);
pitch_I += err_pitch*G_Dt;
pitch_I = constrain(pitch_I,-50,50);
control_pitch = KP_QUAD_PITCH*err_pitch + KD_QUAD_PITCH*pitch_D +
KI_QUAD_PITCH*pitch_I;
// dieu khien goc yaw
err_yaw_ant = err_yaw;
if (AP_mode==2){
err_yaw = comando_rx_yaw - ToDeg(yaw);
if (err_yaw > 180)
err_yaw -= 360;
else if(err_yaw < -180)
err_yaw += 360;
}
else
err_yaw = 0;
err_yaw = constrain(err_yaw,-60,60);
yaw_I += err_yaw*G_Dt;
yaw_I = constrain(yaw_I,-50,50);
yaw_D = comando_rx_yaw_diff*KD_QUAD_COMMAND_PART - ToDeg(Omega[2]);
// Dieu khien PID
```

```
control_yaw = KP_QUAD_YAW*err_yaw + KD_QUAD_YAW*yaw_D +
KI_QUAD_YAW*yaw_I;
}
int channel_filter(int ch, int ch_old)
{
int diff_ch_old;
int result;
result = ch;
diff_ch_old = ch - ch_old;
if (diff_ch_old<0)
{
if (diff_ch_old<-30)
result = ch_old-30;
}
else
{
if ((diff_ch_old>30)&&(ch_old>900))
result = ch_old+30;
}
return((ch+ch_old)/2);
}
90
long timer=0;
long timer_old;
void setup(){
int i;
int aux;
Serial.begin(38400);
pinMode(2,OUTPUT);
digitalWrite(2,HIGH);
pinMode(8,INPUT);
pinMode(9,OUTPUT); // motor 1
pinMode(10,OUTPUT); // Motor 2
pinMode(11,OUTPUT); // Motor 3
pinMode(12,OUTPUT); // Motor 4
pinMode(13,OUTPUT);
ch1=MIN_THROTTLE;
ch2=MIN_THROTTLE;
ch3=MIN_THROTTLE;
ch4=MIN_THROTTLE;
delay(100);
```

```
comando_rx_yaw = 0;
Servo1 = 1500;
91
Servo2 = 1500;
Serial.println();
RxServoInput_ini();
delay(1000);
// lay gia tri dieu khien
Neutro[1] = RxGetChannelPulseWidth(1);
Neutro[2] = RxGetChannelPulseWidth(2);
Neutro[3] = RxGetChannelPulseWidth(3);
Neutro[4] = RxGetChannelPulseWidth(4);
Neutro[5] = RxGetChannelPulseWidth(5);
Neutro[6] = RxGetChannelPulseWidth(6);
for (i=0; i<80; i++)
{
Neutro[1] = (Neutro[1]*0.8 + RxGetChannelPulseWidth(1)*0.2);
Neutro[2] = (Neutro[2]*0.8 + RxGetChannelPulseWidth(2)*0.2);
Neutro[3] = (Neutro[3]*0.8 + RxGetChannelPulseWidth(3)*0.2);
Neutro[4] = (Neutro[4]*0.8 + RxGetChannelPulseWidth(4)*0.2);
Neutro[5] = (Neutro[5]*0.8 + RxGetChannelPulseWidth(5)*0.2);
Neutro[6] = (Neutro[6]*0.8 + RxGetChannelPulseWidth(6)*0.2);
delay(25);
}
Serial.print("Rx values: ");
Serial.print(Neutro[1]);
Serial.print(",");
Serial.print(Neutro[2]);
Serial.print(",");
Serial.print(Neutro[3]);
Serial.print(",");
Serial.println(Neutro[4]);
Serial.print(",");
Serial.println(Neutro[5]);
Serial.print(",");
Serial.println(Neutro[6]);
#if SPEKTRUM==1
Neutro[3] = CHANN_CENTER;
Neutro[2] = CHANN_CENTER;
Neutro[1] = MIN_THROTTLE;
#else
```

```
Neutro[1] = CHANN_CENTER;
Neutro[2] = CHANN_CENTER;
Neutro[3] = MIN_THROTTLE;
#endif
num_servos = 4;
Servos[0].pin = 10; // motor 1
Servos[1].pin = 11; // motor 3
Servos[2].pin = 12; // motor 2
Servos[3].pin = 13; // motor 4
Servo_Timer2_set(0,MIN_THROTTLE);
Servo_Timer2_set(1,MIN_THROTTLE);
Servo_Timer2_set(2,MIN_THROTTLE);
Servo_Timer2_set(3,MIN_THROTTLE);
Servo_Timer2_ini(); // Servo Interrupt initialization
Analog_Reference(EXTERNAL);
Analog_Init();
I2C_Init();
delay(100);
```