BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ HẢI PHÒNG

ĐỒ ÁN TỐT NGHIỆP
NGÀNH ĐIỆN TỬ - TRUYỀN THÔNG

Sinh viên : Nguyễn Vũ Hoàng Anh

Giảng viên hướng dẫn:TS. Đoàn Hữu Chức

Hải Phòng -2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ HẢI PHÒNG

ĐỀ TÀI : THIẾT KẾ CHẾ TẠO THIẾT BỊ BAY KHÔNG NGƯỜI LÁI DRONE

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
NGÀNH ĐIỆN TỬ - TRUYỀN THÔNG

Sinh viên thực hiện : Nguyễn Vũ Hoàng Anh
Giảng viên hướng dẫn: TS. Đoàn Hữu Chức

Hải Phòng - 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ HẢI PHÒNG

NHIỆM VỤ ĐỀ TÀI TỐT NGHIỆP

Sinh viên : Nguyễn Vũ Hoàng Anh **- MSV** : 2213103002

Lớp : DTL 2601

Ngành : Điện tử - Truyền thông

Tên đề tài : Thiết kế chế tạo thiết bị bay không người lái Drone

NHIỆM VỤ ĐỀ TÀI

1.Nội dung và các yêu cầu cần giải quyết trong nhiệm vụ đề tài tốt nghiệp (về lý luận, thực tiễn, các số liệu cần tính toán và các bản vẽ).

……..

……...……

……

……

……

…….....

.....……

2. Các số liệu cần thiết để tính toán.

……

……

……

……

……

……

……

……

3.Địa điểm thực tập tốt nghiệp.

……

……

…….....

……...

CÁC CÁN BỘ HƯỚNG DẪN ĐỀ TÀI TỐT NGHIỆP

Họ và tên : Đoàn Hữu Chức

Học hàm, học vị : Tiến sĩ

Cơ quan công tác : Trường Đại học Thủy lợi

Nội dung hướng dẫn:

…….....

…….....

…….....

…….....

Đề tài tốt nghiệp được giao ngày ... tháng ... năm 2024

Yêu cầu phải hoàn thành xong trước ngày ... tháng ... năm 2024

Đã nhận nhiệm vụ ĐTTN	Đã giao nhiệm vụ ĐTTN
Sinh viên	*Giảng viên hướng dẫn*
Nguyễn Vũ Hoàng Anh	Đoàn Hữu Chức

Hải Phòng, ngày tháng năm 2024

TRƯỞNG KHOA

Cộng Hòa Xã Hội Chủ Nghĩa Việt Nam
Độc lập - Tự do - Hạnh phúc

PHIẾU NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN TỐT NGHIỆP

Họ và tên giảng viên :Đoàn Hữu Chức

Đơn vị công tác :Trường Đại học Thủy lợi

Họ và tên sinh viên :Nguyễn Vũ Hoàng Anh

Chuyên ngành : Điện tử - Truyền thông

Nội dung hướng dẫn : Toàn bộ đề tài

1. Tinh thần thái độ của sinh viên trong quá trình làm đề tài tốt nghiệp

.....

.....

.....

.....

2. Đánh giá chất lượng của đồ án/khóa luận (so với nội dung yêu cầu đã đề ra trong nhiệm vụ Đ.T.T.N, trên các mặt lý luận, thực tiễn, tính toán số liệu...)

.....

.....

.....

3. Ý kiến của giảng viên hướng dẫn tốt nghiệp

Được bảo vệ ☐ Không được bảo vệ ☐ Điểm hướng dẫn ☐

Hải Phòng, ngày.....tháng.....năm 2024

Giảng viên hướng dẫn

(ký và ghi rõ họ tên)

Cộng hòa xã hội chủ nghĩa Việt Nam
Độc lập - Tự do - Hạnh phúc

PHIẾU NHẬN XÉT CỦA GIẢNG VIÊN CHẤM PHẢN BIỆN

Họ và tên giảng viên ……

Đơn vị công tác:.....

Họ và tên sinh viên: .....**Chuyên ngành:**.....

Đề tài tốt nghiệp: .....

.....

1. Phần nhận xét của giảng viên chấm phản biện

.....

.....

.....

.....

2. Những mặt còn hạn chế

.....

.....

.....

.....

3. Ý kiến của giảng viên chấm phản biện

Được bảo vệ ☐ Không được bảo vệ ☐ Điểm phản biện ☐

Hải Phòng, ngày.....tháng.....năm 2024

Giảng viên chấm phản biện

(ký và ghi rõ họ tên)

MỤC LỤC

LỜI MỞ ĐẦU

Thiết bị bay không người lái ngày càng được sử dụng rộng rãi trong nhiều lĩnh vực của cuộc sống. Hơn thế nữa, việc sử dụng thiết bị drone trong quân sự ngày càng trở nên phổ biến và hiệu quả. Với mục đích nghiên cứu tìm hiểu, thực hiện các thiết kế với một thiết bị cụ thể em đã chọn đề tài "*Thiết kế chế tạo thiết bị bay không người lái drone"* làm đề đề tốt nghiệp của mình.

Sau thời gian làm đồ án, em đã thực hiện xong các nội dung chính gồm:

- Nghiên cứu tổng quan về thiết bị drone;
- Thiết kế các mạch điều khiển;
- Lắp đặt phần cơ khí của drone.

Do thời gian có hạn và kiến thức còn hạn chế nên đồ án còn nhiều vấn đề cần tiếp tục phải giải quyết vì vậy rất mong các thầy và các bạn góp ý để đồ án của em hoàn thiện hơn.

Qua đây em xin trân trọng gửi lời cảm ơn tới các thầy trong Khoa Điện – Điện tử, Trường Đại học Quản lý và Công nghệ Hải phòng, đặc biệt là TS Đoàn Hữu Chức đã giảng dạy và hướng dẫn em suốt những thời gian em học tập tại trường.

Sinh viên thực hiện

Nguyễn Vũ Hoàng Anh

CHƯƠNG 1. TỔNG QUAN VỀ DRONE

1.1. Khái niệm về Drone

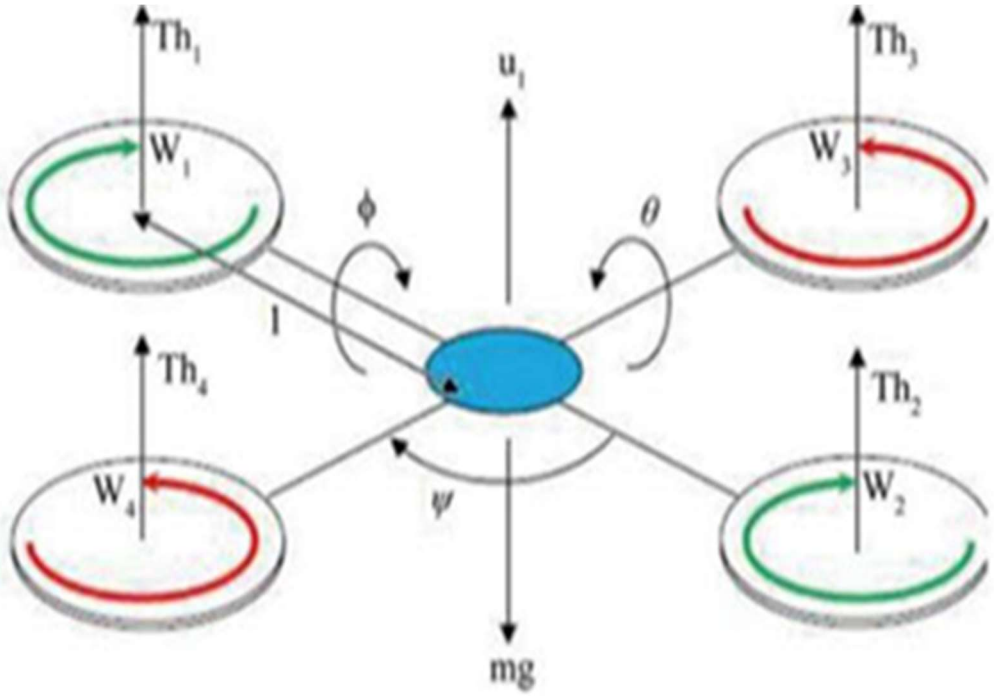
Drone là tên gọi chung chỉ các thiết bị bay được điều khiển từ xa hoặc tự động hóa. Các **thiết bị bay không người lái** (flycam, UAV…) này không có người lái hay điều khiển trực tiếp bên trong mà thường được vận hành nhờ hệ thống, phần mềm được lập trình tự động hoặc có người vận hành, điều khiển từ xa, ở khu vực an toàn hơn.

Trong những năm gần đây, drone giành được nhiều sự quan tâm đặc biệt, điển hình là quadcopter. Quadcopter là một mô hình bay hay thiết bị bay không người lái, được xếp vào loại máy bay có khả năng cất cánh và hạ cánh thẳng đứng. Quadcopter có khả năng di chuyển theo 6 hướng cơ bản (lên-xuống, trước-sau, trái-phải) và có thể di chuyển đa hướng rất linh hoạt.

Hình 1.1: Một số loại Drone hiện có

1.2. Cách thức hoạt động của Drone (Quadcopter)

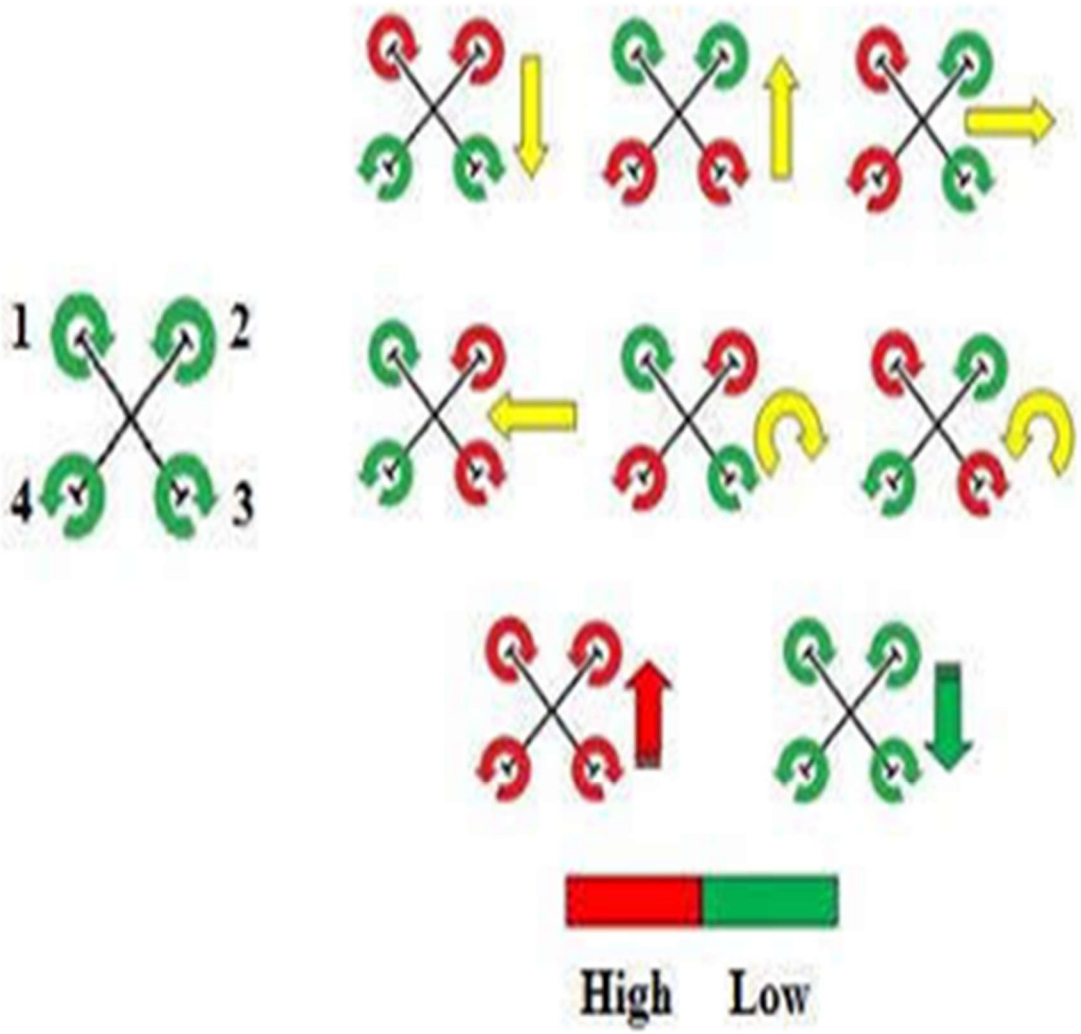
Máy bay Quadcopter có dạng chữ thập hay dấu cộng với bốn động cơ đặt ở bốn góc. Hai trục chéo của máy bay được đặt theo hai trục X, Y của hệ trục tọa độ Descartes. Mỗi động cơ kết hợp với cánh quạt trong Quadcopter sẽ tạo ra một lực đẩy và moment xoắn nhất định, bốn cánh quạt được chia thành hai nhóm có chiều quay ngược nhau, hai cánh đối diện nhau quay cùng chiều. Kết quả là moment xoắn bị triệt tiêu nếu 4 cánh quạt đều có cùng một vận tốc góc, do đó có thể làm cho máy bay không bị xoay tròn khi bay. Để cân bằng mô hình thì các động cơ phải được điều khiển sao cho mô hình có góc lệch so với trục chuẩn trong phạm vi cho phép. Các hệ trục tọa độ của Quadcopter thể hiện trong hình sau:



Hình 1.2: Mô phỏng Drone trong hệ trục tọa độ Descartes

Để điều khiển mô hình bay Quadcopter, có hai phương pháp đó là điều khiển theo kiểu chữ X và điều khiển theo kiểu chữ thập (hay còn gọi là điều khiển theo kiểu dấu cộng). Phương pháp điều khiển theo dạng chữ X được sử dụng trong điều khiển hướng bay của mô hình, bởi vì việc thay đổi hướng được thực hiện bởi hai động cơ, vì vậy khả năng đáp ứng và tính linh hoạt cao hơn so với việc thay đổi tốc độ bằng một động cơ theo kiểu dấu cộng.

Việc thay đổi tốc độ quay của các động cơ vừa làm cho mô hình cân bằng, vừa dùng để điều khiển mô hình di chuyển. Hướng di chuyển được minh họa ở hình sau:



Hình 1.3: Mô tả hướng di chuyển của Drone

Hướng của máy bay khi di chuyển được điều khiển bởi hai động cơ, tùy theo hướng di chuyển mà các động cơ này thay đổi tốc độ để tạo ra góc nghiêng so với trục cân bằng. Như hình 2.4, hướng di chuyển của máy bay được thể hiện qua sự thay đổi tốc độ của các động cơ. Ví dụ: Để máy bay di chuyển hướng tới thì động cơ 1,2 sẽ giữ nguyên hoặc giảm tốc độ còn cặp động cơ 3,4 sẽ quay nhanh hơn. Tương tự với các hướng di chuyển khác thì việc thay đổi tốc độ quay tương ứng sẽ giúp máy bay di chuyển cũng như xoay tròn hoặc thay đổi độ cao.

1.3. Cấu tạo cơ bản của Drone (Quadcopter)

Hình 1.4: Cấu tạo cơ bản của Drone

- Tay Cầm Điều Khiển (TX)
- Bộ Thu nhận Sóng Điều Khiển (RX)
- Bộ Khung Kit (cover)
- Cánh Quạt (Fans)
- Động Cơ (Motor)
- Bộ Điều Tốc (ESC)
- Mạch điều khiển (FC)

1.4. Phương thức điều khiển vô tuyến sử dụng Module NRF24L01

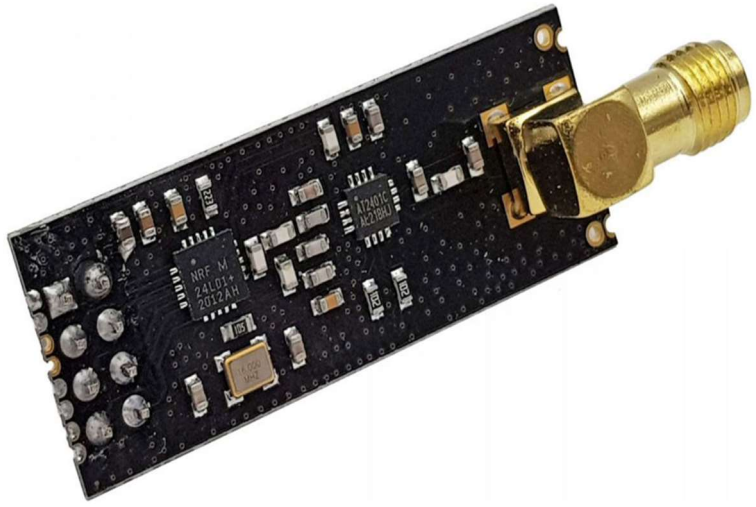
Bộ điều khiển drone thường sử dụng sóng radio tần số 2,4GHz, hình thức không khác mấy so với những bộ điều khiển từ xa của máy bay mô hình truyền thống, gồm hai nút bấm và ăng ten có thể gấp gọn. Một số bộ điều khiển có sự kết hợp cả tín hiệu 2,4GHz và Wi-Fi, trông giống tay cầm điều khiển máy chơi game hoặc chúng có thể dựa trên ứng dụng điều khiển chạy trên smartphone hay máy tính bảng. Tuy nhiên trong quá trình học tập và nghiên cứu, em quyết định sử dụng Module NRF24L01 làm phương thức chính trong điều khiển vô tuyến Drone.

1.4.1. Giới thiệu tổng quan về Module truyền thông NRF24L01

Là module thu phát có con chip đơn 2.4GHz với giao thức nhúng baseband, ứng dụng cho truyền nhận không dây với năng lượng cực thấp .

NRF24L01 được thiết kế để hoạt động theo chuẩn ISM với tần số 2.400-2.4835Ghz.

Module được kết nối với các thiết bị ngoại vi thụ động hay cần một vi điều khiển để có thể giao tiếp với nhau thông qua giao thức SPI và khoảng cách tối đa cho không vật cản lên đến 100m.



Hình 1.5: Mạch thu phát RF NRF24L01 + PA LNA 2.4Ghz Anten rời

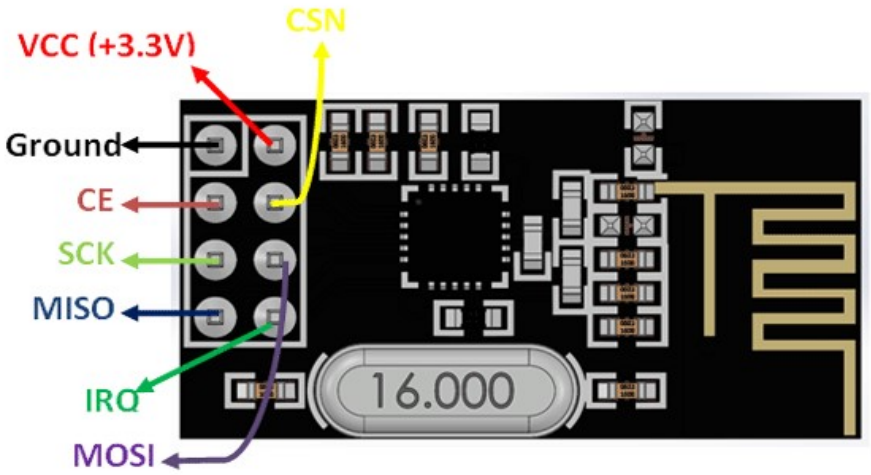
Khi nói đến các giải pháp RF hai chiều giá rẻ nhưng đáng tin cậy, không có mô-đun nào làm tốt hơn nRF24L01. Đây là lý do:

- Giá thành thấp - NRF24L01 là một trong những mô-đun truyền nhận không dây rẻ nhất trên thị trường, bạn có thể dễ dàng mua một cái trực tuyến với giá dưới 2$
- Dễ kết nối với vi điều khiển/Bo mạch Arduino - nRF24L01 có thể dễ dàng kết nối với một loạt các hệ thống vi điều khiển: MCU/ARM/PIC/AVR/STM32 bằng cách sử dụng giao thức SPI hoặc thư viện RF24 khi kết nối với Arduino.
- 2.4Ghz cho sự linh hoạt trong truyền thông không dây - Tần số hoạt động 2.4GHz cho phép sử dụng tốc độ bit cao hơn so với các tần số hoạt động thấp hơn. Nó sử dụng phương pháp điều chế GFSK cho việc truyền dữ liệu, có nghĩa là tốc độ truyền dữ liệu có thể là 250kbps, 1Mbps hoặc 2Mbps.
- Dải phát sóng cao - Khi sử dụng với cài đặt phù hợp, nRF24L01 có thể phát sóng sóng dài hơn vài kilomet. Tuy nhiên, nếu bạn cần một mô-đun độc lập với dải phát sóng cao, bạn có thể kiểm tra mô-đun nRF24L01+ này được thiết kế với bộ khuếch đại công suất và ăng-ten IPX cho truyền thông không dây lên đến 1000 mét (không có rào cản)

1.4.2. Thông số kỹ thuật của NRF24L01

- IC chính: NRF24L01+
- Điện áp cung cấp: 3.3VDC
- Điện áp giao tiếp GPIO: 3.3VDC, khi giao tiếp với các board mạch 5VDC cần nối tiếp qua trở hoặc sử dụng các mạch chuyển mức điện áp.
- Giao tiếp: SPI
- Dòng tiêu thụ: 45mA
- Tần số sóng: 2.4Ghz
- Sử dụng tương tự như NRF24L01 không có khuếch đại và có thể giao tiếp với các module không có khuếch đại PA và LNA.
- Tích hợp khuếch đại công suất phát PA (power amplifier) và LNA (Low Noise Amplifier)
- Công suất thu phát: 20dBm

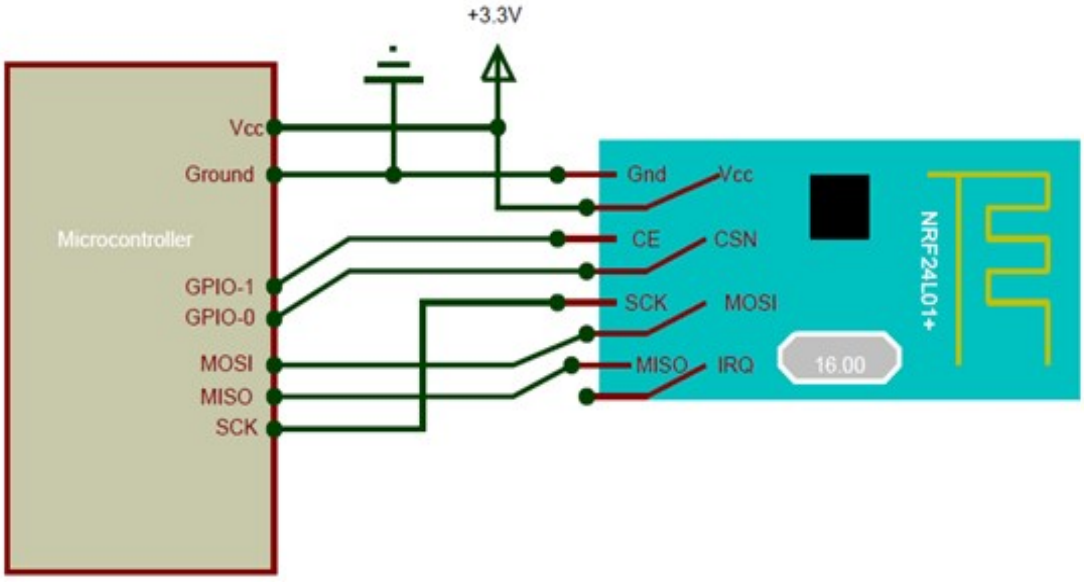
- Tốc độ truyền nhận tối đa: 2Mbit/s
- Chuẩn chân 2×8 tương tự các mạch NRF24L01 không có khuếch đại.



Hình 1.6: Sơ đồ chân Module NRF24L01

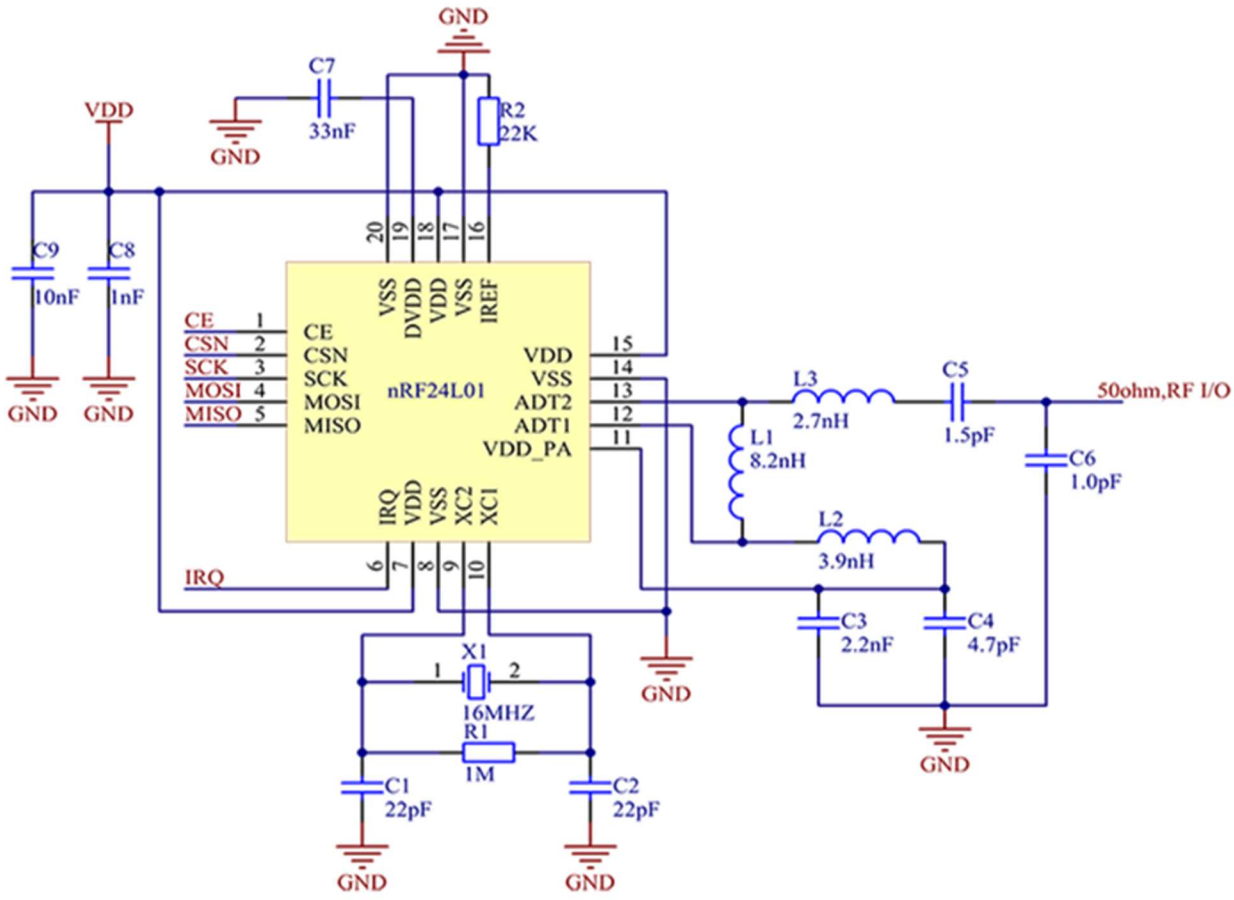
Pin number	Pin name	Abbreviation	Function
1	Ground	Ground	Kết nối chân Ground với hệ thống
2	Vcc	Nguồn	Nguồn dùng cho Module là 3.3V
3	CE	Chip Enable	Sử dụng để giao tiếp SPI
4	CSN	Ship Select Not	Pin này luôn kích hoạt ở mức cao, nếu không sẽ ngắt giao tiếp SPI
5	SCK	Serial Clock	Cung cấp xung clock phục vụ trong giao tiếp SPI
6	MOSI	Master Out Slave In	Kết nối với chân MOSI của MCU để Module nhận dữ liệu từ MCU
7	MISO	Master In Slave Out	Kết nối với chân MISO của MCU để Module gửi dữ liệu từ MCU

8	IRQ	Interrupt	Hoạt động ở mức thấp và chỉ được sử dụng nếu ngắt được yêu cầu

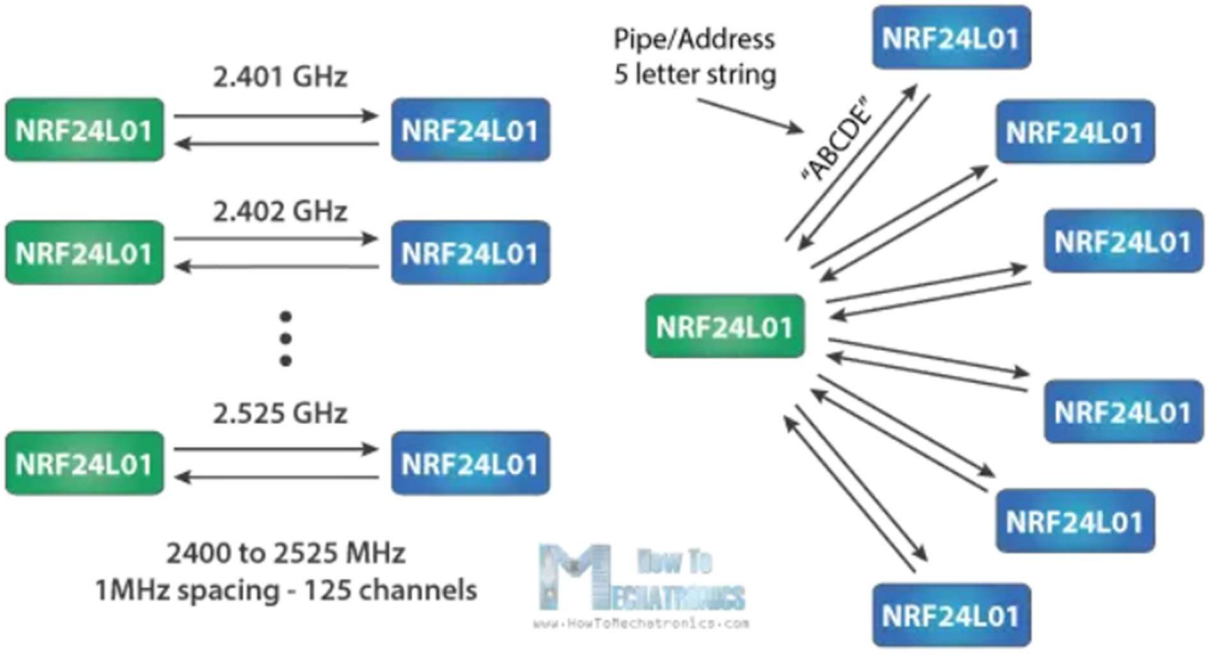


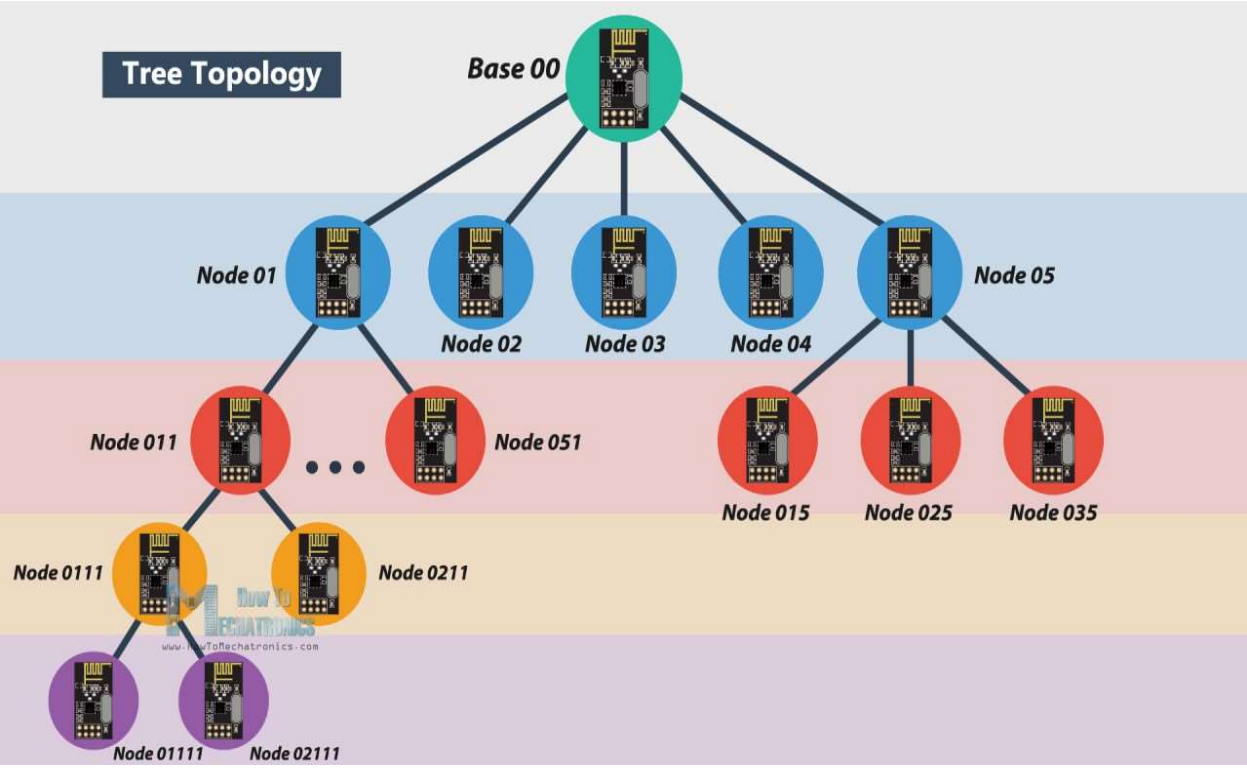
Hình 1.7: Sơ đồ nối chân NRF24L01 với MCU

1.4.3. Sơ đồ nguyên lý và phương thức hoạt động



Hình 1.8: Sơ đồ nguyên lý Module NRF24L01





Hình 1.9: Phương thức hoạt động của các Module NRF24L01

1.4.4. Ứng dụng của NRF24L01 trong thực tế

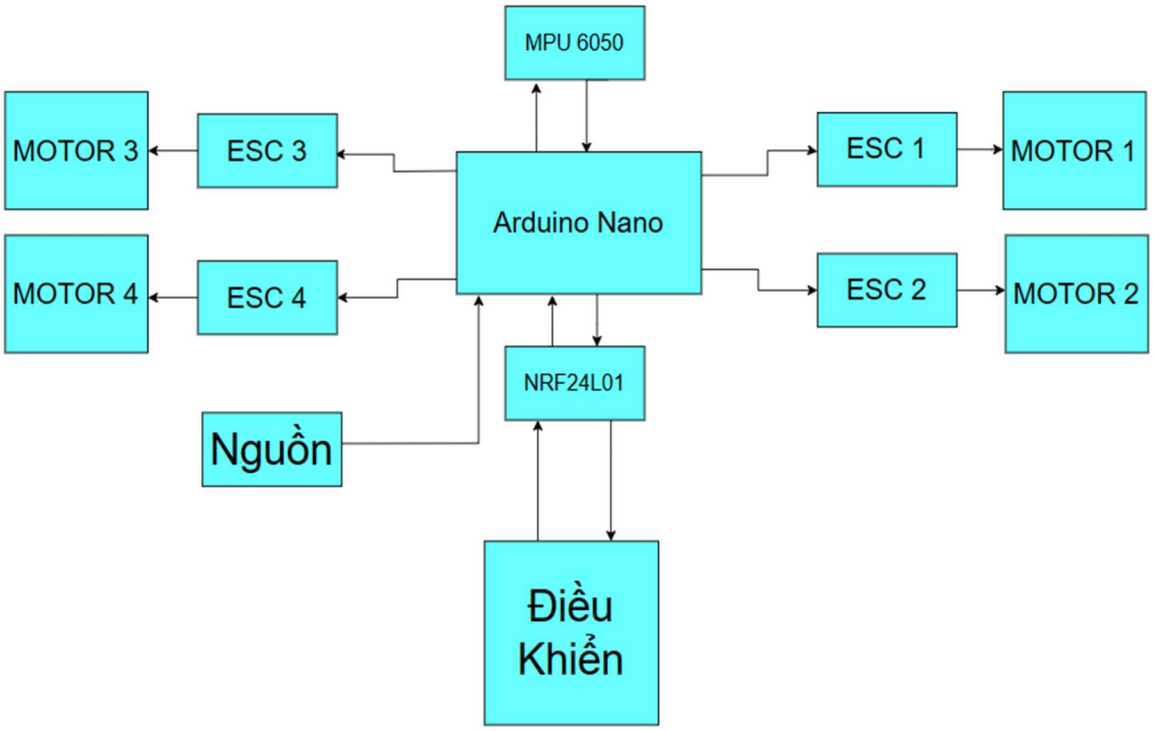
- Điều khiển từ xa: NRF24L01 có thể được sử dụng để tạo ra các hệ thống điều khiển từ xa cho đèn, quạt, hoặc các thiết bị điện gia dụng khác.
- Hệ thống báo động và an ninh: Nó có thể được tích hợp vào các hệ thống báo động nhà cửa hoặc cửa hàng để gửi thông báo khi có sự kiện xảy ra.
- Đo lường và giám sát môi trường: NRF24L01 có thể được kết nối với các cảm biến để thu thập dữ liệu từ môi trường như nhiệt độ, độ ẩm, ánh sáng và gửi dữ liệu về một trạm cơ sở.
- Hệ thống đo xa và điều khiển robot: Trong các dự án robot, NRF24L01 có thể được sử dụng để thiết lập kết nối không dây giữa điều khiển và robot.

- Mạng cảm biến không dây: NRF24L01 cho phép tạo ra một mạng cảm biến không dây với nhiều nút cảm biến gửi dữ liệu đến một trạm cơ sở hoặc trung tâm thu thập dữ liệu.
- Hệ thống truyền thông IoT (Internet of Things): NRF24L01 có thể được sử dụng trong các dự án IoT để kết nối các thiết bị IoT với nhau hoặc với Internet thông qua các gateway.
- Trò chơi và giải trí: Nó cũng có thể được sử dụng để tạo ra các hệ thống trò chơi không dây hoặc các hệ thống giải trí không dây khác.
- Mạng cảm biến môi trường trong nông nghiệp: NRF24L01 có thể được sử dụng để giám sát môi trường trong nông nghiệp, giúp người nông dân theo dõi và điều chỉnh các yếu tố như nhiệt độ, độ ẩm và độ phơi sáng.

CHƯƠNG 2. THIẾT KẾ HỆ THỐNG DRONE

2.1. Sơ đồ hệ thống thiết bị bay Drone

***Drone**



Hình 2.1. Sơ đồ khối của Drone

- Khối nguồn:

  Pin Lipo 2200 mAh

  IC ASM 1117

  Module mạch hạ nguồn LM2596

- Khối vi điều khiển

  Vi điều khiển Arduino Nano

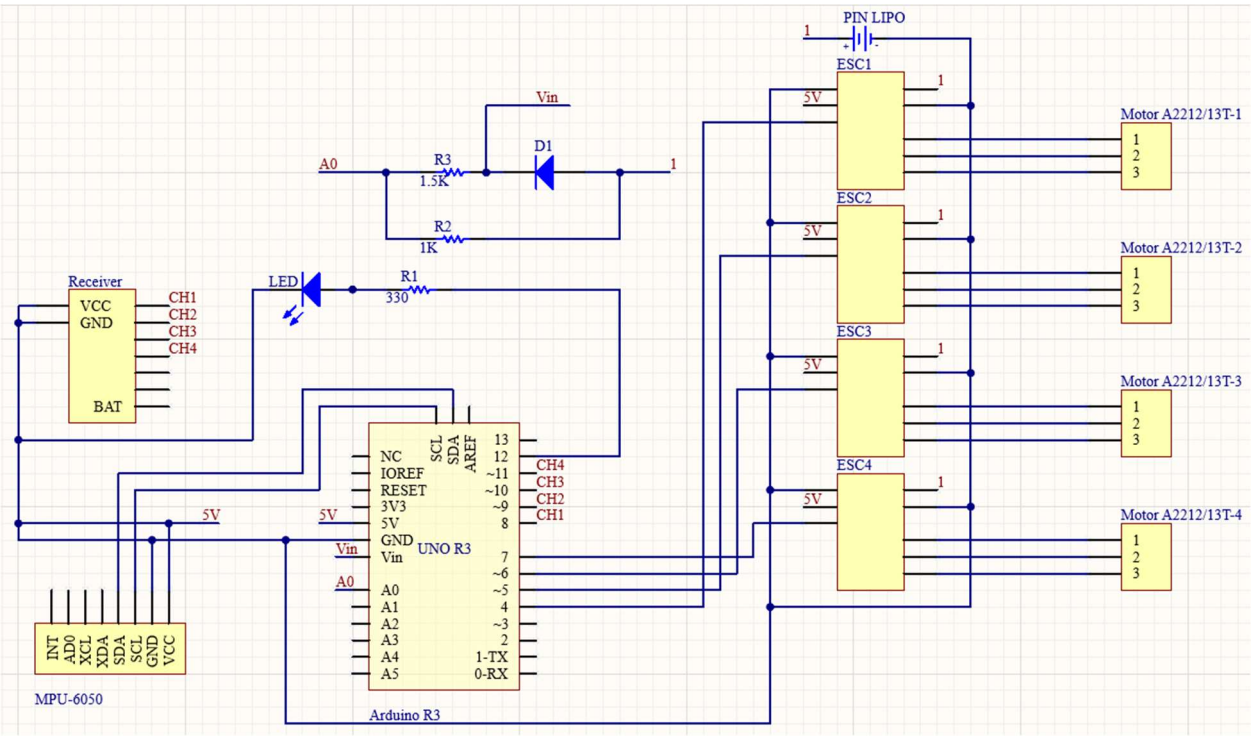
- Khối cảm biến

  Cảm biến gia tốc MPU6050

- Khối thu phát tín hiệu

  Module thu phát NRF24L01

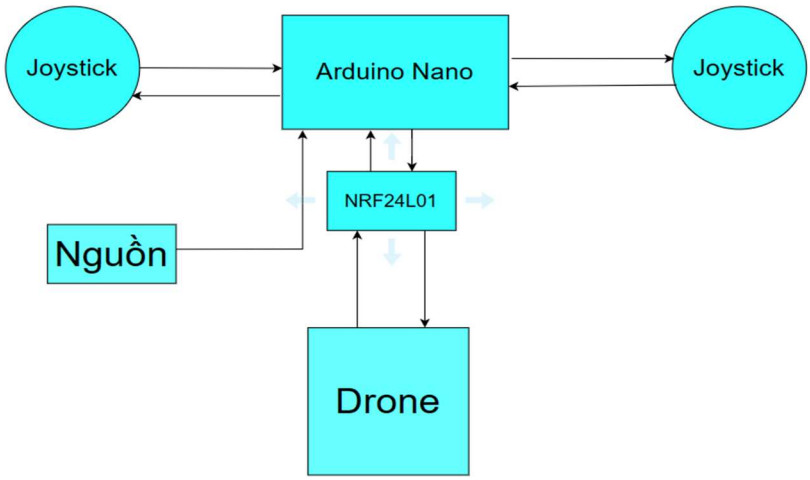
Từ sơ đồ khối ta thực hiện thiết kế sơ đồ nguyên lý trên phần mềm Altium như đưa ra ở hình dưới đây.



Hình 2.2. Sơ đồ nguyên lý của thiết bị drone được thiết kế

***Tay điều khiển**

Sơ đồ khối tay điều khiển được trình bày ở hình 2.3.



Hình 2.3. Sơ đồ khối tay điều khiển

- **Khối nguồn:**

  IC ASM 1117

  Pin 7.4 V

- Khối vi điều khiển

Gồm 1 Kit vi điều khiển Arduino Nano

- Khối điều khiển

  Joystick

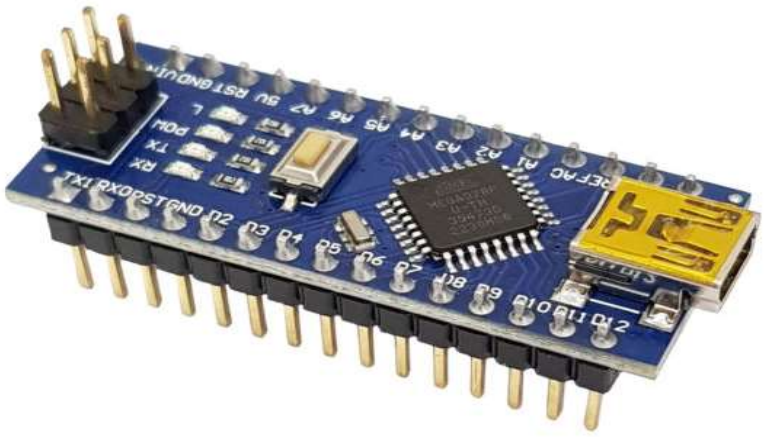
- Khối thu phát tín hiệu

  Gồm 1 Kit mạch thu phát NRF24L01

2.2. Lựa chọn linh kiện

2.2.1. Vi điều khiển Arduino Nano

Arduino Nano là 1 bo mạch phát triển thông minh được thiết kế để xây dựng các mẫu prototype nhanh chóng với kích thước nhỏ nhất. Arduino Nano cung cấp đầy đủ giao diện cho các ứng dụng thân thiện với breadboard. Trung tâm của bo mạch là vi điều khiển Atmega328 hoạt động với tần số 16 MHz, có chức năng gần giống Arduino Duemilanove. Bo mạch cung cấp 20 chân input/output digital, 8 chân analog và 1 cổng mini-USB.



Hình 2.1: Vi điều khiển Arduino Nano

Với 1 số lượng lớn các chân input/output, đem lại lợi thế của việc sử dụng nhiều giao tiếp serial như UART, SPI, I2C. Phần cứng tương thích với Arduino IDE, Arduino CLI… Từ đó mở ra nhiều hướng phát triển ứng dụng với vi điều khiển Arduino Nano:

- An ninh: Khả năng cung cấp hiệu suất cao và tiết kiệm năng lương mở ra cơ hội phát triển các ứng dụng an ninh như hệ thống kiểm soát truy cập sử dụng

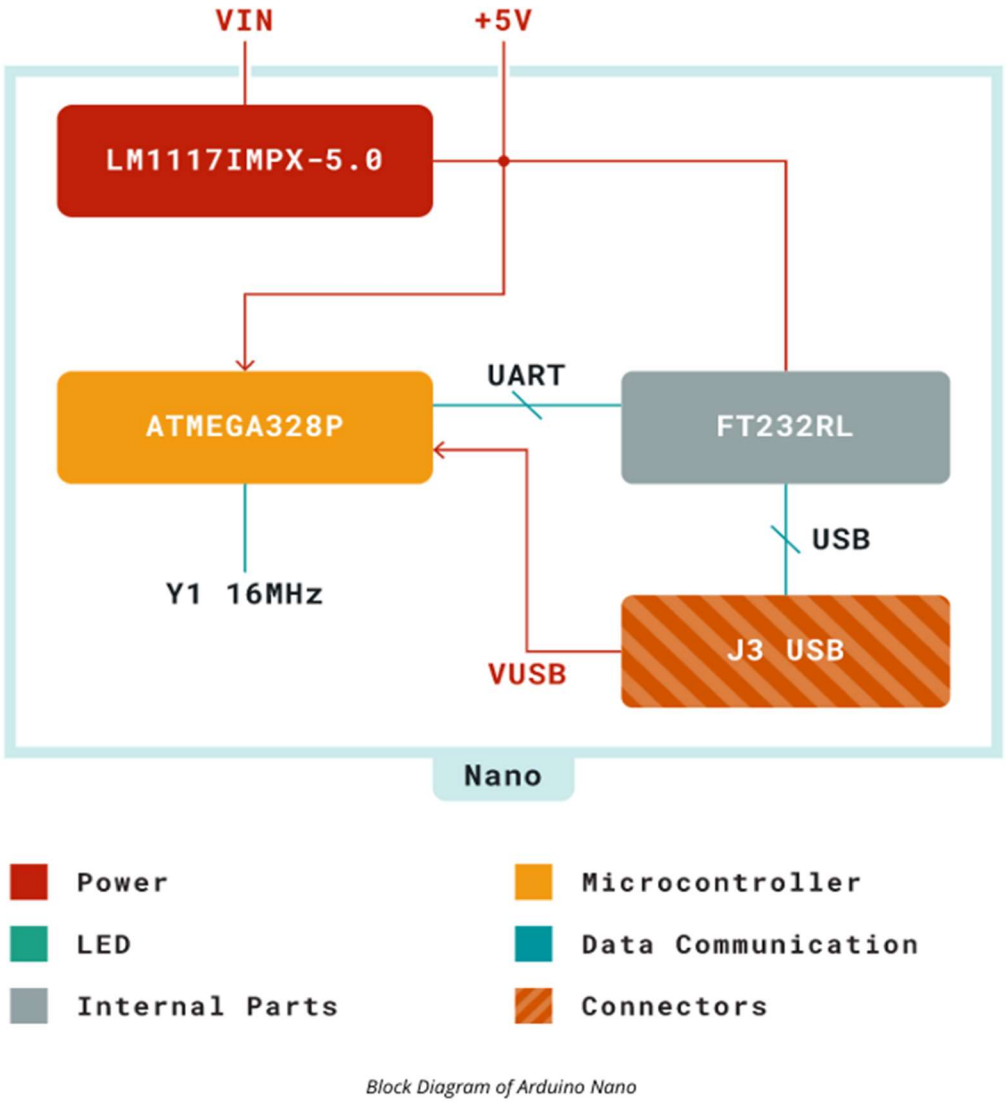
các loại cảm biến. Sự linh hoạt trong việc giao tiếp với cảm biến và các thiết bị ngoại vi bằng cách sử dụng giao tiếp serial đã nâng cao phạm vi sử dụng.

- Môi trường: Tính năng tiết kiệm năng lượng của vi điều khiển Arduino Nano và các tùy chọn cung cấp nguồn điện cho bo mạch đã tăng cường khả năng triển khai các dự án IOT từ xa liên quan đến vấn đề môi trường.
- Điều khiển – Robot: Đây luôn là lĩnh vực yêu thích của cộng đồng phát triển Arduino. Với phần cứng nhỏ gọn, ta có thể tạo ra các ứng dụng robot phức tạp hoặc các thiết bị điều khiển từ xa tiên tiến như UAV, robot, oto…

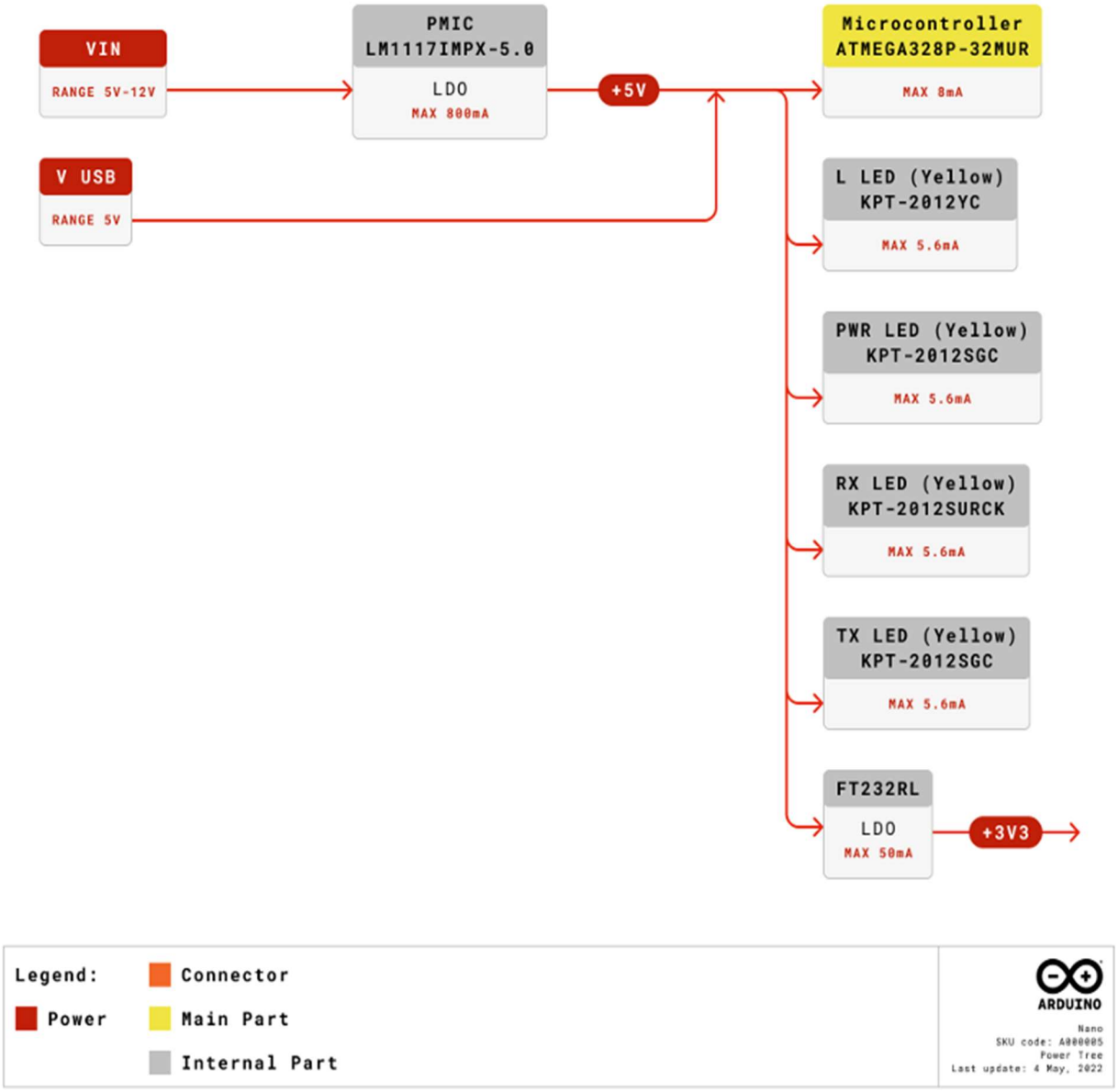
***Thông số kỹ thuật:**

- **Atmega328** Microcontroller:
  - Bộ xử lý 8-bit hiệu suất cao, tiết kiệm năng lượng
  - Đạt tới 16 MIPS cho tần số 16 MHz
  - Bộ nhớ 32 kB, trong đó có 2 kB được sử dụng cho bootloader
  - 2 kB SRAM
  - 1 kB EEPROM
  - 32x8 thanh ghi
  - Bộ đếm thời gian thực với dao động riêng biệt
  - 6 kênh PWM
  - Giao tiếp serial USART có thể lập trình
  - Giao tiếp serial SPI Master/Slave
- Nguồn:
  - Cổng kết nối Mini-B USB
  - Nguồn bên ngoài không ổn định từ 7-15V (chân 30)
  - Nguồn bên ngoài ổn định 5V (chân 27)
- Sleep Modes:
  - Idle
  - Giảm nhiễu ADC
  - Power-save
  - Power-down
  - Standby
- I/O:
  - 20 chân Digital
  - 8 chân Analog
  - 6 chân output PWM

***Sơ đồ khối và Power Tree của Arduino Nano:**



Hình 2.2: Sơ đồ khối Arduino Nano

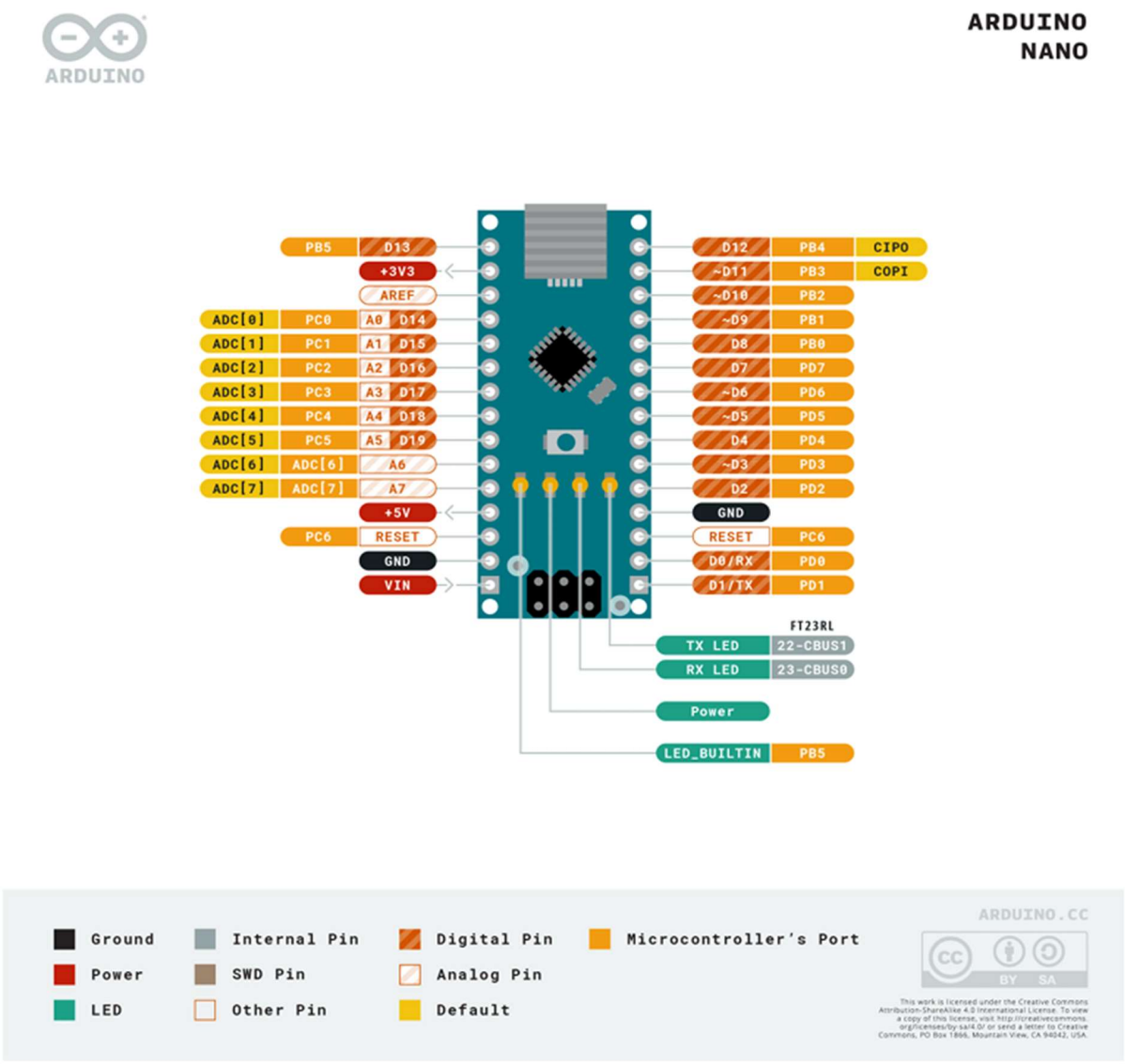


Power Tree of Arduino Nano

Hình 2.3: Power tree Arduino Nano

Arduino Nano có thể được cung cấp nguồn thông qua cổng USB hoặc thông qua chân VIN. Nguồn vào của chân VIN được điều chỉnh bởi 1 LDO (Low Dropout Regulator), do đó nguồn được giới hạn ở mức 5V để đảm bảo hoạt động tối ưu của bo mạch. Ngoài ra, còn có 1 bộ điều chỉnh áp khác giới hạn điện áp xuống 3,3V để cung cấp nguồn cho các linh kiện có yêu cầu điện áp thấp.

***Connector Pinouts**



Hình 2.4: Sơ đồ chân Arduino Nano

***Atmega328**

Pin	Function	Type	Description
1	PB0	Internal	Serial Wire Debug
2	PB1	Internal	Serial Wire Debug
3	PB2	Internal	Serial Wire Debug
4	PB3	Internal	Serial Wire Debug
5	PB4	Internal	Serial Wire Debug
6	PB5	Internal	Serial Wire Debug

***Analog:**

Pin	Function	Type	Description
1	+ 3V3	Power	5V USB power
2	A0	Analog	Analog input 0/GPIO
3	A1	Analog	Analog input 1/GPIO
4	A2	Analog	Analog input 2/GPIO
5	A3	Analog	Analog input 3/GPIO
6	A4	Analog	Analog input 4/GPIO
7	A5	Analog	Analog input 5/GPIO
8	A6	Analog	Analog input 6/GPIO
9	A7	Analog	Analog input 7/GPIO
10	+5V	Power	+5V Power Rail
11	Reset	Reset	Reset
12	GND	Power	Ground
13	VIN	Power	Voltage input

***Digital**

Pin	Function	Type	Description
1	D1/TX1	Digital	Digital Input 1/GPIO
2	D0/RX0	Digital	Digital Input 0/GPIO
3	D2	Digital	Digital Input 2/GPIO
4	D3	Digital	Digital Input 3/GPIO
5	D4	Digital	Digital Input 4/GPIO
6	D5	Digital	Digital Input 5 /GPIO
7	D6	Digital	Digital Input 6/GPIO
8	D7	Digital	Digital Input 7/GPIO
9	D8	Digital	Digital Input 8/GPIO
10	D9	Digital	Digital Input 9/GPIO

Có tổng cộng 14 chân kỹ thuật số và 8 chân Analog trên bo mạch Arduino Nano. Các chân kỹ thuật số có thể được sử dụng để giao tiếp với cảm biến bằng cách sử dụng chúng như là chân vào hoặc điều khiển tải bằng cách sử dụng chúng như là chân ra. Một hàm đơn giản như pinMode (…) và digitalWrite (…) có thể được sử dụng để kiểm soát hoạt động của chúng. Điện áp hoạt động là 0V và 5V cho các chân kỹ thuật số. Các chân Analog có thể đo điện áp tương tự từ 0V đến 5V bằng cách sử dụng bất kỳ trong 8 chân Analog thông qua một hàm đơn giản như analog-Read ().

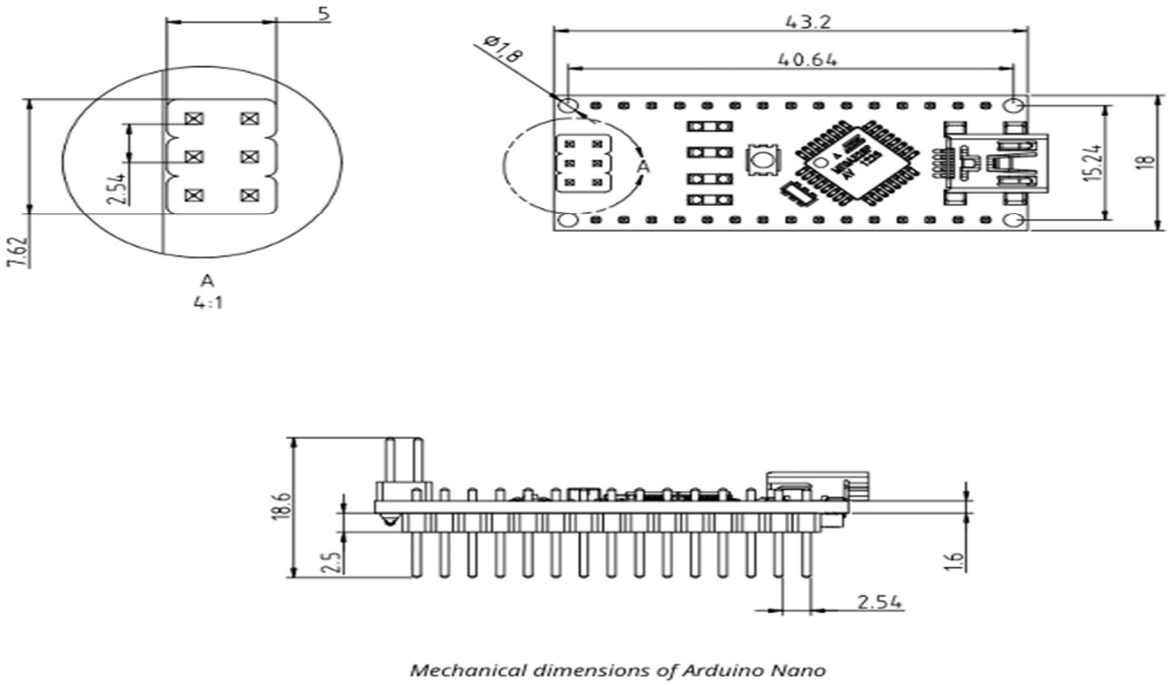
Ngoài việc phục vụ mục đích chính của chúng, các chân này cũng có thể được sử dụng cho mục đích đặc biệt, được thảo luận dưới đây:

- Chân Serial 0 (Rx) và 1 (Tx): Các chân Rx và Tx được sử dụng để nhận và truyền dữ liệu serial TTL. Chúng được kết nối với vi điều khiển ATmega328P thông qua chip USB sang TTL tương ứng.
- Chân Ngoại vi Ngắt 2 và 3: Các chân này có thể được cấu hình để kích hoạt một ngắt trên mức thấp, một cạnh dốc dương hoặc âm, hoặc một thay đổi trong giá trị.
- Các chân PWM 3, 5, 6, 9 và 11: Các chân này cung cấp một đầu ra PWM 8-bit bằng cách sử dụng hàm analog-Write ().
- Các chân SPI 10 (SS), 11 (MOSI), 12 (MISO) và 13 (SCK): Các chân này được sử dụng cho giao tiếp SPI.
- Chân LED 13: Chân này được kết nối với một đèn LED tích hợp sẵn. Khi chân 13 là HIGH - LED sẽ sáng và khi chân 13 là LOW, nó sẽ tắt.
- I2C A4 (SDA) và A5 (SCL): Sử dụng cho giao tiếp I2C bằng thư viện Wire.
- AREF: Sử dụng để cung cấp điện áp tham chiếu cho các chân vào tương tự với hàm analogReference(…).
- Chân Reset: Khi chân này LOW, vi điều khiển sẽ được reset.

***Các chân sử dụng trong sản phẩm:**

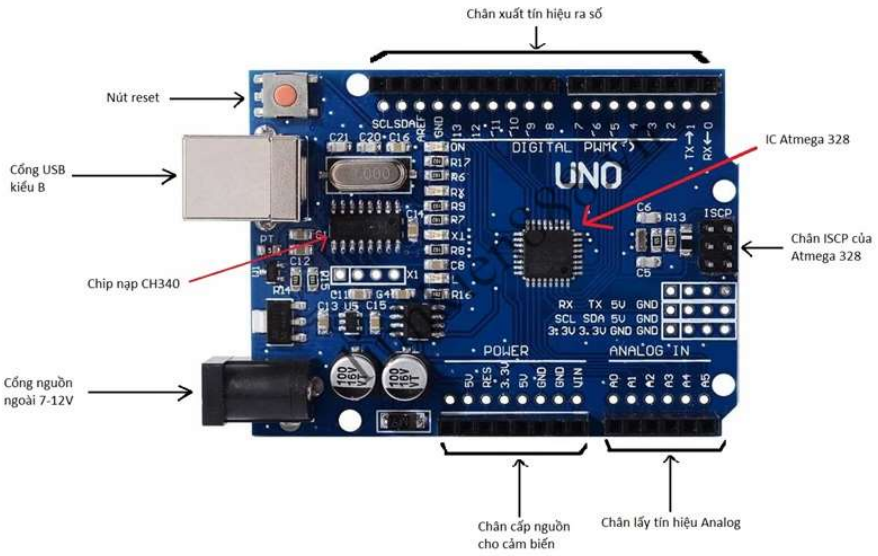
- D2-D5 đùng dể điều khiển động cơ motor.
- MISO: Đầu vào hoặc đầu ra, dùng trong giao tiếp SPI.
- SCK: Đầu ra, đồng hồ từ Master đến Slave.
- MOSI: Đầu vào hoặc đầu ra, dùng trong giao tiếp SPI.
- RST: Đầu vào, dùng để đặt lại (hoạt động ở mức thấp).
- Vin: Đầu ra, cung cấp điện thế.
- GND: Nối đất.

***Thiết kế cơ khí:**



Hình 2.5: Bản vẽ cơ khí Arduino Nano

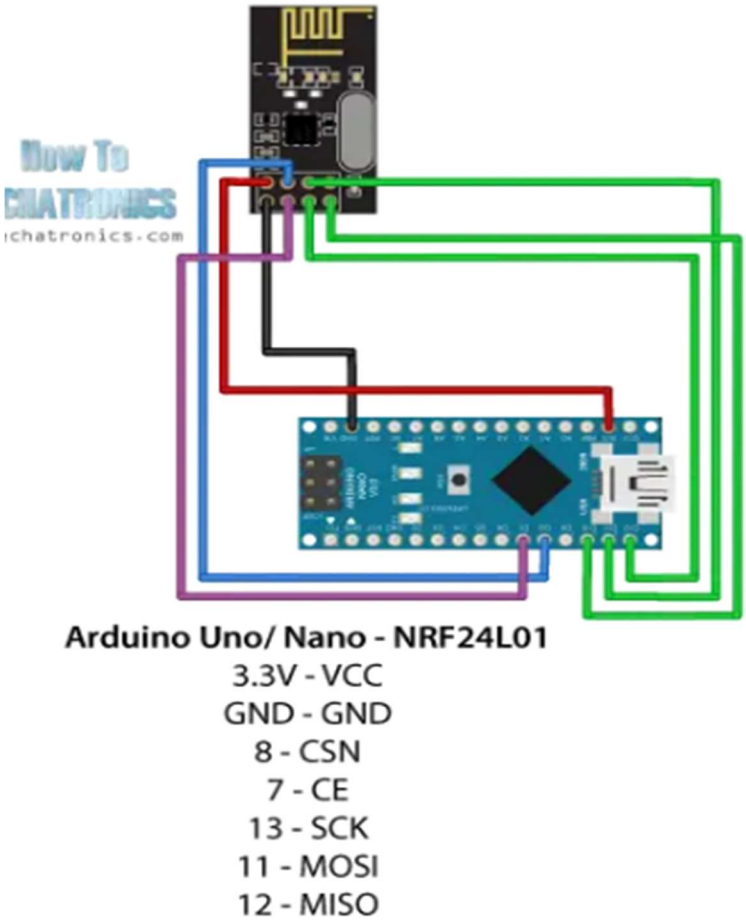
2.2.2. Arduino UNO R3 :



Hình 2.6: Arduino UNO R3

Vi điều khiển	ATmega328 họ 8bit
Điện áp hoạt động	5V DC (chỉ được cấp qua cổng USB)
Tần số hoạt động	16 MHz
Dòng tiêu thụ	khoảng 30mA
Điện áp vào khuyên dùng	7-12V DC
Điện áp vào giới hạn	6-20V DC
Số chân Digital I/O	14 (6 chân hardware PWM)
Số chân Analog	6 (độ phân giải 10bit)
Dòng tối đa trên mỗi chân I/O	30 mA
Dòng ra tối đa (5V)	500 mA
Dòng ra tối đa (3.3V)	50 mA
Bộ nhớ flash	32 KB (ATmega328) với 0.5KB dùng bởi bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)

2.2.3: Modulde NRF24L01



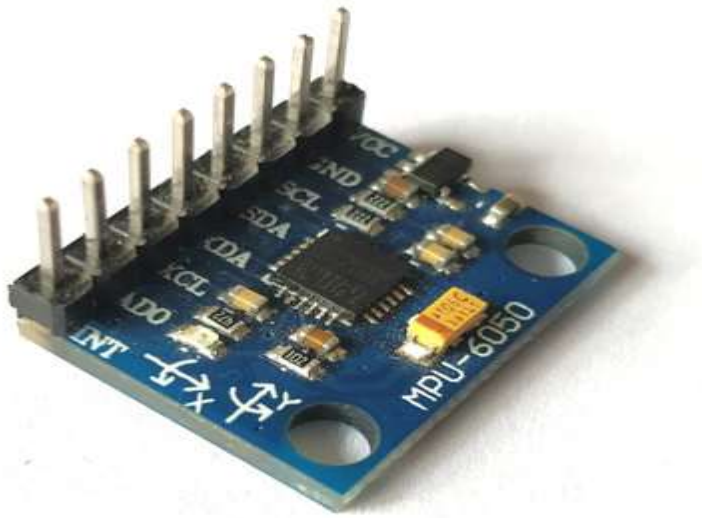
Hình 2.7: Sơ đồ nối chân NRF24L01 với Arduino Nano

*** Các chân dùng trong bài này:**

Pin numbers	Pin name	Function
1	MISO	Kết nối với chân D12 của MCU
2	MOSI	Kết nối với chân D11 của MCU
3	SCK	Kết nối với chân D13 của MCU
4	CE	Kết nối với chân D9 của MCU
5	CSN	Kết nối với chân D10 của MCU
6	GND	Ground
7	VCC	Kết nối với 3.3V của MCU

2.2.4. MPU 6050

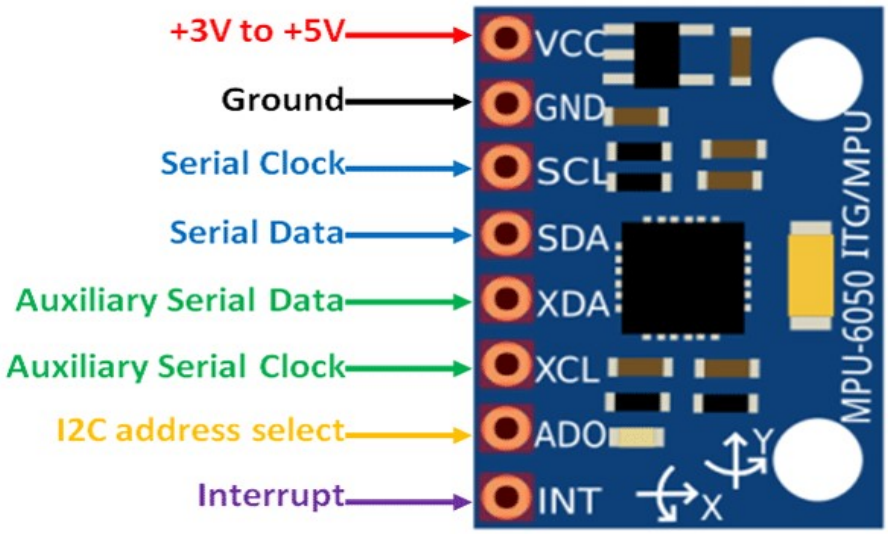
Mô-đun MPU6050 là Hệ thống cơ điện vi mô (MEMS) bao gồm Gia tốc kế 3 trục và Con quay hồi chuyển 3 trục bên trong nó. Điều này giúp chúng ta đo gia tốc, vận tốc, định hướng, độ dịch chuyển và nhiều thông số liên quan đến chuyển động khác của một hệ thống hoặc vật thể.



Hình 2.8: Cảm biến gia tốc MPU 6050

*** Thông số kĩ thuật:**

- Chip: MPU-6050 tích hợp 6 trục cảm biến (Đo được 3 trục góc + 3 trục gia tốc)
- Điện áp làm việc: 3 - 5V
- Giao tiếp: I2C
- Hỗ trợ ADC 16 Bit
- Độ phân giải góc: ±250 ±500 ±1000 ±2000 dps
- Độ phân giải gia tốc: ±2 ±4 ±8 ±16g
- Kích thước: 20.2 x 15.5mm



Hình 2.9: Sơ đồ chân cảm biến MPU 6050

Pin number	Pin name	Description
1	VCC	Nguồn cấp (3.3V hoặc 5V)
2	GND	Chân đất (Ground)
3	SCL	Chân xung Clock trong giao tiếp I2C
4	SDA	Chân dữ liệu (Data) trong giao tiếp I2C
5	XDA	Chân dữ liệu (Data) cho bus giao tiếp I2C thứ hai
6	XCL	Chân xung Clock cho giao tiếp I2C thứ hai
7	AD0	Chân địa chỉ định danh I2C (nếu được sử dụng)
8	INT	Chân ngắt (Interrupt) cho MPU6050

***Các chân sử dụng trong dự án:**

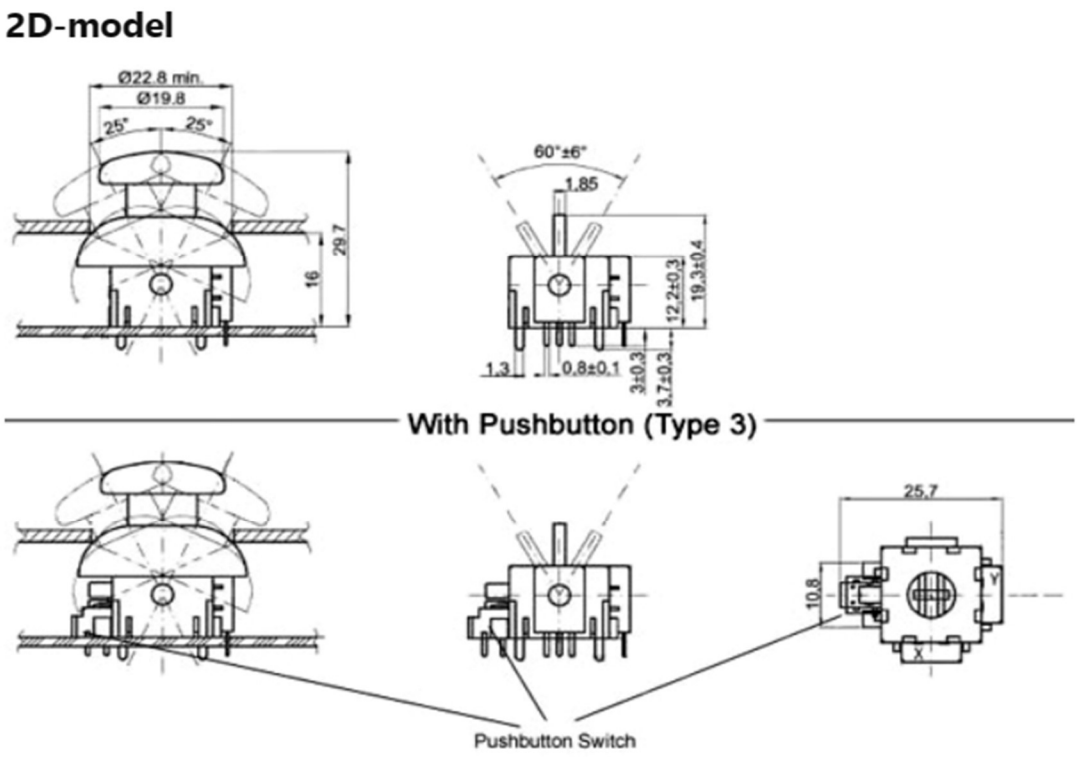
Pin Number	Pin Name	Description
1	Vcc	Cung cấp nguồn cho module, có thể là + 3V đến + 5V. Thông thường + 5V được sử dụng

2	Ground	Kết nối với mặt đất của hệ thống
3	Đồng bộ nối tiếp (SCL)	Được sử dụng để truyền dữ liệu thông qua giao tiếp I2C
4	Dữ liệu nối tiếp (SDA)	Có thể được sử dụng để giao tiếp các mô-đun I2C khác với MPU6050. Nó là tùy chọn

2.2.5. Joysticks

Mạch Joystick module thường được sử dụng để làm tay cầm hoặc cần điều khiển, mạch có kích thước nhỏ gọn với bốn lỗ ốc trên mạch rất dễ gá bắt. Mạch Joystick module rất dễ sử dụng, chỉ cần cấp nguồn cho mạch, các tín hiệu trả ra bao gồm tín hiệu Ananog của hai trục X, Y và 1 nút bấm.

Hình 2.10: Cần điều khiển đa hướng Joystick



Hình 2.11: Bản vẽ cơ khí Joystick

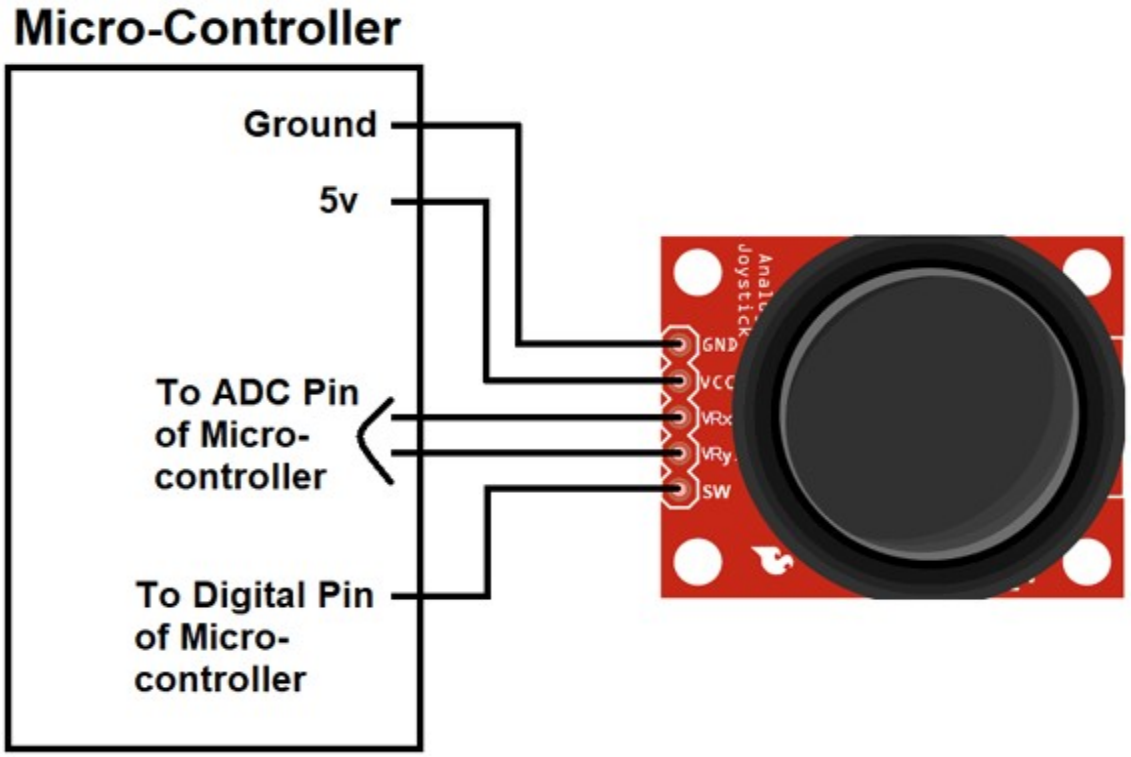
Mạch Joystick module thường được sử dụng để làm tay cầm hoặc cần điều khiển, mạch có kích thước nhỏ gọn với bốn lỗ ốc trên mạch rất dễ gắn. Mạch Joystick module rất dễ sử dụng, chỉ cần cấp nguồn cho mạch, các tín hiệu trả ra bao gồm tín hiệu Ananog của hai trục X, Y và 1 nút bấm...

*** Thông số kĩ thuật:**

- Nguồn cấp: Tùy chọn, thường cấp 3.3 hoặc 5VDC.
- Kiểu dạng tín hiệu ngõ ra 1 Digital và 2 Analog (1 nút nhấn và hai trục X, Y), mức tín hiệu theo nguồn cấp vào.
- Kích thước: 4.0 cm x 2.6 cm x 3.2 cm
- Trọng lượng: 12g.

*** Các chân sử dụng:**

Pin numbers	Pin name	Description
1	GND	Chân Ground của Module
2	+5V	Mức điện áp hoạt động
3	VRx	Điện áp tỉ lệ với trục X
4	Vry	Điện áp tỉ lệ với trục Y
5	SW	Nút nhấn



Hình 2.12: Sơ đồ nối chân Joystick với MCU

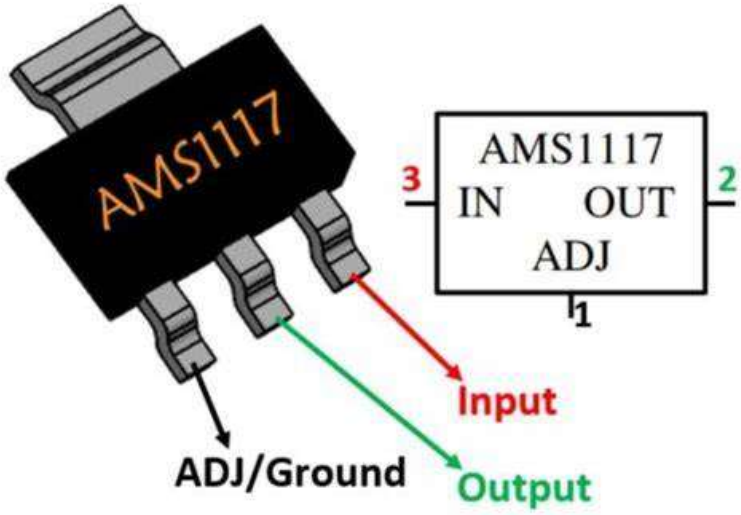
2.2.6. IC AMS 1117

AMS1117 là IC ổn áp 3 chân gói SMD phổ biến có nhiều model cho các yêu cầu điện áp cố định và có thể điều chỉnh. IC có thể cung cấp dòng điện tối đa 1A và điện áp đầu ra có thể thay đổi từ 1,5V đến 5V. Nó cũng có điện áp sụt thấp là 1,3V khi hoạt động ở dòng điện tối đa.

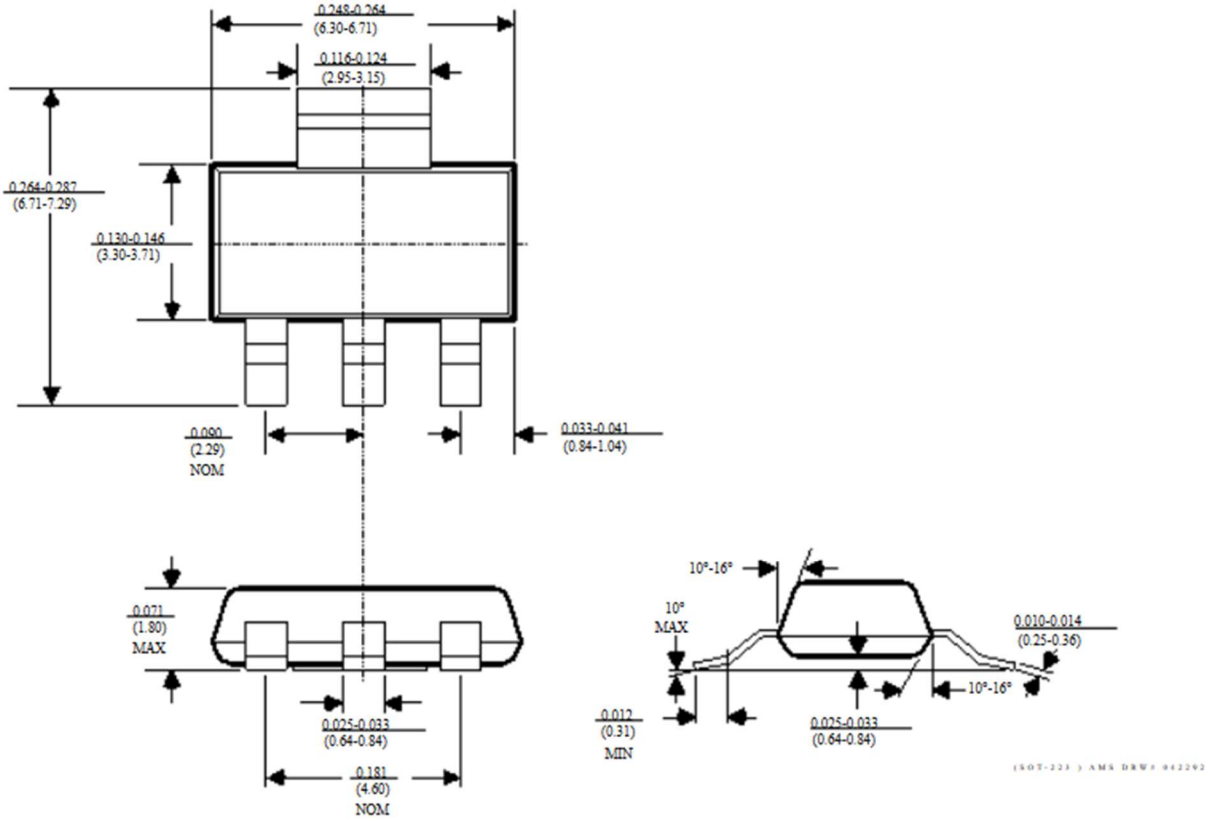
*** Thông số kĩ thuật:**

- Bộ điều chỉnh điện áp tuyến tính 3 cực có thể điều chỉnh hoặc cố định.
- Bộ điều chỉnh điện áp sụt thấp (LDO).
- Loại điện áp cố định: 1.5V, 1.8V, 2.5V, 2.85V, 3.3V và 5V.
- Phạm vi điện áp thay đổi: 1,25V đến 13,8V.
- Dòng điện đầu ra là 1000mA.
- Điện áp sụt tối đa: 1.3V.
- Giới hạn dòng điện tích hợp và bảo vệ nhiệt.
- Nhiệt độ hoạt động lớp tiếp giáp là 125 ° C.

- Gói SOT-223, TO-252 và SO-8.



Hình 2.13: IC AMS 1117



Hình 2.14: Bản vẽ cơ khí IC AMS 1117

2.2.7. Mạch giảm áp LM2596

Mạch Giảm Áp LM2596 là module giảm áp có khả năng điều chỉnh được dòng ra đến 3A. **LM2596** là IC nguồn tích hợp đầy đủ bên trong. Tức là khi cấp nguồn 12V vào module, sau khi giảm áp ta có thể lấp được nguồn 3A < 9v... như 5V hay 3.3V.

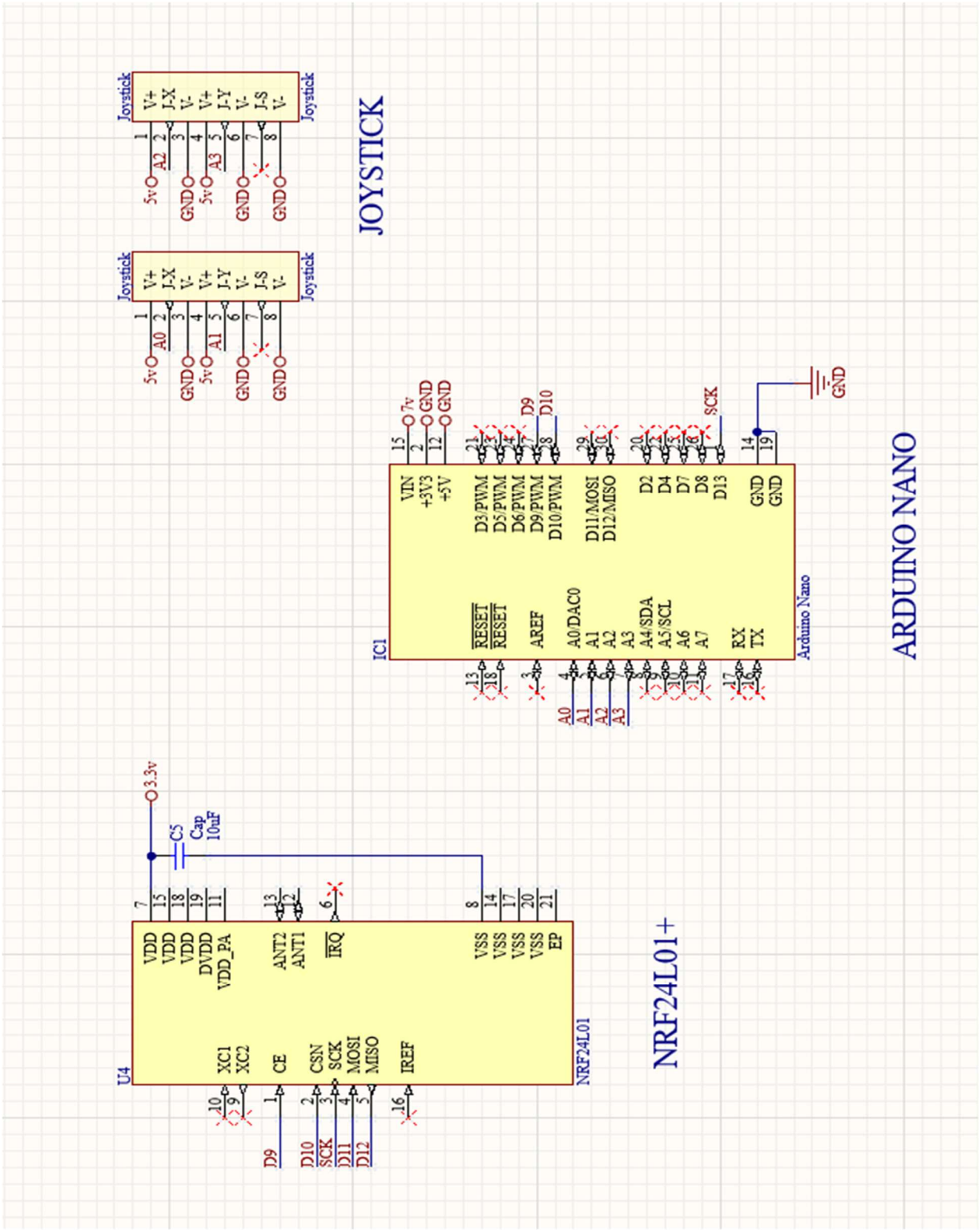
***Thông số kỹ thuật:**

- Module nguồn không sử dụng cách ly
- Nguồn đầu vào từ 4V - 35V.
- Nguồn đầu ra: 1V - 30V.
- Dòng ra Max: 3A
- Kích thước mạch: 53mm x 26mm
- Đầu vào: INPUT +, INPUT-
- Đầu ra: OUTPUT+, OUTPUT-

Hình 2.15: Mạch giảm áp LM2596

2.3. Thiết kế mạch nguyên lý

***Tay cầm điều khiển:**

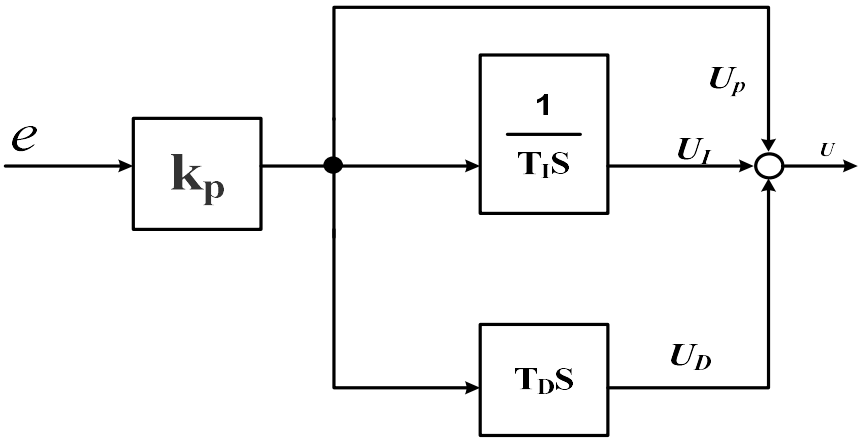


Hình 2.16: Mạch nguyên lý tay điều khiển Drone

2.4. Thuật toán PID

2.4.1. Khái quát về bộ điều khiển PID

Cấu trúc của bộ điều khiển PID (hình 2.17) gồm 3 thành phần là khâu khuếch đại (P), khâu tích phân (I) và khâu vi phân (D). Khi sử dụng thuật toán PID nhấ thiết phải lựa chọn chế độ làm việc P, I hay D và sau đó đặt tham số cho các chế độ đã chọn. Một cách tổng quát, có ba thuật toán cơ bản được sử dụng là P, PI và PID.



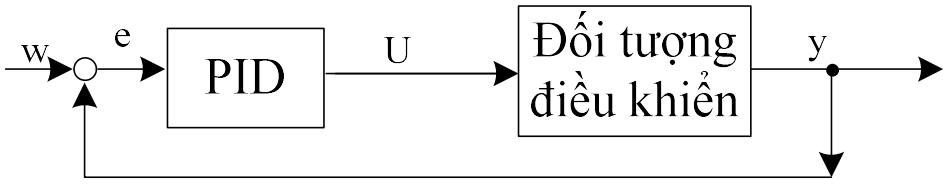
Hình 2.17: Cấu trúc bộ điều khiển PID. U_p U_I U_D

Bộ điều khiển PID có cấu trúc đơn giản, dễ sử dụng nên được sử dụng rộng rãi trong điều khiển các đối tượng SISO theo nguyên lý hồi tiếp (hình 4.4) bộ PID có nhiệm vụ đưa sai lệch e(t) của hệ thống về 0 sao cho quá trình quá độ thỏa mãn các yêu cầu cơ bản về chất lượng:

- Nếu sai lệch tĩnh e(t) càng lớn thì thông qua thành phần $u_P(t)$, tín hiệu điều chỉnh u(t) càng lớn.

- Nếu sai lệch e(t) chưa bằng 0 thì thông qua thành phần $u_I(t)$, PID vẫn còn tạo tín hiệu điều chỉnh.

- Nếu sự thay đổi của sai lệch e(t) càng lớn thì thông qua thành phần $u_D(t)$, phản ứng thích hợp của u(t) sẽ càng nhanh.



Hình 2.18: Điều khiển hồi tiếp với bộ điều khiển PID

Bộ điều khiển PID được mô tả bằng mô hình vào-ra:

$$u(t) = k_p \left[e(t) + \frac{1}{T} \int_0^t e(\tau) d\tau + \mathrm{T_D} \frac{de(t)}{dt} \right]$$

Trong đó :

e(t) – tín hiệu đầu vào.

u(t) – tín hiệu đầu ra.

k_p – tín hiệu khuếc đại

T_I – hằng số tích phân.

T_D – hằng số vi phân.

Từ mô hình vào – ra trên, ta có được hàm truyền đạt của bộ điều khiển PID

$$R(s) = k_p \left(1 + \frac{1}{T_I s} + T_D s \right)$$

Có nhiều phương pháp xác định tham số của bộ điều khiển PID:

- **Phương pháp Ziegler-Nichols.**
- Phương pháp Chien-Hrones-Reswick.
- Phương pháp tổng T của kuhn.
- Phương pháp tối ưu modul và phương pháp tối ưu đối xứng.
- Phương pháp tối ưu theo sai lệch bám

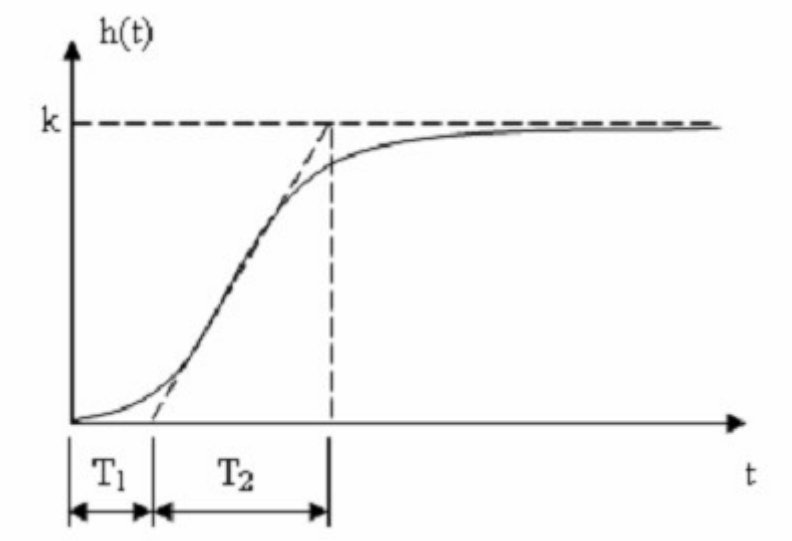
Đề tài sử dụng phương pháp Ziegler-Nichols nên tôi chỉ đi sâu vào phương pháp này, các bạn có thể tìm hiểu thêm các nguyên tắc khác như đã nêu ở trên.

2.4.2. Phương pháp Ziegler-Nichols.

Phương pháp Ziegler-Nichols là phương pháp thực nghiệm để xác định tham số bộ điều khiển P, PI hoặc PID bằng cách dựa vào đáp ứng quá độ của đối tượng điều khiển. Tùy theo đặc điểm của từng đối tượng, Ziegler và Nichols đưa ra hai phương pháp lựa chọn tham số của bộ điều khiển

Phương pháp Ziegler-Nichols thứ nhất:

Phương pháp này áp dụng cho các đối tượng có đáp ứng đối với tín hiệu vào là hàm nấc có dạng chữ S (hình 2.19) như nhiệt độ lò nhiệt, tốc độ động cơ…



Hình 2.19: đáp ứng nấc của hệ hở có dạng chữ S.

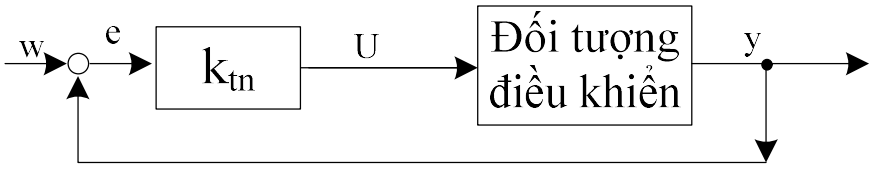
Thông số của bộ điều khiển được chọn theo bảng sau:

Bảng 2.1: các tham số PID theo phương pháp Ziegler-Nichols thứ nhất

Thông số / BĐK	k_p	T_I	T_D
P	$T_2/(k.T_1)$	-	-
I	$0,9T_2/(k.T_1)$	$T_1/0,3$	-
D	1,2T2/(k.T1)	2T1	0,5T1

Phương pháp Ziegler-Nichols thứ hai.

Phương pháp này áp dụng cho đối tượng có khâu tích phân lý tưởng như mực chất lỏng trong bồn chứa, hệ truyền động dùng động cơ… đáp ứng quá độ của hệ hở của đối tượng tăng đến vô cùng. Phương pháp này được thực hiện như sau:

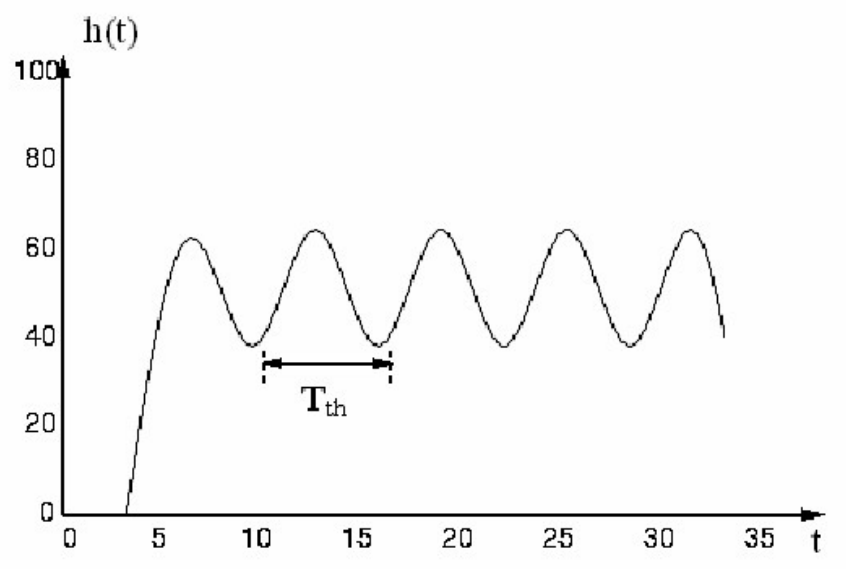


Hình 2.20: xác định hằng số khuếch đại tới hạn

- Thay bộ điều khiển PID trong hệ kín bằng bộ khuếc đại (hình 2.20)

- Tăng hệ số khuếc đại tới giá trị tới hạn k_{tn} để hệ kín ở chế độ biên giới ổnđịnh, tức là h(t) có dạng dao động điều hòa.

- Xác định chu kỳ T_{th} của dao động.



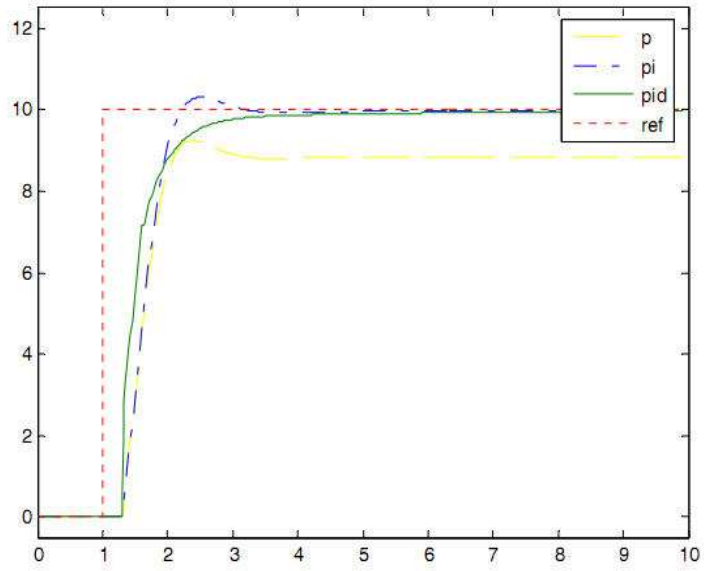
Hình 2.21: Đáp ứng nấc của hệ kín khi k= k_{tn}

Thông số của các bộ điều khiển được chọn theo bảng sau:

Bảng 2.2: Các tham số PID theo phương pháp Ziegler-Nichols thứ hai.

Thông số / BĐK	k_p	T_I	T_D
P	$0,5k_{tn}$	-	-
I	$0,45\ k_{tn}$	$0,85\ T_{th}$	-
D	$0,6\ k_{tn}$	$0,5\ T_{th}$	$0,125\ T_{th}$

Tính toán thông số cho bộ điều khiển PID.



Hình 2.22: So sánh đặc tính P,PI,PD,PID

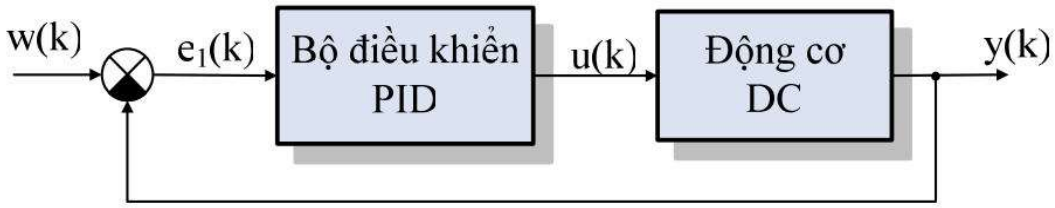
Hệ thống trên chỉ thực sự đạt hiệu quả khi ta thiết kế cho nó 1 bộ điều khiển phù hợp.Với kết cấu phần cứng và đặc tính điều khiển là điều khiển tốc độ thì bộ điều khiển PI số là thích hợp hơn cả, ta chỉ dùng thêm khâu vi phân nếu liên quan tới điều khiển vị trí.Do ứng dụng vi điều khiển ngày càng rộng rãi, chính vì vậy việc số hóa bộ điều khiển PI là rất quan trọng.Đi cùng với nó là các phương pháp cho ta số hóa một cách tương đối chính xác như Ziegler-Nichols…….v v.

Thuật toán điều khiển.

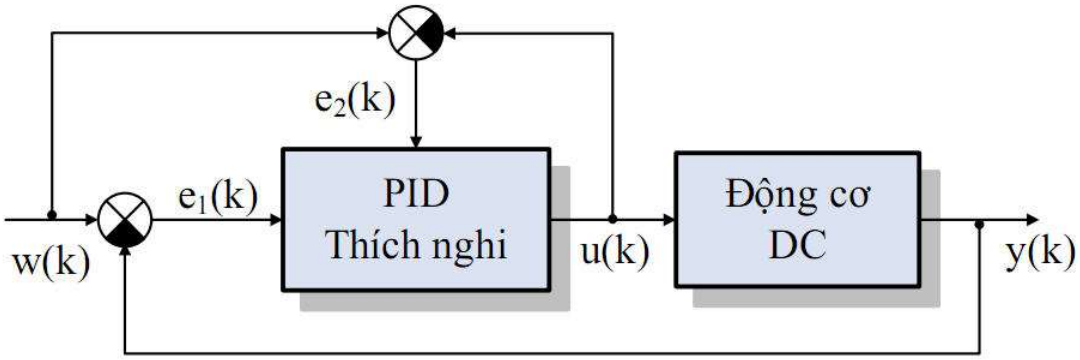
Về nguyên tắc khi xây dựng bộ điều khiển cho một đối tượng nào đó thì ta cần biết được cấu trúc, đặc tính, hàm truyền…của đối tượng.

Với động cơ 1 chiều bất kì thì ta không phải lúc nào cũng mổ xẻ động cơ ra rồi đo đạc các thông số của nó, chính vì vậy bộ điều khiển phải có nhiệm vụ tạo ra tín hiệu điều khiển phù hợp nhất, có khả năng điều khiển linh hoạt với nhiều động cơ khác nhau.

Với điều khiển PID thì có rất nhiều cách thức thực hiện, riêng bộ PID thì có PID thường và PID thích nghi



Hình 2.23: PID thường

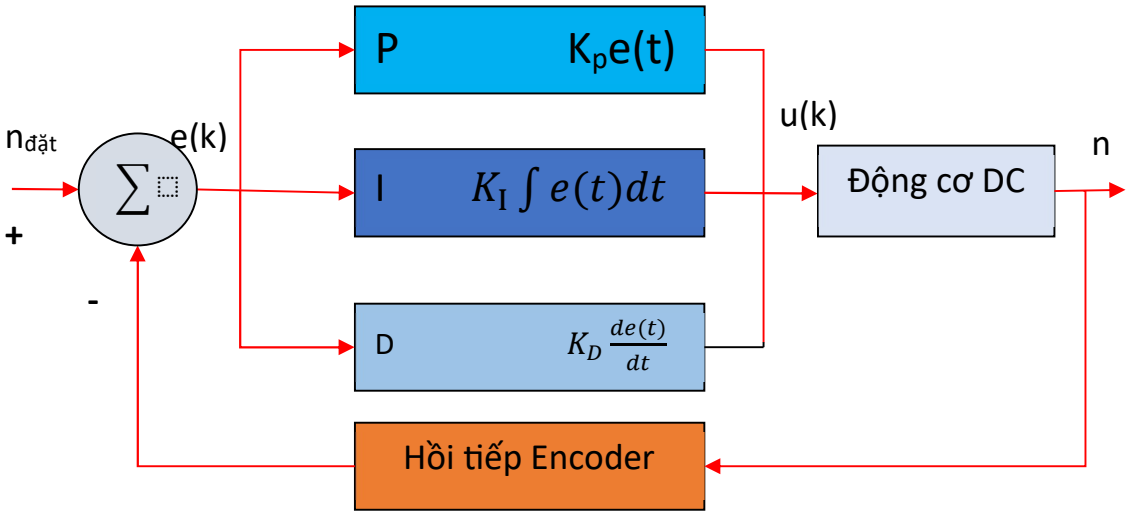


Hình 2.24: PID thích nghi

Với PID thường thì đầu ra chỉ phụ thuộc vào tín hiệu $e_1(k)$, còn với PID thích nghi thì còn phụ thuộc thêm vào $e_2(k)$.

Thực tế cho thấy PID thích nghi hoạt động tốt và ổn định hơn PID thường.

Phương trình toán học bộ PID.

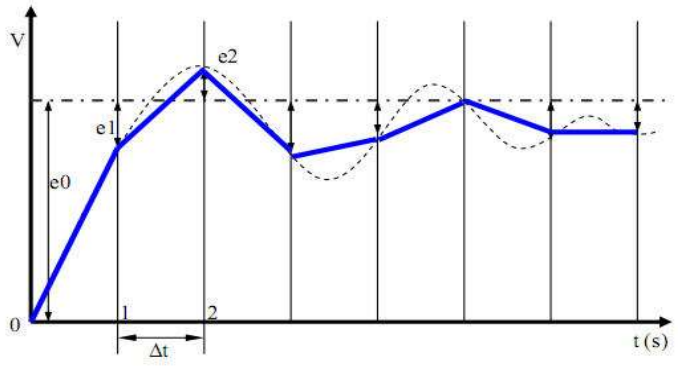


Hình 2.25: Sơ đồ khối bộ PID

$$U(t) = K_P * e(t) + K_I \int e(t) * dt + K_D \frac{de(t)}{dt}$$

Nếu bỏ thành phần vi phân đi ta chỉ còn PI:

$$U(t) = K_P * e(t) + K_I \int e(t) * dt$$



Chú thích: - đường chấm gạch biểu diễn vận tốc cần thiết v_set.
- đường gạch gạch biểu diễn vận tốc thực tế của động cơ.

Hình 2.26: Trích mẫu lấy tín hiệu

Δt là thời gian lấy mẫu (Samplingtime) là rất nhỏ.

Ta tính thành phần tích phân $\int_0^t e(t) * dt = \lim(\sum e(t) * \Delta t)_{\Delta t \to 0}$ Lấy gần đúng ta có

$\int_0^t e(t) * dt = \lim(\sum e(t) * \Delta t)_{\Delta t \to 0} = \sum e(i) * \Delta t$ trong đó i=0,1,2,…

Tóm lại ta có $U(t) = K_P * e + K_I * \sum e(i)\Delta t$

Đặt $e\_sum(i+1) = \sum e(i) = e\_sum(i) + e(i+1)$ Công thức trên được viết lại như sau: $U(t) = K_P * e + K_I e\_sum$

Áp dụng vào phần cứng điều khiển ta có ngõ ra bộ PI là %duty (PWM).

Ngõ vào là $e\_sum = v\_cur - v\_set$

v_cur là vận tốc hiện tại

v_set là vận tốc đặt

Sự phụ thuộc của vận tốc vào %duty là gần như tuyến tính cho nên để đơn gian ta coi nó là tuyến tính.Vì vậy ta có thể điều khiển vận tốc thông qua %duty.

- **Giải thuật lập trình để tính PWM như sau**

Kp,K_I là các hệ số xác định nhờ phương pháp Ziegler-Nichols.

e_2 =(v_set-v_cur) là sai lệch đang xét

e_1 là sai lệch trước đó

e_sum=e_2+e_1 là tổng các sai lệch tới thời điểm đang xét

e_sub=(e_2 - e_1) là độ biến thiên sai lệch

v_cur là vận tốc hiện tại

v_set là vận tốc đặt

Với encoder 100xung/vòng, ta trích mẫu 50ms, như vậy vận tốc động cơ tại thời điểm đang xét là v_cur = 12*INT0 (INT0 là số xung đếm được trong 50ms)

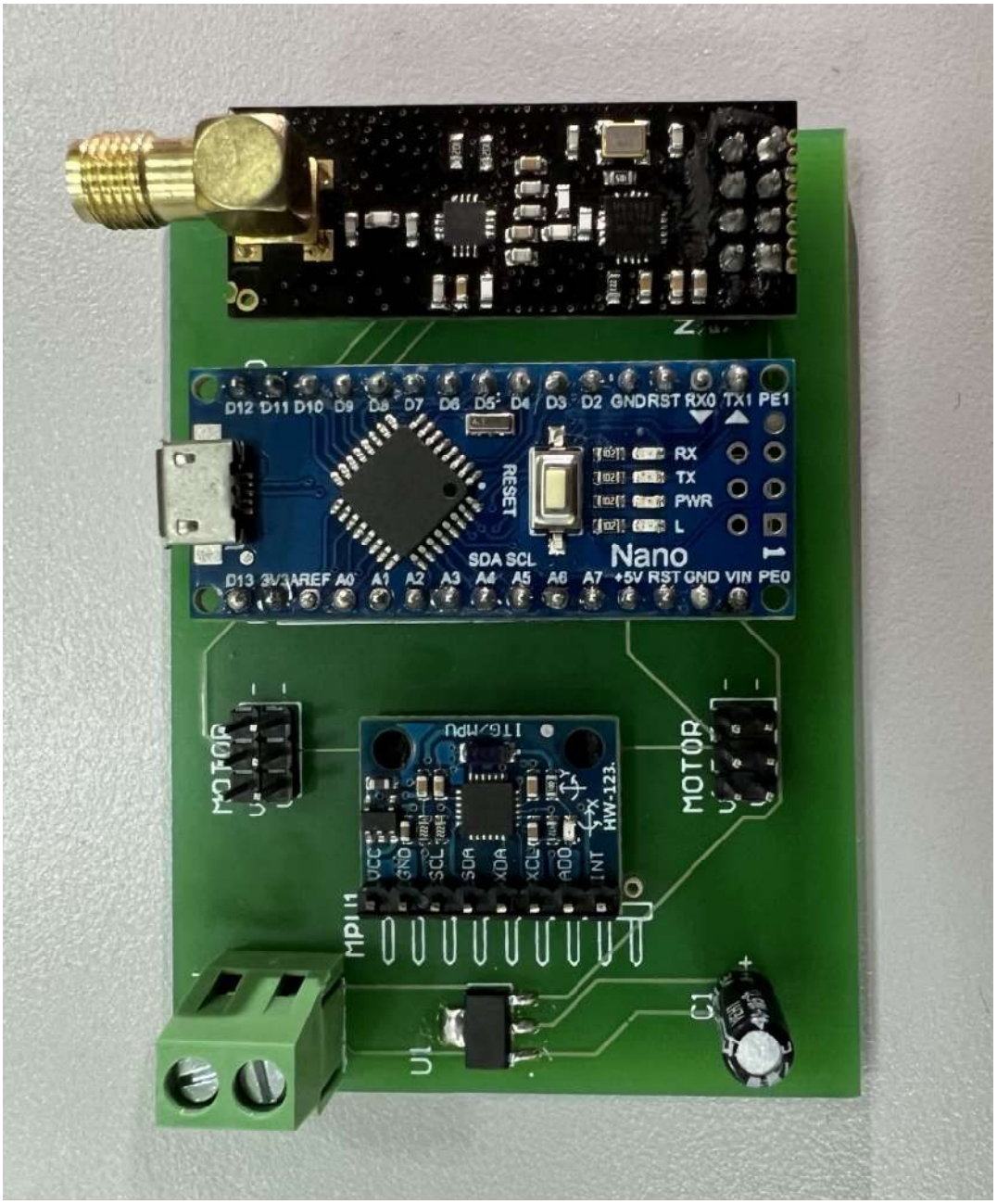
Ta đi đến hệ thức tính PWM cho PI cuối cùng là:

$$\mathbf{PWM=PWM + Kp*e\_2 + K_I*e\_sum}$$

CHƯƠNG 3. CHẾ TẠO VÀ THỬ NGHIỆM

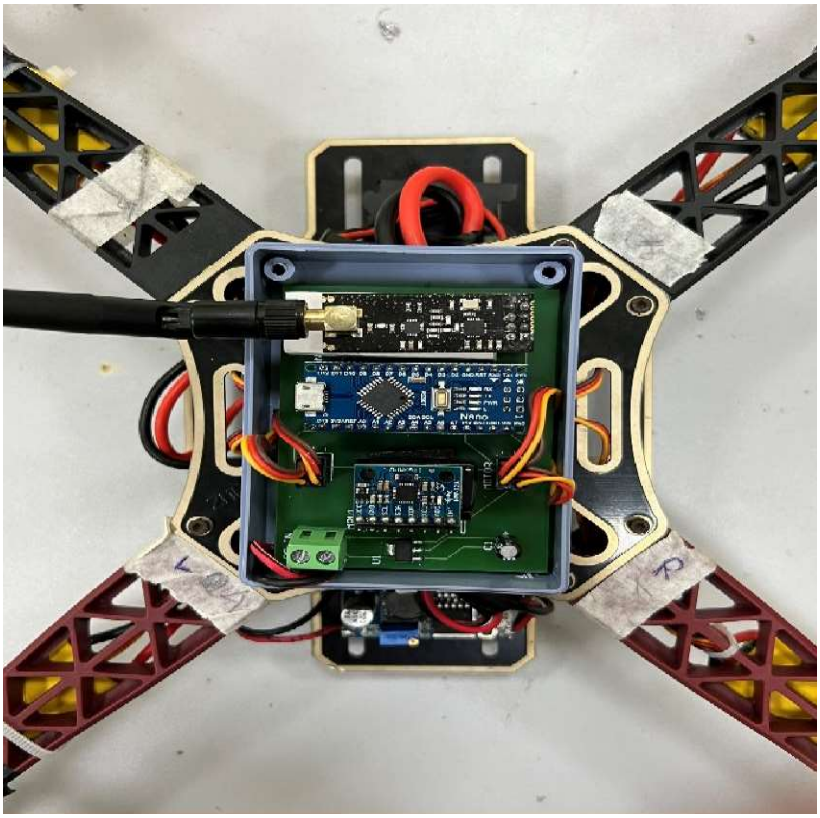
3.1. Lắp đặt sản phẩm

Sau khi thiết kế và lựa chọn linh kiện ta thực hiện chế tạo mạch in và lắp đặt cơ khí toàn bộ thiết bị drone. Chi tiết từng phần mạch PCB và cơ khí như các hình dưới đây.

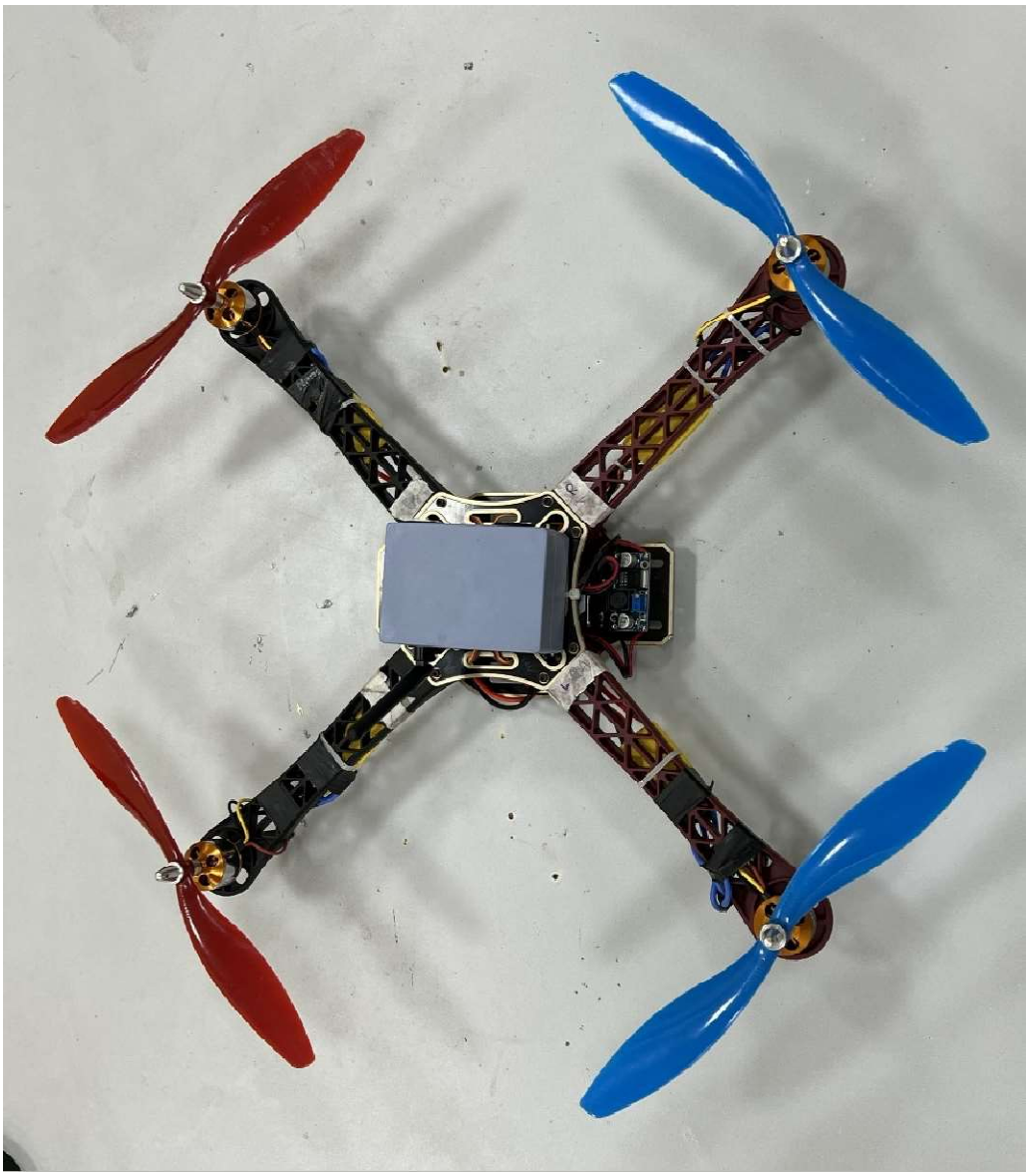


Hình 3.1: Mạch PCB drone sau khi gắn các bộ phận

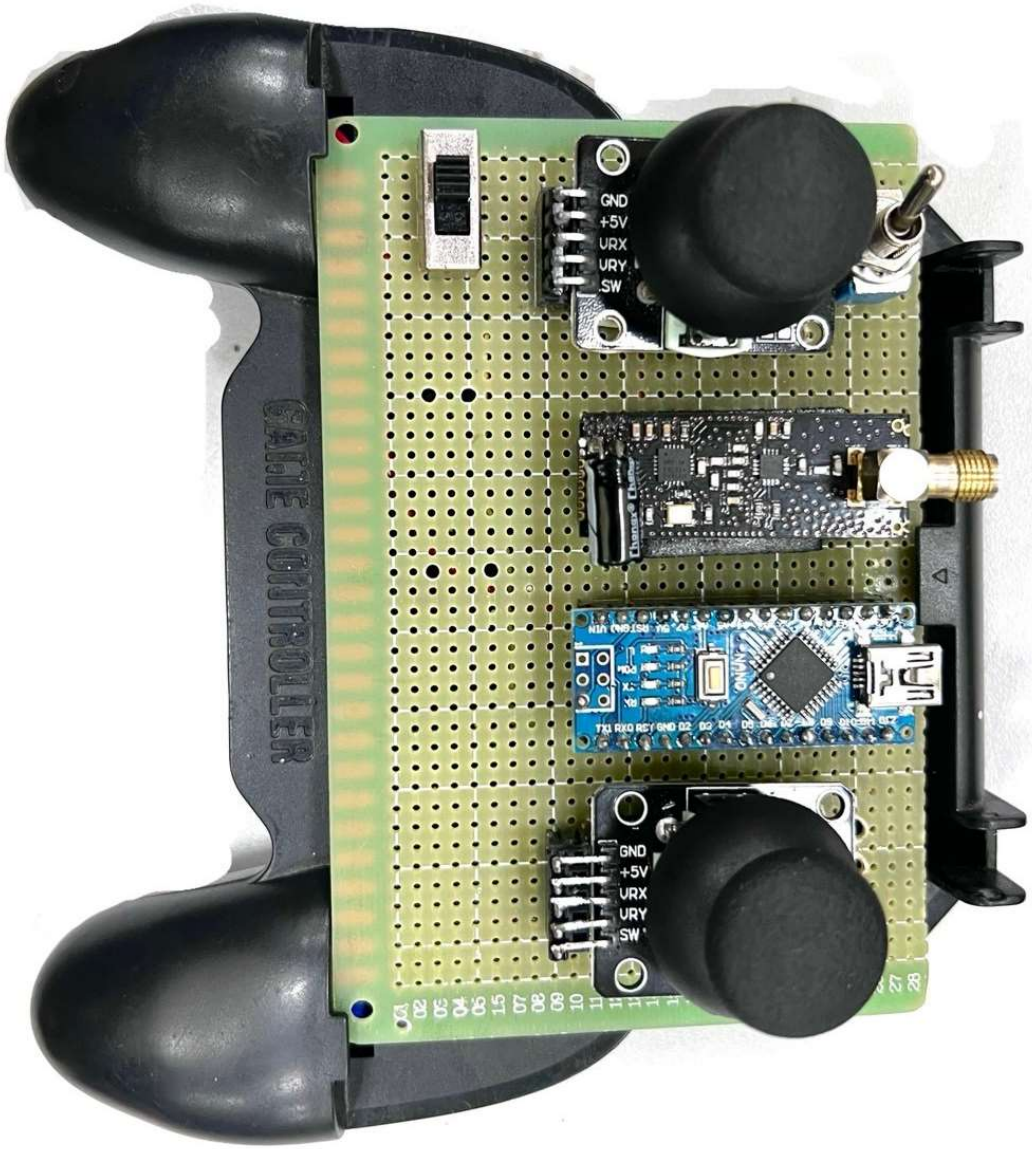
Hình 3.2: Lắp ráp khung Drone



Hình 3.3: Gắn mạch PCB vào khung drone



Hình 3.4: Thiết bị Drone sau khi lắp ghép hoàn chỉnh



Hình 3.5: Tay cầm điều khiển sau khi lắp ráp hoàn chỉnh

3.2. Chương trình điều khiển

Chương trình thực hiện điều khiển drone được viết trên Arduino IDE và sử dụng thuật toán PID cho động cơ như trình bày ở chương 2. Chi tiết chương trình được trình bày chi tiết ở phụ lục 1.

Kết luận

Sau thời gian thực hiện đồ án em đã thực hiện được các nội dung chính gồm:

- Hiểu được những kiến thức tổng quan về thiết bị bay không người lái, đi sâu tìm hiểu thiết bị qua Quadcopter bao gồm các mạch điều khiển, các thành phần cơ khí và thuật toán điều khiển PID cho các động cơ được sử dụng trong đồ án.
- Nghiên cứu tìm hiểu và thiết kế, thực hiện các mạch và lắp đặt thử nghiệm thiết bị drone loại Quadcopter.
- Thực hiện viết chương trình điều khiển cho thiết bị bay đã chế tạo.

Trong thời gian làm đồ án em đã nhận được nhiều sự giúp đỡ, hướng dẫn từ các thầy cô trong Khoa Điện – Điện tử, Trường Đại học Quản lý và Công nghệ Hai Phòng, nhờ những điều đó mà em đã hoàn thành được đồ án của mình. Em xin trân trọng cảm ơn tới tất cả thầy cô và các bạn.

Phụ lục 1

```
#include <Wire.h>
#include <EEPROM.h>

float pid_p_gain_roll = 1.3;
float pid_i_gain_roll = 0.04;
float pid_d_gain_roll = 18.0;
int pid_max_roll = 400;

float pid_p_gain_pitch = pid_p_gain_roll;
float pid_i_gain_pitch = pid_i_gain_roll;
float pid_d_gain_pitch = pid_d_gain_roll;
int pid_max_pitch = pid_max_roll;
float pid_p_gain_yaw = 4.0;
float pid_i_gain_yaw = 0.02;
float pid_d_gain_yaw = 0.0;
int pid_max_yaw = 400;

boolean auto_level = true;

//Khai báo các biến toàn cục

byte last_channel_1, last_channel_2, last_channel_3, last_channel_4;
byte eeprom_data[36];
byte highByte, lowByte;
volatile int receiver_input_channel_1, receiver_input_channel_2,
receiver_input_channel_3, receiver_input_channel_4;
int counter_channel_1, counter_channel_2, counter_channel_3,
counter_channel_4, loop_counter;
int esc_1, esc_2, esc_3, esc_4;
int throttle, battery_voltage;
int cal_int, start, gyro_address;
int receiver_input[5];
int temperature;
int acc_axis[4], gyro_axis[4];
float roll_level_adjust, pitch_level_adjust;

long acc_x, acc_y, acc_z, acc_total_vector;
unsigned long timer_channel_1, timer_channel_2, timer_channel_3,
timer_channel_4, esc_timer, esc_loop_timer;
unsigned long timer_1, timer_2, timer_3, timer_4, current_time;
unsigned long loop_timer;
double gyro_pitch, gyro_roll, gyro_yaw;
```

```
double gyro_axis_cal[4];
float pid_error_temp;
float pid_i_mem_roll, pid_roll_setpoint, gyro_roll_input, pid_output_roll,
pid_last_roll_d_error;
float pid_i_mem_pitch, pid_pitch_setpoint, gyro_pitch_input, pid_output_pitch,
pid_last_pitch_d_error;
float pid_i_mem_yaw, pid_yaw_setpoint, gyro_yaw_input, pid_output_yaw,
pid_last_yaw_d_error;
float angle_roll_acc, angle_pitch_acc, angle_pitch, angle_roll;
boolean gyro_angles_set;

//Quy trình thiết lập
void setup(){
  Serial.begin(57600);
  for(start = 0; start <= 35; start++)eeprom_data[start] = EEPROM.read(start);
  start = 0;                                                  //Đặt điểm bắt đầu về 0.

  gyro_address = eeprom_data[32]; //Lưu trữ địa chỉ con quay trong biến.

  Wire.begin();                           //Khởi động I2C với tư cách là chủ.

  TWBR = 12;                              //Đặt tốc độ xung nhịp I2C thành 400kHz.

  DDRD |= B11110000;        //Định cấu hình kỹ thuật số 4, 5, 6 và 7 làm đầu ra.
  DDRB |= B00110000;          //Định cấu hình kỹ thuật số 12 và 13 làm đầu ra.

  digitalWrite(12,HIGH);        //Bật đèn led cảnh báo.

  while(eeprom_data[33] != 'J' || eeprom_data[34] != 'M' || eeprom_data[35] !=
'B')delay(10);

  //Cần có MPU-6050 với con quay hồi chuyển và gia tốc kế

  if(eeprom_data[31] == 2 || eeprom_data[31] == 3)delay(10);

  set_gyro_registers();                   //Đặt các thanh ghi con quay cụ thể.

  for (cal_int = 0; cal_int < 1250 ; cal_int ++){       //Đợi 5 giây trước khi tiếp tục.
    PORTD |= B11110000;
    delayMicroseconds(1000);
    PORTD &= B00001111;
    delayMicroseconds(3000);
  }
```

```
  //lấy nhiều mẫu dữ liệu con quay hồi chuyển để xác định độ lệch trung bình
của con quay hồi chuyển
  for (cal_int = 0; cal_int < 2000 ; cal_int ++){     //Lấy 2000 bài đọc để hiệu chuẩn.
    if(cal_int % 15 == 0)digitalWrite(12, !digitalRead(12));  //Thay đổi trạng thái đèn
LED để biểu thị hiệu chuẩn.
    gyro_signalen();                          //Đọc đầu ra con quay hồi chuyển.
    gyro_axis_cal[1] += gyro_axis[1];
    gyro_axis_cal[2] += gyro_axis[2];
    gyro_axis_cal[3] += gyro_axis[3];
    //cung cấp xung 1000us trong khi hiệu chỉnh con quay hồi chuyển để esc k phát ra
tiếng bíp.

    PORTD |= B11110000;
    delayMicroseconds(1000);
    PORTD &= B00001111;
    delay(3);
  }

  //Bây giờ có 2000 số đo,-> độ lệch con quay hồi chuyển trung bình.
  gyro_axis_cal[1] /= 2000;
  gyro_axis_cal[2] /= 2000;
  gyro_axis_cal[3] /= 2000;

  PCICR |= (1 << PCIE0);                          //Đặt PCIE0 để bật quét PCMSK0.
  PCMSK0 |= (1 << PCINT0);
  PCMSK0 |= (1 << PCINT1);
  PCMSK0 |= (1 << PCINT2);
  PCMSK0 |= (1 << PCINT3);

  //Đợi cho đến khi bộ thu hoạt động và bướm ga được đặt ở vị trí thấp hơn.

  while(receiver_input_channel_3 < 990 || receiver_input_channel_3 > 1020 ||
receiver_input_channel_4 < 1400){
    receiver_input_channel_3 = convert_receiver_channel(3);
    receiver_input_channel_4 = convert_receiver_channel(4);
    start ++;
    //Cung cấp xung 1000us trong khi chờ đầu vào máy thu để esc k phát ra tiếng bíp.
    PORTD |= B11110000;
    delayMicroseconds(1000);
    PORTD &= B00001111;
    delay(3);
    if(start == 125){
      digitalWrite(12, !digitalRead(12));
```

```
      start = 0;
    }
  }
  start = 0;


  //Nạp điện áp pin vào biến battery_voltage.
  //65 là điện áp bù cho diode.
  //12.6V equals ~5V @ Analog 0.
  //12.6V equals 1023 analogRead(0).
  //1260 / 1023 = 1.2317.
  //battery_voltage giữ 1050 nếu điện áp pin là 10,5V.
  battery_voltage = (analogRead(0) + 65) * 1.2317;

  loop_timer = micros();
  // tắt đèn led.
  digitalWrite(12,LOW);
}

//Vòng lặp chương trình chính

void loop(){

  //65.5 = 1 deg/sec (kiểm tra bảng dữ liệu của MPU-6050 để biết thêm thông tin).
  gyro_roll_input = (gyro_roll_input * 0.7) + ((gyro_roll / 65.5) * 0.3);   //Gyro
pid input is deg/sec.
  gyro_pitch_input = (gyro_pitch_input * 0.7) + ((gyro_pitch / 65.5) *
0.3);//Gyro pid input is deg/sec.
  gyro_yaw_input = (gyro_yaw_input * 0.7) + ((gyro_yaw / 65.5) * 0.3);
//Gyro pid input is deg/sec.

  //Tính toán góc con quay
  //0.0000611 = 1 / (250Hz / 65.5)
  angle_pitch += gyro_pitch * 0.0000611;
  angle_roll += gyro_roll * 0.0000611;

  //0.000001066 = 0.0000611 * (3.142(PI) / 180degr)
  angle_pitch -= angle_roll * sin(gyro_yaw * 0.000001066);
  angle_roll += angle_pitch * sin(gyro_yaw * 0.000001066);

  acc_total_vector = sqrt((acc_x*acc_x)+(acc_y*acc_y)+(acc_z*acc_z));
//Tính tổng vectơ gia tốc.
```

```
  if(abs(acc_y) < acc_total_vector){
    angle_pitch_acc = asin((float)acc_y/acc_total_vector)* 57.296; //Tính góc nghiêng.
  }
  if(abs(acc_x) < acc_total_vector){
    angle_roll_acc = asin((float)acc_x/acc_total_vector)* -57.296; //Tính góc cuộn.
  }

  //Đặt thước thủy MPU-6050 và ghi lại các giá trị ở hai dòng sau để hiệu chuẩn.
  angle_pitch_acc -= 0.0;
  angle_roll_acc -= 0.0;

  angle_pitch = angle_pitch * 0.9996 + angle_pitch_acc * 0.0004;          //Điều
chỉnh độ lệch của góc nghiêng của con quay hồi chuyển bằng góc nghiêng của
gia tốc kế.
  angle_roll = angle_roll * 0.9996 + angle_roll_acc * 0.0004;  //Hiệu chỉnh độ lệch của
góc cuộn con quay bằng góc cuộn gia tốc.
  pitch_level_adjust = angle_pitch * 15;                                  //Tính toán hiệu chỉnh
góc nghiêng
  roll_level_adjust = angle_roll * 15;                                  //Tính toán hiệu
chỉnh góc cuộn

  if(!auto_level){                 //Nếu máy bay bốn cánh không ở chế độ tự động
cấp
    pitch_level_adjust = 0;                 //Đặt hiệu chỉnh góc nghiêng về 0.
    roll_level_adjust = 0;                  //Đặt hiệu chỉnh góc cuộn về 0.
  }


  //Để khởi động động cơ: throttle thấp và yaw trái (step 1).
  if(receiver_input_channel_3 < 1050 && receiver_input_channel_4 <
1050)start = 1;
  //Khi cần điều hướng trở lại vị trí chính giữa, khởi động động cơ (step 2).
  if(start == 1 && receiver_input_channel_3 < 1050 &&
receiver_input_channel_4 > 1450){
    start = 2;

    angle_pitch = angle_pitch_acc;
    angle_roll = angle_roll_acc;
    gyro_angles_set = true;

    //Đặt lại bộ điều khiển PID để khởi đầu suôn sẻ.
    pid_i_mem_roll = 0;
    pid_last_roll_d_error = 0;
```

```
    pid_i_mem_pitch = 0;
    pid_last_pitch_d_error = 0;
    pid_i_mem_yaw = 0;
    pid_last_yaw_d_error = 0;
  }
  //Dừng động cơ: throttle thấp và yaw sang phải.
  if(start == 2 && receiver_input_channel_3 < 1050 &&
receiver_input_channel_4 > 1950)start = 0;

  //Điểm đặt PID tính bằng độ trên giây được xác định bởi đầu vào máy thu
cuộn.
  //Trong trường hợp chia cho 3, tốc độ cuộn tối đa là khoảng 164 độ mỗi giây (
(500-8)/3 = 164d/s ).
  pid_roll_setpoint = 0;
  //Chúng ta cần một dải chết nhỏ 16us để có kết quả tốt hơn.
  if(receiver_input_channel_1 > 1508)pid_roll_setpoint =
receiver_input_channel_1 - 1508;
  else if(receiver_input_channel_1 < 1492)pid_roll_setpoint =
receiver_input_channel_1 - 1492;

  pid_roll_setpoint -= roll_level_adjust;
  pid_roll_setpoint /= 3.0;

  //Điểm đặt PID tính bằng độ trên giây được xác định bởi đầu vào máy thu cao
độ.
  //Trong trường hợp chia cho 3, tốc độ cao độ tối đa là khoảng 164 độ mỗi giây
( (500-8)/3 = 164d/s ).
  pid_pitch_setpoint = 0;
  //Cần một dải chết nhỏ 16us để có kết quả tốt hơn.
  if(receiver_input_channel_2 > 1508)pid_pitch_setpoint =
receiver_input_channel_2 - 1508;
  else if(receiver_input_channel_2 < 1492)pid_pitch_setpoint =
receiver_input_channel_2 - 1492;

  pid_pitch_setpoint -= pitch_level_adjust;
  pid_pitch_setpoint /= 3.0;
  //Điểm đặt PID tính bằng độ trên giây được xác định bởi đầu vào máy thu
lệch.
  //Trong trường hợp chia cho 3, tốc độ yaw tối đa là khoảng 164 độ mỗi giây (
(500-8)/3 = 164d/s ).
  pid_yaw_setpoint = 0;
  //Chúng ta cần một dải chết nhỏ 16us để có kết quả tốt hơn.
  if(receiver_input_channel_3 > 1050){ //Không yaw khi tắt động cơ.
```

```
   if(receiver_input_channel_4 > 1508)pid_yaw_setpoint =
(receiver_input_channel_4 - 1508)/3.0;
   else if(receiver_input_channel_4 < 1492)pid_yaw_setpoint =
(receiver_input_channel_4 - 1492)/3.0;
  }

  calculate_pid();

  //Điện áp pin là cần thiết để bù đắp.
  //Một bộ lọc bổ sung được sử dụng để giảm tiếng ồn.
  //0.09853 = 0.08 * 1.2317.

  battery_voltage = battery_voltage * 0.92 + (analogRead(0) + 65) * 0.09853;

  //Bật đèn led nếu điện áp pin yếu.

  if(battery_voltage < 1000 && battery_voltage > 600)digitalWrite(12, HIGH);


  throttle = receiver_input_channel_3;
  if (start == 2){                                        //Các động cơ được khởi
động.
    if (throttle > 1800) throttle = 1800;
    esc_1 = throttle - pid_output_pitch + pid_output_roll - pid_output_yaw;
//Tính xung cho esc 1 (front-right - CCW)
    esc_2 = throttle + pid_output_pitch + pid_output_roll + pid_output_yaw;
//Tính xung cho esc 2 (rear-right - CW)
    esc_3 = throttle + pid_output_pitch - pid_output_roll - pid_output_yaw;
//Tính xung cho esc 3 (rear-left - CCW)
    esc_4 = throttle - pid_output_pitch - pid_output_roll + pid_output_yaw;
//Tính xung cho esc 4 (front-left - CW)

    if (battery_voltage < 1240 && battery_voltage > 800){                //Pin đã
được kết nối chưa?
      esc_1 += esc_1 * ((1240 - battery_voltage)/(float)3500);           //Bù xung
esc-1 khi sụt áp.
      esc_2 += esc_2 * ((1240 - battery_voltage)/(float)3500);           //Bù xung
esc-2 khi sụt áp.
      esc_3 += esc_3 * ((1240 - battery_voltage)/(float)3500);           //Bù xung
esc-3 khi sụt áp.
      esc_4 += esc_4 * ((1240 - battery_voltage)/(float)3500);           //Bù xung
esc-4 khi sụt áp.
    }
```

```
    if (esc_1 < 1100) esc_1 = 1100;                    //Giữ cho động cơ chạy.
    if (esc_2 < 1100) esc_2 = 1100;                    //Giữ cho động cơ chạy.
    if (esc_3 < 1100) esc_3 = 1100;                    //Giữ cho động cơ chạy.
    if (esc_4 < 1100) esc_4 = 1100;                    //Giữ cho động cơ chạy.

    if(esc_1 > 2000)esc_1 = 2000;              //Giới hạn xung esc-1 ở mức
  2000us.
    if(esc_2 > 2000)esc_2 = 2000;              //Giới hạn xung esc-2 ở mức
  2000us.
    if(esc_3 > 2000)esc_3 = 2000;              //Giới hạn xung esc-3 ở mức
  2000us.
    if(esc_4 > 2000)esc_4 = 2000;              //Giới hạn xung esc-4 ở mức
  2000us.
   }

   else{
    esc_1 = 1000;      //Nếu bắt đầu không phải là 2, hãy giữ xung 1000us cho
  ess-1.
    esc_2 = 1000;      //Nếu bắt đầu không phải là 2, hãy giữ xung 1000us cho
  ess-2.
    esc_3 = 1000;      //Nếu bắt đầu không phải là 2, hãy giữ xung 1000us cho
  ess-3.
    esc_4 = 1000;      //Nếu bắt đầu không phải là 2, hãy giữ xung 1000us cho
  ess-4.
   }

   if(micros() - loop_timer > 4050)digitalWrite(12, HIGH);                //Bật đèn
  LED nếu thời gian vòng lặp vượt quá 4050us.

//Tốc độ làm mới là 250Hz. Điều đó có nghĩa là esc cần có xung cứ sau 4ms.
   while(micros() - loop_timer < 4000);                                   //Đợi cho đến khi
  4000us được thông qua.
   loop_timer = micros();                                                 //Đặt bộ đếm thời gian
  cho vòng lặp tiếp theo.

   PORTD |= B11110000;
  //Tính thời gian của cạnh yếu của xung esc.
   timer_channel_1 = esc_1 + loop_timer;
   timer_channel_2 = esc_2 + loop_timer;
   timer_channel_3 = esc_3 + loop_timer;
   timer_channel_4 = esc_4 + loop_timer;
```

```
  gyro_signalen();

  while(PORTD >= 16){
    esc_loop_timer = micros();                    //Đọc thời gian hiện tại.
    if(timer_channel_1 <= esc_loop_timer)PORTD &= B11101111;
    if(timer_channel_2 <= esc_loop_timer)PORTD &= B11011111;
    if(timer_channel_3 <= esc_loop_timer)PORTD &= B10111111;
    if(timer_channel_4 <= esc_loop_timer)PORTD &= B01111111;
  }
}

ISR(PCINT0_vect){
  current_time = micros();
  //Channel 1=========================================
  if(PINB & B00000001){                 //Đầu vào 8 có cao không
    if(last_channel_1 == 0){            //Đầu vào 8 đã thay đổi từ 0 thành 1.
      last_channel_1 = 1;               //Ghi nhớ trạng thái đầu vào hiện tại.
      timer_1 = current_time;           //Đặt timer_1 thành current_time.
    }
  }
  else if(last_channel_1 == 1){       //Đầu vào 8 không cao và thay đổi từ 1 thành 0.
    last_channel_1 = 0;                         //Ghi nhớ trạng thái đầu vào hiện tại.
    receiver_input[1] = current_time - timer_1; //Kênh 1 là current_time - timer_1.
  }
  //Channel 2=========================================
  if(PINB & B00000010 ){
    if(last_channel_2 == 0){                                    .
      last_channel_2 = 1;
      timer_2 = current_time;                                     }
  }
  else if(last_channel_2 == 1){
    last_channel_2 = 0;
    receiver_input[2] = current_time - timer_2;
  }
  //Channel 3=========================================
  if(PINB & B00000100 ){
    if(last_channel_3 == 0){
      last_channel_3 = 1;
      timer_3 = current_time;
    }
  }
  else if(last_channel_3 == 1){
    last_channel_3 = 0;
```

```
    receiver_input[3] = current_time - timer_3;

  }
  //Channel 4=========================================
  if(PINB & B00001000 ){
    if(last_channel_4 == 0){
      last_channel_4 = 1;
      timer_4 = current_time;
    }
  }
  else if(last_channel_4 == 1){
    last_channel_4 = 0;
    receiver_input[4] = current_time - timer_4;
  }
}

//Chương trình con để đọc con quay hồi chuyển

void gyro_signalen(){
  //Đọc MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);
    Wire.write(0x3B);
    Wire.endTransmission();
    Wire.requestFrom(gyro_address,14);

    receiver_input_channel_1 = convert_receiver_channel(1);
    receiver_input_channel_2 = convert_receiver_channel(2);
    receiver_input_channel_3 = convert_receiver_channel(3);
    receiver_input_channel_4 = convert_receiver_channel(4);

    while(Wire.available() < 14);
    acc_axis[1] = Wire.read()<<8|Wire.read();
    acc_axis[2] = Wire.read()<<8|Wire.read();
    acc_axis[3] = Wire.read()<<8|Wire.read();
    temperature = Wire.read()<<8|Wire.read();
    gyro_axis[1] = Wire.read()<<8|Wire.read();
    gyro_axis[2] = Wire.read()<<8|Wire.read();
    gyro_axis[3] = Wire.read()<<8|Wire.read();
  }

  if(cal_int == 2000){
    gyro_axis[1] -= gyro_axis_cal[1];
```

```
    gyro_axis[2] -= gyro_axis_cal[2];
    gyro_axis[3] -= gyro_axis_cal[3];
  }
  gyro_roll = gyro_axis[eeprom_data[28] & 0b00000011];
  if(eeprom_data[28] & 0b10000000)gyro_roll *= -1;
  gyro_pitch = gyro_axis[eeprom_data[29] & 0b00000011];
  if(eeprom_data[29] & 0b10000000)gyro_pitch *= -1;
  gyro_yaw = gyro_axis[eeprom_data[30] & 0b00000011];
  if(eeprom_data[30] & 0b10000000)gyro_yaw *= -1;

  acc_x = acc_axis[eeprom_data[29] & 0b00000011];
  if(eeprom_data[29] & 0b10000000)acc_x *= -1;
  acc_y = acc_axis[eeprom_data[28] & 0b00000011];
  if(eeprom_data[28] & 0b10000000)acc_y *= -1;
  acc_z = acc_axis[eeprom_data[30] & 0b00000011];
  if(eeprom_data[30] & 0b10000000)acc_z *= -1;
}

//Chương trình con tính toán đầu ra pid

void calculate_pid(){
  //Tính toán cuộn
  pid_error_temp = gyro_roll_input - pid_roll_setpoint;
  pid_i_mem_roll += pid_i_gain_roll * pid_error_temp;
  if(pid_i_mem_roll > pid_max_roll)pid_i_mem_roll = pid_max_roll;
  else if(pid_i_mem_roll < pid_max_roll * -1)pid_i_mem_roll = pid_max_roll * -1;

  pid_output_roll = pid_p_gain_roll * pid_error_temp + pid_i_mem_roll +
pid_d_gain_roll * (pid_error_temp - pid_last_roll_d_error);
  if(pid_output_roll > pid_max_roll)pid_output_roll = pid_max_roll;
  else if(pid_output_roll < pid_max_roll * -1)pid_output_roll = pid_max_roll * -1;

  pid_last_roll_d_error = pid_error_temp;

  //Tính toán cao độ
  pid_error_temp = gyro_pitch_input - pid_pitch_setpoint;
  pid_i_mem_pitch += pid_i_gain_pitch * pid_error_temp;
  if(pid_i_mem_pitch > pid_max_pitch)pid_i_mem_pitch = pid_max_pitch;
  else if(pid_i_mem_pitch < pid_max_pitch * -1)pid_i_mem_pitch = pid_max_pitch * -
1;

  pid_output_pitch = pid_p_gain_pitch * pid_error_temp + pid_i_mem_pitch +
pid_d_gain_pitch * (pid_error_temp - pid_last_pitch_d_error);
```

```
  if(pid_output_pitch > pid_max_pitch)pid_output_pitch = pid_max_pitch;
  else if(pid_output_pitch < pid_max_pitch * -1)pid_output_pitch =
pid_max_pitch * -1;

  pid_last_pitch_d_error = pid_error_temp;

  //Yaw calculations
  pid_error_temp = gyro_yaw_input - pid_yaw_setpoint;
  pid_i_mem_yaw += pid_i_gain_yaw * pid_error_temp;
  if(pid_i_mem_yaw > pid_max_yaw)pid_i_mem_yaw = pid_max_yaw;
  else if(pid_i_mem_yaw < pid_max_yaw * -1)pid_i_mem_yaw =
pid_max_yaw * -1;

  pid_output_yaw = pid_p_gain_yaw * pid_error_temp + pid_i_mem_yaw +
pid_d_gain_yaw * (pid_error_temp - pid_last_yaw_d_error);
  if(pid_output_yaw > pid_max_yaw)pid_output_yaw = pid_max_yaw;
  else if(pid_output_yaw < pid_max_yaw * -1)pid_output_yaw = pid_max_yaw
* -1;

  pid_last_yaw_d_error = pid_error_temp;
}

//chuyển đổi tín hiệu thu thực tế thành giá trị tiêu chuẩn
//Dữ liệu được lưu trữ trong EEPROM được sử dụng.

int convert_receiver_channel(byte function){
  byte channel, reverse;
  int low, center, high, actual;
  int difference;

  channel = eeprom_data[function + 23] & 0b00000111;
  if(eeprom_data[function + 23] & 0b10000000)reverse = 1;
  else reverse = 0;

  actual = receiver_input[channel];
  low = (eeprom_data[channel * 2 + 15] << 8) | eeprom_data[channel * 2 + 14];
center = (eeprom_data[channel * 2 - 1] << 8) | eeprom_data[channel * 2 - 2];
  high = (eeprom_data[channel * 2 + 7] << 8) | eeprom_data[channel * 2 + 6];
  if(actual < center){
    if(actual < low)actual = low;
    difference = ((long)(center - actual) * (long)500) / (center - low);
    if(reverse == 1)return 1500 + difference;
    else return 1500 - difference;
```

```
  }
  else if(actual > center){
    if(actual > high)actual = high;
    difference = ((long)(actual - center) * (long)500) / (high - center);
    if(reverse == 1)return 1500 - difference;
    else return 1500 + difference;
  }
  else return 1500;
}

void set_gyro_registers(){
  // MPU-6050
  if(eeprom_data[31] == 1){
    Wire.beginTransmission(gyro_address);
    Wire.write(0x6B);
    Wire.write(0x00);
    Wire.endTransmission();
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1B);
    Wire.write(0x08);
    Wire.endTransmission();
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1C);
    Wire.write(0x10);
    Wire.endTransmission();
    //kiểm tra ngẫu nhiên
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1B);
    Wire.endTransmission();
    Wire.requestFrom(gyro_address, 1);
    while(Wire.available() < 1);
    if(Wire.read() != 0x08){
      digitalWrite(12,HIGH);
      while(1)delay(10);
    }
    Wire.beginTransmission(gyro_address);
    Wire.write(0x1A);
    Wire.write(0x03);
    Wire.endTransmission();
  }
}
```

Tài liệu tham khảo

1. Arduino.cc
2. Quách Tuấn Ngọc(2003). Ngôn Ngữ Lập Trình C, Nhà xuất bản Thống Kê.
3. http://www.brokking.net
4. https://github.com/parakhm95/Custom-Quadrotor-Drone