BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH
KHOA CƠ KHÍ CHẾ TẠO MÁY

HCMUTE

DỰ ÁN NGHIÊN CỨU

NGHIÊN CỨU, THIẾT KẾ, CHẾ TẠO MÔ HÌNH VÀ HỆ THỐNG ĐIỀU KHIỂN CÂN BẰNG QUADCOPTER DÒ ĐƯỜNG TỰ ĐỘNG

SVTH:

LÝ MINH NHỰT	**20146162**
CAO TẤN PHI	**20146513**
DƯƠNG HIỂN PHÚC	**20146519**

KHÓA: 2020 - 2024
NGÀNH: CÔNG NGHỆ KỸ THUẬT CƠ ĐIỆN TỬ
GVHD: TS. HUỲNH QUANG DUY

Thành phố Hồ Chí Minh, tháng 07 năm 2024

LỜI CẢM ƠN

Để hoàn thành dự án với đề tài "*Nghiên cứu, thiết kế, chế tạo mô hình và hệ thống điều khiển cân bằng quadcopter hỗ trợ dò đường tự động*", chúng em xin gửi lời cảm ơn chân thành đến tất cả những người đã hỗ trợ, động viên và giúp đỡ chúng em trong suốt quá trình thực hiện đồ án này.

Trước hết, chúng em xin gửi lời cảm ơn sâu sắc đến Tiến sĩ Huỳnh Quang Duy, giảng viên hướng dẫn, đã tận tình chỉ dẫn, cung cấp kiến thức và kinh nghiệm quý báu trong suốt thời gian nghiên cứu và thực hiện dự án. Sự hỗ trợ và khích lệ của thầy là nguồn động lực lớn lao giúp chúng em vượt qua những khó khăn và hoàn thành công việc này.

Chúng em cũng xin gửi lời cảm ơn đến Ban Giám hiệu, các thầy cô trong Khoa Cơ khí Chế tạo máy, Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh, đã truyền đạt kiến thức, tạo điều kiện thuận lợi cho chúng em trong suốt quá trình học tập và nghiên cứu. Những kiến thức và kỹ năng mà các thầy cô đã truyền đạt là nền tảng vững chắc để chúng em thực hiện và hoàn thành dự án này.

Đặc biệt, chúng em xin cảm ơn gia đình và bạn bè, những người luôn bên cạnh ủng hộ, động viên và chia sẻ những khó khăn trong suốt thời gian qua. Sự quan tâm và động viên của gia đình và bạn bè đã tiếp thêm sức mạnh và quyết tâm giúp chúng em hoàn thành tốt công việc.

Chúng em cũng xin cảm ơn các bạn sinh viên trong nhóm đã hợp tác, làm việc chăm chỉ và chia sẻ những ý tưởng sáng tạo. Sự đoàn kết và hỗ trợ lẫn nhau trong nhóm là yếu tố quan trọng giúp chúng em vượt qua những thách thức và đạt được kết quả như mong đợi.

Cuối cùng, chúng em xin chân thành cảm ơn tất cả những ai đã trực tiếp hoặc gián tiếp giúp đỡ chúng em trong suốt quá trình thực hiện dự án này. Chúng em hy vọng rằng kết quả nghiên cứu của mình sẽ góp phần vào sự phát triển của lĩnh vực cơ điện tử và công nghệ điều khiển tự động.

Trân trọng!

TÓM TẮT

Dự án với đề tài "*Nghiên cứu, thiết kế, chế tạo mô hình và hệ thống điều khiển cân bằng quadcopter hỗ trợ dò đường tự động*" được thực hiện bởi nhóm chúng em với mục tiêu là phát triển một hệ thống quadcopter điều khiển cân bằng được tích hợp công nghệ Lidar hỗ trợ quét bản đồ và dò đường tự động, cùng hệ các cảm biến siêu âm hỗ trợ né vật cản.

Hệ thống cân bằng của quadcopter sử dụng mạch điều khiển cân bằng NAZA của DJI, kết hợp với ESC (Electronic Speed Controller) Hobbywing XRotor để điều khiển các động cơ. Sự kết hợp này đảm bảo cho quadcopter duy trì được sự ổn định trong suốt quá trình bay, ngay cả khi gặp phải điều kiện môi trường thay đổi hay các tác động bất ngờ.

Trong quá trình thực hiện, quadcopter đã được tích hợp hệ thống quét bản đồ lidar để thu thập thông tin về môi trường xung quanh. Dữ liệu từ lidar giúp xây dựng bản đồ môi trường thực tế, từ đó trả về hệ thống dữ liệu môi trường bay hiện tại dưới dạng bản đồ.

Quadcopter được tích hợp thêm hệ các cảm biến siêu âm với nhiệm vụ phát hiện vật cản xung quanh môi trường bay. Nhờ đó, dữ liệu được trả về để quadcopter có hành trình hợp lý và tránh các chướng ngại trong quá trình bay.

Quá trình thử nghiệm cho thấy quadcopter có thể tự động né tránh các vật cản được quét trong bản đồ lidar, thể hiện tính linh hoạt và khả năng thích ứng cao. Hệ thống tự động bay của quadcopter cho phép nó thực hiện các nhiệm vụ bay định hướng mà không cần sự can thiệp trực tiếp từ người điều khiển, góp phần nâng cao hiệu quả và an toàn trong các ứng dụng thực tế.

Kết quả của đề tài này đã chứng minh tính khả thi của việc tích hợp các công nghệ hiện đại như lidar vào quadcopter, mở ra nhiều triển vọng phát triển trong lĩnh vực công nghệ tự động hóa.

ABSTRACT

The project titled "Research, Design, and Fabrication of a Quadcopter Model and Control System for Automatic Pathfinding" was conducted by our group with the aim of developing a balanced quadcopter system integrated with Lidar technology for map scanning and automatic pathfinding, along with ultrasonic sensors to assist in obstacle avoidance.

The balancing system of the quadcopter uses the DJI NAZA balancing control circuit, combined with the Hobbywing XRotor Electronic Speed Controller (ESC) to control the motors. This combination ensures the quadcopter maintains stability throughout the flight, even when encountering changing environmental conditions or unexpected impacts.

During the implementation, the quadcopter was integrated with a Lidar mapping system to collect information about the surrounding environment. Data from the Lidar helps build a real-time environmental map, providing current flight environment data in the form of a map.

The quadcopter is also integrated with a set of ultrasonic sensors tasked with detecting obstacles in the flight environment. As a result, the data is returned to allow the quadcopter to navigate a reasonable path and avoid obstacles during the flight.

Testing has shown that the quadcopter can automatically avoid obstacles detected in the Lidar map, demonstrating high flexibility and adaptability. The quadcopter's automatic flight system allows it to perform directional flight tasks without direct intervention from the operator, contributing to increased efficiency and safety in practical applications.

The results of this project have demonstrated the feasibility of integrating modern technologies such as Lidar into quadcopters, opening up many prospects for development in the field of automation technology.

MỤC LỤC

DANH MỤC CÁC TỪ VIẾT TẮT

LIDAR:	Light Detection and Ranging
GPS:	Global Positioning System
DOF:	Degree of Freedom
PID:	Proportional Integral Derivative
OSRF:	Open Source Robotics Foundation
ROS:	Robot Operating System
SLAM:	Simultaneous Localization and Mapping
HTML:	Hyper Text Markup Language
CSS:	Cascading Style Sheets
CI/CD:	Continuous Integration/Continuous Deployment
API:	Application Programming Interface
URL:	Uniform Resource Locator
SQL:	Structured Query Language
GUI:	Graphical User Interface
OPENCV:	Open Source Computer Vision Library
HOG:	Histogram of Oriented Gradients
SVM:	Support Vector Machines
MRI:	Magnetic Resonance Imaging
CSV:	Comma Separated Values
TXT:	Text file

DANH MỤC CÁC BẢNG BIỂU

DANH MỤC CÁC HÌNH ẢNH

CHƯƠNG 1: TỔNG QUAN

1.1 Giới thiệu

Quadcopter, hay còn được gọi là drone, là một loại thiết bị bay không người lái được cấu tạo với bốn cánh quạt, mang lại khả năng di chuyển linh hoạt và ổn định trong không gian. Được thiết kế dựa trên nguyên lý của máy bay không người lái, quadcopter đã trở thành một công nghệ độc đáo, mang lại nhiều lợi ích và ứng dụng đa dạng.

Lợi ích lớn nhất của quadcopter nằm ở khả năng thực hiện các nhiệm vụ mà con người gặp khó khăn hoặc nguy hiểm. Việc sử dụng quadcopter trong lĩnh vực quân sự và an ninh là một trong những ứng dụng phổ biến đầu tiên. Chúng có thể thực hiện nhiệm vụ giám sát, do thám, và theo dõi khu vực một cách hiệu quả, giảm rủi ro cho nhân viên và cung cấp thông tin chính xác.

Một số ưu điểm nổi bật của quadcopter bao gồm khả năng quay video và chụp ảnh từ góc độ không thể đạt được khi tích hợp với các thiết bị ghi hình, chụp ảnh. Điều này đã mở ra một loạt các ứng dụng trong lĩnh vực truyền thông và giải trí, cũng như trong lĩnh vực nông nghiệp, nơi quadcopter có thể được sử dụng để giám sát đất đai, theo dõi mức nước, và đánh giá tình trạng cây trồng.

Tình hình phát triển của quadcopter hiện nay là sự kết hợp giữa việc giảm kích thước, tăng cường khả năng cảm biến, và cải thiện hiệu suất pin. Điều này đã làm tăng tính ứng dụng và sự linh hoạt của chúng trong nhiều lĩnh vực. Các công ty công nghệ hàng đầu đang đầu tư nhiều vào nghiên cứu và phát triển quadcopter để nâng cao hiệu suất và tính tiện ích. Với các lý do trên, chúng tôi quyết định chọn quadcopter là đối tượng chính của dự án này với mục tiêu xây dựng mô hình và hệ thống điều khiển cân bằng quadcopter dò đường tự động.

Hình 1.1: Drone của hãng Hubsan

Quadcopter được tích hợp với Lidar, một công nghệ đo khoảng cách bằng cách sử dụng tia laser, để tạo ra bản đồ tự động và thực hiện nhiệm vụ dò đường. Ứng dụng của quadcopter kết hợp với Lidar trong lĩnh vực địa kỹ thuật không gian là không ngừng mở rộng. Chúng có thể được sử dụng để tự động hóa quy trình vẽ bản đồ địa hình, xác định đường di chuyển an toàn, và thậm chí hỗ trợ trong công tác cứu thương và phục hồi sau thảm họa tự nhiên. Trong đề tài này, một mô hình quadcopter tích hợp với lidar được nghiên

cứu, thiết kế và chế tạo có thể dò đường và vẽ bản đồ tự động bằng cách ứng dụng các chức năng sẵn có của lidar.

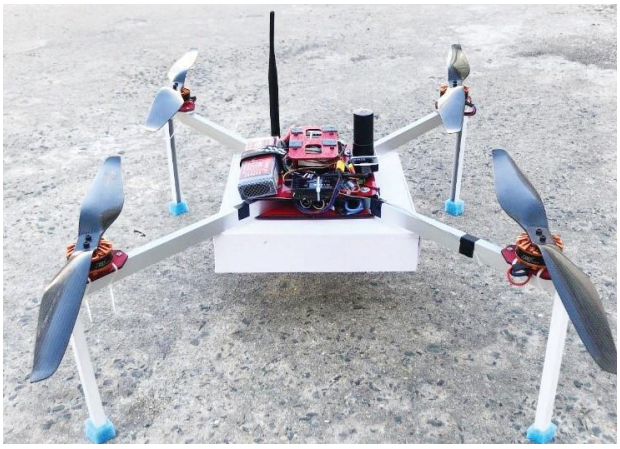
1.2 .Tổng quan

Trên toàn thế giới, nghiên cứu về quadcopter đang ngày càng trở nên phổ biến và quan trọng trong nhiều lĩnh vực. Các tổ chức nghiên cứu và trường đại học hàng đầu đã tiến hành nhiều dự án để khám phá tiềm năng của quadcopter và cải thiện khả năng ứng dụng của chúng. Các đội ngũ nghiên cứu tại các trung tâm nghiên cứu công nghệ hàng đầu ở Mỹ, Châu Âu, và châu Á đều tập trung vào phát triển các công nghệ mới, cảm biến cũng như các thuật toán điều khiển để tối ưu hóa hiệu suất của quadcopter.

Ở Việt Nam, lĩnh vực nghiên cứu về quadcopter cũng đang có những bước tiến tích cực. Nhiều trường đại học và các tổ chức nghiên cứu đã đưa ra những dự án nghiên cứu với mục tiêu ứng dụng quadcopter vào các lĩnh vực như nông nghiệp thông minh, quản lý môi trường, và giáo dục. Các nhóm nghiên cứu tại các trường đại học lớn ở Việt Nam đang tập trung vào việc phát triển quadcopter có khả năng chịu được điều kiện thời tiết khắc nghiệt, giảm chi phí sản xuất và nâng cao khả năng tự động hóa.

Ngoài ra, cộng đồng nghiên cứu và doanh nghiệp ở Việt Nam cũng đang hợp tác để tận dụng tiềm năng của quadcopter. Việc này không chỉ giúp thúc đẩy sự phát triển của ngành công nghiệp chế tạo máy bay không người lái mà còn mở ra những cơ hội mới trong lĩnh vực đào tạo và phát triển nhân sự chuyên gia về quadcopter.

Tổng quan, việc nghiên cứu quadcopter trên toàn thế giới và tại Việt Nam đều đang chứng kiến sự đầu tư đáng kể. Sự hợp tác quốc tế và cũng như nỗ lực nội địa đang giúp thúc đẩy sự tiến bộ trong công nghệ này, mở ra những triển vọng mới cho ứng dụng quadcopter trong nhiều lĩnh vực quan trọng.



Hình 1.2: Kết quả công trình nghiên cứu Drone của sinh viên trường Đại học Công nghệ - ĐHQGHN

1.3 Mục đích đề tài

Nghiên cứu, thiết kế, chế tạo mô hình và hệ thống điều khiển cân bằng quadcopter dò đường tự động.

1.4 Mục tiêu đề tài

- Ứng dụng mô hình cơ khí và thiết kế hệ thống mạch điện cho quadcopter.
- Mô phỏng hoạt động Lidar.
- Ứng dụng hệ thống quét và tạo bản đồ.
- Xây dựng mô hình quadcopter.
- Thiết kế hệ thống né vật cản sử dụng cảm biến siêu âm.
- Xây dựng hệ thống bay tự động sử dụng GPS.
- Điều khiển bay quadcopter thực hiện vẽ bản đồ, né chướng ngại vật.
- Hoàn thành đề tài đúng tiến độ đề ra.

1.5 Đối tượng nghiên cứu

- Sử dụng hệ thống điều khiển cân bằng quadcopter NAZA.
- Phương pháp né vật cản từ cảm biến siêu âm.
- Phương pháp quét bản đồ sử dụng công nghệ từ Lidar.

1.6 Phạm vi nghiên cứu

- Nghiên cứu ứng dụng hệ thống cân bằng NAZA và thiết kế hệ thống dò đường né vật cản.
- Chế tạo phương pháp điều khiển bay quadcopter bằng nút nhấn trên laptop.
- Hệ thống dò đường ứng dụng lidar được giới hạn trong phạm vi 30$\mathbf{m^2}$.
- Vật liệu và thiết bị cơ bản đáp ứng đủ yêu cầu tối thiểu của đề tài.

1.7 Nội dung nghiên cứu

Trong bài báo cáo này nhóm sẽ chia làm 8 chương chính:

Chương 1: Tổng quan – Sẽ trình bày ngắn gọn tổng quát về hệ thống, nói rõ về mục tiêu đề tài, đối tượng và phạm vi nghiên cứu đề tài.

Chương 2: Cơ sở lý thuyết – Chương này sẽ trình bày sơ lược về nguyên lý hoạt động của hệ thống, những khái niệm, lý thuyết liên quan đến quadcopter. Một số lý thuyết cơ bản về Gazebo và Lidar.

Chương 3: Mô phỏng khảo sát – Chương này sẽ trình bày qui trình tiến hành mô phỏng hoạt động của Lidar.

Chương 4: Hệ thống cơ khí, mạch điện – Chương này trình bày về việc lựa chọn thiết bị cũng như trình tự kết nối, hệ thống mạch điện.

Chương 5: Hệ thống điều khiển – Trong chương này có các nội dung được trình bay như sơ lược về kiến trúc hệ thống và cấu trúc của bộ điều khiển NAZA.

Chương 6: Hệ thống dò đường tự động – Đây là chương trình bày hệ thống né vật cản, tự động bay và vẽ bản đồ bằng Lidar

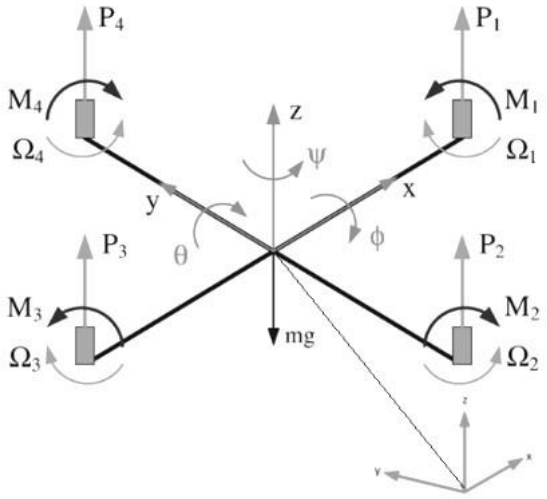
Chương 7: Kết quả – Chương này sẽ trình bày các kết quả mà nhóm thu được trong quá trình thực hiện đề tài.

Chương 8: Kết luận và hướng phát triển – Kết luận quá trình nghiên cứu và hướng phát triển tiếp theo cho đề tài.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1 Tổng quan hệ thống

Quadcopter cơ bản được xây dựng với một cấu trúc vững chắc, bao gồm bốn cánh quạt độc lập, mỗi cánh quạt đều cố định. Trong số bốn cánh quạt này, hai cánh quạt (M1, M3) quay theo chiều kim đồng hồ, và hai cánh quạt còn lại (M2, M4) quay theo chiều ngược chiều kim đồng hồ. Hệ thống điều khiển của quadcopter được thực hiện thông qua việc điều chỉnh tốc độ góc của từng cánh quạt $\boldsymbol{\Omega_i}$, (i = 1, 2, 3, 4)). Chuyển động xoay của quadcopter theo các trục x, y, z có thể được mô tả bằng các góc ϕ (roll), góc θ (pitch), góc ψ (yaw). Trên quadcopter, có khả năng thay đổi lực nâng từ động cơ, cũng như tạo moment xoay góc ϕ quanh trục x, góc θ quanh trục y do sự chênh lệch lực nâng từ bốn động cơ, và moment xoay góc ψ quanh trục z. Điều này giúp duy trì sự ổn định và kiểm soát quadcopter trong không gian, đồng thời đảm bảo cân bằng giữa lực và moment được thay đổi. Hình 2.1 miêu tả các thông tin và cấu hình cơ bản của một quadcopter trong hệ trục tọa độ tham chiếu.



Hình 2.1: Cấu hình quadcopter với các hệ trục tọa độ

2.2 Chuyển động của quadcopter

2.2.1 Lực nâng và lực đẩy của quadcopter

Lực nâng và lực đẩy đóng vai trò quan trọng trong việc kiểm soát và duy trì chuyển động của quadcopter trong không gian.

Lực Nâng (Lực Trọng Lực):

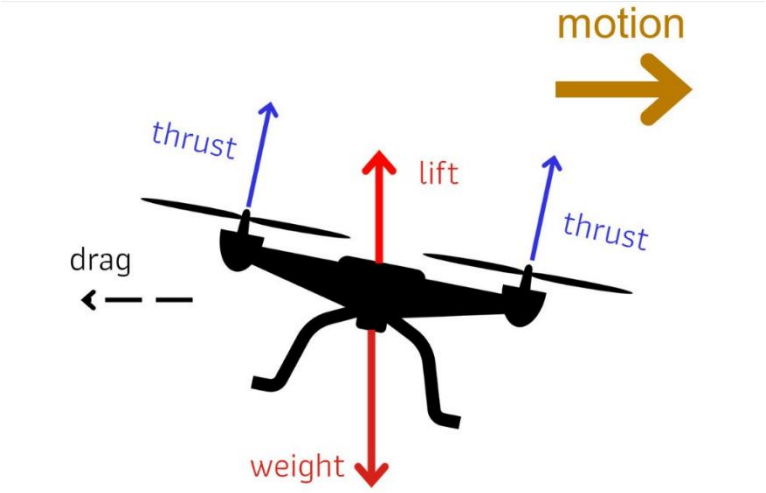
Lực nâng là lực đối với quadcopter được tạo ra bởi cánh quạt khi chúng tạo ra dòng không khí xuống. Theo định luật Newton III, khi một vật tác dụng lực lên vật thể thứ hai, vật thứ hai sẽ tác dụng một lực cùng độ lớn và ngược chiều về phía vật thứ nhất. Do đó, khi cánh quạt tạo ra lực nâng bằng cách đẩy dòng không khí xuống, quadcopter cảm nhận được một lực nâng lên từ dòng không khí. Điều này giúp quadcopter vượt lên trên trọng lực của

nó và có khả năng bay.

Lực Đẩy:

Lực đẩy là lực đối với quadcopter được tạo ra bởi động cơ khi chúng tạo ra dòng không khí chảy ra khỏi cánh quạt. Lực đẩy này được tạo ra theo nguyên lý hành động-đối hành động, tức là quadcopter đẩy dòng không khí ra khỏi mình và nhận được một lực đẩy theo hướng ngược lại. Lực đẩy này giúp quadcopter di chuyển và thực hiện các chuyển động trong không gian, như tăng độ cao, hành trình ngang, và quay vòng.

Đối với quadcopter, sự cân bằng giữa lực nâng và lực đẩy rất quan trọng để duy trì ổn định trong không gian. Điều này thường được kiểm soát thông qua điều chỉnh tốc độ quay của các cánh quạt để tối ưu hóa lực nâng và lực đẩy. Thông qua hệ thống điều khiển, quadcopter có thể điều chỉnh lực nâng và lực đẩy để thực hiện các chuyển động và giữ được vị trí ổn định trong môi trường 3D. Hình 2.2 mô tả vai trò liên quan của lực nâng và lực đẩy trong hướng chỉ định của quadcopter.



Hình 2.2: Lực nâng và lực đẩy trong quadcopter

2.2.2 Các chuyển động góc (Roll, Pitch, Yaw)

a. Các góc Roll, Pitch, Yaw

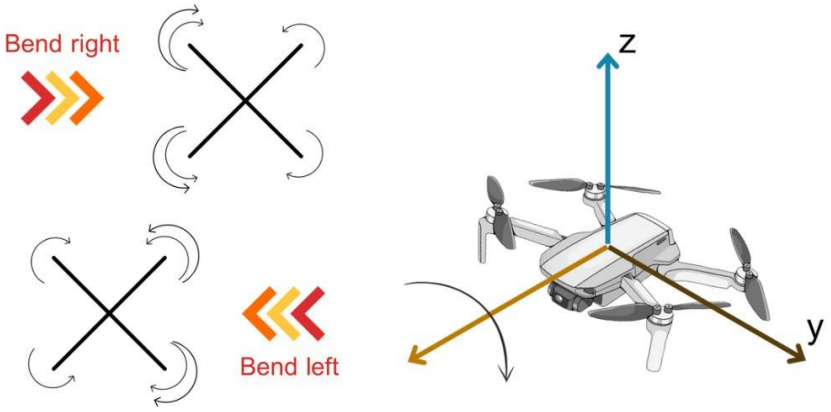
Các chuyển động góc, bao gồm góc Roll (Lăn), góc Pitch (Nhào), và góc Yaw (Quay), là những khái niệm quan trọng giúp quadcopter thực hiện các phương diện chuyển động và kiểm soát hướng di chuyển của mình:

Góc Roll:

Góc roll là góc quay của quadcopter quanh trục dọc thân máy bay. Góc roll được đo bằng độ, với 0 độ là vị trí cân bằng, 90 độ là vị trí nghiêng hết về một bên và -90 độ là vị trí nghiêng hết về phía bên kia.

Góc roll được điều khiển bằng cách thay đổi lực đẩy của hai cánh quạt phía trái và phía

phải. Khi tăng lực đẩy của hai cánh quạt phía trái, quadcopter sẽ nghiêng về phía phải và tăng góc roll. Khi giảm lực đẩy của hai cánh quạt phía trái, quadcopter sẽ nghiêng về phía trái và giảm góc roll. Hình 2.3 mô tả góc roll cho quadcopter qua trái và qua phải.

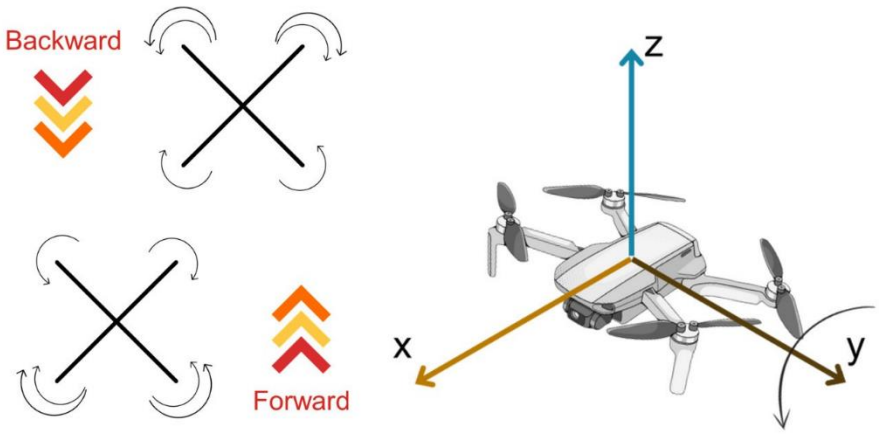


Hình 2.3: Góc roll trong quadcopter

Góc Pitch:

Góc pitch là góc quay của quadcopter quanh trục ngang của máy bay. Góc pitch được đo bằng độ, với 0 độ là vị trí cân bằng, 90 độ là vị trí ngẩng đầu hết lên và -90 độ là vị trí ngẩng đầu hết xuống.

Góc pitch được điều khiển bằng cách thay đổi lực đẩy của hai cánh quạt phía trên và phía dưới. Khi tăng lực đẩy của hai cánh quạt phía trên, quadcopter sẽ ngẩng đầu lên và tăng góc pitch. Khi giảm lực đẩy của hai cánh quạt phía trên, quadcopter sẽ ngẩng đầu xuống và giảm góc pitch. Hình 2.4 mô tả góc pitch trong quadcopter ngẩng đầu lên và xuống.



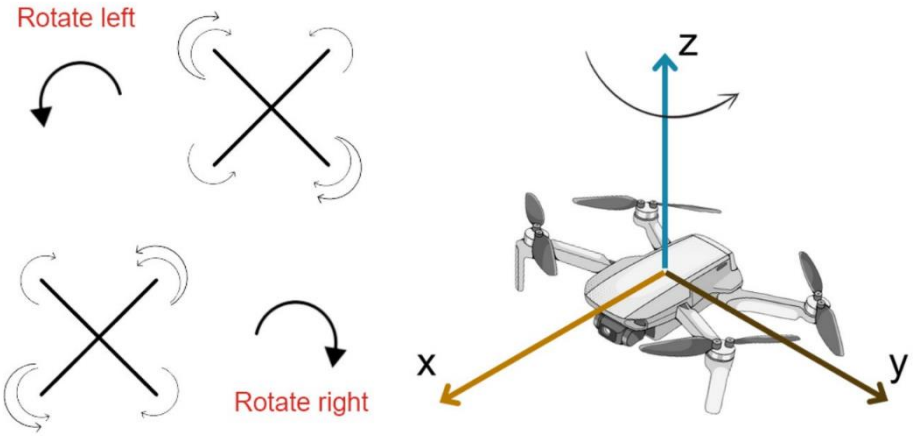
Hình 2.4: Góc pitch trong quadcopter

Góc Yaw:

Góc yaw là góc quay của quadcopter quanh trục z của máy bay. Góc yaw được đo bằng độ, với 0 độ là vị trí cân bằng, 90 độ là vị trí quay hết sang trái và -90 độ là vị trí quay hết sang

phải.

Góc yaw được điều khiển bằng cách thay đổi lực xoắn của hai cánh quạt phía trước và phía sau. Khi giảm lực xoắn của cánh quạt phía trước bên trái, quadcopter sẽ quay sang trái và tăng góc yaw. Khi tăng lực xoắn của cánh quạt phía trước bên trái, quadcopter sẽ quay sang phải và giảm góc yaw. Hình 2.5 mô tả góc yaw trong quadcopter quay trái và phải.



Hình 2.5: Góc yaw trong quadcopter

b. Tương quan giữa góc Roll, Pitch và Yaw

Góc roll, pitch và yaw có mối quan hệ tương quan chặt chẽ với nhau. Khi một trong ba góc thay đổi, hai góc còn lại cũng sẽ thay đổi theo. Ví dụ, khi tăng góc roll, quadcopter sẽ nghiêng về phía trước. Điều này sẽ tạo ra một mô-men lực khiến quadcopter quay sang trái. Do đó, góc yaw cũng sẽ tăng lên. Tương tự, khi giảm góc pitch, quadcopter sẽ ngẩng đầu xuống. Điều này sẽ tạo ra một mô-men lực khiến quadcopter quay sang phải. Do đó, góc yaw cũng sẽ giảm xuống.

Để điều khiển quadcopter bằng góc Roll, Pitch và Yaw, cần sử dụng một bộ điều khiển. Bộ điều khiển sẽ nhận tín hiệu từ người điều khiển, chẳng hạn như thông qua tay cầm hoặc wifi, và sử dụng các tín hiệu này để điều chỉnh lực đẩy và lực xoắn của các cánh quạt. Hệ thống điều khiển của quadcopter thường bao gồm các cảm biến như cảm biến gia tốc, cảm biến độ nghiêng và cảm biến GPS. Các cảm biến này cung cấp thông tin về vị trí, hướng và vận tốc của quadcopter. Hệ thống điều khiển sẽ sử dụng thông tin này để điều chỉnh lực đẩy và lực xoắn của các cánh quạt một cách chính xác, giúp quadcopter bay ổn định và an toàn.

2.3 Cơ sở lý thuyết mô hình bay quadcopter

2.3.1 Xây dựng mô hình động lực học máy bay

Góc Euler, được đặt theo tên của nhà toán học Leonhard Euler, là một phương pháp mô tả hướng của một vật thể trong không gian. Nó sử dụng ba góc quay độc lập để biểu diễn và biến đổi tọa độ của một điểm trong hệ tọa độ thành tọa độ của cùng một điểm trong hệ tọa độ khác. Ba góc quay trong góc Euler bao gồm: góc Roll (Lăn), góc Pitch (Nhào), và góc Yaw (Quay).

Khi kết hợp ba góc quay này, ta có thể xác định một cách đầy đủ vị trí và hướng của vật thể trong không gian ba chiều. Góc Euler là một trong các phương pháp phổ biến để biểu diễn

và điều khiển góc quay của các đối tượng bao gồm cả quadcopter. Các phép quay này bắt đầu từ một hướng chuẩn đã biết. Sự kết hợp này được mô tả bởi các ma trận xoay sau đây:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix} \quad (2.1)$$

$$R_y(\theta) = \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix} \quad (2.2)$$

$$R_z(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

Trong đó ϕ, θ, ψ lần lượt là góc quay quanh trục *x*, *y*, *z*. Các ký hiệu $c(\phi) = \cos(\phi)$, $s(\phi) = sin(\phi)$, $c(\theta) = cos(\theta)$, $s(\theta) = sin(\theta)$, $c(\psi) = cos(\psi)$ và $s(\psi) = sin(\psi)$. Vì vậy, ta tính được ma trận xoay $R_{zyx}(\phi, \theta, \psi)$:

$R_{zyx}(\phi, \theta, \psi) = R_x(\phi).\, R_y(\theta).\, R_z(\psi)$

$$= \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix} \quad (2.4)$$

Lực sinh ra của các rotor:

F_i = b.ω_i^2, i = 1, 2, 3, 4 (2.5)

Khi đó lực nâng của cả khung máy bay là:

T = $\sum_{i=1}^{4} |F_i| = \sum_{i=1}^{4} \omega_i^2$ (2.6)

Phương trình mô tả gia tốc quadcopter:

$$\ddot{r} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \text{g.} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \text{R}\frac{T}{m} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (2.7)$$

Phương trình quan hệ giữa ma trận quán tính $I_R = (I_X, I_Y, I_Z)$, momen quay M và momen quay hồi chuyển:

M_G:I.$\ddot{\Omega}$ = -($\ddot{\Omega}$ x I.$\dot{\Omega}$) - M_G + M (2.8)

Ta có momen quay hồi chuyển phụ thuộc vào các yếu tố vận tốc xoay với u_1 = T, u_2, u_3, u_4 lần lượt là các đơn vị momen quay các chuyển động roll, pitch, yaw hay vận tốc quay $u^T = (u_1, u_2, u_3, u_4)$ và vận tốc góc ω_i máy bay sẽ được

g(u) = $\omega_1 + \omega_2 - \omega_3 - \omega_4$ (2.9)

Kết hợp (2.9) với (2.7) và (2.8) ta có phương trình động lực học:

$$\ddot{x} = -(\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi).\frac{u_1}{m}$$
$$\ddot{y} = -(\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi).\frac{u_1}{m} \quad (2.10)$$
$$\ddot{z} = g - (\cos\phi \cos\theta).\frac{u_1}{m}$$
$$\ddot{\phi} = \dot{\theta}\dot{\psi}(\frac{I_y - I_z}{I_x}) - \frac{I_R}{I_x}\dot{\theta}\, g(u) + \frac{L}{I_x}u_2$$
$$\ddot{\theta} = \dot{\phi}\dot{\psi}(\frac{I_z - I_x}{I_y}) + \frac{I_R}{I_y}\dot{\phi}\, g(u) + \frac{L}{I_y}u_3$$
$$\ddot{\psi} = \dot{\phi}\dot{\theta}(\frac{I_x - I_y}{I_z}) + \frac{1}{I_z}u_4$$

2.3.2 Mô hình toán khí động học

Việc tính toán khí động học mô tả các tác động khi quay của cánh quạt trong không khí. Với các thông số: T_{MT} (N) là lực đẩy của cánh quạt, hướng lên, S (m^2) là diện tích của quạt, ρ_S (kg/m^2) là mật độ không khí

Ta có phương trình của lực đẩy:

$T_{MT} = 2\rho_S S v_I^2$ (N) (2.11)

Do lực đẩy $T_{MT} = W_P = \frac{mg}{4}$ (trọng lượng được mang bởi 1 cánh quạt):

Vận tốc dòng khí cho mỗi cánh quạt:

$V_1 = \sqrt{(T_{MT}')/(2\rho_S S)}$ (m/s) (2.12)

2.4 Hệ thống GPS

Hệ thống định vị toàn cầu (Global Positioning System - GPS) là một hệ thống định vị vệ tinh cung cấp thông tin vị trí và thời gian trong mọi điều kiện thời tiết, bất kể vị trí ở đâu trên hoặc gần Trái Đất. Được phát triển bởi Bộ Quốc phòng Hoa Kỳ, GPS ban đầu được thiết kế cho mục đích quân sự nhưng đã được mở rộng cho sử dụng dân sự và thương mại vào những năm 1980.

Hình 2.6: Hệ thống định vị toàn cầu

2.4.1 Cấu trúc hệ thống GPS

Hệ thống GPS bao gồm ba phần chính:

- Phần Không Gian (Space Segment): Gồm ít nhất 24 vệ tinh GPS hoạt động trong quỹ đạo. Các vệ tinh này được sắp xếp để bất kỳ lúc nào, bất kỳ nơi đâu trên Trái Đất cũng có thể thấy được ít nhất 4 vệ tinh.
- Phần Điều Khiển (Control Segment): Bao gồm các trạm mặt đất theo dõi, giám sát, và điều khiển các vệ tinh. Phần này điều chỉnh quỹ đạo vệ tinh và bảo đảm các tín hiệu từ

vệ tinh là chính xác.

- Phần Người Dùng (User Segment): Bao gồm các thiết bị nhận GPS như điện thoại thông minh, máy định vị trên xe hơi, và thiết bị cầm tay. Những thiết bị này thu tín hiệu từ các vệ tinh và tính toán vị trí hiện tại của người dùng.

2.4.2 Nguyên lý hoạt động

GPS hoạt động dựa trên nguyên lý đo khoảng cách từ thiết bị nhận đến các vệ tinh. Mỗi vệ tinh truyền một tín hiệu mã hóa chứa thời gian gửi tín hiệu và vị trí của vệ tinh tại thời điểm đó. Thiết bị nhận GPS thu nhận tín hiệu từ nhiều vệ tinh (ít nhất là bốn) và sử dụng phép đo tam giác để xác định vị trí chính xác.

- Thời Gian Truyền Tín Hiệu: Tín hiệu từ vệ tinh di chuyển với tốc độ ánh sáng. Bằng cách tính toán thời gian truyền tín hiệu từ vệ tinh đến thiết bị nhận, hệ thống có thể xác định khoảng cách từ vệ tinh đến thiết bị.
- Đo Tam Giác: Khi thiết bị nhận biết khoảng cách từ ít nhất bốn vệ tinh, nó có thể tính toán vị trí ba chiều (vĩ độ, kinh độ và độ cao) của người dùng.

2.4.3 Ứng dụng của GPS

GPS có nhiều ứng dụng trong cả lĩnh vực quân sự và dân sự:

- Quân Sự: Dẫn đường cho tên lửa, máy bay không người lái, và các phương tiện quân sự khác.
- Hàng Không: Hỗ trợ dẫn đường cho máy bay và hệ thống quản lý không lưu.
- Vận Tải: Hệ thống định vị và dẫn đường cho xe hơi, tàu thuyền, và tàu hỏa.
- Viễn Thông: Đồng bộ hóa thời gian trong mạng viễn thông.
- Nông Nghiệp: Hệ thống canh tác chính xác, tối ưu hóa việc sử dụng phân bón và thuốc trừ sâu.
- Giải Trí: Các hoạt động ngoài trời như đi bộ đường dài, leo núi, và câu cá.

Hình 2.7: Ứng dụng GPS để tìm kiếm đường đi

2.4.4 Ưu điểm và hạn chế của GPS

Ưu Điểm:

- Chính Xác: GPS có thể cung cấp thông tin vị trí với độ chính xác cao, thường là trong phạm vi vài mét.

- Toàn Cầu: Hoạt động trên toàn cầu và trong mọi điều kiện thời tiết.
- Miễn Phí: Dịch vụ GPS là miễn phí cho người dùng dân sự.
  Hạn Chế:
- Tín Hiệu Yếu: GPS có thể bị gián đoạn trong các khu vực có vật cản như tòa nhà cao tầng, rừng rậm, hoặc dưới lòng đất.
- Phụ Thuộc Vào Vệ Tinh: Khi vệ tinh không hoạt động hoặc bị tấn công, dịch vụ GPS có thể bị ảnh hưởng.
- Độ Chính Xác Giảm: Trong một số trường hợp, như khu vực đô thị hoặc rừng rậm, tín hiệu GPS có thể bị phản xạ, dẫn đến sai số.

Hệ thống GPS đã cách mạng hóa cách chúng ta xác định vị trí và điều hướng. Từ việc dẫn đường hàng ngày cho đến các ứng dụng chuyên sâu trong quân sự và khoa học, GPS đã trở thành một phần không thể thiếu của cuộc sống hiện đại. Mặc dù có một số hạn chế, nhưng những lợi ích mà hệ thống này mang lại là vô cùng lớn, đóng góp vào sự tiến bộ và phát triển của nhiều lĩnh vực khác nhau.

2.5 Tìm hiểu về Gazebo

Gazebo là một nền tảng mô phỏng robot mã nguồn mở được phát triển bởi OSRF. Nó được sử dụng để mô phỏng các robot vật lý trong môi trường ảo. Gazebo có thể được sử dụng cho nhiều mục đích khác nhau như: Phát triển phần mềm robot bằng cách cho phép các nhà phát triển thử nghiệm các robot của họ trong môi trường an toàn và kiểm soát được, điều này có thể giúp giảm chi phí và thời gian phát triển robot, đồng thời cải thiện chất lượng của phần mềm robot; thử nghiệm robot bằng cách cho phép các nhà nghiên cứu đánh giá hiệu suất của các robot trong các tình huống khác nhau giúp phát hiện các vấn đề tiềm ẩn với robot trước khi chúng được đưa vào sản xuất; sử dụng với mục đích giáo dục về robot bằng cách cho phép sinh viên và giáo viên trải nghiệm các robot trong môi trường ảo giúp sinh viên hiểu rõ hơn về cách hoạt động của robot và cách phát triển phần mềm robot. Gazebo sử dụng mô hình vật lý để mô tả cách các robot tương tác với môi trường của chúng. Mô hình vật lý này được sử dụng để tính toán chuyển động của robot và tương tác của nó với các đối tượng khác trong môi trường. Gazebo cũng có thể được sử dụng để mô phỏng các cảm biến và actuator của robot. Điều này cho phép các nhà phát triển và nhà nghiên cứu thử nghiệm các robot của họ trong môi trường ảo mà không cần phải có robot vật lý.

Gazebo có một số tính năng mạnh mẽ và linh hoạt, bao gồm:

Hình 2.8: Logo Gazebo

- Mô hình vật lý chính xác: để mô tả cách các robot tương tác với môi trường của chúng. Điều này giúp đảm bảo rằng các robot được mô phỏng hoạt động theo cách giống như các robot vật lý.
- Hỗ trợ nhiều loại robot: bao gồm quadcopter, robot cánh tay và robot xe tự lái.
- Tính linh hoạt cao: Gazebo có thể được sử dụng để mô phỏng các robot trong các môi trường khác nhau, từ môi trường trong nhà đến môi trường ngoài trời.
- Cộng đồng lớn: cộng động sử dụng Gazebo có rất nhiều thành viên và đang ngày càng phát triên các nhà phát triển và nhà nghiên cứu, những người cung cấp hỗ trợ và tài nguyên cho người dùng.

Gazebo là một nền tảng giả lập robot mạnh mẽ và linh hoạt có thể được sử dụng cho nhiều mục đích khác nhau. Nó là một lựa chọn tốt cho các nhà phát triển robot, nhà nghiên cứu và giáo viên. Trong đề tài tốt nghiệp, nhóm lựa chọn Gazebo để làm môi trường mô phỏng hoạt động của quadcopter có tích hợp lidar.

2.6 Lidar

Lidar là viết tắt của "Light Detection and Ranging", là một công nghệ sử dụng ánh sáng để đo khoảng cách đến các vật thể. Lidar hoạt động bằng cách phát ra chùm tia laser và đo thời gian ánh sáng quay trở lại. Khoảng cách đến vật thể được tính bằng cách nhân tốc độ ánh sáng với thời gian ánh sáng phát ra và quay trở lại. Lidar có thể được sử dụng để tạo ra mô hình 3D của môi trường. Mô hình này có thể được sử dụng cho nhiều mục đích khác nhau, chẳng hạn như:

- Tạo bản đồ của các khu vực rộng lớn, chẳng hạn như rừng, núi hoặc thành phố.
- Giúp xe tự lái tránh chướng ngại vật.
- Khám phá các khu vực nguy hiểm hoặc khó tiếp cận, chẳng hạn như các khu vực bị chìm dưới nước hoặc các khu vực bị đổ nát.

Cách thức hoạt động của lidar. Lidar có hai loại chính: lidar quét theo xung và lidar quét

Hình 2.9: RPLidar Slamtec A1M8

theo bước sóng.

- Lidar quét theo xung: Lidar quét theo xung phát ra chùm tia laser ngắn và đo thời gian ánh sáng quay trở lại.
- Lidar quét theo bước sóng: Lidar quét theo bước sóng phát ra chùm tia laser liên tục và đo sự thay đổi trong bước sóng của ánh sáng khi nó phản xạ lại từ các vật thể.

Ưu điểm của lidar. Lidar có một số ưu điểm so với các công nghệ định vị khác như:

- Có thể hoạt động trong điều kiện ánh sáng yếu hoặc không có ánh sáng.
- Có thể tạo ra mô hình 3D chính xác của môi trường.
- Có thể được sử dụng để đo khoảng cách đến các vật thể ở xa.

Nhược điểm của lidar. Lidar cũng có một số nhược điểm, chẳng hạn như:

- Giá thành cao.
- Có thể bị nhiễu bởi các nguồn ánh sáng khác, chẳng hạn như mặt trời hoặc đèn đường.

Ứng dụng của lidar. Lidar được sử dụng trong nhiều ứng dụng khác nhau, cụ thể như:

- Tự lái: Lidar là một thành phần quan trọng của hệ thống tự lái giúp xe tự lái tránh chướng ngại vật và xác định vị trí của các vật thể khác trên đường.
- Thám hiểm: Lidar được sử dụng để khám phá các khu vực nguy hiểm hoặc khó tiếp cận, chẳng hạn như các khu vực bị chìm dưới nước hoặc các khu vực bị đổ nát.
- Bản đồ: tạo bản đồ của các khu vực rộng lớn, như rừng, núi hoặc thành phố.
- An ninh: Lidar được sử dụng để phát hiện các vật thể bất thường hoặc xâm nhập.
- Nông nghiệp: Theo dõi sức khỏe của cây trồng và thu hoạch.

Lidar là một công nghệ đang phát triển nhanh chóng. Các nhà nghiên cứu đang phát

Hình 2.10: Một ứng dụng công nghệ Lidar

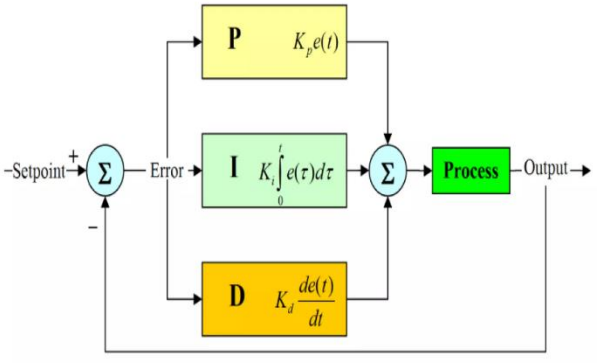
triển các loại lidar mới với độ chính xác cao hơn, phạm vi xa hơn và giá thành thấp hơn. Lidar có tiềm năng trở thành một công nghệ quan trọng trong nhiều lĩnh vực khác nhau. Với các ứng dụng đa dạng cùng giá thành thị trường ở mức trung bình, nhóm quyết định sử dụng lidar làm đối tượng nghiêm cứu cho đề tài tốt nghiệp. Ứng dụng khả năng quét vật thể, quadcopter tích hợp công nghệ lidar sẽ đáp ứng mục tiêu đề tài tốt nghiệp của nhóm: *Nghiên cứu, thiết kế, chế tạo mô hình và hệ thống điều khiển cân bằng quadcopter dò đường tự động.*

2.7 Lý thuyết điều khiển PID

Việc nghiên cứu, mô hình hóa và điều khiển các thiết bị bay quadcopter là một yêu cầu cần thiết. Quadcopter là một hệ thống đa biến với 6 bậc tự do (DOF), và rất khó kiểm soát do sự ghép nối phi tuyến giữa các bộ truyền động và mức độ tự do của thiết bị. Các thuật toán điều khiển bay phổ biến nhất hiện nay được sử dụng là các bộ điều khiển PID, các bộ điều khiển này chỉ có thể thực hiện khi quadcopter bay mang tính thử nghiệm với các điều kiện cụ thể nhất định. Để có thể áp dụng rộng rãi các thiết bị bay điều khiển từ xa hoặc không người lái cần thiết phải nghiên cứu xây dựng một bộ điều khiển, có khả năng xử lý các bậc tự do, cũng như với tác động của môi trường, bao gồm: các góc cuộn (roll) về trục x, là góc lật (pitch) về trục y, và góc nghiêng (yaw) về trục z các góc quay (roll), góc lật (pitch) và góc nghiêng (yaw) được ký hiệu RPY.

Sơ đồ điều khiển PID được đặt tên theo ba khâu hiệu chỉnh của nó, tổng của ba khâu này tạo thành bởi các biến điều khiển (MV).

Ta có:



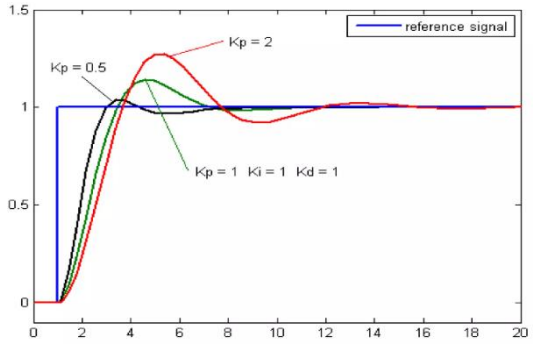
Hình 2.11: Sơ đồ điều khiển PID

$$MV(t) = P_{out} + I_{out} + D_{out}$$

trong đó P_{out}, I_{out}, D_{out} là các thành phần đầu ra từ ba khâu của bộ điều khiển PID, được xác định như hình dưới đây.

2.7.1 Khâu tỉ lệ

Khâu tỉ lệ (đôi khi còn được gọi là *độ lợi*) làm thay đổi giá trị đầu ra, tỉ lệ với giá trị sai



Hình 2.12: Đồ thị PV theo thời gian – Khâu tỉ lệ

số hiện tại. Đáp ứng tỉ lệ có thể được điều chỉnh bằng cách nhân sai số đó với một hằng số K_P, được gọi là hệ số tỉ lệ.

Khâu tỉ lệ được cho bởi: $P_{out} = K_P e(t)$

trong đó

P_{out}: thừa số tỉ lệ của đầu ra

K_P: hệ số tỉ lệ, thông số điều chỉnh

e: sai số = SP – PV

t: thời gian hay thời gian tức thời (hiệu tại)

Hệ số của khâu tỉ lệ lớn là do thay đổi lớn ở đầu ra mà sai số thay đổi nhỏ. Nếu hệ số của khâu tỉ lệ quá cao, hệ thống sẽ không ổn định. Ngược lại, hệ số nhỏ là do đáp ứng ra nhỏ trong khi sai số đầu vào lớn, và làm cho bộ điều khiển kém nhạy, hoặc đáp ứng chậm. Hệ số của khâu tỉ lệ quá thấp, tác động điều khiển có thể sẽ quá bé khi đáp ứng với các nhiễu của hệ thống.

2.7.2 Drop (độ trượt)

Nếu không có nhiễu, điều khiển tỉ lệ thuần túy sẽ không xác lập tại giá trị mong muốn

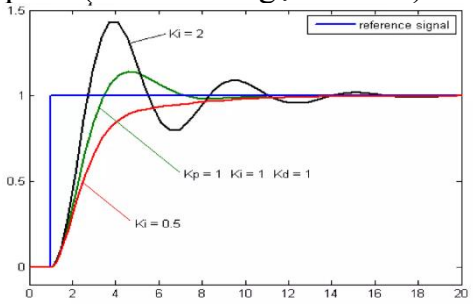
của nó, nhưng nó vẫn duy trì một (độ trượt) sai số ổn định trạng thái, là một hàm của độ lợi tỉ lệ và độ lợi quá trình. Đặc biệt, nếu độ lợi quá trình – trong khoảng thời gian dài bị trôi do thiếu điều khiển, như việc làm mát một lò nung tới nhiệt độ phòng – được ký hiệu G và giả sử sai số xấp xỉ là hằng số, khi đó drop – độ trượt xảy ra khi độ lợi không đổi này bằng thừa số tỉ lệ của đầu ra, P_{out} với sai số là tuyến tính, G = K_P.e, do đó e = G/K_p. Khi thừa số tỉ lệ, đẩy vào thông số tới giá trị đặt, được bù chính xác bởi độ lợi quá trình, nó sẽ kéo thông số ra khỏi giá trị đặt. Nếu độ lợi quá trình giam, khi làm lạnh, thì trạng thái dừng sẽ nằm dưới điểm đặt, ta gọi là drop – độ trượt. Chỉ các thành phần dịch chuyển (trung bình dài hạn, thành phần tần số không) của độ lợi quá trình mới tác động tới độ trượt – các dao động đều hoặc ngẫu nhiên trên hoặc dưới thành phần dịch chuyển sẽ bị triệt tiêu. Độ lợi quá trình có thể thay đổi theo thời gian hoặc theo các thay đổi bên ngoài, ví dụ như nếu nhiệt độ phòng thay đổi, việc làm lạnh sẽ nhanh hơn hoặc chậm hơn.

Độ trượt tỉ lệ thuận với độ lợi quá trình và tỉ lệ nghịch với độ lợi tỉ lệ, và là một khiếm khuyết không thể tránh được của điều khiển tỉ lệ thuần túy. Độ trượt có thể được giảm bớt bằng cách thêm một thừa số độ lệch (cho điểm đặt trên giá trị mong muốn thực tế), hoặc sửa đổi bằng cách thêm một khâu tích phân (trong bộ điều khiển PI hoặc PID), sẽ tính toán độ lệch thêm vào một cách hữu hiệu.

Bất chấp độ trược, cả lý thuyết điều chỉnh lẫn thực tế công nghiệp chỉ ra rằng khâu tỉ lệ là cần thiết trong việc tham gia vào quá trình điều khiển.

2.7.3 Khâu tích phân

Phân phối của khâu tích phân (đôi khi còn gọi là reset) tỉ lệ thuận với cả biên độ sai số



Hình 2.13: Đồ thị PV theo thời gian - Khâu tích phân

lẫn quãng thời gian xảy ra sai số. Tổng sai số tức thời theo thời gian (tích phân sai số) cho ta tích lũy bù đã được hiệu chỉnh trước đó. Tích lũy sai số sau đó được nhân với độ lợi tích phân và cộng với tín hiệu đầu ra của bộ điều khiển. Biên độ phân phối của khâu tích phân trên tất cả tác động điều chỉnh được xác định bởi độ lợi tích phân, K_I.

Thừa số tích phân được cho bởi:

$$I_{out} = K_I \int_0^t e(\tau)d\tau$$

trong đó

I_{out}: thừa số tích phân của đầu ra

K_I: độ lợi tích phân, 1 thông số điều chỉnh

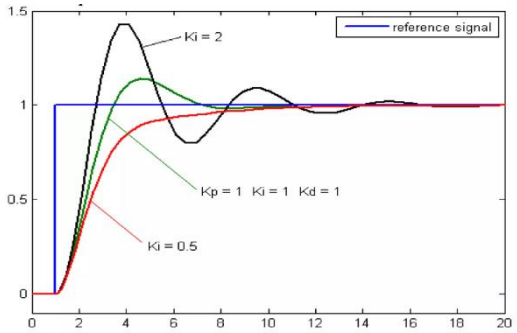
e: sai số = SP – SV

t: thời gian hoặc thời gian tức thời (hiện tại)

τ: một biến tích phân trung gian

Khâu tích phân (khi cộng thêm khâu tỉ lệ) sẽ tăng tốc chuyển động của quá trình tới điểm đặt và khử số dư sai số ổn định với một tỉ lệ phụ thuộc vào bộ điều khiển. Tuy nhiên, vì khâu tích phân là đáp ứng của sai số tích lũy trong quá khứ, nó có thể khiến giá trị hiện tại vọt lố qua giá trị đặt (ngang qua điểm đặt và tạo ra một độ lệch với các hướng khác).

2.7.4 Khâu vi phân



Hình 2.14: Đồ thị PV theo thời gian - Khâu vi phân

Tốc độ thay đổi của sai số quá trình được tính toán bằng cách xác định độ dốc của sai số theo thời gian (tức là đạo hàm bậc một theo thời gian) và nhân tốc độ này với độ lợi tỉ lệ K_D. Biên độ của phân phối khâu vi phân (đôi khi được gọi là tốc độ) trên tất cả các hành vi điều khiển được giới hạn bởi độ lợi vi phân, K_D.

Thừa số vi phân được cho bởi:

$$D_{out} = K_D \frac{d}{dt} \mathrm{e(t)}$$

trong đó

D_{out}: thừa số vi phân của đầu ra

K_D: độ lợi vi phân, một thông số điều chỉnh

e: sai số = SP – SV

t: thời gian hoặc thời gian tức thời (hiện tại)

Khâu vi phân làm chậm tốc độ thay đổi của đầu ra bộ điều khiển và đặc tính này là đáng chú ý nhất để đạt tới điểm đặt của bộ điều khiển. Từ đó, điều khiển vi phân được sử dụng để làm giảm biên độ, được tạo ra bởi thành phần tích phân và tăng cường độ ổn định của bộ điều khiển hỗn hợp. Tuy nhiên, phép vi phân của một tín hiệu sẽ khuếch đại nhiễu và do đó khâu này sẽ nhạy hơn đối với nhiễu trong sai số, và có thể khiến quá trình trở nên không ổn định nếu nhiễu và độ lợi vi phân đủ lớn. Do đó một xấp xỉ của bộ vi sai với băng thông giới hạn thường được sử dụng hơn. Chẳng hạn như mạch bù sớm pha.

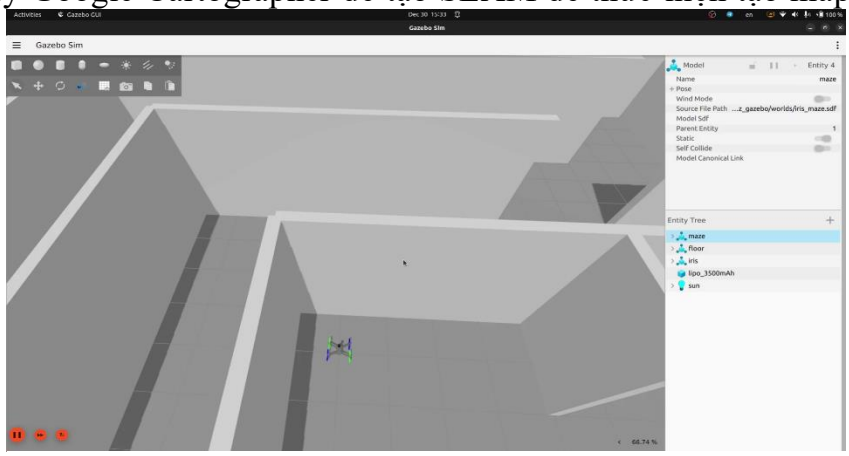
CHƯƠNG 3: MÔ PHỎNG KHẢO SÁT

3.1 Mô phỏng Gazebo

Khởi chạy mô phỏng chứa một quadcopter được trang bị Lidar 2D 360 độ trong một map hình mê cung, câu lệnh sẽ khởi chạy Rviz và Gazebo:

Cartographer SLAM with ROS 2 in SITL: Terminal 1: ros2 launch ardupilot_gz_bringup iris_maze.launch.py

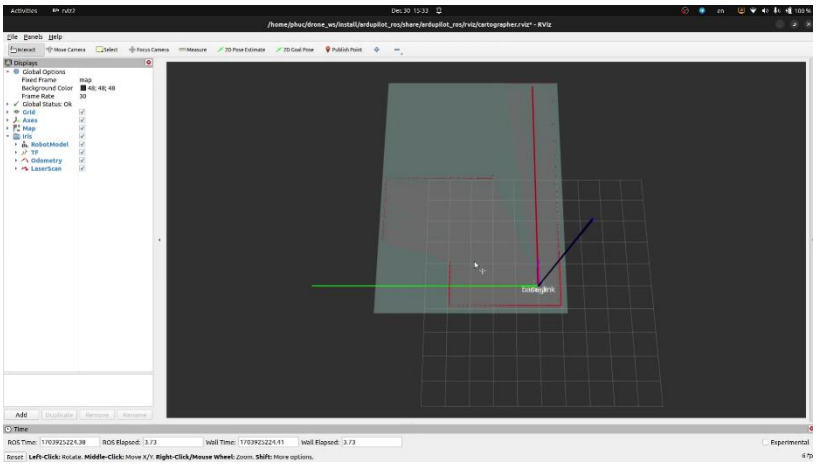
Khởi chạy Google Cartographer để tạo SLAM để thực hiện tạo map, hiển thị nó lên



Hình 3.1: Khởi chạy mô phỏng

Rviz:

ros2 launch ardupilot_ros cartographer.launch.py

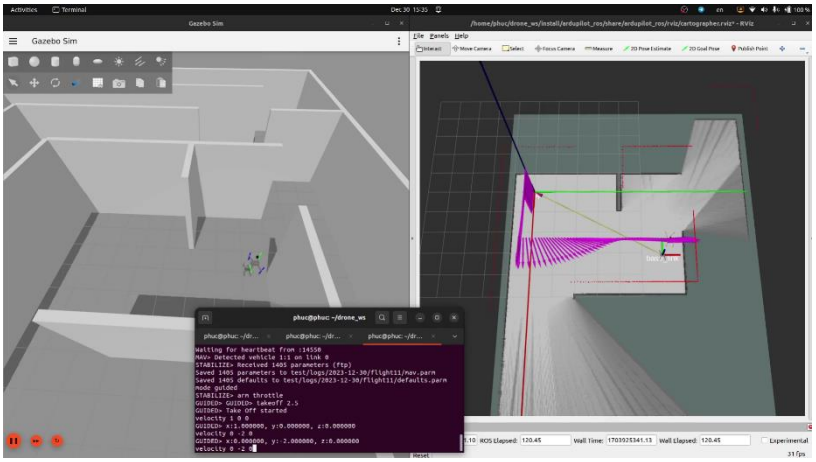


Hình 3.2: Khởi chạy Google Cartographer

Khởi chạy MAVProxy để tương tác với hệ thống điều khiển bay tự động thông qua giao thức MAVLink, cụ thể điều khiển ví dụ như sau:

- mode guided: chọn chế độ thực hiện các lệnh từ bên ngoài.
- arm throttle: kích hoạt cần gạt ga cho phép điều khiển động cơ máy bay.
- takeoff 2.5: cho máy bay cất cánh lên độ cao 2.5 mét.
- velocity 1 0 0: thiết lập vận tốc cho máy bay.

Cụ thể, trong trường hợp này, vận tốc được thiết lập là 1 m/s theo hướng trục x, 0 m/s theo hướng trục y và 0 m/s theo hướng trục z. Điều này có thể được sử dụng để điều khiển phương tiện bay di chuyển theo hướng và vận tốc cụ thể.



Hình 3.3: Khởi chạy MAVProxy

CHƯƠNG 4: HỆ THỐNG CƠ KHÍ, MẠCH ĐIỆN

4.1 Lựa chọn thiết bị

Bảng 4.1: Các thành phần thiết bị

Khung	DJI F450 Frame kit
Động cơ	DJI Phantom 3 – 2312A
Cánh quạt	1045 carbon fiber propeller
Bộ điều tốc	4in1 Hobbywing xrotor
Pin	3S Tatto 2300mAh
Transmitter	Arduino nano
Receiver	Arduino nano
Mạch điều khiển	DJI Naza－M V2
GPS	DJI Naza－M V2 GPS module
LiDAR	Slamtec RPLIDAR A1M8

4.1.1 Khung

Khung F450 là một trong những loại khung quadcopter phổ biến nhất dành cho các dự án DIY và nghiên cứu về máy bay không người lái. Nhóm em chọn khung F450 vì giá thành rẻ, được cộng đồng đông đảo sử dụng, chất lượng bộ khung đến khối lượng nhẹ đều ở mức tốt so với giá thành.

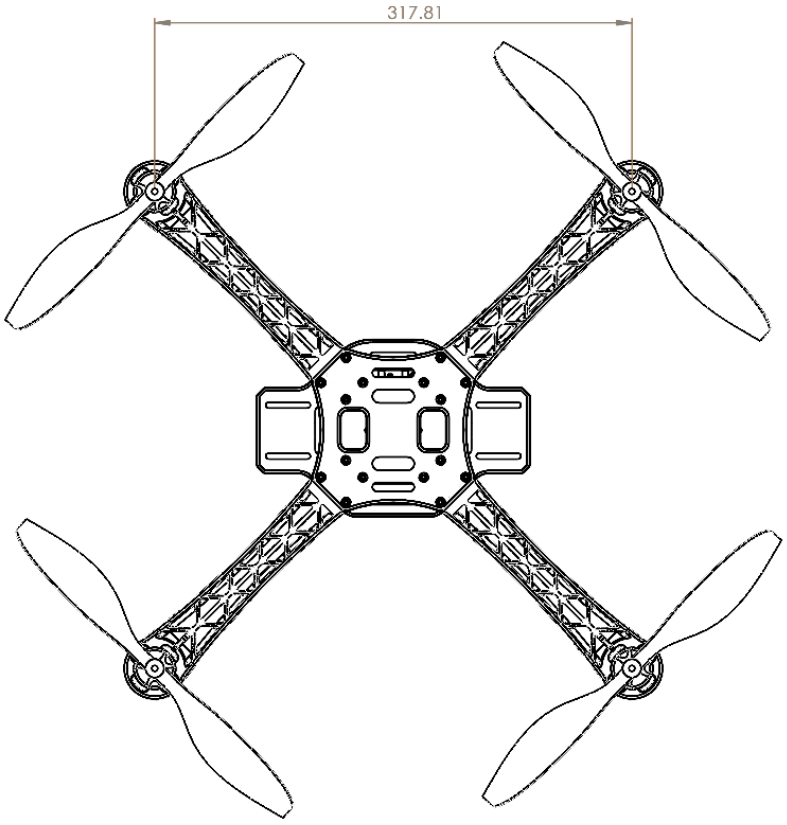
Thiết kế và cấu trúc:

- **Kích thước:** Khung F450 có đường chéo (diagonal wheelbase) khoảng 450mm, phù hợp với các dự án drone cỡ trung.
- **Vật liệu:** Khung thường được làm từ sợi thủy tinh hoặc sợi carbon. Sợi thủy tinh mang lại sự chắc chắn và độ bền cao, trong khi sợi carbon nhẹ hơn và cung cấp độ cứng tốt hơn.
- **Hình dạng:** Khung có thiết kế hình chữ X với bốn cánh tay đối xứng, mỗi cánh tay gắn một động cơ ở đầu.

Đặc điểm nổi bật:

- **Dễ dàng Lắp Ráp:** Khung F450 được thiết kế để dễ dàng lắp đặt và tháo gỡ các bộ phận như động cơ, ESC, bộ điều khiển bay, pin và các cảm biến.
- **Tính Tương Thích Cao:** Khung này tương thích với nhiều loại động cơ, bộ điều khiển bay, và thiết bị điện tử khác trên thị trường, giúp dễ dàng nâng cấp và tùy chỉnh.

- **Khả Năng Mở Rộng:** Người dùng có thể gắn thêm nhiều phụ kiện như camera, gimbal, và các cảm biến khác để phục vụ cho nhiều mục đích khác nhau như quay phim, chụp ảnh hoặc khảo sát



Hình 4.1: Khung quadcopter F450

4.1.2 Hệ thống nâng

Sau khi xác định khung, bước tiếp theo là lựa chọn động cơ và cánh quạt phù hợp. Quá trình này dựa trên các thông số kỹ thuật được cung cấp trong bảng mô tả sản phẩm. Các yếu tố cần xem xét bao gồm:

- Công suất động cơ: Phải đủ mạnh để nâng tổng trọng lượng của drone.
- Kích thước cánh quạt: Phải tương thích với khung và động cơ đã chọn.
- Hiệu suất năng lượng: Cân bằng giữa công suất và thời gian bay.
- Độ ồn: Quan trọng đối với các ứng dụng đòi hỏi hoạt động yên tĩnh.
  Từ đó ta chọn động cơ và cánh quạt phù hợp sau:
  Động cơ DJI Phantom 3 - 2312A

Hình 4.2: Động cơ DJI Phantom 3

Cánh quạt 10x4.5

Hình 4.3: Cánh quạt 10x4.5

Động cơ DJI Phantom 3 - 2312A được lựa chọn vì nhiều ưu điểm phù hợp với yêu cầu của đồ án:

- Công suất phù hợp: Với thông số 960KV, công suất tối đa khoảng 350W, động cơ này cung cấp đủ lực đẩy cho khung F450 (Khoảng 800-1000g trên một động cơ) , đảm bảo khả năng nâng và di chuyển ổn định của drone.
- Tương thích pin: Khả năng hoạt động với pin 2-3S cho phép linh hoạt trong việc lựa chọn nguồn điện, cân bằng giữa trọng lượng và thời gian bay.
- Độ tin cậy cao, được kiểm chứng qua sử dụng trong Phantom 3.
- Hiệu suất nhiệt: Tản nhiệt tốt, cho phép hoạt động ổn định trong thời gian dài.
- Tiếng ồn thấp và rung động ít
- Trọng lượng nhẹ: Góp phần giảm tổng trọng lượng của drone, tăng thời gian bay và khả năng cơ động.

Cánh quạt 1045 được chọn để phối hợp với động cơ A2216 1250KV vì:

- Kích thước phù hợp: Đường kính 10 inch phù hợp với khung F450, tạo ra lực đẩy tối ưu mà không gây cản trở.
- Bước cánh 4.5 inch: Cung cấp sự cân bằng tốt giữa lực đẩy và hiệu suất năng lượng.

- Hiệu quả: Tỷ lệ đường kính và bước cánh này tạo ra lực đẩy mạnh mẽ với mức tiêu thụ điện năng hợp lý.
- Độ ồn thấp: Thiết kế này giúp giảm tiếng ồn so với các cánh quạt có bước cánh lớn hơn, phù hợp cho các ứng dụng yêu cầu hoạt động yên tĩnh.
- Độ bền: Cánh quạt 1045 được làm từ vật liệu sợi cacbon chất lượng cao, đảm bảo độ bền, nhẹ và an toàn trong quá trình sử dụng.

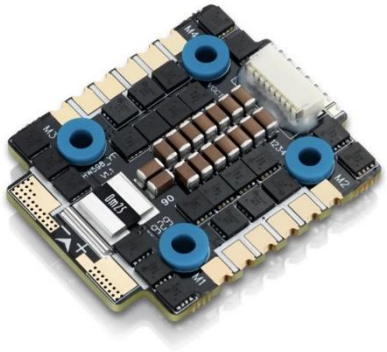
4.1.3 Bộ điều tốc 4in1 Hobbywing xrotor (ESC):

Đặc điểm:

- Tích hợp 4 ESC trong 1 board.
- Dòng điện liên tục: Thường từ 40A đến 60A cho mỗi motor (tùy model cụ thể).
- Hỗ trợ giao thức DShot và BLHeli_32 firmware.
- Điện áp hoạt động: Thường hỗ trợ 3-6S LiPo.

Ưu điểm:

- Giảm trọng lượng và đơn giản hóa việc lắp đặt.
- Hiệu suất cao với khả năng xử lý tín hiệu nhanh.
- Dễ dàng cấu hình và tinh chỉnh.



Hình 4.4: ESC 4in1 Hobbywing Xrotor

Thông số kỹ thuật:

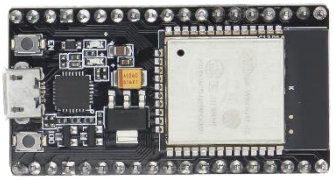
- Mã Sản Phẩm: XRotor 4in1 35A
- Số Kênh: 4in1 (tích hợp 4 ESC trong một module)
- Dòng Định Mức: 35A liên tục, 45A đỉnh (burst current) mỗi kênh
- Điện Áp Hoạt Động: Hỗ trợ pin LiPo từ 3S (11.1V) đến 6S (22.2V)
- Kích Thước: Khoảng 45.5 x 37.5 x 7mm
- Trọng Lượng: Khoảng 12g

4.1.4 Bộ phát tín hiệu (Transmitter):

Sử dụng Arduino Nano làm bộ phát tín hiệu mang lại nhiều ưu điểm:

- Kích thước nhỏ gọn, phù hợp cho thiết bị điều khiển cầm tay.
- Khả năng lập trình linh hoạt, cho phép tùy chỉnh giao diện điều khiển.
- Tiêu thụ điện năng thấp, kéo dài thời gian sử dụng của bộ điều khiển.
- Chi phí thấp nhưng hiệu quả cao, phù hợp cho các dự án nghiên cứu và phát triển.

4.1.5 Bộ thu tín hiệu (Receiver):



Hình 4.5: ESP32

Việc sử dụng cùng loại ESP32 cho cả bộ phát và bộ thu đảm bảo:

- Wi-Fi và Bluetooth tích hợp: ESP32 có cả kết nối Wi-Fi và Bluetooth, bao gồm cả BLE (Bluetooth Low Energy), giúp nó có thể kết nối và giao tiếp với các thiết bị khác một cách linh hoạt.
- Hiệu suất cao: Với vi xử lý lõi kép (dual-core) 32-bit LX6, ESP32 có khả năng xử lý nhanh và hiệu quả, đáp ứng được các yêu cầu tính toán phức tạp.
- Tiết kiệm năng lượng: ESP32 được thiết kế để tiết kiệm năng lượng với nhiều chế độ ngủ (sleep modes), giúp kéo dài tuổi thọ pin cho các ứng dụng chạy bằng pin.
- Bộ nhớ: Nó có bộ nhớ RAM và Flash lớn, cho phép lưu trữ và thực hiện các chương trình phức tạp.
- Đa dạng các cổng I/O: ESP32 cung cấp nhiều chân GPIO, ADC, DAC, SPI, I2C, UART, và PWM, cho phép kết nối với nhiều loại cảm biến và thiết bị ngoại vi khác nhau.
- Cộng đồng hỗ trợ mạnh mẽ**:** Có một cộng đồng lớn người dùng và nhà phát triển hỗ trợ, cùng với rất nhiều tài liệu, hướng dẫn và thư viện mã nguồn mở giúp việc phát triển và triển khai các dự án với ESP32 trở nên dễ dàng hơn.

4.1.6 Mạch điều khiển:

Hình 4.6: DJI Naza M Lite

Mạch điều khiển DJI Naza M Lite là một lựa chọn chuyên nghiệp, mang lại nhiều tính

năng:

- Hệ thống điều khiển bay tự động tiên tiến.
- Khả năng ổn định và cân bằng tốt, đặc biệt quan trọng cho việc chụp ảnh và quay phim.
- Tích hợp sẵn các thuật toán điều khiển phức tạp, giúp tiết kiệm thời gian phát triển.
- Giao diện người dùng thân thiện, dễ dàng cấu hình và tinh chỉnh.

4.1.7 GPS:

DJI Naza M Lite GPS module Module tích hợp với mạch điều khiển DJI Naza M Lite cung cấp:

- Định vị chính xác, hỗ trợ các tính năng như bay theo điểm đánh dấu và tự động quay về.
- Tích hợp seamless với hệ thống điều khiển, tăng độ ổn định và độ tin cậy.
- Cải thiện khả năng giữ vị trí khi bay treo, đặc biệt hữu ích cho việc chụp ảnh, quay phim và dò đường.

Hình 4.7: GPS Naza

4.1.8 LiDAR:

Slamtec RPLIDAR A1M8 Cảm biến LiDAR Slamtec RPLIDAR A1M8 bổ sung khả năng cảm nhận môi trường 360 độ:

- Tạo bản đồ 2D của môi trường xung quanh với độ chính xác cao.
- Hỗ trợ tính năng tránh vật cản và định vị trong không gian.
- Tốc độ quét nhanh (5.5Hz) cho phép cập nhật thông tin môi trường kịp thời.
- Phạm vi đo từ 0.15m đến 12m, phù hợp cho nhiều môi trường hoạt động khác nhau.

Hình 4.8: Lidar A1M8

4.1.9 Pin:

Đặc điểm:

- Dung lượng: 2300mAh
- Loại: LiPo (Lithium Polymer)
- Điện áp: Có thể là 3S (11.1V) hoặc 4S (14.8V) - cần xác nhận
- C-rating: 45C

Ưu điểm:

- Thương hiệu uy tín, chất lượng cao.
- Cân bằng tốt giữa trọng lượng và dung lượng.
- Phù hợp cho các ứng dụng đòi hỏi hiệu suất cao.

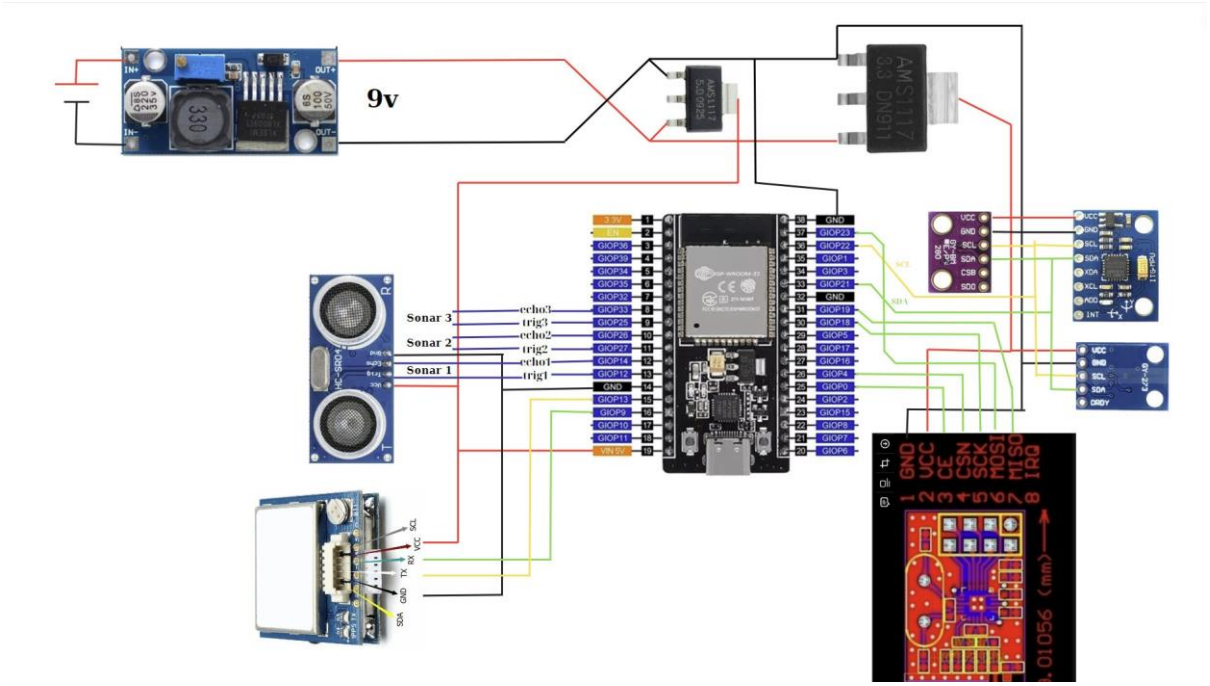
Thông số kỹ thuật

- Thương Hiệu: TATTU
- Loại Pin: Lithium Polymer (LiPo)
- Số Cell: 4S (4 cells)
- Dung Lượng: 2300mAh
- Tốc Độ Xả: 45C liên tục, 90C đỉnh (burst)
- Điện Áp Danh Định: 14.8V
- Kích Thước: Khoảng 105 x 33 x 29mm
- Trọng Lượng: Khoảng 240g
- Đầu Nối: XT60 hoặc các loại đầu nối khác tùy thuộc vào yêu cầu của hệ thống

4.2 Khung quadcopter

Bản vẽ các chi tiết khung xem ở tập bản vẽ.

4.3 Hệ thống mạch điện



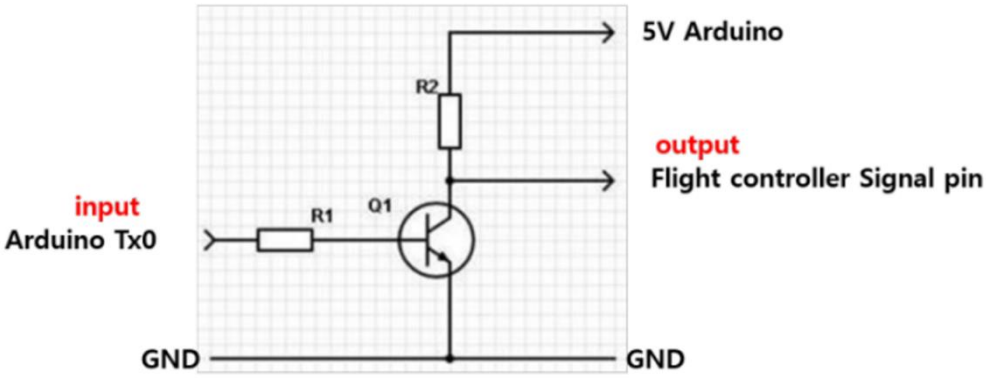
Hình 4.9: Sơ đồ hệ thống

- Arduino Nano: Đây là bộ vi điều khiển trung tâm, đóng vai trò xử lý tín hiệu và điều khiển.
- NRF24L01 Transceiver module: Đây là module truyền nhận không dây, cho phép giao tiếp từ xa với thiết bị.
- Flight controller: Bộ điều khiển chuyến bay (NAZA) , nhận tín hiệu từ Arduino để kiểm soát động cơ và các thành phần khác của drone.
- BEC 5V: Battery Elimination Circuit, cung cấp điện áp 5V ổn định cho hệ thống.
- LED: Đèn báo hiệu, có thể dùng để chỉ thị trạng thái hoạt động.

Kết nối:

- Module NRF24L01 được kết nối với Arduino Nano qua các chân tín hiệu (CE, CSN, SCK, MOSI, MISO).
- Flight controller kết nối với Arduino Nano qua 2 dây: SIGNAL (tín hiệu), GND (đất).
- ESP32: Đây là bộ vi điều khiển trung tâm, đóng vai trò xử lý tín hiệu và điều khiển.
- Cảm biến siêu âm HC SR-04: giúp phát hiện vật cản để Quadcopter né trong quá trình bay.
- GY-273: Cảm biến la bàn được dùng để đo từ trường của Trái Đất nhằm xác định phương hướng.
- GY-BMP280 là mạch cảm biến áp suất được dùng để đo áp suất khí quyển và các chỉ số nhiệt độ.
- MPU6050 là hệ thống cơ điện vi mô bao gồm Gia tốc kế 3 trục và Con quay hồi chuyển 3 trục bên trong nó.

LED được kết nối với một chân digital của Arduino Nano.



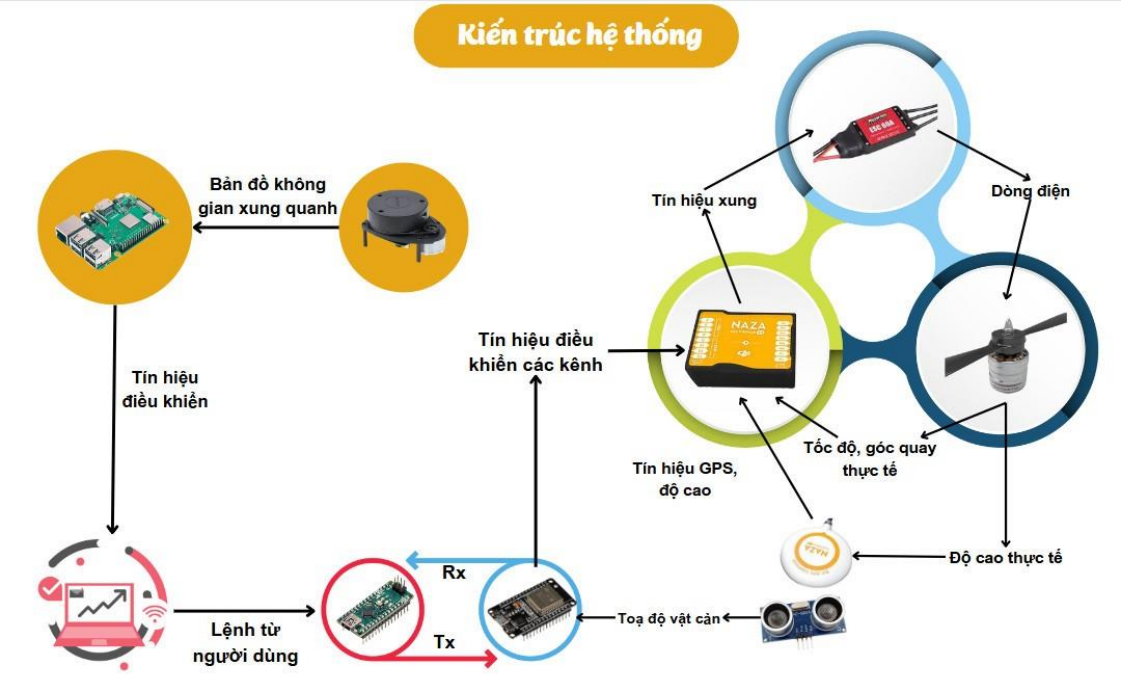
Hình 4.10: Sơ đồ Module NRF24L01

4.4 Mô hình quadcopter

CHƯƠNG 5: ĐIỀU KHIỂN CÂN BẰNG QUADCOPTER

Hình 4.11: Mô hình quadcopter

5.1 Kiến trúc hệ thống



Hình 5.1: Sơ đồ kiến trúc hệ thống

Điều khiển từ người dùng:

- Giao diện người dùng: Là giao diện trên máy tính cho phép người điều khiển nhập các lệnh như điều hướng, thay đổi độ cao, hoặc thực hiện các nhiệm vụ cụ thể.
- Bộ phát tín hiệu (Tx): Chuyển đổi lệnh từ người dùng thành tín hiệu vô tuyến. Hoạt động ở tần số 2.4GHz để đảm bảo khoảng cách điều khiển xa và giảm nhiễu.
- Bộ thu tín hiệu (Rx): Đặt trên drone, nhận tín hiệu từ Tx và chuyển đổi thành dữ liệu kỹ thuật số mà bộ điều khiển bay có thể xử lý.

Xử lý tín hiệu và điều khiển chính:

- Bộ điều khiển bay NAZA (DJI): Đây là "bộ não" của drone. Nó tích hợp nhiều chức năng:
  - Xử lý tín hiệu điều khiển từ Rx.
  - Tích hợp và xử lý dữ liệu từ các cảm biến như GPS, gia tốc kế, và la bàn.
  - Thực hiện các thuật toán ổn định và điều khiển bay.
  - Điều phối hoạt động của các động cơ thông qua ESCs.
- Module GPS: Cung cấp thông tin về vị trí, độ cao, và tốc độ di chuyển của drone. Quan trọng cho các tính năng như giữ vị trí, bay theo waypoints, và chức năng "return to home".

Hệ thống động cơ:

- ESC (Electronic Speed Controller): Nhận tín hiệu từ NAZA và điều chỉnh dòng điện cung cấp cho động cơ.

- Động cơ brushless: Chuyển đổi năng lượng điện thành chuyển động quay. Loại động cơ này có hiệu suất cao, ít ma sát, và tuổi thọ dài.
- Cánh quạt: Tạo lực đẩy để nâng và di chuyển drone. Kích thước và pitch của cánh quạt ảnh hưởng đến hiệu suất bay.

Hệ thống dò đường:

- LiDAR (Light Detection and Ranging): Sử dụng laser để quét môi trường xung quanh, tạo bản đồ 3D chi tiết dùng cho việc dò đường, tránh vật cản và định vị trong không gian.
- Bộ xử lý - Raspberry Pi: Xử lý dữ liệu từ LiDAR, thực hiện các thuật toán SLAM (Simultaneous Localization and Mapping) để định vị drone trong môi trường không xác định.

Phản hồi và điều chỉnh:

- Cảm biến IMU (Inertial Measurement Unit): Tích hợp trong NAZA, bao gồm gia tốc kế và con quay hồi chuyển, đo lường chuyển động và định hướng của drone.
- Barometer: Đo áp suất không khí để xác định độ cao chính xác.
- NAZA liên tục xử lý dữ liệu từ tất cả các cảm biến này để điều chỉnh hoạt động của động cơ, duy trì ổn định và thực hiện các lệnh bay.

5.2 NAZA M LITE

Cấu trúc:

- Khởi tạo và cấu hình hệ thống:
    - Khởi tạo các cảm biến (IMU, barometer, compass, GPS).
    - Cấu hình các tham số bay (PID, giới hạn góc nghiêng, tốc độ tối đa, etc.).
    - Khởi tạo giao tiếp với các thiết bị ngoại vi (ESCs, receiver, etc.).
- Main loop:
    - Đọc dữ liệu từ các cảm biến.
    - Đọc tín hiệu điều khiển từ receiver.
    - Xử lý dữ liệu và tính toán điều khiển.
    - Gửi tín hiệu điều khiển đến ESCs.
    - Cập nhật telemetry.
- Xử lý tín hiệu cảm biến:
    - Lọc nhiễu và fusion data từ IMU (accelerometer, gyroscope).
    - Xử lý dữ liệu GPS.
    - Tính toán độ cao từ barometer.
    - Xử lý dữ liệu la bàn

- Thuật toán điều khiển bay:
    - PID controllers cho attitude control (roll, pitch, yaw).
    - Altitude hold algorithm.
    - Position hold algorithm (khi có GPS).
    - Waypoint navigation (cho chế độ bay tự động).
- Xử lý chế độ bay:
    - Manual mode.
    - Attitude mode.
    - GPS mode.
    - Failsafe mode (return to home, auto-landing).
- Quản lý nhiệm vụ:
    - Xử lý các lệnh từ ground station.
    - Quản lý waypoints cho chế độ bay tự động.
    - Xử lý các nhiệm vụ đặc biệt (chụp ảnh, quay video, etc.).
- Hệ thống an toàn:
    - Kiểm tra điện áp pin.
    - Xử lý mất tín hiệu điều khiển.
    - Giới hạn độ cao và khoảng cách bay.
    - Tránh va chạm (nếu có cảm biến phù hợp).
- Giao tiếp:
    - Xử lý giao thức truyền thông với ground station.
    - Gửi dữ liệu telemetry.
    - Nhận và xử lý cập nhật firmware (nếu có).
- Logging và diagnostics:
    - Ghi log các thông số bay.
    - Hệ thống báo lỗi và cảnh báo.
- Tối ưu hóa hiệu suất:
    - Quản lý tài nguyên CPU.
    - Tối ưu hóa vòng lặp chính để đảm bảo tần suất cập nhật cao.

5.3 Phần mềm điều khiển quadcopter

Chương trình cho giao diện: Phụ lục 1

Giải thích cụ thể chương trình:

Sử dụng PyQt5 cho giao diện người dùng (GUI) và thư viện ‘serial’ để giao tiếp với

phần cứng drone. Cho phép người dùng điều khiển drone bằng cả đầu vào bàn phím máy tính và nút trên GUI, gửi lệnh thông qua kết nối nối tiếp.

Ứng dụng Điều khiển Drone được thiết kế để gửi lệnh đến drone thông qua kết nối nối tiếp. Ứng dụng có giao diện đồ họa người dùng trực quan với các nút cho các thao tác drone khác nhau và cho phép điều khiển thời gian thực bằng đầu vào bàn phím.

Thư viện và Phụ thuộc

- PyQt5: Để tạo GUI.
- `serial`: Để xử lý giao tiếp nối tiếp với drone.
- `sys`: Để xử lý các tham số và chức năng đặc thù hệ thống.
- `keyboard`: Để bắt các sự kiện bàn phím.

Các thành phần chính của mã bao gồm lớp `MainWindow`, kế thừa từ `QMainWindow` của PyQt5, và các phương thức khác nhau để xử lý tương tác GUI và giao tiếp nối tiếp.

Lớp `MainWindow` khởi tạo các thành phần GUI, thiết lập kết nối nối tiếp và xử lý tương tác người dùng.

Khởi tạo (`__init__`)

- Thiết lập các thành phần UI.
- Cố gắng thiết lập kết nối nối tiếp.
- Khởi tạo bộ đếm thời gian và trạng thái lệnh.

Thiết lập UI (`setupUi`)

- Tạo các nút điều khiển drone.
- Kết nối các nút với các trình xử lý sự kiện tương ứng.
- Thêm nút "Arm/Disarm" với chỉ báo trạng thái.

Giao tiếp Nối tiếp

- Thiết lập kết nối đến cổng COM được chỉ định.
- Gửi lệnh đến quadcopter qua cổng nối tiếp.

Lệnh Nút

- Mỗi nút gửi một chuỗi lệnh cụ thể đến drone.
- Lệnh được gửi liên tục khi nút được nhấn.

Chức năng Arm/Disarm

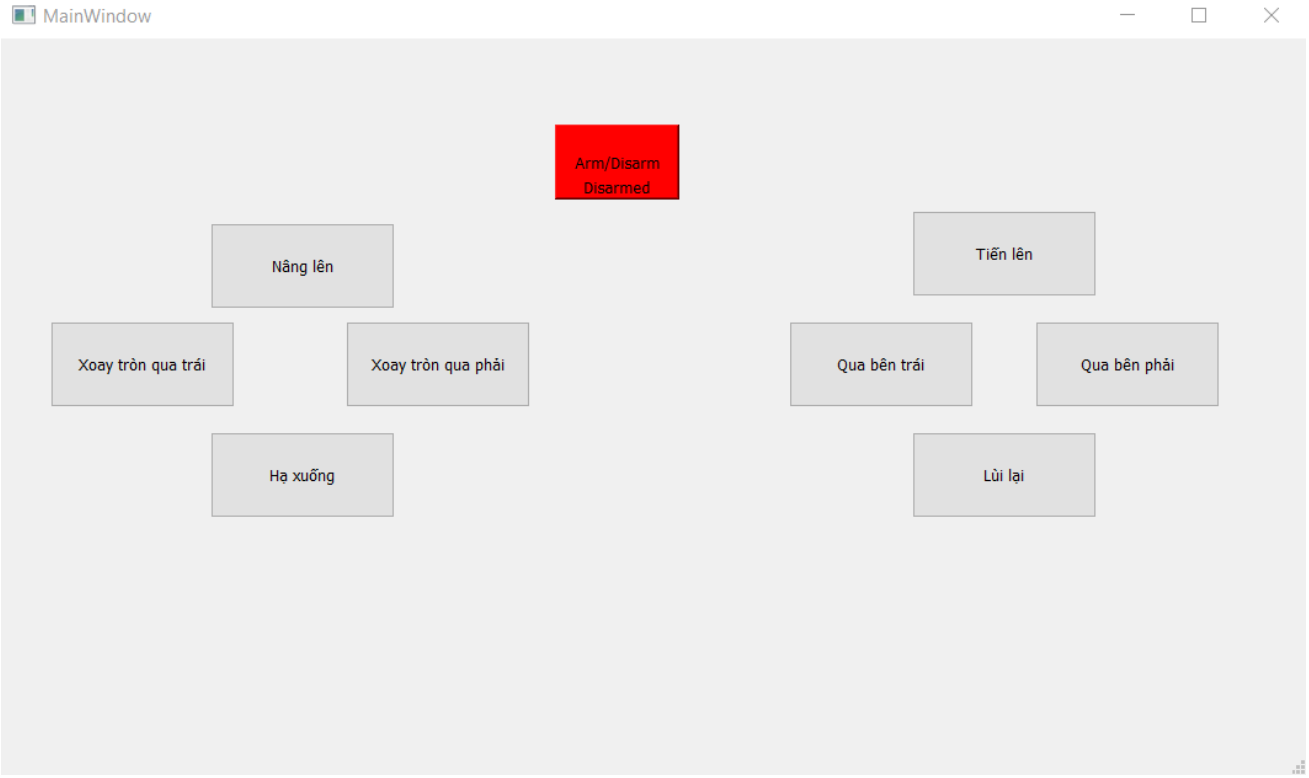
- Thực hiện việc arm và disarm drone.
- Cập nhật GUI để phản ánh trạng thái hiện tại.
- Nhấn nút để kích hoạt động cơ. Sau 1,5 giây, trạng thái “Armed” sẽ được hiển thị trên giao diện. Khi đã ở trạng thái “Armed”, nhấn lại nút để tắt động cơ và trở về trạng thái “Disarmed”.

Xử lý Đầu vào Bàn phím

- Bắt các phím được nhấn để điều khiển chuyển động của drone.
- Đặt lại các giá trị điều khiển khi phím được thả.

Lệnh được gửi đến drone theo các khoảng thời gian đều đặn bằng bộ đếm thời gian. Điều này đảm bảo kiểm soát mượt mà và liên tục đối với chuyển động của drone.

Điều khiển quadcopter di chuyển bằng các nút nhấn trên giao diện



Hình 5.2: Giao diện điều khiển

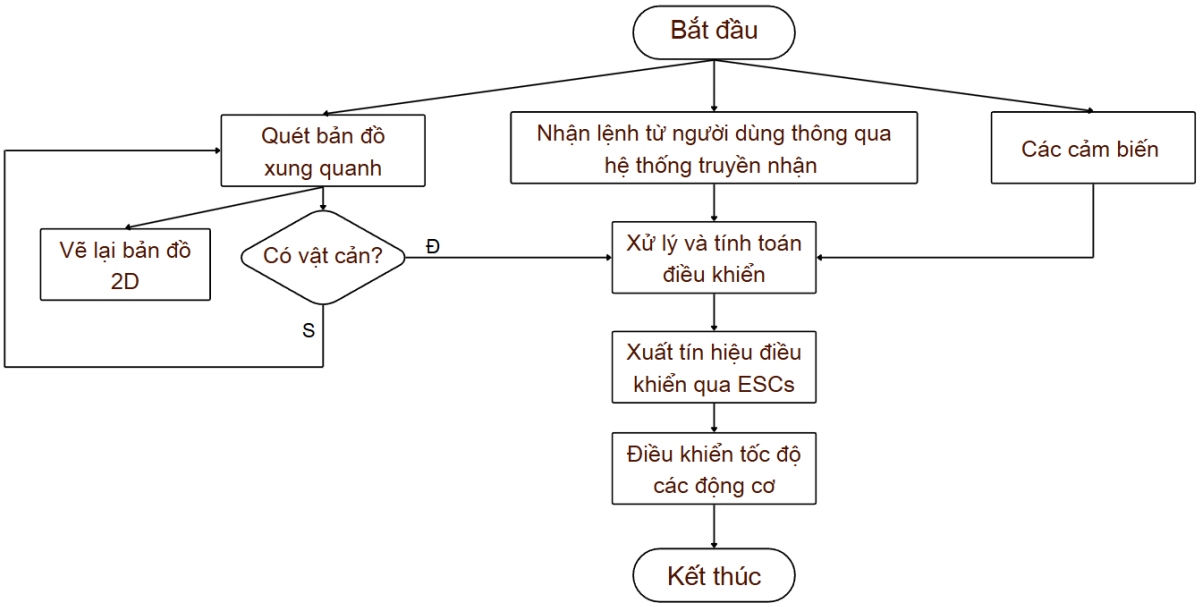
- **Tiến lên:** Nhấn và giữ nút "Tiến lên" để drone bay về phía trước.
- **Lùi lại:** Nhấn và giữ nút "Lùi lại" để drone bay về phía sau.
- **Qua trái:** Nhấn và giữ nút "Qua trái" để drone di chuyển sang trái.
- **Qua phải:** Nhấn và giữ nút "Qua phải" để drone di chuyển sang phải.
- **Xoay trái:** Nhấn và giữ nút "Xoay tròn qua trái" để drone xoay ngược chiều kim đồng hồ.
- **Xoay phải:** Nhấn và giữ nút "Xoay tròn qua phải" để drone xoay theo chiều kim đồng hồ.
- **Tăng ga:** Nhấn và giữ nút "Nâng lên" để drone bay lên cao hơn.
- **Giảm ga:** Nhấn và giữ nút "Hạ xuống" để drone bay xuống thấp hơn.

Điều khiển quadcopter di chuyển bằng các phím trên bàn phím máy tính

- **Tiến lên (Pitch tăng):** Nhấn và giữ phím "I" để drone bay về phía trước.
- **Lùi lại (Pitch giảm):** Nhấn và giữ phím "K" để drone bay về phía sau.
- **Qua trái (Roll giảm):** Nhấn và giữ phím "J" để drone di chuyển sang trái.
- **Qua phải (Roll tăng):** Nhấn và giữ phím "L" để drone di chuyển sang phải.

- **Xoay trái (Yaw giảm):** Nhấn và giữ phím "A" để drone xoay ngược chiều kim đồng hồ.
- **Xoay phải (Yaw tăng):** Nhấn và giữ phím "D" để drone xoay theo chiều kim đồng hồ.
- **Tăng ga (Throttle tăng):** Nhấn và giữ phím "W" để drone bay lên cao hơn.
- **Giảm ga (Throttle giảm):** Nhấn và giữ phím "S" để drone bay xuống thấp hơn.

Lưu đồ giải thuật toàn bộ hệ thống quadcopter:



CHƯƠNG 6: HỆ THỐNG DÒ ĐƯỜNG TỰ ĐỘNG

6.1 Quét bản đồ

Mục tiêu của đồ án này là sử dụng cảm biến Lidar kết nối với Raspberry Pi để quét và tạo bản đồ 2D của môi trường xung quanh, sau đó gửi dữ liệu này về laptop để hiển thị.

6.1.1 Thiết bị và phần mềm sử dụng

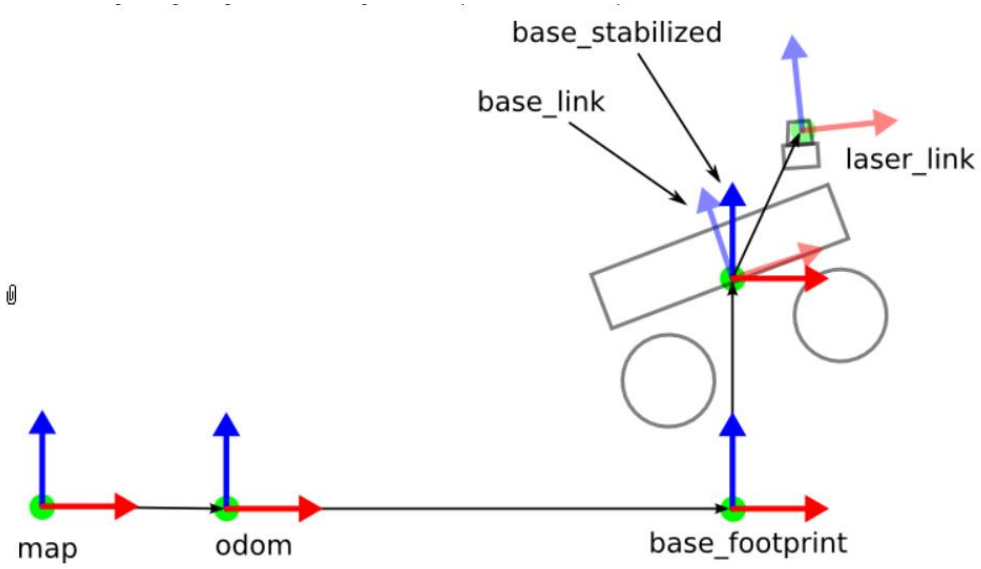
Thiết bị:

Cảm biến Lidar A1M8 kết nối với Raspberry Pi 4 phiên bản 8GB Ram thông qua cổng USB.

Hình 6.1: Các thiết bị sử dụng cho Lidar

Phần mềm sử dụng:

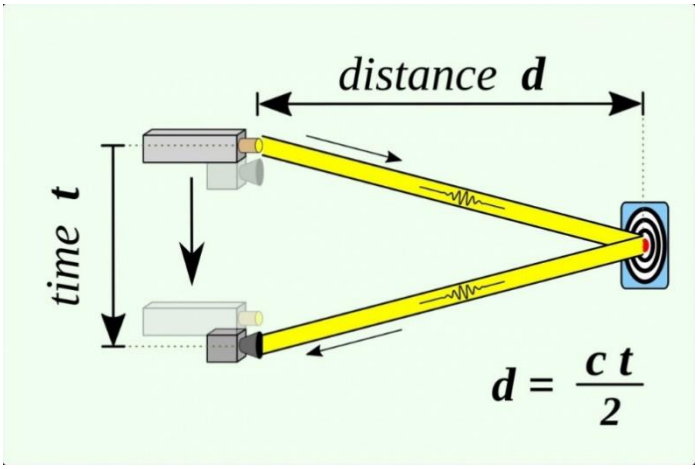
- Raspberry Pi 4 chạy hệ điều hành Ubuntu Server 20.04 và truyền dữ liệu thông qua ROS Noetic
- Thuật toán Hector_slam (Simultaneous Localization and Mapping): là một thuật toán SLAM dựa trên quét laser (Lidar) được phát triển bởi Stefan Kohlbrecher và các cộng sự tại TU Darmstadt. Hector SLAM sử dụng các phép đo quét laser để xây dựng bản đồ môi trường và đồng thời xác định vị trí của robot trong bản đồ đó.



Hình 6.2: Ước tính các tư thế của robot trong Hector SLAM

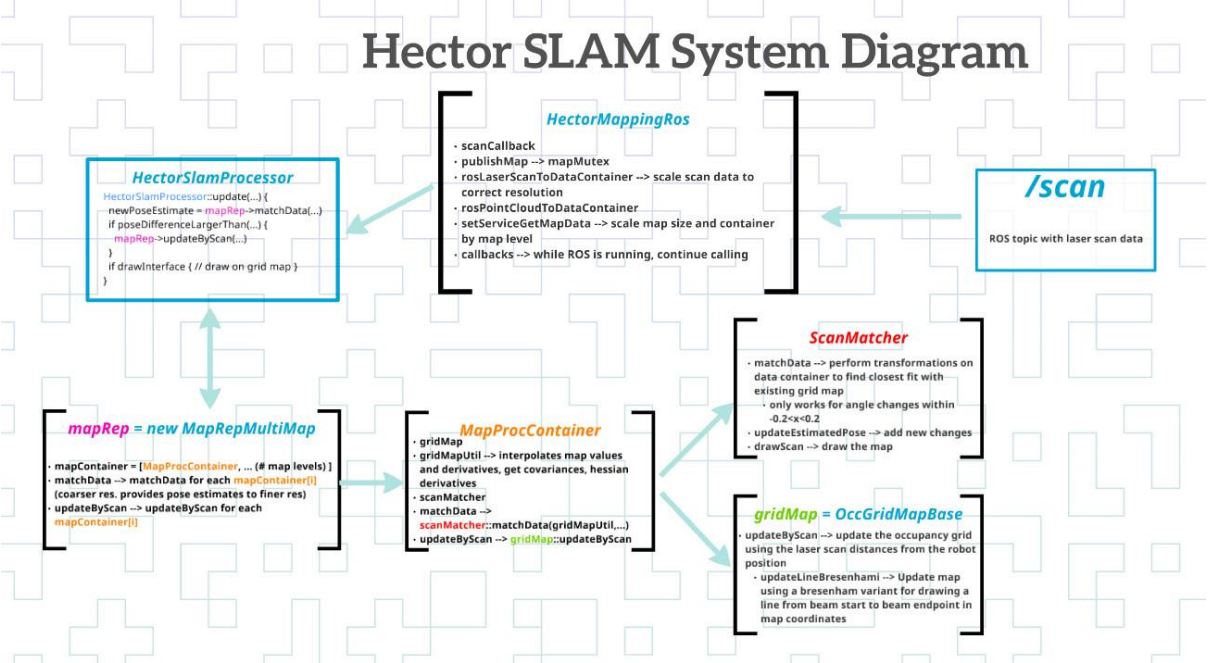
Thuật toán Hector_slam:

- Lidar sẽ phát ra các tia laser và đo khoảng cách với các vật cản dựa trên thời gian phản hồi của tia laser.



Hình 6.3: Công thức tính khoảng cách của Lidar

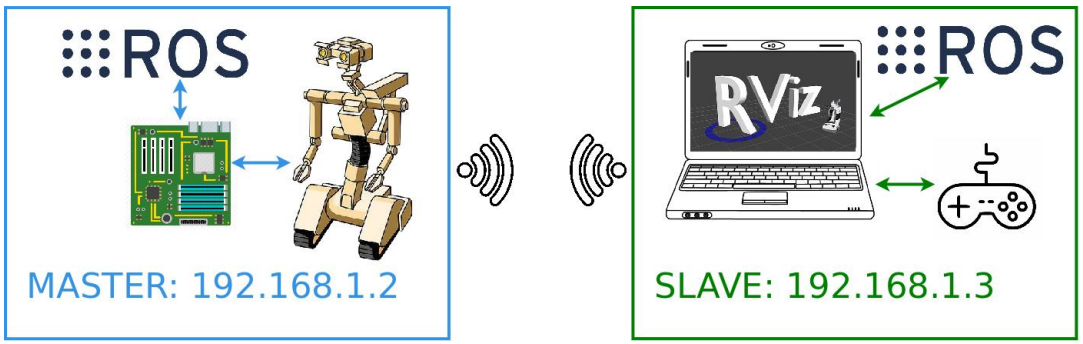
- Tiếp theo sẽ tạo ra một bản đồ ban đầu của môi trường lưu trữ vị trí các vật cản được phát hiện trong không gian.
- Khi robot di chuyển, Hector_slam liên tục cập nhật vị trí của robot bằng cách so sánh các phép đó lidar mới so với bản đồ hiện tại và cập nhật bản đồ mới, các điểm hiện có có thể được điều chỉnh dựa trên thông tin mới. Lặp lại liên tục quá trình này sẽ giúp cho Robot có thể tạo ra bản đồ chính xác.



Hình 6.4: Sơ đồ hệ thống Hector SLAM

6.1.2 Các bước thực hiện

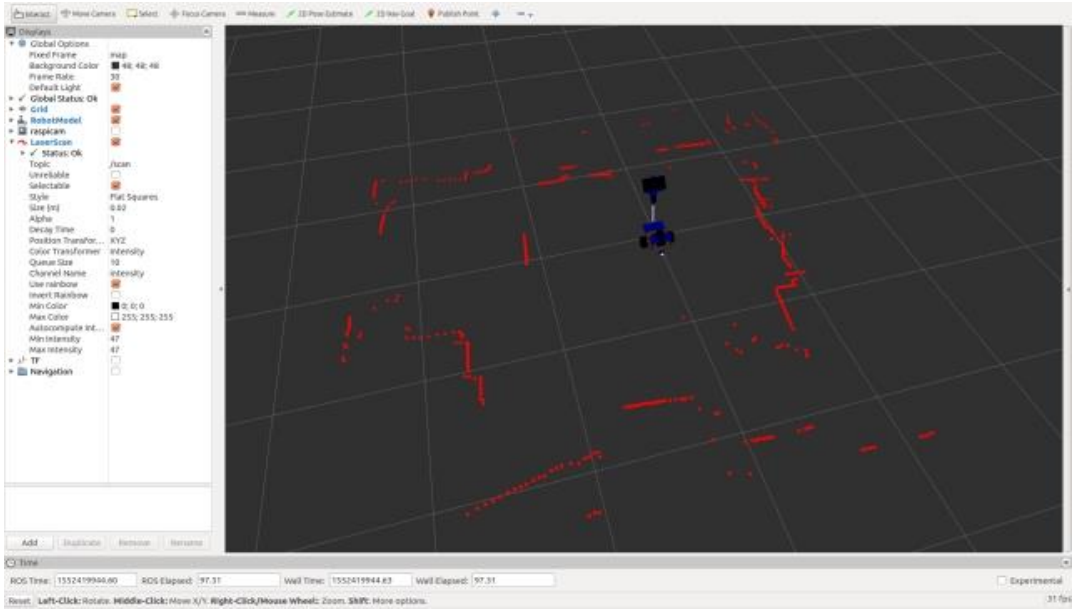
- Bước 1: Thiết lập địa chỉ Raspberry là Master và địa chỉ của Laptop là Slave để có thể gửi dữ liệu từ Raspberry về Laptop, lúc này Raspberry đóng vai trò là Master và Laptop đóng vai trò là Slave
  export ROS_MASTER_URI=http://192.168.129.184:11311
  export ROS_IP=192.168.129.184



Hình 6.5: MASTER và SLAVE

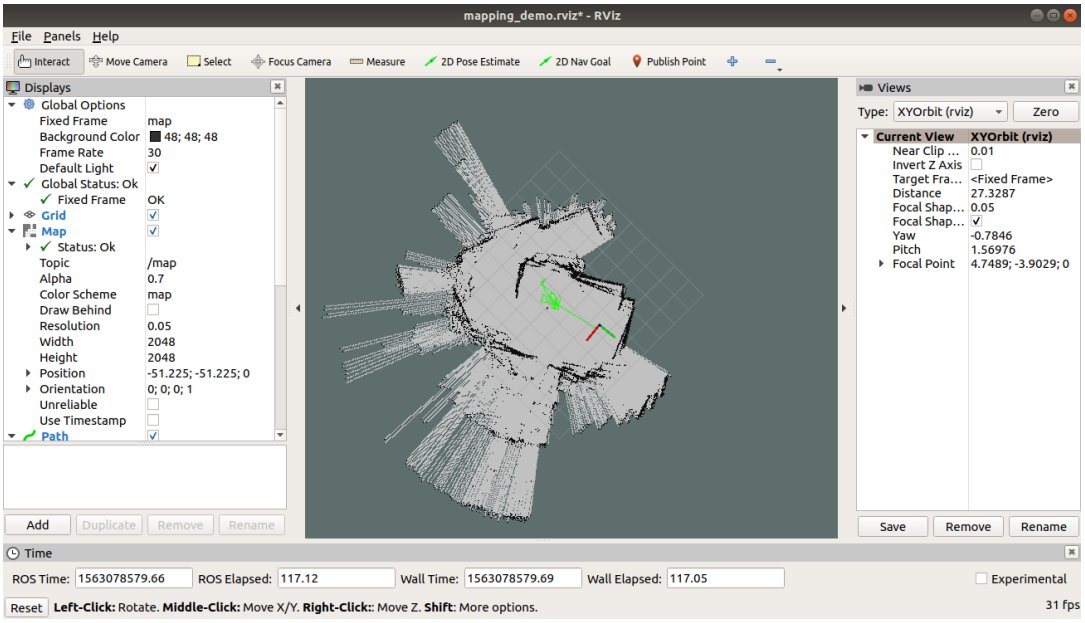
- Bước 2: Khởi động quá trình quét dữ liệu của Lidar đồng thời gửi topic ***/scan*** về cho Alaptop (Bước này giúp ta có thể nhìn thấy được dữ liệu quét được từ môi trường xung quanh hiển thị dưới dạng các chấm đỏ)

roslaunch rplidar_ros rplidar_a1.launch _serial_port:=/dev/ttyUSB0 _serial_baudrate:=115200



Hình 6.6: Lidar bắt đầu quá trình quét

- Bước 3: Khởi chạy công cụ Hector_slam đồng thời gửi topic /map để tiến hành vẽ map: *roslaunch hector_slam_launch tutorial.launch*



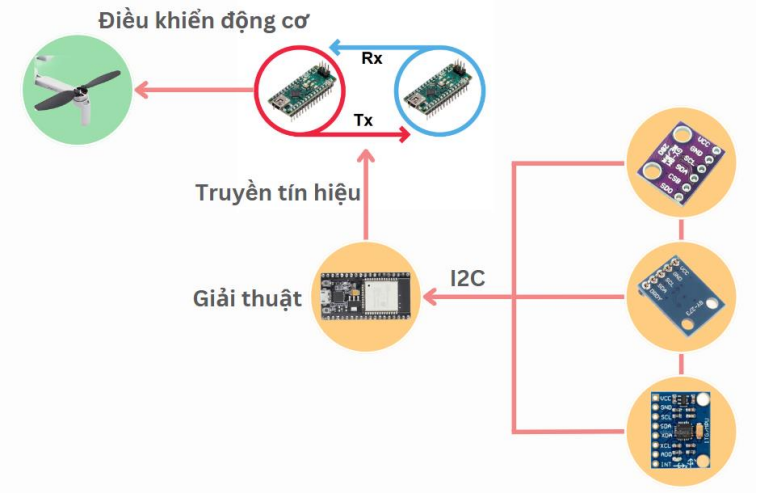
Hình 6.7: Tiến hành vẽ map

6.2 Chế độ tự động bay

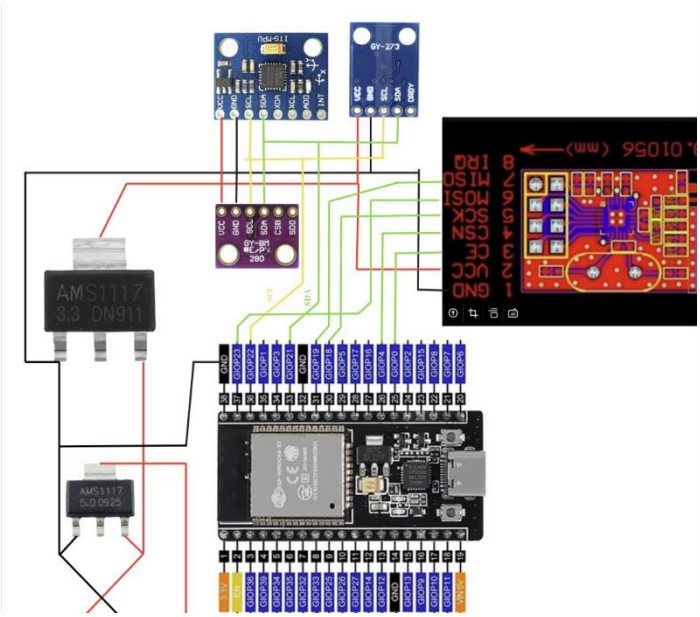
Chế độ tự động bay trong đề tài này được sử dụng để điều khiển quadcopter thông qua việc đọc dữ liệu từ các cảm biến, tính toán các giá trị điều khiển, và truyền tín hiệu điều khiển thông qua bộ thu phát không dây NRF24L01. Việc tự động bay bao gồm các thành phần chính như đọc dữ liệu cảm biến Adafruit HMC5883, GPS và la bàn, tính toán điều khiển PID cho

góc pitch và yaw, thiết lập các kênh điều khiển, và tự động thực hiện các thao tác khởi động và hoạt động của quadcopter.

6.2.1 Phương pháp và kết nối



Hình 6.8: Phương pháp nghiên cứu

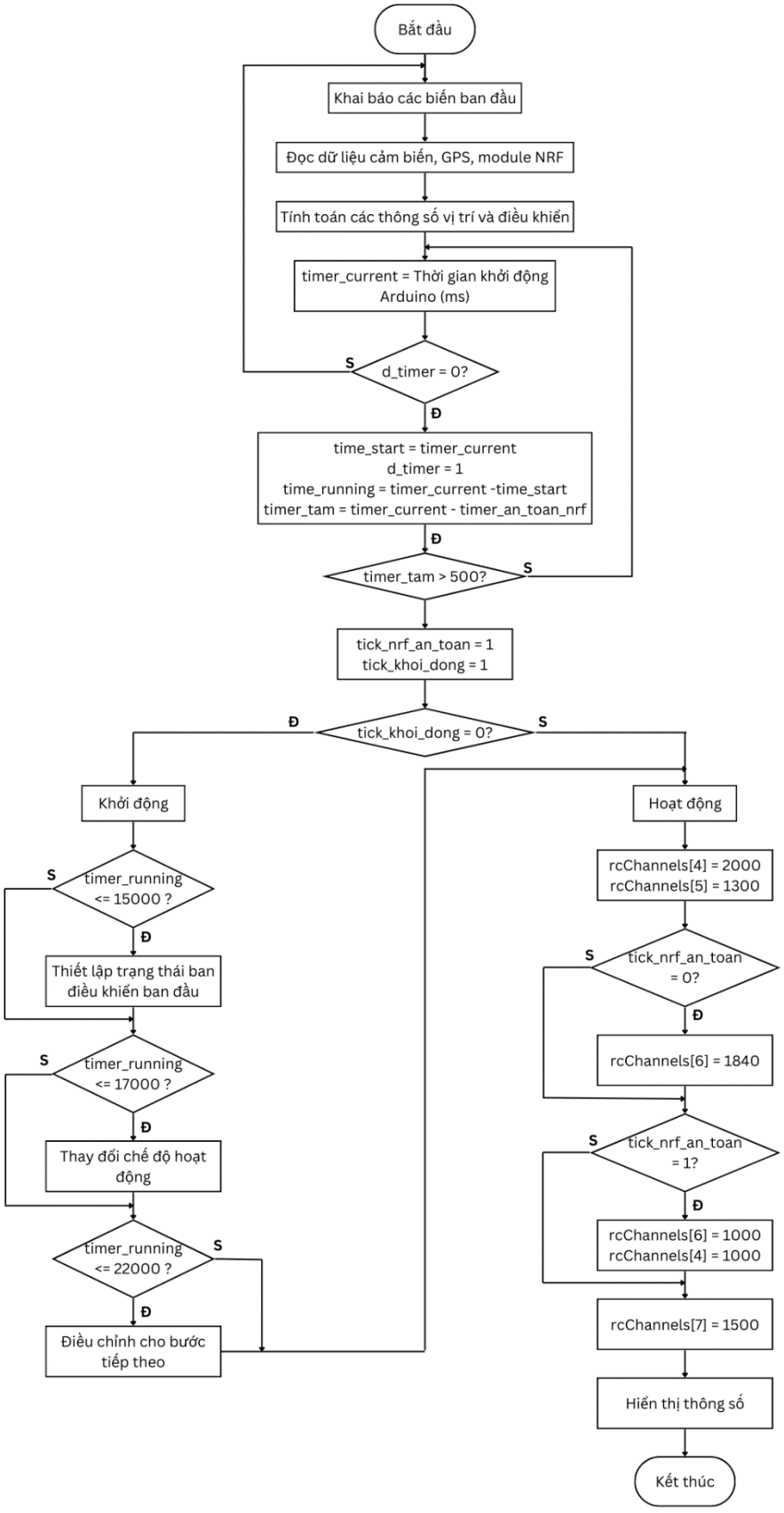


Hình 6.9: Sơ đồ kết nối

6.2.2 Lưu đồ giải thuật

Chương trình bay tự động: Phụ lục 2

Lưu đồ giải thuật chương trình:


Bắt đầu
Khai báo các biến ban đầu
Đọc dữ liệu cảm biến, GPS, module NRF
Tính toán các thông số vị trí và điều khiển
timer_current = Thời gian khởi động Arduino (ms)
S
d_timer = 0?
Đ
time_start = timer_current
d_timer = 1
time_running = timer_current -time_start
timer_tam = timer_current - timer_an_toan_nrf
Đ
timer_tam > 500?
S
tick_nrf_an_toan = 1
tick_khoi_dong = 1
Đ
tick_khoi_dong = 0?
S
Khởi động
S
timer_running <= 15000 ?
Đ
Thiết lập trạng thái ban điều khiển ban đầu
S
timer_running <= 17000 ?
Đ
Thay đổi chế độ hoạt động
S
timer_running <= 22000 ?
Đ
Điều chỉnh cho bước tiếp theo
Hoạt động
rcChannels[4] = 2000
rcChannels[5] = 1300
S
tick_nrf_an_toan = 0?
Đ
rcChannels[6] = 1840
S
tick_nrf_an_toan = 1?
Đ
rcChannels[6] = 1000
rcChannels[4] = 1000
rcChannels[7] = 1500
Hiển thị thông số
Kết thúc

6.2.3 Giải thích sơ lược chương trình

Thư viện và định nghĩa biến

- `RF24.h`, `nRF24L01.h`, `SPI.h`: Thư viện để giao tiếp với module NRF24L01.
- `Wire.h`, `Adafruit_Sensor.h`, `Adafruit_HMC5883_U.h`: Thư viện để giao tiếp với cảm biến la bàn số HMC5883.
- `SoftwareSerial.h`, `TinyGPS++.h`: Thư viện để giao tiếp với GPS qua cổng serial.

Biến toàn cục

- `RXPin`, `TXPin`: Chân RX và TX cho module GPS.
- `gps`, `ss`: Đối tượng GPS và Serial để giao tiếp với GPS.
- `currentPoint`, `MucTieu_lng`, `MucTieu_lat`: Các biến lưu trữ tọa độ hiện tại và mục tiêu.
- `phuong_vi`, `khoang_cach`: Biến lưu phương vị và khoảng cách tới mục tiêu.
- Các biến khác liên quan đến điều khiển PID và điều khiển kênh.

Cấu hình phần cứng trong `setup`

- Khởi tạo giao tiếp với GPS, la bàn số và module NRF24L01.
- Thiết lập chế độ nghe cho NRF24L01 và cấu hình các kênh truyền.

Vòng lặp chính trong `loop`

- Đọc dữ liệu từ la bàn số (`heading_read`) và GPS (`GPS_read`).
- Tính toán điều khiển PID cho pitch (`PID_pitch`) và yaw (`PID_yaw`).
- Thiết lập các kênh điều khiển (`set_chanel`).
- Nhận dữ liệu từ NRF24L01 (`nrf_thu`).
- Quản lý thời gian và chuyển trạng thái hệ thống (khởi động, hoạt động).

Đọc dữ liệu GPS (`GPS_read`)

- Đọc tọa độ hiện tại từ GPS.
- Tính toán phương vị và khoảng cách tới mục tiêu.
- Nếu khoảng cách <= 300 mét, chuyển sang mục tiêu tiếp theo.

Đọc hướng từ la bàn (`heading_read`)

- Đọc dữ liệu từ cảm biến la bàn và tính toán hướng.

Điều khiển PID cho pitch và yaw (`PID_pitch`, `PID_yaw`)

- Tính toán các thành phần P, I, D cho điều khiển pitch và yaw.

Nhận dữ liệu từ NRF24L01 (`nrf_thu`)

- Đọc dữ liệu từ điều khiển từ xa và cập nhật các biến điều khiển.

Các hàm hỗ trợ khác

- `display()`: Hiển thị thông tin lên Serial Monitor.
- `khoi_dong()`, `hoat_dong()`, `landing_1()`, `landing_2()`: Các trạng thái hoạt động của drone.
- `MIN_MAX()`: Giới hạn giá trị trong khoảng cho phép.
- `sbus_xuat()`: Xuất dữ liệu điều khiển qua giao thức S.BUS.

6.3 Hệ thống né vật cản

6.3.1 Hệ thống cảm biến SR04

Cảm biến siêu âm HC-SR04 là một trong những loại cảm biến phổ biến được sử dụng trong các dự án điện tử và robot để đo khoảng cách. Cảm biến này hoạt động dựa trên nguyên lý phản xạ của sóng siêu âm và có khả năng đo khoảng cách từ 2cm đến 400cm với độ chính xác cao.

Thông số kỹ thuật:

- Điện áp hoạt động: 5 VDC
- Dòng điện hoạt động: < 2mA
- Góc quét: < 15 độ
- Tần số phát sóng: 40Khz
- Chân tín hiệu: Echo, Trigger
- Sai số: 0.3cm
- Kích thước: 45mm x 20mm x 15mm

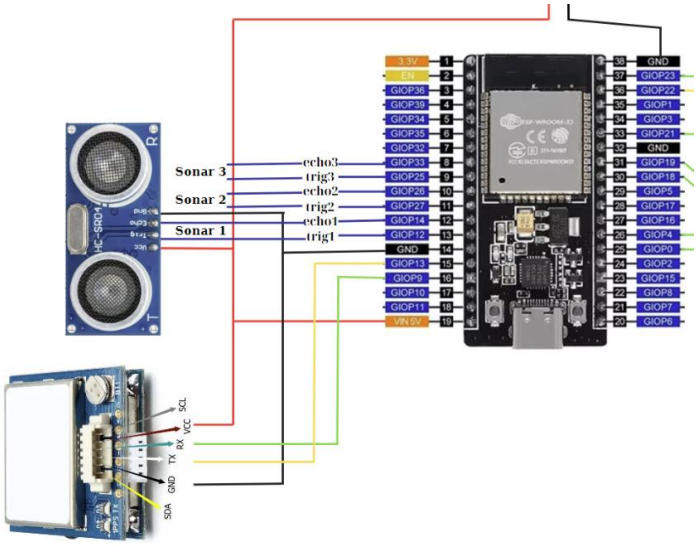
Hình 6.10: Cảm biến siêu âm

Nguyên lý hoạt động:

Để đo khoảng cách từ cảm biến đến vật thể, ta sẽ phát một xung rất ngắn (tối thiểu 10 microSeconds) đến chân Trig. Sau đó, cảm biến sẽ tạo ra 1 xung HIGH ở chân Echo đến khi nhận lại được sóng phản xạ ở Pin này. Chiều rộng của xung sẽ bằng với thời gian sóng siêu âm được phát ra và quay trở lại cảm biến. Từ đó tính được khoảng cách.

Kết nối:

Sử dụng 3 cảm biến siêu âm kết nối với ESP32



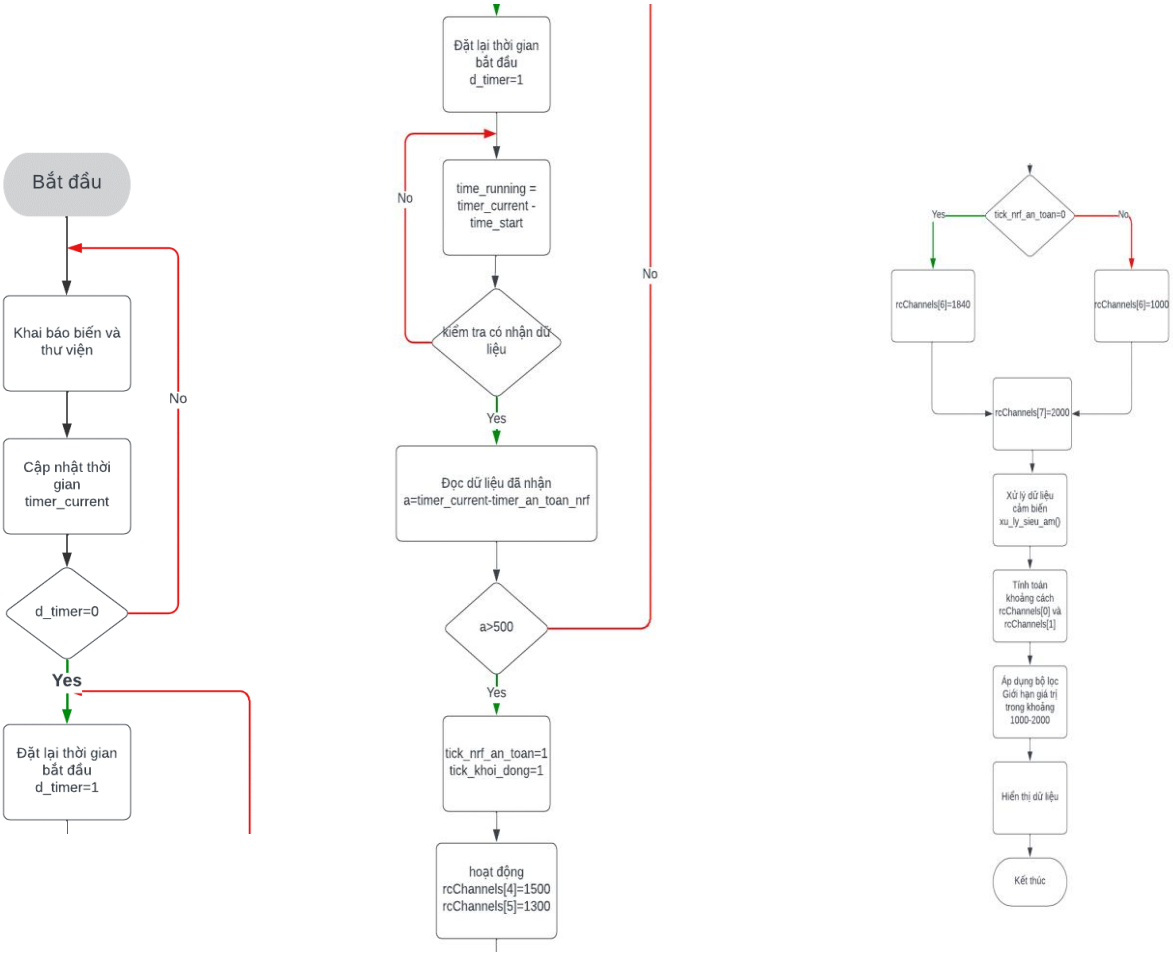
Hình 6.11: Sơ đồ kết nối

6.3.2 Tổng quan hệ thông

Chương trình hệ thống né vật cản: Phụ lục 3

Lưu đồ giải thuật hệ thống né vật cản:

6.3.3 Giải thích chương trình hệ thống



Cách Hoạt Động Của Đoạn Mã

Để hiểu rõ cách hoạt động của đoạn mã này, chúng ta sẽ phân tích từng phần chính trong

mã và cách chúng phối hợp với nhau để điều khiển drone và tránh vật cản.

Khởi Tạo và Thiết Lập Ban Đầu

Trong hàm setup(), chúng ta thực hiện các bước khởi tạo và thiết lập ban đầu cho các thành phần như module NRF24L01, cảm biến siêu âm, và Serial:

- Khởi tạo module NRF24L01:
  - radio.begin(): Bắt đầu sử dụng module NRF24L01.
  - radio.openReadingPipe(1, diaChi): Mở ống dẫn để nhận dữ liệu từ địa chỉ diaChi.
  - radio.setPALevel(RF24_PA_MIN), radio.setChannel(90), radio.setDataRate(RF24_250KBPS): Thiết lập các thông số cho module như mức năng lượng, kênh và tốc độ truyền dữ liệu.
  - radio.startListening(): Bắt đầu lắng nghe tín hiệu điều khiển từ xa.
- Thiết lập cảm biến siêu âm:
  - Sử dụng mảng pingTimer để thiết lập thời gian ping cho các cảm biến siêu âm. Mỗi cảm biến sẽ được kích hoạt sau một khoảng thời gian nhất định.
- Khởi tạo Serial:
  - Serial.begin(115200): Thiết lập tốc độ truyền dữ liệu qua cổng Serial là 115200 baud.

Vòng Lặp Chính

Trong hàm loop(), các công việc chính được thực hiện để điều khiển drone và tránh vật cản:

- Đọc dữ liệu từ module NRF24L01:
  - nrf_thu(): Hàm này đọc dữ liệu điều khiển từ xa thông qua module NRF24L01 và cập nhật các giá trị điều khiển cho drone.
- Xử lý hoạt động chính của drone:
  - hoat_dong(): Hàm này thực hiện các thao tác điều khiển chính cho drone, bao gồm các lệnh bay cơ bản.
- Tính toán khoảng cách từ cảm biến siêu âm:
  - calculate_distance(): Hàm này tính toán khoảng cách từ các cảm biến siêu âm đến các vật cản.
- Xử lý và điều chỉnh giá trị điều khiển dựa trên khoảng cách từ cảm biến:
  - distance_truoc, distance_sau, distance_phai, distance_trai: Các giá trị khoảng cách từ cảm biến siêu âm phía trước, sau, phải, và trái.

- Các giá trị điều khiển của drone (các kênh điều khiển) được điều chỉnh dựa trên các giá trị khoảng cách này để tránh va chạm với vật cản.

- Làm mịn các giá trị điều khiển:

  - rcChannels[0] = ch1_filter.updateEstimate(rcChannels[0]), rcChannels[1] = ch2_filter.updateEstimate(rcChannels[1]): Sử dụng bộ lọc Kalman để làm mịn các giá trị điều khiển.

- Hiển thị các giá trị điều khiển và khoảng cách:

  - display(): Hàm này hiển thị các giá trị điều khiển và khoảng cách từ cảm biến để người điều khiển có thể theo dõi.

Các Hàm Phụ Trợ

Ngoài các hàm chính trong vòng lặp loop(), còn có các hàm phụ trợ để xử lý các tác vụ cụ thể:

- sbusPreparePacket(): Chuẩn bị gói dữ liệu SBUS để gửi đi.
- antoan_ch(): Đảm bảo rằng các giá trị điều khiển nằm trong khoảng an toàn.
- khoi_dong(): Hàm khởi động drone.
- landing_1(), landing_2(): Các hàm hạ cánh drone.
- calculate_distance(): Tính toán khoảng cách từ cảm biến siêu âm.
- echoCheck(): Kiểm tra phản hồi từ cảm biến siêu âm.
- oneSensorCycle(): Xử lý chu kỳ của cảm biến siêu âm.

CHƯƠNG 7: KẾT QUẢ

7.1 Mô hình phần cứng

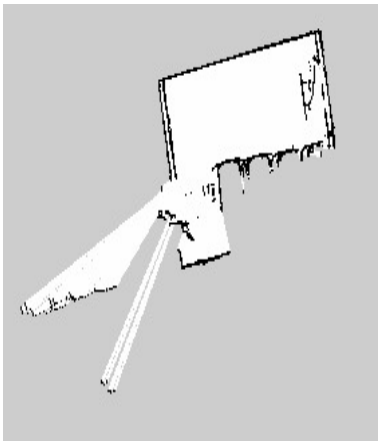
Hình 7.1: Mô hình quadcopter hoàn chỉnh

Bảng 7.1: Thông số mô hình quadcopter hoàn chỉnh

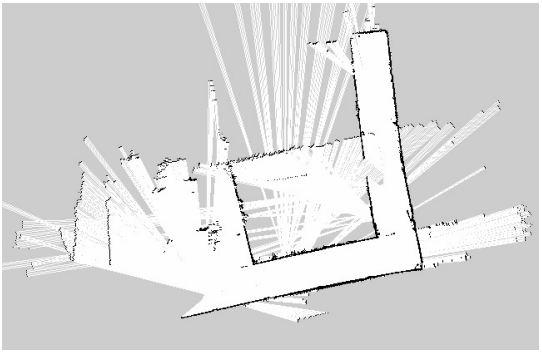
Thông số	Giá trị
Kích cỡ	47x47x24 (cm)
Khối lượng	
Nguồn pin	
Thời gian hoạt động	15 phút liên tục
Phạm vi hoạt động	Bán kính 500m
Hình thức di chuyển	Bay, qua trái, qua phải, xoay, đáp, tiến, lùi
Phương pháp điều khiển	Điều khiển qua laptop và bay tự động

7.2 Lidar quét bản đồ

Nhóm đã thực hiện quét bản đồ thực tế tại 02 địa điểm: Sân thượng khu C và khu vực khoa In và Truyền thông của Trường Đại học Sư phạm Kỹ thuật Thành phố Hồ Chí Minh. Kết quả Lidar đã quét được tường cùng các chướng ngại vật xung quanh khu vực được chọn



Hình 7.2: Bản đồ số 1



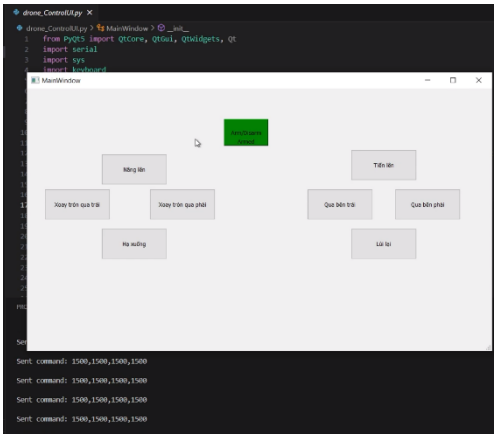
Hình 7.3: Bản đồ số 2

7.3 Hoạt động bay của quadcopter

Nhóm thực hiện điều khiển quadcopter bay trong môi trường bằng laptop. Theo đó, điều khiển drone bay theo các chức năng cơ bản như cất cánh, qua trái, qua phải, đi thẳng, đi lùi, xoay qua trái, xoay qua phải. Cùng lúc theo dõi thông số gửi lên từ laptop để điều khiển quadcopter. Sử dụng các nút nhấn trên giao diện và các nút bàn phím trên laptop. Thử nghiệm chức năng fail-safe.

Hình 7.4: Quadcopter bay trên bãi cỏ

Hình 7.5: Quadcopter bay trong sân trường



Hình 7.6: Giao diện điều khiển cùng các thông số gửi để điều khiển drone

7.4 Quadcopter bay né vật cản

Để điều khiển và thử nghiệm chức năng né vật cản nhóm quyết định chọn bãi cỏ rộng và vắng người để tránh các rủi ro khi bay. Chuẩn bị các tấm bìa carton và tấm xốp để cho cảm biến siêu âm nhận diện là vật cản. Sau đó, nhóm điều khiển drone bay ở độ cao trung bình khoảng 2m. Thực hiện đưa các tấm bìa lại gần từ từ, đến khoảng cách nhận biết đã đặt drone sẽ tự động di chuyển theo hướng ngược lại. Thử nghiệm trên 3 phía trái, phải, trước lần lượt drone sẽ tự động di chuyển theo các hướng phải, trái, sau. Khi có vật cản ở cả 2 phía trái, phải drone sẽ ở di chuyển ở khoảng cách an toàn giữa hai phía. Các hình dưới mô tả quá trình bay né vật cản.

Quadcopter di chuyển sang phải khi phát hiện vật cản ở phía trái

Hình 7.7: Quadcopter bay sang phải

Quadcopter di chuyển sang trái khi phát hiện vật cản ở phía phải

Hình 7.8: Quadcopter bay sang trái

Quadcopter di chuyển lùi về sau khi phát hiện vật cản ở phía trước

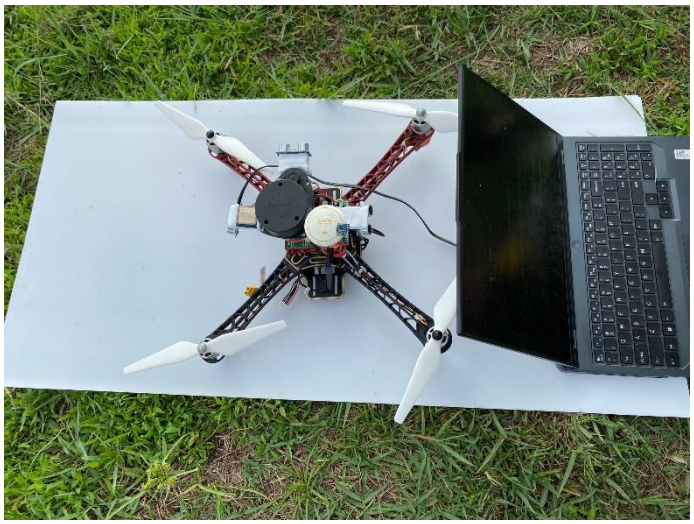
Hình 7.9: Quadcopter bay lùi

Quadcopter ở khoảng cách an toàn khi cả hai phía trái phải đều có vật cản

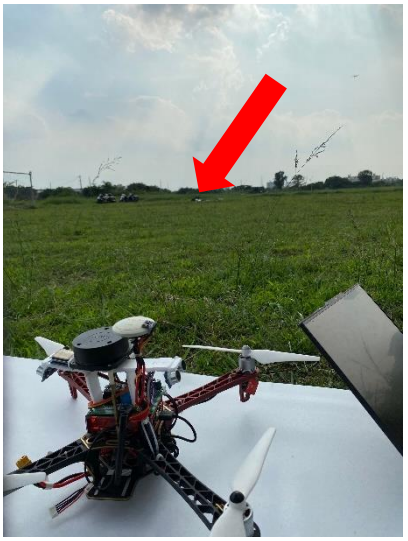
Hình 7.10: Quadcopter ở vị trí an toàn

7.5 Quadcopter bay tự động

Tại địa điểm bãi cỏ trên, nhóm cập nhật 2 điểm tọa độ theo vĩ độ và kinh độ. Đặt quadcopter tại điểm bắt đầu có tọa độ (10.821828;106.616163) và chọn điểm mục tiêu có tọa độ (10.821972;106.616667). Các tọa độ lấy theo GPS đặt 2 tấm xốp làm đặc điểm nhận dạng. Hình phía dưới mô tả quá trình nạp chương trình và lấy tọa độ từ GPS của quadcopter.



Hình 7.11: Nhận tọa độ điểm



Hình 7.12: Xác định điểm mục tiêu

Bắt đầu quá trình bay tự động.

Hình 7.13: Bay từ điểm đầu

Hình 7.14: Bay đến điểm mục tiêu

7.6 Quadcopter bay dò đường tự động cùng né vật cản

Cài đặt quadcopter bay tự động từ 2 điểm xác định như trên. Nhóm thực hiện đặt các xe máy cùng bìa carton làm chướng ngại vật trên đường đi để quadcopter nhận biết và thực hiện quá trình né các chướng ngại vật nhưng đồng thời đáp ứng yêu cầu tự động dò đường đến điểm mục tiêu. Hình ảnh dưới mô tả các chướng ngại vật và quá trình quadcopter bay tự động.

Hình 7.15: Tự động bay né vật cản

Hình 7.16: Tự động đổi hướng

CHƯƠNG 8: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

8.1 Kết quả đạt được

Sau quá trình nghiên cứu và thực hiện đề tài: "*Nghiên cứu, thiết kế, chế tạo mô hình và hệ thống điều khiển cân bằng quadcopter dò đường tự động*" những kết quả nhóm đã đạt được:

- Hiểu được kiến thức cơ bản về quadcopter.
- Thi công, lắp ráp phần cứng quadcopter.
- Hoàn thành quét lidar và vẽ bản đồ.
- Hoàn thành điều khiển bay quadcopter né vật cản và dò đường tự động.
- Điều khiển quadcopter ở môi trường ngoài trời.

8.2 Kết luận

Ưu điểm:

- Mô hình quadcopter nhỏ gọn, giá thành rẻ, đáp ứng mục tiêu đề tài, dễ tiếp cận cho nhiều đối tượng.
- Ứng dụng công nghệ Lidar để quét và vẽ bản đồ, phục vụ nhận diện môi trường xung quanh.
- Quadcopter hoàn thiện chức năng điều khiển bay tự động, có khả năng né tránh chướng ngại vật và dò đường.
- Quadcopter có khả năng hoạt động tốt trong môi trường ngoài trời, mở rộng phạm vi ứng dụng thực tế.
- Sản phẩm hoàn thiện là cơ sở cho các nghiên cứu và phát triển về các ứng dụng khác của quadcopter trong tương lai.

Hạn chế:

- Lidar có phạm vi hoạt động ngắn, chỉ phù hợp cho ứng dụng 2D và môi trường đơn giản.
- Giới hạn về khả năng xử lý dữ liệu trong thời gian thực và thời gian hoạt động.
- Quadcopter có thể gặp khó khăn trong điều kiện thời tiết không thuận lợi.
- Hiệu suất phụ thuộc vào chất lượng của các cảm biến và linh kiện sử dụng.
- Hạn chế về tài nguyên nguồn lực để phát triển đầy đủ các tính năng mong muốn.

8.3 Hướng phát triển

- Cải tiến, phát triển hướng đến mục tiêu thương mại hóa sản phẩm.
- Nâng cấp công nghệ Lidar để vẽ bản đồ thực tế.
- Cải thiện phần mềm điều khiển để tăng tính hiệu quả.
- Phát triển ứng dụng cho các lĩnh vực khác như nông nghiệp, giao thông, giám sát an ninh.
- Nâng cấp các thành phần phần cứng để tăng cường hiệu suất và độ bền hệ thống.

- Hợp tác với các tổ chức nghiên cứu và công ty công nghệ để tiếp tục phát triển và hoàn thiện sản phẩm.

TÀI LIỆU THAM KHẢO

[1] Lâm Ngọc Tâm, "Thiết kế và chế tạo mô hình máy bay - Quadcopter", *Trường Đại học Bách Khoa, Đại học Đà Nẵng.*

[2] Nguyễn Hùng Thái Sơn, Võ Nguyên Phúc, "Thiết kế và thi công mô hình bay Quadcopter", *Journal of Science of Lac Hong University.*

[3] Vũ Văn Tiến, "Điều khiển PID ứng dụng cho điều khiển thiết bị bay loại bốn động cơ", *Trường Đại học Cộng nghệ thông tin và Truyền thông, Đại học Thái Nguyên.*

[4] A.K. Bhatia, J. Jiang, Z. Zhen, N. Ahmed, and A. Rohra, "Projection Modification Based Robust Adaptive Backstepping Control for Multipurpose Quadcopter UAV", College of Automation Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China, 2019.

[5] Lindsay Kleeman, "Understanding and Applying Kalman Filtering", *Department of Electrical and Computer Systems Engineering, Monash University.*

[6] Mayur Tank, "Autonomous control of quadcopter UAV", *Department of Instrumentation and Control Engineering, Institute of Technology, Nirma University.*

[7] Matteo Ragni, "Mechanical Model for Quadcopter UAV", *Università degli Studi di Trento.*

[8] Quan Quan, "Introduction to Multicopter Design and Control", *Department of Automatic Control, Beihang University.*

PHỤ LỤC 1

Chương trình giao diện điều khiển:

```
from PyQt5 import QtCore, QtGui, QtWidgets, Qt
import serial
import sys
import keyboard

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)
        self.serial_connected = False
        try:
            self.serial = serial.Serial('COM5', 115200, timeout=1)  # Adjust 'COM3' to your actual
COM port
            self.serial_connected = True
        except serial.SerialException as e:
            self.show_error_message(str(e))

        self.arm_timer = QtCore.QTimer()  # Khởi tạo arm_timer ở đây
        self.arm_timer.timeout.connect(self.reset_controls)

        self.button_timers = {}  # Lưu trữ timer cho từng nút
        self.is_sending_commands = {}  # Lưu trữ trạng thái gửi lệnh cho từng nút
        self.button_commands = {  # Dictionary để lưu trữ lệnh điều khiển
            self.pushButton_1: "1700,1500,1500,1500\n",  # Tiến lên
            self.pushButton_2: "1500,1700,1500,1500\n",  # Qua phải
            self.pushButton_3: "1500,1300,1500,1500\n",  # Qua trái
            self.pushButton_4: "1300,1500,1500,1500\n",  # Lùi lại
            self.pushButton_5: "1500,1500,1500,1300\n",  # Xoay trái
            self.pushButton_6: "1500,1500,1700,1500\n",  # Nâng lên
            self.pushButton_7: "1500,1500,1300,1500\n",  # Hạ xuống
            self.pushButton_8: "1500,1500,1500,1700\n",  # Xoay phải
        }
        for button in self.button_commands.keys():
            self.is_sending_commands[button] = False
            button.pressed.connect(lambda b=button: self.start_sending_command(b))
            button.released.connect(lambda b=button: self.stop_sending_command(b))

        # Khởi tạo giá trị mặc định cho các trục
        self.roll = 1500
```

```
        self.pitch = 1500
        self.throttle = 1500
        self.yaw = 1500
        self.is_armed = False

        # Tạo timer để gửi lệnh liên tục
        self.timer = QtCore.QTimer()
        self.timer.timeout.connect(self.send_current_command)
        self.timer.start(50)  # Gửi lệnh mỗi 100ms / 10Hz

    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1062, 600)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")

        # Khởi tạo các nút nhấn
        self.pushButton_1 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_1.setGeometry(QtCore.QRect(740, 140, 150, 70))
        self.pushButton_1.setObjectName("pushButton")

        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(840, 230, 150, 70))
        self.pushButton_2.setObjectName("pushButton_2")

        self.pushButton_3 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_3.setGeometry(QtCore.QRect(640, 230, 150, 70))
        self.pushButton_3.setObjectName("pushButton_3")

        self.pushButton_4 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_4.setGeometry(QtCore.QRect(740, 320, 150, 70))
        self.pushButton_4.setObjectName("pushButton_4")

        self.pushButton_5 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_5.setGeometry(QtCore.QRect(40, 230, 150, 70))
        self.pushButton_5.setObjectName("pushButton_5")

        self.pushButton_6 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_6.setGeometry(QtCore.QRect(170, 150, 150, 70))
        self.pushButton_6.setObjectName("pushButton_6")
```

```
        self.pushButton_7 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_7.setGeometry(QtCore.QRect(170, 320, 150, 70))
        self.pushButton_7.setObjectName("pushButton_7")

        self.pushButton_8 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_8.setGeometry(QtCore.QRect(280, 230, 150, 70))
        self.pushButton_8.setObjectName("pushButton_8")

        # Dictionary để lưu trữ lệnh điều khiển

        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)

        # Kết nối các nút bấm với hàm xử lý
        #         self.pushButton_1.clicked.connect(lambda         checked=False: self.handle_button_press(self.pushButton_1))
        #         self.pushButton_2.clicked.connect(lambda         checked=False: self.handle_button_press(self.pushButton_2))
        #         self.pushButton_3.clicked.connect(lambda         checked=False: self.handle_button_press(self.pushButton_3))
        #         self.pushButton_4.clicked.connect(lambda         checked=False: self.handle_button_press(self.pushButton_4))
        #         self.pushButton_5.clicked.connect(lambda         checked=False: self.handle_button_press(self.pushButton_5))
        #                                           self.pushButton_6.clicked.connect(lambda: self.handle_button_press(self.pushButton_6))
        #         self.pushButton_7.clicked.connect(lambda         checked=False: self.handle_button_press(self.pushButton_7))
        #         self.pushButton_8.clicked.connect(lambda         checked=False: self.handle_button_press(self.pushButton_8))  # Bỏ qua tham số checked

        # Thêm nút nhấn "Arm/Disarm"
        self.armButton = QtWidgets.QPushButton(self.centralwidget)
        self.armButton.setGeometry(QtCore.QRect(450, 70, 101, 61))  # Vị trí tùy chỉnh
        self.armButton.setObjectName("armButton")
        self.armButton.setText("Arm/Disarm")

        self.armButton.clicked.connect(self.arm_drone)

        # Thêm QLabel để hiển thị trạng thái arm
        self.armLabel = QtWidgets.QLabel(self.centralwidget)
```

```
        self.armLabel.setGeometry(QtCore.QRect(450, 90, 101, 61))  # Vị trí tùy chỉnh
        self.armLabel.setObjectName("armLabel")
        self.armLabel.setText("Disarmed")
        self.armButton.setStyleSheet("background-color: red")  # Đặt màu đỏ khi Disarmed
        self.armLabel.setAlignment(QtCore.Qt.AlignCenter)

        self.menubar.setGeometry(QtCore.QRect(0, 0, 583, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.pushButton_1.setText(_translate("MainWindow", "Tiến lên"))
        self.pushButton_2.setText(_translate("MainWindow", "Qua bên phải"))
        self.pushButton_3.setText(_translate("MainWindow", "Qua bên trái"))
        self.pushButton_4.setText(_translate("MainWindow", "Lùi lại"))
        self.pushButton_5.setText(_translate("MainWindow", "Xoay tròn qua trái"))
        self.pushButton_6.setText(_translate("MainWindow", "Nâng lên"))
        self.pushButton_7.setText(_translate("MainWindow", "Hạ xuống"))
        self.pushButton_8.setText(_translate("MainWindow", "Xoay tròn qua phải"))

    def show_error_message(self, message):
        msg = QtWidgets.QMessageBox()
        msg.setIcon(QtWidgets.QMessageBox.Critical)
        msg.setText("Error")
        msg.setInformativeText(message)
        msg.setWindowTitle("Error")
        msg.exec_()

    def arm_drone(self):
        if self.is_armed == True:
            self.roll = 1050
            self.pitch = 1050
            self.throttle = 1050
```

```
            self.yaw = 1950
            self.is_armed = False
            self.send_current_command()
            self.arm_timer.start(1500)
            self.armLabel.setText("Disarmed")
            self.armButton.setStyleSheet("background-color: red")  # Đặt màu đỏ khi Disarmed
            QtCore.QTimer.singleShot(1500, self.update_disarm_label)

        else:
            self.roll = 1050
            self.pitch = 1050
            self.throttle = 1050
            self.yaw = 1950
            self.is_armed = True
            self.send_current_command()
            self.arm_timer.start(1500)
            self.armLabel.setText("Armed")
            self.armButton.setStyleSheet("background-color: green")  # Đặt màu xanh khi Armed


            # Tạo timer để hiển thị trạng thái "Armed" sau 1.5 giây
            QtCore.QTimer.singleShot(1500, self.update_arm_label)

    def update_arm_label(self):
        self.armLabel.setText("Armed")  # Cập nhật trạng thái arm hiển thị lên giao diện

    def update_disarm_label(self):
        self.armLabel.setText("Disarmed")  # Cập nhật trạng thái disarmed hiển thị lên giao diện

    def reset_controls(self):
        self.roll = 1500
        self.pitch = 1500
        self.throttle = 1500
        self.yaw = 1500
        self.send_current_command()
        self.arm_timer.stop()

    def start_sending_command(self, button):
        self.is_sending_commands[button] = True
        self.button_timers[button] = QtCore.QTimer()
        self.button_timers[button].timeout.connect(lambda: self.gui_lenh(button))
```

```
    self.button_timers[button].start(5)

def stop_sending_command(self, button):
    self.is_sending_commands[button] = False
    self.button_timers[button].stop()
    self.reset_controls()

def gui_lenh(self, button=None):
    if button:  # Nếu có nút được nhấn giữ
        command_str = self.button_commands.get(button)
        if command_str and self.serial_connected and self.is_sending_commands[button]:
            try:
                self.serial.write(command_str.encode())
                print(f"Sent command: {command_str}")
            except serial.SerialException as e:
                self.show_error_message(str(e))
    else:  # Nếu không có nút nào được nhấn giữ, gửi lệnh mặc định
        self.send_command(self.roll, self.pitch, self.throttle, self.yaw)

def send_command(self, roll, pitch, throttle, yaw):
    command = f"{roll},{pitch},{throttle},{yaw}\n"
    if self.serial_connected:
        try:
            self.serial.write(command.encode())
            print(f"Sent command: {command}")
        except serial.SerialException as e:
            self.show_error_message(str(e))
            print(f"Failed to send command: {command}")
    else:
        self.show_error_message("Serial connection not established.")
        print(f"Failed to send command (serial not connected): {command}")

def send_current_command(self):
    self.send_command(self.roll, self.pitch, self.throttle, self.yaw)

def handle_button_press(self, command, button):
    self.send_command(command)
    self.flash_button(button)

def keyPressEvent(self, event):
    key = event.key()
```

```
        if keyboard.is_pressed('w'):
            self.throttle = 1700
        if keyboard.is_pressed('s'):
            self.throttle = 1300
        if keyboard.is_pressed('a'):
            self.yaw = 1300
        if keyboard.is_pressed('d'):
            self.yaw = 1700
            self.yaw = min(2000, self.yaw)  # Giới hạn yaw
        if keyboard.is_pressed('i'):
            self.pitch = 1700
            self.pitch = min(2000, self.pitch)
        if keyboard.is_pressed('k'):
            self.pitch = 1300
            self.pitch = max(1000, self.pitch)
        if keyboard.is_pressed('l'):
            self.roll = 1700
            self.roll = min(2000, self.roll)
        if keyboard.is_pressed('j'):
            self.roll = 1300
            self.roll = max(1000, self.roll)

    def keyReleaseEvent(self, event):
        # Trả các trục về giá trị trung tính khi nhả phím
        self.roll = 1500
        self.pitch = 1500
        self.throttle = 1500
        self.yaw = 1500

    def flash_button(self, button):
        original_color = button.palette().color(QtGui.QPalette.Button)
        button.setStyleSheet("background-color: yellow")
        QtCore.QTimer.singleShot(200, lambda: button.setStyleSheet(f"background-color:
{original_color.name()}"))

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = MainWindow()
    MainWindow.show()
    sys.exit(app.exec_())
```

PHỤ LỤC 2

Chương trình bay tự động:

```
#include <RF24.h>
#include <nRF24L01.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
#include <SoftwareSerial.h>
#include <TinyGPS++.h>
#define pi M_PI

static const int RXPin = 7, TXPin = 6;
static const uint32_t GPSBaud = 9600;
TinyGPSPlus gps;
SoftwareSerial ss(RXPin, TXPin);

static double currentPoint[2];
static double MucTieu_lng[5];
static double MucTieu_lat[5];
double phuong_vi;
unsigned long khoang_cach;
bool t=0,xac_nhan_tin_hieu_gps=0,danh_dau_start=0;
int n_muc_tieu=0;
int lenh_hoat_dong;

Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
float headingDegrees, heading, declinationAngle;


float k_p_roll=100,k_i_roll=0,k_d_roll=0;
float k_p_pitch=2,k_i_pitch=0,k_d_pitch=0;
float k_p_yaw=5,k_i_yaw=0,k_d_yaw=0;
float k_p_throttle=100,k_i_throttle=0,k_d_throttle=0;
float P_roll,I_roll,D_roll,E_roll,E_roll_0;
float P_pitch,I_pitch,D_pitch,E_pitch,E_pitch_0;
float P_yaw,I_yaw,D_yaw,E_yaw,E_yaw_0;
float P_throttle,I_throttle,D_throttle,E_throttle,E_throttle_0;
int   U_roll, U_pitch, U_yaw, U_throttle;
int ch1_roll_PWM, ch2_pitch_PWM,ch3_throttle_PWM, ch4_yaw_PWM;
int gh_300=300;
float goc_mong_muon;



#define RC_CHANNEL_MIN 1000
#define RC_CHANNEL_MAX 2000
```

```
#define SBUS_MIN_OFFSET 173
#define SBUS_MID_OFFSET 992
#define SBUS_MAX_OFFSET 1811
#define SBUS_CHANNEL_NUMBER 16
#define SBUS_PACKET_LENGTH 25
#define SBUS_FRAME_HEADER 0x0f
#define SBUS_FRAME_FOOTER 0x00
#define SBUS_FRAME_FOOTER_V2 0x04
#define SBUS_STATE_FAILSAFE 0x08
#define SBUS_STATE_SIGNALLOSS 0x04
#define SBUS_UPDATE_RATE 15 //ms

#define SIGNAL_TIMEOUT 100 //Signal timeout in milli seconds
#define LEDPin 2
unsigned long lastRecvTime = 0;
RF24 radio(9,10);//CE & CSN
const byte diaChi[6]="94147";

struct PacketData {
          int ch1;
          int ch2;
          int ch3;
          int ch4;
          int ch5;
          int ch6;
          int ch7;
          int ch8;
  float lat;
  float lon;
  float lenh;
};
PacketData receiverData;



void sbusPreparePacket(uint8_t packet[], int channels[], bool isSignalLoss, bool isFailsafe){

   static int output[SBUS_CHANNEL_NUMBER] = {0};

   /*
    * Map 1000-2000 with middle at 1500 chanel values to
    * 173-1811 with middle at 992 S.BUS protocol requires
    */
   for (uint8_t i = 0; i < SBUS_CHANNEL_NUMBER; i++) {
      output[i] = map(channels[i], RC_CHANNEL_MIN, RC_CHANNEL_MAX,
SBUS_MIN_OFFSET, SBUS_MAX_OFFSET);
   }
```

```
    uint8_t stateByte = 0x00;
    if (isSignalLoss) {
        stateByte |= SBUS_STATE_SIGNALLOSS;
    }
    if (isFailsafe) {
        stateByte |= SBUS_STATE_FAILSAFE;
    }
    packet[0] = SBUS_FRAME_HEADER; //Header

    packet[1] = (uint8_t) (output[0] & 0x07FF);
    packet[2] = (uint8_t) ((output[0] & 0x07FF)>>8 | (output[1] & 0x07FF)<<3);
    packet[3] = (uint8_t) ((output[1] & 0x07FF)>>5 | (output[2] & 0x07FF)<<6);
    packet[4] = (uint8_t) ((output[2] & 0x07FF)>>2);
    packet[5] = (uint8_t) ((output[2] & 0x07FF)>>10 | (output[3] & 0x07FF)<<1);
    packet[6] = (uint8_t) ((output[3] & 0x07FF)>>7 | (output[4] & 0x07FF)<<4);
    packet[7] = (uint8_t) ((output[4] & 0x07FF)>>4 | (output[5] & 0x07FF)<<7);
    packet[8] = (uint8_t) ((output[5] & 0x07FF)>>1);
    packet[9] = (uint8_t) ((output[5] & 0x07FF)>>9 | (output[6] & 0x07FF)<<2);
    packet[10] = (uint8_t) ((output[6] & 0x07FF)>>6 | (output[7] & 0x07FF)<<5);
    packet[11] = (uint8_t) ((output[7] & 0x07FF)>>3);
    packet[12] = (uint8_t) ((output[8] & 0x07FF));
    packet[13] = (uint8_t) ((output[8] & 0x07FF)>>8 | (output[9] & 0x07FF)<<3);
    packet[14] = (uint8_t) ((output[9] & 0x07FF)>>5 | (output[10] & 0x07FF)<<6);
    packet[15] = (uint8_t) ((output[10] & 0x07FF)>>2);
    packet[16] = (uint8_t) ((output[10] & 0x07FF)>>10 | (output[11] & 0x07FF)<<1);
    packet[17] = (uint8_t) ((output[11] & 0x07FF)>>7 | (output[12] & 0x07FF)<<4);
    packet[18] = (uint8_t) ((output[12] & 0x07FF)>>4 | (output[13] & 0x07FF)<<7);
    packet[19] = (uint8_t) ((output[13] & 0x07FF)>>1);
    packet[20] = (uint8_t) ((output[13] & 0x07FF)>>9 | (output[14] & 0x07FF)<<2);
    packet[21] = (uint8_t) ((output[14] & 0x07FF)>>6 | (output[15] & 0x07FF)<<5);
    packet[22] = (uint8_t) ((output[15] & 0x07FF)>>3);

    packet[23] = stateByte; //Flags byte
    packet[24] = SBUS_FRAME_FOOTER; //Footer
}

uint8_t sbusPacket[SBUS_PACKET_LENGTH];
int rcChannels[SBUS_CHANNEL_NUMBER];
int rcChannels_in[SBUS_CHANNEL_NUMBER];
uint32_t sbusTime = 0;

unsigned long timer_curent, time_start, time_runing, timer_an_toan_nrf;
bool d_timer=0, tick_khoi_dong=0,   tick_nrf_an_toan=0, tick_gps=0;

void setup() {
        radio.begin();
```

```
  ss.begin(GPSBaud);
  if(!mag.begin())
  {
    while(1);
  }
  radio.openReadingPipe(1,diaChi);// mặc định đường truyền TX đã là 0 nên RX=1
  radio.setPALevel(RF24_PA_MIN);// cài đặt bộ khuếch đại công suất
  radio.setChannel(90);//lưu ý tx và rx phải cùng kênh(0-124)
  radio.setDataRate(RF24_250KBPS);
  radio.startListening();//cài đặt là RX

    for (uint8_t i = 0; i < SBUS_CHANNEL_NUMBER; i++) {
        rcChannels[i] = 1500;
    }
  pinMode(LEDPin, OUTPUT);
  // Serial.begin(100000, SERIAL_8E2);
  Serial.begin(115200);
}

void loop() {
  heading_read();
  GPS_read();
  PID_pitch();
  PID_yaw();
  set_chanel();
  nrf_thu();
  timer_curent=millis();
  if(d_timer=0){
    time_start=timer_curent;
    d_timer=1;
  }
  time_runing=  timer_curent - time_start;
  if(timer_curent - timer_an_toan_nrf>500){
  tick_nrf_an_toan=1;
  tick_khoi_dong = 1;
  }
  if(tick_khoi_dong==0){
    khoi_dong();
  }else{
    hoat_dong();
  }
  // else if(time_runing<=50000){
  //   landing_1();
  // }else if(time_runing>=50000){
  //   landing_2();
  // }
  rcChannels[4]=2000;
```

```
  rcChannels[5]=1300;

  if(tick_nrf_an_toan==0){  rcChannels[6]=1840; }
  else if(tick_nrf_an_toan==1){rcChannels[6]=1000; rcChannels[4]=1000;}
  rcChannels[7]=1500;
  display();
  // sbus_xuat();

}

void GPS_read(){
  while(ss.available() > 0) {
    gps.encode(ss.read());
    if (gps.location.isUpdated() ) {
      currentPoint[0] =  gps.location.lat();
      currentPoint[1] =  gps.location.lng();
      MucTieu_lat[0]=10.807556;
      MucTieu_lng[0]=106.646574;  //Chọn mục tiêu
      goc_mong_muon =
gps.courseTo(currentPoint[0],currentPoint[1],MucTieu_lat[n_muc_tieu],MucTieu_lng[n_mu
c_tieu]);// tính toán phương vị với mục tiêu
      khoang_cach=
gps.distanceBetween(currentPoint[0],currentPoint[1],MucTieu_lat[n_muc_tieu],MucTieu_ln
g[n_muc_tieu])*100;
      if(khoang_cach<=300){n_muc_tieu=n_muc_tieu+1;
      t=1;
      }
      }

    }
}

void heading_read(){
  sensors_event_t event;
  mag.getEvent(&event);
  heading = atan2(event.magnetic.y, event.magnetic.x);
  declinationAngle = 0.2;
  heading += declinationAngle;
  if(heading < 0)
    heading += 2*PI;
  if(heading > 2*PI)
    heading -= 2*PI;
  headingDegrees = heading * 180/M_PI;

}
void display(){
  Serial.print("(");
```

```
  Serial.print(currentPoint[0], 7);
  Serial.print(" , ");
  Serial.print(currentPoint[1], 7);
  Serial.print(")\t");
  Serial.print(rcChannels[0]);
  Serial.print("\t");
  Serial.print(rcChannels[1]);
  Serial.print("\t");
  Serial.print(rcChannels[2]);
  Serial.print("\t");
  Serial.print(rcChannels[3]);
  Serial.print("\t");
  Serial.print(rcChannels[6]);
  Serial.print(" H: ");
  Serial.print(headingDegrees);
  Serial.print("\t");
  Serial.print(khoang_cach);
  Serial.print("\t");
  Serial.println(goc_mong_muon);
}
void khoi_dong(){
  if(time_runing <= 15000){
    rcChannels[0]=1500;
    rcChannels[1]=1500;
    rcChannels[2]=1500;
    rcChannels[3]=1500;
  }else if(time_runing <= 17000){
    rcChannels[0]=1000;
    rcChannels[1]=1000;
    rcChannels[2]=1000;
    rcChannels[3]=2000;
  }else if(time_runing <= 22000){
    rcChannels[0]=1500;
    rcChannels[1]=1500;
    rcChannels[2]=1700;
    rcChannels[3]=1500;

  }else{
    tick_khoi_dong=1;
  }
}
void hoat_dong(){

    rcChannels[0]=1500;
    rcChannels[1]=ch2_pitch_PWM;
    rcChannels[2]=1500;
    rcChannels[3]=ch4_yaw_PWM;
```

```
}

void landing_1(){
  rcChannels[0]=1500;
  rcChannels[1]=1500;
  rcChannels[2]=1400;
  rcChannels[3]=1500;
}

void landing_2(){
  rcChannels[0]=1500;
  rcChannels[1]=1500;
  rcChannels[2]=1100;
  rcChannels[3]=1500;
}

void nrf_thu(){
                if(radio.available()){
                        radio.read(&receiverData, sizeof(PacketData));
                        rcChannels_in[0] = receiverData.ch1; //sua lai
                        rcChannels_in[1] = receiverData.ch2;
                        rcChannels_in[2] = receiverData.ch3;
                        rcChannels_in[3] = receiverData.ch4;
                        rcChannels_in[4] = receiverData.ch5;
                        rcChannels_in[5] = receiverData.ch6;
                        rcChannels_in[6] = receiverData.ch7;
                        rcChannels_in[7] = receiverData.ch8;
    MucTieu_lat=receiverData.lat;
    MucTieu_lon=receiverData.lon;
    lenh_hoat_dong= receiverData.lenh;


    // rcChannels[0] = 1500; //sua lai
                        // rcChannels[1] = 1500;
                        // rcChannels[2] = 1500;
                        // rcChannels[3] = 1500;
                        // rcChannels[4] = 1500;
                        // rcChannels[5] = 1300;
                        // rcChannels[6] = 1840;
                        // rcChannels[7] = 1500;

                        digitalWrite(LEDPin, HIGH);
                        timer_an_toan_nrf=timer_curent;
        }
 if(timer_curent-timer_an_toan_nrf>100){
                        digitalWrite(LEDPin, LOW);
```

```
  }
}

void sbus_xuat(){
  //              unsigned long now = millis();
  //              if (now - lastRecvTime > SIGNAL_TIMEOUT ) {
  //              digitalWrite(LEDPin, LOW);
  //              for (uint8_t i = 0; i < SBUS_CHANNEL_NUMBER; i++) {
  //              rcChannels [i] = 900;
  //      }
  // }
    uint32_t currentMillis = millis();
    if (currentMillis > sbusTime) {
        sbusPreparePacket(sbusPacket, rcChannels, false, false);
        Serial.write(sbusPacket, SBUS_PACKET_LENGTH);
        sbusTime = currentMillis + SBUS_UPDATE_RATE;
    }
}
void PID_pitch(){
  E_pitch   =  khoang_cach;
  P_pitch   =  k_p_pitch * E_pitch;
  I_pitch   =  I_pitch + k_i_pitch * E_pitch;
  D_pitch   =  k_d_pitch * (E_pitch - E_pitch_0);
  E_pitch_0 =  E_pitch;
  U_pitch   =  P_pitch + I_pitch + D_pitch;

  P_pitch=MIN_MAX(P_pitch,200);
  I_pitch=MIN_MAX(I_pitch,150);
  D_pitch=MIN_MAX(D_pitch,150);
  U_pitch=MIN_MAX(U_pitch,200);
}

void PID_yaw(){
  E_yaw    =  headingDegrees - goc_mong_muon;
  if(E_yaw>180 ){E_yaw=(360-E_yaw)*(-1);}
  else if(E_yaw<-180){E_yaw=(-360-E_yaw)*(-1);}

  P_yaw    =  k_p_yaw * E_yaw ;
  I_yaw    =  I_yaw + k_i_yaw * E_yaw ;
  D_yaw    =  k_d_yaw * (E_yaw - E_yaw_0);
  E_yaw_0  =  E_yaw;
  U_yaw    =  P_yaw + I_yaw + D_yaw;
  P_yaw=MIN_MAX(P_yaw,250);
  I_yaw=MIN_MAX(I_yaw,150);
  D_yaw=MIN_MAX(D_yaw,150);
  U_yaw=MIN_MAX(U_yaw,250);
}
```

```
int MIN_MAX(float X, int max) {
  if (X > max) {
    X = max;
  } else if (X < -max) {
    X = -max;
  }
  return X;
}

void set_chanel(){
  //if(E_yaw > -gioi_han_E_yaw && E_yaw < gioi_han_E_yaw ){
  ch1_roll_PWM=1500-U_roll;
  ch2_pitch_PWM=1500+abs(U_pitch);
  //}else{
  // ch1_roll_PWM=1500;
  // ch2_pitch_PWM=1500;
  //}
  ch3_throttle_PWM=1500-U_throttle;
  ch4_yaw_PWM=1500-U_yaw;
}
```

PHỤ LỤC 3

Chương trình hệ thống né vật cản:

```
#include <SimpleKalmanFilter.h>
SimpleKalmanFilter  ch1_filter(30,30,1);
SimpleKalmanFilter  ch2_filter(30,30,1);

#include <NewPing.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <SPI.h>

#define RC_CHANNEL_MIN 1000
#define RC_CHANNEL_MAX 2000

#define SBUS_MIN_OFFSET 173
#define SBUS_MID_OFFSET 992
#define SBUS_MAX_OFFSET 1811
#define SBUS_CHANNEL_NUMBER 16
#define SBUS_PACKET_LENGTH 25
#define SBUS_FRAME_HEADER 0x0f
#define SBUS_FRAME_FOOTER 0x00
#define SBUS_FRAME_FOOTER_V2 0x04
#define SBUS_STATE_FAILSAFE 0x08
#define SBUS_STATE_SIGNALLOSS 0x04
#define SBUS_UPDATE_RATE 15 //ms

#define SIGNAL_TIMEOUT 100 //Signal timeout in milli seconds
#define LEDPin A1
unsigned long lastRecvTime = 0;
RF24 radio(9,10);//CE & CSN
const byte diaChi[6]="94147";

struct PacketData {
  int ch1;
  int ch2;
  int ch3;
  int ch4;
  int ch5;
  int ch6;
  int ch7;
  int ch8;
```

```
};
PacketData receiverData;



void sbusPreparePacket(uint8_t packet[], int channels[], bool isSignalLoss, bool isFailsafe){

  static int output[SBUS_CHANNEL_NUMBER] = {0};

  /*
   * Map 1000-2000 with middle at 1500 chanel values to
   * 173-1811 with middle at 992 S.BUS protocol requires
   */
  for (uint8_t i = 0; i < SBUS_CHANNEL_NUMBER; i++) {
    output[i] = map(channels[i], RC_CHANNEL_MIN, RC_CHANNEL_MAX,
     SBUS_MIN_OFFSET, SBUS_MAX_OFFSET);
  }

  uint8_t stateByte = 0x00;
  if (isSignalLoss) {
    stateByte |= SBUS_STATE_SIGNALLOSS;
  }
  if (isFailsafe) {
    stateByte |= SBUS_STATE_FAILSAFE;
  }
  packet[0] = SBUS_FRAME_HEADER; //Header

  packet[1] = (uint8_t) (output[0] & 0x07FF);
  packet[2] = (uint8_t) ((output[0] & 0x07FF)>>8 | (output[1] & 0x07FF)<<3);
  packet[3] = (uint8_t) ((output[1] & 0x07FF)>>5 | (output[2] & 0x07FF)<<6);
  packet[4] = (uint8_t) ((output[2] & 0x07FF)>>2);
  packet[5] = (uint8_t) ((output[2] & 0x07FF)>>10 | (output[3] & 0x07FF)<<1);
  packet[6] = (uint8_t) ((output[3] & 0x07FF)>>7 | (output[4] & 0x07FF)<<4);
  packet[7] = (uint8_t) ((output[4] & 0x07FF)>>4 | (output[5] & 0x07FF)<<7);
  packet[8] = (uint8_t) ((output[5] & 0x07FF)>>1);
  packet[9] = (uint8_t) ((output[5] & 0x07FF)>>9 | (output[6] & 0x07FF)<<2);
  packet[10] = (uint8_t) ((output[6] & 0x07FF)>>6 | (output[7] & 0x07FF)<<5);
  packet[11] = (uint8_t) ((output[7] & 0x07FF)>>3);
  packet[12] = (uint8_t) ((output[8] & 0x07FF));
  packet[13] = (uint8_t) ((output[8] & 0x07FF)>>8 | (output[9] & 0x07FF)<<3);
  packet[14] = (uint8_t) ((output[9] & 0x07FF)>>5 | (output[10] & 0x07FF)<<6);
```

```
    packet[15] = (uint8_t) ((output[10] & 0x07FF)>>2);
    packet[16] = (uint8_t) ((output[10] & 0x07FF)>>10 | (output[11] & 0x07FF)<<1);
    packet[17] = (uint8_t) ((output[11] & 0x07FF)>>7 | (output[12] & 0x07FF)<<4);
    packet[18] = (uint8_t) ((output[12] & 0x07FF)>>4 | (output[13] & 0x07FF)<<7);
    packet[19] = (uint8_t) ((output[13] & 0x07FF)>>1);
    packet[20] = (uint8_t) ((output[13] & 0x07FF)>>9 | (output[14] & 0x07FF)<<2);
    packet[21] = (uint8_t) ((output[14] & 0x07FF)>>6 | (output[15] & 0x07FF)<<5);
    packet[22] = (uint8_t) ((output[15] & 0x07FF)>>3);

    packet[23] = stateByte; //Flags byte
    packet[24] = SBUS_FRAME_FOOTER; //Footer
}

uint8_t sbusPacket[SBUS_PACKET_LENGTH];
int rcChannels[SBUS_CHANNEL_NUMBER];
int rcChannels_in[SBUS_CHANNEL_NUMBER];
uint32_t sbusTime = 0;

unsigned long timer_curent, time_start, time_runing, timer_an_toan_nrf;
bool d_timer=0, tick_khoi_dong=0,   tick_nrf_an_toan=0;

#define limit_distance 100
#define k_p_kc 3

#define SONAR_NUM     4 // Number of sensors.
#define MAX_DISTANCE 300 // Maximum distance (in cm) to ping.
#define PING_INTERVAL 33 // Milliseconds between sensor pings (29ms is about the min
        to avoid cross-sensor echo).

unsigned long pingTimer[SONAR_NUM]; // Holds the times when the next ping should
        happen for each sensor.
unsigned int cm[SONAR_NUM];         // Where the ping distances are stored.
uint8_t currentSensor = 0;          // Keeps track of which sensor is active.
NewPing sonar[SONAR_NUM] = {     // Sensor object array.
  NewPing(A0, 2, MAX_DISTANCE), // Each sensor's trigger pin, echo pin, and max
        distance to ping.
  NewPing(3, 4, MAX_DISTANCE),
  NewPing(5, 6, MAX_DISTANCE),
  NewPing(7, 8, MAX_DISTANCE),
};
```

```
void setup() {
  radio.begin();
  radio.openReadingPipe(1,diaChi);// mặc định đường truyền TX đã là 0 nên RX=1
  radio.setPALevel(RF24_PA_MIN);// cài đặt bộ khuếch đại công suất
  radio.setChannel(90);//lưu ý tx và rx phải cùng kênh(0-124)
  radio.setDataRate(RF24_250KBPS);
  radio.startListening();//cài đặt là RX

    for (uint8_t i = 0; i < SBUS_CHANNEL_NUMBER; i++) {
        rcChannels[i] = 1500;
    }
  pinMode(LEDPin, OUTPUT);


  pingTimer[0] = millis() + 75;          // First ping starts at 75ms, gives time for the Arduino
        to chill before starting.
  for (uint8_t i = 1; i < SONAR_NUM; i++) // Set the starting time for each sensor.
  {
    pingTimer[i] = pingTimer[i - 1] + PING_INTERVAL;
  }

  // Serial.begin(100000, SERIAL_8E2);
  Serial.begin(115200);
}
  float distance_truoc, distance_sau, distance_phai, distance_trai;

void loop() {
  timer_curent=millis();
  if(d_timer=0){
    time_start=timer_curent;
    d_timer=1;
  }
  time_runing=  timer_curent - time_start;
  nrf_thu();
  if(timer_curent - timer_an_toan_nrf>500){
  tick_nrf_an_toan=1;
  tick_khoi_dong = 1;
  }

  hoat_dong();
```

```
  rcChannels[4]=1500;
  rcChannels[5]=1300;
  if(tick_nrf_an_toan==0){  rcChannels[6]=1840; }
  else if(tick_nrf_an_toan==1){rcChannels[6]=1000;}
  rcChannels[7]=2000;

  calculate_distance();

  distance_truoc =xu_ly_sieu_am(cm[0], limit_distance);
  distance_sau   =xu_ly_sieu_am(cm[1], limit_distance);
  distance_phai  =xu_ly_sieu_am(cm[2], limit_distance);
  distance_trai  =xu_ly_sieu_am(cm[3], limit_distance);

  rcChannels[1]= 1500 + ((rcChannels[1]-
        1500)*(distance_truoc/limit_distance)*(distance_sau/limit_distance)
                - ne_vat_can( distance_truoc, k_p_kc, limit_distance)
                + ne_vat_can( distance_sau, k_p_kc, limit_distance));
  rcChannels[0]= 1500 + ((rcChannels[0]-
        1500)*(distance_phai/limit_distance)*(distance_trai/limit_distance)
                - ne_vat_can( distance_phai, k_p_kc, limit_distance)
                + ne_vat_can( distance_trai, k_p_kc, limit_distance));

  rcChannels[0] =ch1_filter.updateEstimate(rcChannels[0]);
  rcChannels[1] =ch2_filter.updateEstimate(rcChannels[1]);

  if(rcChannels[0]<1000)  rcChannels[0]=1000;
  if(rcChannels[1]<1000)  rcChannels[1]=1000;
  if(rcChannels[0]>2000)  rcChannels[0]=2000;
  if(rcChannels[1]>2000)  rcChannels[1]=2000;
  display();
  // sbus_xuat();

}

void display(){
  Serial.print(rcChannels[0]);
  Serial.print("\t");
  Serial.print(rcChannels[1]);
  Serial.print("\t");
  Serial.print(rcChannels[2]);
  Serial.print("\t");
```

```
  Serial.print(rcChannels[3]);
  Serial.print("\t");
  Serial.print(rcChannels[6]);
  Serial.print("\t");
  Serial.print(distance_truoc);
  Serial.print("\t");
  Serial.print(distance_sau);
  Serial.print("\t");
  Serial.print(distance_phai);
  Serial.print("\t");
  Serial.println(distance_trai);
}

int xu_ly_sieu_am( float distance, int limit_distance_in){
  if (distance==0){distance = limit_distance_in;}
  else if (distance > limit_distance_in) {distance = limit_distance_in;}
  return distance;
}

int ne_vat_can( float distance, float k_p , int limit_distance_in){
  float e_distance;
  int u_distance;
  if (distance==0){distance = limit_distance_in;}
  else if (distance > limit_distance_in) {distance = limit_distance_in;}
  e_distance = limit_distance_in - distance;
  u_distance = MIN_MAX((int)(e_distance*k_p),500);
  return u_distance;
}

int MIN_MAX(int X, int max) {
    if (X > max) {
        return max;
    } else if (X < -max) {
        return -max;
    } else {
        return X; // Return the original value if within range
    }
}

int antoan_ch(int ch){
  if(ch>=1000 && ch<=2000){
```

```
    return ch;
  }else{
    return 1500;
  }
}

void hoat_dong(){
rcChannels[0]=antoan_ch(rcChannels_in[0]);
    rcChannels[1]=antoan_ch(rcChannels_in[1]);
    rcChannels[2]=antoan_ch(rcChannels_in[2]);
    rcChannels[3]=antoan_ch(rcChannels_in[3]);

}

void khoi_dong(){
  if(time_runing <= 15000){
    rcChannels[0]=1500;
    rcChannels[1]=1500;
    rcChannels[2]=1500;
    rcChannels[3]=1500;
  }else if(time_runing <= 17000){
    rcChannels[0]=1000;
    rcChannels[1]=1000;
    rcChannels[2]=1000;
    rcChannels[3]=2000;
  }else if(time_runing <= 22000){
    rcChannels[0]=1500;
    rcChannels[1]=1500;
    rcChannels[2]=1700;
    rcChannels[3]=1500;

  }else{
    tick_khoi_dong=1;
  }
}

void landing_1(){
  rcChannels[0]=1500;
  rcChannels[1]=1500;
  rcChannels[2]=1400;
  rcChannels[3]=1500;
```

```
}

void landing_2(){
  rcChannels[0]=1500;
  rcChannels[1]=1500;
  rcChannels[2]=1100;
  rcChannels[3]=1500;
}

void nrf_thu(){
    if(radio.available()){
      radio.read(&receiverData, sizeof(PacketData));
      rcChannels_in[0] = receiverData.ch1; //sua lai
      rcChannels_in[1] = receiverData.ch2;
      rcChannels_in[2] = receiverData.ch3;
      rcChannels_in[3] = receiverData.ch4;
      rcChannels_in[4] = receiverData.ch5;
      rcChannels_in[5] = receiverData.ch6;
      rcChannels_in[6] = receiverData.ch7;
      rcChannels_in[7] = receiverData.ch8;

      digitalWrite(LEDPin, HIGH);
      timer_an_toan_nrf=timer_curent;
  }
  if(timer_curent-timer_an_toan_nrf>100){
      digitalWrite(LEDPin, LOW);
  }
}

void sbus_xuat(){

    uint32_t currentMillis = millis();
    if (currentMillis > sbusTime) {
        sbusPreparePacket(sbusPacket, rcChannels, false, false);
        Serial.write(sbusPacket, SBUS_PACKET_LENGTH);
        sbusTime = currentMillis + SBUS_UPDATE_RATE;
    }
}

void calculate_distance(){
    for (uint8_t i = 0; i < SONAR_NUM; i++) { // Loop through all the sensors.
```

```
    if (millis() >= pingTimer[i]) {         // Is it this sensor's time to ping?
      pingTimer[i] += PING_INTERVAL * SONAR_NUM;  // Set next time this sensor will
          be pinged.
      if (i == 0 && currentSensor == SONAR_NUM - 1)
      //oneSensorCycle(); // Sensor ping cycle complete, do something with the results.
      sonar[currentSensor].timer_stop();          // Make sure previous timer is canceled before
          starting a new ping (insurance).
      currentSensor = i;                          // Sensor being accessed.
      cm[currentSensor] = 0;                      // Make distance zero in case there's no ping echo
          for this sensor.
      sonar[currentSensor].ping_timer(echoCheck); // Do the ping (processing continues,
          interrupt will call echoCheck to look for echo).
    }
  }
  // Other code that *DOESN'T* analyze ping results can go here.
}

void echoCheck() { // If ping received, set the sensor distance to array.
  if (sonar[currentSensor].check_timer())
    cm[currentSensor] = sonar[currentSensor].ping_result / US_ROUNDTRIP_CM;
}

void oneSensorCycle() { // Sensor ping cycle complete, do something with the results.
  // The following code would be replaced with your code that does something with the ping
        results.
  for (uint8_t i = 0; i < SONAR_NUM; i++) {
    Serial.print(i);
    Serial.print("=");
    Serial.print(cm[i]);
    Serial.print("cm ");
  }
  Serial.println();
}
```