MỤC LỤC

[MỞ ĐẦU 2](#_Toc8236389)

[1.Tổng quát 2](#_Toc8236390)

[2.Mục tiêu đề tài: 2](#_Toc8236391)

[3. Phương pháp nghiên cứu: 3](#_Toc8236392)

[4. Đối tượng và phạm vi nghiên cứu 3](#_Toc8236393)

[CHƯƠNG 1: GIỚI THIỆU VỀ ROS 4](#_Toc8236394)

[1. Giới thiệu về ROS: 4](#_Toc8236395)

[2. Các thành phần trong ROS 4](#_Toc8236396)

[2.1 Node 4](#_Toc8236397)

[2.2 Topic 4](#_Toc8236398)

[2.3 Service 5](#_Toc8236399)

[2.4 Package 5](#_Toc8236400)

[3. Các lệnh cơ bản trong ROS 5](#_Toc8236401)

[3.1 Sử dụng rospack 5](#_Toc8236402)

[3.2 Sử dụng tập lệnh catkin\_create\_pkg 6](#_Toc8236403)

[3.3 Roscore 6](#_Toc8236404)

[3.4 Rosnode 6](#_Toc8236405)

[3.5 Rosrun 7](#_Toc8236406)

[3.6 Dùng rqt\_graph 7](#_Toc8236407)

[3.7.1 Rostopic echo 8](#_Toc8236408)

[3.7.2 Dùng rostopic list 9](#_Toc8236409)

[3.8 Dùng Roslaunch 10](#_Toc8236410)

[CHƯƠNG 2: TÌM HIỂU VỀ NAVIGATION STACK VÀ HECTOR MAPPING 11](#_Toc8236411)

[1. Navigation stack. 11](#_Toc8236412)

[1.1 Giới thiệu về navigation stack. 11](#_Toc8236413)

[1.2 Các thành phần trong navigation stack. 13](#_Toc8236414)

[2 Hector Mapping 14](#_Toc8236415)

[2.1.Giới thiệu về Hector Mapping 14](#_Toc8236416)

[2.2. Các thành phần trong Hector Mapping 14](#_Toc8236417)

[2.2.1. Subscribed Topics 15](#_Toc8236418)

[2.2.2. Published Topics 15](#_Toc8236419)

[2.2.3. Hoạt động của Hector mapping: 15](#_Toc8236420)

[2.2.4.Hector\_mapping ROS API 16](#_Toc8236421)

[2.2.5. Hector Mapping – Coordinate frames (khung tọa độ hector\_mapping) 16](#_Toc8236422)

[2.2.6. Attitude Estimation 16](#_Toc8236423)

[2.2.7. Map 17](#_Toc8236424)

[2.2.8. Sự quay vòng của hector\_mapping 17](#_Toc8236425)

[2.2.9. Xác định vị trí trên bản đồ 2D 18](#_Toc8236426)

[CHƯƠNG 3: PHÁT TRIỂN ROBOT HÚT BỤI THÔNG MINH 19](#_Toc8236427)

[1. Giới thiệu các chức năng của Robot 19](#_Toc8236428)

[2. Sơ đồ hoạt động của Robot 19](#_Toc8236429)

[3. Các thành phần của phần cứng 20](#_Toc8236430)

[3.1 RPlidar 20](#_Toc8236431)

[3.2 Raspberry Pi 21](#_Toc8236432)

[3.3 Arduino Mega 23](#_Toc8236433)

[3.4 Động cơ bước 24](#_Toc8236434)

[3.5 Mạch điều khiển động cơ bước A4988 25](#_Toc8236435)

[3.6 Relay 5V 26](#_Toc8236436)

[3.7 Vacuum motor 27](#_Toc8236437)

[3.8 Buck (5V) 27](#_Toc8236438)

[3.9 Batery 12V 28](#_Toc8236439)

[4 Cài đặt trung tâm sử lý 29](#_Toc8236440)

[4.1Cài đặt ROS 29](#_Toc8236441)

[4.2 Cài đặt NavigationStack và TF 29](#_Toc8236442)

[4.3 Cài đặt Hector Mapping 29](#_Toc8236443)

[4.4 Biên dịch và source: 30](#_Toc8236444)

[4.5 Cài đặt RPLidar trên Raspberry 30](#_Toc8236445)

[5. Lắp ráp trên Robot 32](#_Toc8236446)

[6 Xây dựng giải thuật điều khiển Robot: 34](#_Toc8236447)

[A/ Điều khiển động cơ bước: 34](#_Toc8236448)

[B/ Điều khiển mục tiêu di chuyển của robot: 36](#_Toc8236449)

[CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ 38](#_Toc8236450)

[1.Thử nghiệm: 38](#_Toc8236451)

[A) Thử nghiệm giải thuật hector mapping trong vẽ bản đồ 2D 38](#_Toc8236452)

[B) Thử nghiệm điều khiển robot bằng navigation stack 39](#_Toc8236453)

[2.Đánh giá 39](#_Toc8236454)

[CHƯƠNG 5: KẾT LUẬN 41](#_Toc8236455)

[Các kết quả đạt được: 41](#_Toc8236456)

[Hướng phát triển: 41](#_Toc8236457)

[Tài liệu tham khảo: 42](#_Toc8236458)

DANH MỤC HÌNH VẼ

[Hình 1: Các thành phần trong Navigation Stack 15](#_Toc8236501)

[Hình 2 Sơ đồ logic 21](#_Toc8236502)

[Hình 3 Sơ đồ phần cứng 22](#_Toc8236503)

[Hình 4 RPlidar A1 22](#_Toc8236504)

[Hình 5 Raspberry Pi 3B+ 23](#_Toc8236505)

[Hình 6 Arduino Mega 25](#_Toc8236506)

[Hình 7 Động Cơ Bước 6V 0.9 Độ CB609-3527 26](#_Toc8236507)

[Hình 8 Mạch điều khiển động cơ bước A4988 27](#_Toc8236508)

[Hình 9 Relay 5V 28](#_Toc8236509)

[Hình 10 Module hạ áp 5V5A 29](#_Toc8236510)

[Hình 11 Pin UltraFire 18650 4800mAH 3.7V 30](#_Toc8236511)

[Hình 12 Cụm động cơ 34](#_Toc8236512)

[Hình 13 Cụm cảm biến 35](#_Toc8236513)

[Hình 14 cụm nguồn 35](#_Toc8236514)

[Hình 15 Hình ảnh hoàn chỉnh 36](#_Toc8236515)

# MỞ ĐẦU

## 1.Tổng quát

Cách mạng công nghiệp (CMCN) 4.0 là xu hướng tự động hóa và trao đổi dữ liệu trong công nghệ sản xuất. Bản chất của CMCN 4.0 là dựa trên nền tảng công nghệ số và tích hợp tất cả các công nghệ thông minh để tối ưu hóa quy trình, phương thức sản xuất; nhấn mạnh những công nghệ đang và sẽ có tác động lớn nhất là công nghệ in 3D, công nghệ sinh học, công nghệ vật liệu mới, công nghệ tự động hóa, người máy... bao gồm các hệ thống không gian mạng, internet vạn vật và điện toán đám mây. Qua đó, tạo ra những nhà máy thông minh với hệ thống máy móc tự kết nối với nhau, tự tổ chức và quản lý. Đây còn được gọi là cuộc cách mạng số.

Cùng với sự phát triển của khoa học kĩ thuật thế kỉ 21, công nghệ 4.0 mang lại cho con người những điều tuyệt vời không tưởng. Robot hút bụi thông minh là một trong số đó. Các loại máy hút bụi hoặc robot thông minh hút bụi giúp con người có thời gian nghỉ ngơi đã khá phổ biến. Robot hút bụi thông minh hoạt động được trên các bề mặt sàn nhà, loại bỏ lông vật nuôi, bụi, và rác xung quanh phòng với hiệu quả ấn tượng. Hiện nay có rất nhiều mẫu máy hút bụi thông minh được sản xuất như :Neato, Xiaomi,….Các mẫu máy này được ứng dụng các tiến bộ mới nhất trong lĩnh vực cảm biến và điều khiển. Ví dụ như các mẫu sử dụng các giải thuật SLAM để vẽ bản đồ và lập kế hoạch di chuyển một cách thông minh. Những mẫu máy này thường rất đắt tiền và do các công ty lớn phát triển. Nắm bắt từ thực tế hiện nay nhiều nước trên thế giới đã ứng dụng Robot để phục vụ trong nhiều lĩnh vực như: công nghiệp, nông nghiệp, quân sự và trong cuộc sống sinh hoạt hằng ngày. Trong tương lai Robot sẽ là một công cụ hỗ trợ đắc lực cho con người và giúp con người vươn tới những tầm cao mới. Nhận thức được xu hướng và tầm quan trọng của Robot trong tương lai nhằm góp phần xây dựng và phát triển công nghệ Robot ở Việt Nam , nhóm đã quyết định chọn đề tài : **“Nghiên Cứu Phát Triển Robot Hút Bụi Nhà Thông Minh”.**

## 2.Mục tiêu đề tài:

- Nghiên cứu và tìm hiểu công nghệ liên quan đến phát triển robot thông minh

- Tìm hiểu và vận dụng các thành phần phần cứng phổ biến trong phát triển robot

- Ứng dụng các công nghệ và phần cứng sẵn có để phát triển robot hút bụi thông minh

## 3. Phương pháp nghiên cứu:

Công việc nghiên cứu và phát triển được thực hiện qua 3 giai đoạn:

- Giai đoạn 1: Tìm hiểu nghiên cứu các công nghệ liên quan đến phát triển Robot như hệ điều hành ROS và các phần cứng như Arduino, Raspberry Pi…

- Giai đoạn 2: Phát triển Robot hút bụi thông minh dựa trên ROS và giải thuật Hector SLAM

- Giai đoạn 3: Thử nghiệm hoạt động của Robot và đánh giá

## 4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu chính trong đề tài là sử dụng giải thuật SLAM trên hệ điều hành ROS để xây dựng Robot thông minh có khả năng vẽ bản đồ môi trường và tự định hướng, tự di chuyển theo chỉ thị của người dùng hoặc theo một lộ trình có sẵn. Phạm vi nghiên cứu được giới hạn trong việc tìm hiểu nghiên cứu lý thuyết và xây dựng nguyên mẫu Robot hút bụi nhằm thử nghiệm hoạt động của các giải thuật và phần cứng.

# CHƯƠNG 1: GIỚI THIỆU VỀ ROS

## 1. Giới thiệu về ROS:

ROS( Robot Operating System) là hệ thống phần mềm chuyên dụng để lập trình và điều khiển robot, gồm các công cụ để lập trình , hiển thị, tương tác trực tiếp với phần cứng, kết nối với cộng đồng Robot trên thế giới. Nó ra đời cho phép các nhà nghiên cứu tăng tốc độ phát triển các hệ thống robot mới, hỗ trợ tái sử dụng mã lệnh thông qua các công cụ và giao diện tiêu chuẩn. ROS hỗ trợ lập trình không đồng bộ do lập trình theo hướng call-back. Hệ điều hành mang tính phân tán nên các tiến trình nằm riêng biệt nhưng vẫn có thể kết nối thông qua message ( thông điệp). Điều đó thuận tiện cho làm việc nhóm trên các máy khác nhau có thể kết nối. Hơn nữa, việc dùng hệ điều hành ROS còn tránh được việc phụ thuộc phần cứng do sử dụng phương thức truyền tin kiểu message và dễ quản lý chức năng hệ thống thông qua danh sách node.

## 2. Các thành phần trong ROS

### 2.1 Node

Một node là một đơn vị tính toán trong ROS. Các node có thể liên kết với nhau thông qua các luồng như topic, service. Một robot có thể có nhiều node để thực hiện quá trình giao tiếp. Hệ điều hành ROS được thiết kế mô-đun hóa, một hệ thống điều khiển robot thường chứa nhiều node, các node phải được xây dựng chi tiết và chuyên biệt. Ví dụ, một con robot hút bui sẽ có node đọc dữ liệu, rồi gửi lệnh đến cho hai bánh xe, một node điều khiển động cơ bánh xe, một node định vị, một node hoạch định đường đi,..

### 2.2 Topic

Topic dùng để giao tiếp và trao đổi tin nhắc giữa hai node. Messages được định tuyến thông qua một hệ thống trao đổi ( transport system), trong đó phân loại thành 2 dạng: publish ( đưa tin ) và subcrible ( đăng ký nhận tin). Một node gửi đi một message bằng việc đưa thông tin tới một topic. Tên topic dùng để quy định nội dung của message. Một node liên quan đến loại dữ liệu thế nào sẽ đăng ký nhận tin từ topic tương ứng. Có nhiều đối tượng nhận tin cũng như một topic có thể truyền nhiều thông tin khác nhau, hay nhận nhiều thông tin khác nhau. Các nguồn tin và đối tượng nhận tin không cần biết lẫn nhau. ROS xây dựng tách biệt nguồn thông tin với bộ phận xử lý thông tin. Topic như là một kênh truyền thông điệp được định kiểu. Mỗi kênh có tên riêng, node nào cũng có thể kết nối với kênh này để gửi/nhận thông điệp, miễn cùng topic đó.

### 2.3 Service

Service là một phương thức giao tiếp không đồng bộ được sử dụng cho giao tiếp nhiều-nhiều. Một service đáp ứng một loại nhu cầu khác nhau cho các node.

### 2.4 Package

Gói dữ liệu là đơn vị chính trong tổ chức phần mềm của hệ điều hành ROS . Một package có thể chứ nhiều lệnh thực thi của ROS (các nodes).Một thư viện dành cho ROS, tập dữ liệu, các file cấu hình, hoặc các dữ liệu cần thiết khác trong hệ thống. Package chính là thành phần nguyên tử nhỏ nhất được xây dựng và đưa vào sử dụng trong ROS.Có nghĩa là rằng hạt nhân nhỏ nhất mà ta có thể xây dựng và đưa vào trong ROS chính là một package.

Metapakages: là các packagecụ thể chỉ làm nhiệm vụ đại diện cho nhóm các package khác có liên quan tới nhau. Hầu hết các metapackages đều được sử dụng giống như việc đảm bảo tính tương thich sau này khi chuyển đổi sang rosbuild stack.

Package Manifests: bảng kê khai thông tin gữi liệu của package (package.xml), cung cấp siêu gữi liệu về package đó bao gồm tên gọi, phiên bản, thông tin bản quyền (license) và những yếu tố phụ thuộc của gói dữ liệu đó. Manifests còn chưa thông tin về đặc trưng của ngôn ngữ lập trình ví dụ như các cờ báo (flags) của trình biên dịch.

## 3. Các lệnh cơ bản trong ROS

### 3.1 Sử dụng rospack

[Rospack](http://wiki.ros.org/rospack) cho phép bạn có được thông tin về các gói. Trong hướng dẫn này sẽ chỉ đề cập đến tùy chọn tìm , trả về đường dẫn đến gói.

Sử dụng: *$ rospack [pack\_name]*

Thí dụ: *$ rospack roscpp*

Sẽ trở lại: *YOU\_INSTALL\_PATH / share / roscpp*

### 3.2 Sử dụng tập lệnh [catkin\_create\_pkg](http://wiki.ros.org/catkin/commands/catkin_create_pkg)

Để tạo gói catkin mới và những gì bạn có thể làm với nó sau khi được tạo.

Đầu tiên thay đổi thư mục không gian nguồn của không gian làm việc catkin mà đã tạo trong [hướng dẫn Tạo không gian làm việc cho catkin](http://wiki.ros.org/catkin/Tutorials/create_a_workspace) :

Nên tạo cái này trong Hướng dẫn tạo không gian làm việc

*$ cd ~/catkin\_ws / src*

Bây giờ, hãy sử dụng tập lệnh catkin\_create\_pkg để tạo gói mới có tên 'Beginner\_tutorials', tùy thuộc vào std\_msss, roscpp và rospy:

*$ catkin\_create\_pkg Beginner\_tutorials std\_msss roscpp*

Điều này sẽ tạo ra một beginner\_tutorials thư mục mà chứa một [package.xml](http://wiki.ros.org/catkin/package.xml) và [CMakeLists.txt](http://wiki.ros.org/catkin/CMakeLists.txt) , đã được điền hoàn chỉnh với các thông tin mà đã catkin\_create\_pkg . Catkin\_create\_pkg yêu cầu cung cấp cho nó một tên\_bảng và tùy chọn một danh sách các phụ thuộc mà gói đó phụ thuộc vào:

*# catkin\_create\_pkg<pack\_name>* *[Dep1] [Dep2] [Dep3]*

Catkin\_create\_pkg cũng có các chức năng nâng cao hơn được mô tả trong [catkin / Command / catkin\_create\_pkg](http://wiki.ros.org/catkin/commands/catkin_create_pkg) .

### 3.3 Roscore

Roscore là nút đầu tiên bạn phải chạy khi dùng ROS.

Thử chạy: *$ roscore*

### 3.4 Rosnode

Rosnode hiển thị thông tin về các nút ROS hiện đang chạy. Lệnh rosnode list liệt kê các nút đang hoạt động này: *$ rosnode list*

sẽ thấy: */rosout*

Điều này cho thấy rằng chỉ có một nút chạy: [**rosout**](https://wiki.ros.org/rosout). Nút này luôn luôn chạy khi nó thu thập và ghi lại thông tin gỡ lỗi của các nút. Lệnh rosnode info trả về thông tin về một nút cụ thể. *$ rosnode info /rosout*

### 3.5 Rosrun

Rosrun cho phép bạn sử dụng tên gói để chạy trực tiếp một nút bên trong một gói (mà không cần phải biết đường dẫn gói).

Dùng: *$ rosrun [package\_name] [node\_name]*

Bây giờ chúng ta có thể chạy nút turtlesim\_node trong gói turtlesim. Sau đó, trong một **terminal mới:** *$ rosrun turtlesim\_node*

### 3.6 Dùng rqt\_graph

Rqt\_graph tạo ra một biểu đồ động về những gì đang xảy ra trong hệ thống. Rqt\_graph là một phần của gói rqt. Một khi đã cài đặt xong, hãy chạy:

*$ sudo apt-get install ros- <distro> -rqt*

*$ sudo apt-get install ros- <distro> -rqt-common-plugins*

Thay thế <distro> với tên của phiên bản ROS dùng (indigo, jade, kinetic)

**Trong một terminal mới:** *$ rosrun rqt\_graph rqt\_graph*

sẽ thấy giống như sau:



Nếu đặt con trỏ chuột trên / turtle1 / command\_velocity nó sẽ làm nổi bật các nút ROS (ở đây màu xanh và màu xanh lá cây) và các chủ đề (ở đây màu đỏ). Như bạn thấy, turtlesim\_node và các nút turtle\_teleop\_key đang truyền đạt về chủ đề có tên / turtle1 / command\_velocity.

3.7 Dùng rostopic

Công cụ rostopic cho phép lấy thông tin về các chủ đề ROS. Có thể sử dụng tùy chọn trợ giúp để lấy các lệnh khác trong rostopic

*$ rostopic -h*

*rostopic bw* : băng thông hiển thị được sử dụng bởi chủ đề

*rostopic echo* : in tin nhắn lên màn hình

*rostopic hz* : hiển thị tỷ lệ xuất bản của chủ đề

*rostopic list* : in thông tin về các chủ đề đang hoạt động

*rostopic pub* : xuất bản dữ liệu đến chủ đề

*rostopic type* : in chủ đề

Hoặc nhấn phím tab sau khi lệnh rostopic in các lệnh có thể có:

*$ rostopic*

*bw echo find hz info list pub type*

### 3.7.1 Rostopic echo

Rostopic echo cho thấy dữ liệu xuất bản trên một chủ đề.

Dùng: *$ rostopic echo [topic]*

Sử dụng *rostopic list [/topic]*

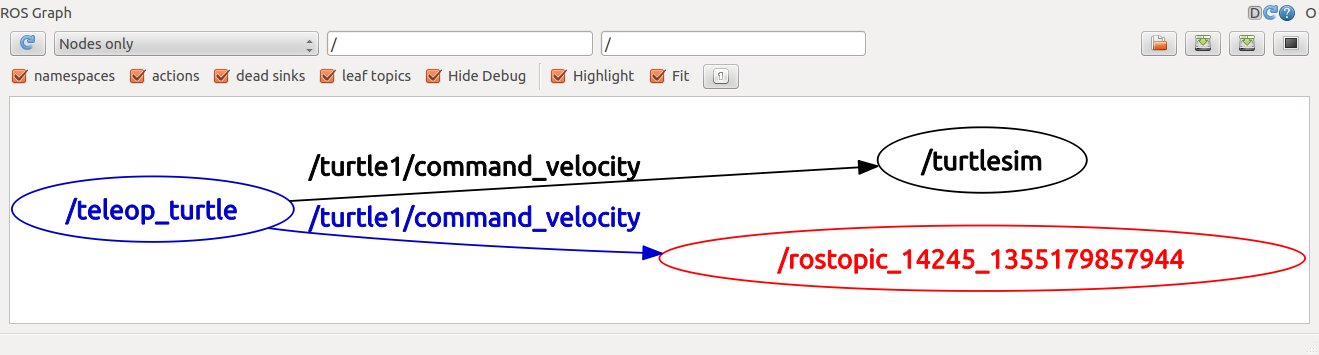
Tùy chọn*: -h, --help* : hiển thị thông báo trợ giúp này và thoát

*-b BAGFILE, --bag=BAGFILE* : liệt kê các chủ đề trong tập tin .bag

*-v, --verbose* : liệt kê đầy đủ chi tiết về từng chủ đề

*-p* : chỉ liệt kê các nhà xuất bản

*-s* : chỉ liệt kê những người đăng ký



### 3.7.2 Dùng rostopic list

Rostopic list trả về một danh sách tất cả các chủ đề hiện đang đăng ký và xuất bản.Chúng ta hãy tìm ra các đối số cần trong danh sách lệnh. Trong một **thiết bị đầu cuối mớ**i chạy:

*$ rostopic list -h*

Cho rostopic list dùng lựa chọn **verbose***: $ rostopic list -v*

Điều này sẽ hiển thị một danh sách đầy đủ các chủ đề để xuất bản lên và đăng ký và loại của topic.

Published topics: *\* /turtle1/color\_sensor [turtlesim/Color] 1 publisher*

*\* /turtle1/command\_velocity [turtlesim/Velocity] 1 publisher*

*\* /rosout [roslib/Log] 2 publishers*

*\* /rosout\_agg [roslib/Log] 1 publisher*

*\* /turtle1/pose [turtlesim/Pose] 1 publisher*

Subscribed topics: *\* /turtle1/command\_velocity [turtlesim/Velocity] 1 subscriber*

*\* /rosout [roslib/Log] 1 subscriber*

### 3.8 Dùng Roslaunch

Roslaunch để khởi chạy nhiều nút một lúc. Roslaunch bắt đầu một nút như định nghĩa trong tập tin launch.

Dùng: *$ roslaunch [package] [filename.launch]*

Trước tiên, hãy đi tới gói Beginner\_tutorials mà chúng tôi đã tạo và xây dựng trước đó*: $ roscd beginner\_tutorials*

Nếu roscd thông báo giống như *roscd: Không tìm thấy gói (node/stack) 'beginner\_tutorials'*, bạn cần thiết đặt mã nguồn trong tập tin môi trường giống như trong cuối hướng dẫn [create\_a\_workspace](http://wiki.ros.org/catkin/Tutorials/create_a_workspace) :

*$ cd ~/catkin\_ws*

*$ source devel/setup.bash*

*$ roscd beginner\_tutorials*

Tạo một thư mục launch: *$ mkdir launch*

*$ cd launch*

Thư mục lưu trữ các tệp khởi chạy không nhất thiết phải được đặt tên là khởi chạy. Trong thực tế, thậm chí không cần phải lưu trữ chúng trong một thư mục. Lệnh roslaunch tự động tìm kiếm gói đã được thông qua và phát hiện các tệp khởi chạy có sẵn. Tuy nhiên, đặt tên thư mục launch là một thực hành tốt.

# CHƯƠNG 2: TÌM HIỂU VỀ NAVIGATION STACK VÀ HECTOR MAPPING

## 1. Navigation stack.

### 1.1 Giới thiệu về navigation stack.

Trước khi nói về navigation stack, chúng ta sẽ làm rõ khái niệm về SLAM trước. SLAM (simultaneous localization and mapping) là hệ thống sử dụng thông tin ảnh thu được từ camera để tái tạo môi trường bên ngoài, bằng cách đưa thông tin môi trường vào một map (2D hoặc 3D). Từ đó, thiết bị (robot, camera, xe) có thể định vị (localization) đang ở đâu, trạng thái, tư thế của nó trong map để tự động thiết lập đường đi (path planning) trong môi trường hiện tại.

Điều khiển tự động thiết bị robot chia làm 3 vấn đề chính: định vị (localization), tái tạo môi trường (mapping) và hoạch định đường đi (path planning). Ban đầu, hai vấn đề định vị và tái tạo môi trường được nghiên cứu độc lập, tuy nhiên, sau khi nhận thấy:

• Định vị: cần xác định vị trí hiện tại của robot dựa vào bản đồ tái tạo.

• Tái tạo bản đồ: cần xác định vị trí của đối tượng trong bản đồ, để xây dựng bản đồ chính xác nhất, ít sai số.

Vì thế, trong giai đoạn 1985-1990, Raja Chatila và Jean-Paul Laumond (1985) và Randall Smith (1990) đã đề xuất gộp hai vấn đề với nhau để nghiên cứu. Một thời gian sau đó, SLAM ra đời đại diện cho localization và mapping.

Trong ROS có một gói hỗ trợ SLAM, đó chính là Navigation stack. ROS cũng có một node là ACML (node này không đi kèm với navigation) , node này giúp robot định vị vị trí hiện tại thông qua một bản đồ tĩnh đã được tải vào trước đó. Gói này hoạt động rất tốt trong những môi trường mà địa hình hiếm khi thay đổi do nó tốn ít nguồn lực của robot hơn navigation stack. Nhưng chính điều đó lại là nhược điểm của node này, bởi mỗi khi địa hình có sự thay đổi thì ta lại phải cập nhật lại bản đồ mới, điều này rất tốn thời gian và công sức tính toán và không linh hoạt. Robot có thể phải hoạt động ở những môi trường hay thay đổi như trường học, hay những nơi tương tự khác. Khắc phục nhược điểm này, navigation stack cung cấp hai hệ thống chính để làm nhiệm vụ định vị là: gmapping và hector mapping.

Cả hai hệ thống này đều dựa trên SLAM, hai thứ này chỉ khác nhau ở chỗ gmapping tính đến thông tin về vị trí và vận tốc (odometry information) để tạo và cập nhật bản đồ robot đặt ra, tuy nhiên một số robot không cần đến bộ đo vân tốc (endcoders) (vd như robot bay). Thông tin về vị trí và vận tốc rất tốt vì chúng hỗ trợ tạo ra các bản đồ chính xác hơn, vì hiểu được chuyển động (động lực học) của robot, chúng ta có thể đánh giá được đúng vấn đề hơn.

Động lực học bị ảnh hưởng bởi cách các thiết bị đảm bảo chuyển động của robot như: loại bánh xe, số lượng bánh xe, vị trí bánh xe và góc quay chúng được xử lý.

Dù rất hữu ích nhưng thông tin về vị trí và vận tốc vẫn sẽ bị lỗi, có thể lo do sự thiếu chính xác về độ chụm, ma sát, trôi, và các yếu tố khác, chúng có thể tích lũy theo thời gian (sai số ban đầu có thể rất nhỏ) dẫn đến tạo ra dữ liệu không đồng nhất làm ảnh hưởng đến sự hình thành bản đồ khiến bản đồ bị biến dạng.

Chi tiết về hectormapping sẽ được giới thiệu trong phần sau do cỗ máy của chúng ta sử dụng hectormapping.

Các yêu cầu về phần cứng đối với navigation stack:

- Stack được xây dựng cho việc điều khiển trên các robot dạng khối (diﬀerential drive) và có hai bánh điều khiển độc lập, tuy nhiên ta vẫn có thể sử dụng một số tính năng của nó cho các loại robot khác.

- Robot sẽ nhận được một thông báo có kiểu dữ liệu là twist với vận tốc X, Y, và Theta và có thể điều khiển các bánh xe hay những thứ tương tự để đạt được vận tốc này. Nếu robot không thể đạt tới vận tốc thì ta chỉ cần điều chỉnh phần cứng hoặc tạo 1 node để navigation stack chuyển thành lại tin phù hợp nhất với yêu cầu.

-Thông tin về môi trường được lấy từ một chủ đề có kiểu LaserScan. Nếu dữ liệu nhận từ loại laser phẳng thì chỉ cần xuất dữ liệu ra tức là cài đặt node. Vẫn có thể sử dụng cảm biến khác miễn là có thể chuyển sang kiểu LaserScan.

-Navigation stack hoạt động tốt nhất với robot hình vuông hoặc hình tròn. Ta vẫn có thể sử dụng nó với các robot có hình dạng tùy ý khác. Tuy nhiên những robot có hình dạng lạ, đăc biệt có thể gặp khó khăn trong việc di chuyển.

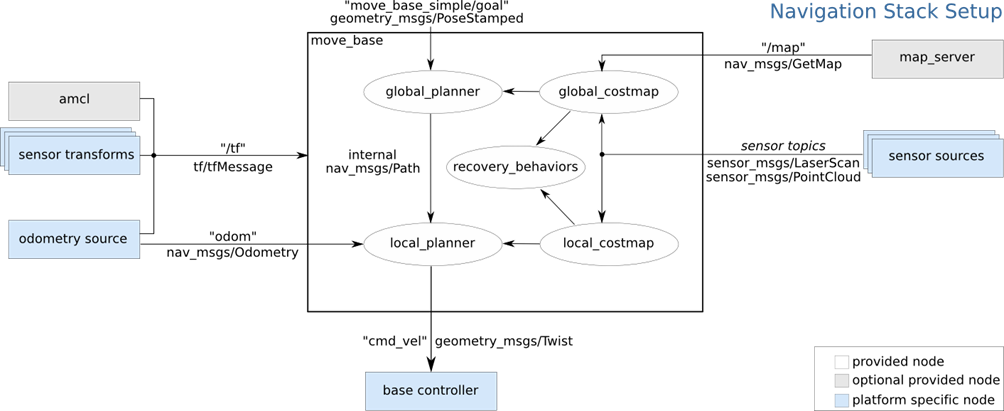
Các yêu cầu về phần mếm khi sử dụng Navigation stack:

-Nên sử dụng ROS Hydro hoặc Indigo để sử dụng hết các tính năng.

-Nên cài Navigation stack trong ROS, nó ở trong gói nhưng tùy thuộc cài đặt mà có thể có có một vài node chưa được cài.

-Có một chương trình để điều khiển robot, một chương trình đọc dữ liệu về cảm biến và đăng lên topic.

### 1.2 Các thành phần trong navigation stack.



Hình 1: Các thành phần trong Navigation Stack

**Global cost map**: Nhận dữ liệu từ map server và sensor sources đển lưu trữ cũng như cập nhập để thành lập một bản đồ 2d dưới dạng lưới ô vuông. Có 3 vùng trong lưới ô vuông này có thể tạm hiểu là: đã bị chiếm (không thể đi), tự do (có thể đi) và vùng không xác định (các vùng chưa có dữ liệu hoặc chưa đủ dữ liệu để vẽ bản đồ).

**Global planner**: Dùng để hoạch định con đường mà robot sẽ đi từ vị trí hiện tại tới một điểm bất kỳ trên bản đồ. Global planer sẽ nhận bản đồ 2d từ Global cost map. Node này cho ta nhiều tùy biến như cài đặt tất cả tham số ở chế độ mặc định, lựa chọn loại thuật toán tìm đường đi (Dijkstra, A\*), hay làm mềm đường đi của robot…

**Recorvery behaviors**: khi robot mắc vào bẫy (tức là vào vùng không thể di chuyển bằng cách tránh chướng ngại vật nữa) thì node này tiến hành các biện pháp để robot chủ động thoát bẫy như quay lại đường cũ, xoay 360 độ để dọn dẹp chướng ngại vật.

**Local costmap:** chức năng giống với global costmap nhưng được dùng trong kế hoạch ngắn hạn, tức thời.

**Local planer:** chức năng giống với global planer nhưng cũng được dùng trong kế hoạch ngắn hạn, tức thời.

## 2 Hector Mapping

### 2.1.Giới thiệu về Hector Mapping

Hector\_mapping là một cách tiếp cận SLAM mà được sử dụng mà không cần odometry cũng như là không cần nền tảng để đưa ra các hành động lên, xuống và quay quanh tr. Nó được hỗ trợ bởi tỷ lệ cập nhật cao của hệ thống LIDAR hiện đại như Hokuyo UTM-30LX và cung cấp vị trí tượng hình 2D dựa trên tốc độ quét của cảm biến (40Hz cho UTM 30LX). Trong khi các hệ thống không cung cấp khả năng kết thúc vòng lặp 1 cách rõ rang , nó là đủ chính xác cho nhiều tình huống trên thực tế. Hệ thống này đã được sử dụng thành công trên Robot mặt đất không người lái, Phương tiện không người lái, Thiết bị lập bản đồ cầm tay và dữ liệu được ghi lại từ các UAV bốn hướng.

### 2.2. Các thành phần trong Hector Mapping

Để sử dụng hector\_mapping , bạn cần một nguồn dữ liệu sensor Msg / LaserScan (ví dụ: Hokuyo UTM-30LX LIDAR hoặc bagfiles). Nút sử dụng tf để chuyển đổi dữ liệu quét, do đó LIDAR không phải cố định liên kết với khung cơ sở. Dữ liệu về chuyển động là không cần thiết.

Hector\_mapping là một nút cho SLAM dựa trên LIDAR không có hình học và tài nguyên tính toán thấp. Để đơn giản,ROS API cung cấp thông tin chi tiết bên dưới về các tùy chọn thường được sử dụng từ góc độ người dùng, nhưng không phải tất cả các tùy chọn có sẵn cho mục đích gỡ rối.

### 2.2.1. Subscribed Topics

scan (sensor\_msgs/LaserScan)

• Topic chứa dữ liệu quét laser

syscommand (std\_msgs/String)

• Lệnh hệ thống . Nếu chuỗi bằng “reset” bản đồ và vị trí của robot sẽ được đặt lại trạng thái ban đầu.

### 2.2.2. Published Topics

map\_metadata (nav\_msgs/MapMetaData)

• Lấy thông tin bản đồ từ topic, cái mà được chốt,và cập nhật liên tục.

map (nav\_msgs/OccupancyGrid)

• Lấy thông tin bản đồ từ topic, cái mà được chốt,và cập nhật liên tục.

slam\_out\_pose (geometry\_msgs/PoseStamped)

• Ước lượng vị trí của robot mà ko có phương sai

poseupdate (geometry\_msgs/PoseWithCovarianceStamped)

• Ước lượng vị trí của robot cùng với ước lượng Gauusian.

\*\*\* dynamic\_map ( nav\_msgs / GetMap )

• Gọi dịch vụ này để lấy dữ liệu bản đồ

### 2.2.3. Hoạt động của Hector mapping:

Điều hướng dựa trên SLAM yêu cầu bản đồ SLAM. Nó là cơ sở của phần điều hướng. Nó phải là một bản đồ chi tiết, vì vậy thuật toán định vị sẽ hoạt động đúng. Các bản đồ được sử dụng trong nghiên cứu này đều được tạo bởi thư viện hector\_slam.

Thuật toán này sử dụng dữ liệu cảm biến LiDAR để tạo bản đồ cho toàn bộ khu vực, trong đó chiếc xe sẽ lái sau đó. Cách tiếp cận của thuật toán này là thực hiện cái gọi là FastSLAM. Toàn bộ khu vực được thể hiện dưới dạng bản đồ lưới 2D. Do tốc độ cập nhật cao của cảm biến LiDAR, chỉ có thể sử dụng dữ liệu gần đúng. Dữ liệu điểm cuối được quét của cảm biến là chuyển đổi thành đám mây điểm bằng cách sử dụng định hướng nền tảng ước tính và giá trị chung. Là thuật toán so khớp quét, chỉ lọc dựa trên tọa độ điểm cuối z là đủ, do đó, chỉ các điểm cuối trong ngưỡng của mặt phẳng quét dự định được sử dụng trong quy trình khớp quét

### 2.2.4.Hector\_mapping ROS API

Node SLAM chính : hector\_mapping

• Đầu vào chính

• Quét dữ liệu trên topic “/scan”

• Chuyển đổi dữ liệu thông qua tf

Đầu ra chính

• Bản đồ thu được từ topic “/map”

• Tf ‘map’ -> dạng ‘odom’

### 2**.2.5. Hector Mapping – Coordinate frames (khung tọa độ hector\_mapping)**

• Khung ‘odom’ là không cần thiết , chủ yếu tương thích với gmapping/ROS

• Khung ‘/base\_stabilized’ cần thiết để chuyển đổi thông tin của dữ liệu LIDAR

### 2.2.6. Attitude Estimation

• Sử dụng IMU để ước tính biểu hiện di chuyển

• Chúng tôi cung cấp nút hector\_imu\_attitude\_to\_tf

• Cung cấp base\_stabilized biến đổi thành base\_link

• LIDAR không được gắn chắc chắn vào khung cơ sở

• Cung cấp thiết lập robot / tf chính xác

• Kết quả tốt nhất nếu LIDAR được kích hoạt và giữ mức ổn định

### 2.2.7. Map

• Bản đồ được biểu thị bằng lưới 2D

• Truy cập dữ liệu bản đồ trên tọa độ không nguyên bằng cách sử dụng bộ lọc song tuyến.

• Map đưa ra chỉ gần đúng

• Map đưa ra nhanh

• Bộ nhớ Cache truy cập vào các ô lưới gần nhất .

### 2.2.8. Sự quay vòng của hector\_mapping

• Nhận quét từ LIDAR.

• Chuyển đổi các điểm cuối quét thành khung hình / khung cơ sở

• Vứt bỏ các điểm cuối bên ngoài giới hạn:

+ laser\_z\_min\_value

+ laser\_z\_max\_value

+ laser\_min\_dist

+ laser\_max\_dist

• Thực hiện ước tính vị trí của robot trong bản đồ 2D

• Cập nhật bản đồ nếu robot được ước tính đã di chuyển nhiều hơn ngưỡng được xác định bởi

+ map\_update\_distance\_thresh

+ map\_update\_angle\_thresh

### 2.2.9. Xác định vị trí trên bản đồ 2D

Việc ước tính tư thế của robot được thực hiện bằng một phép lặp đơn giản. Đầu tiên dự án điểm cuối lên bản đồ dựa trên ước tính tư thế hiện tại. Tiếp theo, ước tính độ dốc xác suất chiếm chỗ bản đồ tại điểm cuối quét. Và cuối cùng nhưng không kém phần thực hiện phép lặp Gauss-Newton để tinh chỉnh ước lượng tư thế. Dựa trên ước tính tư thế của robot và dữ liệu cảm biến, có thể xây dựng một bản đồ được xác định rõ ràng với tất cả các chướng ngại vật có trong khu vực.

# CHƯƠNG 3: PHÁT TRIỂN ROBOT HÚT BỤI THÔNG MINH

## 1. Giới thiệu các chức năng của Robot

Robot hút bụi được xây dựng trong đề tài này là một nguyên mẫu robot hút bụi thông minh có các tính năng tương tự các robot hút bụi hiện có trên thị trường. Các tính năng đó là hút bụi, thiết lập bản đồ. Với chức năng thiết lập bản đồ, Robot sẽ hoạt động thông minh hơn, ghi nhớ đường đi hiệu quả hơn và tiết kiệm năng lượng. Bên cạnh đó, Robot có thể lên kế hoạch đường đi làm sạch, khoa học và không bỏ sót đồng thời nhận biết được vật cản, việc di chuyển dễ dàng hơn, bảo vệ thân máy cũng như đồ dùng trong gia đình

## 2. Sơ đồ hoạt động của Robot

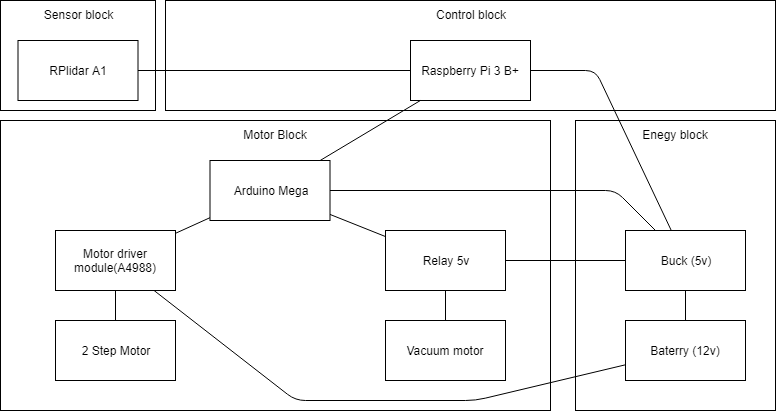
A/ Sơ đồ logic

A close up of a map

Description automatically generated

Hình 2 Sơ đồ logic

B/ Sơ đồ phần cứng

****

Hình 3 Sơ đồ phần cứng

## 3. Các thành phần của phần cứng

### 3.1 RPlidar

****

Hình 4 RPlidar A1

Cảm biến Laser Radar (Lidar) RPLIDAR A1 được sản xuất bởi hãng SLAMTEC được sử dụng cho các ứng dụng phát hiện vật cản, lập bản đồ bằng tia Laser trong xe, robot tự hành, hệ thống chống trộm, ..., cảm biến có độ ổn định và độ chính xác cao.

Cảm biến Laser Radar (Lidar) RPLIDAR A1 sử dụng giao tiếp UART nên có thể dễ dàng giao tiếp với Vi điều khiển, Máy tính nhúng hoặc kết nối máy tính qua mạch chuyển USB-UART và phần mềm đi kèm, cảm biến có khả năng quét xa với khoảng cách lên đến 12m, tần số tối đa 10Hz với 8000 samples per time, phù hợp cho vô số các ứng dụng khác nhau.

Thông số kỹ thuật:

-Điện áp sử dụng: 5VDC

-Chuẩn giao tiếp: UART

-Phương pháp phát hiện vật cản: Laser

-Khoảng cách phát hiện vật cản tối đa: 12m

-Góc quay: 360 độ.

-Tốc độ lấy mẫu tối đa: 8000 Samples per time.

-Tần số quét tối đa: 10Hz

-Kích thước: 71 x 97mm

### 3.2 Raspberry Pi

A circuit board

Description automatically generated

Hình 5 Raspberry Pi 3B+

Máy tính Raspberry Pi 3 Model B+ (Made in UK) là board mạch máy tính nhúng, sử dụng hệ điều hành Linux hoặc Windows 10 IoT, máy có khả năng xuất tín hiệu ra 40 chân GPIO giúp bạn có thể giao tiếp và điểu khiển vô số các board mạch phần cứng khác để thực hiện vô số các ứng dụng khác nhau.

Máy tính Raspberry Pi 3 Model B+ (Made in UK) có kích thước nhỏ gọn, giá thành phải chăng, cách sử dụng dễ dàng, chỉ cần cài hệ điều hành vào thẻ nhớ và cấp nguồn là có thể sử dụng.

CPU phiên bản mới BCM2837B0 từ Boardcom với vỏ bọc kim loại tản nhiệt cho khả năng tăng tốc độ lên đến 1.4Ghz 4 nhân với kiến trúc ARM Cortex-A53 64-bit.

Tốc độ Ethernet nhanh hơn (Gigabit via USB) lên đến 300Mb/s bằng việc nâng cấp chip xử lý mới LAN7515 từ Microchip.

Wifi Dual-band 2.4GHz & 5GHz với vỏ bọc IC kim loại chống nhiễu và tản nhiệt cho tốc độ và độ ổn định cao hơn.

Hỗ trợ Socket cắm Power over Ethernet (PoE) sử dụng với loại Raspberry Pi HAT tương thích.

IC quản lý nguồn thông minh MXL7704 giúp đơn giản hóa phần thiết kế nguồn trên board cho khả năng cấp nguồn ổn định và an toàn hơn

Thông số kỹ thuật chi tiết:

-Sản xuất tại: nhà máy Sony tại Anh (Made in UK), chính hãng RS Components.

-CPU Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz

-GPU Broadcom Videocore-IV

-RAM 1GB LPDDR2 SDRAM

-Networking: Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi

-Bluetooth 4.2, Bluetooth Low Energy (BLE)

-Storage: Micro-SD

-GPIO: 40-pin GPIO header, populated

-Ports: HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

-Dimensions: 82mm x 56mm x 19.5mm, 50g

### 3.3 Arduino Mega

A circuit board

Description automatically generated

Hình 6 Arduino Mega

Arduino Mega 2560 R3 là phiên bản nâng cấp của Arduino Uno R3 với số chân giao tiếp, ngoại vi và bộ nhớ nhiều hơn, có 54 chân I / O kỹ thuật số, 16 đầu vào tương tự và không gian rộng hơn. Hầu hết các Shield của Arduino Uno R3 đều chạy được với Arduino Mega 2560 R3.

Arduino Mega 2560 R3 không dùng được thư viện SoftwareSerial vì đã được tích hợp sẵn 4 cổng Hardware Serial trên board.

Thông số kỹ thuật:

-Vi điều khiển chính: ATmega2560

-IC nạp và giao tiếp UART: ATmega16U2.

-Nguồn nuôi mạch: 5VDC từ cổng USB hoặc nguồn ngoài cắm từ giắc tròn DC

-Số chân Digital I/O: 54 (trong đó 15 chân có khả năng xuất xung PWM)

-Số chân Analog Input: 16

-Dòng điện DC Current trên mỗi chân I/O: 20mA

-Dòng điện DC Current chân 3.3V: 50mA

-Flash Memory: 256 KB trong đó 8 KB sử dụng cho bootloader.

-SRAM: 8 KB

-EEPROM: 4 KB

-Clock Speed: 16 MHz

-LED\_BUILTIN: 13

-Kích thước: 101.52 x 53.3 mm

### 3.4 Động cơ bước

A close up of a device

Description automatically generated

Hình 7 Động Cơ Bước 6V 0.9 Độ CB609-3527

Thường được sử dụng trong các thiết bị: máy in 3D, máy vẽ, máy móc nghành dệt, thiết bị y tế, tự động hóa Robot…

Thông số kĩ thuật:

- Điện áp: 6V

- Dòng: 0.66A

- Mô-men xoắn: 0.03Nm (ước tính)

- Kích thước

+ Chiều rộng: 35mm

+ Chiều cao: 27mm

+ Đường kính trục ra: 5mm

- Góc bước: 0.9°

- 2 pha 4 dây

- Trọng lượng: 134g

### 3.5 Mạch điều khiển động cơ bước A4988

A circuit board

Description automatically generated

Hình 8 Mạch điều khiển động cơ bước A4988

A4988 là driver điều khiển động cơ bước cực kỳ nhỏ gọn, hổ trợ nhiều chế độ làm việc, điều chỉnh được dòng ra cho động cơ, tự động ngắt điện khi quá nóng . A4988 hỗ trợ nhiều chế độ hoạt động của động cơ bước lưỡng cực như: Full, 1/2, 1/4 , 1/8 và 1/16.

Thông số kỹ thuật:

-Công suất ngõ ra lên tới 35V, dòng đỉnh 2A.

-Có 5 chế độ: full bước, 1/2 bước, 1/4 bước, 1/8 bước, 1/16 bước

-Điểu chỉnh dòng ra bằng triết áp, nằm bên trên Current Limit = VREF × 2.5

-Tự động ngắt điện khi quá nhiệt.

### 3.6 Relay 5V

A circuit board

Description automatically generated

Hình 9 Relay 5V

Mạch Relay 5V10A 1 Kênh VN-05-01

Mạch điều khiển Relay 5V10A 1 kênh

Thông số kĩ thuật:

-Điện áp DCIN : 5V

-Điện áp kích Relay 2: 5VDC

-Điện áp qua tiếp điểm Relay:

+ 10A 250VAC / 10A 125VAC

+ 10A 30VDC / 10A 25VDC

### 3.7 Vacuum motor

Động cơ mini 3-6V V2

Thích hợp cho mô hình học tập. Thường được dùng nhiều để làm robot dò đường

Thông số kĩ thuật:

- Đường kính trục: 2MM

- Độ dài trục: 10MM

- Kích thước Động cơ: 20x28x15MM

- Động cơ chạy ổn định ở 3V

- Tốc độ: 3V 12.000r/Min

- Dòng tiêu thụ: 0.3-0.4A

### 3.8 Buck (5V)

A circuit board

Description automatically generated

Hình 10 Module hạ áp 5V5A

Module hạ áp DC-DC . Ứng dụng hạ áp cấp nguồn cho các mạch, thiết bị sử dụng các nguồn 5V5A. Module sử dụng IC TPS40057. Bảo vệ dòng ngược , cuộn lọc, mos công suất cao

Thông số kĩ thuật:

-Nguồn vào DC 9V – 35V

-Đầu ra 5V – 5A / 25W

-Kích thước 45\*31\*16mm

### 3.9 Batery 12V

A close up of a device

Description automatically generated

Hình 11 Pin UltraFire 18650 4800mAH 3.7V

Sử dụng làm nguồn nuôi cho xe robot, sạc pin dự phòng, .... Sử trọng nuôi nguồn cho các mạch điện tử. Bộ điều khiển cầm tay. Nguồn duy trì với thời gian dài

Thông số kĩ thuật:

-Điện Áp: DC 3.7V (Khi đầy pin lên đến 4.2V)

- Pin được sử dụng nhiều lần

- Kích Thước: 65x18MM

- Kiểu 18650

- Dung lượng: 4800mAH

- Nhiệt độ làm việc: -20 ℃ ~ 45 ℃

## 4 Cài đặt trung tâm sử lý

### 4.1Cài đặt ROS

- Cài đặt source.list:

*sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'*

- Cài đặt khóa:

*$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116 -Installation:*

*$ sudo apt-get update*

*$ sudo apt-get install ros-kinetic-desktop-full*

-Cài đặt rosdep:

*$ sudo rosdep init*

*rosdep update*

-Cài đặt môi trường:

*$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc*

*$ source ~/.bashrc*

-Cài đặt các gói phụ thuộc:

*$ sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential*

### 4.2 Cài đặt NavigationStack và TF

- Cài đặt NavigationStack sử dụng lệnh :

*$ sudo apt-get install ros-kinetic-navigation*

- TF sử dụng lệnh:

*$ sudo apt-get install ros-kinetic-tf*

### 4.3 Cài đặt Hector Mapping

Sử dụng lệnh*:*

*$ sudo apt-get install ros-kinetic-hector-slam*

### 4.4 Biên dịch và source:

*$ cd ~/catkin\_ws*

*$ catkin\_make*

*$ source ./devel/setup.bash*

### 4.5 Cài đặt RPLidar trên Raspberry

-Cài đặt driver cho RPlidar:

+ Tải và giải nén gói rplidar trên https://github.com/Slamtec/rplidar\_ros

+ Copy gói rplidar vào catkin\_ws/src

+ Biên dịch lại catkin\_ws:

*$ sudo ~/catkin\_ws*

*$ catkin\_make*

-Cài đặt Hector SLAM: sudo apt-get install ros-kinetic-hector-slam

-Cài đặt Odometry: tạo package mới Odometry, tạo file xử lý Odometry

#include <ros/ros.h>

#include <tf/transform\_broadcaster.h>

#include <nav\_msgs/Odometry.h>

geometry\_msgs::Pose pose;

geometry\_msgs::Pose prepose;

void p\_sub (const geometry\_msgs::PoseStamped::ConstPtr& p\_msg) {

pose=p\_msg->pose;

}

int main(int argc, char\*\* argv){

ros::init(argc, argv, "odometry\_publisher");

ros::NodeHandle n;

ros::Publisher odom\_pub = n.advertise<nav\_msgs::Odometry>("odom", 100);

tf::TransformBroadcaster odom\_broadcaster;

ros::Subscriber cmd\_vel=n.subscribe("slam\_out\_pose" , 100 ,p\_sub);

ros::Time current\_time, last\_time;

current\_time = ros::Time::now();

last\_time = ros::Time::now();

ros::Rate r(10);

double vx = 0.0;

double vy = 0.0;

double vth = 0.0;

while(n.ok()){

ros::spinOnce(); // check for incoming messages

current\_time = ros::Time::now();

//compute odometry in a typical way given the velocities of the robot

double dt = (current\_time - last\_time).toSec();

vx=(pose.position.x-prepose.position.x)/dt;

vy=(pose.position.y-prepose.position.y)/dt;

tf::Quaternion q(prepose.orientation.x, prepose.orientation.y, prepose.orientation.z, prepose.orientation.w);

tf::Matrix3x3 m(q);

double roll, pitch, oyaw,yaw;

m.getRPY(roll, pitch, oyaw);

tf::Quaternion q2(pose.orientation.x, pose.orientation.y, pose.orientation.z, pose.orientation.w);

tf::Matrix3x3 m2(q2);

m2.getRPY(roll, pitch, yaw);

vth=(yaw-oyaw)/dt;

//ROS\_INFO("dt= %lf,pose.position.x=%lf, yaw=%lf ", dt,pose.position.x,yaw);

//since all odometry is 6DOF we'll need a quaternion created from yaw

geometry\_msgs::Quaternion odom\_quat = pose.orientation;

//first, we'll publish the transform over tf

geometry\_msgs::TransformStamped odom\_trans;

odom\_trans.header.stamp = current\_time;

odom\_trans.header.frame\_id = "odom";

odom\_trans.child\_frame\_id = "base\_link";

odom\_trans.transform.translation.x = pose.position.x;

odom\_trans.transform.translation.y = pose.position.y;

odom\_trans.transform.translation.z = 0.0;

odom\_trans.transform.rotation = odom\_quat;

//send the transform

odom\_broadcaster.sendTransform(odom\_trans);

//next, we'll publish the odometry message over ROS

nav\_msgs::Odometry odom;

odom.header.stamp = current\_time;

odom.header.frame\_id = "odom";

//set the position

odom.pose.pose=pose;

//set the velocity

odom.child\_frame\_id = "base\_link";

odom.twist.twist.linear.x = vx;

odom.twist.twist.linear.y = vy;

odom.twist.twist.angular.z = vth;

//publish the message

odom\_pub.publish(odom);

last\_time = current\_time;

prepose=pose;

r.sleep();

}

}

## 5. Lắp ráp trên Robot

A) Lắp ráp cụm động cơ:

- Sơ đồ:

- Hình ảnh

A close up of a device

Description automatically generated

Hình 12 Cụm động cơ

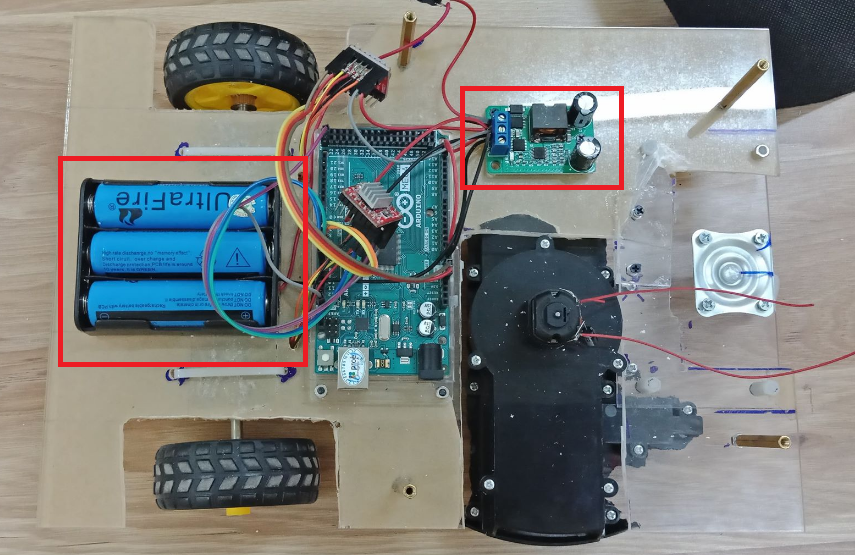
B) Lắp ráp cụm cảm biến

A picture containing floor, table, sitting, indoor

Description automatically generated

Hình 13 Cụm cảm biến

C) Lắp ráp cụm nguồn:



Hình 14 cụm nguồn

D) Hình ảnh hoàn chỉnh

A picture containing floor, indoor, small

Description automatically generated

Hình 15 Hình ảnh hoàn chỉnh

## 6 Xây dựng giải thuật điều khiển Robot:

### A/ Điều khiển động cơ bước:

- Nguyên lý hoạt động:

Phần điều khiển động cơ được viết thành một node hoạt động trên ROS nhằm thực thi các lệnh điều khiển từ navigation stack. Dữ liệu điều khiển được gửi trên topic cmd\_vel ở dạng thông điệp Twist gồm các thông tin về vận tốc và vận tốc góc theo ba trục x,y,z. Các dữ liệu này sẽ được xử lý để cho ra các số lượng các xung tác động lên hai động cơ bước.

- Code

|  |
| --- |
| #include "ros.h" |
|  | #include "geometry\_msgs/Twist.h" |
|  | #define STEP1 5 |
|  | #define DIR1 4 |
|  | #define STEP2 7 |
|  | #define DIR2 6 |
|  | float x; |
|  | float z; |
|  | long step1; |
|  | long kc; |
|  | ros::NodeHandle nh; |
|  | void velCallback( const geometry\_msgs::Twist& vel\_cmd) |
|  | { |
|  | x = vel\_cmd.linear.x; |
|  | z = vel\_cmd.angular.z; |
|  | if(x==0&&z!=0) |
|  | x=0.2; |
|  | } |
|  | ros::Subscriber<geometry\_msgs::Twist> sub("cmd\_vel" , velCallback); |
|  | void setup() |
|  |  |
|  | { |
|  | nh.initNode(); |
|  | nh.subscribe(sub); |
|  | pinMode(STEP1, OUTPUT); // Step |
|  | pinMode(DIR1, OUTPUT); // Dir - chiều quay |
|  | pinMode(STEP2, OUTPUT); // Step |
|  | pinMode(DIR2, OUTPUT); // Dir - chiều quay |
|  |  |
|  | } |
|  | void tinh (float x) |
|  | { |
|  | step1 = x / 0.00047124; |
|  | kc = 1000000 / step1;//microsecond |
|  | //in log |
|  | char log\_msg[50]; |
|  | char str\_x[6]; |
|  | dtostrf(x, 4, 2, str\_x); |
|  | sprintf(log\_msg,"x=%s,step1=%ld, kc=%ld",str\_x,step1,kc); |
|  | nh.loginfo(log\_msg); |
|  | } |
|  | long buoc (float x, float z) |
|  | { |
|  | long stepp = (x - tan(z) \* 0.15) / 0.00047124; |
|  | long tyle = step1 / stepp; |
|  |  |
|  | return tyle; |
|  | } |
|  | void loop() |
|  | { |
|  | nh.spinOnce(); |
|  | tinh (x); |
|  | Serial.println(step1); |
|  | Serial.print(kc); |
|  | digitalWrite(DIR1, LOW); // Set Dir high - Set chiều quay mức cao |
|  | digitalWrite(DIR2, HIGH); // Dir - chiều quay |
|  | long tyle=buoc(x, z); |
|  | //tiến thẳng |
|  | if (z == 0) |
|  | for (long i = 0; i < step1; i++) |
|  | { |
|  | digitalWrite(STEP1, HIGH); // Output high - xung mức cao |
|  | digitalWrite(STEP2, HIGH); // Output high - xung mức cao |
|  | delayMicroseconds(kc / 2); |
|  | digitalWrite(STEP1, LOW); // Output low - xung mức thấp |
|  | digitalWrite(STEP2, LOW); // Output low - xung mức thấp |
|  | delayMicroseconds(kc / 2); // Trễ kc/2 ms |
|  | } |
|  | else if ( z > 0) |
|  | // trái:phải(re trai) |
|  | for (long i = 1; i <= step1; i++) |
|  |  |
|  | { if (i % tyle == 0) digitalWrite(STEP1, HIGH); //Output high-xung mức cao |
|  | digitalWrite(STEP2, HIGH); // Output high - xung mức cao |
|  | delayMicroseconds(kc / 2); |
|  | if (i % tyle == 0) digitalWrite(STEP1, LOW); //Output low-xung mức thấp |
|  | digitalWrite(STEP2, LOW); // Output low - xung mức thấp |
|  | delayMicroseconds(kc / 2); |
|  | } |
|  | Else |
|  | //rẽ phải |
|  | for (long i = 1; i <= step1; i++) |
|  |  |
|  | { if (i % tyle == 0) digitalWrite(STEP2, HIGH);//Output high - xung mức cao |
|  | digitalWrite(STEP1, HIGH); // Output high - xung mức cao |
|  | delayMicroseconds(kc / 2); |
|  | if (i % tyle == 0) digitalWrite(STEP2, LOW); // Output low - xung mức thấp |
|  | digitalWrite(STEP1, LOW); // Output low - xung mức thấp |
|  | delayMicroseconds(kc / 2); |
|  | } |
|  |  |
|  | } |

### B/ Điều khiển mục tiêu di chuyển của robot:

- Khám phá bản đồ: sử dụng gói cung cấp dịch vụ khám phá bản đồ explore\_lite

- Lập kế hoạch di chuyển

Lập kế hoạch di chuyển theo các điểm chưa được thăm trên bản đồ 2D.

# CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ

## 1.Thử nghiệm:

### A) Thử nghiệm giải thuật hector mapping trong vẽ bản đồ 2D

- Bố trí thử nghiệm:

+ Mục đích : nghiên cứu khả năng sử dụng RPlidar A1 và hector mapping để vẽ bản đồ 2D

+Nơi thí nghiệm: phòng 309 nhà thí nghiệm và khu vực hành lang.

+ Các thiết bị: Raspberry Pi , RP Lidar A1 và máy chủ xử lý

- Kết quả: với tốc độ di chuyển chậm

A screenshot of a map

Description automatically generated

Ta thấy bản đồ được vẽ khá chính xác.

A screenshot of a map

Description automatically generated

Bản đồ được vẽ không chính xác, các chướng ngại vật và tường bị nhận diện trùng lặp và chồng lên nhau.

- Đánh giá: vì tốc độ của RPlidar A1 khá thấp (5-10Hz) nên giải thuật hector mapping chỉ hoạt động chính xác khi vận tốc di chuyển là chậm.

### B) Thử nghiệm điều khiển robot bằng navigation stack

+ Mục đích : thử nghiệm khả năng tích hợp hector mapping với navigation stack để định hướng robot, đồng thời đánh giá hoạt động của cụm động cơ.

+ Các thiết bị: Robot được lắp ráp hoàn chỉnh gồm cụm động cơ và cụm cảm biến.

- Kết quả: Robot đã nhận lệnh di chuyển theo đúng hướng người dùng ra lệnh và lộ trình đã tránh được các chướng ngại vật. Tuy nhiên, thực tế robot di chuyển chưa chính xác so với hướng được yêu cầu.

- Đánh giá: Hệ thống Navigation Stack và Hector Mapping đã hoạt động tốt. Vấn đề là do cụm động cơ chưa vận hành một cách chính xác theo lệnh nhận được.

## 2.Đánh giá

Qua các thử nghiệm cho thấy sự khả thi khi kết hợp navigation stack và hector mapping trong việc xây dựng bản đồ và dẫn hướng cho robot di chuyển. Kết quả thu được phản ánh hiệu quả của phương pháp cũng như các hạn chế của cảm biến RP Lidar A1 và cụm động cơ. Các vấn đề này sẽ được giải quyết bằng cách hạn chế tốc độ của robot và tinh chỉnh sự hoạt động của cụm động cơ cho hiệu quả và chính xác hơn.

# CHƯƠNG 5: KẾT LUẬN

## Các kết quả đạt được:

Qua quá trình thực hiện đề tài, nhóm nghiên cứu đã tiến hành khảo sát và tìm hiểu các công nghệ trong phát triển robot như hệ điều hành ROS và các thiết bị phần cứng Adruino,Raspberry Pi, RP Lidar và các loại cảm biến khác. Để phát triển robot hút bụi thông minh nhóm đã đi sâu nghiên cứu và áp dụng các công nghệ như navigation stack và hector mapping cũng như các phương pháp điều khiển động cơ và lắp ráp robot. Kết quả về mặt nghiên cứu, nhóm đã cài đặt và thử nghiệm các giải thuật trên với các nền tảng khác nhau như Raspberry Pi và máy chủ trên phòng thí nghiệm. Về mặt ứng dụng nhóm đã phát triển một nguyên mẫu robot hút bụi thông minh có thể vẽ bản đồ và di chuyển một cách tự động.

## Hướng phát triển:

Tiếp tục hoàn thiện các thành phần của robot để robot có thể hoạt động chính xác và thông minh hơn. Đồng thời phát thiển các công cụ giúp người dùng có thể quản lý và điều khiển robot trên điện thoại qua mạng nội bộ hoặc mạng internet. Cùng với đó việc mở rộng các chức năng của robot ví dụ như điều khiển bằng giọng nói, camera giám sát , thiết bị liên lạc,… cũng là các hướng phát triển nhiều tiềm năng.

# Tài liệu tham khảo:

[1] <https://www.slideshare.net/lexNTA/xy-dng-module-nh-v-v-tinh-chnh-xc-cao>

[2] <http://wiki.ros.org>

[3] Hector SLAM for robust mapping in USAR environments ROS RoboCup Rescue Summer School Graz 2012 Stefan Kohlbrecher (with Johannes Meyer, Karen Petersen, Thorsten Graber)

[4]<https://www.stdio.vn/articles/gioi-thieu-tong-quat-ve-slam-405>

[5]Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, Kurt Konolige.The Office Marathon: Robust Navigation in an Indoor Office Environment. Willow Garage Inc., USA

# [6]ROS Navigation: Concepts and Tutorial- João A Fabro.

[7]Eitan Marder-Eppstein, Eric Berger, Tully Foote, Brian Gerkey, Kurt Konolige.