**Chương 1: Tìm hiểu về ROS**

**1.1 Giới thiệu về Robot:**

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.

**1.2 Các thành phần trong ROS**

**1.2.1** Node

ROS nodes là quá trình dùng ROS API để giao tiếp với nhau. Một robot có thể có nhiều node để thực hiện quá trình giao tiếp.Hệ điều hành ROS được thiết kế mô-dun 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,..

**1.2.2** 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.

**1.2.3** 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

**1.2.4** 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.

**1.3 Các lệnh cơ bản trong ROS**

**1.3.1** Sử dụng rospack

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

Sử dụng: $ rospack tìm [pack\_name]

Thí dụ: $ rospack tìm roscpp

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

**1.3.2** 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 bạn đã tạo trong [hướng dẫn Tạo không gian làm việc cho catkin](http://wiki.ros.org/catkin/Tutorials/create_a_workspace) :

# Bạn 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 bạn đã catkin\_create\_pkg . Catkin\_create\_pkg yêu cầu bạn cung cấp cho nó một tên\_bảng và tùy chọn một danh sách các phụ thuộc mà gói đó phụ thuộc vào:

# Đây là một ví dụ, đừng cố chạy nó

# 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) .

**1.3.3** Roscore

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

Thử chạy: $ roscore

**1.3.4** Rosnode

Mở một terminal mới, và hãy sử dụng rosnode để xem những gì roscore đã thực thi

**Chú ý**: Khi mở một thiết bị đầu cuối mới, môi trường của bạn được đặt lại và tập tin ~ / .bashrc của bạn được lấy ra. Nếu bạn gặp rắc rối khi chạy các lệnh như rosnode, bạn có thể cần thêm một số tập tin thiết lập môi trường vào ~ / .bashrc của bạn.

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

Bạn 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

**1.3.5** 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

**1.3.6** 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 bạn 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 bạn dùng (indigo, jade, kinetic)

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

Bạn sẽ thấy giống như sau:



Nếu bạn đặt con trỏ chuột của bạn 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.

**1.3.7** Dùng rostopic

Công cụ rostopic cho phép bạn lấy thông tin về các chủ đề ROS. Bạn 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

**1.3.7.1** 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

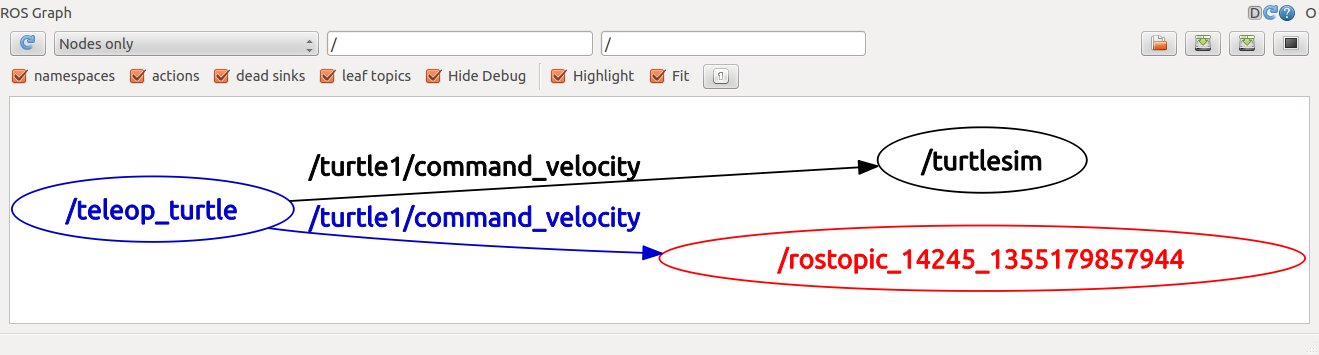
-v, --verbose : liệt kê đầy đủ chi tiết về từng chủ đề

-p : chỉ liệt kê các nhà xuất bản

-s : chỉ liệt kê những người đăng ký

Chúng ta hãy nhìn vào lệnh dữ liệu vận tốc xuất bản bởi nút turtle\_teleop\_key.ơn Đối với ROS Hydro và mới hơn, dữ liệu này được xuất bản trên chủ đề / turtle1 / cmd\_vel. Trong một thiết bị đầu cuối mới, chạy: $ rostopic echo /turtle1/cmd\_vel

Bây giờ chúng ta hãy nhìn lại rqt\_graph. Nhấn nút làm mới ở phía trên bên trái để hiển thị nút mới. Như bạn có thể nhìn thấy echo rostopic, hiển thị ở đây màu đỏ, bây giờ cũng đã đăng ký chủ đề  turtle1 / command\_velocity.



**1.3.7.2** 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

**1.3.8** 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

LƯU Ý: Thư mục lưu trữ các tệp khởi chạy không nhất thiết phải được đặt tên là khởi chạy. Trong thực tế, bạn 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 Navigation stack và Hector mapping**

**2.1 Navigation stack.**

**2.1.1** 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.

Tuy nhiên, ROS cũng có một node khác là ACML, 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 (động lực học) chuyển động 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 lưới ô vuông vi phân (diﬀerential drive) và các robot hình tròn, 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.

**2.1.2** Các thành phần trong navigation stack.

Gồm:

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.

**2.2 Hector Mapping**

**2.2.1**.Giới thiệu về Hector Mapping

Hector\_mapping là một cách tiếp cận SLAM mà được sử dụng mà không cần odometry cũng như là nền tảng để đưa ra các hành động cuộn/cắm (của cảm biến). Nó thúc đẩy 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.

**2.2.2**. Các thành phần trong Hector Mapping

Để sử dụng hector\_mapping , bạn cần một nguồn dữ liệu sensor mss / 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 hình học 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.

**2.2.2.1**. Subscribed Topics

scan (sensor\_msgs/LaserScan)

• Quét laser được sử dụng bởi hệ thong SLAM

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.

**2.2.2.2**. Published Topics

map\_metadata (nav\_msgs/MapMetaData)

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

map (nav\_msgs/OccupancyGrid)

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

**2.2.2.3**. 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 luận án 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 (http://tedusar.eu/cms/sites/tedusar.eu.cms/files/Hector\_SLAM\_USAR\_Kohlbrecher\_RRSS\_Graz\_2012.pdf?fbclid=IwAR2cX0UG\_CuUw0CtrC\_jPkL6GFqAP22GpLvX1qMxt4FdI7hn0VGU-YjND3U)

**2.2.2.4**.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’

**2.2.2.5**. 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

**2.2.2.6**. Attitude Estimation

• Ước tính cuộn / cắm của nền đất /LIDAR cần thiết (Roll/Pitch Estimation of platform/LIDAR required)

• 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

**2.2.2.7**. 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 .

**2.2.2.8**. 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

**2.2.2.9**. 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**

**3.1 Giới thiệu các chức năng của Robot**

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. Tuy nhiên, nó chỉ có thể loại bỏ các hạt bụi rời rạc, chứ không có khả năng hút bụi nếu bụi bẩn bị dính bết xuống sàn.

Ngoài ra, Robot có sự tích hợp của nhiều tính năng thông minh, đặc biệt là chức năng thiết lập bản đồ đã giúp ích rất nhiều trong quá trình hoạt động của Robot.Với chức năng này, 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

**3.4 Cài đặt Navigation Stack và Hector Mapping**

**3.4.1** Cài đặt ROS (http://wiki.ros.org/ROSberryPi/Installing%20ROS%20Kinetic%20on%20the%20Raspberry%20Pi)

**3.4.2** 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

**3.4.3** Download từ https://github.com/tu-darmstadt-ros-pkg/hector\_slam rồi vào thư mục catkin\_ws/src trên Raspberry

**3.4.4** Biên dịch và source:

- cd ~/catkin\_ws

- catkin\_make

- source ./devel/setup.bash

Tài liệu tham khảo

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

<http://wiki.ros.org>