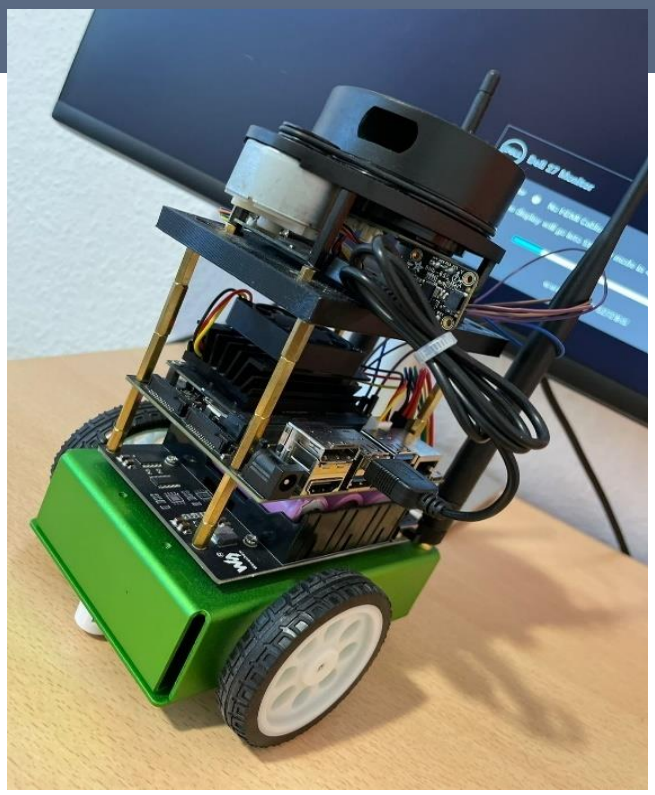
2022



JETBOT NAVIGATION

Guided by: Prof. Dr. Frank Schrödel

(PROJECT WORK)

Abstract

[Indoor autonomous single point navigation-RoPro bot]

Urvashiba Parmar

Mechatronics and Robotics, Hochschule Schmalkalden

Task

The goal of this project is to build an autonomous indoor single point navigation robot using Jetbot, RPLIDAR A1 laser sensor and Jetson nano B01 board. No IMU and encoders are used which makes it cost efficient. Options and selection for the hardware and software can be found in the **Padlet**(access via QR-Code).

Agenda

→ How to build this robot?

→ How to select sensors, actuators, computational boards for the robot?

→ How to program the robot and install Software frameworks for programming this robot?

Index

1. Introduction

1.1. Motivation

Autonomous Navigation is referred as the most advanced and challenging ability in mobile robotics that includes key components such as Mapping, Localization, Path planning and Navigation.

Navigation of a robot is enabled by scanning its surroundings to find the obstacles and determine the hurdles in it ‘s path towards destination while computing the obstacle free path. The capability of a robot to perceive a relative position with respect to the target helps in Navigation without collision with the impediments and reaching via optimal path towards a destination.

With advancement of new technology, the capability of Robotics has increased to ten folds and their complexity. The era of self driving cars, modern-age computers with automation features enables their application and gives a great idea of Robotics and numerous automobile companies have been expanding their research and development in this field to bring self driving cars into a reality for human dreams. In order to get real hands on experience, I decided to learn from scratch and build a practical Prototype using Robotic Operating System (ROS). My real motivation was also to discover and bring clarity on the mysterious path of building ROS based robotics project. This not only helped me in learning the framework but also gave me several new ideas for real world application with ROS based Projects that could be used as **“Out of the BOX”** solutions in a short period of time.

In the past, prior to the appearance of ROS, a huge amount of time was invested in building the software infrastructure that can be integrated with the embedded parts, sensors, drivers and actuators. With advent of ROS, the open source community has played a vital role in saving this dependency and bringing libraries, and packages at a ready to use stage that are considered as building modules for every project in Robotics with easily replicable tools over the internet community.

1.2. Basics about mobile robots

ROS is a meta-operating system for the robots, as per the WIKI definition about ROS. Since ages we know about OS for example Linux, windows, Macintosh for computers. As beginner I used to think robot operating system , ROS, itself declares as an operating system but is it a real operating system? Well, it is not as conventional as windows or Macintosh or any other operating system but it is an operating system that runs on an existing operating system. To run it you need to install an operating system like Ubuntu as it is not a stand alone operating system. Answer for the question why ROS would be it offers a standard software platform to

developers across industries that will carry them from research and prototyping all the way through to deployment and production, it is 100% open source, provides multi-platform and ready to use robot application. Whether indoor or outdoor, underwater or space, industrial or home application it provides multi-domain support.

In the following section, theoretical explanations for necessary topics including research are elaborated.

Kinematics of differential drive: A differential drive is a two-wheel drive where each wheel driven independently. The motion of each wheel is controlled by a DC motor. A DC motor receives voltage as input and produces torques on the motor drive axis that spins the wheels. The movement of the wheels will produce an evolution of the robot's pose over time which is what we want to quantify.

We know that the robot pose is the position and orientation of the body frame(x^r,y^r) with respect to the world frame(x^w,y^w)[1]. As shown in figure1.1 the origin of coordinate A is located in the middle of the axis. Let us assume that the robot is symmetrical about its longitudinal axis, that is, the x-direction of the robot frame. Therefore, the wheels are assumed to have the same diameter (2R) and both are at a distance L from point A. The center of gravity of the robot is on the x-axis and at a distance C from point A.

Here, World frame: {x^w,y^w} And Robot body frame: {x^r,y^r}

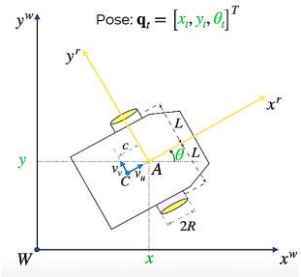


Figure 1.1: Differential drive robot [Self-Driving Cars with Duckietown]

Suppose: Robot is symmetric along x direction and robot is a rigid body. That is, the distance between two points on the robot does not change over time.

To obtain a frame-based representation of the world, first determine the velocity of point A within the robot body.

$$V_A{}^r = \begin{bmatrix} V_u \\ V_v - c\dot{\theta} \end{bmatrix}$$

Now we switch from the robot body frame to the world frame using a rotation matrix defined by the orientation angle (θ).

$$V_A^w = \begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} = R_\theta V_A^r = \begin{bmatrix}cos\theta & -sin\theta\\ sin\theta & cos\theta\end{bmatrix}\begin{bmatrix}V_u\\ V_v - c\dot{\theta}\end{bmatrix}$$

The general kinematics equation can be expressed as,

$$\dot{x} = V_u cos\theta - (V_v - c\dot{\theta})sin\theta$$

$$\dot{y} = V_v sin\theta + (V_v - c\dot{\theta})cos\theta$$

$$\dot{\theta} = \omega$$

To simplify the model lets make an additional pure rolling assumption.

1.) Moving no sideways: There will be no lateral movements means no lateral velocity.

$$V_A^r = \begin{bmatrix}V_u\\ V_v - c\dot{\theta}\end{bmatrix} = \begin{bmatrix}V_u\\ 0\end{bmatrix} \quad (\because V_v - c\dot{\theta} = 0)$$

2.) Wheels are not slipping: No slipping implies that every full rotation each wheel travels a distance equal to its circumference.

Generalized equations of vehicle speed and vehicle heading angular velocity:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}cos\theta & 0\\ sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}V\\ \omega\end{bmatrix}$$

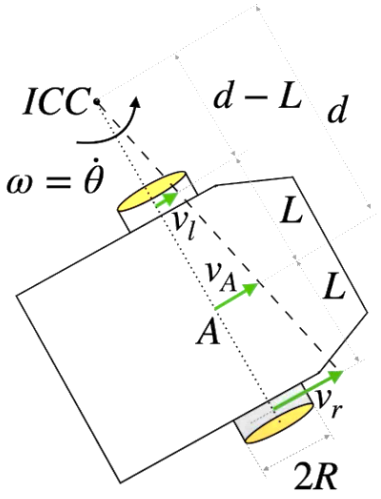
Where,

$$V(vehicle\ speed) = \frac{R}{2}(\dot{\phi}_l + \dot{\phi}_r) \text{ and}$$
$$\omega(angular\ velocity) = \frac{R}{2}(\dot{\phi}_r - \dot{\phi}_l)$$

Wheel speed equation,

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\frac{R_r}{2}cos\theta & \frac{R_l}{2}cos\theta\\ \frac{R_r}{2}sin\theta & \frac{R_l}{2}sin\theta\\ \frac{R_r}{2L} & -\frac{R_l}{2L}\end{bmatrix}\begin{bmatrix}\dot{\phi}_r\\ \dot{\phi}_l\end{bmatrix}$$

As the robot rotates without slipping, all of its points have velocity equal to the robot's angular velocity multiplied by the distance to the ICC(Instantaneous center of curvature), which is the only point that does not move in a rotational field and whose direction is orthogonal to that identified passes through the center of curvature by controlling the distance of the ICC according to the linear speed of the wheel.



Instantaneous center of curvature(ICC):

Assume there's no slipping→ All points in a pure rotation filed have velocity orthogonal to distance to ICC.

$$V_l = \omega(d - L)$$

$$V_r = \omega(d + L)$$

$$V_A = \omega\, d$$

$$\therefore d = L\,\frac{V_r + V_l}{V_r - V_l}$$

Figure 1.2: Understanding robots behavior using ICC[Self-Driving Cars with Duckietown]

There are three interesting situations for the differential drives(ideally):

1) If $V_r = V_l$ ⇒ There will be no turn because distance(d) is undefined. We will have only linear motion.
2) If $V_r = -V_l$ ⇒ So, d=0 therefore robot will rotate on the spot(ICC=A).
3) If $V_r = 0$ or $V_l = 0$ ⇒ Here, d = -L or d = L. in first case robot will rotate about the right wheel and in second case it will rotate about the left wheel.

Forward kinematics of a differential drive robot which is particularly useful when determining future estimation of the robot state. And the inverse kinematics model obtained by inverting the relation between wheel and vehicle velocities allows us instead to map the desire vehicle velocities to wheel commands which is particularly useful in controlling the robot.

Mapping: The process of creating a map of an environment is called mapping. A map is a picture of the environment created by the robot's sensor measurements. Here a LIDAR sensor based on laser detection is used. To do mapping with Jetbot, ROS's SLAM package is used. Teleoperation is used to control the robot motion to map the environment.

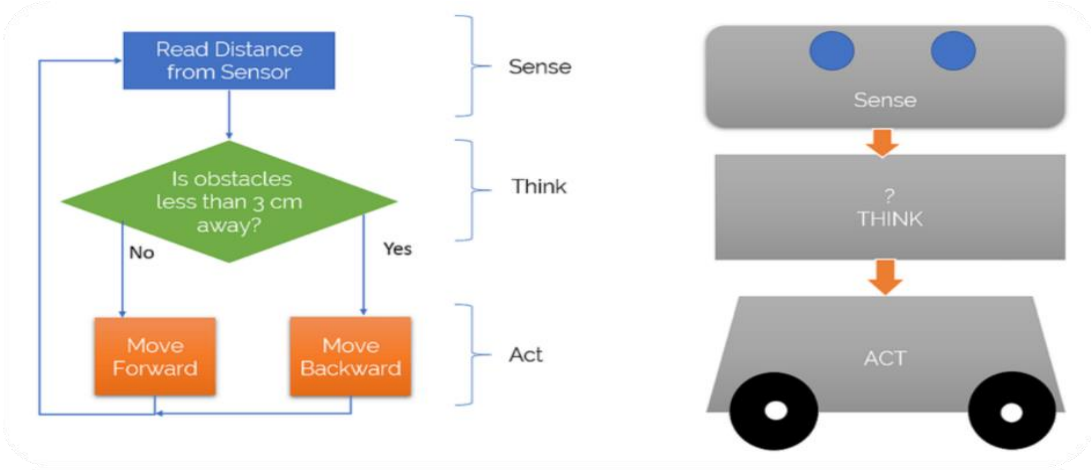


Figure 1.3: Basic robot structure

Localization: The technique of finding a mobile robot in relation to its surroundings is known as robot localization. One of the most important skills needed by an autonomous robot is localization because knowing one's own location is a necessary step in deciding what to do next. In a typical robot localization situation, the environment is mapped out and the robot is outfitted with sensors that keep an eye on both the outside world and its own movement. The extended Kalman filter (EKF) and the particle filter are two popular algorithms for combining sensor data to estimate the robot's location.

SLAM: Simultaneous localization and mapping (SLAM) is a technology that allows you to create a map while also localizing your vehicle within that map. The robot can map unexplored environments using SLAM techniques.

Imagine robot wants to go from point 1 to point 2 avoiding obstacles. what it needs to reach the goal?

1.position: Estimating the robot's position.

2.Sensing: Knowing environment such as walls, objects and obstacles.

3.Map: Getting environment information.

4.Path: Finding nearest path to goal point and follow it.

Navigation: One of the most important topics in localization that signifies the ability of a robot to recognize it's position and orientation within the frame of reference is Path planning, Starting from determination of it's location in the frame of reference to reaching and planning a path towards a destination in same frame. While navigating in an environment, the goal of a robot is to avoid every obstacle that may come in it's path.

For an execution of a ROS based robot is the Navigation Stack covering from the conceptual level where it retrieves the data from the odometry and sensor streams. An output signal is transferred to a mobile base.

2 Planning of the project

2.1 V-model method

The figure below depicts a V-model view of product development. By definition, a v-model has a verification phase to the left and a validation phase to the right of a v-model. Verification is a set of objective tests that confirm that a product meets requirements, while validation is an attempt to prove that a product meets its original intent. V models focus more on the design phase than the execution. The coding step connects the two sides of the model.

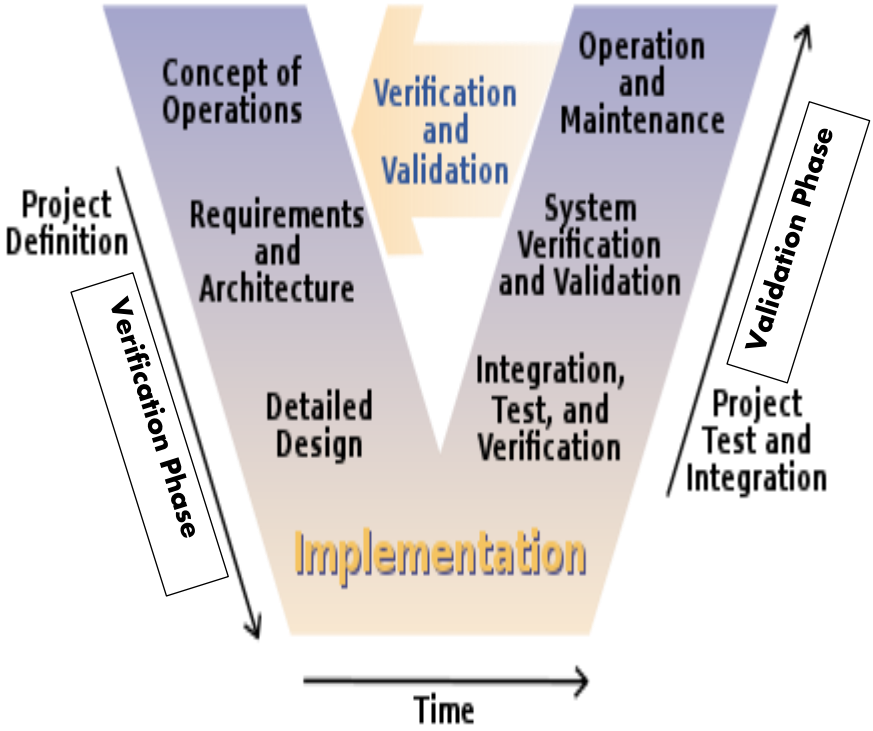


Figure 2.1 : V-model[en.wikipedia.org/wiki/V-Model]

V-Model also known as the Verification and Validation Model[3]. A limitation of the V model is the implementation of requirements at the concept stage. This does not apply to all product development. Requirements, models and evaluations are often combined and repeated many times before they are accepted.

Verification: It involves a static analysis method done without executing code. Process of evaluation of the product development process to find whether specified requirements meet is known as Varification phase in V-model.

Validation: It involves dynamic analysis method, testing is done by executing code. The process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements is known as Validation phase in V-model.

Various phases of Verification Phase of V-model:

Requirement analysis: This is the first step where product requirements understood from the customer's side. This phase includes detailed communication to understand the client's expectations and exact requirements.

1. **System Design:** In this phase, the system engineer analyzes and interprets the business of the proposed system by studying the user requirements document.
2. **Module Design:** The system breaks down into small modules In the module design phase.
3. **Coding Phase:** The coding phase is started after designing phase. A suitable programming language is decided based on the requirements. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

Phases of Validation Phase of V-model:

1. **Unit Testing:** In the V-Model, Unit Test Plans are developed during the module design phase. These test plans are executed to eliminate errors at code level or unit level.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. System Test makes sure that expectations from an application are met.
4. **Acceptance Testing:** Acceptance tests are linked to the requirements analysis part. This also includes software testing. Acceptance testing shows compatibility issues with different systems. At the same time, it also discovers non-functional problems such as load and poor performance in real user environments.

Based on that the v-model for the project is created as shown below.

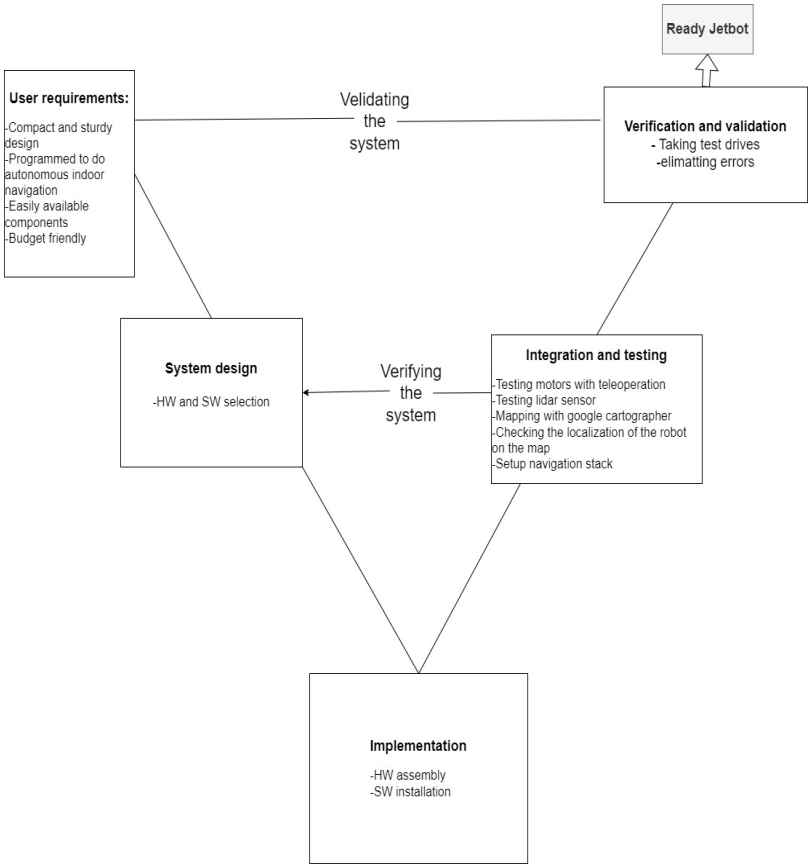


Figure 2.2: V-Model of the project

3 Concept and design

Project development starts with the concept phase. In this phase I have developed concept and set the user requirements. As per the user requirement such as robot should be lightweight and have robust design, able to reach destination without human assistance also should not be extravagant.

After obtaining clear and precise product requirements, the entire system must be designed. System design allows you to fully understand and describe the interactions between the hardware and the software. Based on the system design, a system test plan was developed. By doing this in advance, you will free up more time for the actual testing.

3.1 Hardware setup

Waveshare Jetbot, you might wonder why this robot. Why not other DIY kits and the answer is its highly integrated expansion board with safe and stable circuit design, also, interesting configuration, simpler design with possibility to expand with other development kits and high quality aluminum alloy chassis. Ahead you can find everything you need to know about how to get started, assembly and the complete set up.

While selecting the hardware the major concern has been given to selecting a cost-efficient module with low cost and future expansion opportunity to use it in other relevant projects like mobile application controlled robot, Food Delivering Prototypes, Logistics Robot.

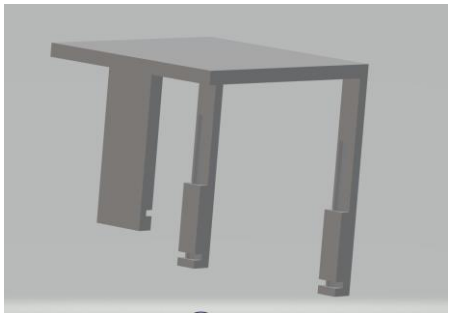


Figure 3.1 : First design

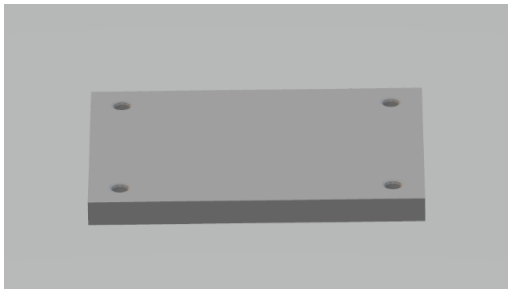


Figure 3.2 : Final plate design

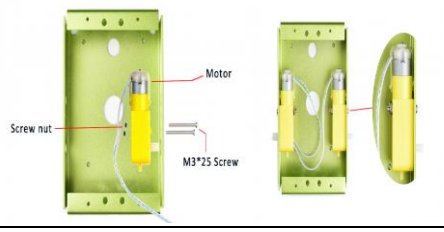
To mount the laser sensor a support structure was necessary. So designed a plate in Creo (designing software) and printed the part in university lab but the first design was not fitting perfectly on the board so decided to make a simple plate which can be implemented on the Jetbot by simply using standoffs as shown above in figure 3.2.

3.1.1 JETBOT Assembly [JetBot AI Kit Assemble Manual]

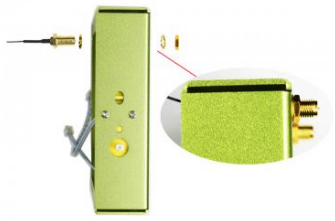
In this kit these components are included.

Figure 3.1.1 : Jetbot kit

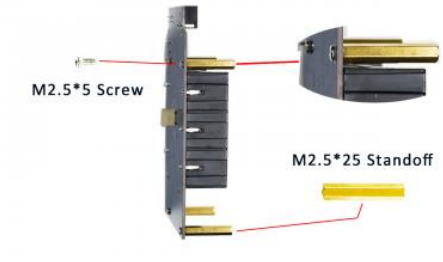
1. Assemble Motors:



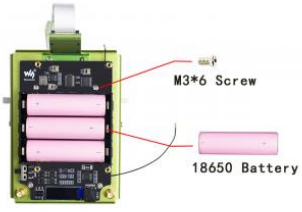
2. Setup antennas:



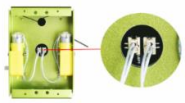
3. Set standoffs on expansion board:



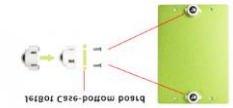
4. Mount expansion board on Jetbot as well as batteries:



5. Connect motor to the expansion board:

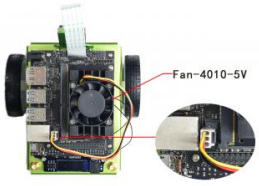


6. Connect omni wheels to the bottom board:

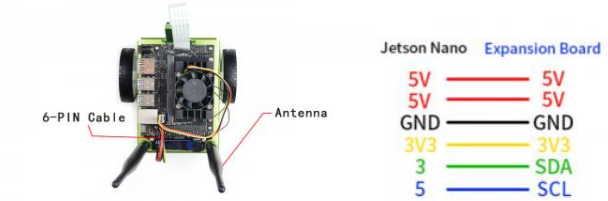


7. Connect jetson nano B01, assemble wheels and wireless AC-8265:

8. Connect the cooling fan

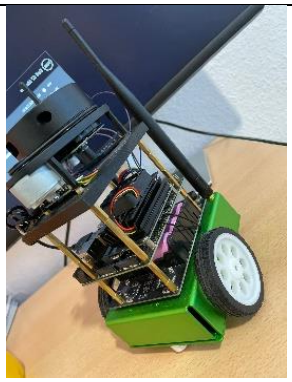


9. Connect jetson nano and expansion board:



10. Connect RPLIDAR with jetson nano and mount 3D printed plate using standoffs:

- The plate used to mount rplidar is 3d printed. Link for the plate is(https://drive.google.com/file/d/116rilGhIZq5-5HMqrqek_K2QAaGxpVy9/view?usp=sharing)



3.2. Software setup

In the following section, you will find the description about the required software configuration and tools used for building the robot.

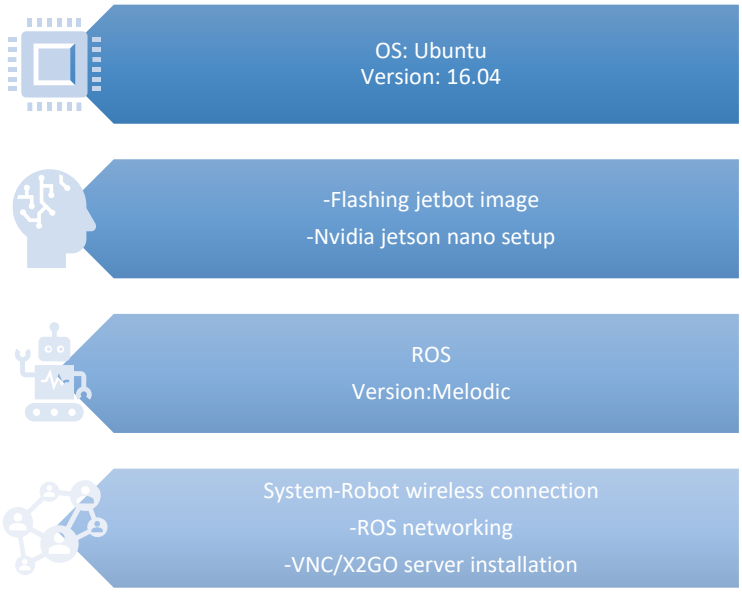


Figure 3.2.1: Software used in the project

- You need to have an 64G SD card at least.
- Download the JetBot image provided by NVIDIA and unzip it. Click to download .
- Connect the SD card to the system.
- Use Etcher software to write the image to SD card. Click to download Etcher software
- Insert SD card on jetson nano and connect jetbot to the power.
- Now, connect HDMI display, keyboard and mouse to the jetson nano B01.

- Configure the jetson nano and setup Wi-Fi using bellow code where replace <wifi-name> and <wifi-password> with your Wi-Fi configuration.

```
sudo nmcli device wifi connect <wifi-name> password <wifi-password>
```

- After the configuration Wi-Fi IP will display on the OLED display.
- So now that your jetson nano is configured, we can go ahead and install ROS Melodic on the system.
- Open a terminal window and paste below command, it configures your machine to receive software from packages.ros.org.

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

- Next, update your system's package list.

```
sudo apt update
```

- Now setup secure key so that our pc accepts our download.

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key C1CF6E31 E6
```

- We are going to install full desktop version of ROS Melodic. Which includes all tools along with simulation. During the process you will be asked to press Y to continue the process. It is going to take some time to install.

```
sudo apt-get update

sudo apt install ros-melodic-desktop-full
```

- Now press Y to continue the process and enter below command to setup environment variable.

```
echo "source /opt/ros/melodic/setup.bash" >> ~/. bashrc
```

- We better install some other tools too as we will need while working with ROS.

```
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python
-wstool build-essential
```

- Press Y to continue process
- To install all the dependencies, we will have while working we need rosdep on our system.

```
sudo apt install python-rosdep

sudo rosdep init

rosdep update
```

- Now if you want to confirm the Ros version enter

```
rosversion -d
```

- Open a new terminal and run below code to verify it has been installed without any problems.

```
roscore
```

Computer - Robot Wireless Communication: As we already know, ROS is a distributed computing environment that consists of several nodes spread over the multiple machines. Simplest example is controlling a ROS -based robot via a network of computers. These nodes enables the bi directional communication to achieve full connectivity, where every system or a machine has it's name to recognize the communication protocol.

The IP addresses used in our Prototype :

- XPS17: 192.168.0.109(client)
- ROS: 192.168.0.108(server)

Now open terminal on Jetbot and open .bashrc file(sudo gedit .bashrc) and add these lines:

Figure 3.2.2: Host setup on jetson nano

This will assign Jetbot as server and the same we will write in our client server which is our laptop. Vice versa

```
126 export ROS_MASTER_URI=http://192.168.0.108:11311/
127 export ROS_HOSTNAME=192.168.0.109
```

Figure 3.2.3: Client setup on PC

A Linux based servers follows the non- GUI based manual configurations, similarly in this project. I have followed the X2GO setup that supplies a remote-accessibility and better response in comparison to VNC server. An X2Go client-server application builds the connection that creates a robust GUI-environment for ROS operating system.

Once you setup x2go server you will see the screen like this, all work can be done from here now.

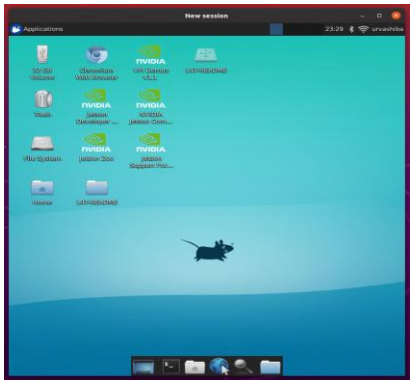


Figure 3.2.4: X2GO server screen

4 Implementation

In this section, you will find the core structure and program setup to prepare the development environment.

4.1 Robot programming

4.1.1 Catkin Workspace

A directory to build , modify and install the typical build system required for a ROS. It allows a better distribution of packages with ability to cross compile. It follows a CMAKE workflow with an extension of extensive “find Package” support and simultaneously multiple project instance setup.

- This is the place where we are going to put our code and build the package.
- More then one catkin packages can be built in this workspace.
- To make catkin ws please follow below steps.
- Go to the source folder by entering this code.

```
source /opt/ros/melodic/setup.bash
```

- Below command will make catking workspace and src folder.

```
mkdir -p ~/catkin_ws/src
```

- Now to go to the catkin_ws folder use below command.

```
Cd ~/catkin_ws
```

- To build the workspace use below code. Also when ever, you add or edit c++ code you have to build the package. Keep eye on code if there’s any error you have to locate error and try to solve it. Package is not built if error is there and your program will not work as expected.

```
Catkin_make
```

- We are in melodic so will use this command, for noetic catkin build is there.
- Most crucial step, source the catkin workspace. Without sourcing no code will work because your system doesn’t know the path of the folder and it will give error.

```
Source devel/setup.bash
```

4.1.2. Clone the repository

To maintain a version source control of the project files and historical changes, GitHub has been used in this project.

- In a new terminal, the following commands were executed :

```
cd catkin_ws/src

git clone “https://github.com/urvashiba/jetbotnavigation-RoPro.git”
```

4.1.3. Installing the dependencies

For a successful development, there has been multiple dependencies being installed in this project. Below are the few important dependencies mentioned with their commands.

- ROSDEP: The rosdep feature improves user ease by making it simple to install dependent packages while using or compiling ROS’s basic components. Make sure to install rosdep before using ROS.

```
rosdep install —-from-paths src —-ignore-src -r -y
```

```
sudo apt-get install ros-melodic-cartographer-ros

sudo apt-get install ros-melodic-navigation

sudo apt-get install ros-melodic-teb-local-planner

sudo apt-get install ros-melodic-dwa*
```

- You can install dependencies for individual package too.

```
rosdep install <package-name>
```

Install these dependencies to make our code run errorless.

4.1.4 Install google-cartographer ROS

Cartographer ROS Integration: Through a Cartographer, the system provides a 2D and 3D mapping to achieve the real-time localization and mapping for sensor configuration.

a.) Downloading and Compiling Cartographer

```
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build stow
cd
mkdir cartographer_ws
cd cartographer_ws
wstool init src
wstool merge -t src https://raw.githubusercontent.com/cartographer-
project/cartographer_ros/master/cartographer_ros.rosinstall
wstool update -t src
sudo apt install python-rosdep python-rosinstall python-rosinstall-generator python-
wstool build-essential
sudo apt install python-rosdep
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src –rosdistro melodic -y
src/cartographer/scripts/install_abseil.sh
sudo apt-get remove ros-melodic-abseil-cpp
catkin_make_isolated --install --use-ninja
```

b.) Create a new Workspace for Cartographer

In case you receive errors while running catkin, try the following commands:

```
source /opt/ros/melodic/setup.bash
```

Add Cartographer to .bashrc and source it

```
echo "source ~/cartographer_ws/devel_isolated/setup.bash"
>> ~/.bashrc
```

```
source ~/.bashrc
```

To Test the Cartographer is Sourced Properly, execute this command:

```
roscd carto[Press tab to check for auto-complete]
cd
```

If it doesn't tab auto-complete or roscd throws an error, re-check bashrc and sourcing.

5 Validation

After a successful preliminary integration of the environment setup and dependencies for robot, the following section illustrates the working sample from Jetbot in a closed environment.

First, we will assess our motors and move the robot using teleoperation.

```
rosrun Jetbot_ros motor_subscriber.py

rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Try moving Jetbot move forward, backward, clockwise and anticlockwise. Check its maximum speed and also minimum speed which we will need while setting up the navigation stack parameters. To check minimum speed press `z´ which will decrease the speed. Reduce until Jetbot stop running and mark that speed. Same press `q´ to increase the speed and increase it till Jetbot speed get constant and mark that speed.

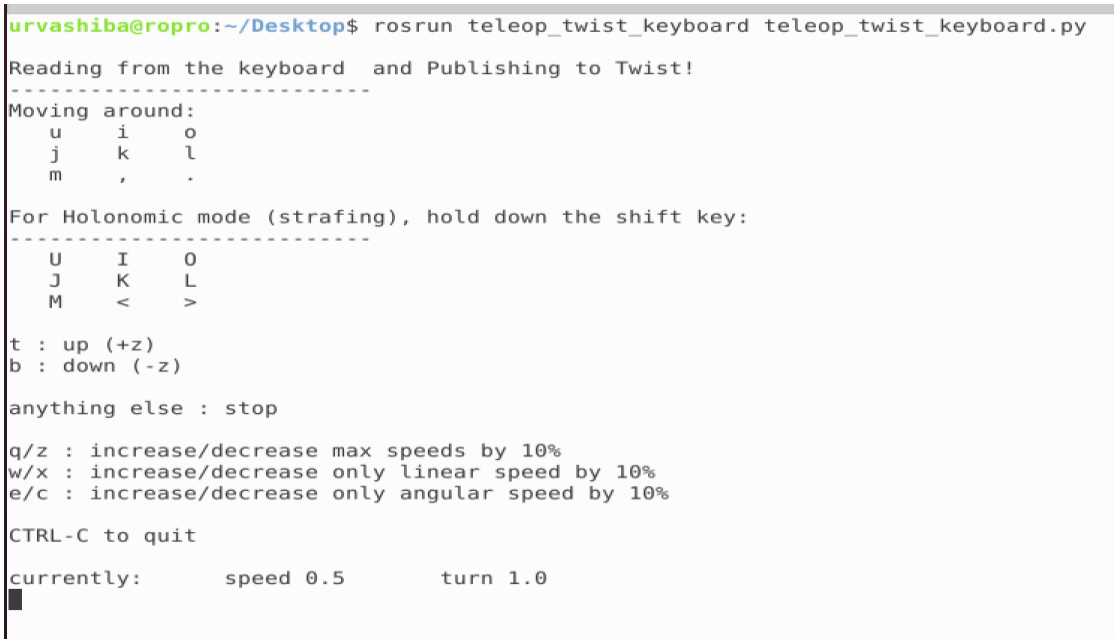


Figure 5.1: Teleoperation command window

To quit the process, enter **ctrl+c.**

Here Jetbot is running at 240 rpm speed and 80 PWM so I took velocity of the robot as 1.4m/s and using this value built the navigation stack package.

Next step is mapping using cartographer. Mapping file is located in the folder called x_series>launch>slam.launch. At the end you will get map like this in Rviz as shown in figure 5.2.

Figure 5.2: Map using Hector SLAM

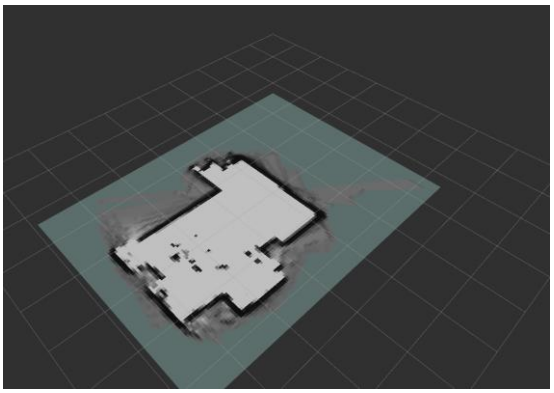


Figure 5.3: Map using Cartographer

Once you get a perfect map of the environment save it using below command.

```
rosservice call /finish_trajectory 0

rosservice call /write_state  "filename: '/home/urvashiba/catkin_ws/src/x_series/maps/cartographer_map.pbstream
```

To launch navigation file, Open a terminal in Jetbot and type.

```
roslaunch x_series navigation.launch
```

To check whether ROS networking has set perfectly try ping Jetbot IP or open a terminal and type rostopic list on client server (pc). This will give all the active topics names. If it gives error check setup and source .bashrc file(source ~/.bashrc). If the parameter setting is done perfectly, you will see this map.

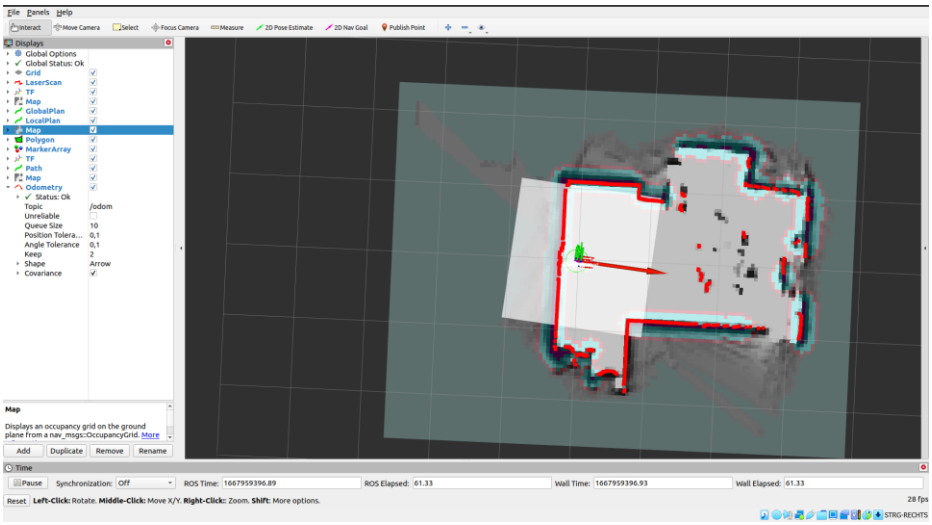


Figure 5.4: Rviz view of navigation file

Here red arrow along with a green circle depicts the position of the Jetbot in the map. To check whether odometry is precise or not mark one meter in the environment and run robot towards that point using teleoperation. 1 meter in environment = 1 box in rviz. Another way is to echo the odom topic and move Jetbot at 0.1m/s speed and see odometry is giving the same output.
Give a destination 2d navigation goal anywhere in the map and you will see a green path generation to the goal position which means you costmap setup is working well. After giving destination goal, if there are error in terminal, change costmap params until there are no more errors.
Via tf frames command, communication between all the nodes can be checked.

```
rosrun tf view_frames

evince frames.pdf
```

There are various methods for mapping and navigation in which mostly used algorithms are GMapping and Hector SLAM. The GMapping algorithm is based on the particle filter pairing algorithm, Hector-SLAM is based on the scan matching algorithm, and Cartographer is based on the scan matching algorithm with loop detection. Google's SLAM solution called Cartographer is a graph optimization algorithm. Google's open-source code consists of two parts: Cartographer and Cartographer_ROS. Cartographer's function is to process data from IMUs and odometers to create maps. Cartographer_ROS then captures sensor data via the ROS communication mechanism, converts it to his Cartographer format for processing by Cartographer, and releases Cartographer's processing results for display or storage.

Impressive real-time results for solving SLAM in 2D are described by the authors of the software[2].

First Hector SLAM was used along with AMCL (Adaptive Monte Carlo Localization) approach for localization. To compensate the odometer rf2o laser odometry was used but it wasn't efficient enough to provide odometry. Due to that while navigation robot was spinning on the position. For GMapping odometry is important and in our case there is no encoders or IMU so can not use it.

On other hand, Cartographers can provide decent quality 2D odometry using only cheap 360-degree LDS with fairly low data rates (5-7 Hz).Using an IMU and additional odometry sources (such as wheel platform odometry and visual odometry) can improve the quality of maps obtained over large surrounding areas. But for indoor mapping of about 50-60 square meters, they are less important. Cartographer's inner-loop closure algorithm can keep such small maps consistent. To tune Cartographer's parameters good understanding of it is necessary.

5.1. Results

In the rviz give 2D Nav Goal and robot will start moving and reaches the destination. During this navigation mainly two behaviors I have noticed:

1. Cost map is rotating along with the lidar due to that robot is recomputing its location in map the whole time.
2. Jetbot can reach the goal, but the trajectory is not straight.

6 Conclusion

In this project, I have achieved the wireless communication between the Jetbot and remote system, which builds the prerequisites for working of this robot. The main task of the project has enabled me to differentiate between Hector mapping and Cartographer mapping practically, it also helped me distinguish between different planners such as dwa and teb local planners. Both mapping has its own pro and cons.

Mapping with hector SLAM, rf2o odometry package was used which gives odometry using lidar. But it falls back giving output therefore while running navigation stack robot was not reaching the destination point and rather it goes to random places. Also due to the poor stability of the Jetbot wheels, best results are difficult to achieve. On other hand, although there are no encoders and imus, the cartographer provides accurate odometry and impressive map.

While running robot straight forward or backward laser scan and costmap working well, it rotates only when robot rotates. So tuning TRAJECTORY_BUILDER_2D.ceres_scan_matcher_translation_weight and rotation weight could help solve the issue. According to my research this issue is due to poor localization which can only be solved by tuning localization params.

7 Future scope

Migration from ROS1 to ROS2 is possible. To make navigation fine tuned encoders would be a good add-on. As velocity is important factor while setting up the costmap parameters. Without encoders calculated velocity is not precise which results unprecise movements of the robot. Also, it will improvise the location of the robot. AI functions can also be integrated, such as facial recognition and object detection with the camera included in Jetbot. For collision avoidance ultrasonic sensors would be advisable. As mentioned above, Jetbot wheels are not sturdy. There is noticeable vibration resulting in noise in the sensor results. Therefore, larger wheels or a good firm build would be effective. Also Testing different global and local planners can be done. To enable object detection or semantic segmentation with the RPi Camera.

Reference

[1] https://docs.duckietown.org/daffy/AIDO/out/embodied_tasks.html

[2] Cartographer ROS Documentation, The Cartographer Authors (https://google-cartographer-ros.readthedocs.io/en/latest/compilation.html)

[3] https://www.javatpoint.com/software-engineering-v-model