A REPORT OF SIX MONTHS INDUSTRIAL TRAINING

at

[LEAP ROBOTS LLP]

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD

OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

(Computer Science)

January – June, 2022

SUBMITTED BY:

SIMRANPREET KAUR

UG1833020019

DEPARTMENT OF COMPUTER SCIENCE

CT UNIVERSITY

CONTENT

CHAPTER 6 CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

6.2 Future Scope

6.2.1 First Step

6.2.2 Second Step

CANDIDATE'S DECLARATION

I **Simranpreet Kaur** hereby declare that I have undertaken 6 months of Online Software Training at **Leap Robots LLP** during a period from March 2022 to September 2022 in partial fulfilment of requirements for the award of degree of B.Tech (Computer Science) at CT UNIVERSITY, LUDHIANA. The work which is being presented in the training report submitted to Department of Computer Science and Engineering at CT UNIVERSITY, LUDHIANA is an authentic record of training work

Signature of the Student

The Software training Viva–Voce Examination of_____ has been held on _____ and accepted.

Signature of Internal Examiner Signature of External Examiner

CANDIDATE'S DECLARATION

I **Simranpreet Kaur** hereby declare that I have undertaken 6 months of Industry Oriented Project Training at **Leap Robots LLP** during a period from March 2022 to September 2022 in partial fulfilment of requirements for the award of degree of B.Tech (Computer Science) at CT UNIVERSITY, LUDHIANA. The work which is being presented in the training report submitted to Department of Computer Science and Engineering at CT UNIVERSITY, LUDHIANA is an authentic record of training work

Signature of the Student

The Industry Oriented Project Training Viva–Voce Examination of_____ has been held on _____ and accepted.

Signature of Internal Examiner Signature of External Examiner

ABSTRACT

Everyone who has worked with research knows how rewarding experimenting and developing new algorithms can be. However, in some cases, the hard part is not the invention of these algorithms, but their evaluation. To try and make that evaluation easier, this thesis focuses on the collection of data that can be used as positional ground truths using an autonomous measurement platform, which should assist Mobile Robots in the evaluation and improvement of their Wi-Fi based indoor positioning service.

Imagine a robot assisting a workspace such as in a university, the robot is performing various tasks from greeting people to assisting them to navigate to their desired locations, from cleaning floors regularly to transporting items from one place to another, from taking care of the labs to setting up the lab equipment to their desired positions. And this is possible by using a Camera and an LIDAR sensor – Light Detecting and Ranging in a ROS – Robotic Operating System Environment

The result of this thesis ends up recommending the following for measuring inside an unknown environment: the Simultaneous Localization and Mapping (SLAM) algorithm Google Cartographer for navigation, and the exploration algorithm Hector Exploration for planning the exploration. To measure inside a known environment the following is recommended: the Adaptive Monte Carlo Localization (AMCL) positioning algorithm and the Spanning Tree Covering algorithm.

ACKNOWLEDGEMENT

We would first like to extend our sincerest gratitude to our Company's CEO - Mr. Satyanarayana at Leap Robots LLP. Their extensive knowledge and experience have been an immense help in ensuring the success of this thesis. We are grateful for the multiple in-depth discussions which have inspired multiple ideas, and we would like to express our thanks for the encouragement and critique of this thesis.

We would also like to extend our gratitude to everyone at Leap Robots for allowing us to conduct our master's thesis at their company and for providing a workplace.

Huge thanks to Ms. N Siva Mouni (Ph.D. IIT Hyd) for helping us understand existing indoor solution, her many in-depth reviews, and for listening to us ramble on without actually having any crucial role in this thesis. This thesis builds upon the excellent work of the Robot Operating System community, which have our heartfelt thanks for making this thesis possible. Last, we would like to thank our friends and families, whom have supported us during both this thesis, and the duration of our studies.

We could not have done it without your support!

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 – INTRODUCTION TO ORGANIZATION

1.1 About Leap Robots LLP

Leap Robots strives to be a duty bounded and engaged community member through collaborating with local non-profits to recognize the needs of those in our communities. The Leap Gives program exists to promote community engagement among Leap team members and goes to the extra mile on the side of excellence and making the world a better place by volunteering with local non-profits and non-profit clients.

Fig. 1.1

As a non-profit, we strive to provide robotics to everyone. Partnering up with Leap Robots, we will teach a course using coding and BOOYAKA DIY mobile Robots to solve everyday problems. Not only will this help you learn about robotics, but also improve your problem-solving skills. In this course, you will learn to code and understand the basics of Arduino.

4.4 The foundation of company's commitment

Trusting in empowering robotics technology with Research & Development is what we focus on. We believe the quality of education is the right thing to create future opportunities for the students. We expertise in design & development of Robots, training courses.

- **Creative Leadership:** We inspire growth and innovation through learning and bold action.
- **Passion for Progress:** We have a vision to impact and excellence in the education sector.
- **Responsibility:** We shine by honesty, integrity, and thoughtfulness.

1.3 Mission and Vision

Our mission is to change the way practical education and automate the future domains.

Vision to develop robotics tools and technology for practical education through stem learning and future automation.

"Education is not the learning of facts, but the training of the mind to think. "

We are young technology driven designers, developers, and marketers in the new generation educational robots. Leap Robots leaps technology in robotics by setting Robotics trend in Education with STEM Learning. We offer STEM-based practical educational solutions which include programming Building blocks with training curriculum. Leap robots provide better solutions in Indian education and bring a new era of Robotics in Education.

4.4 Research and Development

A better robot lives in the world by traveling in its environment more intelligently and effectively, through cooperating with the people. The serving of robot is compelling, and physical interaction made a way to step into the next level of future innovation.

Leap Labs is your very own, After School Activity Program, Technology Revolution in Robotics and Industrial Automation, tailor-made for you. 'It is an advanced, state of the art platform to explore, learn and build robots and related technologies.

We provide Systematic, innovative and interactive training by experts consistent with the STEM practical education and the industry requirements.

Along with educational AI products, leap robots developing customized AI solutions for industries.

CHAPTER 2 – SOFTWARE TRAINING WORK UNDERTAKEN

2.1 Introduction to Raspberry Pi 4

Raspberry Pi is a small single board computer. By connecting peripherals like Keyboard, mouse, display to the Raspberry Pi, it will act as a mini personal computer. It is popularly used for real time Image/Video Processing, IoT based applications and Robotics applications.

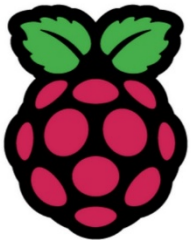


Fig. 2.1.1

Raspberry Pi is slower than laptop or desktop but is still a computer which can provide all the expected features or abilities, at a low power consumption. Raspberry Pi Foundation officially provides Debian based Raspbian OS. Also, they provide NOOBS OS for Raspberry Pi. We can install several Third-Party versions of OS like Ubuntu, Arch Linux, RISC OS, Windows 10 IOT Core, etc.

Raspbian OS is official Operating System available for free to use. This OS is efficiently optimized to use with Raspberry Pi. Raspbian have GUI which includes tools for Browsing, Python programming, office, games, etc.

Raspberry Pi is more than computer as it provides access to the on-chip hardware i.e. GPIOs for developing an application. By accessing GPIO, we can connect devices like LED, motors, sensors, etc and can control them too.

Raspberry Pi also provides on-chip SPI, I2C, I2S and UART modules.
Raspberry Pi 3 On-chip Hardware

The On-chip hardware of Raspberry Pi 4 is as shown in below figure,

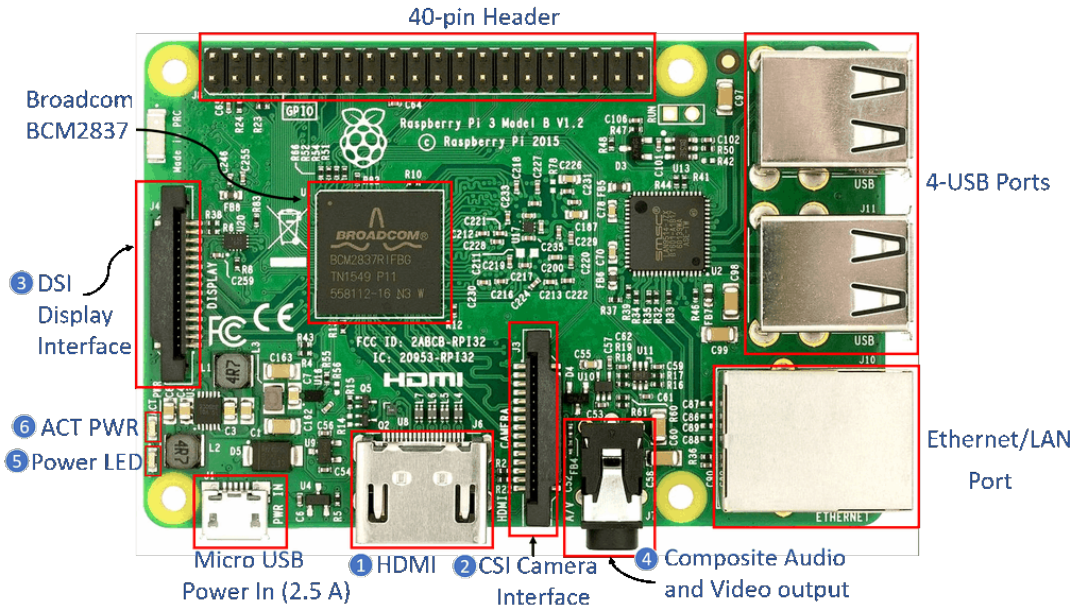


Fig. 2.1.2

Some Hardware Components shown above are mention below:

1. **HDMI (High-Definition Multimedia Interface):** It is used for transmitting uncompressed video or digital audio data to the Computer Monitor, Digital TV, etc. Generally, this HDMI port helps to connect Raspberry Pi to the Digital television.
2. **CSI Camera Interface:** CSI (Camera Serial Interface) interface provides a connection in between Broadcom Processor and Pi camera. This interface provides electrical connections between two devices.
3. **DSI Display Interface:** DSI (Display Serial Interface) Display Interface is used for connecting LCD to the Raspberry Pi using 15-pin ribbon cable. DSI provides fast High-resolution display interface specifically used for sending video data directly from GPU to the LCD display.
4. **Composite Video and Audio Output:** The composite Video and Audio output port carries video along with audio signal to the Audio/Video systems.
5. **Power LED:** It is a RED coloured LED which is used for Power indication. This LED will turn ON when Power is connected to the Raspberry Pi.
6. **ACT PWR:** ACT PWR is Green LED which shows the SD card activity.

2.2 Introduction to ROS – Robotic Operating System

2.2.1 What is ROS

ROS (Robot Operating System) is a BSD-licensed system for controlling robotic components from a PC. A ROS system is comprised of a number of independent nodes, each of which communicates with the other nodes using a publish/subscribe messaging model. For example, a particular sensor's driver might be implemented as a node, which publishes sensor data in a stream of messages. These messages could be consumed by any number of other nodes, including filters, loggers, and also higher-level systems such as guidance, pathfinding, etc.

2.2.2 Why ROS?

Note that nodes in ROS do not have to be on the same system (multiple computers) or even of the same architecture! You could have a Arduino publishing messages, a laptop subscribing to them, and an Android phone driving motors. This makes ROS really flexible and adaptable to the needs of the user. ROS is also open source, maintained by many people.

2.2.3 General Concepts

Let's look at the ROS system from a very high-level view. ROS starts with the ROS Master. The Master allows all other ROS pieces of software (Nodes) to find and talk to each other. That way, we do not have to ever specifically state "Send this sensor data to that computer at 127.0.0.1. We can simply tell Node 1 to send messages to Node 2.

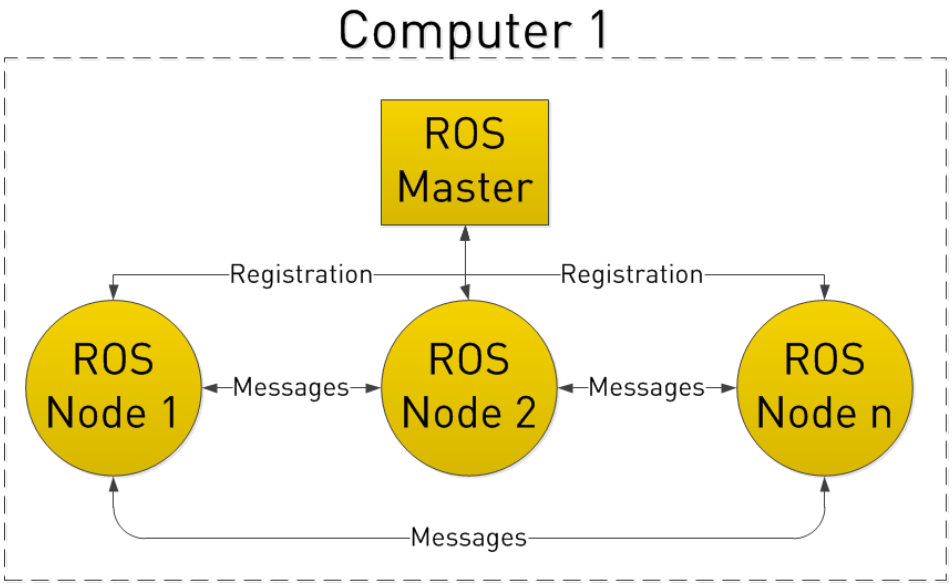


Fig. 2.2.1

How do Nodes do this? By publishing and subscribing to Topics.

Let's say we have a camera on our robot. We want to be able to see the images from the camera, both on the Robot itself, and on another laptop.

In our example, we have a Camera Node that takes care of communication with the camera, an Image Processing Node on the robot that process image data, and an Image Display Node that displays images on a screen. To start with, all Nodes have registered with the Master. Think of the Master as a lookup table where all the nodes go to find where exactly to send messages.

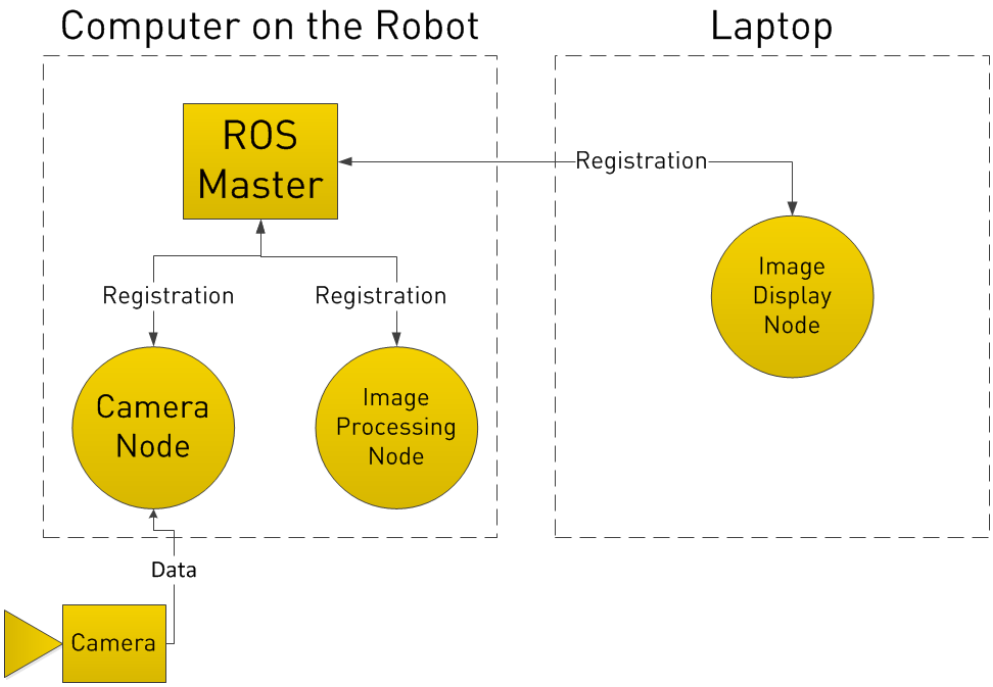


Fig. 2.2.2

In registering with the ROS Master, the Camera Node states that it will Publish a Topic called /image_data (for example). Both of the other Nodes register that they are subscribed to the Topic /image_data.

Thus, once the Camera Node receives some data from the Camera, it sends the /image_data message directly to the other two nodes. (Through what is essentially TCP/IP)

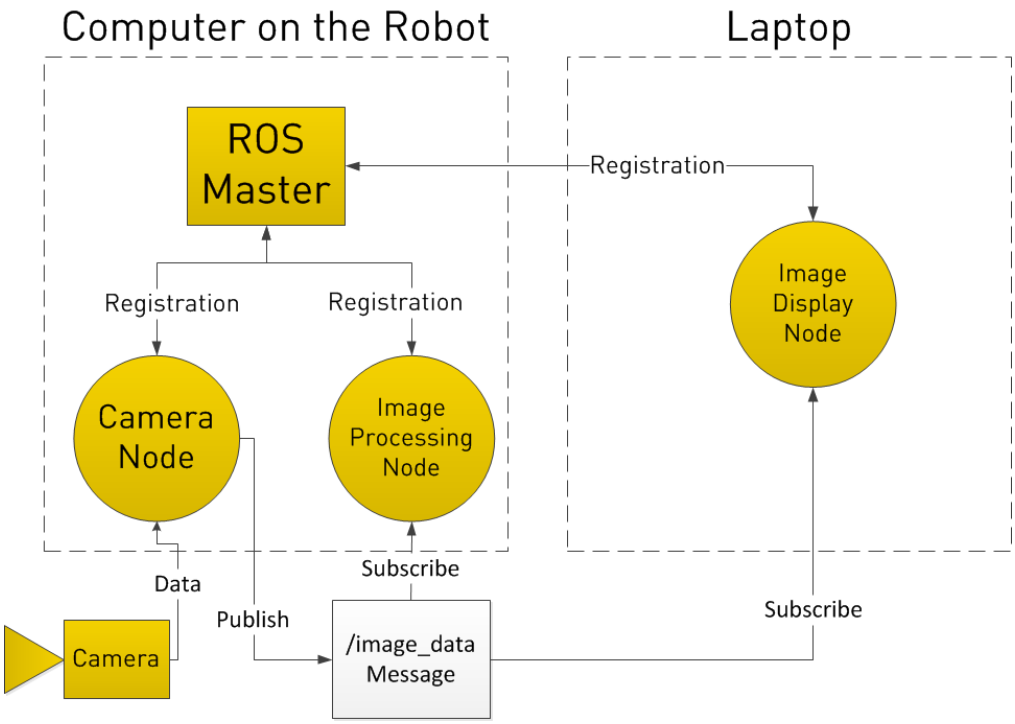


Fig. 2.2.3

Now you may be thinking, what if I want the Image Processing Node to request data from the Camera Node at a specific time? To do this, ROS implements Services.

A Node can register a specific service with the ROS Master, just as it registers its messages. In the below example, the Image Processing Node first requests /image_data, the Camera Node gathers data from the Camera, and then sends the reply.

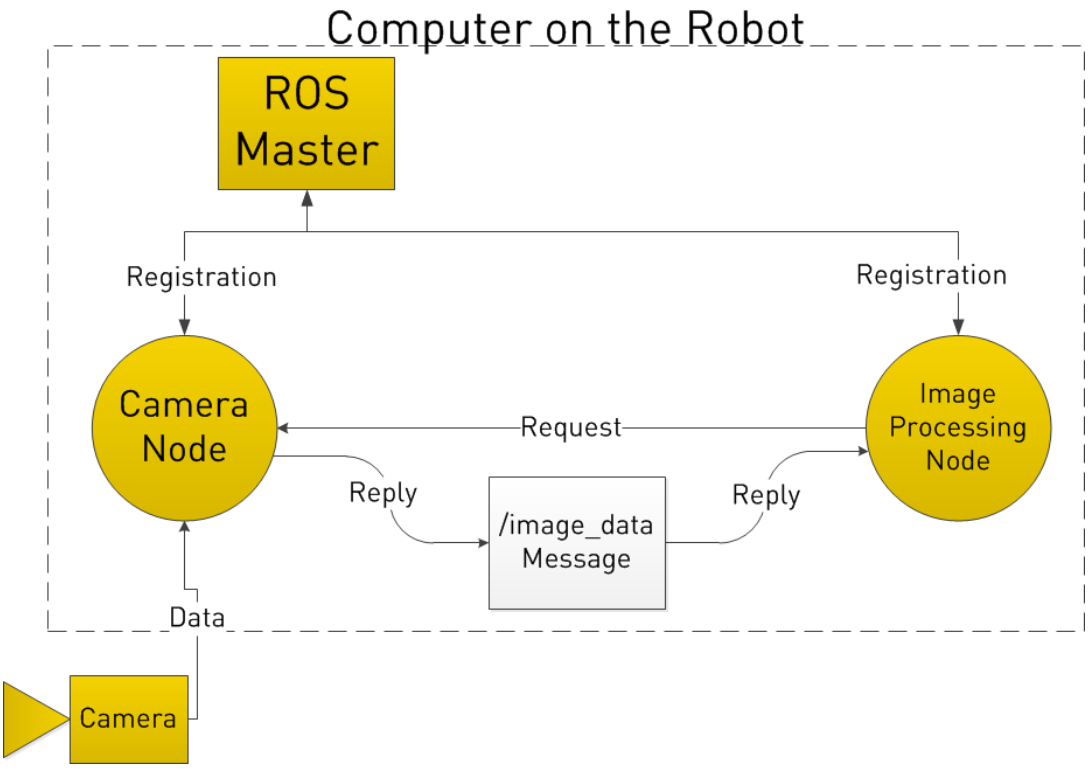


Fig. 2.2.4

2.2.4 ROS and underlying operating system

ROS depends on the underlying Operating System. ROS demands a lot of functionality from the operating system. On top of that ROS must be freely available to a large population, otherwise, a large population may not be able to access it. Much of the popularity of ROS is due to its open nature and easy availability to the mass population. It also needs an operating system that is open source so the operating system and ROS can be modified as per the requirements of application.

There is close proximity between ROS and OS, so much so that it becomes almost necessary to

know more about the operating system in order to work with ROS. Using Linux as a newbie can be a challenge, one is bound to run in issues with Linux especially when working with ROS, and a good knowledge of Linux will be helpful to avert/fix these issues.

- A **Meta Operating system** has a huge amount of functionality, so much that it cannot be classified as a framework or a cluster of libraries but not so much that it can be categorized as an operating system either. It provides functionalities of both Operating Systems as well as frameworks but not fully hence, it cannot be classified as either e.g, it does not provide the core functionalities that an operating system is supposed to provide but provides APIs.
- **Rviz** is a 3D Visualization tool for ROS. It is one of the most popular tools for visualization. It takes in a topic as input and visualizes that based on the message type being published. It lets us see the environment from the perspective of the robot.

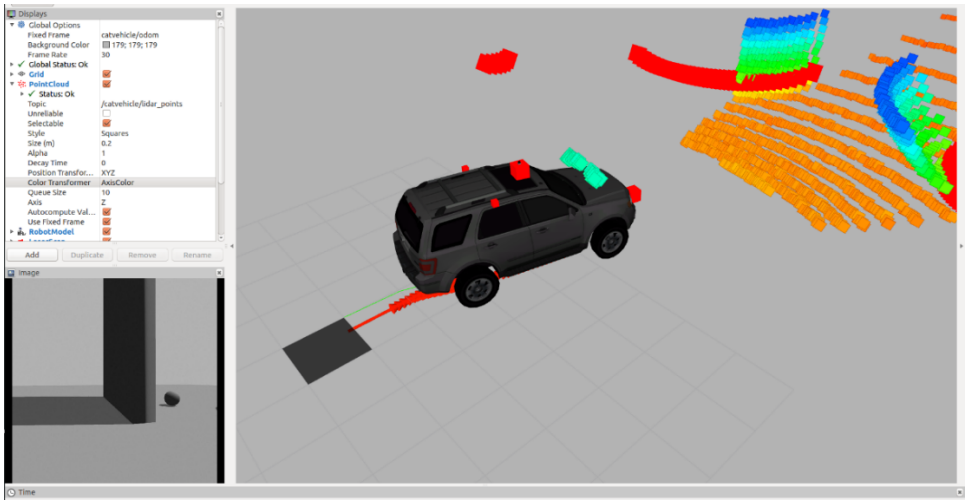


Fig. 2.2.5

Once we have all the code ready and running, we need to test our code so that we can make changes if necessary. Doing this on a real robot will be costly and may lead to a wastage of time in setting up robot every time. Hence, we use robotic simulations for that. The most popular simulator to work with ROS is **Gazebo**. It has good community support; it is open source and it is easier to deploy robots on it.

CHAPTER 3 INDUSTRIAL TRAINING WORK UNDERTAKEN

3.1 Installation of ROS on Ubuntu

Setup your computer to accept software from packages.ros.org.

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

Setup your Keys

```
sudo apt install curl # if you haven't already installed curl

curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
```

First, make sure your Debian package index is up-to-date

```
sudo apt update
```

Desktop-Full Install: (Recommended): Everything in **Desktop** plus 2D/3D simulators and 2D/3D perception packages

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

Environment setup – You must source this script in every **bash** terminal you use ROS in.

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

It can be convenient to automatically source this script every time a new shell is launched. These commands will do that for you.

```
Echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

Up to now you have installed what you need to run the core ROS packages. To create and manage your own ROS workspaces, there are various tools and requirements that are distributed separately. For example, rosinstall is a frequently used command-line tool that enables you to

easily download many source trees for ROS packages with one command.

To install this tool and other dependencies for building ROS packages, run:

```
sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool
build-essential
```

Before you can use many ROS tools, you will need to initialize rosdep. Rosdep enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS. If you have not yet installed rosdep, do so as follows.

```
Sudo apt install python3-rosdep
```

With the following, you can initialize rosdep.

```
Sudo rosdep init
rosdep update
```

3.2 Interfacing Raspberry Pi with ROS through server

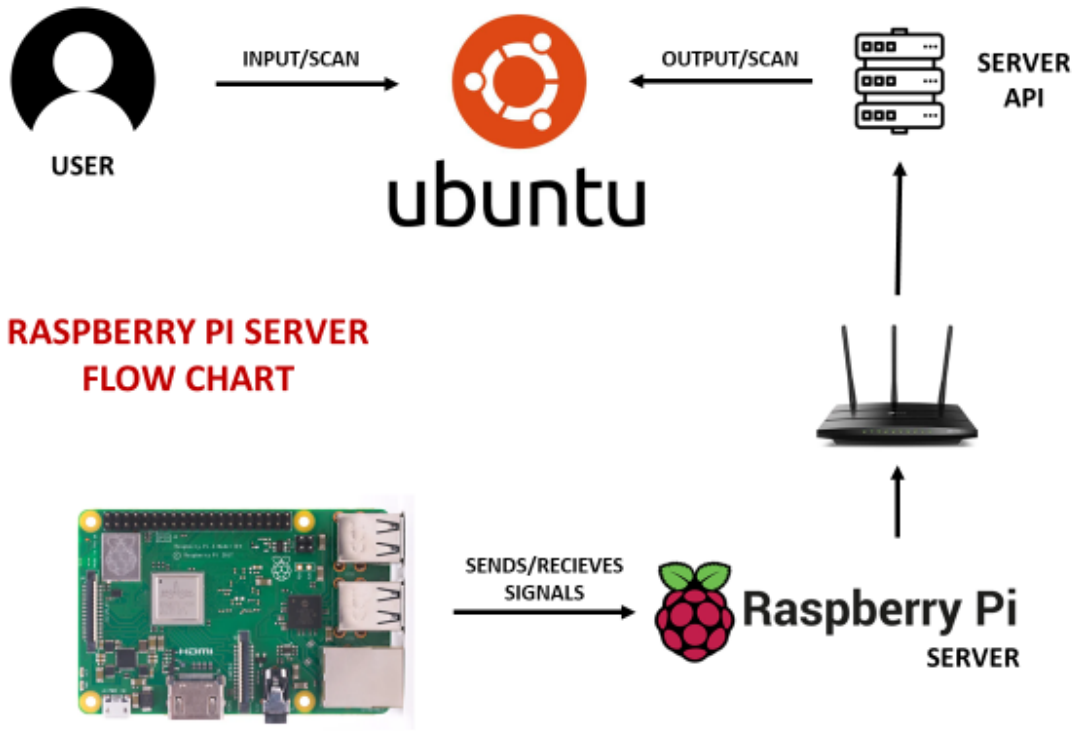


Fig. 3.2

You'll need the following things to install Raspberry pi server

- A micro-SD card (32 GB or greater recommended)
- A computer (running Linux, Windows or macOS) with a micro-SD card reader
- A Raspberry Pi 2, 3 or 4
- Good internet connection
- An HDMI cable for the Pi 2 & 3 and a micro-HDMI cable for the Pi 4 (optional)
- A USB keyboard set (optional)

3.2.1 Installing Raspberry Server OS

Make sure you have inserted the microSD card into your computer, and install the Raspberry Pi Imager at your computer.

Despite I use Ubuntu, I won't use the Debian package that is listed above, but I will install the snap package using the command line. This method can be applied to wider range of Linux distributions.

Sudo snap install rpi-imager

Once you have installed Raspberry Pi Imager tool, find and open it and click on the "CHOOSE OS" menu.

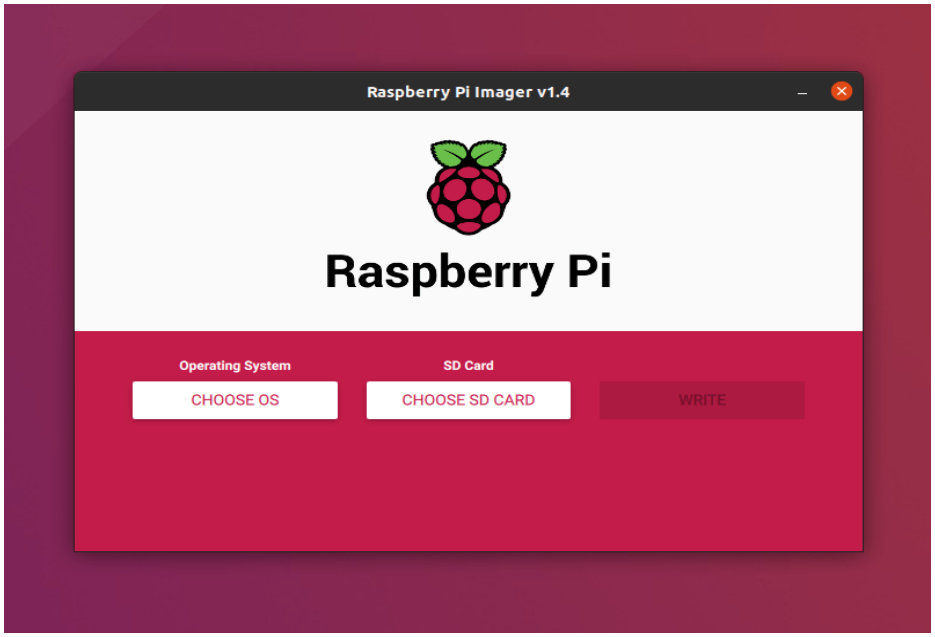


Fig. 3.2.1

Scroll across the menu and click on “Ubuntu” (Core and Server Images).

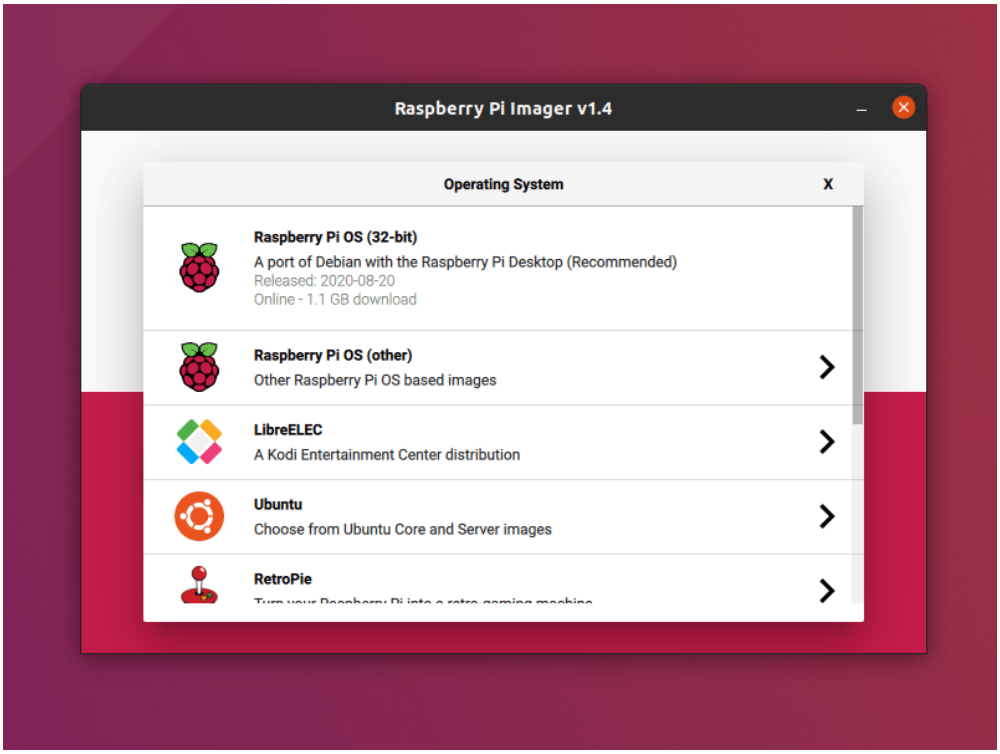


Fig. 3.2.2

From the available images, I choose the Ubuntu 20.04 LTS 64 bit. If you have a Raspberry Pi 2, you are limited to the 32bit image.

Important Note: If you use the latest Raspberry Pi 4 – 8 GB RAM model, you should choose the 64bit OS, otherwise you will be able to use 4 GB RAM only.

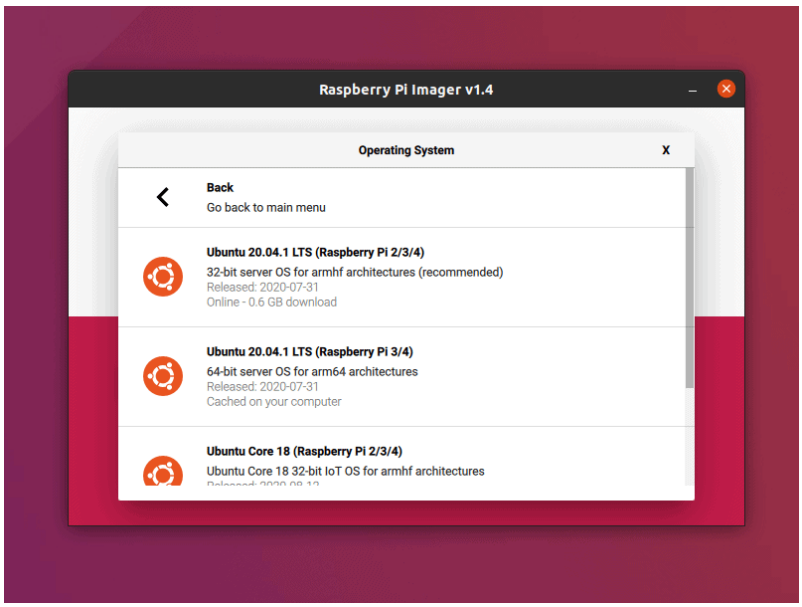


Fig. 3.2.3

Select your microSD card from the “SD Card” menu, and click on “WRITE “after.

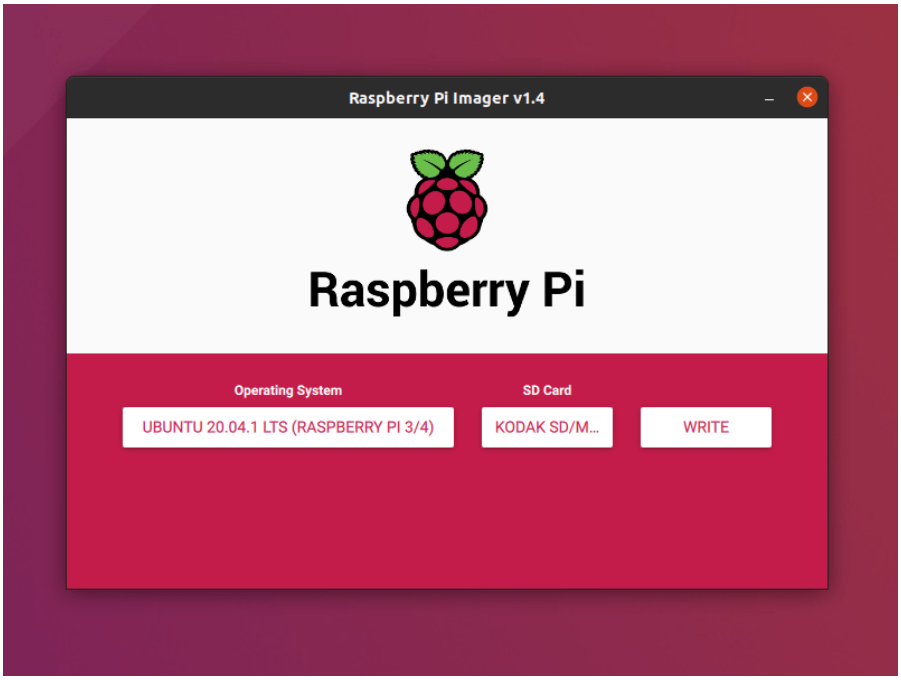


Fig. 3.2.4

If it shows some error, try writing it again. It will now download the Ubuntu server image and write it to the micro-SD card.

It will notify you when the process is completed.

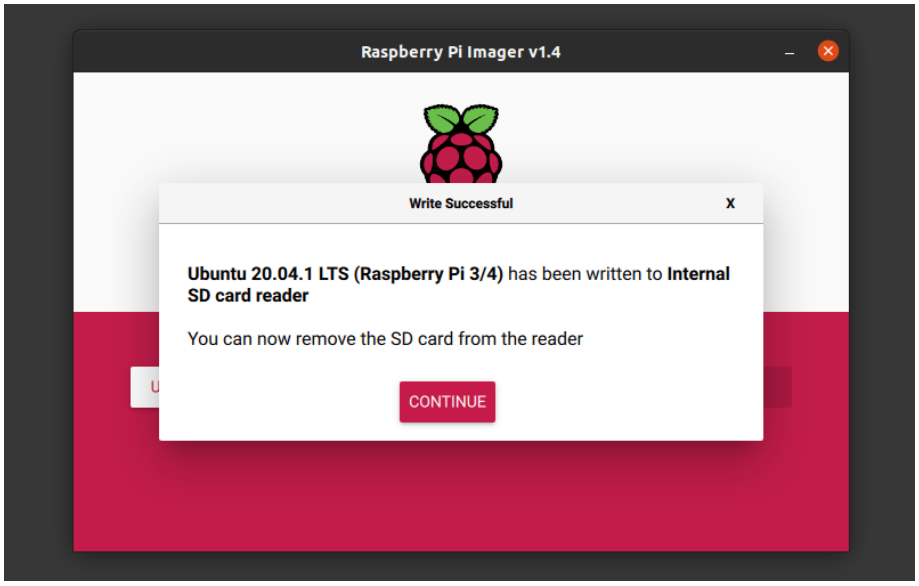


Fig. 3.2.5

3.2.2 Add WI-FI support to Ubuntu server

Once the micro-SD card flashing is done, you are almost ready to use it. There is one thing that you may want to do before using it and that is to add Wi-Fi support. If you don't do it right now, you'll have to put extra effort later in connecting to Wi-Fi from terminal in Ubuntu server.

With the SD card still inserted in the card reader, open the file manager and locate the "system-boot" partition on the card.

The file that you are looking for and need to edit is named **network-config**.

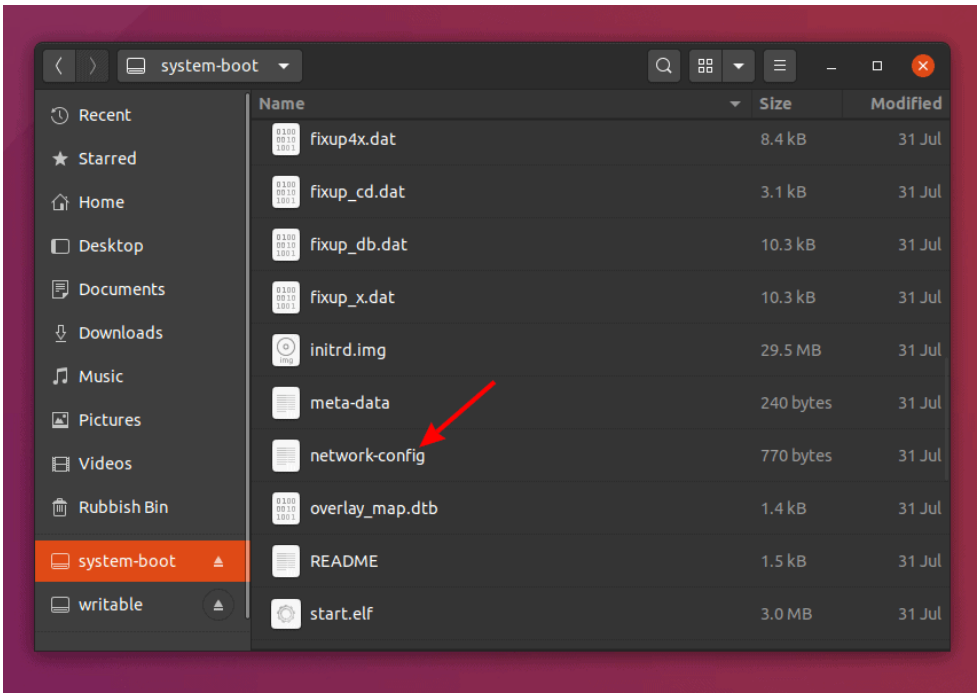


Fig. 3.2.6

This process can be done on Windows and MacOS too. Edit the **network-config** file as already mentioned to add your Wi-Fi credentials.

Firstly, uncomment (remove the hashtag "#" at the beginning) from lines that are included in the rectangular box.

After that, replace myhomewifi with your Wi-Fi network SSID name enclosed in quotation marks, such as "Predator" and the "S3kr1t" with the Wi-Fi password enclosed in quotation marks, such as "12345679".

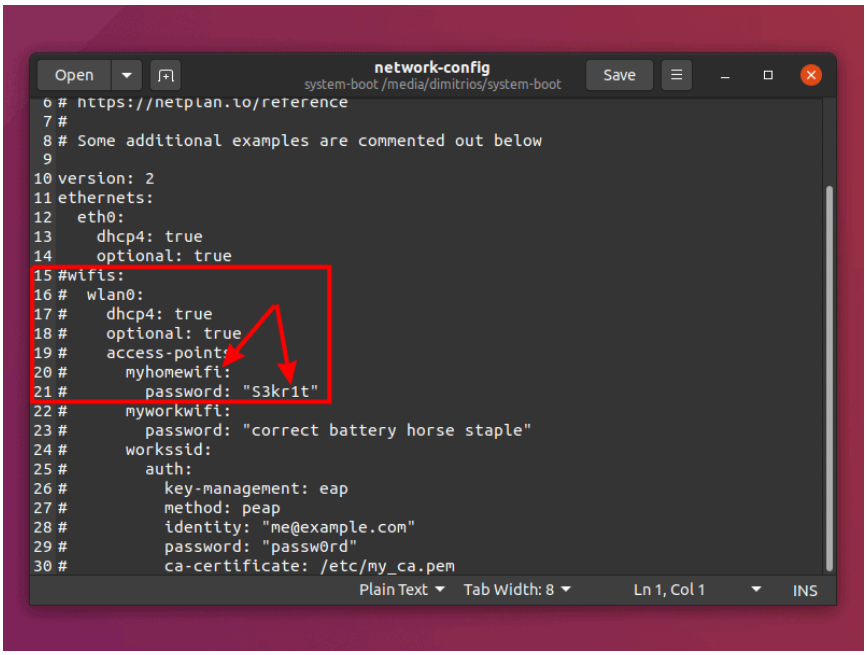


Fig. 3.2.7

It may look like this:

wifis: wlan0: dhcp4: true optional: true access-points: “your wifi name”: password: “your_wifi_password”

Save the file and insert the micro-SD card into your Raspberry Pi. During the first boot, if your Raspberry Pi fails connect to the Wi-Fi network, simply reboot your device.

3.2.3 Connect Raspberry Pi to Ubuntu server

On Ubuntu and Mac OS, an SSH client is usually already installed. To connect remotely to your Raspberry Pi, you need to discover its IP address. Check the devices connected to your network and see which one is the Raspberry Pi.

Open a terminal and run the following command:

ssh ubuntu@raspberry_pi_ip_address

You will be asked to confirm the connection with the message:

Are you sure you want to continue connecting (yes/no/[fingerprint])?

Type “yes” and click the enter key.

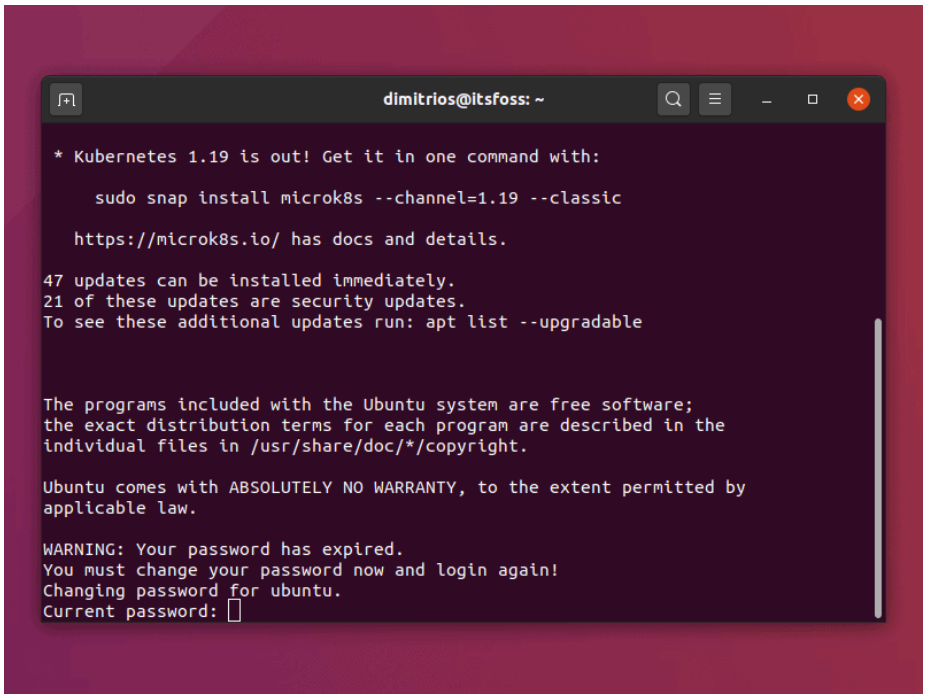


Fig. 3.2.8

When prompted, use “ubuntu” for the password as mentioned earlier. You’ll be asked to change the password of course.

Once done, you will be automatically logged out and you have to reconnect, using your new password.

Your Ubuntu server is up and running on a Raspberry Pi!

Conclusion

Installing Ubuntu Server on a Raspberry Pi is an easy process and it comes pre-configured at a great degree which the use a pleasant experience.

3.3 Interfacing Camera with Raspberry Pi

For our experiment, we had chosen the following models: tiny YOLO and SSD Mobile Net lite.

You only look once (YOLO) is a state-of-the-art, real-time object detection system implemented on Darknet. Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales. High scoring regions of the image are considered detections. This model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by a global context in the

image. It also makes predictions with a single network evaluation unlike systems like R-CNN which require thousands for a single image.

To carry out the detection, the image is divided in a SxS grid (left image). Each of the cells will predict N possible “bounding boxes” and the level of certainty (or probability) of each one of them (image at the centre), this means SxSxN boxes are calculated. The vast majority of these boxes will have a very low probability, that’s why the algorithm proceeds to delete the boxes that are below a certain threshold of minimum probability. The remaining boxes are passed through a “non-max suppression” that will eliminate possible duplicate detections and thus only leave the most precise of them (image on the right).

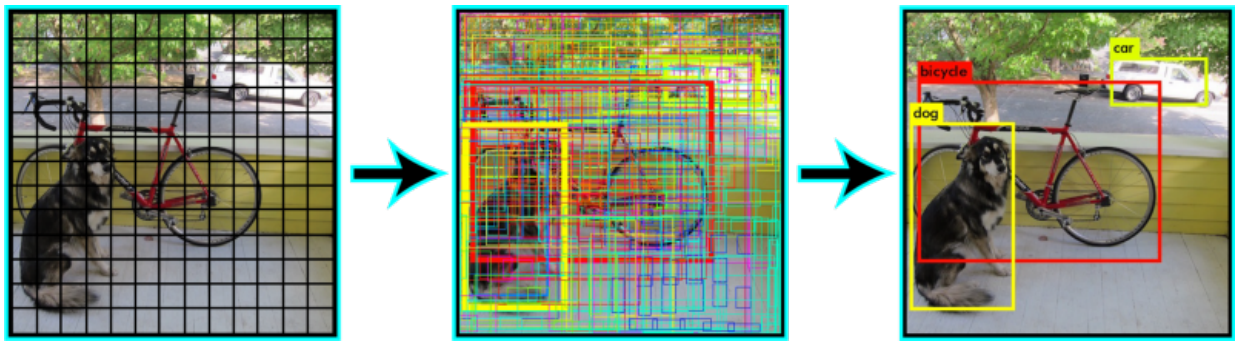
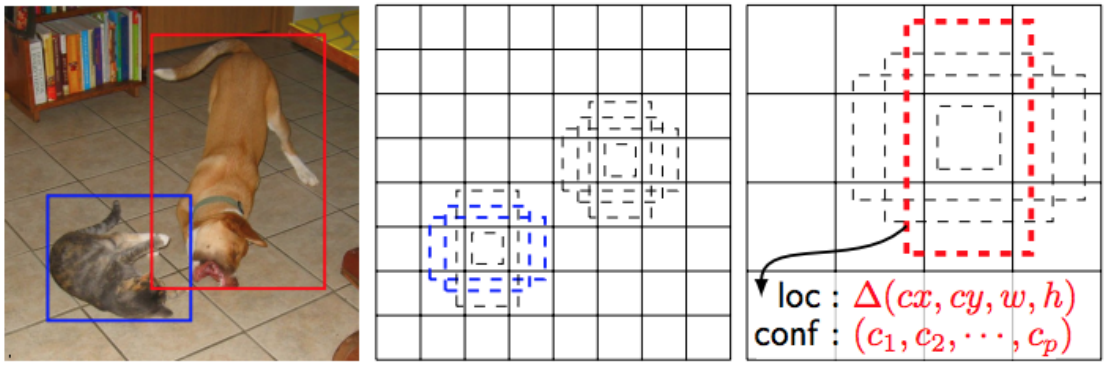


Fig. 3.3

SSD (Single Shot MultiBox Detector) is a popular algorithm in object detection. SSD speeds up the process by eliminating the need for a region proposal network. To recover the drop in accuracy, SSD applies a few improvements including multi-scale features and default boxes. These improvements allow SSD to match the Faster R-CNN’s accuracy using lower resolution images, which further improves the speed. SSD with Mobile Net is, an object detection model optimized for inference on mobile devices.

The key idea here is a single network (for speed) and no need for region proposals. Instead, it uses different bounding boxes and then adjusts the bounding box as part of the prediction. Different bounding box predictions are achieved by each of the last few layers of the network responsible for predictions for progressively smaller bounding box and final prediction is a union of all these predictions.



(a) Image with GT boxes (b) 8 × 8 feature map (c) 4 × 4 feature map

Fig. 3.4

For object detection, we used OpenCV, TensorFlow Object Detection API and Dark flow. TensorFlow's Object Detection API is a very powerful tool that can quickly enable anyone (especially those with no real machine learning background) to build and deploy powerful image recognition software. The API provides to end users' instruments for training and running detection models and models trained on COCO dataset like Faster R-CNN, SSD Mobile and etc. Since YOLO is implemented on a C++ based framework for deep learning called Darknet we use a translation of Darknet to TensorFlow called Dark flow.

3.3.1 Environmental Setup

Next, we'll set up the environment. First thing is installing OpenCV on a Raspberry Pi 3 with all the dependencies.

Next step is to set up and enable the camera. We also need to install a python module called picamera [array]. The module provides an interface to represent images from the camera as NumPy arrays.

As mentioned above, TF 1.9+ officially supports Raspberry Pi. But it doesn't mean that we can just install it using pip. Pip installs only the 0.11.0 version, that does not satisfy the requirements for the object detection API. We should build TensorFlow from source code. But building on the Raspberry Pi is not recommended, because of the slow processor and limited RAM. It takes a lot of time. It's easier to build a TensorFlow.whl package on a more powerful host machine and then

install it on the Raspberry.

We can use official guide to build the package or download an already built package. After that, copy the wheel file to the Raspberry Pi and install it with pip:

```
pip install [file_name].whl
```

Next, we need to set up the environment for YOLO. YOLO is implemented in a C based framework for deep learning called Darknet. To avoid building Darknet on a Raspberry Pi we used Darkflow, it is Darknet translation to run over TensorFlow. Darkflow easy to install with instruction from the official repository. Also, we used YOLO's pretrained weights *tiny-yolo-voc.weights* from authors of Darkflow (https://drive.google.com/drive/folders/0B1tW_VtY7onidEwyQ2FtQVplWEU) and network config *tiny-yolo-voc.cfg* from the Darkflow source repository.

Now, we need to download the MobileNet SSD Lite model from the TensorFlow detection model zoo. The model zoo is Google's collection of pre-trained object detection models that have various levels of processing speed and accuracy. The Raspberry Pi has a weak processor and limited RAM, so we need to use a model that takes less processing power. Though the model runs faster, it comes at a trade-off of having lower accuracy. We tried to use SSD MobileNet model but during loading model graph it throws a memory allocation exception, Raspberry Pi doesn't have the required amount of memory for this task. Then, download the SSD Lite-MobileNet model and unpack it. We need the only *frozen_inference_graph.pb* file.

COCO-trained models

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes

Fig. 3.5

First, define an abstract class for detectors:

```
from abc import ABC, abstractmethod
class ObjectDetector(ABC):
        @abstractmethod
        def detect (self, frame, threshold=0.0):
                Pass
```

Next, implement the interface for SSD and YOLO models. For SSD we used code from object detection API. And YOLO object detector is just wrapped around Darkflow TFNet class.

Now let's implement a detection script. Firstly, initialize the camera:

```
# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 32
rawCapture = PiRGBArray(camera, size=(640, 480))
# allow the camera to warm up
time.sleep(0.1)
```

Next, take images from the stream and make detection on it. And visualize results:

```
for frame in camera.capture_continuous(rawCapture, format="bgr",use_video_port=True):
# grab the raw NumPy array representing the image, then
# initialize the timestamp and occupied/unoccupied text
image = frame.array

result = predictor.detect(image)

for obj in result:
        logger.info('coordinates: {} {}. Class: "{}". Confidence: {:.2f}'.
                format(obj[0], obj[1], obj[3], obj[2]))

        cv2.rectangle(image, obj[0], obj[1], (0, 255, 0), 2)
        cv2.putText(image, '{}: {:.2f}'.format(obj[3], obj[2]),
                        (obj[0][0], obj[0][1] – 5),
                        cv2.FONT_HERSHEY_PLAIN, 1, (0, 255, 0), 2)

# show the frame
cv2.imshow("Stream", image)
key = cv2.waitKey(1) & 0xFF
```

We ran both models and got these results:

- YOLO tiny — **0.32 FPS**;
- SSD MobileNet Light — **1.07 FPS**.

So, looking at these results we can state that we can use a mobile detector with SSD MobileNet for real-life simple pedestrian tracking or for house security systems, that can detect cats on your lawn. Or another case where sizes and power usage are more critical, and 1 FPS is sufficient.

Also, we tried to use MXNet network, but we have troubles with *../libmxnet.so* on Raspberry Pi during module importing.

Below results visualization:

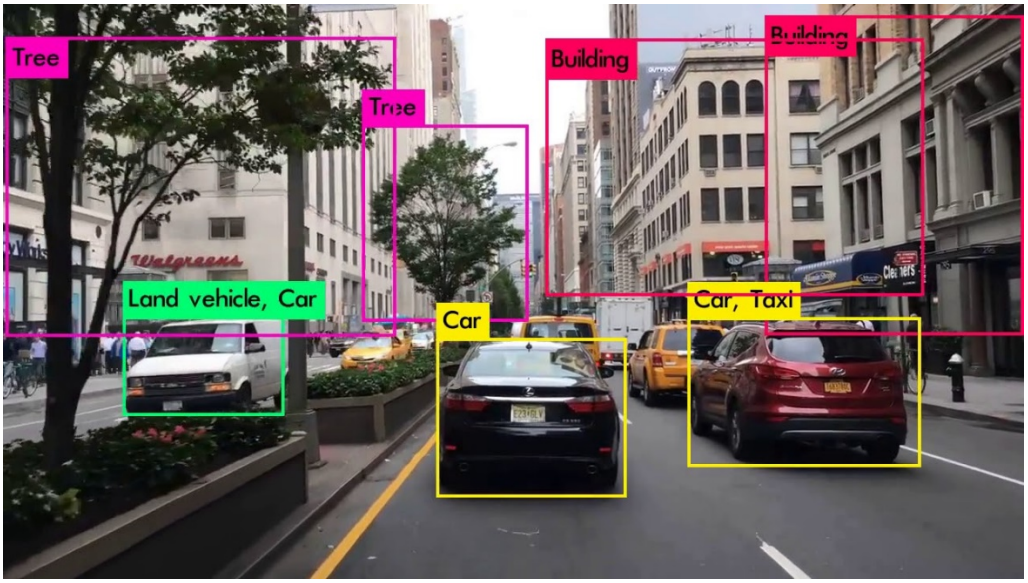


Fig. 3.6.1

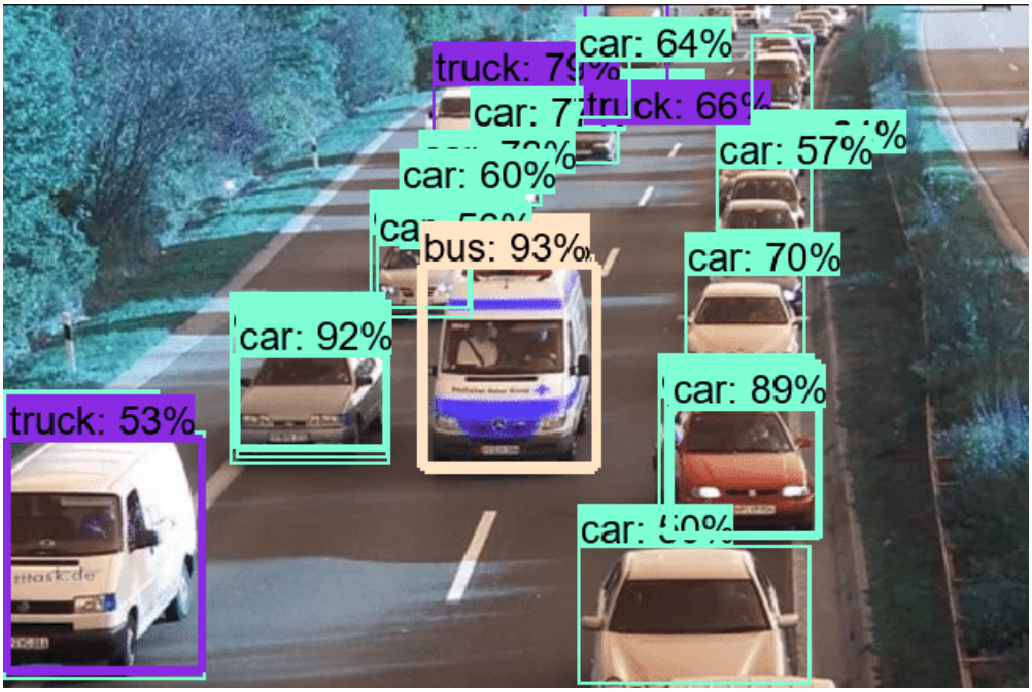


Fig. 3.6.2

CHAPTER 4 PROJECT WORK

4.1 Introduction

Simultaneous Localisation and Mapping (SLAM) algorithms are typically used when a robot is tasked with exploring an unknown area. SLAM allows the robot to map the area, while simultaneously determining its location within the resulting map. In order to estimate its position, the algorithm is provided with data from a variety of sensors. The type of sensor depends on which type of SLAM algorithm is being used. For example, vision-based SLAM requires visual data from a camera sensor, and, in certain cases, odometry data from an odometry sensor. If a range-based SLAM algorithm is utilised, then a ranging sensor and an odometry sensor would be required (this can be visualised in Figure). In this research project, we focus on range-based SLAM algorithms.

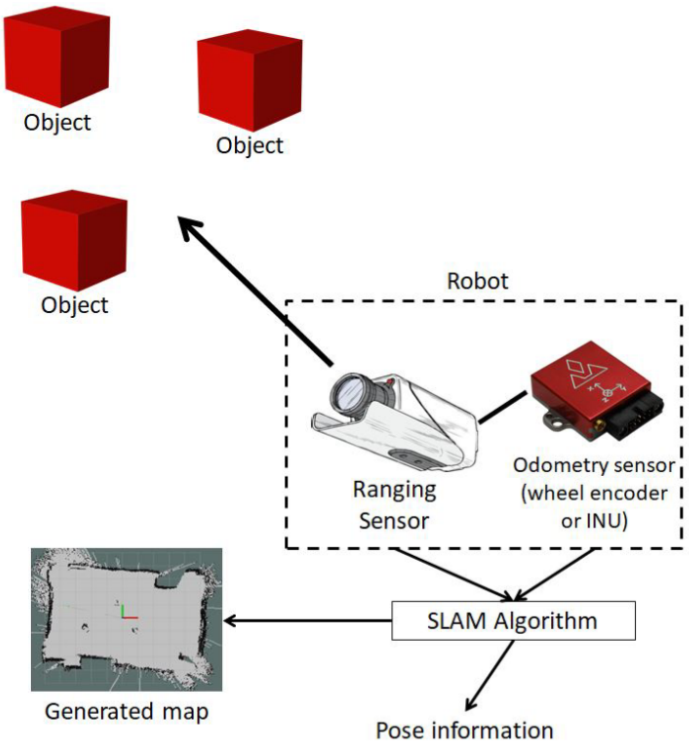


Fig. 4.1.1

As stated above, range-based SLAM algorithms require data from a ranging sensor and odometry source. The SLAM algorithm utilizes the data from each sensor in different steps of the SLAM process. The odometry data provides information that will tell the algorithm how the robot has

moved over a short time period. Estimating the position based purely on odometry data (called dead reckoning) becomes inaccurate over longer time periods, so the algorithm utilizes the ranging data to help correct the location of the robot. At the same time, the algorithm will utilize the ranging data to generate a map of the area. Accurate odometry data is highly desired, but cannot always be easily obtained, particularly in the case of airborne robots.

As SLAM algorithms have been developed over time, many different techniques have been discovered. One of the key components of most SLAM algorithm is called scan matching. Scan matching is the process of matching range data "snapshots" taken at different points in time, to each other, in order to estimate the displacement of the robot between scans. This is explained in Section 2.1. Some SLAM algorithms rely purely on scan matching, whereas others incorporate more complex techniques such as pose-graph optimization, and particle-filter-based estimation. While the outputs of a SLAM algorithm include both the map and the robot pose, this thesis is primarily concerned with the assessment of the maps.

Different types of ranging sensors will be tested with various SLAM algorithms to determine the best combinations and the quality of the resulting maps. The majority of the ranging sensors will be light-based Time-of-Flight (ToF) based sensors, however, some structured light and stereo vision based ranging sensors will also be incorporated. Ultrasonic sensors, while cheap and lightweight, are not typically accurate enough to use with SLAM, and thus are not investigated here. Some of the ToF sensors will be "scene-capturing" sensors (i.e., depth cameras), while others are rotating-laser sensors. To make a fair comparison, data from the scene-capturing sensors will be converted to rotating laser data, this is explained in Section 3.4.1. One specific new type of scene-capturing sensor that will be included in this comparison will be the Single Photon Avalanche Diode (SPAD) sensor. The SPAD sensor has the ability to capture data at a faster rate than existing scene-capturing ToF sensors. It also has the ability to capture data in low light and is a solid-state sensor. Determining the best, or optimum, combination of sensor and SLAM algorithm is a critical issue as it can save time in future projects by having already

determined optimum combinations.

4.2 Simultaneous Localization and Mapping Algorithms

The literature is rich with various types of SLAM algorithms that use different techniques to enable a robot to simultaneously localize and map an unknown environment. Most of these algorithms spawned from the first known SLAM algorithm presented in 1986 (Durant-Whyte & Bailey, 2006), which involved utilizing a probabilistic method to attempt map building. It was at this time that the problem of SLAM became a fundamental issue that needed to be solved. The simultaneous localization and mapping problem can be defined as follows:

"The simultaneous localization and mapping (SLAM) problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map." (Durant-Whyte & Bailey, 2006)

Over time, SLAM algorithms have evolved to include critical features such as scan matching algorithms and loop closure methods. Despite the many varying techniques, one iteration of a SLAM algorithm can, in general, be broken down into the following steps (assuming a laser rangefinder and an INU are utilized in this instance):

1. The robot moves in the unknown environment; the SLAM algorithm obtains the odometry data from the INU sensor. This data shows which direction it has moved and at what speed.
2. Utilizing the most recent odometry data, as well as the previous pose estimate, the SLAM algorithm will predict the new pose of the robot within the environment.
3. New ranging data is received by the SLAM algorithm.
4. The SLAM algorithm compares the new ranging data to the existing map and previous ranging scans, and updates the pose estimate in a way that maximizes compatibility of the current ranging data with existing information.
5. After the pose of the robot has been updated, the SLAM algorithm will then update the map of the environment.

4.2.1 Scan Matching

Scan matching is a common feature found in SLAM algorithms. As its name suggests, scan matching matches a scan from a rangefinder to either the complete existing map or to previous scans, by solving the rotation and translation of the robot necessary to make the new scan most compatible with previous data.

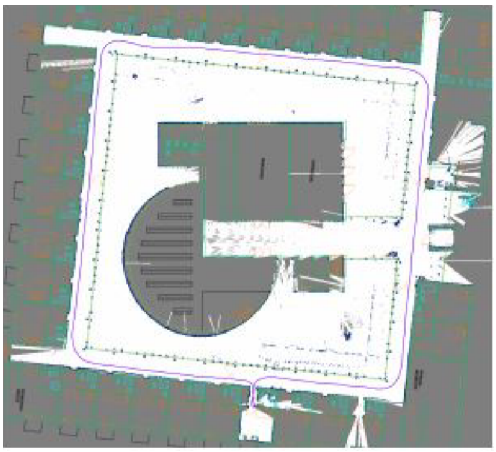


Fig. 4.1.2

This can either be done using the scans in their entirety, or by first extracting features such as corners and lines from each scan, and matching them. Scan matching does not require odometry data, but odometry data can be used to provide an initial guess of the unknown rotation and translation for an iterative scan matching algorithm. Scan matching will fail if there is insufficient information in the scans that are being matched. One example of a popular SLAM algorithm that is based purely on scan matching is Hector Mapping. Hector Mapping is widely available through the Robot Operating System and has been documented by the community. Diosi and Kleeman (2005) classified scan matching algorithms into three different categories, feature-to-feature scan matching, point-to-feature scan matching, and point-to-point scan matching.

Feature-to-feature scan matching is as stated above. The algorithm finds features (e.g., corners, straight line areas, etc.) and attempts to match them between scans. Often, feature-to-feature scan matching is commonly used with sensors with a high Field of View (FoV) (or Field of Regard (FoR)) as the high FoR provides the algorithm with more chances of capturing features and thus

give a higher chance of features being matched together.

Point-to-feature scan matching is based on matching points in the laser scan to features. The features, which can be straight line segments, corners, etc., can be obtained from a pre-existing map or Gaussian distributions. Gaussian distributions can provide features only if the mean and variance can be calculated from the scan points that are placed into grid cells.

There are various types of point-to-point matching, however, the most popular type is Iterative Dual Correspondence (IDC). IDC is a point-to-point matching algorithm that combines two other types of point-to-point matching algorithms, Iterative Closest Point (ICP) and Iterative Matching Range Point (IMRP). Point-to-point algorithms may often require multiple iterations before a match can be made between scans.

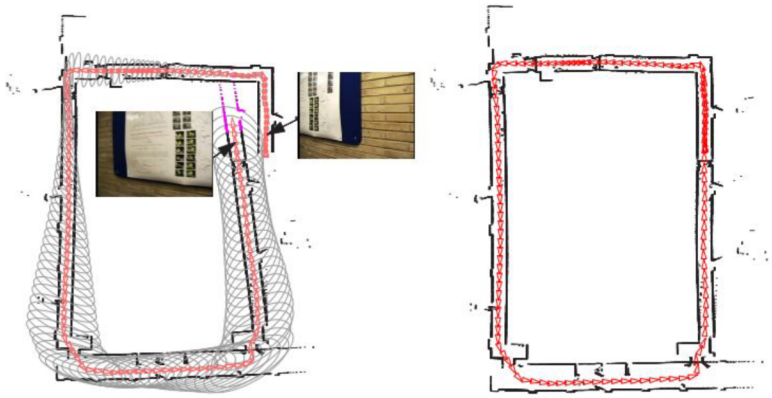


Fig. 4.1.3

4.2.2 Loop Closure

Loop closure is defined as the ability of the robot to identify if it has returned to locations that it had previously visited (Granstrom et al., 2009). Loop closure detection is seen as an important part of the SLAM algorithm as it enables the correction of accumulated drift, which greatly improves the quality of the map and pose estimates. Loop closure can be thought of as a scan matching method. The main difference between loop closure and scan matching is that loop closure observes the entire map while scan matching matches a current laser scan with the previous laser scan(s). Loop closure observes the entire map and matches areas that have been previously visited with incoming scans. By introducing loop closure, the error that was

accumulated from scan matching can be removed from the map.

Particle filter algorithms can be implemented with loop closure methods to reduce any error that is generated from the poses. Often, particle filters use a resampling method to perform loop closures. Fast SLAM is an example of how resampling determines loop closure and can be seen in Figure below,

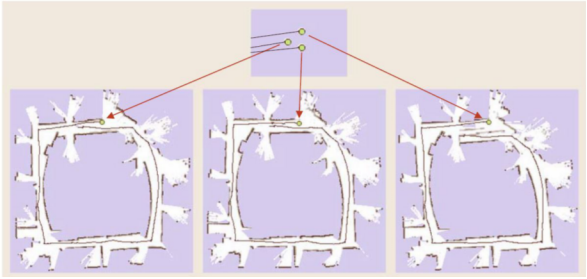


Fig. 4.1.4

it can be seen that there are three different particles, each with their own map and paths. The resampling method chooses the particle which has a map that are the close representations of the measurements that were taken. In this case, either the left or middle particles would be chosen.

4.3 Ranging Sensors

4.3.1 Time-of-Flight

The dToF technique measures the time difference between when a pulse of light is sent out and when its reflection is received. This technique requires precise timing in order to calculate the exact distance of an object because the speed of light is so fast (Bellisai et al., 2013, Piatti et al., 2013). Pulsed light is when the illuminator for the ToF sensor sends one pulse at a time. Pulsed light can be used for both indirect and direct Time-of-Flight, however, it is fairly uncommon in indirect ToF systems. For a dToF system, distance can be calculated easily with Equation 2.1 (Piatti et al., 2013). Equation 2.1 shows the simplicity of a direct time of flight system as there is only real one unknown variable.

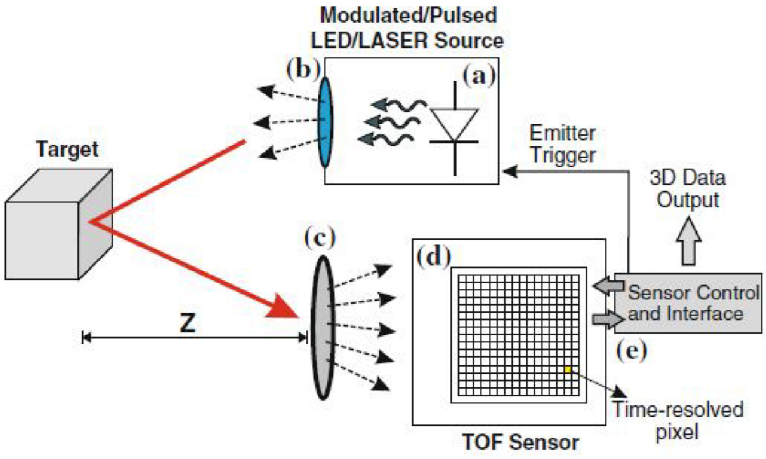


Fig. 4.2.1

$$Z = c/2 * \tau TOF$$

Where:

- z = distance
- c = speed of light
- τTOF = total time from when the light was emitted and received again

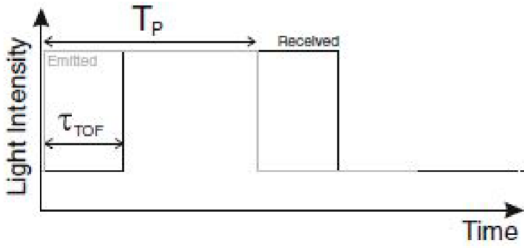


Fig. 4.2.2

$$z = c/4\pi fm * \varphi TOF$$

Where:

- c = speed of light
- fm = frequency of modulation
- φTOF = total time from when the light was emitted and received again

$$z = c/4\pi fm * \arctan(C3 - C1/C4 - C2)$$

Where:

- c = speed of light

- fm = frequency of modulation

4.4 Rotating laser vs Scene capturing ToF sensors

Time-of-Flight sensors can be constructed in various forms. The most common is a single rotating laser and photodiode, and a camera-like array of photodiodes together with a wider-angle transmitter (either laser or LED). We use the terms “rotating laser”, and “scene-capturing sensor” for these two forms respectively. The difference between the two is explained further below.

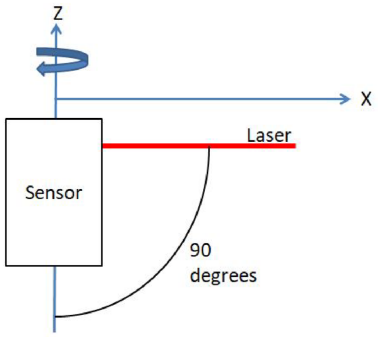
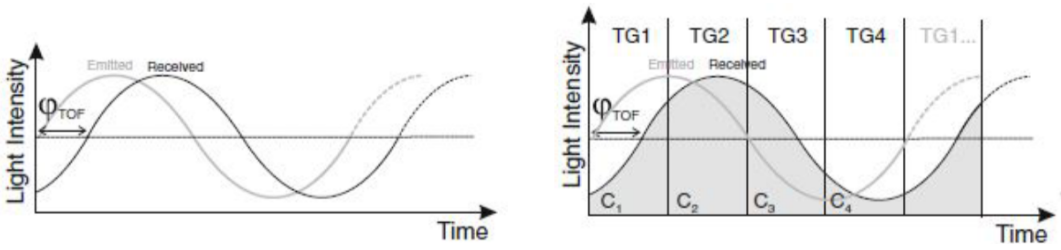


Fig. 4.2.3

4.4.1 Rotating-laser ToF sensors

Rotating-laser sensors operate by taking a slice out of the environment to take distance data, as seen in Figure 2.9. Often, the majority of rotating laser sensors tilize only one laser, however, there are some exceptions where multiple lasers are rotated to take multiple slices out of the environment from different angles. Regardless of how many lasers are rotated, the rotating lasers operate off the ToF principle.



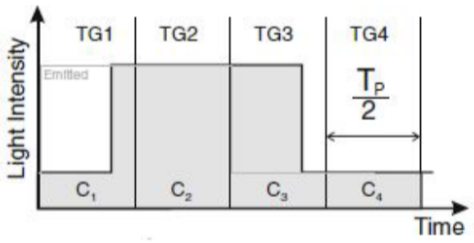


Fig. 4.2.4

A rotating laser ToF sensor can be described as a sensor that rotates on a single axis and sends out a laser that is perpendicular to the rotation axis. This laser is the ToF illuminator that is needed by the sensor to calculate the distance. The laser can be reflected to the perpendicular angle by various different methods. The most common method is to use a mirror to deflect the laser. The mirror would rotate along the z-axis, as shown in Figure 2.9 and reflect the laser out. Another simple method is to rotate the laser instead of the mirror. The total degree that the laser rotates around the Z axis is often referred to as the Field of Regard (FoR).

4.4.2 Scene-capturing ToF sensors

The scene-capturing ToF sensor operates differently to a rotating laser ToF sensor, in that it operates like a camera. The figure shows how a scene capturing ToF sensor acquires depth data for an array of pixels at once, effectively generating a “depth image” of the scene. Unlike the single rotating laser sensors, a scene-capturing sensor does not need moving parts and is thus referred to as a solid-state sensor. In the figure, the sensor is using the x-axis for the distance data, while the Z and Y axis is for the array of pixels. While a single rotating laser sensor can take up to 360 degrees slice of the environment, a scene capturing sensor will only capture the scene that is inside its Field of View (FoV).

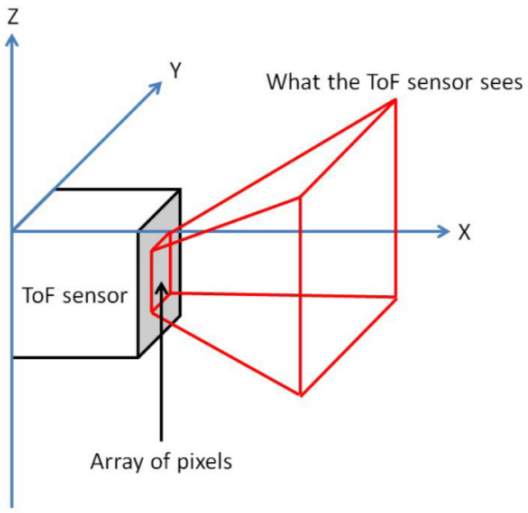


Fig. 4.2.5

4.5 RPLIDAR ROS

RPLIDAR is a low-cost LIDAR sensor suitable for indoor robotic SLAM application. It provides 360-degree scan field, 5.5hz/10hz rotating frequency with guaranteed 8-meter ranger distance, current more than 16m for A2 and 25m for A3. By means of the high-speed image processing engine designed by RoboPeak, the whole cost is reduced greatly, RPLIDAR is the ideal sensor in cost sensitive areas like robot's consumer and hardware hobbyist.

RPLIDAR A3 performs high speed distance measurement with more than 16K samples per second, RPLIDAR A2 performs high speed distance measurement with more than 4K/8K samples per second, RPLIDAR A1 supports 2K/4K samples per second. For a scan requires 360 samples per rotation, the 10hz scanning frequency can be achieved. Users can customize the scanning frequency from 2hz to 10hz freely by control the speed of the scanning motor. RPLIDAR will self-adapt the current scanning speed.

Fig. 4.3.1

Features

- Measures distance data in more than 8000 times/s
- 360 Degree Omnidirectional Laser Range Scanning
- 8000 times sample rate, the highest in the current economical LIDAR industry
- OPTMAG Original Design, prolong the life-span
- Configurable Scan Rate from 2-10Hz
- Ideal for Robot Navigation and Localization

Technical details

Dimensions	98.5mm x70mm x60mm
Weight	G.W 170g
Battery	Exclude
Distance Range	0.15 - 6m, White objects
Angular Range	0-360 Degree
Distance Resolution	<0.5mm
Angular Resolution	≤1Degree
Sample Duration	0.5ms

Sample Frequency	2000~2010Hz
Scan Rate	1~10Hz, Typical 5.5Hz,

Table 4.1

4.5.1 Procedure to run RPLIDAR

Open your ubuntu desktop, and open terminal

The first step is giving the USB port the authority of reading and writing by following command

Sudo chmod 666 /dev/ttyUSB0

it will ask for the admin's password, type your password and hit Enter

After this, we need to download and clone the RPLIDAR package in our Catkin Workspace

cd catkin_ws/src

git clone https://github.com/robopeak/rplidar_ros.git

Then after, we need to build the libraries into catkin workspace in order to make them executable

cd..

catkin_make

there after we need to source our bash file in order to locate the packages for the ROS nodes for spanning it into Catkin workspace

source devel/setup.bash

Till here we cloned the repository for a1m8 RPLIDAR and converted it into an package which is accessible by the ROS from catkin workspace,

Now to start 3d mapping, we launch ROS core first

roscore

Then after, we launch the RPLIDAR executable file in different terminal

Roslaunch rplidar_ros view_rplidar.launch

After this command the launch file starts the nodes and connects it to Rviz for simulation and 3d

mapping as shown below

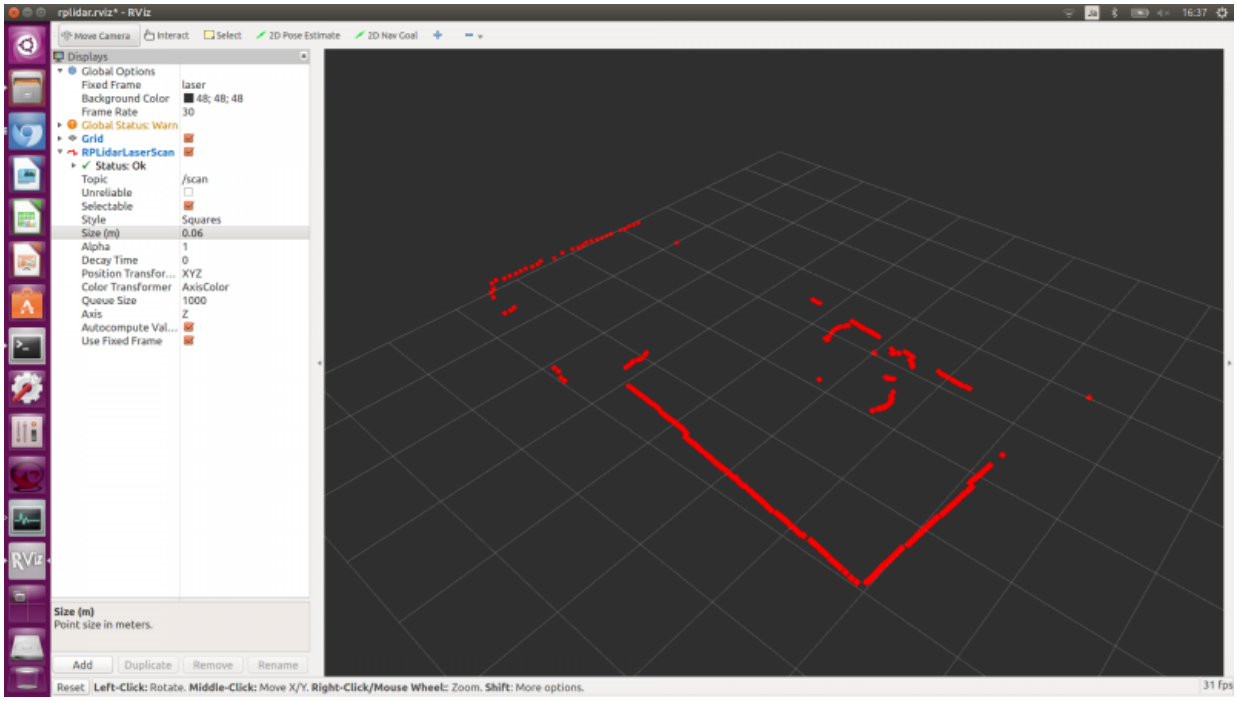


Fig. 4.4

4.5.2 Hector Slam

Hector_slam uses the hector_mapping node for learning a map of the environment and simultaneously estimating the platform's 2D pose at laser scanner frame rate. The frame names and options for hector_mapping have to be set correctly. This tutorial explains the different options. The image below shows all potential frames of interest in a simplified 2D view of a robot travelling through rough terrain, leading to roll and pitch motion of the platform:

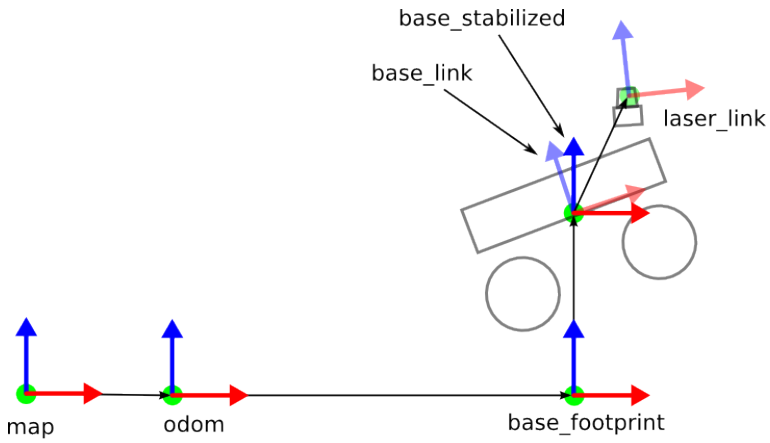


Fig. 4.5.1

Use in place of gmapping

G-mapping publishes the map->odom transform. You have to set the following parameters of hector_mapping to reflect the setup on your platform:

```
<param name="pub_map_odom_transform" value="true"/>
  <param name="map_frame" value="map" />
  <param name="base_frame" value="base_frame" />
  <param name="odom_frame" value="odom" />
```

Use without odom frame

If you do not require the use of an odom frame (for example because your platform does not provide any usable odometry) you can directly publish a transformation from map to base_link:

```
 <param name="pub_map_odom_transform" value="true"/>
  <param name="map_frame" value="map" />
  <param name="base_frame" value="base_frame" />
  <param name="odom_frame" value="base_frame" />
```

Use without Broadcasting of transformations

The system can also be used without broadcasting any tf transformation. This can be useful if another node is responsible for publishing the map->odom or map->base_frame transform. This can be done by setting:

```
<param name="pub_map_odom_transform" value="false"/>
```

4.5.2.1 Running Hector Slam Package

Firstly, we should clone the repository inside the catkin workspace from GitHub by implementing following command

git_clone https://github.com/tu-darmstadt-ros-pkg/hector_slam.git

after, we should run roscore by following command

roscore

Then in catkin workspace we should run the following command to initiate communication between LIDAR and Ubuntu to get mapping values using synchronization,

roslaunch rplidar_ros rplidar.launch

Then after we will launch Rviz by bridging a communication between synchronized data with Rviz,

roslaunch hector_slam_launch tutorial.launch

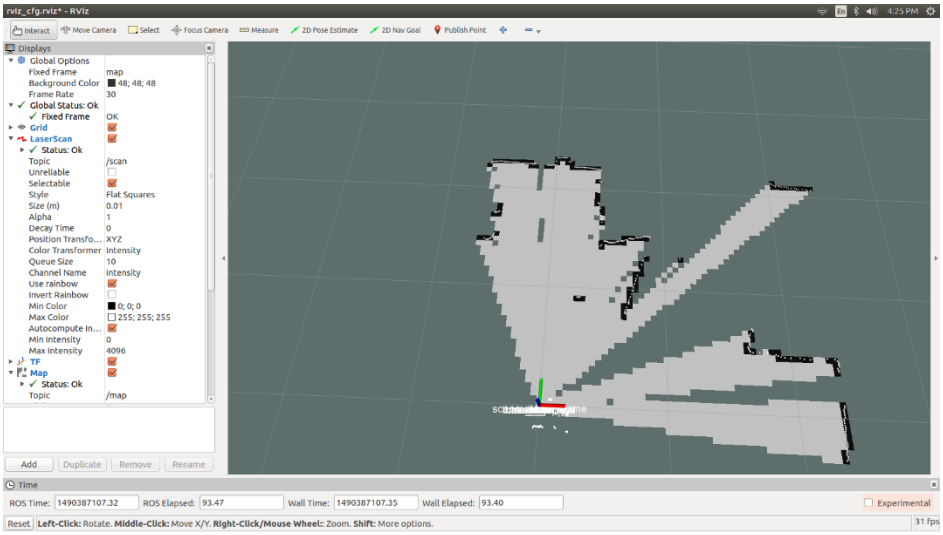


Fig. 4.5.2

CHAPTER 5 RESULT AND DISCUSSION

5.1 Result Obtained

By executing the Hector_Slam algorithm we obtained the 3D map of the current room residing.

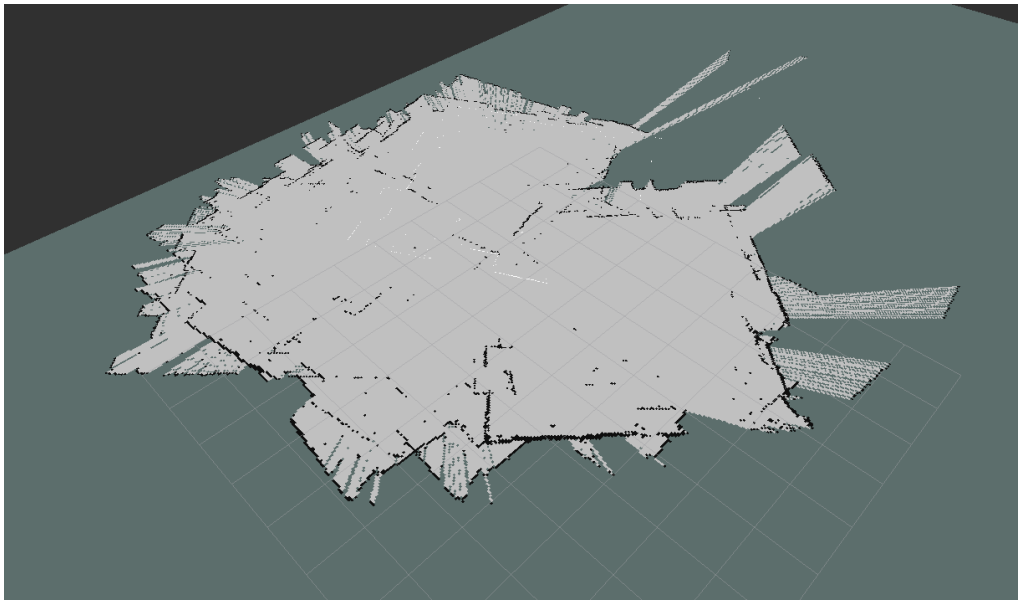


Fig. 5.1

Greater automation is not only evident in production and manufacturing environments, but also in everyday life. Service robots are already taking on many tasks in hospitals, cruise ships, hardware stores, airports, and hotels. Robots are booming in the industrial sector and now for private use.

In banks and other businesses, service robots can distribute flyers, entertain those waiting in lines or lobbies, and provide information to potential customers. Robots in the travel industry take on some of the tasks of the personnel, providing relief to human workers.

With the help of the 2D LiDAR, service and security robots detect every detail of their surroundings, dynamically adapting to changing conditions in order to interact with people and react to unknown situations – similar to a human.

In recent years, demand has also risen for a sensor that offers both field evaluation and the output of measurement data. The main feature of this sensor is the simultaneous use of the measurement values for localization tasks and the field evaluation for collision prevention.

CHAPTER 6 CONCLUSION AND FUTURE SCOPE

6.1 Conclusion

This thesis had the following research question to answer:

1. How do genuine real-world sensors work with real world SLAM algorithms?

    (a) How does nominally perfect odometry and odometry based off a real IMU influence the quality of map produced by sensor/SLAM algorithm combinations?

    (b) Are there optimum combinations of sensors and SLAM algorithm that produce the best quality maps?

Answer:

- Odometry error can influence the quality of map that is produced by various sensor and SLAM algorithm combinations. The odometry error caused maps to be generated with the following map errors: ghost maps, elongation, and excess noise.
- When the odometry was polluted with drift and noise, Hector Mapping and the RPLIDAR was found to be the optimum combination. Hector Mapping was the only SLAM algorithm to have significant map generation success. This is again due to the fact that wide FoR sensors could capture more features then a low FoV sensor. The more features there are present in a scan, the better scan matching algorithms can perform.

6.2 Future Scope

6.2.1 First Step

The main application of Hector Slam algorithm using RPLIDAR is 3d mapping a indoor environment and navigating a Robot.

Imagine a robot assisting an environment such as, in a university the robot is performing various tasks from greeting people to assisting them to reach to their desired locations, from cleaning floors regularly to transporting items from one place to another, from taking care of the labs to placing back the lab equipment after use to their desired positions. And this is possible by using a Camera and an LIDAR sensor – Light Detecting and Ranging in a ROS – Robotic Operating System Environment.

So, the project in this thesis is the first step for building an **Artificial Intelligence Robot.**

6.2.2 Second Step

After 3d mapping, the robot will remember its current position and the map will make robot easier to travel from one place to another.

So, the second stage of implementation includes –

- Mechanical build of robot
- Setting up micro-controller, LIDAR, Camera, Mic, Screen and basic equipment on to the robot
- Interfacing all input and output sensors/devices with micro controller
- Remote control of whole robot using ROS catkin workspace
- Travel path planning of robot from one place to another
- Addition and testing of Robotic hand to pick and place objects
- Smart AI development, to interact with people
- Addition of Safety and Surveillance
- Auto charging system