DEVELOPMENT OF AN EYE TRACKER AND WHEELCHAIR SYSTEM THAT CAN SELF NAVIGATE IN A CLOSED ENVIRONMENT

CHONG WEI SENG

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Mechanical Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

May 2020

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : Chong

Name : CHONG WEI SENG

ID No. : 1504337

Date : 16/05/2020

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"DEVELOPMENT OF AN EYE TRACKER AND WHEELCHAIR SYSTEM THAT CAN SELF NAVIGATE IN A CLOSED ENVIRONMENT"** was prepared by **CHONG WEI SENG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honour) Mechanical Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature :

Supervisor : IR. PROF. DATO' DR. GOH SING YAU

Date : 16/05/2020

Signature :

Co-Supervisor : IR DANNY NG WEE KIAT

Date : 16/05/2020

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2020, Chong Wei Seng. All right reserved.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, PROF. DATO' IR DR GOH SING YAU for his invaluable advice and gives his opinion based on his professional experience. Besides that, I also would like to thank my research co-supervisor IR DANNY NG WEE KIAT for his guidance and his enormous patience throughout the development of the research.

ABSTRACT

A wheelchair is used to support people with mobility difficulties due to injuries or motor disabilities. For patients with serious motor disabilities who might not be able to move their hands, a joystick wheelchair that needs to be operated by hand is not feasible. Therefore, an eye tracker controlled wheelchair that only needs the user's eye movement to control the wheelchair can be used by such patients. The aim of this project is to design and develop an eye tracker controlled wheelchair to help such patients. Two methods will be developed to control the wheelchair. The first method is to directly control the wheelchair by the instant eye movement of a user. The user can control the wheelchair to move in four directions: moving forward and backward, as well as turning left and right. The second method is developed for the user when the user needs to move from one location to another location. The user can use an eye tracker to select a location through a screen interface. The wheelchair will move to the destination with a navigation system while avoiding obstacles along the path. There are six locations provided for selection. For both methods, a node was developed for the eye tracker to retrieve the user's eye movement. Another node was developed to collect the information from the eye tracker and uses this information to control the simulated wheelchair according to the eye movement, or to send the request to carry out the navigation. For the navigation, the mapping of the environment was completed using RVIZ to get the coordinates of the locations. The simulated wheelchair will move to the destination based on the user's selected location while avoiding obstacles.

TABLE OF CONTENTS

DECLARATION			**ii**
APPROVAL FOR SUBMISSION			**iii**
ACKNOWLEDGEMENTS			**v**
ABSTRACT			**vi**
TABLE OF CONTENTS			**vii**
LIST OF FIGURES			**ix**
LIST OF ABBREVIATIONS			**xi**
LIST OF APPENDICES			**xii**
CHAPTER			
1	**INTRODUCTION**		**1**
	1.1	General Introduction	1
	1.2	Importance of the Study	2
	1.3	Problem Statement	2
	1.4	Aim and Objectives	3
	1.5	Scope and Limitation of the Study	3
	1.6	Contribution of the Study	3
	1.7	Outline of the Report	4
2	**LITERATURE REVIEW**		**5**
	2.1	Introduction	5
	2.2	Eye-tracking Methods	5
		2.2.1 Electrooculography	5
		2.2.2 Video-oculography	7
		2.2.3 Video-Based Combined Pupil/Corneal Reflection	9
	2.3	Eye Controlled Wheelchair Projects	10
	2.4	Introduction to Tobii	14
	2.5	Summary	15
3	**METHODOLOGY AND WORK PLAN**		**16**
	3.1	Introduction	16

	3.2	Hardware	16
		3.2.1 Eye Tracker	16
		3.2.2 Wheelchair	16
		3.2.3 Eye tracker mounting	17
	3.3	Software	17
		3.3.1 Ubuntu	17
		3.3.2 Tobii Core SDK	17
		3.3.3 Robot Operating System (ROS)	18
		3.3.4 Gazebo	18
		3.3.5 RVIZ	19
	3.4	Command-based Coding	21
	3.5	Move to goal/ Destination-based Coding	22
	3.6	Communication between Two Laptops	22
4	**RESULTS AND DISCUSSION**		**24**
	4.1	Assembly of Wheelchair and Eye Tracker	24
	4.2	Calibration Process	26
	4.3	Command-based Action	27
	4.4	Destination-based Action	30
5	**CONCLUSIONS AND RECOMMENDATIONS**		**35**
	5.1	Conclusions	35
	5.2	Recommendations for Future Works	35
REFERENCES			**37**
APPENDICES			**39**

LIST OF FIGURES

FIGURE	TITLE	PAGE
2.1	The Position of the Electrodes to Obtain EOG.	6
2.2	Features from High-resolution Eye Images.	9
2.3	Purkinje Reflections..	10
2.4	Relative Positions of the Pupil and First Purkinje Reflections.	10
2.5	Command Interface.	11
2.6	High-level Schematic of a System.	12
2.7	Outputs of the Image Processing Software.	13
2.8	Eye-tracker.	14
3.1	A Gazebo World.	19
3.2	SLAM Launch File in RVIZ.	20
3.3	Generated Map of Gazebo World.	20
3.4	Command-based Control Interface.	21
3.5	Destination-based Control Interface.	22
3.6	Setting of Windows Environment Variables.	23
3.7	Modify "/etc/hosts'' File to Resolve the Master's Name to IP.	23
4.1	(a) Wheelchair Base, and (b) Motor Driver with Micro-Controller.	24
4.2	Wheelchair with Tray.	25
4.3	Eye Tracker Mounting	26
4.4	Calibration Dots in Calibration Process	27
4.5	Exports ROS_MASTER_URI and Starts ROS Master.	28

4.6	Console Window in Windows OS Laptop.	28
4.7	Terminal of Ubuntu Running the "controller" Node.	29
4.8	ROS Nodes and ROS Topics in Command-based Action.	29
4.9	Display of the Selected Location.	30
4.10	Running of the "map_navigation_node".	30
4.11	Initial Position of the Simulated Wheelchair in RVIZ	31
4.12	The Simulated Wheelchair is Navigating to the Destination.	31
4.13	The Simulated Wheelchair Reached the Destination	31
4.14	ROS Nodes, Topics and Service in Destination-based Action.	34

LIST OF ABBREVIATIONS

BCI	Brain-Computer Interface
CRP	Corneal-Retinal Potential
DAQ	Data Acquisition
EEG	Electroencephalograhy
EGTS	Eye-gaze Tacking System
EOG	Electro-oculography
EPW	Electric Powered Wheelchair
GUI	Graphical User Interface
HCI	Human-Computer Interface
IROG	InfraRed Oculography
ROS	Robot Operating System
VOG	Video-oculography

LIST OF APPENDICES

APPENDIX A: Command-based Code in Window OS Laptop 39

APPENDIX B: Command-based Code in Ubuntu Laptop 41

APPENDIX C: Destination-based Code in Window OS Laptop 43

APPENDIX D: Destination-based Code in Ubuntu Laptop 45

CHAPTER 1

INTRODUCTION

1.1 General Introduction

A wheelchair is usually used by patients in a hospital who suffer moderate or severe injuries. The conventional wheelchair needs help from other people to move the patient around unless the patient is in good condition and is able to move his hands. The conventional wheelchair is only suitable for temporary use. For the patients who suffer permanent paralysis, they need to sit in the wheelchair for the rest of their lives. Hence, the conventional wheelchair is no longer suitable to use. Therefore, the electric wheelchair with joystick control is introduced. Besides that, patients also can have an autonomous wheelchair that is equipped with a navigation system and is able to avoid obstacles along the path. To consider different kinds of patients and different levels of severity of injuries, many types of control interfaces are introduced including voice control, tongue control, brain-computer interface, body-machine interface, eye control and electromyography-based interfaces (Mandel, Laue and Autexier, 2018). Among the control interfaces mentioned, eye control is the most suitable as it is mostly applicable to all kinds of the patient except for those patients who suffer eye diseases. Eye control can even be used by patients who suffer from motor neuron diseases and may not be able to move their hands and legs.

Eye movement is one of the fastest motions produced by the human body and it can quickly reflect the response of the human brain. Hence, the eye movement is one of the best inputs for the control interface. In order to measure eye movement, an eye tracker is needed. In fact, eye tracking is not a new technology and is widely used nowadays. According to iMotions (2015), eye-tracking applications are generally utilized in eight types of research including academic and scientific research, market research, psychological research, medical research, usability research, packing research, PC and gaming research and lastly human factors and simulation. For instance, in medical research, the eye tracker can be used and work together with conventional research methods or biosensors to diagnose diseases like Autism Spectrum Disorder (ASD), Parkinson's and Alzheimer's disease, Obsessive Compulsive Disorder (OCD)

and Attention Deficit Hyperactivity Disorder (ADHD). Besides that, eye tracking also helps in fields of neuroscience and psychology to understand how and why eye movements are made as well as how we gather information with our eyes.

1.2 Importance of the Study

This study may provide some insights into the application of the eye-tracking system. The eye-tracking system is widely used in different fields of study. However, there is still some room for improvement and further studies can be conducted to integrate it into our daily life. This study will show the application of the eye-tracking system on a human-computer interface. Besides that, the results of this study may further improve the autonomous wheelchair which will improve the quality of life of paralyzed patients.

1.3 Problem Statement

An eye-tracking assistive wheelchair has brought good news for the paralyzed patients as they can finally leave their bed and move around. The video-oculography method is a preferable eye-tracking method to work with the wheelchair compare to other methods as it is non-invasive and has better eye-movement detection (Furman and Wuyts, 2012).

A lot of similar projects were conducted with different types of eye trackers and also different eye-tracking methods. However, there are still some limitations. First, the users can make a selection of the command action on the interface. However, the time to activate the command action usually takes time. For example, the users have to gaze on certain commands action areas for at least a few seconds to activate the command. If the time taken to activate the command is too short, for instance, one second, sometimes it might be the wrong input as the users might just idle around on the screen. Hence, the optimum time taken and the method to make a selection can be further investigated. Besides that, when the users focus on the screen that is usually placed in front of them, they might ignore the circumstances around them, and the interface also only displays the scene in front of them. They cannot know what is behind them. Moreover, they might also overlook the obstacles on the ground.

1.4 Aim and Objectives

The main aim of this project is to design an eye tracker/autonomous wheelchair system for patients with severe motor disabilities. The wheelchair will bring the users to the desired destination with a navigation system (destination-based action) and the users also can directly control the wheelchair (command-based action). Therefore, they need to select a destination and also control the wheelchair via their eye movement with the aid of an eye tracker device installed on the wheelchair. The specific objectives are

1. To write programming codes for command based action and destination based action.
2. To design a suitable and appropriate layout of the interfaces for the users to input commands.
3. To install an eye tracker on the wheelchair with a mounting to hold it in position and minimize vibration when the wheelchair is moving.

1.5 Scope and Limitation of the Study

In this project, the application of the eye tracker on the electric-powered wheelchair is studied. Besides that, the eye-tracking technique to trace the eye movement is video-oculography (VOG). Table-mounted type of eye-tracking system is used as it is non-invasive and it is more convenient for our targeted users - half and fully-paralyzed individuals.

1.6 Contribution of the Study

By the end of this project, a user is able to control the wheelchair with two different methods, which are direct control and with the navigation system. It is beneficial to the target user as he or she can move around freely no matter it is just a short distance or a far destination. The communication between two laptops with different operating systems and programming languages used is successfully established.

1.7 Outline of the Report

For Chapter 2, there are some literature reviews on the eye-tracking methods and some eye-controlled wheelchair project. There is also a brief introduction to the Tobii company as its product is used in the project. The following chapter includes the hardware and software involved in this project. Besides that, the step to set up communication between the two laptops and also the two methods of controlling the wheelchair would also introduced. The result would be shown in Chapter 4. The conclusion and recommendations for the project are outlined in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

There are some eye-tracking methods which include electro-oculography (EOG), video-oculography (VOG), InfraRed Oculography (IROG), scleral search coil and video-based combined pupil/corneal reflection (Hari and Jaswinder, 2012). The usage of the methods depends on the requirements, constraints, the conditions and what is the application. Besides that, there are two types of systems which are table-mounted system and head-mounted system.

The eye-tracking system is also applied to an electric-powered wheelchair to help the patients that suffer neurological diseases that affect their movement in daily life. There are some projects on electric power wheelchair using different eye-tracking methods.

2.2 Eye-tracking Methods

There are many eye-tracking methods available. However, in this paper, the three prevalent methods are discussed. They are Electrooculography (EOG), Video-oculography (VOG) and Video-based combined pupil/corneal reflection.

2.2.1 Electrooculography

Electrooculography (EOG) is a method that detecting the eye movement by measuring the standing potential of retinal-corneal that result from the depolarizations and hyperpolarizations found between the retina and the cornea (Navarro, Vazquez and Guillen, 2018). The standing potential can be illustrated as an electrical dipole with a positive pole at the cornea and a negative pole at the fundus. This potential is obtained by measuring the voltage induced across the electrodes that placed around the eyes as the eye movement changes. Thus, EOG is obtained as a result of electric impulses triggered with different vertical and horizontal eye movements like gaze, wink, frown and blink. The amplitude of EOG can be varied from 50 to 3500μV and the frequency varies from 0 to 100Hz (Choudhari, et al., 2019). Besides that, the linear gaze angle is ± 30 degrees and every degree in eye movement causes 20 μV changes in

voltage. Generally, five electrodes are placed around the eyes in order to obtain the EOG. The position of the five electrodes is shown in Figure 2.1. The electrode D and E are used to detect the horizontal eye movement while the electrode B and C detect the vertical eye movement. Electrode A acts as a reference and is placed on the forehead.

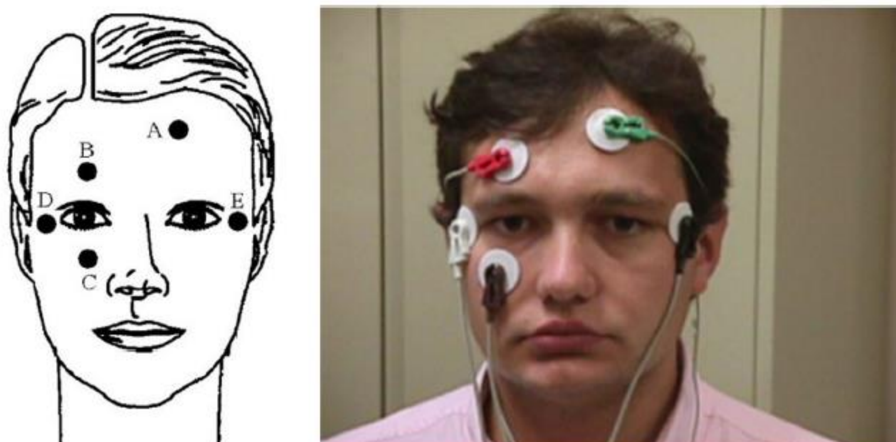


Figure 2.1: The Position of the Electrodes to Obtain EOG. (Navarro, Vazquez and Guillen, 2018)

According to Haslwanter and Clarke (2010), there are five sources of artifacts for EOG. First, the fluctuation of the corneal-retinal potential (CRP). The CRP is subject to impromptu fluctuation and has a high dependence on the amount of light. The CRP can reduce by up to 50% when thr human eye is adapted to dark circumstances. Next, the electrode drift is another source of artifacts. The electrode will produce slow drift potentials if there is improper care or damage on the electrode. High electrode impedance at the electrode-to-skin interface also can introduce signal drift and noise. The third source of artifacts is the placement of the electrode. The wrong placement of electrode can lead to crosstalk between vertical and horizontal components and directional asymmetries. A proper calibration measure is taken to correct these artifacts. The following source of artifacts is biological artifacts. The low-level signal is always confused by the electrocardiogram activity and muscle potentials. Lastly, the non-linearities in the vertical eye-position signal can also be the source of artifacts and it is caused by the bone structure around the eye and eyelid movement.

EOG method is more reliable than electroencephalography (EEG) as EOG is easier to detect than EEG. It is due to more consistent signal patterns that characterize EOG. In addition, the eye movement is easier to perform. Hence, it is used in many EOG-based human-computer interfaces (HCI). For instance, the blinking of an eye is treated as a "click" and used to make selection (Li, et al., 2017).

2.2.2 Video-oculography

Video-oculography (VOG) is a method that including various eye recording techniques to record the eye movement comprising the measurement of detectable eye features under translation or rotation, for instance, the limbus position, the pupil apparent shape and corneal reflection of the directed light source (Duchowski, 2007). It is video-based and is not invasive. VOG device usually has a camera that send images to the computer to process the images. There are generally two methods to be used in VOG for eye-tracking and it is depending on the condition of application such as the specification of the camera and also hardware, the desired head movement, cost and some other factors. The two methods are appearance-based and feature-based methods.

The appearance-based method is suitable for low-resolution images in various environments. The problem of gaze estimation is solved by mapping function from eye images to directions of the eye is looking. All the eye region pixel values are treated as high-dimensional feature vectors and as input for estimating gaze directions. On the other hand, gaze direction which is the output can be illustrated as x-y coordinates on the screen. It is a great advantage for low-quality images compared to the techniques used in the feature-based method. Although the precision of these methods is not satisfactory for some applications, they make the system become less restrictive and are robust when applied to comparably low-resolution cameras, for instance with a computer or a phone application or human-computer interaction. According to Larrazabu, Garci Cena and Martinez (2019), these methods have the main issue which is that the eye appearance depends on many considerations besides gaze direction. The considerations include the condition of imaging, the position of the head and the identities of subjects. Due to this issue, it is mandatory to have a person-specific training. Moreover, about thousands of individual training samples are

needed to determine the mapping coefficient because of the high dimensional feature vectors that necessary to be mapped into gaze direction. Estimating gaze direction from only eye appearance is still challenging even with the current research evolution in the study of computer vision. In short, the quality and diversity of the training data, as well as the generalization ability of the regression algorithm, will determine the performance of the appearance-based methods.

The feature-based methods are the methods that use extracted local features, for instance, eye corners, eye reflections or contours which are the most popular used method in VOG. In these methods, geometrically derived eye features from high-resolution eye images (Figure 2.2) are used and they are captured by focusing on the eye of the user. These methods are categorized into two main groups which are: 2D mapping-based gaze estimation methods and 3D model-based gaze estimation methods. For 3D model-based methods, the 3D gaze direction vector is directly computed from eye features based on the geometric eye model. Next, the gaze direction is intersected with the object being view in order to estimate the point of gaze. This 3D model-based method requires accurate user-dependent parameters such as cornea radii, angles between visual and optical axes, the distance between cornea center and pupil center to determine the cornea center and the eye vector. This method has very high accuracy on handling the head movements but requires a relatively complex initial set up process in which a single camera is needed with numbers of calibrated light sources or stereo cameras. Although the calibration process time can be reduced or avoided by using a very simplified eye model, the accuracy of results would be decreased significantly. For the 2D- mapping-based gaze estimation methods, they are based on finding the mapping function from 2D feature space like Pupil-Center-Corneal-Reflection (PCCR), contours, etc. to a gaze point, for instance, computer screen coordinates (Larrazabu, Garci Cena and Martinez, 2019). According to the authors, the direct measurement of the eye model parameter is not necessary throughout the system setup. They are comprised of the mapping function learning which simplifying the setup process itself.

Figure 2.2: Features from High-resolution Eye Images. (Larrazabu, Garci Cena and Martinez, 2019)

2.2.3 Video-Based Combined Pupil/Corneal Reflection

The point of regard is the direction or point of gaze on the perpendicular plane from the eyes. In order to get the point of regard measurement, the head movement should be fixed to the corresponding position of the eye relative to the head with the point of regard. The other way is measuring multiple ocular features to separate head movement from eye rotation. The features mentioned above are pupil center and corneal reflection.

Infra-red is generally used as a light source and the corneal reflection of the light source is measured relative to the pupil center. The direction or distance between the cornel reflection and pupil center is obtained to determine the point of regard of gaze direction. Corneal reflections are also known as Purkinje's images or Purkinje reflections. There are four Purkinje images formed and usually first Purkinje image is located by the video-based eye tracker. The four Purkinje reflections are a reflection from the outer surface of the cornea (P1), reflection from the rear surface of the cornea (P2), reflection from the outer surface of the lens (P3) and reflection from the rear surface of the lens (P4). The Purkinje reflection is illustrated in Figure 2.3. The P4 is an inverted image that is different from others and it is visible from within the eye itself. It is also reflected and formed at the same plane as the P1 with almost the same size with it but the intensity is less than 1% of that of P1 due to the change in the index of refraction at the rear of the lens.

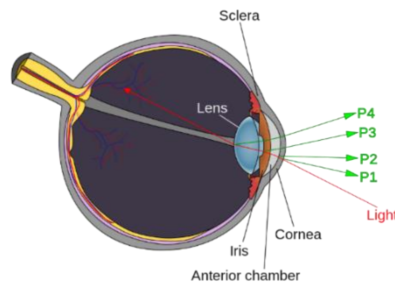


Figure 2.3: Purkinje Reflections. (Duchowski, 2007)

In order to isolate the eye movement from the head movement, two reference points are needed. The difference in position between corneal reflection and pupil center remains comparatively unchanged with minor head movements but it changes with pure eye rotation. In the calibration process of the eye tracker, the point of regard of a viewer can be measured on a planar surface that is perpendicular. Figure 2.4 below shows the relative positions of the pupil and First Purkinje reflections as the rotation of the left eye to fixate nine corresponding calibration points. The small white circle near the pupil in Figure 2.4 is the Purkinje's reflection (Duchowski, 2007).

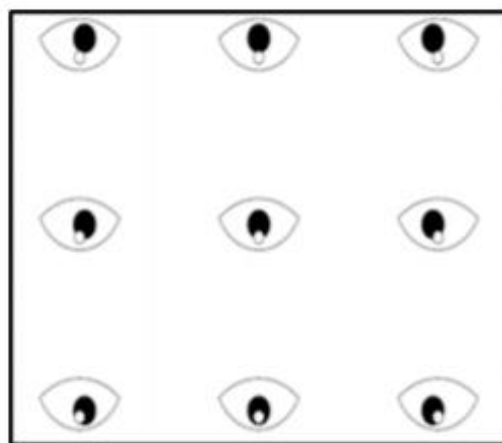


Figure 2.4: Relative Positions of the Pupil and First Purkinje Reflections. (Duchowski, 2007)

2.3 Eye Controlled Wheelchair Projects

Some engineers in Taiwan developed a powered wheelchair that controlled by using an eye-tracking system (Lin, et al., 2006). According to their article, a pupil-tracking goggle was used with a video CCD camera and a frame grabber

to analyze the pupil images. The pupil-tracking goggle allowed for consistent eye-tracking regardless of the head movement. In order to capture clearer and accurate pupil image without affecting the field of view of the user, a pinhole CCD camera was mounted under the rim of googles. A light bulb was also installed to brighten the image captured. Then, the gaze direction acted as "mouse" to select the command for the wheelchair. The interface for the command was divided into nine zones, as shown in Figure 2.5, which upper and lower zone represent forward and backward movement while the left and right zone represent the left and right movement respectively. On the other hand, the four corners and the middle zone represented stop command when the gaze direction falls onto it. The direction and the speed of the wheelchair depend on

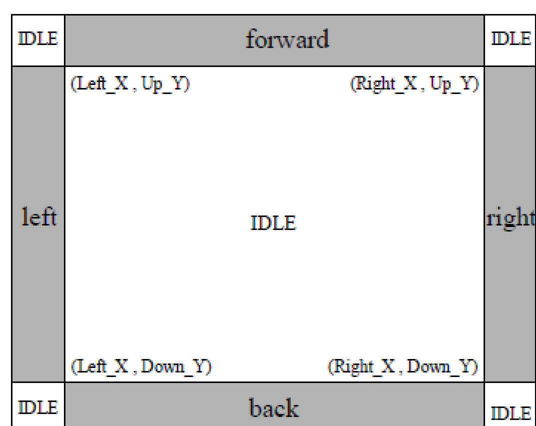


Figure 2.5: Command Interface. (Lin, et al., 2006)

the duration and the command zone that the users look at. The input command for a certain direction increases continuously until the increment value set is reached when the user gazes in that direction, then the control command is sent out. The movement of the wheelchair depends on the increment magnitude of the control command which ranges from 1 to 10. The larger the increment value, the more the difficulty to make a fine adjustment to change the direction. However, it is also time-consuming to make a large angle of rotation if the increment value is small. As a result of the practical test, they decided to set the increment value to 5 in a spacious room and increment value of 3 to be used in a school hallway.

Besides that, Gneo, et al. (2011) proposed a high-level schematic of a system (Figure 2.6) that integrates an eye-gaze tracking system (EGTS) and brain-computer interface (BCI) which using electroencephalogram (EEG). The eye movement in EGTS was used to select the desired command and the BCI acted as a mouse click to activate the selected command action. In the paper, the authors claimed that there was a lack of the systems that allow the user to directly look where the users want to go and in the existing systems, require the user to look at the GUI continuously in order to control the electric-powered wheelchair (EPW). Therefore, the current eye-controlled EPWs arise the two main problems. First problem is that the undesired command may be generated when the user is gazing at somewhere else, so-called Midas touch. Next, the user is required to stay focused in the desired direction as the GUI hardware might obstruct the visibility. EEG-actuated devices are generally to be considered slow for controlling complex robot movement. However, EEG-based BCI might be an effective binary switch for movement activation. The authors believed that the BCI activation command is reliable compared to the eye dwell time as some authors considered the eye-control techniques is still immature. In a nutshell, the proposal by the authors allowed the user to look around and search for the desired location, then activate the command action by EEG based BCI, overcoming the need to stare at the GUI and Midas touch.

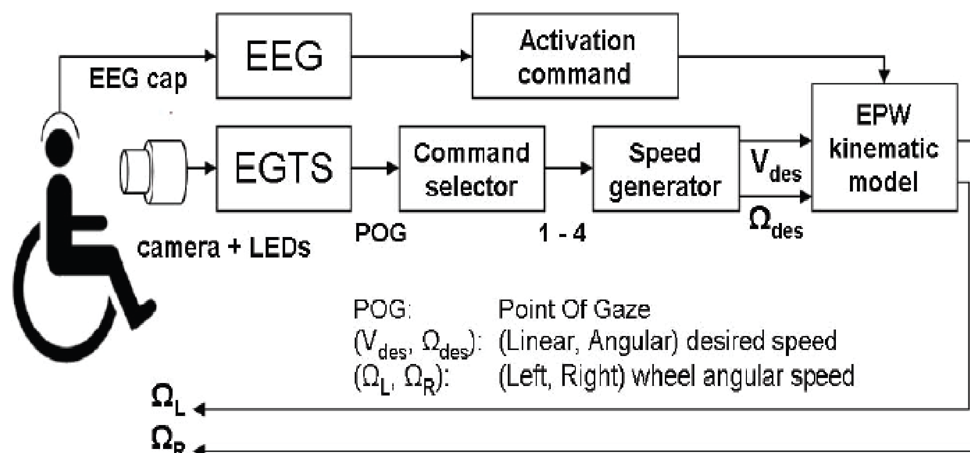


Figure 2.6: High-level Schematic of a System. (Gneo, et al., 2011)

Next, in the paper published by Pai, Ayare and Kapadia (2012), a new method to control and guide the wheelchair was discussed based on human-

computer interaction (HCI). Some eye-tracking techniques are discussed and the preferred technique was using a video camera. The hardware and software used were also discussed in this paper. The hardware included a camera that mounted on to the headgear, laptop, micro-controller and wheelchair controller. On the other hand, the software included micro-controller programming and Matlab which was used for image processing. In the image processing stage, each frame that divided the video was split into 3x3 sectors and the images were converted to grayscale. The sector that the pupil presented was recorded. A signal will be sent to the micro-controller if the pupil remains in that certain sector for a consecutive number of frames. The image processing is shown in Figure 2.7.

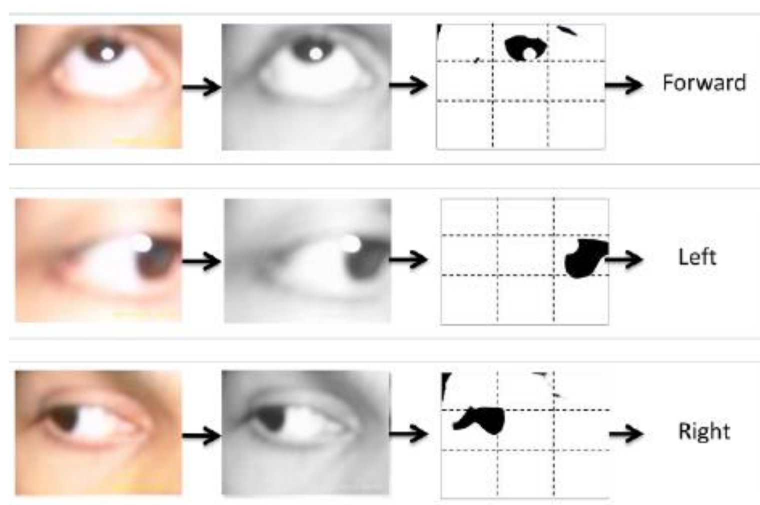


Figure 2.7: Outputs of the Image Processing Software. (Pai, Ayare and Kapadia, 2012)

A group of engineers from Villanova University also came out with a paper suggesting a design of an eye-controlled wheelchair (Plesnick, Repice and Loughnane, 2014). The eye tracker used contains an infrared sensor in the middle of the eye tracker and two infrared sources patched at both ends of the eye tracker which is shown in Figure 2.8. The infrared sources emitted the light into the user’s eye. Then, the infrared reflection was detected by the sensor and the eye location data was transferred to the computer. The calibration process was needed to calibrate the user's eye location. Labview program was used to translate the data from eye tracker and to send analog and digital voltage into

the Data Acquisition (DAQ) unit in order to control the wheelchair. It was discovered that the original joystick of the powered wheelchair attached to the main control module and generated basic analog voltages that directly proportional to the physical movement of the joystick. These voltages were used to determine the movement of the wheelchair and were detected by the control module through the pin connection that connected with the joystick. The joystick was removed and replaced by the DAQ unit. The control interface has four command blocks which indicate four directions (forward, backward, left and right). The user was required to gaze at the command block for at least 15 seconds in order to activate the command and move the wheelchair. It was done to prevent the wheelchair from executes an accidental command input.

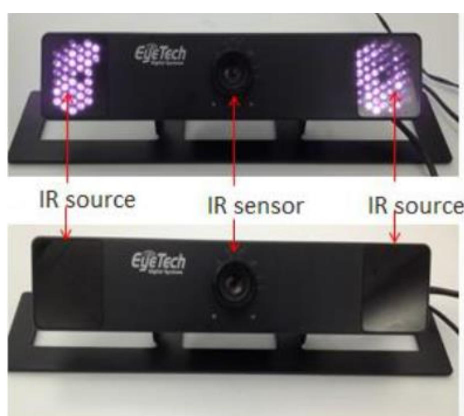


Figure 2.8: Eye-tracker. (Plesnick, Repice and Loughnane, 2014)

2.4 Introduction to Tobii

Tobii is a Swedish high-technology company which was founded in 2001. It develops and sells eye control/ tracking products. Tobii is recognized as a global leader in the field of eye-tracking. Tobii has many products that include licenses to the globally largest portfolio of eye-tracking patents, encompass critical aspects of algorithms, hardware design and user experience concepts. Tobii has three business units, which are Tobii Pro, Tobii Dynavox and Tobii Tech.

Tobii Pro is the unit that develops and provides eye-tracking solutions with deep and unique objectives for understanding human behaviour. It provides services to businesses and also researchers all around the world. This business unit has formed a consultancy organization that helps companies to improve

efficiency and quality on the production line. The business unit also performs marketing research.

Tobii Dynavox specializes in providing assistive technology solutions in helping and empowering individuals with disabilities or with special needs to communicate and live more independent lives. This business unit develops and designs products which include eye-tracking and touch-based communication software and devices.

Tobii Tech is the world's leading supplier of eye-tracking technology and focuses on market development intending to integrate eye-tracking and user-facing sensors and software into consumer electronics and other volume products. This business unit is at a commercially early stage. It focuses on areas include XR, personal computing and cases in the market area like healthcare.

2.5 Summary

There are a variety of eye-tracking methods and also image processing techniques available as well as the type of eye-tracking system. The eye-tracking system can be head-mounted or table-mounted. It is depending on what is the application going to carry out. But most of the cases in eye-controlled wheelchair, VOG method is used. Besides that, the eye-tracker used also should not be invasive if possible. Tobii is a reliable company in providing all the technology services related to eye-tracking. Tobii has three business units, Tobii Pro, Tobii Dynavox and Tobii Tech which are recognized as the market leader in their respective fields.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

This project uses an eye-tracking system to control the wheelchair. There are two methods to control the wheelchair. First, the users can control the wheelchair by moving their eyes and look at the screen interface in four directions which represent forward, backward, left, and right. The next method is to choose a destination from the screen interface through an eye tracker and the wheelchair will move to the destination automatically. Mapping of the place will be done and input to the laptop. It is just like the Google Map and the users just need to select a location, then the wheelchair will travel to that location automatically while avoiding any obstacles along the way with the aid of sensors. This will benefit the users who move to certain locations frequently as they do not need to keep inputting the commands. It will be tiring if they need to travel to a far location.

3.2 Hardware

There are some hardware used in this project such as an eye tracker, an autonomous wheelchair and eye-tracker mounting.

3.2.1 Eye Tracker

The eye tracker used in this project is Tobii eye-tracker 4C. The components comprise Eye core, EyeChip, Illumination Module, Host Interfaced module, and EyeSensor Module.

3.2.2 Wheelchair

An electric-powered wheelchair used in the project came with an original joystick. However, the joystick was then removed and was replaced with the eye tracker for controlling the wheelchair. Besides that, some modifications and features were added to the wheelchair. For example, a tray was installed to place the laptop in front of the user.

3.2.3 Eye tracker mounting

The eye tracker was mounted at the bottom edge of the laptop screen initially with a mounting bracket that is attached to the laptop with an adhesive strip. There is a slot at the back of the eye tracker in the centre to fit into the mounting bracket. It was found out that the eye tracker was not stable and was shaking when the wheelchair was moving. It will affect the performance of the eye tracker as the calibration that was done earlier will be varied. As a result, the eye tracker cannot track the eye position accurately. Therefore, a metal eye-tracker mounting was made to clamp the eye tracker separately from the laptop and to hold it at the two ends. Rubber cushions are added between the clamps and eye tracker to reduce the vibration.

3.3 Software

There are some open-source operating system and simulation tools will be used to develop the eye-tracker and wheelchair system in this project such as Ubuntu and Robot Operating System (ROS). The simulation of the wheelchair is conducted with using Gazebo and RVIZ to test the written codes.

3.3.1 Ubuntu

Ubuntu is an open-source operating system based on the Debian GNU/Linux distribution. It is freely available with both community and professional support. Ubuntu allows people to use it with their language and can customize their software to fit with their research or application.

3.3.2 Tobii Core SDK

The Tobii SDK provides the developer with Application Programming Interface (API) and framework to build gaze interaction application enhanced with the knowledge of the gaze and attention of the user. The developer can create a new interface, the addition of features and deeper immersion to provide the user with the next level of experience. It also provides some samples of the programming code of the Tobii eye tracker. Therefore, modifications of the code are easily carried out to fit the eye tracker with our project.

3.3.3 Robot Operating System (ROS)

With using ROS, the code is easily separated into packages containing small programs, which is called a node. ROS node is an executable file within the ROS package. The nodes can communicate with each other by using the ROS client library. A ROS Master is launched prior to the nodes and topics. It provides the naming and registration services of nodes. Besides that, it tracks the publishers and subscribers to topics as well as services.

A node can publish or subscribe to a topic. A topic is like a medium where the nodes can publish or subscribe to messages from it. There can be multiple nodes to publish or subscribe to a topic. ROS nodes have their message data type when publishing messages to a topic. Therefore, each topic is also classified by the ROS message type and the nodes will only receive a message of the same type. For example, the programming code that obtains the eye tracker position is considered as a node named "user_eye". It publishes the coordinate positions of the user's eye to a topic named "eye_direction". Then, another node using named "controller" subscribe to the topic to obtain the information as input to control the wheelchair.

In addition, there is also a function called service. It allows the software developer to create a client or server communication between nodes. It can be used to change the setting and parameter of the robot or ask for a specific action. For example, change the color background or add another robot in the simulation.

ROS is mainly using two languages which are C++ and Python. The developer can use roscpp library to write C++ code and rospy library to write Python code. Interestingly, the developer can have two nodes using different languages and still communicate with each other.

3.3.4 Gazebo

Gazebo is a robotic tool for robot simulation. It provides an accurate and efficient indoor and outdoor environment for robot simulation. Gazebo provides fundamental shapes such as a sphere, cylinder, and cubes for designing a simple robot model. Gazebo also provides 2D and 3D design interface in order to make the robot model more realistic and dynamic. Besides that, we can build a world

or an environment in Gazebo in which the robot is going to be tested in the real world. We can add objects via libraries such as the dustbin, table, wardrobe or any obstacles. In addition, we can plugin in a 2D layout of a house and build a 3D house based on the fundamental structure in the layout by using a building editor. To make the environment more realistic, Gazebo also has physical options such as forces, resistances, weights and etc. Gazebo also has robust sensors models, for instance, stereo camera, depth camera, laser, scanning lidar, and other general sensors. Figure 3.1 below shows the image of a Gazebo World.

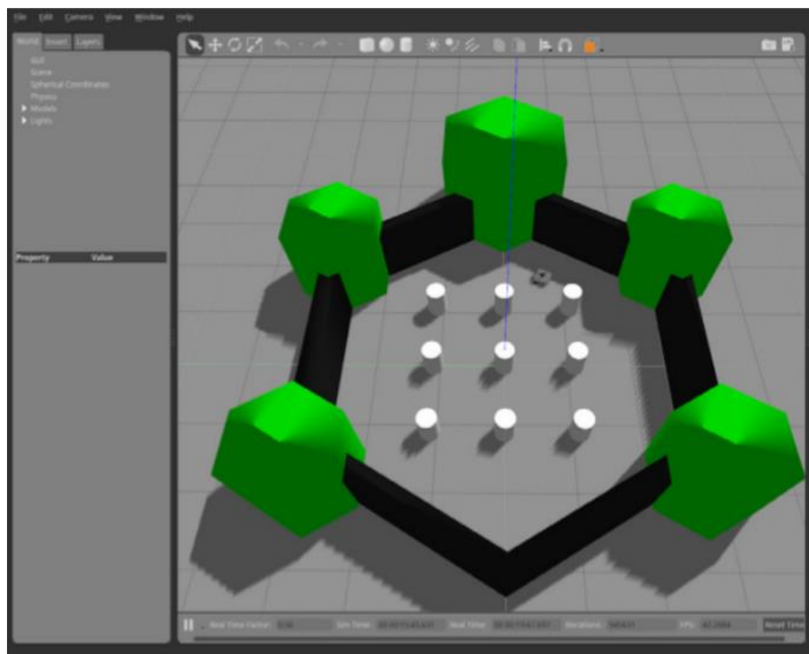


Figure 3.1: A Gazebo World.

3.3.5 RVIZ

RVIZ is a 3D visualization tool of the robot under ROS packages. We can have the vision to look at the world from the robot's eye through a camera that is installed in the robot. There are two main methods of inputting data and information into RVIZ. First, RVIZ understands sensor and state information such as laser scan, point clouds, camera and coordinate frames. Second, through the visualization markers that let the programmer input primitives like cubes, arrows, lines and also colour. The combination of sensor data and custom visualization markers make RVIZ a powerful tool for the development of robot.

Simultaneous Localization and Mapping (SLAM) is a technique to construct, draw or dating a map of an unknown space. It is one of the features

of Turtlebot. It is done by using a Gmapping ROS package and laser distance sensor. To do a virtual SLAM, we set up a robot in Gazebo world and launch the SLAM file in RVIZ as shown in Figure 3.2. Then, we remotely control the robot in Gazebo and control it to move all around the Gazebo environment to do the mapping. By then, a map is formed in RVIZ and it is shown in Figure 3.3. It is the same when using a wheelchair instead of a robot in Gazebo to do mapping in the real world.

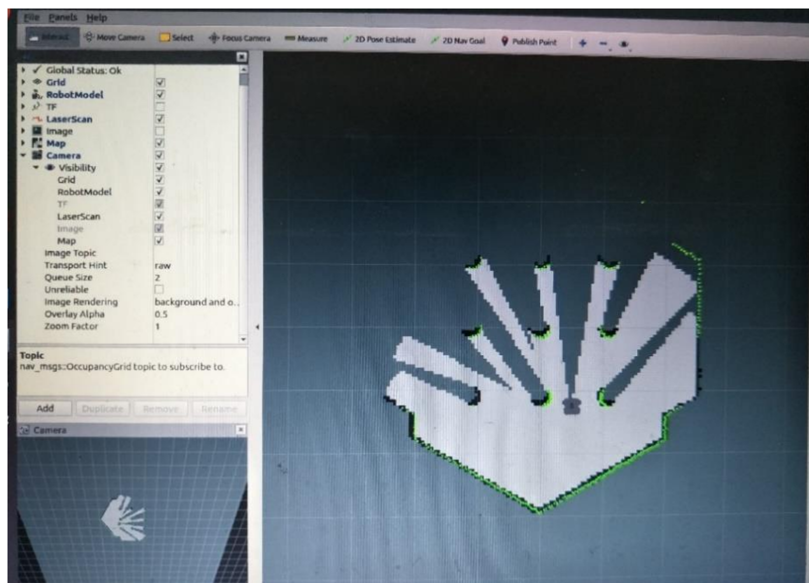


Figure 3.2: SLAM Launch File in RVIZ.

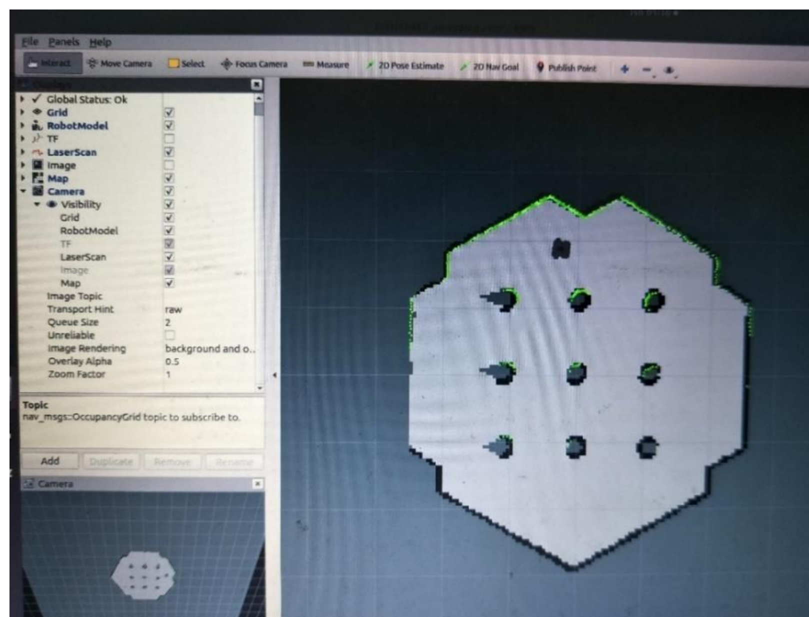


Figure 3.3: Generated Map of Gazebo World.

When the environment is mapped, coordinates can be obtained in the map. Therefore, navigation can be done after the coordinate of the location is obtained. The robot will move to the desired location while avoiding any obstacles along the path with the aid of Laser Distance Sensor.

3.4 Command-based Coding

The first step to start this project was to rewrite the code provided by Tobii SDK samples to get the eye stream data which is the coordinate of the user's eye when looking at the screen. It is done on the Windows OS laptop. After the coordinates of the screen interface is known, the eye positions of the users can be categorized into four directions which are forward, backward, left, and right. For instance, when a user wants to control the wheelchair to move forward, the user needs to gaze at the forward area. If the eye position of the user falls within the area around two seconds, the forward command will be sent to the Ubuntu laptop. Figure 3.4 shows the control interface of command-based action.

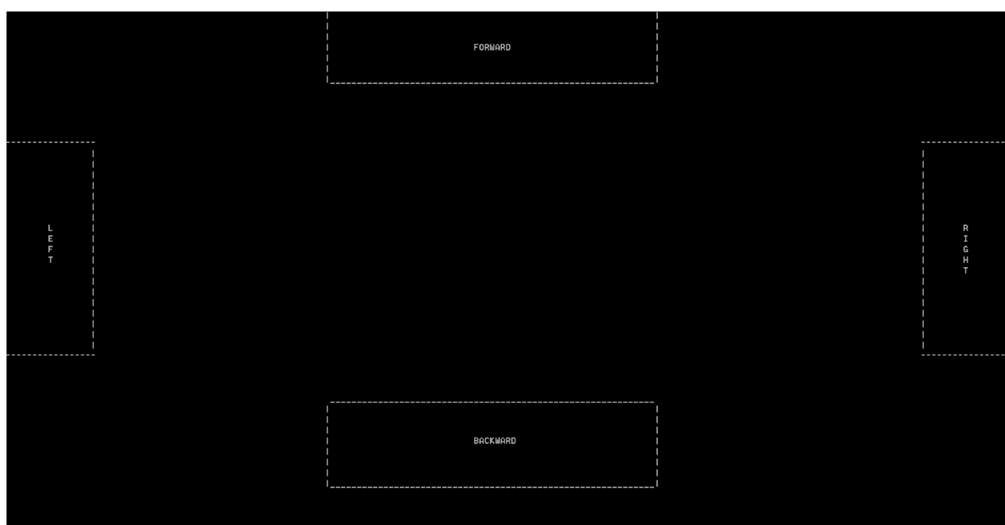


Figure 3.4: Command-based Control Interface.

Next, in the Ubuntu laptop, a C++ ROS code was written to receive the command output from the Windows OS laptop and then the command output was used to set the parameter. The parameter was based on the velocity. For example, the linear velocity is positive for the forward direction and is negative for the backward direction. Besides that, the angular velocity is positive when turning left and is negative when turning right. This parameter is then sent to

the ROS topic that controls the wheelchair and the wheelchair will move according to the parameter.

3.5 Move to goal/ Destination-based Coding

This ROS code was done in Windows OS laptop and it was almost the same with the coding above. The difference was that the coordinates were treated as a selection toward a destination instead of immediate command of action. There are six destinations listed on the screen interface and the user requires to gaze at the desired destination for around two seconds to select it. Then, the output will be sent to the Ubuntu laptop. The figure below shows the control interface of destination-based action.

Figure 3.5: Destination-based Control Interface.

To enable the autonomous navigation, mapping of the place was done using Gazebo and RVIZ in Ubuntu. After the mapping was done, the coordinates of the destination were obtained. Then, the coordinates were recorded in the programming code and were served as inputs to the wheelchair.

3.6 Communication between Two Laptops

There are two laptops used in this project. First, the Windows OS laptop which connected with the eye tracker. The second laptop is the Ubuntu laptop which runs ROS and controls the wheelchair.

To set up communication between two laptops, the first step is to configure the ROS master. There should be only one master for handling the system. Since only the Ubuntu laptop has ROS, we run the roscore on the Ubuntu laptop to configure the ROS master.

Next, in the Windows OS laptop, we are using ROS.Net which is a series of C# projects that allow a MANAGED .NET application to communicate traditional ROS nodes. The code of the Tobii eye tracker is also using C#. To use ROS.Net, the IPv4 address of the Windows OS laptop is set as ROS_HOSTNAME in windows environment variables as shown in Figure 3.6. Besides that, Figure 3.7 shows that the "/etc/hosts" file is modified to resolve the master's name to IP address. The HOST_NAME and IP address in "/etc/hosts" file are from Ubuntu Laptop. After all the configurations above have been completed, the two laptops can communicate with each other. The output of the code written in Windows laptop can be sent to Ubuntu Laptop in the form of ROS nodes and ROS topics with one ROS master.

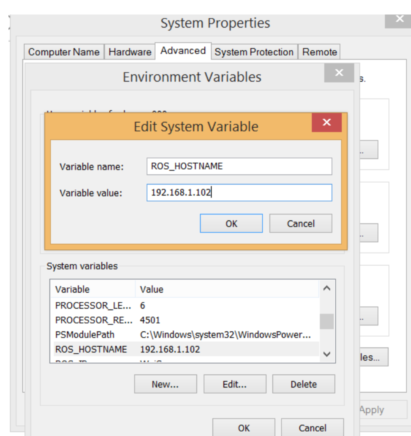


Figure 3.6: Setting of Windows Environment Variables.

```
File  Edit  Format  View  Help
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost

#master IP address  #master hostname
192.168.1.106        WeiSeng
```

Figure 3.7: Modify "/etc/hosts'' File to Resolve the Master's Name to IP.

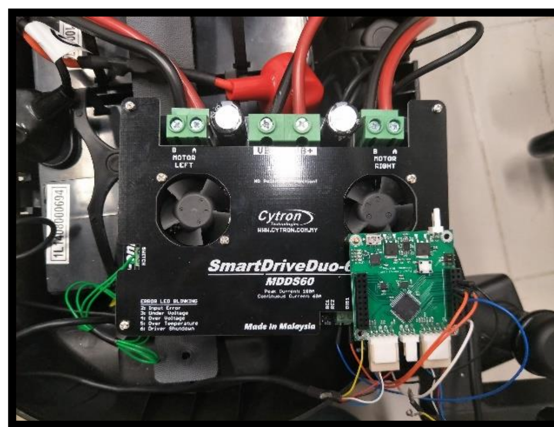
CHAPTER 4

RESULTS AND DISCUSSION

4.1 Assembly of Wheelchair and Eye Tracker

The assembly and the modification of the wheelchair were completed. Figure 4.1 (a) shows the base of the wheelchair which includes the rotors, the batteries, the motor driver with micro-controller (Figure 4.1(b)), and also the wire connections. Besides that, a tray was installed in order to place the laptop in front of the user which is shown as Figure 4.2.

(a)

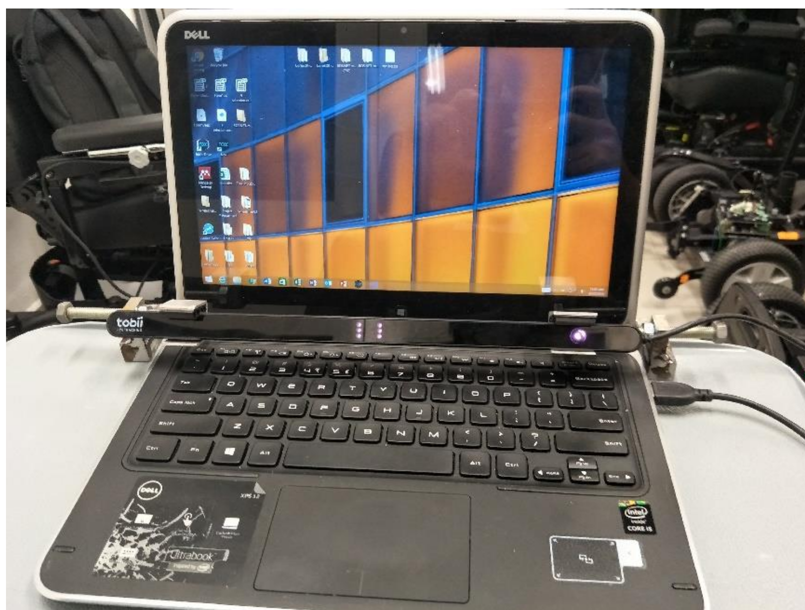


(b)

Figure 4.1: (a) Wheelchair Base, and (b) Motor Driver with Micro-Controller.

Figure 4.2: Wheelchair with Tray.

Besides that, a mounting was also made to hold the two ends of the eye tracker to minimize vibration when the wheelchair is moving. There are rubber cushions between the clamps and the eye tracker to absorb the vibration. Figure 4.3 shows the position of the eye tracker and the mounting on the tray.



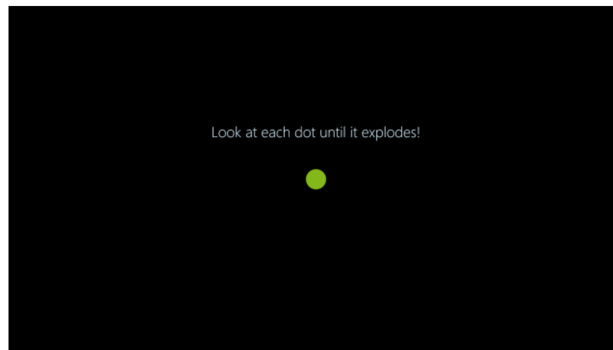
(a)

(b)

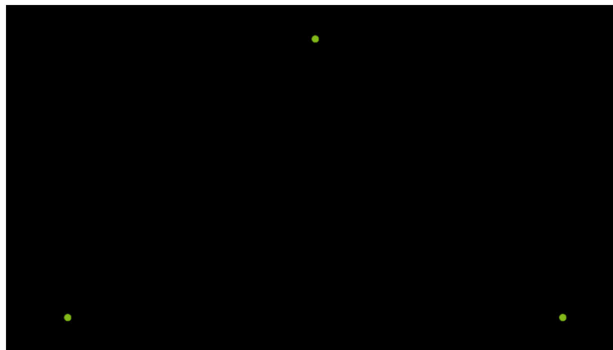
Figure 4.3: Eye Tracker Mounting

4.2 Calibration Process

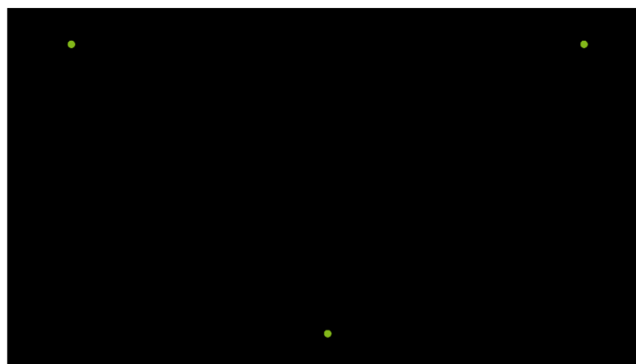
A calibration process of the Tobii eye tracker is required when a user uses the eye tracker. The user is required to do the calibration process every time as the position of the eye may be different when the user sits in the wheelchair the next time. The recorded calibration data previously may have some variation with the current eye position. The calibration process will measure the geometric characteristics of the user's eye and the characteristics are used to integrate with an internal, anatomical 3D eye model to calculate the gaze data. The model obtains the data such as the shapes, light refraction, and refraction properties of different parts of the eyes (TobiiPro, n.d.). An eye tracker consists of a camera, illuminator that emits infrared light, and algorithms. During the calibration, the eye tracker will emit infrared light. The light is reflected in the user's eye and the reflection will be captured by the camera. The calibration requires the user to look at the green dots, which are shown in Figure 4.4, that are displayed on the screen until the dots explode. The camera will capture the eye image of the user when he looks at the dots. After some calculation and filtering via the algorithm, the eye tracker can track the eye movement of the user on the screen.



(a)



(b)



(c)

Figure 4.4: Calibration Dots in Calibration Process

4.3 Command-based Action

After all the configurations were properly set as mentioned in section 3.6, the two laptops were able to communicate with each other. The first step to run the command-based action was to export the ROS_MASTER_URI in Ubuntu laptop and start the ROS master which is shown in Figure 4.5.

```
roscore http://WeiSeng:11311/
File  Edit  View  Search  Terminal  Tabs  Help
roscore http://W...  ×  weiseng@WeiSe...  ×  /home/weiseng/...  ×  weiseng@WeiSe...  ×

weiseng@WeiSeng:~$ export ROS_MASTER_URI=http://192.168.1.106:11311
weiseng@WeiSeng:~$ roscore
... logging to /home/weiseng/.ros/log/49997b6a-73eb-11ea-be25-d07e35f6ba8a/rosla
unch-WeiSeng-15448.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
WARNING: disk usage in log directory [/home/weiseng/.ros/log] is over 1GB.
It's recommended that you use the 'rosclean' command.

started roslaunch server http://WeiSeng:33393/
ros_comm version 1.14.3


SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.3

NODES

WARNING: ROS_MASTER_URI [http://192.168.1.106:11311] host is not set to this mac
hine
auto-starting new master
process[master]: started with pid [15458]
ROS_MASTER_URI=http://WeiSeng:11311/

setting /run_id to 49997b6a-73eb-11ea-be25-d07e35f6ba8a
process[rosout-1]: started with pid [15470]
started core service [/rosout]
```

Figure 4.5: Exports ROS_MASTER_URI and Starts ROS Master.

In the Windows laptop, the command based code with the node named "user_eye" was launched and an eye tracker was connected to the laptop. As shown in Figure 4.6, the connection between the two laptops was established. Then, the control interface in Figure 3.4 would be shown on the screen. The user can start to control the wheelchair by using the eye tracker. In the Ubuntu laptop, the "controller" node was run and the command was successfully received as shown in Figure 4.7. The output was then sent to control the wheelchair.

```
Adding pollthreadlistener :Void checkForShutdown()
Adding pollthreadlistener Ros_CSharp.ConnectionManager:Void removeDroppedConnect
ions()
Publisher update for [/clock]
Began asynchronous xmlrpc connection to http://WeiSeng:45359/ for topic [/clock]

Connecting via tcpros to topic [/clock] at host [WeiSeng:38155]
Init transport publisher link: /clock
Connected to publisher of topic [/clock] at  [WeiSeng:38155]
Adding pollthreadlistener Ros_CSharp.Publication:Void processPublishQueue()
Finalize transport subscriber link for /rosout
```

Figure 4.6: Console Window in Windows OS Laptop.

```
weiseng@WeiSeng: ~
File Edit View Search Terminal Tabs Help
roscore htt... × weiseng@... × /home/weis... × weiseng@W... × weiseng@W... ×
weiseng@WeiSeng:~$ rosrun fyp controller
Move Forward
Move Left
Move Left
Move Right
Move Backward
```

Figure 4.7: Terminal of Ubuntu Running the “controller” Node.

Figure 4.8 below shows a bigger picture of the whole process. The oval shape in the figure represents ROS nodes and rectangular shape represents ROS topics. In Windows laptop, a ROS node named "user_eye" was created and the command was published through a ROS topic named "eye_direction". In the Ubuntu laptop, a "controller" node subscribed to the "eye_direction" topic to get the command sent by the "user_eye" node. Then, the "controller" node published a new command to the "cmd_vel" topic which was subscribed by the "/gazebo" node. The "/gazebo" is the node that represents the wheelchair. As a result, the robot was controlled according to the command received from the "cmd_vel" topic.

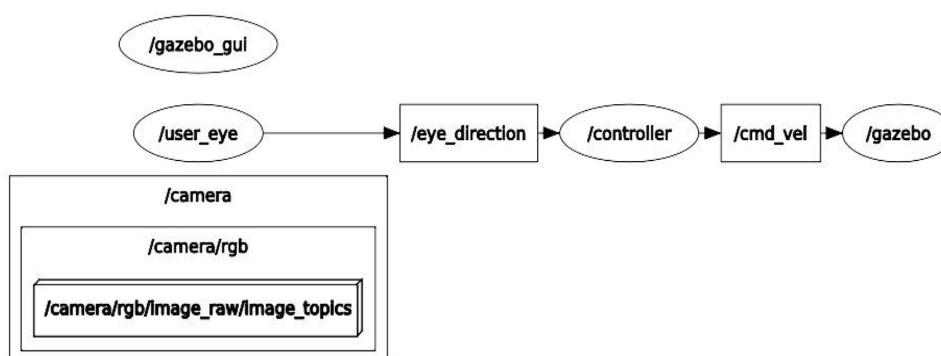


Figure 4.8: ROS Nodes and ROS Topics in Command-based Action.

4.4 Destination-based Action

For the destination based action, the destination-based code in Window OS laptop was run after the ROS master had launched. The ROS node and the ROS topic were the same as command-based which are "user_eye" and "eye_direction". The output in Figure 4.6 was shown again to indicate that the connection between the two laptops was established. Then, the screen interface as in Figure 3.5 was shown to ask the user to choose a location. After the user gazed at a particular selection for around two seconds, a location would be selected and displayed on the screen as shown in Figure 4.9. While in the Ubuntu laptop, the "map_navigation_node" was run. Once the node received data from the "eye_direction" topic, the node sent out the coordinate of location that the user selected and called for the “move_base” service (navigation). Then, the wheelchair moved to the destination according to the planned path while avoiding the obstacles. Figure 4.10 shows the output of the code.

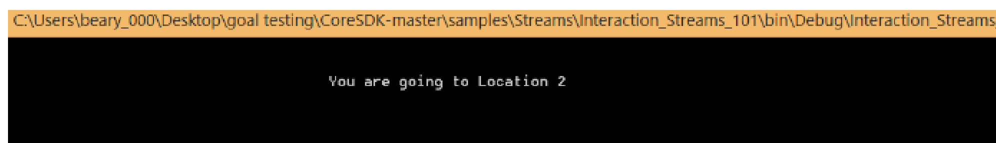


Figure 4.9: Display of the Selected Location.

```
File  Edit  View  Search  Terminal  Tabs  Help
roscore http://WeiSe...  /home/weiseng/catk...  /home/weiseng/catki...  weiseng@WeiSeng: ~
weiseng@WeiSeng:~$ rosrun simple_navigation_goals map_navigation_node
[ INFO] [1585998576.038881097, 140.943000000]: Sending goal location ...
[ INFO] [1585998650.697700410, 160.247000000]: You have reached the destination
[ INFO] [1585998650.787805886, 160.274000000]: Congratulations!
```

Figure 4.10: Running of the “map_navigation_node”.

In RVIZ, the simulated wheelchair was first located at the initial position which is shown in Figure 4.11. The “map_navigation_node” would receive data which was the selection of the user after it had subscribed to the “eye_direction” topic. Then, the “map_navigation_node” request for the service from the “move_base” server to carry out the navigation. The simulated wheelchair was then computed a planned path based on the coordination of the selected location and moved according to the path to the destination. It is shown in Figure 4.12. The path may be changing accordingly when the simulated wheelchair detects

an obstacle or it finds a shorter distance. Figure 4.13 shows that the simulated wheelchair has reached the destination.

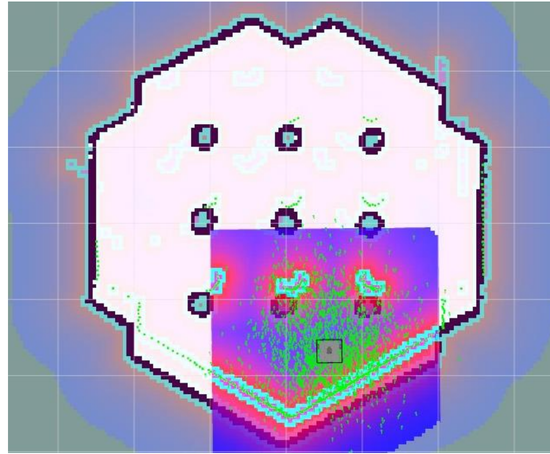


Figure 4.11: Initial Position of the Simulated Wheelchair in RVIZ

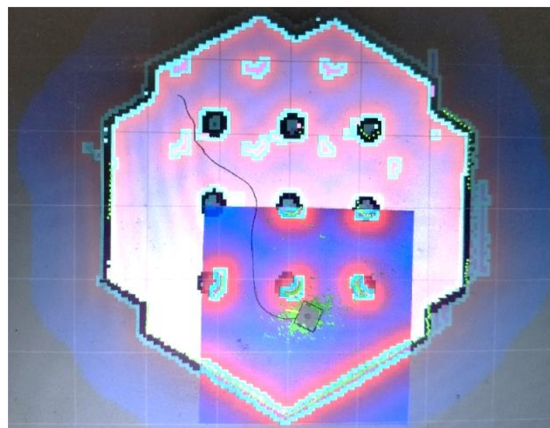


Figure 4.12: The Simulated Wheelchair is Navigating to the Destination.

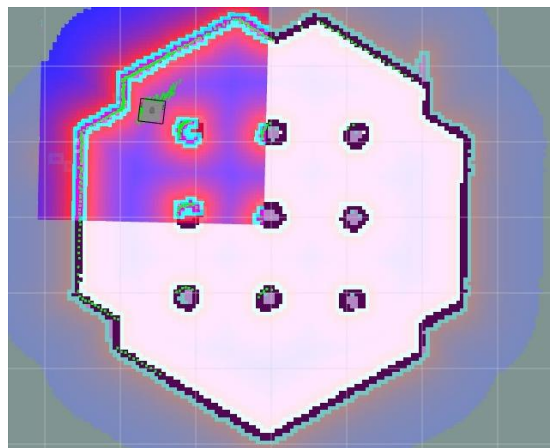
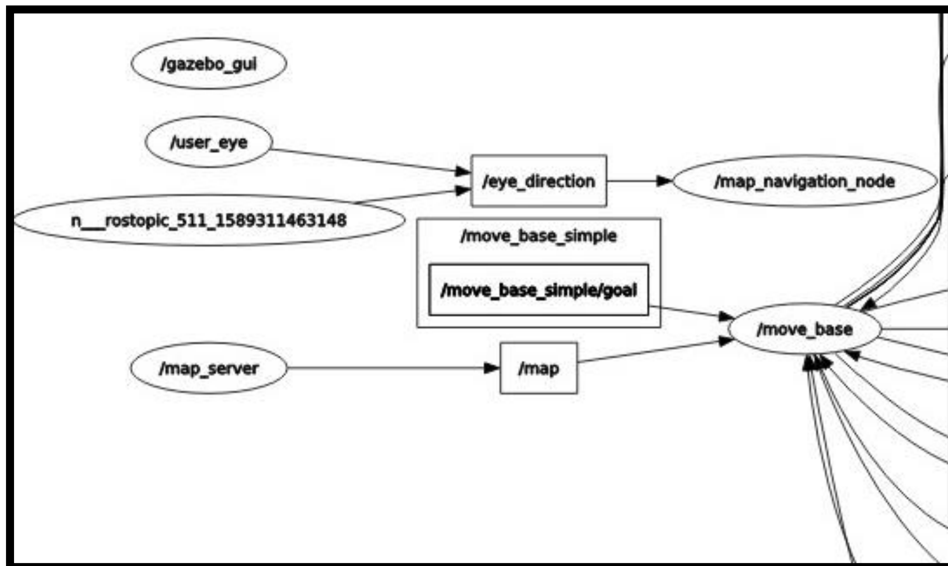


Figure 4.13: The Simulated Wheelchair Reached the Destination

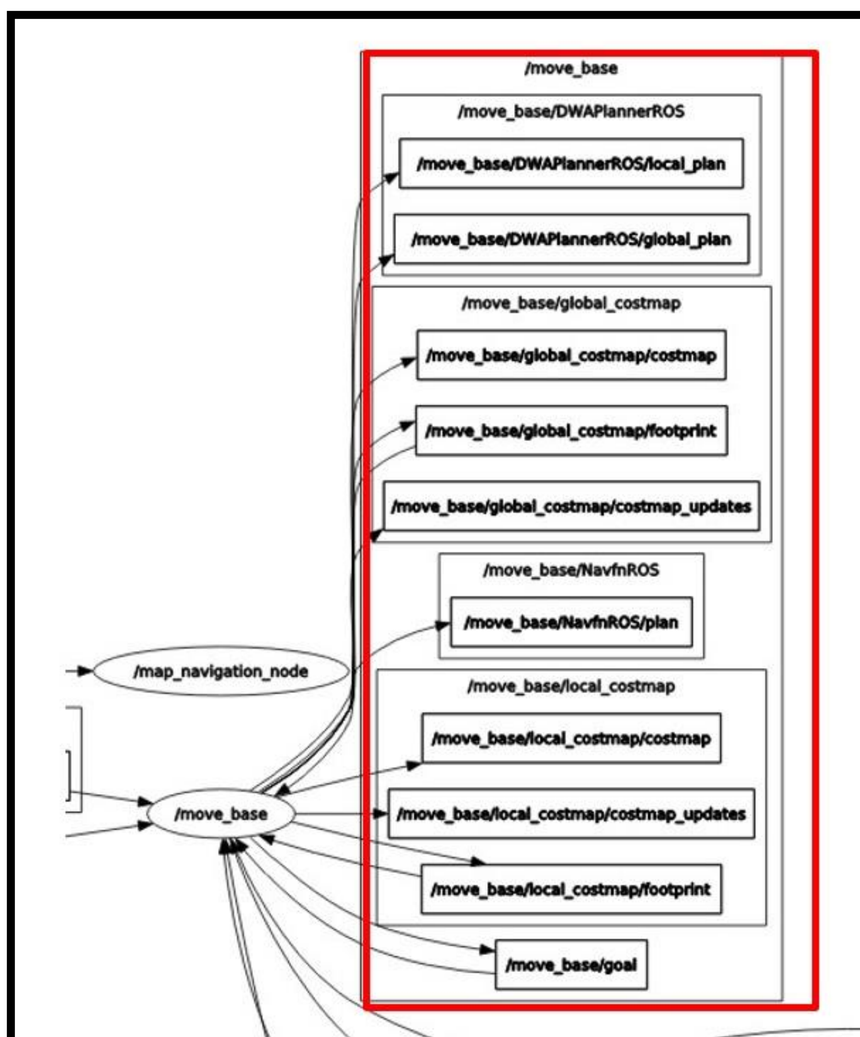
The figures below show the overview of all the nodes, topics, and some plugins. It is the same as the command-based code, in the Windows laptop, a node named “user_eye” was created and the output was then published to a topic named “eye_direction”. While in the Ubuntu laptop, the “map_navigation_node” was subscribed to the “eye_direction” topic and obtained the output that was published by the “user_eye” node. All of these can be seen in Figure 4.14(a).

There is a “move_base” node shown in Figure 4.14(b). The “move_base” node is the major component in the navigation stack. It has some configuration options (components in the red rectangle shape outline in Figure 4.14(b)). For example, the “move_base/global_costmap” is the configuration used to create a long-term path over the environment to do the navigation while the “move_base/local_costmap” is used for local planning and obstacle avoidance. The “move_base” node also subscribe to some topics. First, it is the “/map” topic. It is to set up an environment where the robot is located. The second subscribed topic is “tf”. It is the topic that the simulated wheelchair publishes information about the relationship between the coordinate frames. The third subscribed topic is “odom” which stores information about the estimation of position and velocity of the simulated wheelchair. Next, the “move_base” node also subscribes to the “scan” topic which has the data of the laser and uses the data in the obstacle avoidance. Besides that, the “move_base_simple/goal” provides a non-action interface to “move_base” for users that do not care about tracking the execution status of their goals.

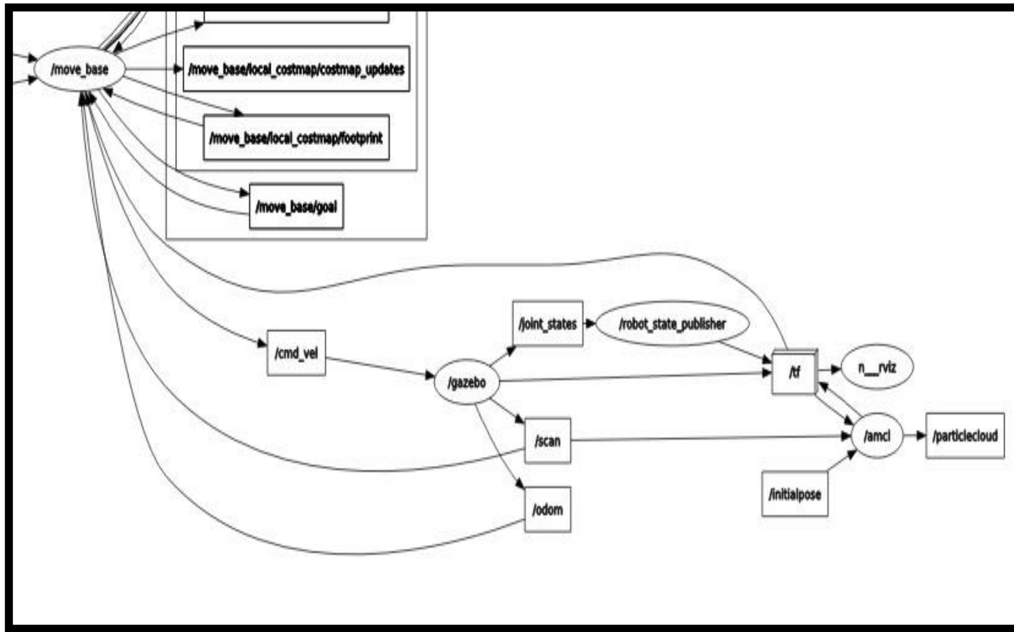
After all the computation, the “move_base” node will publish messages to “cmd_vel”. Then, the “gazebo” node subscribes to the “cmd_vel” to get the information and publishes the data to some topics to control the simulated wheelchair. It is shown in Figure 4.14(c).



(a)



(b)



(c)

Figure 4.14: ROS Nodes, Topics and Service in Destination-based Action.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In this project, an eye tracker controlled wheelchair is developed. There are two controlling methods, namely the command-based method and destination-based method. The user can choose to use either method depending on the travel distance. They are four controlling commands provided for command-based action: forward, backward, left, and right. For the destination-based action, six locations are determined for selection. Both of the controlling methods were successfully developed and simulated with robotic tools.

The eye tracker was originally mounted with a mounting bracket attached to the bottom of the screen frame. However, it was found that the eye tracker was shaking and was not stable whenever the wheelchair moved. Hence, a metal eye tracker mounting was made to clamp the eye tracker separately from the laptop and to hold it at the two ends to increase its stability and to reduce vibration. Besides that, rubber cushions were also added between the clamps and eye tracker to further absorb the vibration. Therefore, it is more stable than the original suggested mounting method by Tobii in which the eye tracker was only supported at the centre with a mounting bracket attached to the laptop frame.

An eye tracker controlled wheelchair is suitable for most of the users. A joystick wheelchair that needs to be operated by hand is not feasible for stroke patients or patients with severe motor disabilities. Therefore, an eye tracker controlled wheelchair that provides two controlling methods in this project is a better choice to be used to improve the mobility of paralyzed patients.

5.2 Recommendations for Future Works

It is recommended that the screen interface can be improved. The graphic and the layout can be designed in a way that is more user-friendly. In this project, the codes for command-based and destination based are written separately. It is suggested that both of the codes can be combined in one code and the user can

switch between command-based action and destination-based action freely. It is also recommended that the eye tracker can be improved so it can be used in Ubuntu. Therefore, only one laptop is needed. Besides that, the mounting of the eye tracker can be manufactured using a light but strong material instead of using metal as in this project.

REFERENCES

Choudhari, A.M., Porwal, P., Jonnalagedda, V. and Meriaudeau, F., 2019. An Electrooculography based Human Machine Interface for Wheelchair Control. *Biocybernetic and Biomedical Engineering*, 39(3), pp. 673-685

Duchowski, A.T., 2017. *Eye Tracking Methodology*. Second Edition. New York: Springer International Publishing.

Furman, J.M. and Wuyts, F.L., 2012. Vestibular Laboratory Testing. In: Aminoff, M.J., Sixth Edition. 2012. Philadelphia: Sauders. pp. 699-723.

Gaitech EDU, 2016. *Map-Based Navigation*. [Online]. Available at: <https://edu.gaitech.hk/turtlebot/map-navigation.html> [Accessed 12 April 2020].

Gneo, M., Severini, G., Conforto., S., Schmid, M. and D'Alessio, T., 2011. Towards a Brain-Activated and Eye-Controlled Wheelchair. *International Journal of Bioelectromagnetism*, 13(1), pp. 44-45.

Hari, S. and Jaswinder, S., 2012. Human Eye Tracking and Related Issues: A Review. *International Journal of Scientic and Research Publication*, 2(9), pp. 1-9.

Haslwanter,T. and Clarke, A.H., 210. Eye movement measurement: electro-oculography and video-oculography. Handbook of Clinical Neurophysiology, Volume 9, pp. 61-79.

iMotions, 2015. Top 8 Eye Tracking Appications in Research. [blog] 4 August 2015. Available at <https://imotions.com/blog/top-8-applications-eye-tracking-research/> [Accessed 28 June 2019]

Larrazabal, A.J., Garci Cena, C.E. and Martinez, C.E., 2019. Video-oculography eye tracking towards clinical application: A review. *Computer in Bioogy and Medicine*, Volume 108, pp. 57-66.

Li, Y., He, S., Huang, Q., Gu, Z. and Yu, Z.L., 2017. A EOG-based Switch and Its Application for "Start/Stop" Control of a wheelchair. *Neurocomputing*, Volume 275, pp. 1350-1357.

Lin, C-S., Ho, C-W., Chen, W-C., Chiu, C-C. and Yeh, M-S., 2006. Power Wheelchair controlled by eye-tracking system. *Optica Application*, 36(2-3), pp. 401-402.

Mandel, C., Laue, T. and Autexier, S., 2018. Smart Wheelchair. Smart Wheelchair and Brain-Computer Interfaces. pp. 291-322

Navarro, R.B., Vazquez, L.B. and Guillen, E.L., 2018. EOG-based wheelchair control. In: Diez, P., 2018. *Smart Wheelchair and Brain-computer Interfaces*. Madrid: Academic Press. pp. 381-403.

Pai, S., Ayare, S. and Kapadia, R., 2012. Eye Controlled Wheelchair. International Journal of Scientific & Engineering Research, 3(10), pp. 1-5.

Plesnick, S., Repice, D. and Loughnane, P., 2014. Eye-Controlled Wheelchair. *2014 IEEE Canada International Humanitarian Technology Conference-(IHTC)*, pp. 1-4.

TobiiPro, n.d. *What happens during the eye tracker calibration*. [online]. Available at < https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/what-happens-during-the-eye-tracker-calibration/>. [Accessed 13 May 2020].

APPENDICES

APPENDIX A: Command-based Code in Window OS Laptop

```
using Ros_CSharp;
using XmlRpc_Wrapper;
using Int32 = Messages.std_msgs.Int32;
using String = Messages.std_msgs.String;
using m = Messages.std_msgs;
using gm = Messages.geometry_msgs;
using nm = Messages.nav_msgs;
using sm = Messages.sensor_msgs;
using System.Text;
using Tobii.Interaction;
#endregion

namespace Interaction_Streams_101
{

    public class Program
    {
        public static void Main(string[] args)
        {

            ROS.Init(args, "user_eye");
            NodeHandle node = new NodeHandle();
            Publisher<m.String> P_Sender = node.advertise<m.String>("eye_direction", 1000);

            while (ROS.ok)
            {
                var host = new Host();
                var gazePointDataStream = host.Streams.CreateGazePointDataStream();

                Thread.Sleep(1000);
                Console.Clear();
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\tFORWARD \t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |__________________________________________________________| ");
                Console.WriteLine("\n\n\n\n\n----------------\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    ----------------");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\tL\t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |\t    R");
                Console.WriteLine("\tE\t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |\t    I");
                Console.WriteLine("\tF\t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |\t    G");
                Console.WriteLine("\tT\t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |\t    H");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |\t    T");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("\t \t| \t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    |  ");
                Console.WriteLine("----------------\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t    ----------------");
                Console.WriteLine("\n\n");
                Console.WriteLine("\t\t\t\t\t\t\t     __________________________________________________________ ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\tBACKWARD \t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |\t\t\t\t\t\t\t\t\t  | ");
                Console.WriteLine("\t\t\t\t\t\t\t    |__________________________________________________________| ");
```

```
                    gazePointDataStream.GazePoint((x, y, ts) =>
                    {
                        if (x > 600 && x < 1200 && y > 0 && y < 110)
                        {
                            String direction = new String("Forward");
                            P_Sender.publish(direction);
                        }
                        else if (x > 600 && x < 1200 && y > 750)
                        {
                            String direction = new String("Backward");
                            P_Sender.publish(direction);

                        }
                        else if (x > 0 && x < 250 && y > 270 && y < 650)
                        {
                            String direction = new String("Left");
                            P_Sender.publish(direction);
                        }
                        else if (x > 1600 && y > 270 && y < 650)
                        {
                            String direction = new String("Right");
                            P_Sender.publish(direction);
                        }
                        else
                        {
                            String direction = new String("Idle");
                            P_Sender.publish(direction);
                        }

                        Thread.Sleep(10);
                    });

                    Console.ReadKey();
                    host.DisableConnection();

                }

                ROS.shutdown();
                ROS.waitForShutdown();

            }
        }
    }
```

APPENDIX B: Command-based Code in Ubuntu Laptop

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"
#include "std_msgs/String.h"
#include <sstream>
#include <iostream>
#include <string>
using namespace std;

ros::Subscriber user_subscriber;
ros::Publisher velocity_publisher;

geometry_msgs::Twist vel_msg;
const double PI = 3.1415926536;

void move(string direction);

int f_count=0, b_count=0, l_count=0, r_count=0 , i_count;

void user_inputCallback(const std_msgs::String::ConstPtr& msg)
{
   std::string a_input=msg->data.c_str();
   std::string user_input=" ";
   if(a_input.compare("Forward")==0)
   {
     f_count++;
     b_count=0;
     l_count=0;
     r_count=0;
     i_count=0;
   }
   else if(a_input.compare("Backward")==0)
   {
     f_count=0;
     b_count++;
     l_count=0;
     r_count=0;
     i_count=0;
   }
   else if(a_input.compare("Left")==0)
   {
     f_count=0;
     b_count=0;
     l_count++;
     r_count=0;
     i_count=0;
   }
   else if(a_input.compare("Right")==0)
   {
     f_count=0;
     b_count=0;
     l_count=0;
     r_count++;
     i_count=0;
   }
   else if(a_input.compare("Idle")==0)
   {
     f_count=0;
     b_count=0;
     l_count=0;
     r_count=0;
     i_count++;
}

   if (f_count==180)
   {
     user_input="Forward";
     f_count=0;
   }
   else if(b_count==180)
   {
     user_input="Backward";
     b_count=0;
   }
   else if (l_count==180)
   {
     user_input="Left";
     l_count=0;
   }
   else if (r_count==180)
   {
     user_input="Right";
     r_count=0;
   }
  else if(i_count==70)
   {
     user_input ="Idle";
     i_count=0;
   }

   move (user_input);
}
```

```
int main(int argc, char **argv)
{
    ros::init(argc, argv, "controller");
    ros:: NodeHandle n;
    geometry_msgs::Twist vel_msg;
    user_subscriber= n.subscribe("eye_direction",1000, user_inputCallback);
    velocity_publisher = n.advertise<geometry_msgs::Twist>("/cmd_vel", 1000);
    ros::Rate loop_rate(1.0);

    ros::spin();
    return 0;
}


void move(string direction)
{
   geometry_msgs::Twist vel_msg;
   double desired_angle;

   if(direction.compare("Forward")==0)
   {
     vel_msg.linear.x= 1;
     cout<< "Move Forward"<<endl;
     velocity_publisher.publish(vel_msg);
   }
   else if(direction.compare("Backward")==0)
   {
     vel_msg.linear.x=-1;
     cout<<"Move Backward"<<endl;
     velocity_publisher.publish(vel_msg);

   }
   else if(direction.compare("Right")==0)
   {
     vel_msg.angular.z=-1;
     cout<<"Move Right"<<endl;
     velocity_publisher.publish(vel_msg);
   }
   else if(direction.compare("Left")==0)
   {
     vel_msg.angular.z=1;
     cout<<"Move Left"<<endl;
     velocity_publisher.publish(vel_msg);
   }
   else if(direction.compare("Idle")==0)
   {
     vel_msg.linear.x= 0;
     vel_msg.angular.z=0;
     cout<<"Idle"<<endl;
     velocity_publisher.publish(vel_msg);
   }
}
```

APPENDIX C: Destination-based Code in Window OS Laptop

```
using nm = Messages.nav_msgs;
using sm = Messages.sensor_msgs;
using System.Text;
using Tobii.Interaction;
#endregion

namespace Interaction_Streams_101
{

    public class Program
    {
        public static void Main(string[] args)
        {
            ROS.Init(args, "user_eye");
            NodeHandle node = new NodeHandle();
            Publisher<m.String> P_Sender = node.advertise<m.String>("eye_direction", 1000);

            while (ROS.ok)
            {
                var host = new Host();
                var gazePointDataStream = host.Streams.CreateGazePointDataStream();

                Thread.Sleep(1000);
                Console.Clear();
                Console.WriteLine("\n\n\t\t\t\t\t\t\t\t\t  Select Your Destination\n\n\n\n");
                Console.WriteLine("\t\t\t\t\t\tLocation 1\t\t\t\t\t\t\t\tLocation 2\n\n\n\n\n\n\n\n");
                Console.WriteLine("\n\n\n\n");
                Console.WriteLine("\t\t\t\t\t\tLocation 3\t\t\t\t\t\t\t\tLocation 4\n\n\n\n\n\n\n\n");
                Console.WriteLine("\n\n\n\n\n");
                Console.WriteLine("\t\t\t\t\t\tLocation 5\t\t\t\t\t\t\t\tLocation 6\n\n\n\n\n\n");

                int count1 = 0, count2 = 0, count3 = 0, count4 = 0, count5 = 0, count6 = 0;

                gazePointDataStream.GazePoint((x, y, ts) =>
                {

                    if (x > 450 && x < 550 && y > 70 && y < 200) //opt1
                    {
                        count1++;
                        count2 = 0;
                        count3 = 0;
                        count4 = 0;
                        count5 = 0;
                        count6 = 0;

                    }
                    else if (x > 1000 && x < 1200 && y > 70 && y < 200) //opt 2
                    {
                        count1 = 0;
                        count2++;
                        count3 = 0;
                        count4 = 0;
                        count5 = 0;
                        count6 = 0;
                    }
                    else if (x > 450 && x < 550 && y > 250 && y < 480) //opt3
                    {
                        count1 = 0;
                        count2 = 0;
                        count3++;
                        count4 = 0;
                        count5 = 0;
                        count6 = 0;
                    }
                    else if (x > 1000 && x < 1200 && y > 250 && y < 480) //opt4
                    {
                        count1 = 0;
                        count2 = 0;
                        count3 = 0;
                        count4++;
                        count5 = 0;
                        count6 = 0;
                    }
                    else if (x > 450 && x < 550 && y > 650 && y < 900) //opt5
                    {
                        count1 = 0;
                        count2 = 0;
                        count3 = 0;
                        count4 = 0;
                        count5++;
                        count6 = 0;

                    }
                    else if (x > 1000 && x < 1200 && y > 650 && y < 900) //opt6
                    {
                        count1 = 0;
                        count2 = 0;
                        count3 = 0;
                        count4 = 0;
                        count5 = 0;
                        count6++;
                    }
```

```
                if (count1 > 180)
                {
                    Console.Clear();
                    Console.WriteLine("\n\n\t\t\t\t\t\t\t\t\tYou are going to Location 1\n\n\n\n\n");
                    String location = new String("Location 1");
                    P_Sender.publish(location);
                    count1 = 0;
                }
                else if (count2 > 180)
                {
                    Console.Clear();
                    Console.WriteLine("\n\n\t\t\t\t\t\t\t\t\tYou are going to Location 2\n\n\n\n\n");
                    String location = new String("Location 2");
                    P_Sender.publish(location);
                    count2 = 0;
                }
                else if (count3 > 180)
                {
                    Console.Clear();
                    Console.WriteLine("\n\n\t\t\t\t\t\t\t\t\tYou are going to Location 3\n\n\n\n\n");
                    String location = new String("Location 3");
                    P_Sender.publish(location);
                    count3 = 0;
                }
                else if (count4 > 180)
                {
                    Console.Clear();
                    Console.WriteLine("\n\n\t\t\t\t\t\t\t\t\tYou are going to Location 4\n\n\n\n\n");
                    String location = new String("Location 4");
                    P_Sender.publish(location);
                    count4 = 0;
                }
                else if (count5 > 180)
                {
                    Console.Clear();
                    Console.WriteLine("\n\n\t\t\t\t\t\t\t\t\tYou are going to Location 5\n\n\n\n\n");
                    String location = new String("Location 5");
                    P_Sender.publish(location);
                    count5 = 0;
                }
                else if (count6 > 180)
                {
                    Console.Clear();
                    Console.WriteLine("\n\n\t\t\t\t\t\t\t\t\tYou are going to Location 6\n\n\n\n\n");
                    String location = new String("Location 6");
                    P_Sender.publish(location);
                    count6 = 0;
                }

            });

            Console.ReadKey();
            host.DisableConnection();

        }

        ROS.shutdown();
        ROS.waitForShutdown();
    }
  }
}
```

APPENDIX D: Destination-based Code in Ubuntu Laptop

```
#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>
#include "std_msgs/String.h"
#include <sstream>
#include <iostream>
#include <string>

using namespace std;

ros::Subscriber user_subscriber;

/** function declarations **/
void user_inputCallback(const std_msgs::String::ConstPtr& msg);
bool moveToGoal(double xGoal, double yGoal);

double xTL = 1.5026;
double yTL = 1.5874;
double xTR = 1.5417 ;
double yTR = -1.2260;
double xBL = -1.8040 ;
double yBL = 1.2683;
double xBR = -1.7369;
double yBR = -1.2630;
double xML = 0.0795;
double yML = 2.1336;
double xMR = 0.0473;
double yMR = -2.1033;

bool goalReached = false;

void user_inputCallback(const std_msgs::String::ConstPtr& msg)
{
    std::string a_input=msg->data.c_str();
    std::string choice ="0";
    if (a_input.compare("Location 1")==0)
    {
       goalReached = moveToGoal(xTL, yTL);
     }
     else if(a_input.compare("Location 2")==0)
    {
       goalReached = moveToGoal(xTR, yTR);
    }
    else if(a_input.compare("Location 3")==0)
   {
       goalReached = moveToGoal(xBL, yBL);
    }
    else if(a_input.compare("Location 4")==0)
   {
      goalReached = moveToGoal(xBR, yBR);
    }
   else if(a_input.compare("Location 5")==0)
  {
      goalReached = moveToGoal(xML, yML);
   }
   else if(a_input.compare("Location 6")==0)
   {
      goalReached = moveToGoal(xMR, yMR);
   }

   if (goalReached)
   {
      ROS_INFO("Congratulations!");
      ros::spinOnce();
    }else
   {
       ROS_INFO("Hard Luck!");
    }
}
```

```
int main(int argc, char** argv)
{
    ros::init(argc, argv, "map_navigation_node");
    ros::NodeHandle n;
    user_subscriber= n.subscribe("eye_direction",1000, user_inputCallback);
    ros::spin();
    return 0;
}

bool moveToGoal(double xGoal, double yGoal)
{
   //define a client for to send goal requests to the move_base server through a SimpleActionClient
   actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> ac("move_base", true);

   //wait for the action server to come up

    while(!ac.waitForServer(ros::Duration(5.0)))
    {
       ROS_INFO("Waiting for the move_base action server to come up");
    }

      move_base_msgs::MoveBaseGoal goal;

     //set up the frame parameters
     goal.target_pose.header.frame_id = "map";
     goal.target_pose.header.stamp = ros::Time::now();

     /* moving towards the goal*/

     goal.target_pose.pose.position.x = xGoal;
     goal.target_pose.pose.position.y = yGoal;
     goal.target_pose.pose.position.z = 0.0;
     goal.target_pose.pose.orientation.x = 0.0;
     goal.target_pose.pose.orientation.y = 0.0;
     goal.target_pose.pose.orientation.z = 0.0;
     goal.target_pose.pose.orientation.w = 1.0;

     ROS_INFO("Sending goal location ...");
     ac.sendGoal(goal);

     ac.waitForResult();

    if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
    {
       ROS_INFO("You have reached the destination");
       return true;
    }
    else
    {
       ROS_INFO("The robot failed to reach the destination");
       return false;
     }

}
```