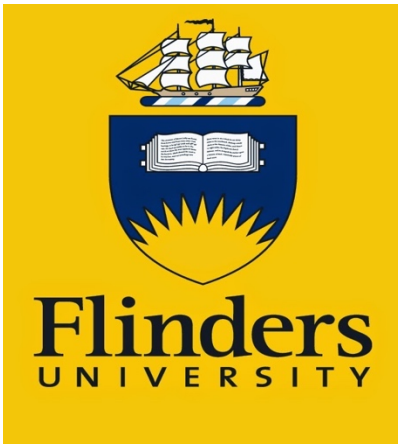


AudioVisual (Brain) Computer Controlled(ABC) Wheelchair

By

RajKunwar Singh Kukreja

A thesis submitted to the College of Science and Engineering in partial fulfilment for the requirements for the degree of **Master of Engineering (Electronics)** at Flinders University Adelaide, South Australia

Supervisor: Dr Nasser Asgari

October 2018

DECLARATION

I certify that the work presented in this thesis to the best of my knowledge and belief does not include material that has previously been submitted for a degree or diploma at any university; and does not contain work previously published or written by another person except where acknowledged in the text.

Signed: RajKunwar Singh Kukreja

Date: 29/11/2018

ACKNOWLEDGEMENTS

I would like to express my appreciation and gratitude to my supervisor and mentor Dr Nasser Asgari for the opportunity to work on the project and for all the support and guidance provided throughout the course of study at Flinders University.

I would like to give a special thanks to Mr Jonathan Wheere who introduced me to ROS and has constantly provided assistance on various aspects in the field of robotics.

And finally, I would like to thank my family for their support.

ABSTRACT

Advancements in technology and medical devices have been providing solutions such as electric and powered wheelchairs for individuals with impairments and physical disabilities to cope with their disabilities; however, there are certain individuals in the disabled community with a high-level of disability which restricts them from using these mobility aids.

This thesis aims to contribute to the ABC Wheelchair project at Flinders University by developing a cost-effective Autonomous Wheelchair which can be controlled using multiple inputs such as speech commands, hand gestures, or facial gestures by converting an existing electric wheelchair to an intelligent robotic system with easily accessible sensors. The study describes various methodology used for designing and developing an intelligent and autonomous wheelchair using Robot Operating System. It can also act as a base for converting the majority of the existing electric wheelchairs into autonomous ones by introducing multiple control inputs as well as generating a map for the wheelchair environment and navigating inside it.

The system was implemented on an electric wheelchair with the new capability of generating 2D and 3D maps of the environment and navigating within the generated map using low-cost sensors. The tests provided convincing results for the developed system to act as a mobility aid not only for individuals with cognitive and certain physical impairments, but also as an accessory for converting existing wheelchairs into autonomous ones.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Recent years have shown an increase in the usage of mobility aids such as electric wheelchairs and electric scooters by individuals with certain disabilities; however, there are still a certain number of individuals within this community with furthermore disabilities restricting them from access to these mobility aids. According to the World Health Organization [22] there are over 1 billion people in the world who face some sort of disability with limited access to any health care, employment or education. A survey on administering the physical and mental wellbeing, functional independence and lifestyle of individuals with disabilities facing financial and technological limitations during a 12-month period [23] showed significant improvement in overall health and lifestyle after these individuals were equipped with a depot style wheelchair. Another survey [24] on the usage of powered wheelchairs for adults concluded an increased independence and improved quality of life for the users but also mentioned the forthcomings of accidents resulting in personal injury to the user or damage to their device.

The ABC Wheelchair aims to provide a cost effective and a safe solution for individuals using a combination of brain signals, gaze tracking or eye-tracking and speech commands as inputs to a system that performs various pre-defined task based on the inputs received. The ultimate goal of the ABC wheelchair project is to integrate brain signals with a graphical user interface that allows a user to interact with the system and the wheelchair can drive itself to perform a task while performing obstacle avoiding for safety of the user and completing the tasks defined by the user. The concept of the ABC wheelchair project aims at integrating and connecting all electronics devices (such as lifts, door actuators, coffee makers, water kettles, and other devices used in most daily tasks) through Internet of Things (IOT).

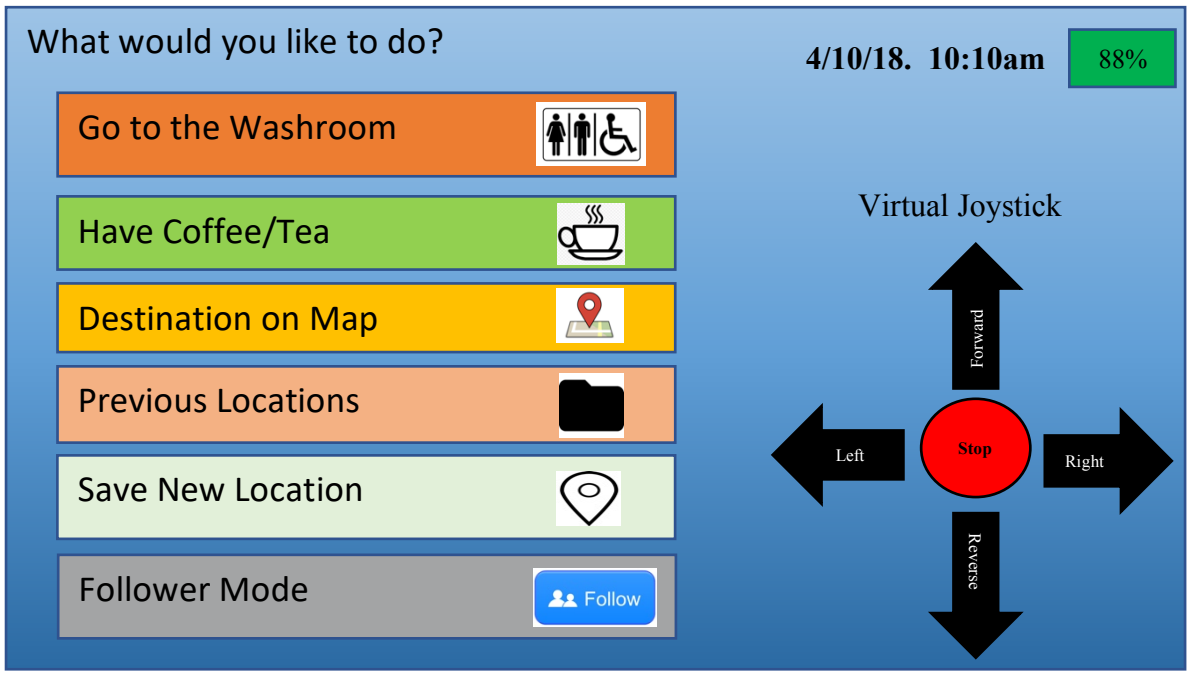


Figure 1. ABC wheelchair concept

The ABC wheelchair concept as shown in figure 1. allows the user to choose from multiple configured options using different inputs such as eye-tracking which allows the user to control a computer's cursor using their eyes, a voice command recognition system or a touch based screen allowing the user to tap on the screen for selecting the options. The ultimate goal will be able to control the interface using brain signals through an electroencephalogram (EEG) headset. A scenario can be considered where the user would like to get a cup of coffee and after the user chooses the option, the system generates a path to travel to the cafeteria from its current position as shown in figure 2. While the wheelchair is travelling to its destination, based on the concept the coffee maker will be connected through IOT and thus automates the task of travelling and preparing the coffee once the user reaches the destination. If the path to be taken involves traveling to different floors by the means of an elevator then by using position estimation techniques an elevator will be called to the specific floor replicating the act of pressing a button for the elevator once the wheelchair reaches within a certain distance from the elevator.

Figure 2. ABC Wheelchair concept for path generation from current position to destination

The entire project has been broken down into multiple parts and the goal for this study is to develop a system or a kit that can be installed on a new or an existing powered wheelchair that creates a map of its environment and navigates within the map autonomously while providing a safety measure during the autonomous drive.

Chapter 2

BACKGROUND

2.1 Related Work

Smart wheelchairs (**SW**) have been a topic of research since the 1980's starting [54] with the Navchair, Wheelesley, SENARIO and VAHM to develop an intelligent system to implement a navigational setup based on inputs from a user. Powered wheelchairs (**PW**) were introduced in 1953 by George Klein for helping people injured during the world war II and was quickly moved to mass production in the year 1956 by the American wheelchair company and Everest & Jennings. The earlier PW consisted of a basic structure which included a Chassis, a Controller, a seating system and a battery and in recent years there has been a more adaptive and a collaborative control approach to provide a much more safe and reliable product. Smart wheelchairs during the early years of development mainly consisted[53] of technology developed for mobile robots with an attached seating system or a PW with a computer and a few sensors attached to it. In the past decade there has been an enormous increase in the research and development for smart wheelchairs with

Figure 3. Smart Wheelchair projects[54]

39 institutions all around the world worked on developing a prototype showing the interests and the need for more research on the subject.

Figure 4. Modern SW form factors[54]

(Parikh et al. 2007) created a simple method of extracting user profiles to select the best trigger as the input for the user. These triggers (table 1) replicate the use of the controller, a joystick in most cases and the sensors attached provide information for the system to understand its environment.

Input Methods
Brain Computer Interface (BCI)
Voice or Speech
Touch
Controller
Gesture
Computer Vision
Multimodal

Table 1. User inputs

SW's are divided into two parts:

- **Semi-Autonomous Wheelchair (SASW):**

  The SW implements only a certain or a specific task to provide assistance such as an obstacle avoidance system when the user is manually controlling the SW.

- **Autonomous (ASW):**

  These SW's execute the entire task of path planning, obstacle avoidance and control using only an input from the user.

Researchers have developed various semi-autonomous systems [4] [9] are two systems to create a **following mode** where [4] is a system which follows their companion using a laser range finder (LSR) which guides the user's footprint using an extend Kalman filter for estimating the path followed whereas [9] uses daily

routine and behaviour to implement a follower without having any prior knowledge of the goal.

A **SASW** can be classified as an **ASW** if it implements any one or a combination of the following tasks:

- **Localization** – Determining its position in a known environment.
- **Navigation** – Path planning for navigating within the map generated.
- **Following** – The ability to follow an object or a person reducing the need for manual control of the PW.

The Singapore – MIT Alliance for Research and Technology (**SMART**) in 2016 deployed[57] a SW at the Singapore's Changi Hospital which navigates throughout the building using a map created with 3 lidars attached on it becoming the first SW project used in public. The SW uses a localization algorithm to determine its position on the map previously generated and the mechanical structure of the SW has been designed to enable it to make sharp or tight turns and as well as fit through normal-sized doorframes. An app-based online booking and a scheduling algorithm allows users to schedule rides on the SW.

A second SW to make public appearance[57] was **Whill Model M** by Panasonic and Whill at the Haneda Airport in Japan. Whill model M uses two lidars to detect obstacles and based on a previously prepared map the user can select the destination through a smartphone app and the onboard computer plans the best route to take due to which it gets the name Uber of wheelchairs. An additional capability is to sync with nearby wheelchairs and travel in a column formation and then can return to its home base automatically thus reducing the need of human labour to collect these wheelchairs. Other projects dealing with smart robots such as the iBot were not commercialized due to the high price attached to them.

University of Toronto and Cyberworks Robotics (Toronto) **[**10**]** by applying the same principles and similar sensors used in self-driving cars to wheelchairs have developed a product costing between a range of US$300 - $700 as compared to the earlier upward costs of $30,000 to make autonomous wheelchairs however has certain limitations of currently intended for working only indoors due to struggles working in full sunlight. The project started in 2015 to help users with upper-body disabilities which restricted their movements like ALS, spinal cord injuries or hand tremors resorting to eye-tracking technology or sip and puff devices to control the PW and *"by enabling autonomous navigation it could dramatically enhance the user's quality of life."*

Brain Computer Interface (BCI) provides a direct interface between the human brain and a machine/computer using invasive and non-invasive techniques. Studies on the subject have provided promising results in the field of Electroencephalography (EEG)[10] where users who undergo a training period have been able to execute external control of a wheelchair by imagining that they were moving a part of their body. EEG offers a promising solution for helping certain individuals with disabilities where BCI headsets available nowadays measure various muscle activities and brain signals that can be used as external triggers. [11] [12] are systems integrated with controlling a wheelchair using a BCI headset. The systems incorporate muscle activity and users' concentration measured through electrodes placed on the EEG headset. Researchers at Federal Institute of Technology , Lausanne [10] also worked on developing a robotic wheelchair with shared control between the user and the wheelchair allowing the user to manoeuvre using just their thoughts and the systems continues the previous command rather than the user controlling the system continuously and using two web-cameras placed on the front to detect any obstacles and provide a safer system.

2.2 ABC Wheelchair project at Flinders University

The ABC wheelchair project at Flinders University has been previously worked on by students who have successfully achieved integration with the motor controller board on the wheelchair and have been able to replicate the values from the joystick controller enabling a microcontroller to act as a bridge between the joystick controller and the controller board. As mentioned in the introduction section, the ABC wheelchair project is divided into various segments and the studies by students have been able to provide promising results and contribute highly to this project. One of the recent study [56] has successfully been able to control the wheelchair using facial and head gestures. The system uses real-time processing of a user's face to recognize various gestures (such as head tilting, opening mouth, raising eyebrows, lip sucking etc.) and calculates a threshold to differentiate between intentional and non-intentional gestures for reducing false-positive outputs from the system.

Figure 5. Existing System

2.2.2 Wheelchair model used

Figure 6. Sterling ruby by sunrise medical [34]

Parameter	Value	Unit
Dimension (LxWxH)	610x482x889	mm
Weight	110	lbs
Max Speed	6	Km/h
Tire Size	203.3	mm
Range	15	km
Kinematic Model	Differential-Drive	N/A

Table 2. Wheelchair technical specifications

2.2.3 Inductive Joysticks:

Inductive joysticks use sets of copper wires laid in a circular format that induces a magnetic field when current is applied. When the joystick (metal shaft) is moved, it cuts through the magnetic lines causing a change in the flow of current in the coil which is proportional to the output voltage from the joystick.

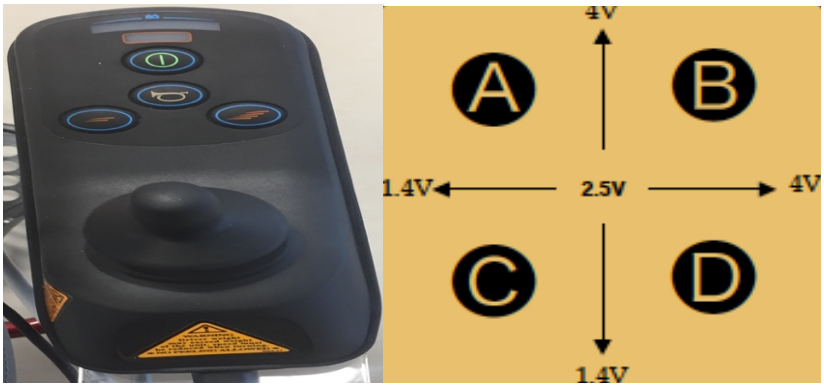


Figure 7. Joystick on wheelchair and voltage output range from joystick

2.2.4 Interfacing with the Motor controller board:

The microcontroller can be interfaced with the motor controller using an interface board developed by [55] which consists of a four channel Analog to Digital Convertor (ADC) **MCP3004** that reads voltage from the joystick and a two channel digital potentiometer (POT) **MCP42010** to output voltage to the motor controller. This bridge created between the joystick and the motor controller helps in replicating the joystick values through a microcontroller using SPI communication.

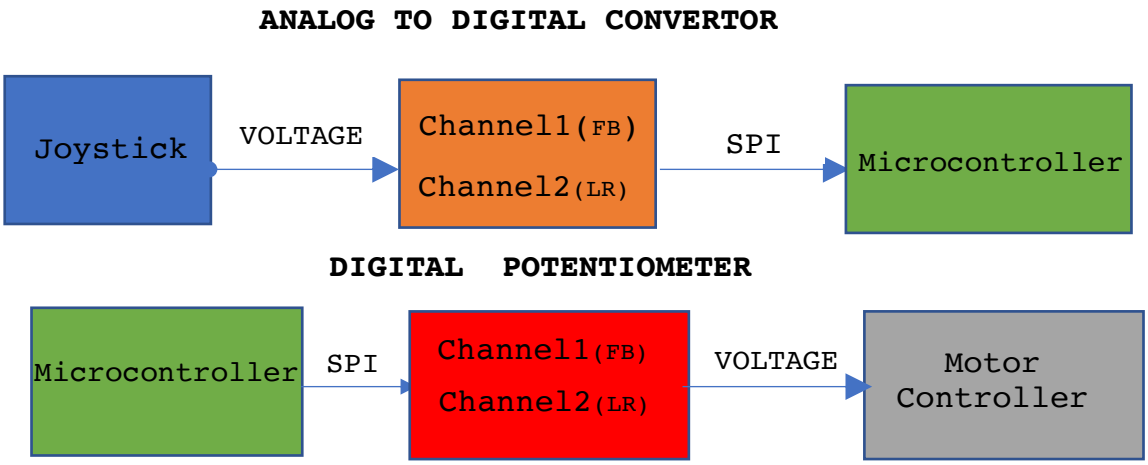


Figure 8. Joystick interface board

2.2.5 Existing sensors

The wheelchair also included a sonar sensor **(MaxSonar MB1013)** and Sharp IR sensors placed on the wheelchair tray and 6 IR sensors placed on the base rod each aligned at an angle of 60° apart for a complete field of view as shown in figure 9.

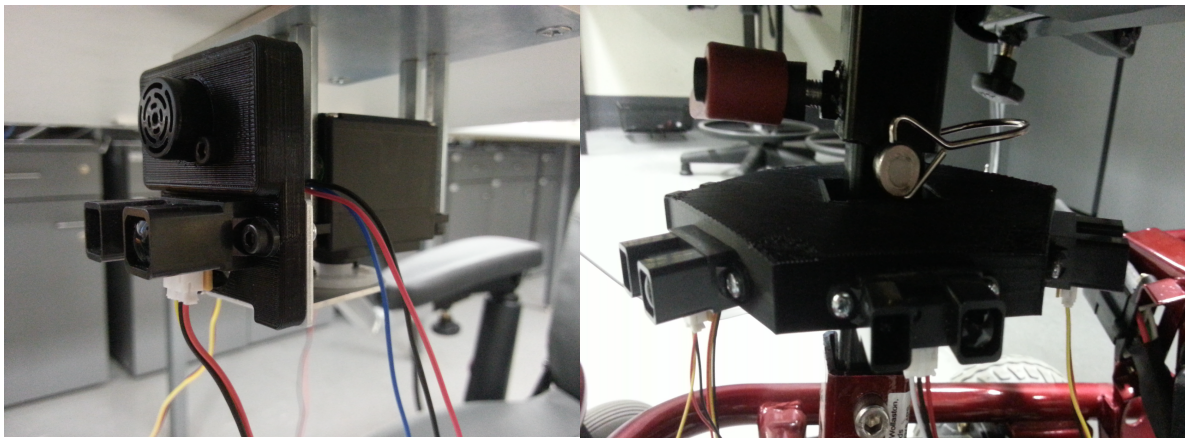


Figure 9. Existing sensors on wheelchair

MaxSonar MB1013 – Ultrasonic Distance measurement sensor

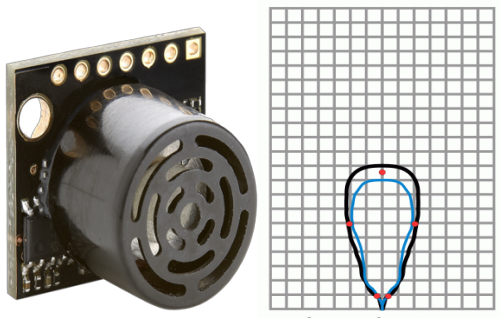


Figure 10. MB1013 and beam pattern

Parameter	Value	Unit
Resolution	1	mm
Sampling Rate	10	Hz
Output	Analog/TTL Serial/PW/RS232	N/A
Minimum Range	300	mm
Maximum Range	5000	mm
Beam Angle	±17	° (deg)
Operating Range	2.5 – 5.5	V

Table 3. MB1013 specifications

SHARP GP2Y0A02YK0F – IR Distance sensor

Figure 11. IR distance sensor

Parameter	Value	Unit
Resolution	1	mm
Sampling Rate	10	Hz
Output	Analog voltage	N/A
Minimum Range	200	mm
Maximum Range	1500	mm
Operating Range	4.5 – 5.5	V

Table 4. IR sensor specifications

2.2.6 Kinematic model

The wheelchair features a combination of 2 driven wheels and 2 castor wheels attached to the front for stability. A **differential drive robot** can be defined as a drive system where the robot's movement depend on independent actuators for each wheel as shown in the figure below.

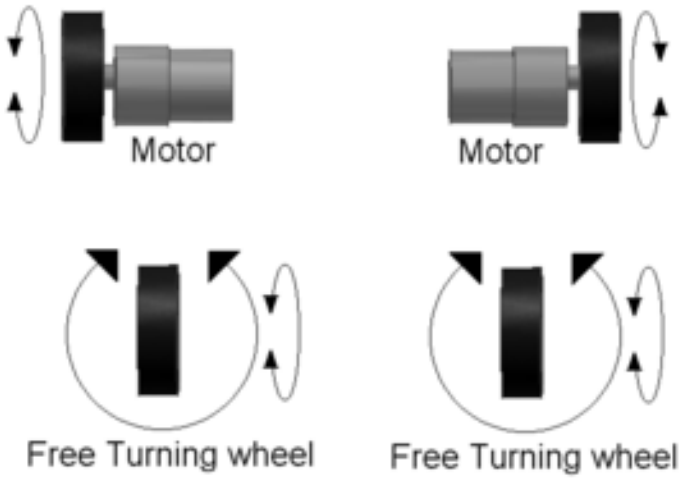


Figure 12. Robot kinematic model [41]

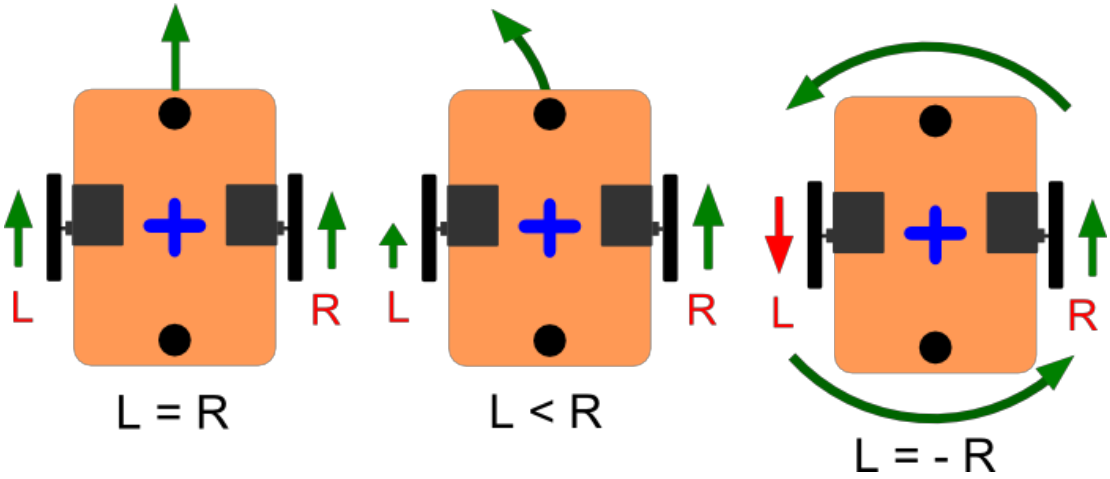


Figure 13. Generating motion for differential drive robot

Figure 13 describes how various motions are generated for a differential drive robot. Since the drive wheels are fixed and independent of each other the speed of the wheels is varied to steer the wheelchair or to generate the desired motion.

Chapter 3

THEORY AND METHODOLOGY

This chapter discusses the theory behind indoor autonomous mobile robots and the background information required for developing one. The section contains information on one of the major components of this study ROS, which is an Open Source platform for designing robots. It describes the process of establishing a communication between ROS and a microcontroller for controlling the wheelchair.

3.1 Autonomous Mobile Robot:

An autonomous robot is a robot that can perform various behavioral tasks by making decisions with a high level of autonomy. An autonomous robot should be able to gather information regarding the environment it is in without the need for any human intervention. It should avoid situations that can cause harm to people or damage to itself. The process of developing an autonomous robot can be divided into three main branches:

1. **Mapping** – Generating a map or modeling of the environment.
2. **Localization** – Finding the robot's position inside the environment.
3. **Navigation/Path Planning** – Navigating within the environment.

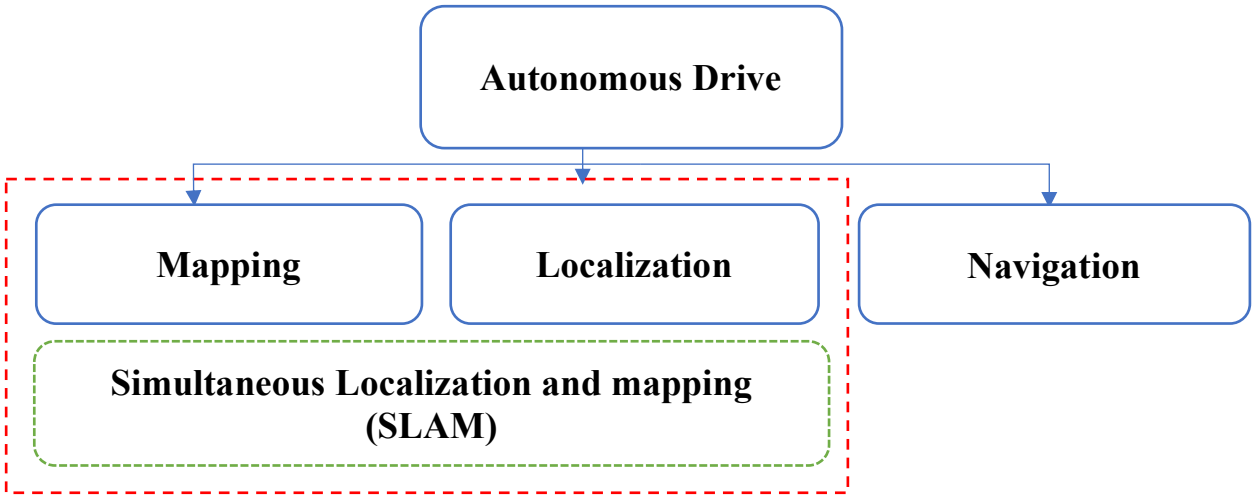


Figure 14. Autonomous Robot block diagram

3.1.1 Mapping:

Since GPS cannot provide reliable information within buildings or when indoors, a mobile robot needs to interpret its environment in order for it to execute navigational task. ***Robotic mapping*** can be defined as the branch of autonomous robots that deals with the construction of the map or a floor plan. The map is built using information from different type of sensors that can interpret the environment that the robot is currently in. A robot contains two types of sources for information:

1. **Active sensors** (Emit energy and probe the environment based on self-generated energy) for example LIDAR, distance sensors, Microsoft Kinect
2. **Passive sensors** (Do not emit energy and wait for a response from the environment) for example digital cameras and IMU (Inertial Measurement Unit).

These sources help solve problems for the robot such as interpreting the environment and helping it to determine its location and generate maps based on the responses from the sensors.

Map representation - Maps can be represented in two ways:

- **Geometric or Grid based representation –** These are the most common types of maps used for representation for humans as it considers a two-dimensional space in which the objects are placed with a precise coordinate. Grid maps break down the world into cells which make the map look like a building floor plan when mapping indoors. The grid is considered to be static where the cells are independent of each other and each cell is inferred to be either free or occupied based on the sensor readings.

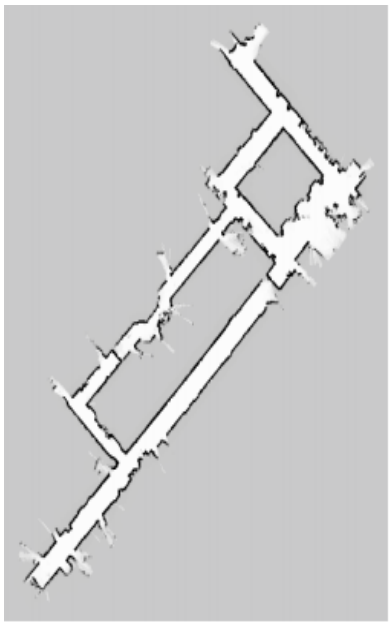


Figure 15. Grid map representation [46]

The probability for occupancy can be defined as:

$$Cell\ Occupied \rightarrow p(m_i) = 1$$
$$Cell\ Not\ Occupied \rightarrow p(m_i) = 0$$
$$Unknown \rightarrow p(m_i) = 0.5$$

- **Topological or Landmark based representation –** The framework remembers landmarks and relations between them. The distance between these landmarks are stored and the map can be considered a graph where the nodes correspond to landmarks and the arcs correspond to various paths.

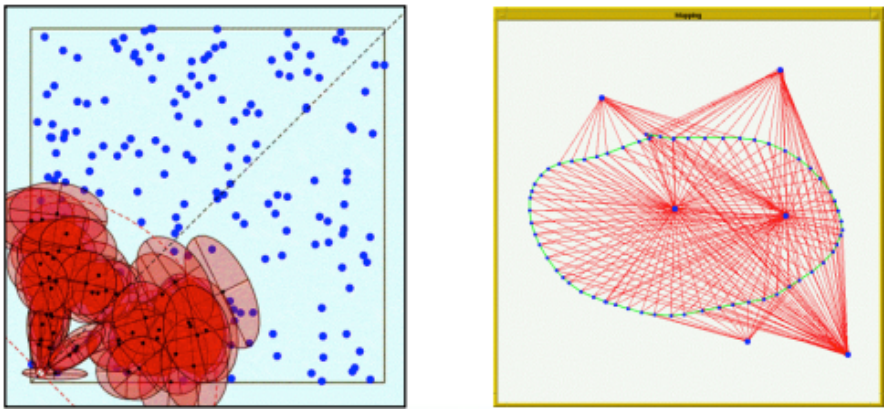


Figure 16. Feature-based or landmark based map [47]

3.1.2 Simultaneous Localization and Mapping (SLAM)

SLAM in the field of robotics can be defined[46] as the computational problem of building a map of the environment the mobile robot is in while simultaneously keeping track of the location of the robot within the map for navigation. SLAM is not a single algorithm but rather a combination of multiple processes running at the same time for solving the problem. It has applications in areas where the environment is unknown for both manned and autonomous vehicles such as:

- Indoors
- Undersea
- Space
- Underground

There are various algorithms for solving the SLAM problem including particle filter, extended Kalman filter and graphSLAM which are discussed in the next sections used in this study for developing an autonomous wheelchair. The wheelchair is a mobile robot and will mostly be used indoors requires a system that can navigate the wheelchair safely while developing a map of the environment simultaneously.

List of a few SLAM techniques available:

1. EKF SLAM
2. **Fast SLAM**
3. **Graph SLAM**
4. Occupancy-Grid SLAM
5. DP-SLAM
6. Parallel Tracking and Mapping (PTAM)
7. Mono-SLAM
8. ORB-SLAM
9. Co-SLAM
10. SeqSLAM
11. **Visual Slam (vSLAM)**

Maps created using SLAM[49] enable a quicker and adaptive response as compared to pre-programmed routes. By using a combination of sensors on a robot such as Lidar, camera, ultrasonic sensors are able to better interpret its environment and effectively improve navigation and obstacle avoiding ability due to the adaptiveness that it enables.

Fast SLAM

A feature based SLAM using Rao-Blackwellization technique of using particle filters as a tool for solving the SLAM problem[19] by applying particle filters where each particle carries an individual map of the environment.

Particle Filter algorithm:

1. Sample the next particle based on the proposal distribution
$$x_t^i \sim proposal(x_t | \dots)$$
2. Calculate importance weights
$$w_t^i = \frac{target(x_t^i)}{proposal(x_t^i)}$$
3. Resample to replace unlikely samples with more likely ones

SLAM posterior

$$p(x_{0:t}, l_{1:M} | z_{1:t}, u_{1:t}) = p(x_{0:t} | z_{1:t}, u_{1:t})\, p(l_{1:M} | x_{0:t}, z_{1:t})$$

Movements

poses Map observations th posterior Map posterior

The robot's path is a sample-based representation where each sample is a path hypothesis. This removes the need for maintaining past poses. FastSLAM uses a combination of particle filters and EKF for calculating the SLAM posterior where the landmarks are conditionally independent and are solved as a 2-dimensional EKF.

Figure 17. FastSLAM

Graph SLAM

Graph-based SLAM[39] uses a graph to represent the problem of localization and mapping where each node in the graph corresponds to a pose of the robot and a sensor measurement while mapping and the edge between the nodes corresponds to the spatial constraints between them. These nodes create a graph and the most likely map is determined by moving the nodes till a map is rendered based on the known poses once a **loop closure** is detected.

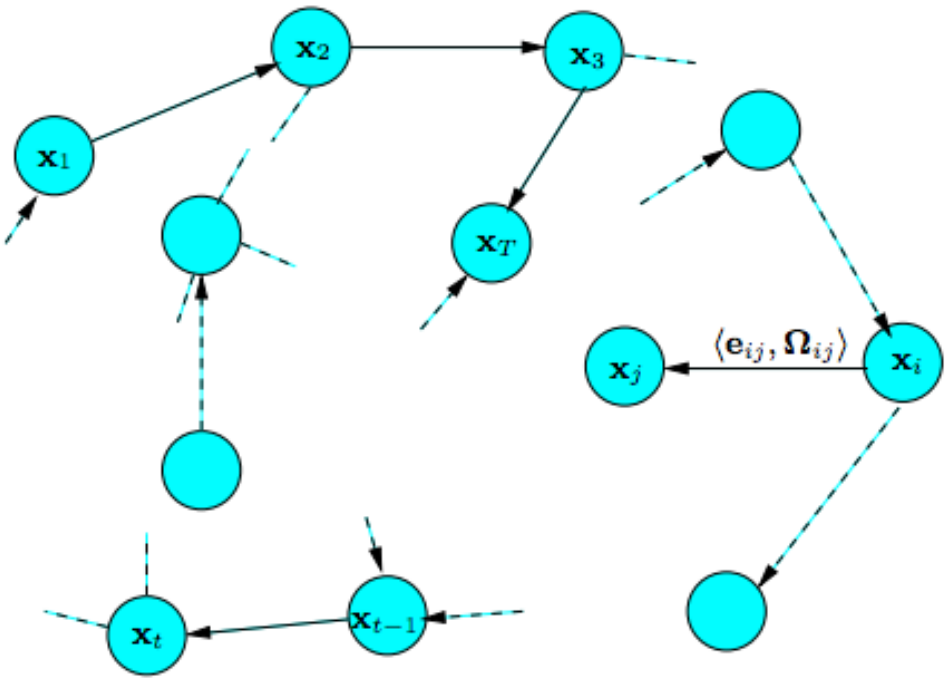


Figure 18. Pose-graph representation of nodes [39]

Loop closure

Loop closure [16] is the problem of recognizing a previously-visited location and updating beliefs accordingly. Typical loop closure methods apply a second algorithm to compute some type of sensor measure similarity and re-set the location priors when a match is detected.

Visual based SLAM

Visual SLAM can be considered as a type of SLAM system which leverages 3D vision to perform localization and mapping functions when the environment and the location of the robot is unknown. The idea behind most vSLAM systems is to track a set of points through successive camera frames which helps to triangulate their 3D position in real time while approximating the robot's pose[15]. Vision sensors[49] are attractive for employment in SLAM systems because of the richness in features detected, accessibility and cost-effectiveness on these sensors. There are a few state-of-the-art visual SLAM techniques which have proven to be effective even during the presence of significant noise in the robot's position and the landmark position sensing.

Different types of Visual based SLAM systems available [21]**:**
(* - Available on ROS)
Monocular cameras

- PTAM
- DSO*****
- LSD-SLAM*****
- ORB-SLAM*****
- SVO-SLAM*****

RGB-D cameras

- OpenCV RGBD-Odometry (Visual Odometry based RGB-D images)
- Dense Visual SLAM for RGB-D Cameras*****
- RTAB MAP - Real-Time Appearance-Based Mapping*****
- ORB2-SLAM*****
- InfiniTAM∞ v2
- Kintinuous
- ElasticFusion
- Co-Fusion

RGB-D camera and Lidar

- Google's Cartographer*****

3.1.3 Navigation and Path planning

Since GPS is not available indoors a mobile robot by using computer vision algorithms or sensors such as laser range finders, sonar sensors or photometric cameras allows the system to extract features from the surrounding environment required for the robot to localize itself. There are different ways of implementing indoor navigation on a mobile robot such as line following, placing beacons, markers or bar codes in the environment and position estimation using odometry. The robot requires a map within which it can navigate. Once a map is available, the robot localizes itself within the map where the location of landmarks or position of points is defined with respect to a relative initial position. After localizing a path can be generated for the robot to reach when given a goal to travel through traversable regions. The map generated is assumed to be static in the terms that the map does not change; however, since the wheelchair is a mobile robot[32] the environment is dynamic and is continuously changing. Obstacles might appear and disappear from the measurements of the sensor which creates a noisy estimate of the surrounding. This results in the need for integrating the system with components that keep track and compensate it according to these changes. There are two changes which have to be continuously tracked:

1. Static and Dynamic Obstacles
2. Map update through new sensor readings

Figure 19 shows illustrates a map with sample cost of the grid cells. A planner can then generate a path from the current position to the goal position by traversing through the cells depending on the type of algorithm chosen.

Figure 19. Costmap of grid cells [47]

3.2 Sensors

In addition to the sensors available on the wheelchair, there are various sensors required for the task of developing an autonomous mobile robot. The mobile robot should be able to generate a map of the environment using easily accessible and low-cost sensors such as a sonar sensor or a RGB-D camera which can generate pointclouds. To estimate the position of the robot for localization the motion of the robot should be tracked which can be done through the use of encoders that keep track of the wheel rotations and an IMU unit which when attached to the robot body provides the velocity and the acceleration of the robot as it moves.

3.2.1 Rotary Encoders:

One of the important aspects in the field of autonomous robots is for the robot to estimate its position relative to a known or starting position using motion sensors such as IMU or encoders. Rotary encoders provide digital pulses for the motion of the shaft which can be used to determine the displacement of a wheel.

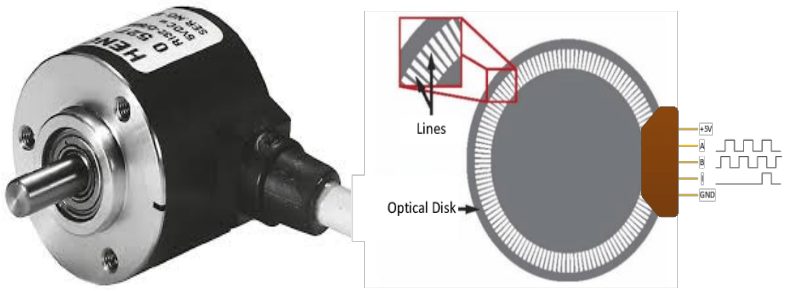


Figure 20. Rotary encoder

By using the process of odometry which is a form of dead-reckoning the encoders can be used to track the change in position over time which estimates the actual position of the wheelchair as shown in the figure 20.

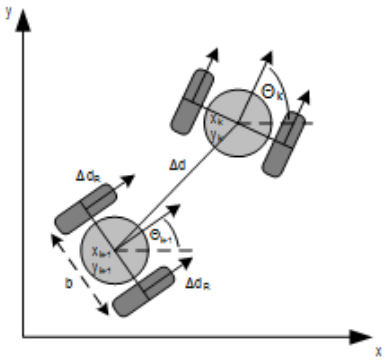


Figure 21. Odometry [42]

The wheelchair is configured with 2 motors that control the wheels offering limited access to the motor shaft. Due to this the encoders are placed externally using a wheel on wheel configuration. Multiple positions tried for placing the encoders faced slippage due to a non-firm contact with the wheel which results in missing of pulses causing a noisy estimate of the position. A new bracket designed so the encoder wheel stays in contact with the middle of the wheelchair wheel. The bracket includes a cross shaped wedge which allows movement for the encoder and a firm contact with the wheelchair wheel allowing adjustment and also for ease of replacement.

Figure 22. Wheel on wheel configuration for encoders

3.2.2 RGB-D camera (Microsoft Kinect for Windows)

RGB-D cameras provide a colour (RGB) image along with the corresponding depth image. The depth image is a per-pixel estimate which relates to the distance between the image plane and the object in the RGB image.

Figure 23. Microsoft Kinect [43]

Parameter	Value	Unit
Colour	640 x 480	@ 30fps
Depth	320 x 240	@ 30fps
Sensor	Structured light	Light coding
Range	0.8 – 4.0	m
Horizontal View	57	degree
Vertical View	43	degree

Table 5. Microsoft Kinect specifications

The Kinect sensor[29] uses an infrared (IR) laser for generating a pseudo-random beam pattern and an IR camera then captures the image of dots reflected from the objects in the environment. By the use of structured light technique, the distortion of the dot points is calculated which correspond to the distance of the dot with respect to the RGB pixel. The Kinect was initially used on the Microsoft X-box for 3D perception of human motion and through reverse engineering, it has been used in robotic applications for indoor navigation. The Kinect sensor [31] had extremely effective results for indoor navigation in robotics application indoors however showed interference from sunlight during outdoor applications. The Kinect uses IR, so it does not detect glass or transparent objects and thus, sensors such as sonars have to be used for detecting obstacles. Since the sensor is cost effective and easily accessible it provides a great advantage over other depth cameras or laser sensors which induce an enormous cost to the project budget. Another idea[31] to use multiple Kinect sensors for stereovision might be applied to enhance the 3D perception and pointclouds which can help improve the functionality outdoors.

3.2.3 Inertial Measurement Unit (IMU):

An IMU is an electronic device that can measure the acceleration, rotational velocity and orientation through a mix of Accelerometers, gyroscopes and magnetometers. The IMU used for this project is the **MPU 9250** by Invense which features a 3-axis gyroscope, 3-axis accelerometer and a 3-axis magnetometer. The IMU provides linear acceleration and rotational velocities, and by using the process of dead-reckoning the wheelchairs position can be estimated.

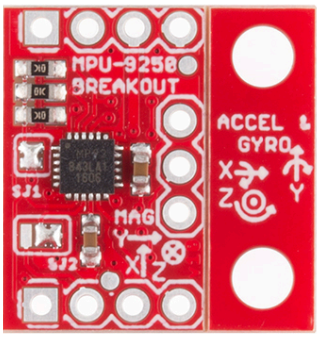


Figure 24. IMU MPU9250 [40]

Reading raw data from accelerometer:

The output from the accelerometer can be configured to work with a programmable full-scale range of ±2g, ±4g, ±8g and ±16g where ($1g = 9.81 m/s^2$, acceleration due to gravity).

$$16 - bit\ output\ with\ a\ \pm 2g\ scale.$$
$$16 - bit\ output = 65535\ bits$$
$$\pm 2g\ scale = 4g\ or\ 4000mg$$
$$1 - bit = 4000/65535 = 0.061mg$$

Converting to m/s^2,

$$Linear_{Accleration} = Raw_{data} * \frac{0.061 * 9.81}{1000} \qquad \text{(Eqn 4)}$$

Reading raw data from gyroscope:

Output from the gyroscope can be configured to work with a programmable full-scale range of ±250°/sec, ±500°/sec, ±1000°/sec and ±2000°/sec

$Sensitivity$: $16.4\ °/sec$

$$16 - bit\ output\ with\ a\ \pm 2000°/\text{sec}\ scale.$$
$$16 - bit\ output = 65535\ bits$$
$$\pm 2000°/\text{sec} = 4000°/\text{sec}$$
$$1 - bit = 4000/65535 = 0.061°/\text{sec}$$

Converting to rad/sec

$$Angular_{Velocity} = Raw_{data} * 0.061 * \frac{pi}{180} \qquad \text{(Eqn 5)}$$

3.3 Robot Operating System (ROS)

3.3.1 What is ROS

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms." ROS [2] was built to promote collaborative software development and provide support for re-usage of code in robotics research and development. It is a distributed framework of processes (called nodes) which enable the executables to be designed individually.

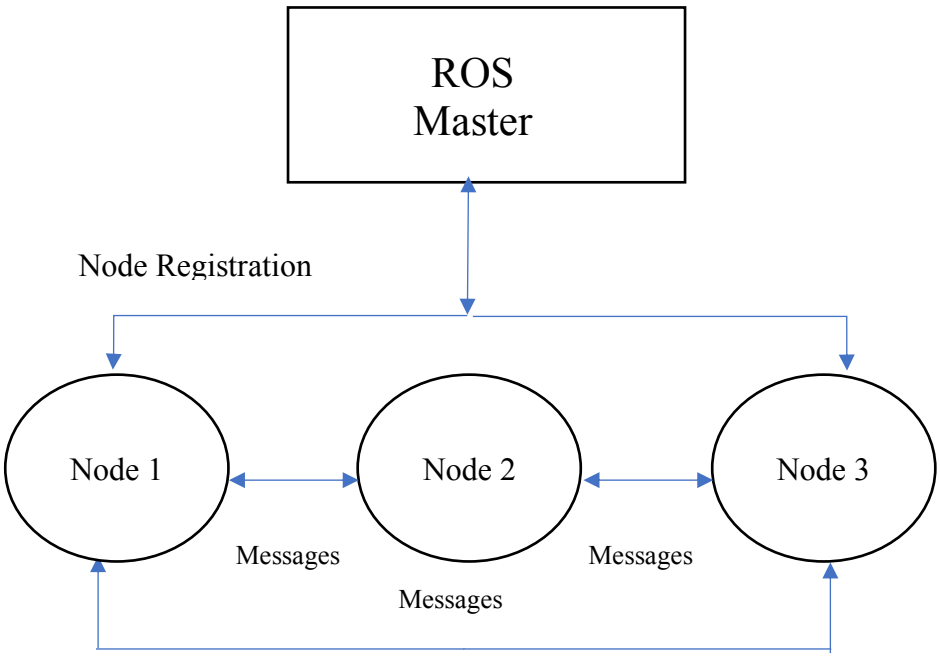


Figure 25. ROS communication block diagram

Communication is ROS takes place through nodes where each node can publish(send) and subscribe(receive) data to and from a topic. Information such as odometry, proximity information from sensors, velocity of the wheelchair can be considered as examples of topics.

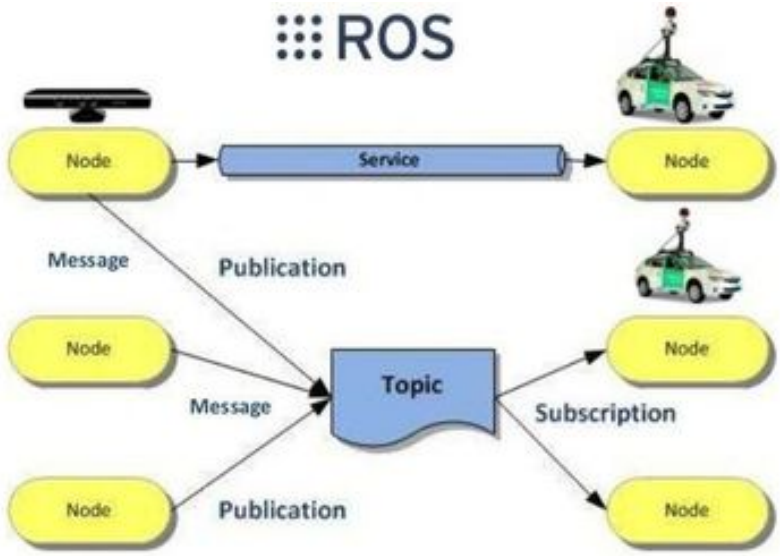


Figure 26. ROS Communication between nodes [44]

Packages in ROS:

Packages is the way the software is organized in ROS which can contain ROS-nodes, libraries, configuration files, datasets or even third-party piece of software. The packages allow the easy to consume functionality for re-usage of software. A ros package, "**ROSSERIAL-ARDUINO**" allows the Arduino microcontroller to act as a node and the sensor readings can be published through the Arduino and the other nodes on the computer side can subscribe to this information and based on the functionality defined, the system will output velocities for the wheelchair in order to achieve the on-going task.

3.3.2 Setting up the robot using ROS:

ROS contains packages which use working algorithms that have been implemented by users that can also be configured on other custom robots. Since a wheelchair can be considered a mobile robot, most of the algorithms that are used for developing autonomous mobile robots can be applied to it as well. As illustrated in figure 14 the task for developing an autonomous robot is branched into three categories mapping, localization and navigation.

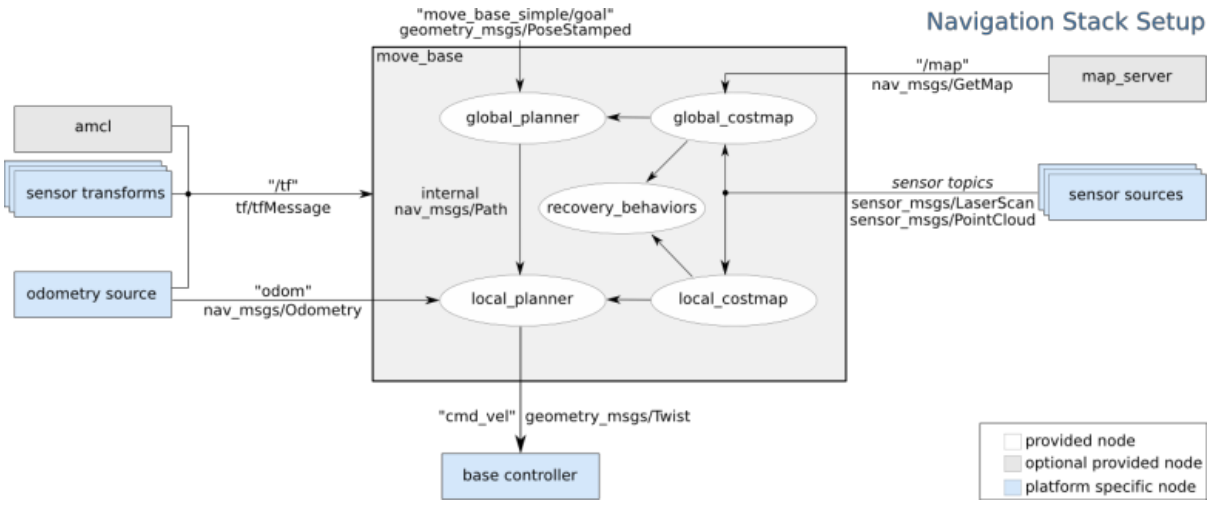


Figure 27. Robot setup [45]

Figure 26. describes the navigation stack on ROS which takes odometry information, sensor measurements and a destination and generates velocity commands for the controller to follow in order to reach the goal.

3.3.3 OpenSLAM's Gmapping

GMapping is a highly efficient Rao-Blackwellized particle filter to learn grid maps from laser data. The approach uses the particle filter by assigning a map to each particle. This leads to a memory extensive complexity and Gmapping offers an adaptive technique which reduces the number of particles in a Rao-Blackwellized particle-filter used for learning grid maps. By accounting for not just the movement but also recent observations, the uncertainty of the robot's pose is reduced significantly by computing an accurate proposal distribution.

ROS package "**gmapping**" is a wrapper for OpenSLAM's Gmapping to provide a laser based SLAM system. A 2-Dimensional occupancy grid map can be generated using the robot's pose (Odometry) and laser measurements. Each robot pose[41] is represented as a particle where the particles are moved according to the information from odometry source and based on how well these laser scans fits the map the robot can be localized.

Gmapping can help generate grid maps for the environment but there are a few drawbacks of using a 2-D map as they can get confusing when the map gets too big and are also not visually appealing to humans. For an autonomous robot to understand the environment, 2-D maps are ideal and can function perfectly using it but for usage with a wheelchair a visual based or a 3-D based map allowing the user for a better visualization of what the map looks like will be more suitable.

3.3.4 Path Planning

Ros package **"move_base"[26]** provides an implementation that can help a mobile robot to reach a given goal in the world. The package links a global and a local planner to accomplish its global navigation task. The global planner uses costmaps to find the minimum cost path from the current position to the destination using the algorithm (Dijkstra's) and generates a series of waypoints which the local planner follows to reach the destination and the local planner uses sensor measurements for updating the costmap and following the path by sending velocity values to the robot controller for traversing through the path.

3.3.5 Costmaps

Obstacles in maps are generated using **Costmaps** which are data structures that represent if it is safe for a robot to be in a grid of cells. The costmaps are created using distance sensor information about obstacles to mark and clear these obstacles at each update. The values represent free spaces or places where robot will be colliding. Ros package **"costmap_2d"**[33]uses sensor data and information from the static map to build a 2D occupancy grid of the data and based the on-sensor measurements and the user specified inflation radius the cost of the grids is inflated. The sensor measurements used for the wheelchair include a pointcloud from the Kinect sensor and a beam from the sonar sensor discussed in section 2.2.5. The range

information and the visualization of the sonar sensor can be viewed in the figure below.

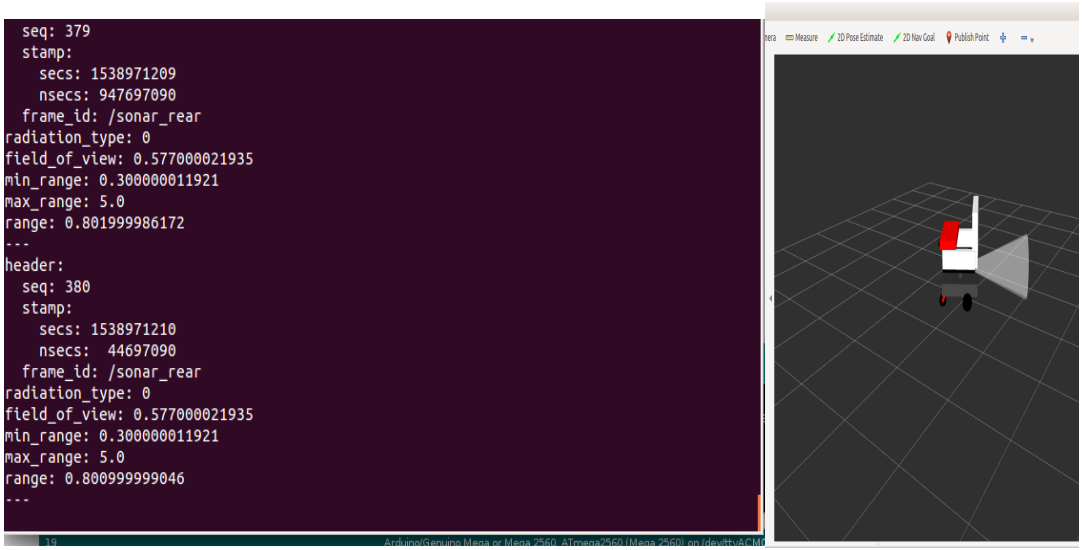


Figure 28. Sonar sensor range and visualization

The cells in the costmap have a value between the range 0-255 where high costs decrease the desirability of the robot being in the cell and a few values which are frequently used[33]:

- costmap_2d::NO_INFORMATION (255) - Reserved for cells where not enough information is sufficiently known.
- costmap_2d::LETHAL_OBSTACLE (254) - Indicates a collision causing obstacle was sensed in this cell.
- costmap_2d::INSCRIBED_INFLATED_OBSTACLE (253) - Indicates no obstacle, but moving the center of the robot to this location will result in a collision.
- costmap_2d::FREE_SPACE (0) - Cells where there are no obstacles and the moving the center of the robot to this position will not result in a collision.

The wheelchair uses two types of costmaps for navigation.

1. **Global Costmap** – This is used for generating a path in the global navigation or a destination on the map which is far.

2. **Local Costmap** – This is used for generating paths for local navigation which includes avoiding obstacles.

Path Generation Algorithm

Ros package **"base_local_planner"** implements the trajectory rollout and dynamic window (DWA) approaches where a plan to follow with the costmap is provided generates velocity commands[36].

1. Discretely sample in the robot's control space (dx,dy,dtheta)
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5. Rinse and repeat.

The only difference between trajectory rollout and dwa approach is how the robot's control space is sampled.

3.3.6 ROS-ARDUINO Block diagram

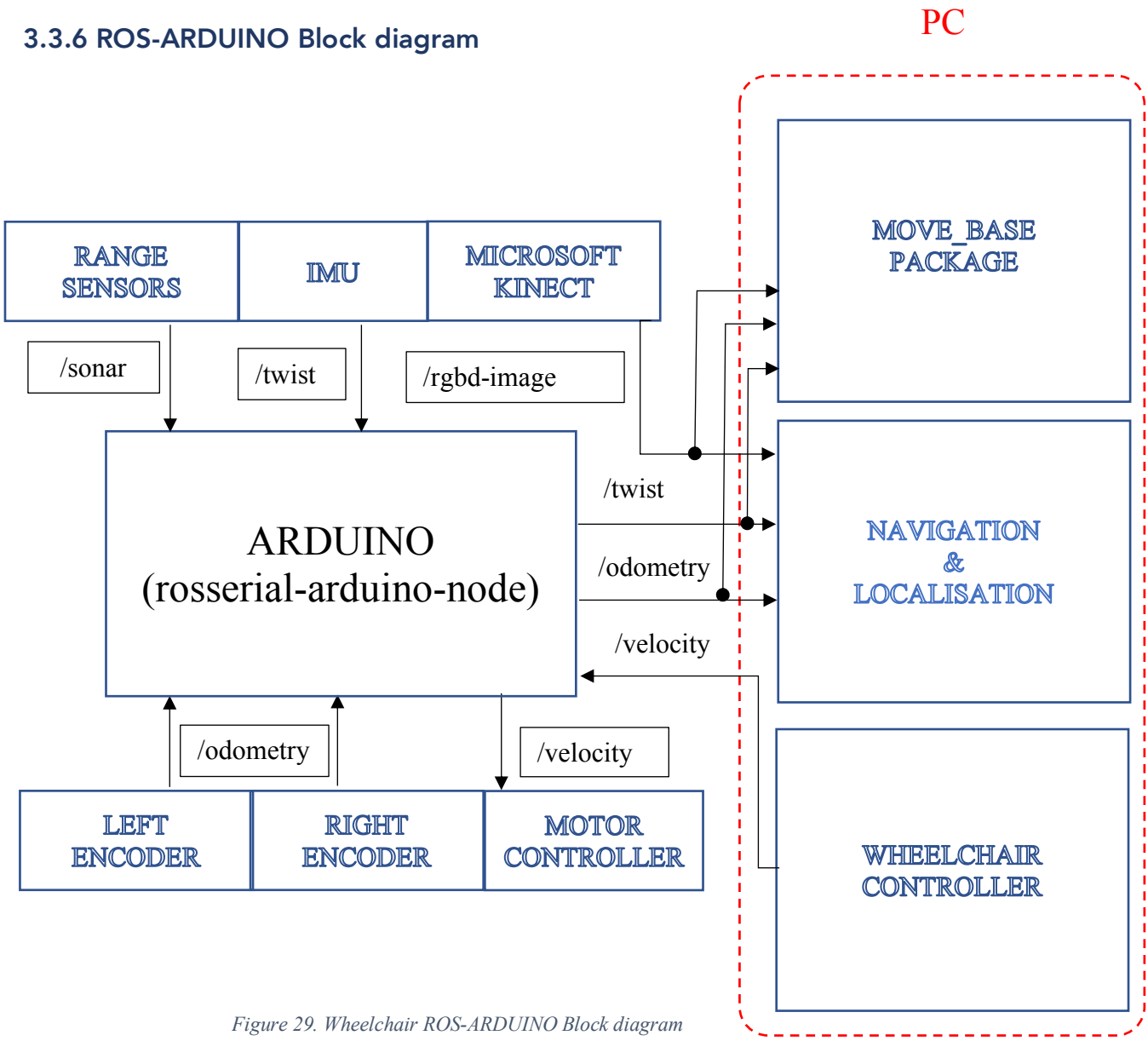


Figure 29. Wheelchair ROS-ARDUINO Block diagram

The block diagram shows how the various sensors are connected to the Arduino and how messages are communicated with ROS to achieve the goal of autonomous robot.

3.3.7 RTAB-Map (Real-Time Appearance Based Mapping)

RTAB-Map [17] is a RGB-D Graph-Based Slam approach based on appearance-based loop closure detector. The loop closure detector uses a bag-of-words approach to determine how likely a new image comes from a previous location or a new location and when a loop closure hypothesis is accepted, a new constraint is added to the map's graph.

Features[25]

- **Visual Odometry** [15] – It is the process of determining the position and orientation of a robot by analyzing the associated camera images. Similar to odometry generated using encoders, Monocular, stereo cameras and RGB-D cameras provide the ability to compute motion using a feature based method where features are extracted in an image and tracked using the image sequence. By correlating the correspondence of the images in sequence, the camera motion can be estimated using optical flow. There are multiple ways of extracting egomotion such as using image intensities or using optical flow to match features detected through multiple frames which provides the direction of motion of a camera and thus an estimate of the camera motion.
- **Using Pointclouds for generating map**: One of the drawbacks for using the Gmapping package is the requirement for using lidars with long range and does not work well with short range sensors while RTAB-Map allows the use of pointclouds to generate the map.
- **3D Map Cloud**: The point clouds are used to generate a cloud map of the entire map which can be used for a 3D visualization of the environment.
- **Allows Handheld Mapping:** The package allows for handheld mapping using an RGB-D camera, which can be highly beneficial for this study as it allows for the wheelchair to localize itself within these areas which are mapped using the handheld mapping technique.

Chapter 4

EXPERIMENTAL RESULTS

This chapter has been broken down into segments starting with the initial approach applied to develop the system for understanding the systematic parameters and its limitations. The next segment discusses the development of a landmark based navigation technique to reach a goal. The third segment contains the integration of the wheelchair with ROS and how the it generates a map of its surrounding and navigates within it.

4.1 Initial Approach

4.1.1 Mechanical Design Modifications

To convert the wheelchair to an autonomous wheelchair there are certain modifications which have to be made. The initial setup of the wheelchair tray included a wooden block covering the entire front making it hard for anyone to enter and exit without lifting the tray completely making it a tiresome task. The first step was to modify the tray for a more ergonomic design and also ease of access. The tray is designed keeping in mind the generic size of a laptop that might be used with the wheelchair in addition to the various sensors to be added and also the sensors that could be included in the future developments of the project and finally, enough space for any user to sit comfortably. The design for the tray is included in Appendix section B3. The tray is attached using hinges which allows it to be lifted providing much more ease for the user to enter and exit the wheelchair. The joystick for manual control of the wheelchair was previously fixed on top of the tray which meant that the user has to lift their hands for the entire time the joystick was used making it a very inefficient way of controlling the wheelchair. A new joystick holder is designed for easier access for manual control of the wheelchair.

Figure 30. Mechanical design changes to the wheelchair

4.1.2 Migration from PIC microcontroller to Arduino:

The initial setup was integrated using a PIC **(Peripheral Interface Controller)** microcontroller which is replaced with an Arduino microcontroller(Arduino Mega) since Arduino is a more generic microcontroller used for prototyping and offers support to a broad range of sensors as compared to the former, thus allowing for further development for the project easier. The initial steps include understanding how the interface board communicates with the joystick and the motor controller. [55] served as reference for the initial communication between the interface board and Arduino which allowed control of the wheelchair through the joystick. The block diagram explains how the communication takes place.

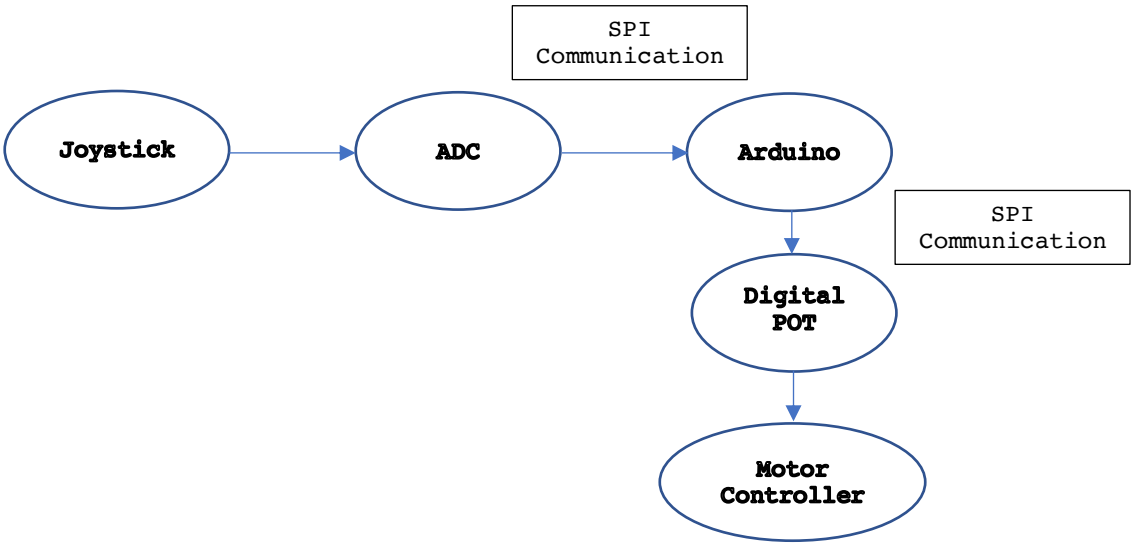


Figure 31. Interface board and Arduino connection

4.1.3 Mapping joystick values for external input to drive the wheelchair

The figure below shows the values mapped that are sent to the potentiometer correlating to the joystick position. Where FB (Front Back) and LR (Left Right) are the voltages received from channel 1 and channel 2 respectively from the ADC shown in figure 8.

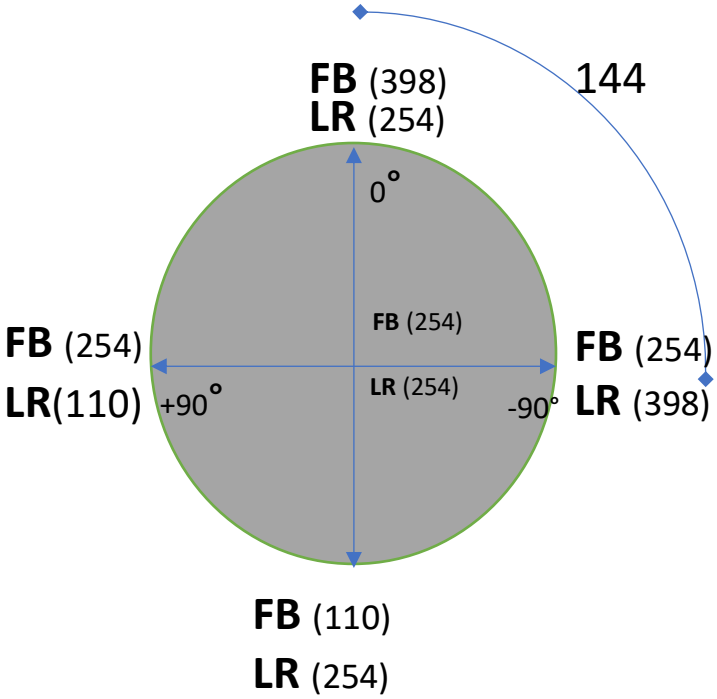


Figure 32. Joystick output values

The digital values from the joystick are then mapped to the potentiometer range using the equation below.

$$Potentiometer\ value = (Joystick\ Voltage - 100) * multiplier \quad \text{(Eqn. 1)}$$

where *multiplier = 0.85*

POT_FB – Forward/Reverse, **POT_LR** – Left/Right

Direction	POT_FB	POT_LR
Forward	256	128
Reverse	9	128
Right	128	9
Left	128	256

Table 6. Potentiometer Inputs

4.1.4 Controlling the wheelchair using a keyboard and moving the wheelchair in a pre-defined sequence

Using the values in figure 32, an Arduino code is written which listens to the serial buffer for the key pressed on the keyboard and drives the wheelchair in the specific direction.

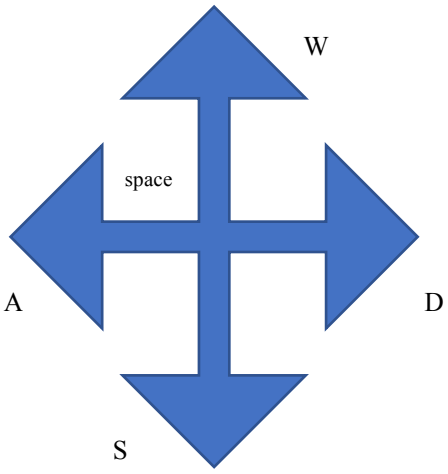


Figure 33. Keyboard inputs

For understanding the delays between the input and the execution of the command, the wheelchair was moved in a pre-defined sequence (a square) that showed an offset for a minimum of 1 meter at each trial. It was observed that the wheels of the wheelchair move at a different speed which resulted in a deviation from the path when the wheelchair moves in a straight line. The slight delay between the commands and the execution was observed which resulted in an offset from the starting point once the sequence was complete. The code for controlling the wheelchair using the keyboard can be found in (APPENDIX A: Section A2)

4.1.5 Speed and Direction Control:

From the previous sections, the wheelchair has been controlled by sending constant values for rotational and linear speed, but a robotic system should be adaptive to its dynamically changing environment and hence needs a better speed controller. The values sent to the potentiometer are 8-bit values ranging from 1-256 for both channels and table 6 shows the values that are sent to the potentiometer.

It is observed from figure 32 that the difference in values for every **90°** is **144**.

At the mid-point the value for **POT_LR** is **254** and at the extreme ends(left & right) the potentiometer values are 110 and 398 respectively each with a difference of 144 from the center point.

Mapping the values:

$$scale = \frac{144}{90} = \mathbf{1.6}$$

Which will be referred to as **scale.** 1° **(degree)** angle can be represented by 1.6 points on the potentiometer scale.

Generating the equations:

$$Turn_{Value} = (254 - \boldsymbol{scale} * angle) \qquad \text{where, } angle \, \epsilon \{-90, 90\} \quad \text{(Eqn 2)}$$

Feeding the Turn value into Eqn 1,

$$POT_{Value} = (Turn_{Value} - 100) * multiplier$$

Using the equations:

Let, $angle = 45$

$$Turn_{Value} = \big(254 + 1.6 * (45)\big) = 326$$

Verifying the value from the first quadrant:

$$144/2 = 72$$

$$398 - 72 = 254 + 72 = 326$$

The equations allowed the wheelchair to turn at any given angle providing a more adaptive controller.

Program code to achieve specific turn angles section A5 of the APPENDIX

4.1.6 Avoiding obstacles Safely

The objective for the wheelchair is to navigate autonomously while avoiding all static and dynamic obstacles. A sonar sensor as shown in figure 9 was attached to the bottom of the wheelchair tray to detect obstacles in view of the wheelchair.

The sonar sensor has a range from **30mm – 5000mm**. Considering the width of the wheelchair and a keeping free radius of 0.20m at all times the following table shows the relation between the distance and the angle required for the wheelchair to safely avoid hitting any obstacle in front of it.

Column	Distance	Angle(degrees)
1	>1000	No Turn
2	1000	±45°
3	900	±50.625°
4	800	±56.25°
5	700	±61.875°
6	600	±67.5°
7	500	±73.125°
8	400	±78.75°
9	300	±84.375°
10	200	±90°
11	<200	Reverse

Table 7. Distance - Angle relation

Using equation 2 generated in the previous section (improved direction control), if the values for the angle and the distance are known, the turn angle can be calculated for the wheelchair using table 1. The angle is mapped from 45° to 90° to a distance from 30mm to 1000mm and the distance range is divided in 10 divisions with an increment of 5.625° for every 100mm.

$$dist_{scale} \rightarrow \frac{90°}{2} = \frac{144}{2} \rightarrow \frac{77}{45} = 1.711$$

Calculating angle in degrees:

Total number of divisions in range=10

To find the value of the column value from table 3.

$$Pointer = Number\ of\ Division - (distance/100)$$

$$Column_{angle} = Pointer * 5.625.$$

$$Turn_{angle}(degree) = 45 + Column_{angle}$$

$$Turn_{angle}(potentiometer) = Turn_{angle}(degree) * dist_{scale}$$

$$POT_{LR_{val}} = 254 - \boldsymbol{Turnangle(potentiometer)}$$

Or $POT_{LRval} = [254 - (10 - (distance\ from\ obstacle/100)) * 5.625]$ (Eqn **3**)

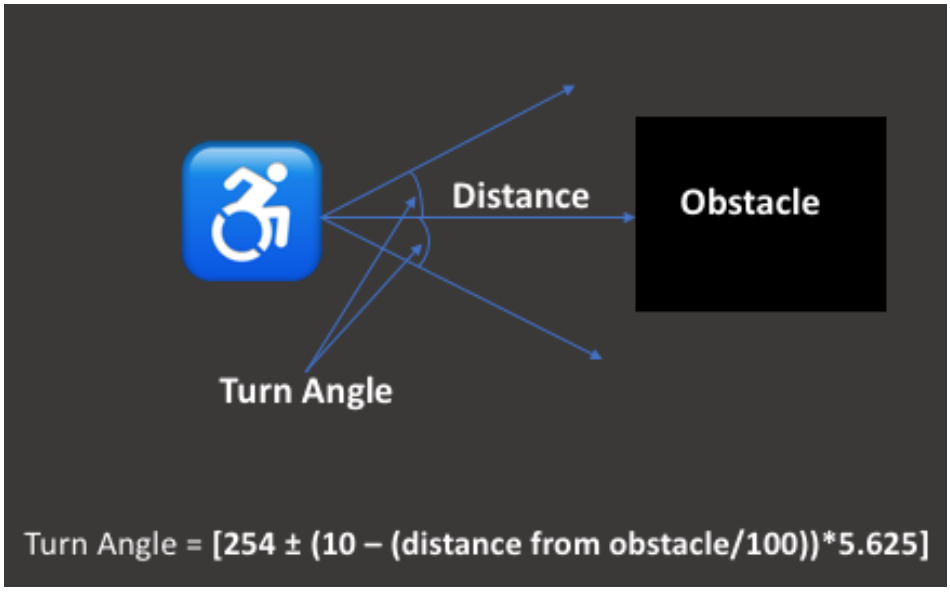


Figure 34. Avoiding distances safely

Using the equations above.

Let distance from obstacle = 550mm

Turn Angle should fall in the range of 67.5 and 73.125° according to Table 3.

$$angle = \left[\left(10 - \left(\frac{550}{100}\right)\right) * 5.625\right] = [4.5 * 5.625] = 25.3125$$

$$Turnangle(degree) = 45 + 25.3125 = 70.3125$$

The performance of the wheelchair matches the expected outcome by avoiding all obstacles and reversing if it gets too close to any wall or obstacle. The program can be found in section A6 of the APPENDIX

4.1.7 Controlling wheelchair through speech commands:

The idea for ABC wheelchair is to be able to use voice or speech commands as an input to execute tasks. Now the wheelchair can navigate freely while avoiding obstacles, similar to being controlled by the keyboard a new capability was tested which included controlling the wheelchair using speech commands.

BitVoicer by BitSophia

"BitVoicer [7] is a speech recognition application that enables simple devices, with low processing power, to become voice-operated. To do that, BitVoicer uses the PC processing power to analyse audio streams, identify the sentences present in these streams and send commands to a microcontroller connected to it."

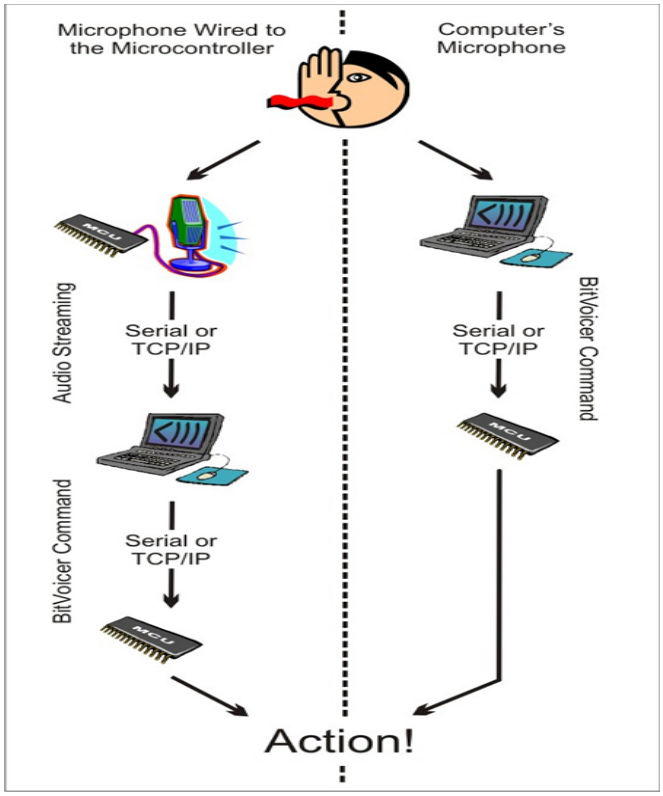


Figure 35. BITVOICER functionality [7]

BitVoicer processes the audio captured through the computer's microphone and compares to the written commands stored in the database. Since BitVoicer uses serial communication interface, it can directly be integrated with the Arduino

microcontroller and various commands can be executed. The code can be found in the APPENDIX: Section A4.

The commands used were Forward, Reverse, Left, Right, Turn Around. The voice commands get recognized with high confidence indoors but however, when outdoors the confidence level of commands recognized reduces significantly. After the commands are recognized the wheelchair performs the expected tasks of travelling in the desired direction. It was observed that keyworks such as forward, right, left, turn and around are commonly used in daily life by a majority of individuals which caused false positives being recognized as commands as the ASR platform could not differentiate between a command and conversation that the user operating the wheelchair is indulging in.

To make the experience safer, the keywords to be recognized were changed to Drive Forward, Drive Reverse, Turn Right and Turn Left which reduces the probability of the common keywords used and not affecting the behavior of the wheelchair. The recognition of commands is not limited to sending directions to the controller but can also be used to navigate to pre-defined positions or landmarks within a map. BitVoicer offers an unlimited number of commands that get can be recognized. An example for implementing the speech command with the autonomous wheelchair could be related to the idea explained during the introduction where the user would like to travel to the cafeteria and the wheelchair can navigate their autonomously by accepting the option.

4.1.8 Detecting known objects:

Robotic systems interpret the environment using sensors such as proximity sensors, distance sensors, bearing sensors and cameras. Cameras can be used to detect objects, people, shapes, signs etc. For a robotic system to provide a safe and the wheelchair must interpret the environment correctly and execute tasks defined.

After the wheelchair could move around using voice commands and avoid obstacles, a beneficial feature for the system would be to identify objects such as doors to pass through or signboards such as toilets or office numbers. As seen in figure 30, the wheelchair tray also includes an USB camera which provides an RGB-image which is used for detecting known objects. Object detection can be defined as a technology that deals with detecting objects of a certain class in images or videos. **OpenCV** (Open Source Computer Vision)[58] is a library of functions for real time computer vison applications.

Using the source code from [3] , an object detector was created using OpenCV and the camera attached to the front tray. The image below was used as the object to be identified in the lab environment to find the exit from the lab which was identified from all locations inside the lab resulting in a reliable object detection system which will used as an accessory during navigation. The idea to be implemented was by using the servo attached which rotates from 0-180° and when an object is detected the servo stops rotating and the angle at which the servo stops is considered as the direction of the object which is them refined by checking for the object multiple times but since the web-camera was replaced with a Kinect sensor the step was not implemented. However, since the Kinect sensor provides depth information in addition to an RGB image the location of the object can be estimated using the pointcloud. This will be used in further development of the project using the same object detector created. The python code for the object detector can be found in (section A7 from APPENDIX A)

Figure 36. Known door from lab at used for object detection

4.2 Project Development

4.2.1 Generating Odometry:

Odometry can be defined as estimating the robots position with time. The encoders attached to the wheelchair wheel provide pulses which can be used to estimate the motion the robot generates as shown in figure37.

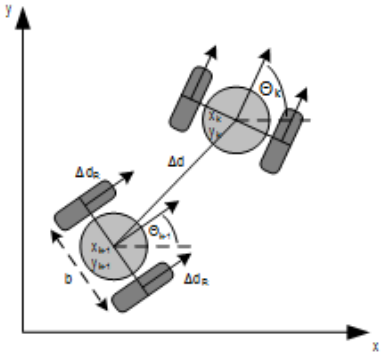


Figure 37. Generating odometry [42]

Current Position[1] of robot (x, y, θ) , solving for (x', y', θ')

Where,

w = wheelchair_wheel

e = encoder_wheel

$Wheelchair\_Baseline, b$

$$b = 462mm$$

$Wheel\_Diameter$, d

$$d_w = 195mm$$

$$d_e = 62mm$$

$$\frac{d_w}{d_e} = 3.145$$

Distance travelled by wheel for 1 complete rotation,

$$\delta_w = \Pi d_w$$

Pulses from encoder, $N \quad \vartheta_e = 400$

Wheelchair resolution $\vartheta_w = 400 * 3.145 = 1258$

Distance travelled by wheel at any time, $d_{r/l} = \frac{N * \delta_w}{\vartheta_w}$

$$d_{center} = \frac{d_r + d_l}{2} \quad \text{(Eqn 6)}$$

$$\phi = \frac{d_r - d_l}{b} \quad \text{(Eqn 7)}$$

$$\theta' = \theta + \phi \quad \text{(Eqn 8)}$$

$$x' = x + d_{center} * \cos(\theta) \quad \text{(Eqn 9)}$$

$$y' = y + d_{center} * \sin(\theta) \quad \text{(Eqn 10)}$$

Arduino code[A9]

4.2.2 Navigating to landmarks:

A basic robot navigation based on landmarks defined by position relative to a starting point was implemented after generating the odometry. The robot takes an input argument for the destination position and travels to the goal using odometry information.

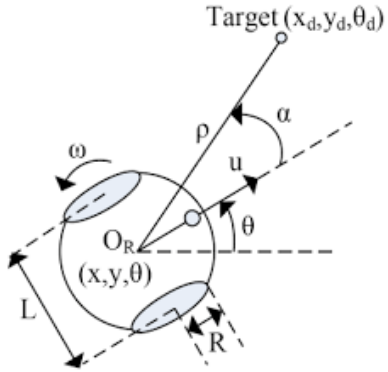


Figure 38. Robot landmark navigation[50]

Using figure 38 as reference, where the current position is defined by $P_0(x_0, y_0, \theta_0)$ and $P_d(x_d, y_d, \theta_d)$ is the destination position.

The distance between the points ρ can be calculated using the Pythagorean theorem or distance formula.

$$\rho = \sqrt{((x_d - x_0)^2 + (y_d - y_d)^2)} \qquad \text{(Eqn 11)}$$

and the heading difference α can be calculated using

$$\alpha = atan2(y_d - y_0, x_d - x_0) - \theta_0 \qquad \text{(Eqn 12)}$$

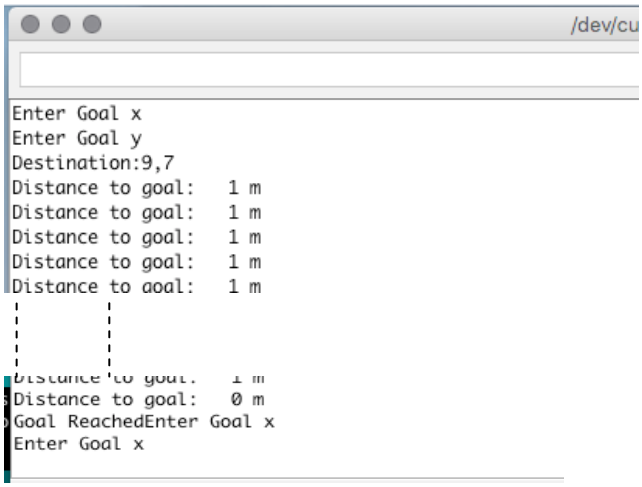


Figure 39. Navigating to landmarks

4.2.3 Calculating dead-reckoning errors for Improving Odometry estimation:

The process described above for generating odometry works well in ideal situations but there are a lot of errors which occur and are not accounted for. The major issue with using a dead-reckoning process to estimate the robot's position is due to the fact it integrates the position over time the errors are also accumulated over time. The errors can be caused due to various reasons encoder wheel slippage because of which linear movement of the wheelchair wheel unaccounted for being the major reason for error in this setup is due to the wheel on wheel configuration for placing the encoders. The errors in odometry can be categorized in two categories[5]:

Systematic errors:

- Unequal wheel diameters
- Misalignment of wheels
- Incorrect wheelbase (the wheels are not in contact with the floor at a point but rather an area)

Non-systematic errors:

- Wheel-slippage
- Travelling over uneven surfaces
- Internal and external force errors (example castor wheels)

For simplicity reasons, since the wheelchair is mostly driven in indoor environments where the floor is mostly even, only the systematic errors are considered which are mostly due to the defects in the mechanical design of the robot. The tyres on the wheelchair are made with rubber, so it is susceptible to wear over time and the wheelchair used for the project is over 7 years old because of the usage and the load over the time the wheels have worn out leading to unequal diameters. The other error not accounted for is the incorrect wheelbase value used for the wheelchair. The wheelbase helps identify the Instantaneous Centre of Rotation (ICR) as shown in the figure below. The wheelbase of a robot is calculated by the distance between the

point of contact of the wheels with the floor but since the wheel is in contact in an area and not a point resulting with an ineffective wheelbase estimate.

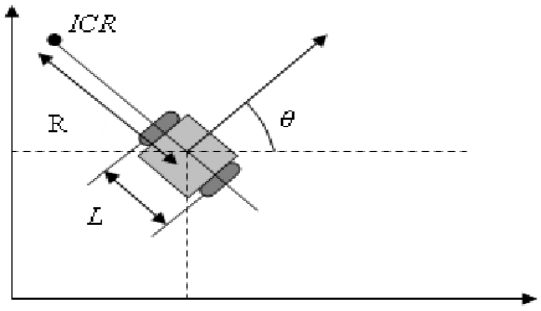


Figure 40. Wheeled robot kinematics [52]

A procedure created in 1994 by the University of Michigan called UMBmark [5] helps to measure, calculate and correct dead-reckoning errors. The test consists of manually moving the robot around a square path in both clockwise and counter-clockwise direction from a starting position as shown in the figure below.

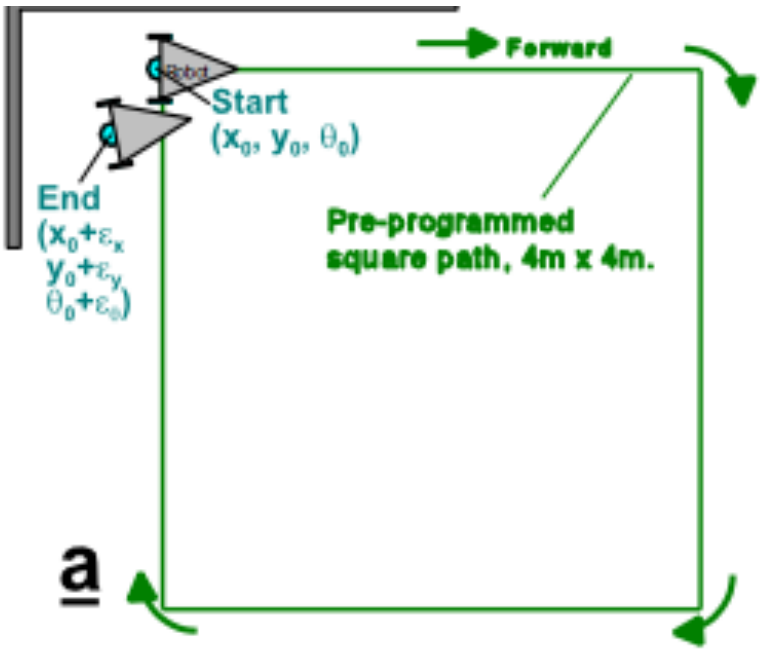


Figure 41. UMBmark test

Performing the test and calculating the correction factors for positional errors caused by various odometry error sources. The results for the path are plotted in MATLAB

for before and after incorporating the correction factor which shows an improvement in the accuracy of the path taken.

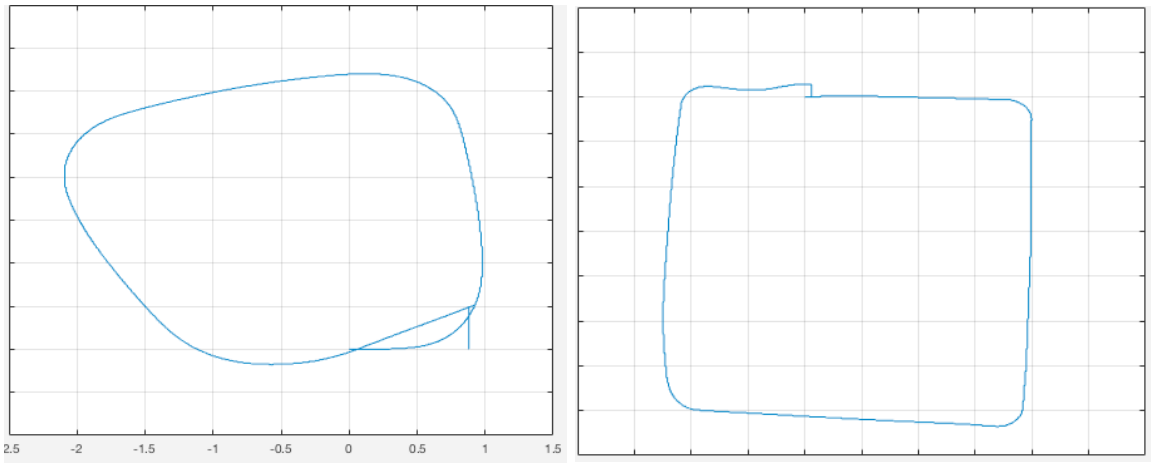


Figure 42. Raw Odometry and corrected odometry

4.2.4 Fusing encoders and orientation from IMU to generate odometry

The odometry generated using the encoders is reliable but since the errors in dead-reckoning accumulate over time, the errors in the position gradually increase in an odometry model. To improve the accuracy of the position estimation, the data from the encoders and the IMU can be fused to generate a motion model.

The gyroscope on the IMU provides the rotational velocity and the encoders provide displacement information for how much the wheels rotate providing translation motion information.

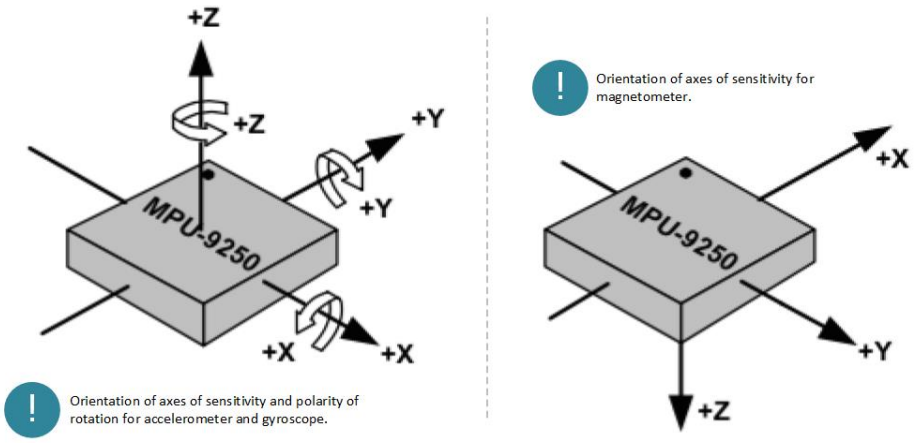


Figure 43. IMU axis [18]

The IMU has three sensors with each sensor has its functionality and limitations.

- **ACCELEROMTER** > X, Y, & Z linear axis motion sensing (sensitive to vibration)
- **GYROSCOPE** > Pitch, Roll Yaw Rotational Sensing (Gyroscope Drift)
- **MAGNETOMETER** > X, Y, & Z axis magnetic field sensing (Sensitive to magnetic interface)

The gyroscope calculates orientation through integrating angular velocities and since there is no frame of reference the values drift over time, but the drift is reasonable enough for a better estimation of position. The orientation is calculated using the equation below.

$$\theta_i = \int g_z dt \qquad \text{(Eqn 14)}$$

The accelerometer tends to distort the acceleration due to external gravitational forces in the motion which accumulates as noise in the system and produce errors in the output. Using accelerometer to produce position requires the acceleration to be integrated twice to determine a position estimate inducing noise in the system.

$$v_i = \int \alpha_i$$

$$p_i = \iint a_i$$

The magnetometer measures the orientation with respect to the true north but is highly susceptible to noise due to electric interference and magnetic fields in electric systems due to which the readings from the magnetometer are not considered for fusion.

Substituting the value from **Eqn 7** with the value of from **Eqn 14** in **Eqn 8**

$$\theta_{imu} = \int g_z dt$$

$$d_{center} = \frac{d_r + d_l}{2}$$

$$x' = x + d_{center} * \cos(\theta_{imu}) \qquad \text{(Eqn 15)}$$

$$y' = y + d_{center} * \sin(\theta_{imu}) \qquad \text{(Eqn 16)}$$

The Arduino code for computing the odometry using encoders and the IMU can be found in Appendix [A11].

4.3 Integration with ROS

4.3.1 Initial communication between ROS and Arduino:

The Arduino code [APPENDIX A8)] is written to move the wheelchair either forward, left, right and reverse using the robot steering plugin available through **rqt** in ROS.

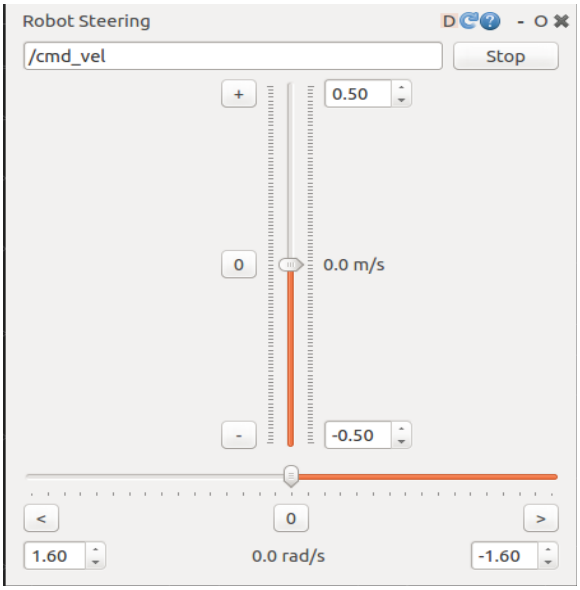


Figure 44. RQT plugin for robot steering

Using the robot steering plugin when the sliders are moved, it enables the wheelchair to move in the respective direction. This step forms the basis of understanding the subscribers and publishers in ROS and how the communication between Arduino and ROS is established.

4.3.2 Creating the Unified Robot Description Format (URDF) for the wheelchair:

ROS package **"URDF"** which is a parser for the XML format for representing a robot model and can be visualized in ROS through RVIZ and simulated in ROS through Gazebo. Robot kinematics is the relationship between the connectivity of the links and joints of the robot. These links define the mechanical structure of the robot and the position and orientation of different components or sensors attached to the robot as shown in figure 45 and 46 below.

The XML code for the URDF model of the wheelchair can be found in Appendix A12.

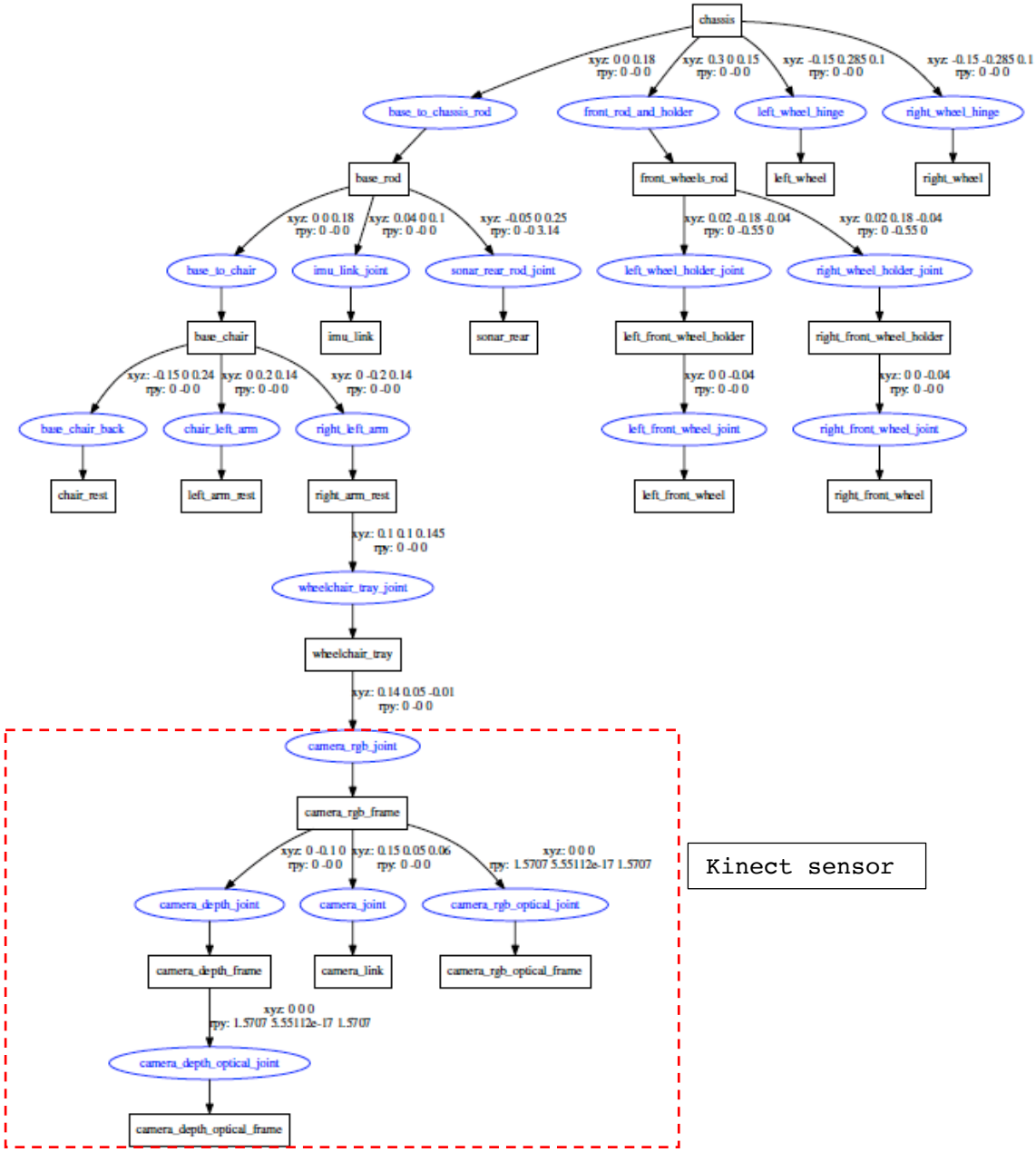


Figure 45. Graphviz of urdf model

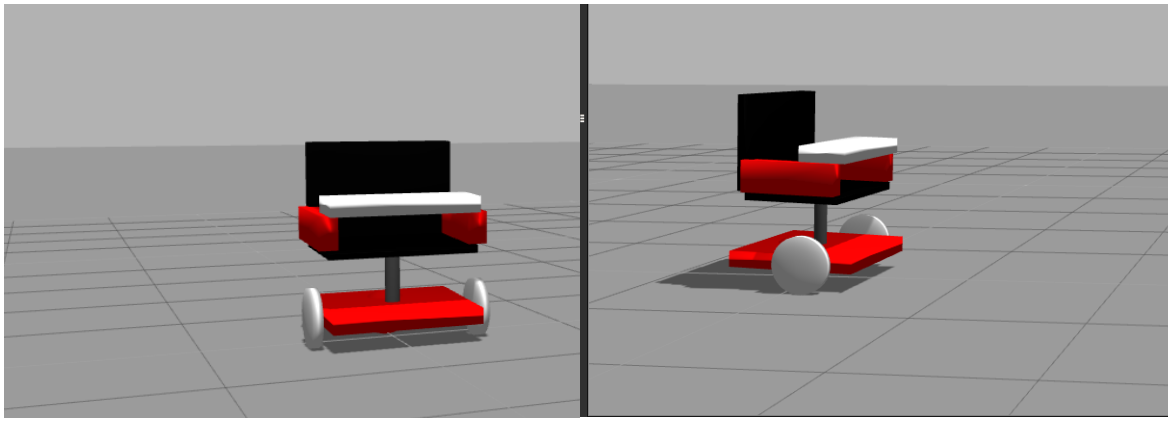


Figure 46. URDF model for wheelchair

4.3.3 Creating a wheelchair follower:

One of the options available to the user through the interface as shown in figure 1 is the follower mode, where the wheelchair can follow a certain individual or an object. In this case the object the wheelchair follows is a QR code.

ViSP (Visual Servoing Platform):

ROS package "**visp**" is a complete cross-platform solution that allows prototyping and developing applications in visual tracking and visual servoing. ViSP can be useful in robotics, computer vision, augmented reality and computer animation. The package provides trackers that rely on visual servoing techniques and can help track an object and even estimate its position in real-time.

Visp Auto Tracker

The package depends on the Visp library for visual servoing. The package wraps model-based trackers that have a QRcode or Flash code pattern, and the computer vision algorithm can compute the position and orientation of the object in the image which is fast enough to allow online or real time tracking.

Figure 47. QR code used for creating a follower

The algorithm detects the barcode using QR-code detection and uses the position of the corners to compute an initial pose. Once the initial pose is estimated the model-based tracker is initialized which tracks the squares around the QR-code. The

tracker considers moving edges and the keypoint features on the barcode for estimation. A python script written to read the position of the QRcode and follow the code based on the change in the x and z position by determining the difference in the direction and displacement of QRcode from its previous position and current position and generates velocity values to be sent to the controller which can be found in the Appendix section A13.

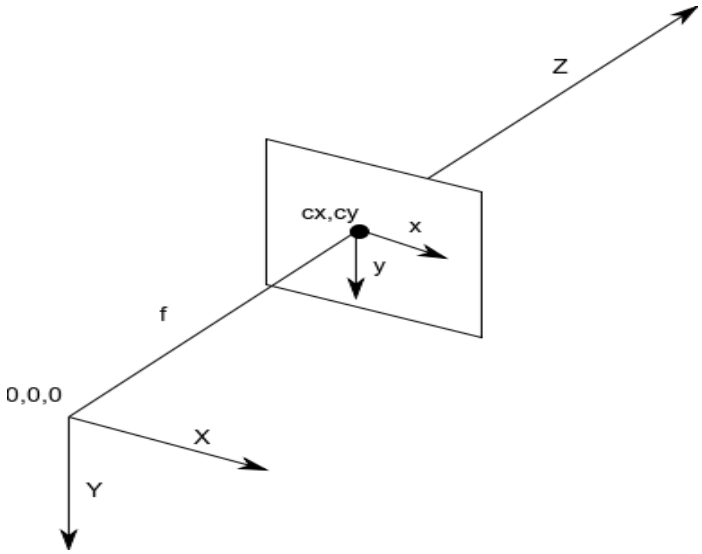


Figure 48. Camera Coordinates[2]

Figure 49. Detecting QRcode

Figure 50 below explains the how the change in position of the QRcode is mapped to the wheelchair follower using the Z-axis to generate linear (forward and reverse) motion and X-axis for rotational motion.

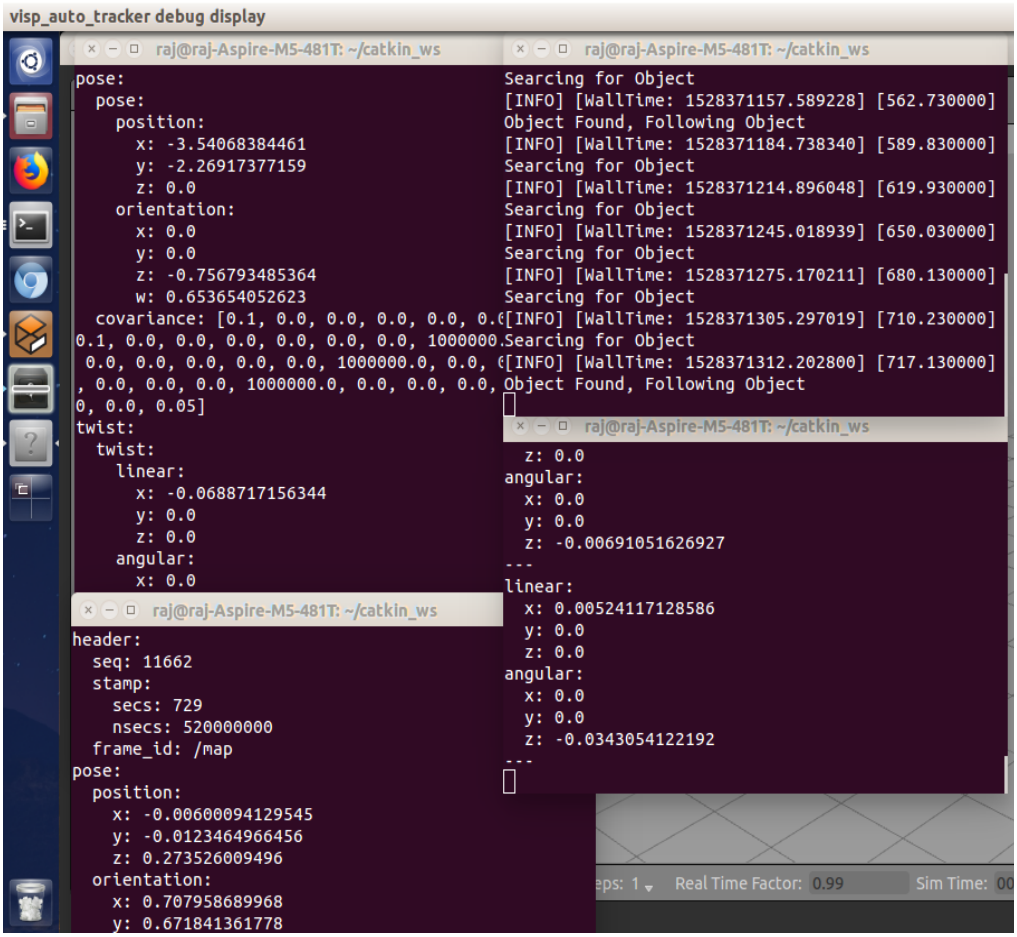


Figure 50. Wheelchair follower using QR code

4.3.4 Creating a Velocity controller:

After testing the wheelchair follower setup, the rotational speed for the wheelchair was always fixed to ±1.57 rad/sec resulting in excess rotation causing oscillation and jerks. To improve this, using the equations (Eqn 1, Eqn 2) to generate a smoother motion. The values for the velocity received through ROS are of the double data type and have to be converted to an integer data type without losing the information in the decimal places so the values cannot be rounded up or down. Arduino code [A14]

4.3.5 Fusing multiple sources for position estimation:

ROS package ***robot_localization*** uses an Extended Kalman Filter (EKF) to fuse multiple sources to better estimate an accurate motion that the wheelchair performs. EKF is a known as a linear quadratic estimation (LQE) which produces an estimate for a system which is accurate enough to estimate the current state of a system. The filter produces the results in two stages known as the **prediction** and the **correction** stage. By using a weighted average method, the relevant data is selected depending on the covariance values and the sensor model matrix. This determines how much a measurement should be trusted or can be relied on by calculating a gain called Kalman gain. Higher gains make the system depend more on the measurements and a lower gain enforces the system to depend on the prediction.

EKF Algorithm [59]:

$$\hat{x}_{k|k-1} = f_{k-1}(\hat{x}_{k-1|k-1}, u_k)$$
$$P_{k|k-1} = \hat{G}_{k-1}\, Q_{k-1}\hat{G}_{k-1}^T + \hat{F}_{k-1}P_{k-1|k-1}\hat{F}_{k-1}^T \quad \left.\right\} \text{ Prediction}$$
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k\,(z_k - h_k(\hat{x}_{k|k-1}))$$
$$P_{k\,|k} = P_{k|k-1} - K_k S_k K_k^T \quad \left.\right\} \text{ Correction/Update}$$

Where,

$$S_k = R_k + \hat{H}_k P_{k|k-1}\hat{H}_k^T$$
$$K_k = P_{k|k-1}\hat{H}_k^T S_k - 1$$
$$\hat{x}_k \rightarrow State\ vector = \begin{bmatrix} x \\ y \\ \theta \\ v \cdot \\ \omega \cdot \end{bmatrix}$$

Sources:
Pose(x, y and θ) from Wheel Odometry
Twist (linear acceleration and rotational velocity) from IMU

4.3.6 Calculating covariance values for wheel odometry:

From the UMBmark procedure the errors for the absolute position $(\mathrm{x}, \mathrm{y}, \theta)$ were calculated as shown in the table below.

Trial	Starting Position $(\mathbf{x_0}, \mathbf{y_0}, \boldsymbol{\theta_0})$	End Position $(\mathbf{x'}, \mathbf{y'}, \boldsymbol{\theta'})$
CW1	(0,0,0)	(0.27,0.04,0.15)
CW2	(0,0,0)	(0.18,0.0,0.13)
CW3	(0,0,0)	(0.01,0.01,0.01)
CW4	(0,0,0)	(0.48,0.16,0.09)
CW5	(0,0,0)	(0.08,0.18,0.03)
CCW1	(0,0,0)	(0.12,0.07,0.12)
CCW2	(0,0,0)	(0.01,0.01,0.16)
CCW3	(0,0,0)	(0.07,0.12,0.14)
CCW4	(0,0,0)	(0.19,0.07,0.21)
CCW5	(0,0,0)	(0.28,0.04,0.16)

Figure 51. Calculating covariance values

Mean: The average of all values

$$\mathrm{x}' = \frac{1}{n}\sum_{i=0}^{n} \mathrm{x_i}$$

Variance: Second moment of all values

$$\sigma_x^2 = \frac{1}{n}\sum_{i=0}^{n} (\mathrm{x_i} - x')\text{^}2$$

Standard Deviation: Square root of variance

$$\sigma_x = \sqrt{\sigma_x^2}$$

Co-variance: Second moment of all values, where x is the vector of values

$$\Sigma_x = \frac{i}{n}\sum_{i=0}^{n} [\mathrm{x_i} - x'][x_i - x']\text{^}T$$

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{vmatrix} 0.0212 & 0 & 0 \\ 0 & 0.0041 & 0 \\ 0 & 0 & 0.0042 \end{vmatrix}$$

The MATLAB script to generate correction factors and covariance values[A16]

4.3.7 Mapping with RTAB-Map

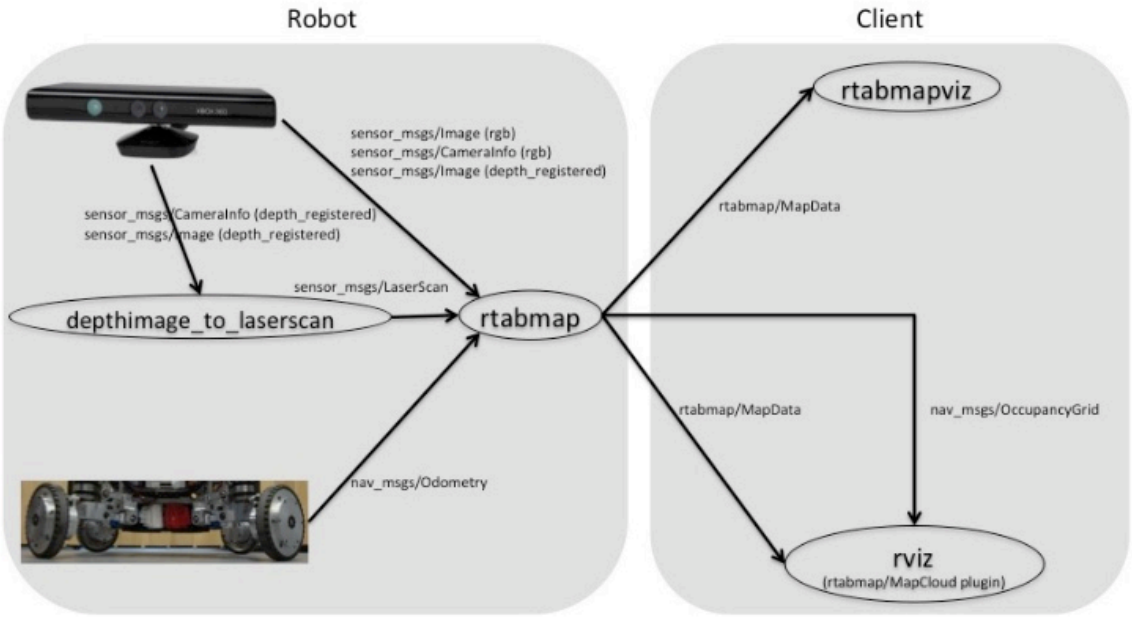


Figure 52. RTAB-Map robot setup [25]

Figure 53. shows the one of the setups for the robot to be used with the RTAB-Map package.

- Section 5.6 discusses how the odometry is generated by using the pulses from the encoders and rotational velocity from the gyroscope.
- The Kinect sensor attached provides a RGB-D image which is read using the "**openni_launch**" and "**freenect**" packages in ROS. The packages contain launch files which can be used for using the Microsoft Kinect in ROS. The packages create a nodelet which transforms the raw data from the sensor into point clouds and other formats which can be used for appropriate visualization or processing.
- Since the robot does not have a laser/lidar configured, the ros package **"depthimage_to_laserscan"**(Wiki.ros.org, n.d.) is used to simulate a 2D laser scan from the depth image from the Kinect as shown in the figure 49 below.
- Other ros packages used **"tf_to_odometry"** [26]

Figure 53. RGB image and depth image from the Kinect sensor

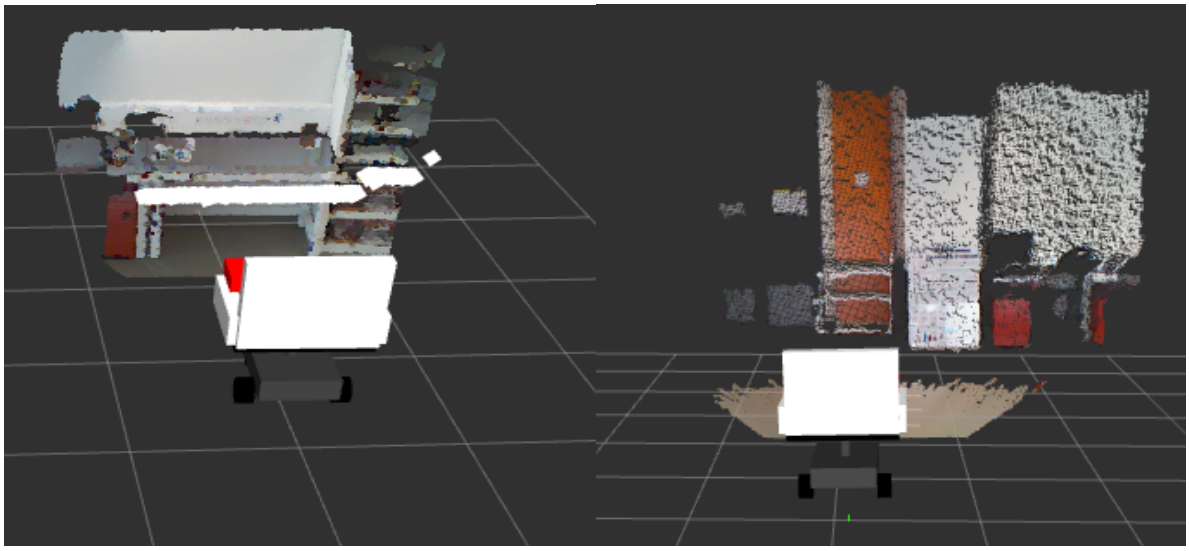


Figure 54. Pointcloud from RGB image and Depth registered points

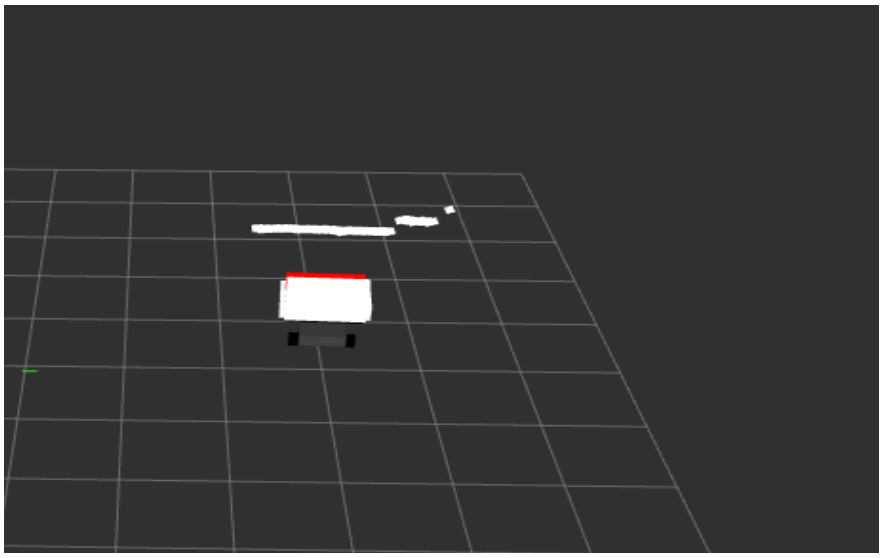


Figure 55. Laserscan from depthimage

4.3.8 Mapping the environment:

Once the requirements are met, the environment can be mapped by manually moving the wheelchair. The roslaunch file which contains the generic and recommended parameters for the RTAB-Map package[25] and the additional nodes used for the system can be found in Appendix A15.

Figure 56. Test lab area

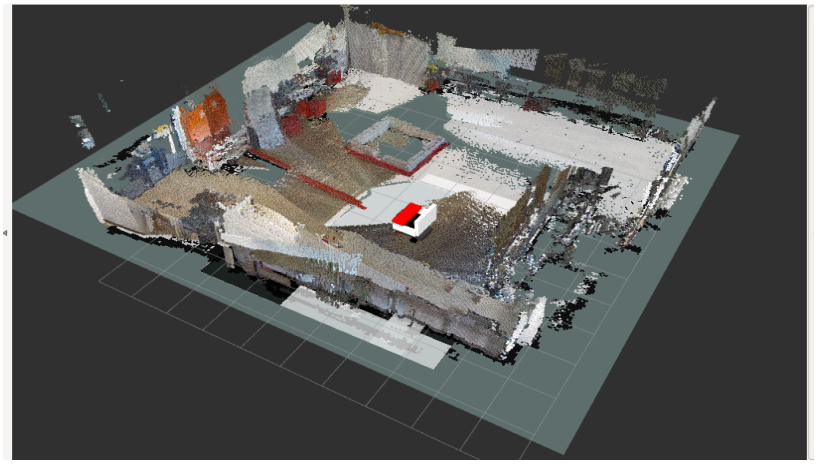


Figure 57. 3D Cloud map of testlab

The figure 57 illustrates the map cloud generated with the use of a Kinect sensor for the test lab. The detector[20] uses a bag-of-words approach to determine if a node has been visited and each loop closure adds a new edge to the graph and then the map is optimized. It can be observed that the algorithm builds dense 3D maps from

point clouds and features surfaces being repeated and a distorted visualization due to odometry noises.

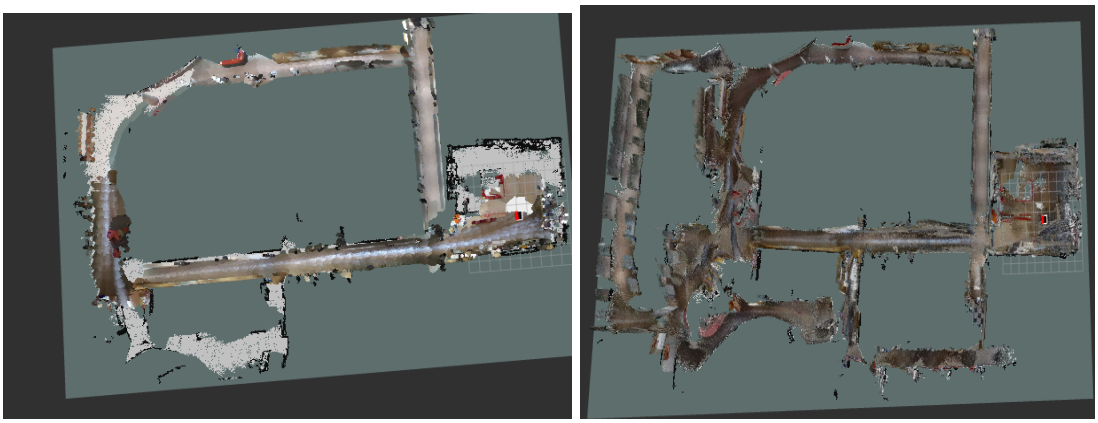


Figure 58. Mapping stages 1 and 2

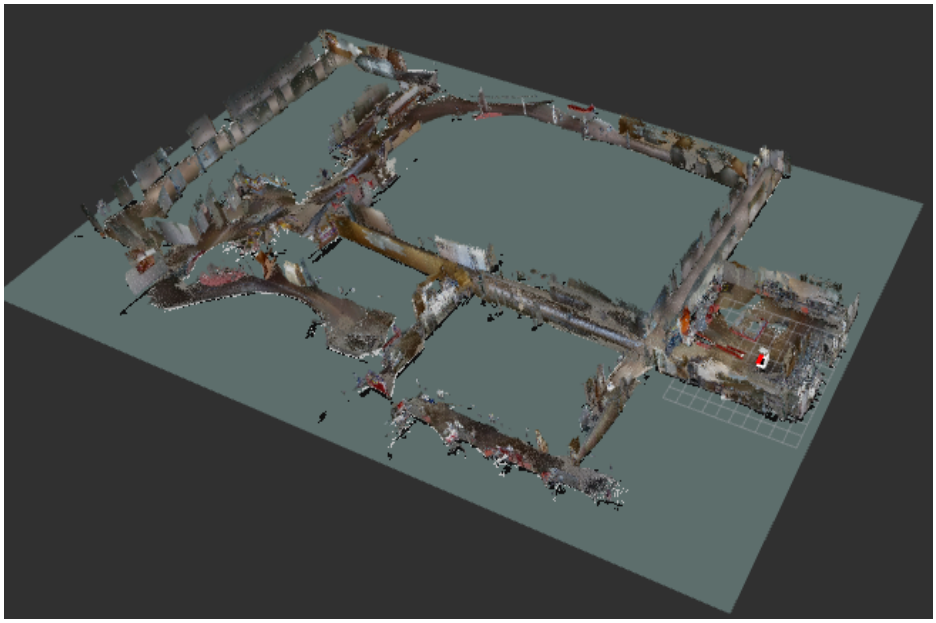


Figure 59. 3D Cloud map of wheelchair accessible area at Tonsley

The figures above show the cloud map for the 4th floor mapped at Tonsley building in stages. The wheelchair is manually moved around in segments where a particular area is mapped and then returned to the initial or starting point. As the wheelchair moves through previously visited areas new nodes are added to the previously saved nodes using odometry information and features from the environment. Once a loop closure occurs at the initial position new edges are added after which the map is optimized based on the most likely map from the information sources resulting in a 3D cloud map of the entire floor. The pointclouds can also be used to determine

obstacles in the environment, where the ground is segmented at a certain height. Then the three-dimensional map is collapsed down to two dimensions to generate an occupancy grid map.

Figure 60. Grid maps generated from depth image for stages 1 and 2

Figure 61. Occupancy Grid map of floor at Tonsley

The grid maps seem slightly distorted due to odometry noises and the narrow beam angle of the Kinect sensor. Since the Kinect only has a horizontal view of 57degree the area covered while mapping is low but provides enough RGB and depth information for generating maps which can be used for navigation. The grid represents the blue print of the wheelchair environment where the size of the pixels

in the real world and the information of the map are stored. The maps show excess grids occupied because the grid map is generated using the depth image rather than the laserscan.

4.3.9 Sending a goal

A goal is the pose (x, y position and the end orientation) of the robot in the map using the 2D Nav Goal shortcut on Rviz(figure 62) . Once a goal is sent, a path is generated to reach the goal as shown in 65 and 66 below. The local planner sends velocity values which generate motion for the wheelchair.

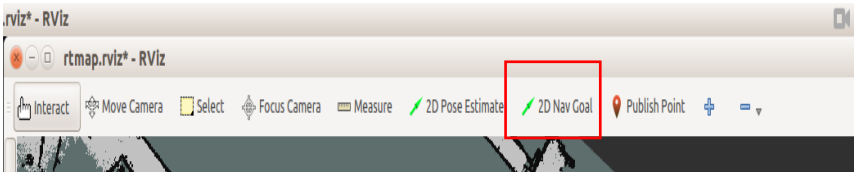


Figure 62. 2D Nav shortcut

Figure 63. Global costmap

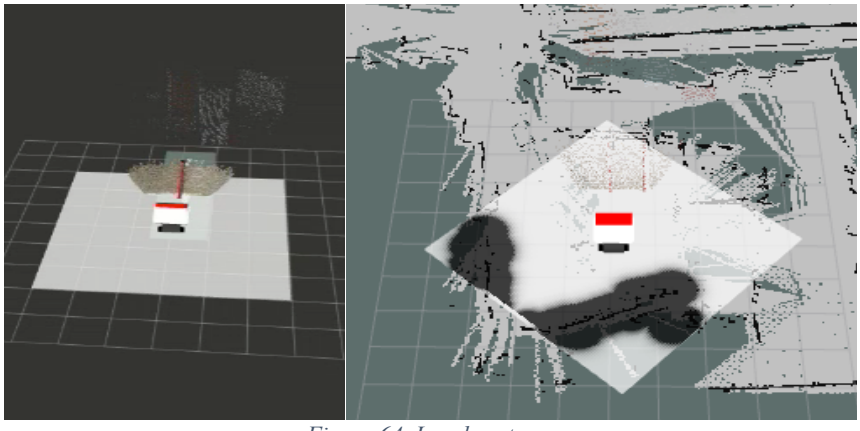


Figure 64. Local costmap

Figure 63 and 64 show the costmaps being generated using the Kinect camera and the sonar sensor. The size of the costmap window depend on the configuration set. The local costmap uses a window of 5x5m and is not static. The window moves as the wheelchair traverses through the path it is following.

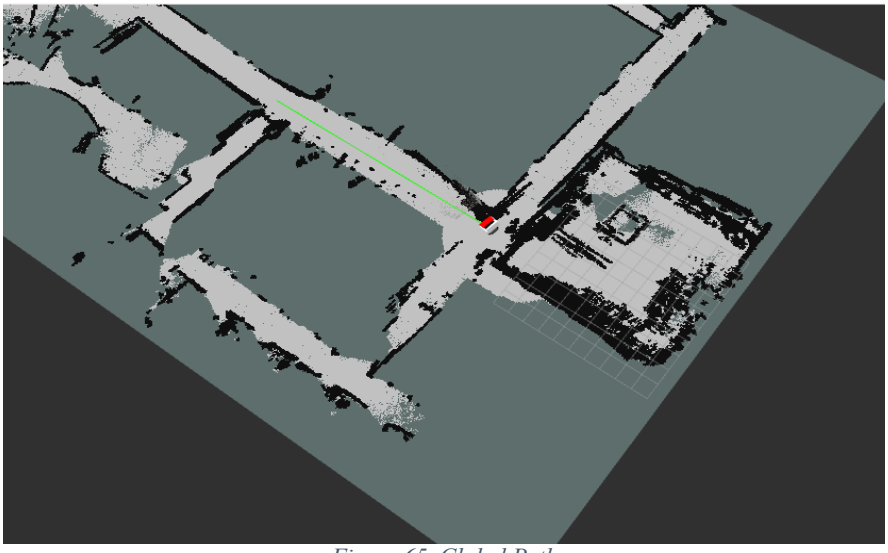


Figure 65. Global Path

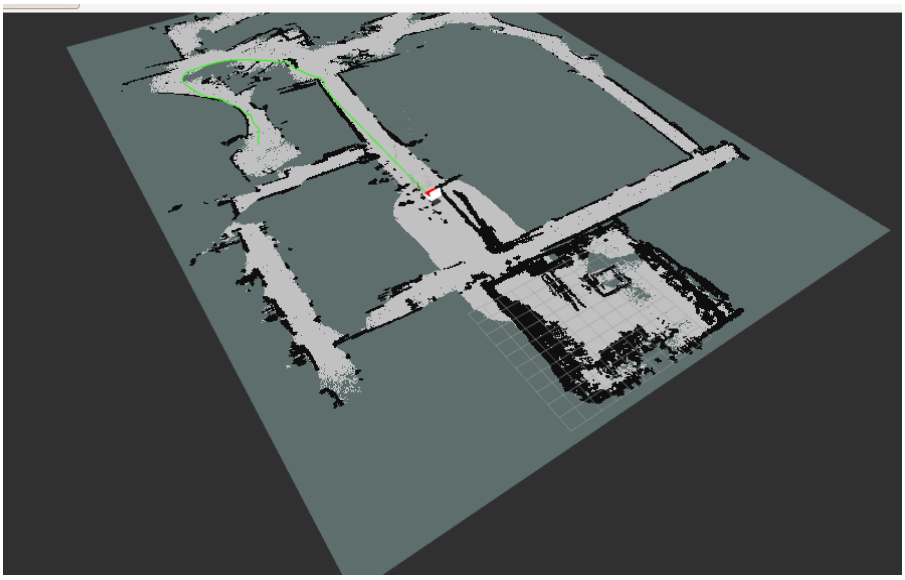


Figure 66. Full Plan

The path is shown in green in the figures above and the wheelchair follows the path to reach the destination set from figure 62. The wheelchair has been able to generate its map and localize itself within it. Once a goal is given a path is planned to reach the destination by sending velocity values to the wheelchair controller.

4.4 Making the wheelchair system kit

4.4.1 Designing the sensor shield

After finalising the sensors to be used, a sensor shield is designed which fits on top of the Arduino microcontroller. The schematic for the sensor shield can be found in Appendix section B1.

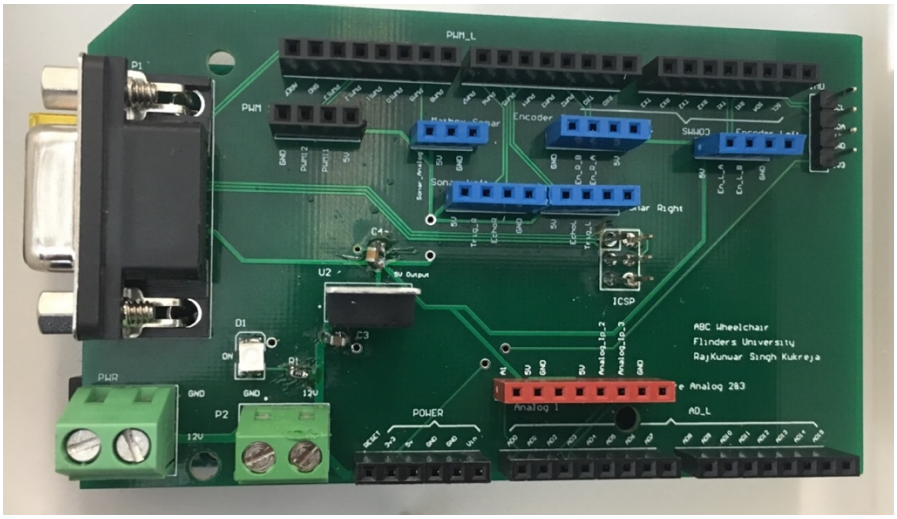


Figure 67. Arduino sensor shield

4.4.2 Designing the box:

To make the wheelchair more presentable and compact, a box was 3d printed which fits the Arduino microcontroller, the shield and the power regulator which powers the entire system.

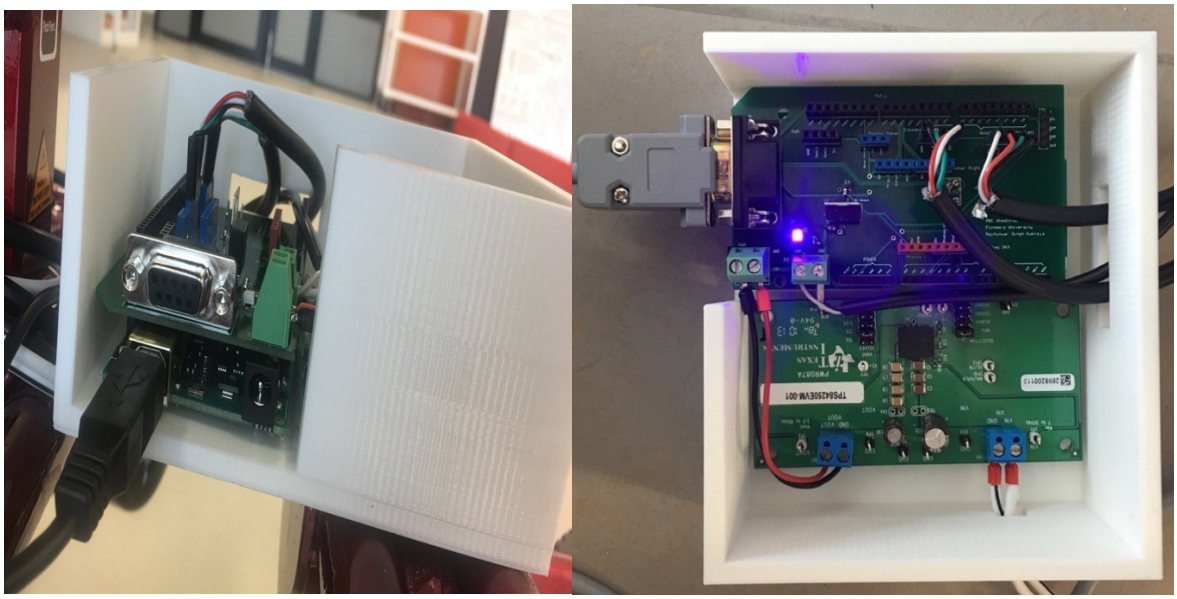


Figure 68. Arduino and sensor shield box

Chapter 5

DISCUSSION

The figures below illustrate the current state of the wheelchair which includes the encoders, a Kinect sensor, the Arduino shield inside the box. The sensor shield is powered through the wheelchair battery which allows it to be manually controlled with the joystick even in the absence of a laptop.

Figure 69. Current state of wheelchair

The sections above discussed the development during the course of the project where the wheelchair now has the capability to create a 3D map of the environment using a cost-effective depth camera which is then collapsed down to a 2D map or a grid map while generating costmaps (global and local) which are used for navigating within the traversable areas. The wheelchair currently accepts controls using the

trackpad on the laptop but have been tested by controlling the cursor using hand-gesture recognition using OpenCV. The maps generated using the sensors provide information about the wheelchair environment but can certainly be improved with further testing and working on the system. The inflation radius for obstacles is currently set high to maintain a certain distance from all obstacles and people as a measure of safety. The inflation radius is the cost of the degradation of the adjacent cells which are occupied. Reducing the inflation cost will generate much cleaner occupancy maps of the environment. Figure's 59 and 61 show a complete map of the entire floor at Tonsley which can be broken down into multiple areas. Segmenting the map based on locations or areas can be better visualized and incorporated to suit the concept of the ABC Wheelchair. As shown in the figure below the multiple maps are generated using the mapping technique used above and then combined to create a global map. The [61] initial and the end nodes of each mapping session are represented as diamonds in the figure above.

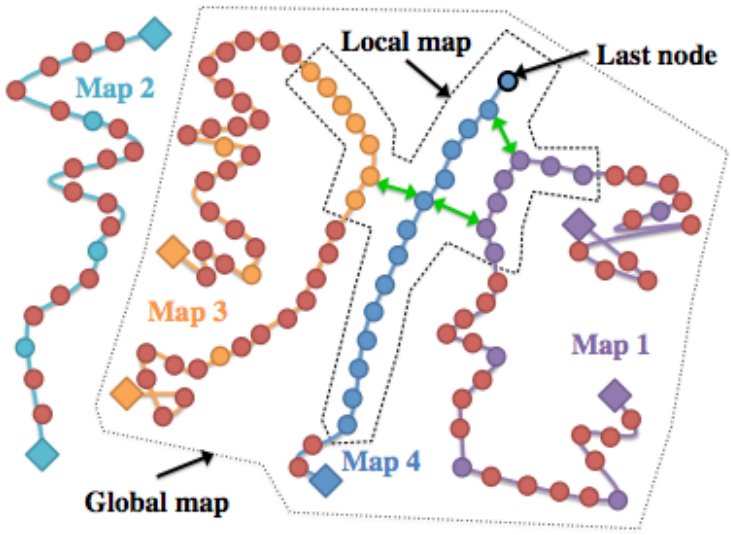


Figure 70. Illustration of Global and local maps through RTAB-map [61]

Multiple tasks described and implemented in chapter 4 such as speech command control, object detection, following mode, navigating to a position on the map can be integrated together with the use of a GUI and finite state machine to generate the interface similar to what was shown in figure or similar to section 4.1.7 where the controls can be replaced with actions and the wheelchair can perform the task.

5.2 Limitations

Kinect Sensor - The Kinect sensor provides RGB-D information about the environment which generates 3D pointclouds and grid maps however, due to the narrow field of view of the sensor there is a lot of information missed about the surroundings while generating maps. The sensor has a blind spot of 1.7m in front of the wheelchair due to which the local costmap is not updated correctly and hence misses obstacles below a certain height (0.1m). This can be improved completely by using a sonar sensor attached to the front of the wheelchair providing a better solution for obstacle avoidance or for updating the local costmaps.

5.3 Further Development

5.3.1 Odometry correction/syncing

As mentioned before, the biggest issue with using dead-reckoning as a method for calculating the robot's position is that the errors accumulate of time and throw the odometry off resulting in an inaccurate position estimation. One way to correct odometry is by using fixed landmarks and using the orientation and distance of the wheelchair with respect to the landmark to estimate the position of the wheelchair and reduce errors.

5.3.2 Generating maps with a Lidar

The current grid maps are generated using depth information which results in excess grid cells being marked occupied. A lidar is a light detection and ranging device which is used for measuring ranges. Some lidars provide a range view of 360degree which will be useful while generating maps as the Kinect sensor has a narrow view which results in missing turns and corner while mapping. While a lidar provides sensor measurement only in two dimensions, the resolution is high and more accurate compared to the fake laser scan generated by the Kinect sensor.

The lidar has not been mechanically attached to the wheelchair but figure 71

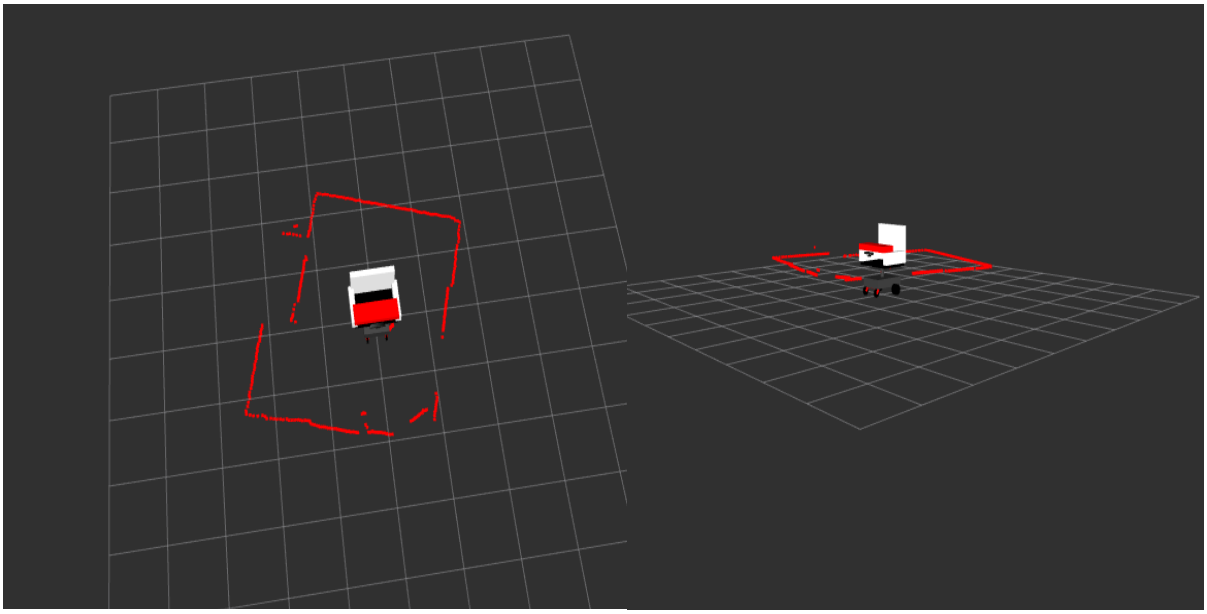


Figure 71. Laserscans using RPLidar

illustrates the output of the laserscans when the lidar is manually held above the wheelchair used for the project. The scans show a highly accurate interpretation of its surrounding and by integrating with the Kinect sensor might generate better maps of the environment and can certainly be used for updating the local and global costmaps to generate a smoother navigational plan.

5.3.3 Creating a follower using an IMU

The follower created in the section 4.3.3 works well to follow the QRcode however has certain limitations. The main limitation being that the QRcode should be in the camera's view the entire time. Since the wheelchair is a mobile robot and travels within a dynamic environment, the follower needs to be adaptable with respect to the changes in the environment and thus a better follower can be created with an IMU using the process of dead-reckoning which estimates the current position based on the previous position and speeds over time. The IMU can be used through a bracelet or an Android phone or any other module publishing acceleration and velocity values from the accelerometer and the gyroscope. The system may provide the solution for a limited time and will have to be calibrated over a short period of time.

5.3.4 Integrating eye tracking control

The idea for sending a goal as shown in section 4.3.9 is not limited to using a trackpad on a laptop, but can be extended to a touch control or integrating the cursor to be controlled using eye-tracking. By incorporating eye-tracking [60] into a system gives the user a new method of input or added control. The Tobii eye tracker can be used with the system developed for the wheelchair which will introduce an additional interaction method for control with the wheelchair interface discussed in section 1. The eye tracker can be used for selecting the options from figure 1 and can be used for sending navigation goals on a map.

Chapter 6

CONCLUSION

The thesis reported the development of an electric wheelchair being converted into an autonomous one with the use of inexpensive and easily accessible sensors using ROS.

Chapter 1 introduces the concept of the ABC wheelchair project at Flinders University and how it can be beneficial for individuals with cognitive, certain physical and mobility impairments.

Chapter 4 contributes to the task of developing a cost-effective solution for building an intelligent wheelchair capable of creating maps of the environment and navigating within using multiple types of inputs. The section also provides background information required for converting an existing wheelchair into an intelligent robotic system.

Chapter 5 discusses the outputs of the experiments and ways of integrating the results from different tasks to create the control method for the interface. Another contribution from the chapter is the further development which can be implemented to improve the functionality of the current system.

The experimental result of the work were positive and provide convincing results to contribute working proof to the concept of the ABC Wheelchair and also meeting the set project goals. The system designed can be integrated with majority of the existing wheelchairs allowing it to be converted into an intelligent one which can enhance lives of individuals who are dependent on others for mobility.

APPENDIX A

A.1 Reading and sending joystick voltage values to the motor controller

```
#include<SPI.h>

#define CS_ADC 10 //chip select pins
#define CS_POT 8
#define channel0 0xE0
#define channel1 0xE8
#define POT_FB 0x12 //00010001
#define POT_FB 0x11 //00010010
#define POT_FB 0x13 //00010011

void setup() {
pinMode(CS_POT, OUTPUT);
pinMode(CS_ADC, OUTPUT);

digitalWrite(CS_ADC,LOW);
digitalWrite(CS_ADC,HIGH);
digitalWrite(CS_POT,HIGH);

Serial.begin(9600);
SPI.begin();
SPI.setBitOrder(MSBFIRST);
SPI.setDataMode(SPI_MODE3);
SPI.setClockDivider(SPI_CLOCK_DIV128);
POTcontrol(POT_ALL,0x80);
delay(2000);
}

void loop()
{
  joystick_control();
  }
int ADCread(byte channel)
{
  int command;
  int voltage;
  byte fbit;
  byte sbit;
  command=channel<<8|0x00;
digitalWrite(CS_ADC,LOW);
voltage=SPI.transfer16(command) & 0x3FF;
digitalWrite(CS_ADC,HIGH);
return voltage;
}
void POTcontrol(byte address,byte value)
{
digitalWrite(CS_POT,LOW);
SPI.transfer(address);
```

```
SPI.transfer(value);
digitalWrite(CS_POT,HIGH);
}

byte joystick_control()
{
  int turnjval;
  int drivejval;
  byte turnpot;
  byte drivepot;
  float mult= 0.85;
  turnjval=ADCread(channel0);
  turnpot=(turnjval-100)*mult;
  drivejval=ADCread(channel1);
  drivepot=(drivejval-100)*mult;
  POTcontrol(POT_FB,drivepot);
  POTcontrol(POT_LR,turnpot);
  return 1;
}
```

A.2 Using a keyboard to control the wheelchair

```
void forward(byte speed)
{
POTcontrol(POT_FB,speed);
}
void reverse(byte speed)
{
POTcontrol(POT_FB,speed);
}
void right(byte speed)
{
POTcontrol(POT_LR,speed);
}
void left(byte speed)
{
POTcontrol(POT_LR,speed);
}
void stop_wheelchair()
{
POTcontrol(POT_ALL,stop);
}
byte keyboard_control()
{
  byte read;
if(Serial.available())
  read=Serial.read();
  if(read==119) //w - Forwad
  {
  forward(linear_speed);
  }
  else if(read==97) //a   - Left
  {
  left(rotational_speed);

  }
else  if(read==115) //s   - Reverse
  {
  reverse(254 - linear_speed)
  }
else  if(read==100) //d   - Right
  {
  right(254 - rotational_speed);
  }
  else  if(read==32) //space    - Stop
  {
stop_wheelchair();
  }
return 1;
}
```

A.3 Moving Wheelchair in a sequence

```
void loop()
{

forward(linear_speed);
pause(5000);
left(rotational_speed);
pause(2000);
forward(linear_speed);
pause(5000);
left(rotational_speed);
pause(2000);
forward(linear_speed);
pause(5000);
left(rotational_speed);
pause(2000);
forward(linear_speed);
pause(2500);
stop_wheelchair();
}
```

A.4 Adding voice commands to the program.

```
#include<SPI.h>
#include <BitVoicer11.h>

BitVoicerSerial bvSerial = BitVoicerSerial();
byte voicecommand() {
 bvSerial.getData();
  //Quits the loop if no string data was returned from getData
  if (bvSerial.strData == "")
    return;
  //Each of the next 'if' statements performs a different task
based on the data received from BitVoicer
  if (bvSerial.strData == "fw") //Forward
  {
    POTcontrol(POT_FB,0xFB);
    POTcontrol(POT_LR,0x80);
    bvSerial.strData = "";
  }
  else if (bvSerial.strData == "rv")   //Reverse
  {
 POTcontrol(POT_FB,0x08);
  POTcontrol(POT_LR,0x80);
    bvSerial.strData = "";
  }
  else if (bvSerial.strData == "rt")         //Right
  {
      POTcontrol(POT_FB,0x80);
  POTcontrol(POT_LR,0xFB);
    bvSerial.strData = "";
  }
  else if (bvSerial.strData == "lt")         //Left
  {
     POTcontrol(POT_FB,0x80);
    POTcontrol(POT_LR,0x0A);
    bvSerial.strData = "";
  }
  else if (bvSerial.strData == "st")   //Stop
  {
      POTcontrol(POT_FB,0x80);
      POTcontrol(POT_LR,0x80);
    bvSerial.strData = "";
  }
  else  {
    Serial.println("ERROR:" + bvSerial.strData);
    bvSerial.strData = "";
  }
}
```

A.5 Calculating turn angles

```
void angle()
{
  int fbpotval;
  int lrpotval;
  byte angle;
  byte temp;
  Serial.println("Enter Heading");
  if(Serial.available() > 0)
  angle=Serial.parseInt();
  Serial.println(angle);
  temp=lrpotval;
  if(angle<90)
  {
  fbpotval=(398-(angle*1.6));
  lrpotval=(254+(angle*1.6));
  }
 else if(angle>90)
 {
  angle=angle-90;
  fbpotval=(398-(angle*1.6));
  lrpotval=(254-(angle*1.6));
 }
if(temp!=lrpotval || lrpotval!=254)
{
 POTcontrol(POT_FB,(fbpotval-100)*0.85);
 POTcontrol(POT_LR,(lrpotval-100)*0.85);
 Serial.println(fbpotval);
 Serial.println(lrpotval);
}
else
 POTcontrol(POT_ALL,128);
  delay(5000);
}
```

A.6 Avoiding obstacles safely

```
void autowheelchair()
{
int fullspeed=398;
int stoppot=254;
int slow sleep=326;
int revfullspeed=110;
jstickval=ADCread(channel1);           //for safety
  if(jstickval>260 || jstickval<240)
joystick_control();
else
volt=analogRead(SonarPin);      //distance reading from sonar
dist=volt*5;
if(dist>1000)              //drive forward if no obstacle
infront
{
forward(fullspeed);
left(stoppot);
}
else if(dist<1000)        //if distance is less than threshold,
calculate turnangle
{
  turnangle=45+(10-(dist/100))*5.625;
  turnangleval=turnangle*1.711;
  turnpotval=254+turnangleval;
forward(slowspeed);
POTcontrol(POT_LR,(turnpotval-100)*0.85);
}
else if(dist<300)  //if the wheelchair is very close to
obstacle
{                  // go reverse
reverse(revfullspeed);
left(stoppot);
}
}
```

A.7 Object detection using opencv [3]

```
import cv2
import numpy as np
MIN_MATCH=30
detector = cv2.SIFT()
FLANN_INDEX_KDITREE=0
flannParam=dict(algorithm=FLANN_INDEX_KDITREE,tree=5)
flann = cv2.FlannBasedMatcher(flannParam,{})
trainImg=cv2.imread('ObjectDetection/Door',0)
trainKP,trainDes=detector.detectAndCompute(trainImg,None)
cam=cv2.VideoCapture(1)
while True:
    ret,qimgrgb=cam.read()
    qimg=cv2.cvtColor(qimgrgb,cv2.COLOR_BGR2GRAY)
    qkp,qdes=detector.detectAndCompute(qimg,None)
    matches=flann.knnMatch(qdes,trainDes,k=2)

    for m,n in matches:
        if(m.distance<0.75*n.distance):
            goodMatch.append(m)
            print"Door Found"
    if(len(goodMatch>Min_Match)):
       tp=[]
       qp=[]
       for m in goodMatch:
           tp.append(trainKP[m.trainIdx].pt)
           qp.append(qkp[m.qIdx].pt)
            tp,qp = np.float32((tp,np))
            H,status= findHomography(tp,qp,cv2.RANSAC,3.0)
        h,w=trainImg.shape
       trainingBorder=np.float32([[[0,0],[0,h-1],[w-1,h-1],[w-
1,0]]])
       qBorder=cv2.perspectiveTransform(trainingBorder,H)

cv2.polylines(qimgrbg,[np.int32(qBorder)],True,(0,255,0),5)
       else:
           print"Not Enough Matches -
%d/%d"%(len(goodMatch),MIN_MATCH)
           cv2.imshow('result',qimgrgb)
           ifcv2.waitKey(10)==ord('q'):
               break

            cam.release()
            cv2.destroyAllWindows()
```

A.8 Integration of ROS with Arduino

```
#include<SPI.h>
#include "ros.h"
#include "geometry_msgs/Twist.h"
int jstickval;
float x;
float y;

ros::NodeHandle nh;
void velCallback(const geometry_msgs::Twist& vel)
{
     x = vel.linear.x;
     y = vel.angular.z;
}
ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel" ,
velCallback);

void setup() {
pinMode(CS_POT, OUTPUT);
pinMode(CS_ADC, OUTPUT);
digitalWrite(CS_ADC,LOW);
digitalWrite(CS_ADC,HIGH);
digitalWrite(CS_POT,HIGH);
Serial.begin(9600);
SPI.begin();

SPI.setBitOrder(MSBFIRST);
SPI.setDataMode(SPI_MODE3);
SPI.setClockDivider(SPI_CLOCK_DIV128);

POTcontrol(POT_ALL,0x80);
delay(2000);
 nh.initNode();
 nh.subscribe(sub);
}
void loop()
{
jstickval=ADCread(channel1);
  if(jstickval>260 || jstickval<240)
       joystick_control();
  else
       roscontrol();
}
int ADCread(byte channel)
{
  int command;
  int voltage;
  byte fbit;
  byte sbit;
```

```
  command=channel<<8|0x00;
digitalWrite(CS_ADC,LOW);
voltage=SPI.transfer16(command) & 0x3FF;
digitalWrite(CS_ADC,HIGH);
return voltage;
}
void POTcontrol(byte address,byte value)
{
digitalWrite(CS_POT,LOW);
SPI.transfer(address);
SPI.transfer(value);
digitalWrite(CS_POT,HIGH);
}
byte joystick_control()
{
  int turnjval;
  int drivejval;
  byte turnpot;
  byte drivepot;
  float mult= 0.85;
  turnjval=ADCread(channel0);
  turnpot=(turnjval-100)*mult;
  drivejval=ADCread(channel1);
  drivepot=(drivejval-100)*mult;
  POTcontrol(POT_FB,drivepot);
  POTcontrol(POT_LR,turnpot);
  return 1;
}
void roscontrol()
{
  if (x>0)     //forward
  {
  POTcontrol(POT_FB,0xFB);
  POTcontrol(POT_LR,0x80);
  }
  else if (x<0) //reverse
  {
 POTcontrol(POT_FB,0x08);
 POTcontrol(POT_LR,0x80);
  }
  else if (y<0) //right
  {
  POTcontrol(POT_FB,0x80);
  POTcontrol(POT_LR,0xFB);
  }
  else if (y>0) //left
  {
    POTcontrol(POT_FB,0x80);
    POTcontrol(POT_LR,0x08);
  }
}
```

A.9 Generating Wheel odometry

```
void compute_odometry()
{
right_pulses = R_count ;
left_pulses = L_count;
right_ticks = right_pulses - prev_right_pulses;
left_ticks = left_pulses - prev_left_pulses;
right_wheel_distance = right_ticks * distance_per_tick;
left_wheel_distance = left_ticks * distance_per_tick;

mean_distance = (left_wheel_distance +
right_wheel_distance)/2;
wheel_theta += (right_wheel_distance -
left_wheel_distance)/wheel_base;

//Limiting value of theta between Pi and -Pi
if(wheel_theta < -PI)
wheel_theta= PI;
if(wheel_theta > PI)
wheel_theta = -PI;
x_pos += mean_distance * cos(wheel_theta);
y_pos += mean_distance * sin(wheel_theta);
}
//Interrupt routines to read encoder pulses
void left_interrupt()
{
  char i;
  i=digitalRead(L_PHASE_B);
  if(i)
  L_count +=1;
  else
  L_count -= 1;
}
void right_interrupt()
{
  char i;
  i=digitalRead(R_PHASE_B);
  if(i)
  R_count +=1;
  else
  R_count -= 1;
}
```

A.10 Landmark Navigation

```
void navigation()
{
  int junk;
  while(!goal_x)
{
  Serial.println("Enter Goal x");
  while(!Serial.available());
  goal_x=Serial.parseFloat();
  Serial.flush();
}
while(!goal_y)
{
  Serial.println("Enter Goal y");
  Serial.flush();
  while(!Serial.available());
  goal_y=Serial.parseFloat();
Serial.print("Destination:");Serial.print(goal_x);Serial.print
(",");Serial.println(goal_y);
}
compute_odometry(); // generate odometry
diff_x = goal_x - x_pos;
diff_y = goal_y - y_pos;
diff_dist = sqrt((diff_x)^2 + (diff_y)^2);
diff_heading = atan2(diff_y,diff_x);
if(diff_dist>0.5)
forward();
if((wheel_theta - diff_heading) > 0.3)
  left();
if((wheel_theta - diff_heading) < -0.3)
  right();
Serial.print("Distance to goal:   ");
Serial.print(diff_dist);
Serial.println(" m");
delay(500);
  if(diff_dist <0.5)
  {
    stop_wheelchair();
    Serial.print("Goal Reached");
    goal_x=0;
    goal_y=0;
  }
}
```

A.11 Computing odometry from encoders and IMU

```
/*
Advanced_I2C.ino
Brian R Taylor
brian.taylor@bolderflight.com
Copyright (c) 2017 Bolder Flight Systems
Permission is hereby granted, free of charge, to any person
obtaining a copy of this software and associated documentation
files (the "Software"), to deal in the Software without
restriction, including without limitation the rights to use,
copy, modify, merge, publish, distribute, sublicense, and/or
sell copies of the Software, and to permit persons to whom the
Software is furnished to do so, subject to the following
conditions:
The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the
Software.
*/

float get_imu_theta()
{
  IMU.readSensor();
  gZ =  IMU.getGyroZ_rads();
return gZ;
}

void compute_odom()
{
right_pulses=R_count;
left_pulses=L_count;
current_time = millis();
imu_theta_rate = get_imu_theta();
right_ticks = right_pulses - prev_right_pulses;
left_ticks = left_pulses - prev_left_pulses;
right_wheel_distance = right_ticks * right_distance_per_tick;
left_wheel_distance = left_ticks * left_distance_per_tick;
mean_distance = (left_wheel_distance +
right_wheel_distance)/2;
diff_time = current_time - previous_time;
imu_theta -= imu_theta_rate*diff_time;
theta_global = imu_theta/1000;
if(theta_global< -PI)
theta_global+= 2*PI;
else if(theta_global > PI)
theta_global -= 2*PI;
x_pos += (mean_distance * cos(theta_global))*1.0/1000;
y_pos += (mean_distance * sin(theta_global))*1.0/1000;

prev_left_pulses=left_pulses;
prev_right_pulses=right_pulses;
previous_time=current_time;
}
```

A.12 Wheelchair URDF model

```
<?xml version='1.0'?>
  <link name='chassis'>
    <pose>0 0 0.12 0 0 0</pose>

    <inertial>
      <mass value="7.0"/>
      <origin xyz="0.0 0 0.1" rpy=" 0 0 0"/>
      <inertia
          ixx="0.5" ixy="0" ixz="0"
          iyy="1.0" iyz="0"
          izz="0.1"
      />
    </inertial>

    <collision name='collision'>
      <geometry>
        <box size=".8 .8 .1"/>
      </geometry>
    </collision>

    <visual name='chassis_visual'>
      <origin xyz="0 0 0" rpy=" 0 0 0"/>
      <geometry>
        <box size=".8 .8 .1"/>
      </geometry>
    </visual>

    <collision name='caster_collision'>
      <origin xyz="-0.18 0 -0.05" rpy=" 0 0 0"/>
      <geometry>
        <sphere radius="0.13"/>
      </geometry>
      <surface>
        <friction>
          <ode>
            <mu>0</mu>
            <mu2>0</mu2>
            <slip1>1.0</slip1>
            <slip2>1.0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>

    <visual name='caster_visual'>
      <origin xyz="-0.18 0 -0.05" rpy=" 0 0 0"/>
      <geometry>
        <sphere radius="0.13"/>
```

```
    </geometry>
  </visual>
</link>

<link name="left_wheel">
  <!--origin xyz="0.1 0.13 0.1" rpy="0 1.5707 1.5707"/-->
  <collision name="collision">
    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
    <geometry>
      <cylinder radius="0.18" length="0.05"/>
    </geometry>
  </collision>
  <visual name="left_wheel_visual">
    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
    <geometry>
      <cylinder radius="0.18" length="0.05"/>
    </geometry>
  </visual>
  <inertial>
    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
    <mass value="10"/>
    <cylinder_inertia m="10" r="0.1" h="0.05"/>
    <inertia
      ixx="1.0" ixy="0.0" ixz="0.0"
      iyy="1.0" iyz="0.0"
      izz="1.0"/>
  </inertial>
</link>

<link name="right_wheel">
  <!--origin xyz="0.1 -0.13 0.1" rpy="0 1.5707 1.5707"/-->
  <collision name="collision">
    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
    <geometry>
      <cylinder radius="0.18" length="0.05"/>
    </geometry>
  </collision>
  <visual name="right_wheel_visual">
    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
    <geometry>
      <cylinder radius="0.18" length="0.05"/>
    </geometry>
  </visual>
  <inertial>
    <origin xyz="0 0 0" rpy="0 1.5707 1.5707"/>
    <mass value="10"/>
    <cylinder_inertia m="10" r="0.1" h="0.05"/>
    <inertia
      ixx="1.0" ixy="0.0" ixz="0.0"
      iyy="1.0" iyz="0.0"
      izz="1.0"/>
```

```
    </inertial>
  </link>


  <joint type="continuous" name="left_wheel_hinge">
    <origin xyz="0.1 0.45 0" rpy="0 0 0"/>
    <child link="left_wheel"/>
    <parent link="chassis"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

  <joint type="continuous" name="right_wheel_hinge">
    <origin xyz="0.1 -0.45 0" rpy="0 0 0"/>
    <child link="right_wheel"/>
    <parent link="chassis"/>
    <axis xyz="0 1 0" rpy="0 0 0"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

<!--link name="chassis_rod">
  <visual>
       <geometry>
        <cylinder length="0.3" radius="0.1"/>
          <origin rpy="0 0 0" xyz="0 0 0.15"/>
       </geometry>
  </visual>
</link>
<joint name="base to chassis rod" type="fixed"/>
 <parent link="chassis"/>
  <child link="chassis_rod"/>
   <origin xyz="0 0 0.15"/>
</joint-->

<link name="base_rod">
    <!--origin xyz="0.1 -0.13 0.1" rpy="0 1.5707 1.5707"/-->
    <collision name="collision">
      <origin xyz="0 0 0.12" rpy="0 0 0"/>
      <geometry>
        <cylinder radius="0.04" length="0.4"/>
      </geometry>
    </collision>
    <visual name="base_rod_visual">
      <origin xyz="0 0 0.12" rpy="0 0 0"/>
      <geometry>
        <cylinder radius="0.04" length="0.4"/>
      </geometry>
    </visual>
  </link>
```

```
  <joint type="fixed" name="base_to_chassis_rod">
    <origin xyz="0.0 0.0 0.1" rpy="0 0 0"/>
    <child link="base_rod"/>
    <parent link="chassis"/>
  </joint>

<link name="base_chair">
    <collision name="collision">
      <origin xyz="0 0 0.15" rpy="0 0 0"/>
      <geometry>
        <box size=".6 .8 .05"/>
      </geometry>
    </collision>
    <visual name="base_chair_visual">
      <origin xyz="0 0 0.15" rpy="0 0 0"/>
      <geometry>
        <box size=".6 .8 .05"/>
      </geometry>
    </visual>
  </link>

  <joint type="fixed" name="base_to_chair">
    <origin xyz="0.0 0.0 0.15" rpy="0 0 0"/>
    <child link="base_chair"/>
    <parent link="base_rod"/>
  </joint>

<link name="chair_rest">
    <collision name="collision">
      <origin xyz="-0.15 0 0.24" rpy="0 0 0"/>
      <geometry>
        <box size=".1 .8 .6"/>
      </geometry>
    </collision>
    <visual name="chair_back_visual">
      <origin xyz="-0.15 0 0.24" rpy="0 0 0"/>
      <geometry>
        <box size=".1 .8 .6"/>
      </geometry>
    </visual>
  </link>
  <joint type="fixed" name="base_chair_back">
    <origin xyz="-0.15 0.0 0.24" rpy="0 0 0"/>
    <child link="chair_rest"/>
    <parent link="base_chair"/>
  </joint>

<link name="left_arm_rest">
    <collision name="collision">
      <origin xyz="0 0.2 0.14" rpy="0 0 0"/>
```

```
      <geometry>
        <box size="0.6 .1 .2"/>
      </geometry>
    </collision>
    <visual name="left_arm_rest_visual">
      <origin xyz="0 0.2 0.14" rpy="0 0 0"/>
      <geometry>
        <box size="0.6 .1 .2"/>
      </geometry>
    </visual>
  </link>


  <joint type="fixed" name="chair_left_arm">
    <origin xyz="0 0.2 0.14" rpy="0 0 0"/>
    <child link="left_arm_rest"/>
    <parent link="base_chair"/>
  </joint>

<link name="right_arm_rest">
    <collision name="collision">
      <origin xyz="0 -0.2 0.14" rpy="0 0 0"/>
      <geometry>
        <box size="0.6 .1 .2"/>
      </geometry>
    </collision>
    <visual name="right_arm_rest_visual">
      <origin xyz="0 -0.2 0.14" rpy="0 0 0"/>
      <geometry>
        <box size="0.6 .1 .2"/>
      </geometry>
    </visual>
  </link>


  <joint type="fixed" name="right_left_arm">
    <origin xyz="0 -0.2 0.14" rpy="0 0 0"/>
    <child link="right_arm_rest"/>
    <parent link="base_chair"/>
  </joint>

<link name="wheelchair_tray">
    <collision name="collision">
      <origin xyz="0.1 0.1 0.145" rpy="0 0 0"/>
      <geometry>
        <box size="0.3 .8 .1"/>
      </geometry>
    </collision>
    <visual name="wheelchair_tray_visual">
      <origin xyz="0.1 0.1 0.145" rpy="0 0 0"/>
      <geometry>
```

```
        <box size="0.3 .8 .1"/>
      </geometry>
    </visual>
  </link>

  <joint type="fixed" name="wheelchair_tray_joint">
    <origin xyz="0.1 0.1 0.145" rpy="0 0 0"/>
    <child link="wheelchair_tray"/>
    <parent link="right_arm_rest"/>
  </joint>

<link name="kinect">
    <collision name="collision">
      <origin xyz="0.15 0.05.05"/>
      </geometry>
    </visual>
  </link>

  <joint type="fixed" name="kinect_wheelchair_tray_joint">
    <origin xyz="0.15 0.05 0.06" rpy="0 0 0"/>
    <child link="kinect"/>
    <parent link="wheelchair_tray"/>
  </joint>

<link name="sonar_link">
    <collision name="collision">
      <origin xyz="0.1 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.01 0.03 0.05"/>
<mesh
filename="package://hector_sensors_description/meshes/sonar_se
nsor/max_sonar_ez4.dae"/>
      </geometry>
    </collision>
    <visual name="sonar_visual">
      <origin xyz="0.1 0 0" rpy="0 0 0"/>
      <geometry>
        <box size="0.01 0.03 0.05"/>
      </geometry>
    </visual>
  </link>

  <joint type="fixed" name="sonar_rod_joint">
    <origin xyz="0.1 0 0" rpy="0 0 0"/>
    <child link="sonar_link"/>
    <parent link="base_rod"/>
  </joint>
</robot>
```

Gazebo Model

```
<?xml version="1.0"?>
```

```
<robot>
  <gazebo>
    <plugin name="differential_drive_controller"
filename="libgazebo_ros_diff_drive.so">
      <legacyMode>false</legacyMode>
      <alwaysOn>true</alwaysOn>
      <updateRate>20</updateRate>
      <leftJoint>left_wheel_hinge</leftJoint>
      <rightJoint>right_wheel_hinge</rightJoint>
      <wheelSeparation>0.4</wheelSeparation>
      <wheelDiameter>0.18</wheelDiameter>
      <torque>20</torque>
      <commandTopipenni_kinect.so">
          <cameraName>camera</cameraName>
          <alwaysOn>true</alwaysOn>
          <updateRate>10</updateRate>
          <imageTopicName>rgb/image_raw</imageTopicName>

<depthImageTopicName>depth/image_raw</depthImageTopicName>

<pointCloudTopicName>depth/points</pointCloudTopicName>

<cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>

<depthImageCameraInfoTopicName>depth/camera_info</depthImageCa
meraInfoTopicName>
          <frameName>camera</frameName>
          <baseline>0.1</baseline>
          <distortion_k1>0.0</distortion_k1>
          <distortion_k2>0.0</distortion_k2>
          <distortion_k3>0.0</distortion_k3>
          <distortion_t1>0.0</distortion_t1>
          <distortion_t2>0.0</distortion_t2>
          <pointCloudCutoff>0.4</pointCloudCutoff>
        </plugin>
      </sensor>
    </gazebo>

<gazebo reference="sonar_link">
      <sensor type="ray" name="sonar">
        <always_on>true</always_on>
        <update_rate>10</update_rate>
        <pose>0 0 0 0 0 0</pose>
        <visualize>true</visualize>
        <ray>
          <scan>
            <horizontal>
              <samples>2</samples>
              <resolution>1</resolution>
              <min_angle>-0.008552113</min_angle>
              <max_angle>-0.008552113</max_angle>
```

```
            </horizontal>
            <vertical>
              <samples>2</samples>
              <resolution>1</resolution>
              <min_angle>-0.008552113</min_angle>
              <max_angle>0.008552113</max_angle>
            </vertical>
          </scan>
          <range>
            <min>0.15</min>
            <max>2</max>
            <resolution>0.01</resolution>
          </range>
        </ray>

        <plugin name="gazebo_ros_sonar_controller"
filename="libhector_gazebo_ros_sonar.so">
          <gaussianNoise>0.005</gaussianNoise>
          <topicName>sonar/distance</topicName>
          <frameId>sonar_link</frameId>
        </plugin>
      </sensor>
    </gazebo>

  <gazebo reference="chassis">
      <material>Gazebo/Red</material>
    </gazebo>
<gazebo reference="base_rod">
      <material>Gazebo/Grey</material>
    </gazebo>
<gazebo reference="base_chair">
      <material>Gazebo/Black</material>
    </gazebo>
<gazebo reference="chair_rest">
      <material>Gazebo/Black</material>
    </gazebo>
<gazebo reference="left_arm_rest">
      <material>Gazebo/Red</material>
    </gazebo>
<gazebo reference="right_arm_rest">
      <material>Gazebo/Red</material>
    </gazebo>
</robot>
```

A.13 Wheelchair Follower

```
#!/usr/bin/env python
import math
import rospy
from geometry_msgs.msg import PoseStamped, Point , Twist
from std_msgs.msg import Int8
x=0.0
z=0.0
status=1;

def callback(msg):
	global previous_x
	global previous_z
	global x
	global z
	previous_x=x
	previous_z=z
	x = msg.pose.position.x
	z = msg.pose.position.z

def status(msg):
	global status
	status=msg.data

rospy.init_node('follower')
rospy.Subscriber("/visp_auto_tracker/object_position",PoseStam
ped,callback)
pub=rospy.Publisher("cmd_vel",Twist,queue_size=1)
rospy.Subscriber("/visp_auto_tracker/status",Int8,status)

speed=Twist()
r=rospy.Rate(10)
previous_x=x
previous_z=z
while not rospy.is_shutdown():
	if(status==3):
		rospy.loginfo_throttle(30,"Object Found,
Following Object")
		diff_x=previous_x-x
		diff_z=previous_z-z
		speed.linear.x= (diff_z*1000)
		speed.angular.z= (diff_x*1000)
	elif(status!=3):
		rospy.loginfo_throttle(30,"Searcing for Object")
		speed.angular.z= 0.0
		speed.linear.x= 0.0
	pub.publish(speed)
	r.sleep()
```

A.14 Velocity Controller

```
void Callback(const geometry_msgs::Twist& vel)
{
double linear_vel= vel.linear.x;
double angular_vel= vel.angular.z;
int rotational_speed;
int linear_speed_sig;
double linear_speed_dec;
int linear_speed;
int linear_speed_val;
int fb_pot_val;
int lr_pot_val;

//Velocity values are of double data type so have to be
converted to integer
if(!checkjoystick())
{
  angular_vel = angular_vel * 57.2958; //convert to degrees
  rotational_speed = int(254 - angular_vel*1.6);//convert to
pot_val

  linear_speed_sig = int(linear_vel/1); //Get integer part
  linear_speed_dec = linear_vel - linear_speed; //get decimal
part
  linear_speed = int(linear_speed_sig*100 +
linear_speed_dec*100);//combine integer and decimal
  linear_speed_val = (254 +linear_speed*1.6);//convert to
pot_val

    if(linear_speed_val > 398)
    linear_speed_val=398;
    if(linear_speed_val<10)
    linear_speed_val=10;
    if(rotational_speed>398)
    rotational_speed=398;
    if(rotational_speed<10)
    rotational_speed=10;

    fb_pot_val=(linear_speed_val-100)*0.85;
    lr_pot_val=(rotational_speed-100)*0.85;
POTcontrol(POT_FB,fb_pot_val);
POTcontrol(POT_LR,lr_pot_val);

      }
}

ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel",Callback);
```

A.15 Mapping with RTAB-Map

```
<?xml version="1.0"?>
<launch>
<!--Launch rosserial for arduino-->
<arg name="port"  default="/dev/ttyACM0" />
<node name="serial_node" pkg="rosserial_python"
type="serial_node.py">
            <param name="port" value="$(arg port)"/>
            <param name="baud" value="57600" />
</node>

<param name="robot_description" command="$(find
xacro)/xacro.py '$(find
mybot_description)/urdf/mybot.xacro'"/>

  <!-- convert tf to odom -->
<node pkg= "tf_to_odometry" name="tf_to_odometry"
type="tf_to_odometry" output="screen">
</node>

  <!-- send fake joint values -->
  <node name="joint_state_publisher"
pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui" value="False"/>
  </node>
  <!-- Combine joint values -->
  <node name="robot_state_publisher"
pkg="robot_state_publisher" type="state_publisher"/>

<!--Launch openni node with depth_registration := true and
publish_tf:=false-->
<include file="$(find wheelchair)/launch/kinect.launch"/>

  <!-- Move base -->
<include file="$(find
wheelchair)/launch/includes/move_base_simulation.launch.xml"/>

<node pkg="rtabmap_ros" type="rtabmap" name="rtabmap">
<!--RTAB-Map's parameters -->
          <param name="RGBD/ProximityBySpace"
type="string" value="false"/>
          <param name="RGBD/AngularUpdate"
type="string" value="0.01"/>
          <param name="RGBD/LinearUpdate"
type="string" value="0.01"/>
          <param name="RGBD/OptimizeFromGraphEnd"
type="string" value="false"/>
          <param name="Optimizer/Slam2D"
type="string" value="true"/>
```

```
          <param name="Reg/Strategy"
type="string" value="1"/> <!-- 1=ICP -->
          <param name="Reg/Force3DoF"
type="string" value="false"/>
          <param name="Vis/MaxDepth"
type="string" value="4.0"/>
          <param name="Vis/MinInliers"
type="string" value="5"/>
          <param name="Vis/InlierDistance"
type="string" value="0.05"/>
          <param name="Rtabmap/TimeThr"
type="string" value="700"/>
          <param name="Mem/RehearsalSimilarity"
type="string" value="0.45"/>
          <param name="Icp/CorrespondenceRatio"
type="string" value="0.5"/>
       <param name="map_negative_poses_ignored" type="bool"
value="true"/>
       <param name="map_filter_radius"         type="double"
value="0.5"/>
         <param name="map_cleanup"             type="bool"
value="false"/>
         <param name="grid_unknown_space_filled" type="bool"
value="true"/>
         <param name="Grid/FromDepth"          type="bool"
value="true"/>
         <param name="Grid/3D"                 type="bool"
value="false"/>
         <param name="Grid/CellSize"             type="double"
value="0.7"/>
         <param name="Grid/RangeMax"             type="double"
value="4.0"/>
         <param name="Grid/RayTracing"           type="string"
value="true"/>
          <param name="RGBD/CreateOccupancyGrid"
type="string" value="true"/>
</node>

  <!-- Choose visualization -->
  <arg name="rviz"                    default="true" />
  <arg name="rtabmapviz"              default="false" />
  <!-- Localization-only mode -->
  <arg name="localization"            default="false"/>
  <!-- Corresponding config files -->
  <arg name="rtabmapviz_cfg"
default="~/.ros/rtabmap_gui.ini" />
  <arg name="rviz_cfg"                default="$(find
wheelchair)/config/rtmap.rviz" />
  <arg name="frame_id"                default="chassis"/>
<!-- Fixed frame id, you may set "base_link" or
"base_footprint" if they are published -->
```

```
  <arg name="database_path"
default="~/.ros/rtabmap.db"/>
  <arg name="rtabmap_args"              default=""/>
<!--delete_db_on_start, udebug -->
  <arg name="launch_prefix"             default=""/>
<!-- for debugging purpose, it fills launch-prefix tag of the
nodes -->
  <arg name="approx_sync"               default="true"/>
<!-- if timestamps of the input topics are not synchronized --
>

  <arg name="rgb_topic"
default="/camera/rgb/image_rect_color" />
  <arg name="depth_registered_topic"
default="/camera/depth_registered/image_raw"/>
  <arg name="camera_info_topic"
default="/camera/rgb/camera_info" />
  <arg name="compressed"                default="false"/>

  <arg name="subscribe_scan"            default="false"/>
<!-- Assuming 2D scan if set, rtabmap will do 3DoF mapping
instead of 6DoF -->
  <arg name="scan_topic"                default="/scan"/>
  <arg name="subscribe_scan_cloud"      default="false"/>
<!-- Assuming 3D scan if set -->
  <arg name="scan_cloud_topic"          default="/scan_cloud"/>
  <arg name="visual_odometry"           default="false"/>
<!-- Generate visual odometry -->
  <arg name="odom_topic"                default="/wheel_odom"/>
<!-- Odometry topic used if visual_odometry is false -->
  <arg name="odom_frame_id"             default="odom"/>
<!-- If set, TF is used to get odometry instead of the topic -
->
  <arg name="namespace"                 default="rtabmap"/>
  <arg name="wait_for_transform"        default="0.2"/>


<!--Creating fake laser scan from kinect sensor-->
<node name="depthimage_to_laserscan"
pkg="depthimage_to_laserscan" type="depthimage_to_laserscan" >
  <remap from="image" to="$(arg depth_registered_topic)"/>
<rosparam>
scan_height : 190
range_min : 0.3
range_max : 4
</rosparam>
</node>

  <include file="$(find rtabmap_ros)/launch/rtabmap.launch">
    <arg name="rtabmapviz"              value="$(arg
rtabmapviz)" />
```

```
    <arg name="rviz"                    value="$(arg rviz)" />
    <arg name="localization"            value="$(arg
localization)"/>
    <arg name="gui_cfg"                 value="$(arg
rtabmapviz_cfg)" />
    <arg name="rviz_cfg"                value="$(arg
rviz_cfg)" />
    <arg name="frame_id"                value="$(arg
frame_id)"/>
    <arg name="namespace"               value="$(arg
namespace)"/>
    <arg name="database_path"           value="$(arg
database_path)"/>
    <arg name="wait_for_transform"      value="$(arg
wait_for_transform)"/>
    <arg name="rtabmap_args"            value="$(arg
rtabmap_args)"/>
    <arg name="launch_prefix"           value="$(arg
launch_prefix)"/>
    <arg name="approx_sync"             value="$(arg
approx_sync)"/>
    <arg name="rgb_topic"               value="$(arg
rgb_topic)" />
    <arg name="depth_topic"             value="$(arg
depth_registered_topic)" />
    <arg name="camera_info_topic"       value="$(arg
camera_info_topic)" />
    <arg name="compressed"              value="$(arg
compressed)"/>
    <arg name="subscribe_scan"          value="$(arg
subscribe_scan)"/>
    <arg name="scan_topic"              value="$(arg
scan_topic)"/>
    <arg name="subscribe_scan_cloud"    value="$(arg
subscribe_scan_cloud)"/>
    <arg name="scan_cloud_topic"        value="$(arg
scan_cloud_topic)"/>
    <arg name="visual_odometry"         value="$(arg
visual_odometry)"/>
    <arg name="odom_topic"              value="$(arg
odom_topic)"/>
    <arg name="odom_frame_id"           value="$(arg
odom_frame_id)"/>
    <arg name="odom_args"               value="$(arg
rtabmap_args)"/>
  </include>

 <!-- Attaching to nodelet manager from OpenNI:
camera_nodelet_manager -->
```

```
    <node pkg="nodelet" type="nodelet" name="data_throttle"
args="load rtabmap/data_throttle camera_nodelet_manager"
output="screen">
      <param name="rate" type="double" value="5.0"/>
      <param name="decimation" type="int" value="1"/>

      <remap from="rgb/image_in"
to="/camera/rgb/image_rect_color"/>
      <remap from="depth/image_in"
to="/camera/depth_registered/image_raw"/>
      <remap from="rgb/camera_info_in"
to="/camera/rgb/camera_info"/>
      <remap from="rgb/image_out"
to="/camera/rgb/image_rect_color_throttle"/>
      <remap from="depth/image_out"
to="/camera/depth_registered/image_raw_throttle"/>
      <remap from="rgb/camera_info_out"
to="/camera/rgb/camera_info_throttle"/>
    </node>
   <!-- use topics from data_throttle at lower frame rate -->
   <node pkg="nodelet" type="nodelet"
name="points_xyz_planner" args="load
rtabmap_ros/point_cloud_xyz camera_nodelet_manager">
      <remap from="depth/image"
to="/camera/depth_registered/image_raw"/>
      <remap from="depth/camera_info"
to="/camera/rgb/camera_info"/>
      <remap from="cloud"                    to="cloudXYZ" />
      <param name="decimation" type="int" value="1"/>
<!-- already decimated in data_throttle above -->
      <param name="max_depth" type="double" value="3.0"/>
      <param name="voxel_size" type="double" value="0.02"/>
    </node>

    <node pkg="nodelet" type="nodelet"
name="obstacles_detection" args="load
rtabmap_ros/obstacles_detection camera_nodelet_manager">
      <remap from="cloud" to="cloudXYZ"/>
      <remap from="obstacles" to="/obstacles_cloud"/> <!--
move_base topic -->
      <remap from="ground"    to="/ground_cloud"/>    <!--
move_base topic -->
      <param name="frame_id" type="string" value="chasssis"/>
      <param name="map_frame_id" type="string" value="map"/>
      <param name="wait_for_transform" type="bool"
value="true"/>
      <param name="min_cluster_size" type="int" value="20"/>
      <param name="max_obstacles_height" type="double"
value="0.4"/>
    </node>
</launch>
```

A.16 Calculating correction factors and covariance using UMBmark

```
clear all
% Offsets at final position
x_cw=[0.27 0.18 0.01 0.48 0.08];
y_cw=[0.04 0.00 0.01 0.16 0.18];
x_ccw=[0.12 0.01 0.07 0.19 0.28];
y_ccw=[0.07 0.01 0.12 0.07 0.04];
theta=[0.15 0.13 0.09 0.01  0.03 0.12 0.16 0.21 0.16];
% Calculating maximum error
x_cw_g=mean(x_cw);
y_cw_g=mean(y_cw);
x_ccw_g=mean(x_ccw);
y_ccw_g=mean(y_ccw);
r_cw=sqrt((x_cw_g)^2 + (y_cw_g)^2);
r_ccw=sqrt((x_ccw_g)^2 + (y_ccw_g)^2);
emax=max(r_cw,r_ccw)
% Calculating correction values
L=3.5;
D_R=195;
D_L=195;
D_A=(D_R+D_L)/2;
alpha = ((x_cw_g + x_ccw_g)/(-4*L)) * 180/pi
beta = ((x_cw_g - x_ccw_g)/(-4*L)) * 180/pi
R=(L/2)/sin(beta/2)*180/3.14
E_d = (R + beta/2)/(R - beta/2)
E_b = 90/(90-alpha)
c_l=2/(E_d +1)
c_r=2/((1/E_d) + 1)
% calculating covariance values
x=[x_cw x_ccw];
y=[y_cw y_ccw];
cox=cov(x)
coy=cov(y)
cot=cov(theta)
```

A.17 Move base configuration files

Move_base_launch

```
<?xml version="1.0"?>
<launch>
  <!-- Move base -->
  <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">
<remap from="cmd_vel" to="cmd_vel"/>
    <!--remap from="odometry/combined" to="odometry"/-->
  <remap from="wheel_odom" to="odom"/>
    <param name="scan" value="scan"/>
     <remap from="map" to="/rtabmap/grid_map"/>
<rosparam file="$(find
wheelchair)/params/simulation/costmap_common_params.yaml"
command="load" ns="global_costmap" />
    <rosparam file="$(find
wheelchair)/params/simulation/costmap_common_params.yaml"
command="load" ns="local_costmap" />
    <rosparam file="$(find
wheelchair)/params/simulation/local_costmap_params.yaml"
command="load" />
    <rosparam file="$(find
wheelchair)/params/simulation/global_costmap_params.yaml"
command="load" />
    <!--rosparam file="$(find
wheelchair)/params/simulation/teb_local_planner_params.yaml"
command="load" /-->
    <rosparam file="$(find
wheelchair)/params/simulation/global_planner_params.yaml"
command="load" />
    <rosparam file="$(find
wheelchair)/params/simulation/base_local_planner_params.yaml"
command="load" />
    <rosparam file="$(find
wheelchair)/params/simulation/navfn_global_planner_params.yaml"
command="load" />
    <rosparam file="$(find
wheelchair)/params/simulation/move_base_params.yaml" command="load"
/>
  </node>
</launch>
```

Base_local_planner_params

```
controller_frequency: 1.0
recovery_behavior_enabled: false
clearing_rotation_allowed: false

TrajectoryPlannerROS:
  max_vel_x: 1.28
  min_vel_x: -1.28
  max_vel_y: 0.0
  min_vel_y: 0.0
```

```
  max_vel_theta: 1.57
  min_in_place_vel_theta: 1.0
  acc_lim_theta: 1.5
  acc_lim_x: 3.6
  acc_lim_y: 0.0
  holonomic_robot: false
  yaw_goal_tolerance: 3 # about 6 degrees
  xy_goal_tolerance: 0.5  # 10 cm
  latch_xy_goal_tolerance: false
  pdist_scale: 0.8
  gdist_scale: 0.6
  meter_scoring: true
  heading_lookahead: 0.325
  heading_scoring: false
  heading_scoring_timestep: 0.8
  occdist_scale: 0.1
  oscillation_reset_dist: 0.05
  publish_cost_grid_pc: false
  prune_plan: true
  sim_time: 1.0
  sim_granularity: 0.025
  angular_sim_granularity: 0.025
  vx_samples: 8
  vy_samples: 0 # zero for a differential drive robot
  vtheta_samples: 20
  dwa: true
  simple_attractor: false


Move_base params
shutdown_costmaps: false
controller_frequency: 5.0
controller_patience: 3.0
planner_frequency: 1.0
planner_patience: 5.0
oscillation_timeout: 10.0
oscillation_distance: 0.2
base_local_planner: "dwa_local_planner/DWAPlannerROS"
base_global_planner: "navfn/NavfnROS"




Costmap_common_params
obstacle_range: 3
raytrace_range: 3.0
#footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
robot_radius: ir_of_robot
robot_radius: 0.5 # distance a circular robot should be clear of the
obstacle
inflation_radius: 0.05

observation_sources: laser_scan_sensor point_cloud_sensorA
```

```
laser_scan_sensor: {data_type: LaserScan, topic: /scan, marking:
true, clearing: true}

point_cloud_sensorA: {
sensor_frame: chassis,
data_type: PointCloud2,
topic: /rtabmap/cloud_obstacles,
expected_update_rate: 1.0,
marking: true,
clearing: true,
min_obstacle_height: 0.05
}
```

Global costmap params

```
global_costmap:
  global_frame: map
  robot_base_frame: chassis
  update_frequency: 1.0
  publish_frequency: 1.0
  resolution: 0.02
  static_map: true
  width: 10.0
  height: 10.0
  rolling_window: false
  transform_tolerance: 1.0
  map_type: costmap

plugins:
- {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
- {name: inflation_layer, type: "costmap_2d::InflationLayer"}
```

Local costmap params

```
local_costmap:
  global_frame: odom
  robot_base_frame: chassis
  update_frequency: 5.0
  publish_frequency: 5.0
  static_map: false
  rolling_window: true
  width: 5.0
  height: 5.0
  resolution: 0.75
  transform_tolerance: 5.0
  map_type: costmap
  sonar_layer:
   topics: ["/sonar_rear"]
   no_readings_timeout: 1.0
  plugins:
  - {name: sonar_layer, type:
"range_sensor_layer::RangeSensorLayer"}
  - {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
  - {name: inflation_layer, type: "costmap_2d::InflationLayer"}
```

A.18 Complete Arduino Code

```
#include <ros.h>
#include <ros/time.h>
#include <tf/tf.h>
#include <tf/transform_broadcaster.h>
#include <geometry_msgs/Twist.h>
#include<SPI.h>
#include "MPU9250.h"
#include <sensor_msgs/Range.h>

MPU9250 IMU(Wire,0x68);
int status;

#define CS_ADC 10
#define CS_POT 8
#define L_PHASE_A 18
#define L_PHASE_B 19
#define R_PHASE_A 2
#define R_PHASE_B 3
#define wheel_base 440
#define wheel_diameter 195
#define left_counts_per_rotation 1242.7
#define right_counts_per_rotation 1242.6
#define channel0 0xE0
#define channel1 0xE8
#define POT_FB  0x12 //00010001
#define POT_LR  0x11//00010010
#define POT_ALL  0x13//00010011
#define SonarPin 9

ros::NodeHandle  nh;
geometry_msgs::TransformStamped t;
tf::TransformBroadcaster broadcaster;
sensor_msgs::Range rear_range_msg;

ros::Publisher pub_range_rear("/sonar_rear", &rear_range_msg);

//Encoder Variables
volatile int L_count=0;
volatile int R_count=0;

//Odom Variables
int right_pulses ;
int left_pulses ;
int prev_right_pulses;
int prev_left_pulses;
float right_wheel_distance;
float left_wheel_distance;
float wheel_theta;
float left_distance_per_tick =
((wheel_diameter*PI)/left_counts_per_rotation);
float right_distance_per_tick =
((wheel_diameter*PI)/right_counts_per_rotation);
```

```
float mean_distance;
float x_pos;
float y_pos;
int right_ticks;
int left_ticks;
float imu_theta_rate;
float imu_theta;
float theta_global;
float current_time;
float previous_time;
float diff_time;
float sonar_range;


//IMU variables
float aX, aY, aZ, gX, gY, gZ,mX,mY,mZ;

//odometry correction
double x_1;
double x_2;
double y_1;
double y_2;


void loop()
{
char chassis[] = "/chassis";
char odom[] = "/odom";
char sonar_rear[] = "/sonar_rear";

compute_odom();

if(checkjoystick())
joystick_control();

  // tf odom->base_link
  t.header.frame_id = odom;
  t.child_frame_id = chassis;
  t.transform.translation.x = x_pos;
  t.transform.translation.y = y_pos;
  t.transform.rotation = tf::createQuaternionFromYaw(theta_global);
  t.header.stamp = nh.now();
  geometry_msgs::Quaternion odom_quat =
tf::createQuaternionFromYaw(theta_global);
  broadcaster.sendTransform(t);

sonar_range=read_sonar();
rear_range_msg.header.frame_id = sonar_rear;
rear_range_msg.header.stamp = nh.now();
rear_range_msg.range=sonar_range;
pub_range_rear.publish(&rear_range_msg);
  nh.spinOnce();
  delay(10);
}
```

```
//Velocity values from ROS
void Callback(const geometry_msgs::Twist& vel)
{
  double linear_vel = vel.linear.x;
  double angular_vel = vel.angular.z;
  int rotational_speed;
  int linear_speed_sig;
  double linear_speed_dec;
  int linear_speed;
  int linear_speed_val;
  int fb_pot_val;
  int lr_pot_val;

//Safety measure to override values from ROS if joystick is used
  while (!checkjoystick())
  {

    angular_vel = angular_vel * 57.2958; //convert to degrees
    rotational_speed = int(254 - angular_vel * 1.6); //convert to
pot_val

    linear_speed_sig = int(linear_vel / 1); //Get integer part
    linear_speed_dec = linear_vel - linear_speed; //get decimal part
    linear_speed = int(linear_speed_sig * 100 + linear_speed_dec *
100); //combine integer and decimal
    linear_speed_val = (254 + linear_speed * 1.6); //convert to
pot_val

    if (linear_speed_val > 398)
      linear_speed_val = 398;
    if (linear_speed_val < 110)
      linear_speed_val = 110;
    if (rotational_speed > 398)
      rotational_speed = 398;
    if (rotational_speed < 110)
      rotational_speed = 110;

    fb_pot_val = (linear_speed_val - 100) * 0.85;
    lr_pot_val = (rotational_speed - 100) * 0.85;
    POTcontrol(POT_FB, fb_pot_val);
    POTcontrol(POT_LR, lr_pot_val);
  }
}

ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", Callback);

//Generate odometry from encoders and IMU
void compute_odom()
{
right_pulses=R_count;
left_pulses=L_count;
current_time = millis();
imu_theta_rate = get_imu_theta();
right_ticks = right_pulses - prev_right_pulses;
left_ticks = left_pulses - prev_left_pulses;
right_wheel_distance = right_ticks * right_distance_per_tick;
```

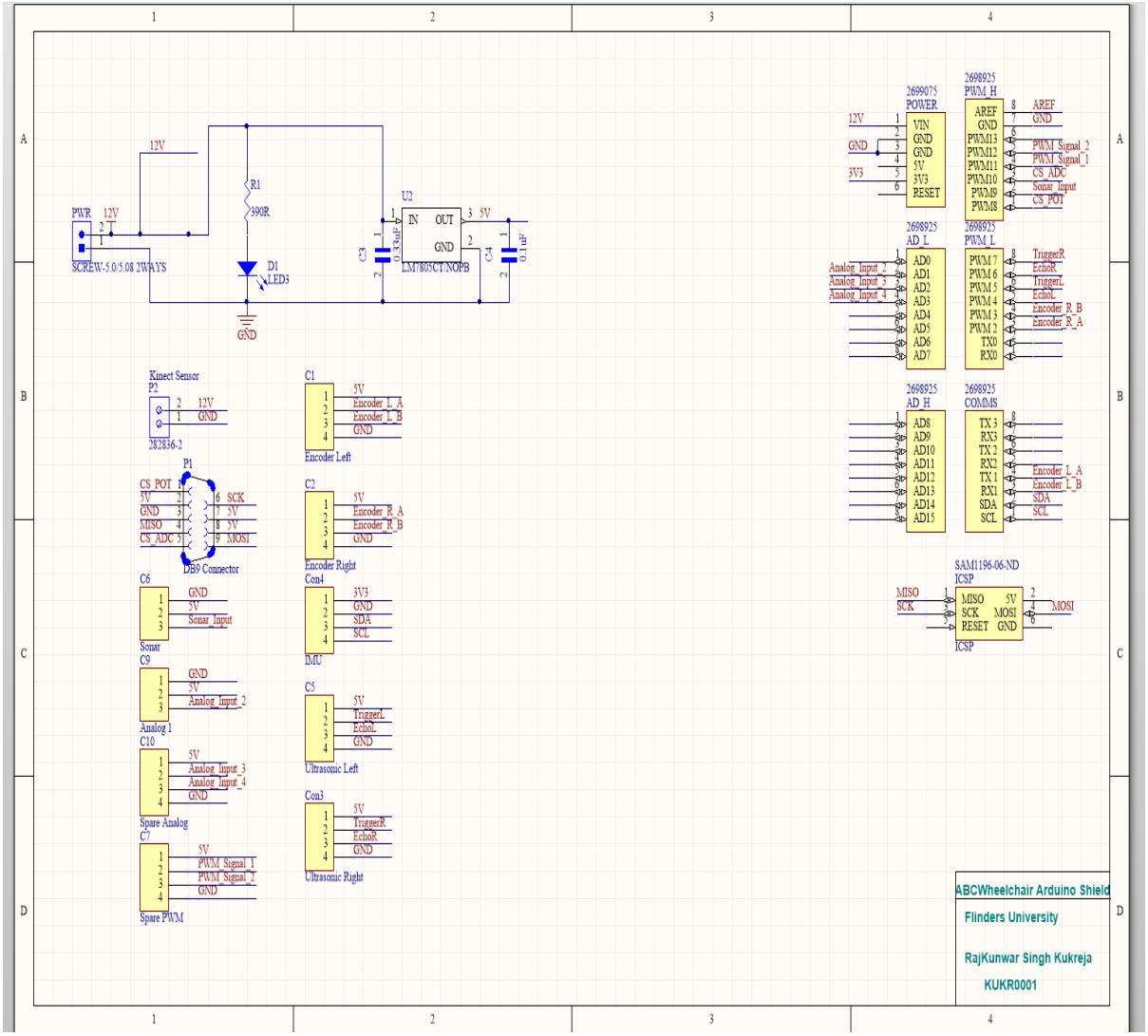
```
left_wheel_distance = left_ticks * left_distance_per_tick;
mean_distance = (left_wheel_distance + right_wheel_distance)/2;
diff_time = current_time - previous_time;
imu_theta -= imu_theta_rate*diff_time;
//wheel_theta += (right_wheel_distance -
left_wheel_distance)/wheel_base;

theta_global = imu_theta/1000;
//theta_global = wheel_theta;

if(theta_global< -PI)
theta_global+= 2*PI;
else if(theta_global > PI)
theta_global -= 2*PI;

x_pos += (mean_distance * cos(theta_global))*1.0/1000;
y_pos += (mean_distance * sin(theta_global))*1.0/1000;

prev_left_pulses=left_pulses;
prev_right_pulses=right_pulses;
previous_time=current_time;
}

//Rotation velocity rate from gyroscope
float get_imu_theta()
{
  IMU.readSensor();
  gZ =  IMU.getGyroZ_rads();
return gZ;

}

//Interrupts for encoder pulses
  void left_interrupt()
{
  char i;
  i=digitalRead(L_PHASE_B);
  if(i)
  L_count -=1;
  else
  L_count += 1;
}
void right_interrupt()
{
  char i;
  i=digitalRead(R_PHASE_B);
  if(i)
  R_count +=1;
  else
  R_count -= 1;
}

int jstickval0;
int jstickval1;
```

```
//Reading voltage from joystick
int ADCread(byte channel)
{
  int command;
  int voltage;
  byte fbit;
  byte sbit;
  command=channel<<8|0x00;

digitalWrite(CS_ADC,LOW);
voltage=SPI.transfer16(command) & 0x3FF;
digitalWrite(CS_ADC,HIGH);
return voltage;
}

//Sending values to potentiometer
void POTcontrol(byte address,byte value)
{
digitalWrite(CS_POT,LOW);
SPI.transfer(address);
SPI.transfer(value);
digitalWrite(CS_POT,HIGH);
}

void joystick_control()
{
  int turnjval;
  int drivejval;
  byte turnpot;
  byte drivepot;
  static float mult= 0.85;

  turnjval=ADCread(channel0);
  turnpot=(turnjval-100)*mult;
  drivejval=ADCread(channel1);
  drivepot=(drivejval-100)*mult;
  POTcontrol(POT_FB,drivepot);
  POTcontrol(POT_LR,turnpot);
  return 1;
}
//Checking if joystick is moved or being used for safety
byte checkjoystick()
{
  jstickval0=ADCread(channel0);
  jstickval1=ADCread(channel1);
  if(jstickval0>260 || jstickval0<248 && jstickval1>260 ||
jstickval1<248)
 return 1;
 else
 return 0;
}
void setup()
{
 status = IMU.begin();
   IMU.setAccelRange(MPU9250::ACCEL_RANGE_8G);
```

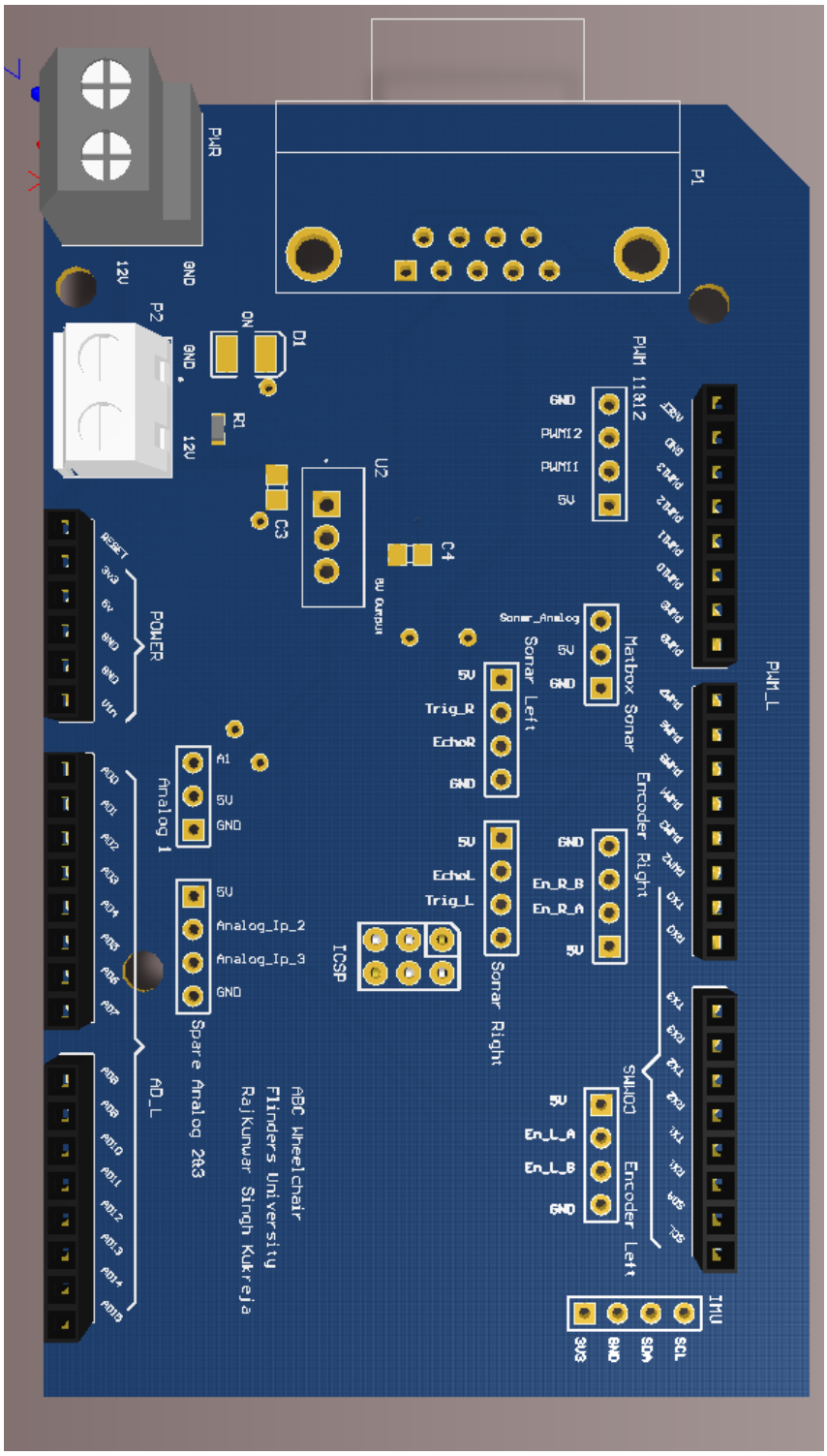
```
  // setting the gyroscope full scale range to +/-500 deg/s
  IMU.setGyroRange(MPU9250::GYRO_RANGE_500DPS);
  // setting DLPF bandwidth to 20 Hz
  IMU.setDlpfBandwidth(MPU9250::DLPF_BANDWIDTH_20HZ);
  // setting SRD to 19 for a 50 Hz update rate
  IMU.setSrd(19);

  nh.initNode();
  broadcaster.init(nh);
  nh.subscribe(sub);
  nh.advertise(pub_range_rear);
  rear_range_msg.radiation_type = sensor_msgs::Range::ULTRASOUND;
  rear_range_msg.field_of_view = 0.577;
  rear_range_msg.min_range = 0.3;
  rear_range_msg.max_range = 5;

pinMode(CS_POT, OUTPUT);
pinMode(CS_ADC, OUTPUT);

digitalWrite(CS_ADC,LOW);
digitalWrite(CS_ADC,HIGH);
digitalWrite(CS_POT,HIGH);

SPI.setBitOrder(MSBFIRST);
SPI.setDataMode(SPI_MODE3);
SPI.setClockDivider(SPI_CLOCK_DIV128);
SPI.begin();
Serial.begin(57600);

pinMode(SonarPin,INPUT);
POTcontrol(POT_ALL,0x80);

pinMode(L_PHASE_A,INPUT);
pinMode(L_PHASE_B,INPUT);
pinMode(R_PHASE_A,INPUT);
pinMode(R_PHASE_B,INPUT);
attachInterrupt(digitalPinToInterrupt(L_PHASE_A),left_interrupt,RISI
NG);
attachInterrupt(digitalPinToInterrupt(R_PHASE_A),right_interrupt,RIS
ING);
delay(2000);
}

float read_sonar()
{
float mm;
mm = pulseIn(SonarPin, HIGH);
return mm/1000; //Return distance in m
}
```

APPENDIX B

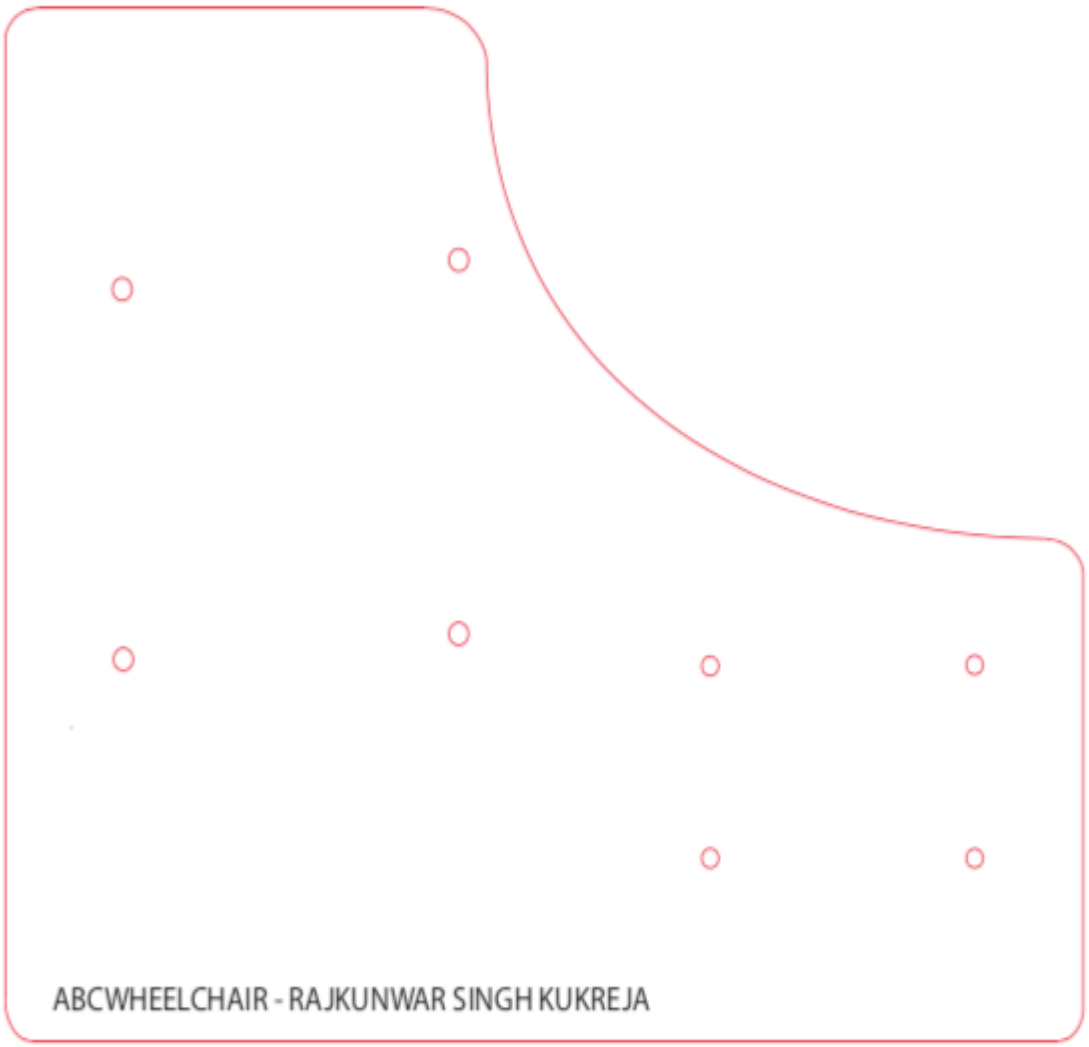
B.1 Arduino Sensor board shield Schematic



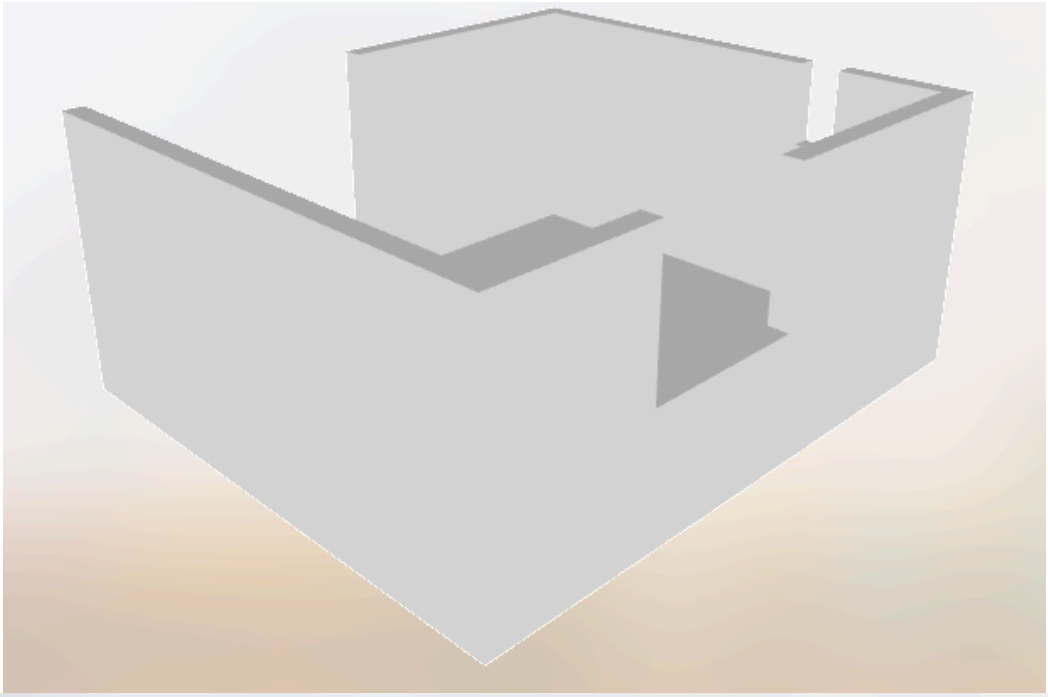
B.2 Arduino Sensor board shield PCB

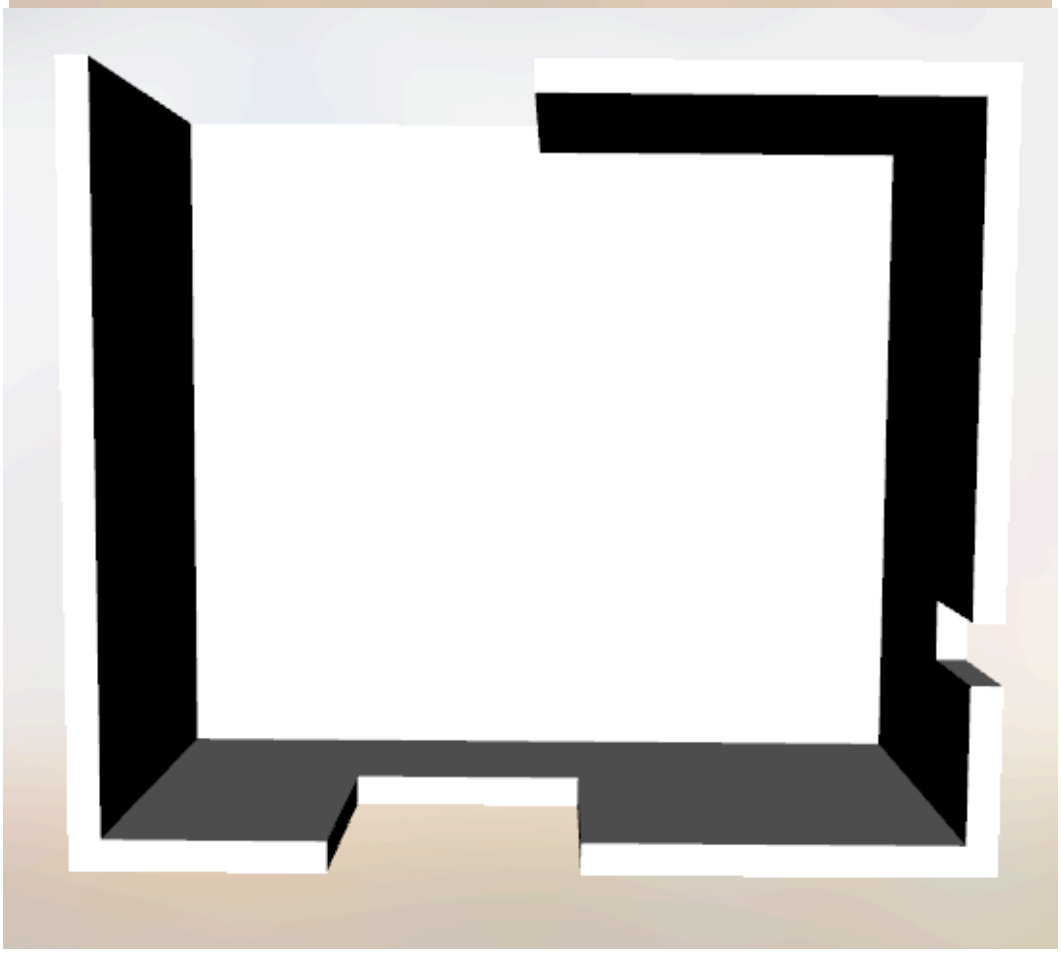


B.3 Wheelchair Tray Design

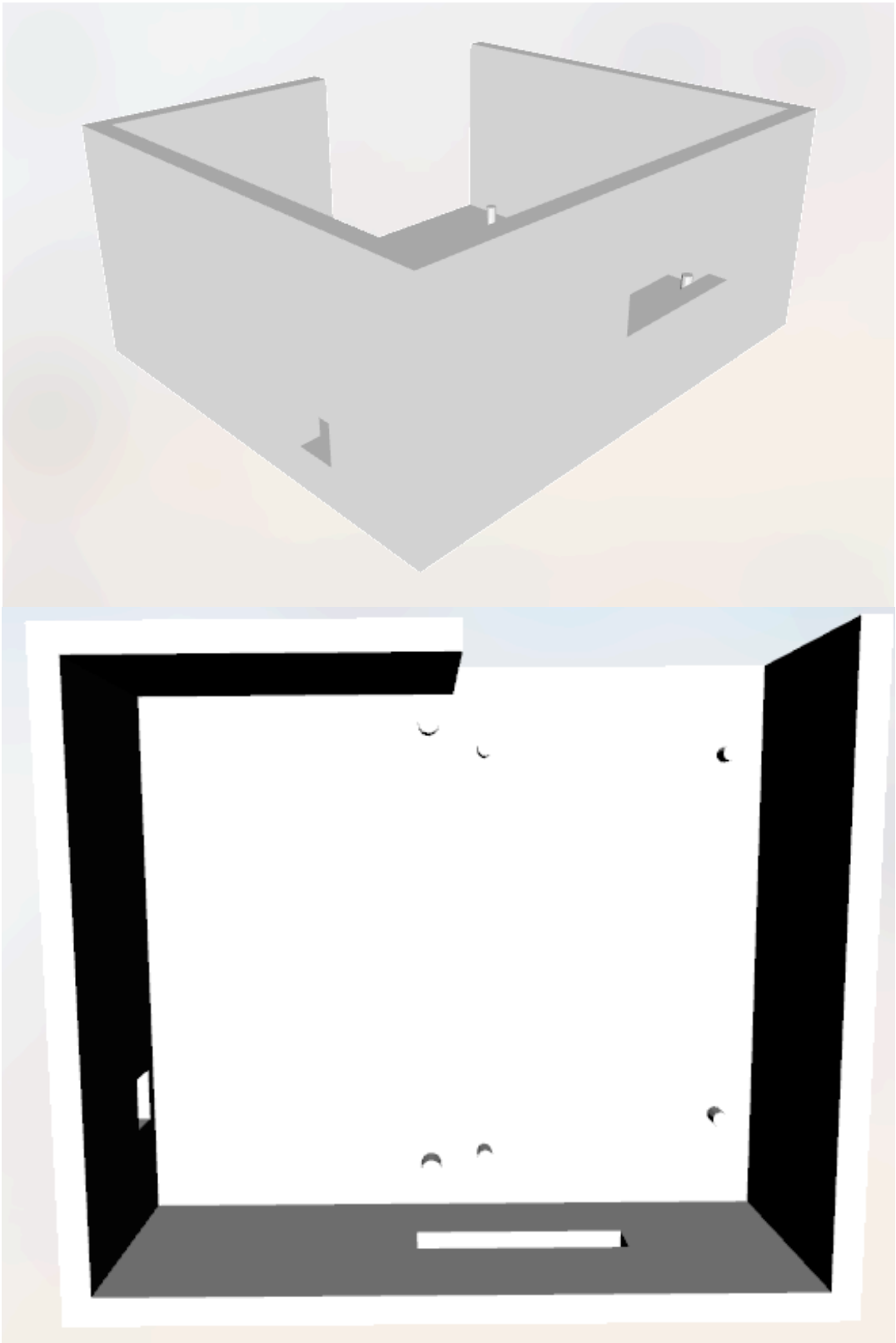


B.4 Controller Box Lid





B.5 Controller Box Base



REFERENCES

[1] Ben-Ari, M. and Mondada, F. (2018). Elements of Robotics. *Springer, Cham*, pp.63-93.

[2] Ros.org. (n.d.). ROS.org | About ROS. *Available a*t: http://www.ros.org/about-ros/

[3] Object Recognition In Any Background Using OpenCV Python - The Codacus. *Available at*: https://thecodacus.com/object-recognition-using-opencv-python/#.W8NXzRMzZ0w

[4] K. Miyazaki, M. Hashimoto, M. Shimada, and K. Takahashi. Guide following control using laser range sensor for a smart wheelchair*. In ICROS-SICE Int. Joint Conf., pages 4613–4616, Fukuoka Int. Congr. Center, Japan*, Aug. 2009.

[5] Borenstein, J. and Feng, L. (1996). Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6), pp.869-880.

[6] Parikh, S., Grassi, V., Kumar, V. and Okamoto, J. Integrating Human Inputs with Autonomous Behaviors on an Intelligent Wheelchair Platform. *IEEE Intelligent Systems, 22(2), pp.33-41*. (2007)

[7] BitSophia. (n.d.). BitVoicer. Available at: http://www.bitsophia.com/en-US/BitVoicer/Overview.aspx

[8] Betke, M. and Gurvits, L. (1997). Mobile robot localization using landmarks. *IEEE Transactions on Robotics and Automation*, 13(2), pp.251-263.

[9] R. Murakami, Y. Morales, S. Satake, T. Kanda, and H. Ishiguro. Destination unknown: Walking side-by-side without knowing the goal. pages 471–478, 2014

[10] (2017). "Self Driving Autonomous Wheelchairs are coming." from https://www.disabledworld.com/assistivedevices/mobility/wheelchairs/electric/autonomous-wheelchairs.php.

[11] Sinha, U., Saxena, P. and M, K. (2017). Mind Controlled Wheelchair*. IOSR Journal of Electrical and Electronics Engineering*, 12(03), pp.09-13.

[12] Cheng, C., Li, S. and Kadry, S. (2018). Mind-Wave Controlled Robot: An Arduino Robot Simulating the Wheelchair for Paralyzed Patients*. International Journal of Robotics and Control,* 1(1), p.6.

[13] Stachniss, C. (2018). Grid Maps.

[14] Do, Q. and Jain, L. (2009). Application of neural processing paradigm in visual landmark recognition and autonomous robot navigation. *Neural Computing and Applications*, 19(2), pp.237-254.

[15] McGarey, P. (2018). Visual Odometry.

[16] Ho, K. L. and P. Newman (2006). "Loop closure detection in SLAM by combining visual and spatial appearance." *Robotics and Autonomous Systems* 54(9): 740-749.

[17] Labbe, M. and Michaud, F. (2013). Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *IEEE Transactions on Robotics*, 29(3), pp.734-745.

[18] 14core.com. (2018). *Wiring the MPU9250 9 AXIS Motion Tracking Micro Electro Mechanical System | 14core.com | Ideas Converts Reality*. [online] Available at: https://www.14core.com/wiring-the-mpu9250-9-axis-motion-tracking-micro-electro-mechanical-system/ [Accessed 14 Oct. 2018].

[19] Grisetti, G., Tipaldi, G., Stachniss, C., Burgard, W. and Nardi, D. (2007). Fast and accurate SLAM with Rao–Blackwellized particle filters. *Robotics and Autonomous Systems,* 55(1), pp.30-38.
[20] Afanasyev, I., Ibragimov,I. (2017). Comparison of ROS-based Visual SLAM methods in homogeneous indoor environment. *Positioning, Navigation and Communications(WPNC)*
[21] GitHub. (2018). tzutalin/awesome-visual-slam. [online] Available at: https://github.com/tzutalin/awesome-visual-slam [Accessed 14 Oct. 2018].
[22] World Health Organization. (2018). Disability. [online*] Available at*: http://www.who.int/disabilities/en/ [Accessed 14 Oct. 2018].
[23] Shore, S. and Juillerat, S. (2012). The impact of a low cost wheelchair on the quality of life of the disabled in the developing world. *Medical Science Monitor*, 18(9), pp.CR533-CR542.
[24] Edwards, K. and Mccluskey, A. (2010). A survey of adult power wheelchair and scooter users. *Disability and Rehabilitation: Assistive Technology*, 5(6), pp.411-419.
[25] Labbe, M. (2018). rtabmap_ros/Tutorials/SetupOnYourRobot - ROS Wiki. [online] Wiki.ros.org. *Available at*: http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot [Accessed 14 Oct. 2018].
[26] Viola, G. (2018). geoffviola/tf_to_odometry. [online] GitHub. *Available at:* https://github.com/geoffviola/tf_to_odometry [Accessed 14 Oct. 2018].
[27] Wiki.ros.org. (n.d.). move_base - ROS Wiki. [online] *Available at*: http://wiki.ros.org/move_base [Accessed 13 Oct. 2018].
[28] Wiki.ros.org. (n.d.). depthimage_to_laserscan - ROS Wiki. [online] *Available at:* http://wiki.ros.org/depthimage_to_laserscan [Accessed 13 Oct. 2018].
[29] Taylor, T. (2011). Kinect for Robotics. [online] Microsoft Robotics Blog. *Available at:* https://blogs.msdn.microsoft.com/msroboticsstudio/2011/11/29/kinect-for-robotics/ [Accessed 14 Oct. 2018].
[30] Sensor (2018). [online] *Available at*: https://www.generationrobots.com/en/401430-microsoft-kinect-sensor.html [Accessed 14 Oct. 2018].
[31] El-laithy, R., Huang, J., Yeh, M. Study on the use of Microsoft Kinect for robotics applications. IEEE/ION *Position, Location and Navigation Symposium* (2012)
[32] Grisetti, G. (2018). Introduction to Navigation using ROS.
[33] Wiki.ros.org. (n.d.). costmap_2d - ROS Wiki. [online] *Available at*: http://wiki.ros.org/costmap_2d [Accessed 13 Oct. 2018].
[34] Amazon.co.uk. (2018). Sterling Ruby Wheelchair. [online] *Available at*: https://www.amazon.co.uk/Sunrise-Medical-Sterling-Ruby-Plus/dp/B000NB726Q [Accessed 14 Oct. 2018].
[35] Zheng, K. (2016). ROS Navigation Tuning Guide
[36] Wiki.ros.org. (n.d.). base_local_planner - ROS Wiki. [online] *Available at*: http://wiki.ros.org/base_local_planner [Accessed 13 Oct. 2018].
[37] Lee, J. (1997). Indoor robot navigation by landmark tracking. *Mathematical and Computer Modelling*, 26(4), pp.79-89.
[38] Qu, X. Landmark based Localization: detection ,matching and update of landmark with uncertainty analysis. (2016)
[39] Www2.informatik.uni-freiburg.de. (2018). [online*] Available at*: http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti10titsmag.pdf [Accessed 14 Oct. 2018].

[40] MPU-9250, S. (2018). *SparkFun IMU Breakout - MPU-9250 - SEN-13762 - SparkFun Electronics*. [online] Sparkfun.com. *Available at*: https://www.sparkfun.com/products/13762 [Accessed 14 Oct. 2018].
[41] En.wikibooks.org. (2018). *Robotics/Print version - Wikibooks, open books for an open world*. [online] Available at: https://en.wikibooks.org/wiki/Robotics/Print_version [Accessed 14 Oct. 2018].
[42] Hackaday.io. (2018). *ROBOT ODOMETRY | Details | Hackaday.io*. [online] *Available at*: https://hackaday.io/project/158496-imcoders/log/147068-robot-odometry [Accessed 14 Oct. 2018].
[43] Windows Central. (2018). *Microsoft to end sales of original Kinect for Windows sensor in 2015*. [online] *Available at:* https://www.windowscentral.com/microsoft-end-sales-original-kinect-windows-sensor-2015 [Accessed 14 Oct. 2018].
[44] W., D. (2018). *Robot Operating System (ROS) - Artificial Intelligence and Machine Learning for Robotics (Houston, TX)| Meetup*. [online] Meetup.com. *Available at:* https://www.meetup.com/ai-ml-for-robotics-and-iot/messages/boards/thread/50451824 [Accessed 14 Oct. 2018].
[45] Wiki.ros.org. (n.d.). navigation - ROS Wiki. [online] *Available at:* http://wiki.ros.org/navigation/Tutorial/RobotSetup [Accessed 13 Oct. 2018].
[46] Hlavac, V. Simultaneous localization and mapping (2002)
[47] Dellaert, F. and Kaess, M. (2006). Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. *The International Journal of Robotics Research*, 25(12), pp.1181-1203.
[48] Au.mathworks.com. (2018). *Costmap representing planning space around vehicle - MATLAB- MathWorks Australia*. [online] Available at: https://au.mathworks.com/help/driving/ref/vehiclecostmap.html [Accessed 14 Oct. 2018].
[49] Muhammad, N., Fofi, D., Ainouz, S. Current State of the art of vision based SLAM. (2012)
[50] Mycoordinates.org. (2018). Coordinates : A resource on positioning, navigation and beyond » Blog Archive » Outdoor mobile field robot navigation. [online] Available at: https://mycoordinates.org/outdoor-mobile-field-robot-navigation/ [Accessed 14 Oct. 2018].
[52] Ionescu, A., (2018). *Kinematic scheme of the two wheeled robot*. [online] Available at: https://www.researchgate.net/figure/Kinematical-scheme-of-the-two-wheeled-mobile-robot_fig1_272051522 [Accessed 14 Oct. 2018].
[53] Katevas, N., Sgouros, N., Tzafestas, S., Papakonstantinou, G., Beattie, P., Bishop, J., Tsanakas, P. and Koutsouris, D. (1997). The autonomous mobile robot SENARIO: a sensor aided intelligent navigation system for powered wheelchairs. *IEEE Robotics & Automation Magazine*, 4(4), pp.60-70.
[54] Simpson, R. (2005). Smart wheelchairs: A literature review. *The Journal of Rehabilitation Research and Development*, 42(4), p.423.
[55] Asayesh, S. (n.d.). *Electronic Design of Brain Controlled Wheelchair*. Masters. Flinders University.
[56] Foeng, V. (n.d). *Facial.* Masters. Flinders University
[57] Scudellari, M. (2017). *In Japan and Singapore, self-driving wheelchairs debut in hospitals and airports*. [online] ITU News. Available at: https://news.itu.int/self-driving-wheelchairs-debut-in-hospitals-and-airports/ [Accessed 15 Oct. 2018].
[58] Opencv.org. (2018). *OpenCV library*. [online] Available at: https://opencv.org/ [Accessed 15 Oct. 2018].

[59] Lammas, A. Kalman Filter Variants (2018)
[60] Tobii.com. (2018). *Tobii and Microsoft Collaborate to bring Eye Tracking Support in Windows 10*. [online] Available at: https://www.tobii.com/group/news-media/press-releases/2017/8/tobii-and-microsoft-collaborate-to-bring-eye-tracking-support-in-windows-10/ [Accessed 15 Oct. 2018].
[61] Labbé, M. and Michaud, F. (2017). Long-term online multi-session graph-based SPLAM with memory management. *Autonomous Robots*, 42(6), pp.1133-1150.