# CER217 - Shared Control of an Autonomous Powered Wheelchair

Supervisor - Héctor Caltenco and Håkan Eftring


LUND INSTITUTE OF TECHNOLOGY
Lund University

# Table of Contents

# Acknowledgements

We would like to thank the Certec department for providing us with valuable feedback and support

We would like to thank Héctor Caltenco for helping us configure the circuits required for the control of the easy rider.

We would like to thank Alfredo Chávez Plascencia for providing us with his previous code of the wheelchair.

We would like to thank Håkan Eftring and Afshin Alyali for helping us design and machine an attachment mechanism for our encoders.

We would like to thank Charlotte Magnusson for giving us valuable feedback during our presentation sessions.

# 1. Introduction

Electrically powered wheelchairs (PWC) are used to assist individuals with severe motor disabilities. Users that maintain some degree of motor control are able to use joysticks. Users experiencing severe motor disabilities have to rely on alternative control methods such as tongue, head, chin and even forehead movement [1][2][3]. These systems may be tiresome for many individuals, while others might restrict the normal use of the user's vision or head, eyes or tongue movement [4]. Autonomy provided to the wheelchair might relieve these burdens allowing a more free use of body parts used for navigation. However, a fully autonomous system may not be desired by some users as they may deprive of a sense of independence.

# 2. Literature Review

## 2.1 Need for Powered wheelchairs

There is a great need for the availability of an autonomous wheelchair. Studies have been conducted to demonstrate the need for such a wheelchair. One study suggests that 9 to 10 percent of patients who receive power wheelchair training find it extremely difficult or impossible to use the wheelchair for activities of daily living. When asked specifically about steering and maneuvering tasks, the percentage of patients reported to find these difficult or impossible jumped to 40. Nearly half of patients unable to control a power wheelchair by conventional methods would benefit from an automated navigation system, according to the clinicians that treat them. 85% of responding clinicians reported seeing some number of patients each year for whom the use of a power wheelchair is not an option because these patients lack the motor skills, strength, or visual acuity needed to control the chair [4].

## 2.2 Overview of the affected population

There are a number of conditions that cause disabilities in individuals and result in the use of wheelchairs for assistance. Alzheimer's Disease causes a progressive deterioration of functions such as language, motor skills, and perception. Amyotrophic Lateral Sclerosis (ALS) which results in premature degeneration of motor neurons. Cerebral Palsy (CP) results from brain injury occurring before cerebral development is complete and it consists of spastic, athetoid and ataxic types. Cerebrovascular Accident (CVA) (aka stroke) is a disruption in the brain's blood supply due to arterial occlusion or rupture, causing irreversible neurological impairment. Multiple Sclerosis damages the myelin sheath surrounding nerve fibers in the central nervous system (CNS) [2].

There are also some impairments that affect wheelchair mobility. In upper body physical impairment, there are several impairments such as Ataxia which is a lack of muscular coordination. Bradykinesia refers to a reduced speed or amplitude of movement. Muscle fatigue or weakness can lead to collisions if a wheelchair user cannot quickly respond to a moving or suddenly appearing obstacle. Spasticity refers to a condition in which muscles are continuously contracted, which results in stiffness that interferes with movement. On the other hand, cognitive impairments also affect wheelchair mobility such as, Neurological impairments which can cause difficulties with concentration, attention, and mood control. This results in impairment in judgment, reasoning, planning, problem solving, decision making, and sequencing actions [5].

## 2.3 Alternative wheelchair control methods

The disabilities of some individuals have encouraged the need for research in order to determine different methods for alternative control such as brain waves, muscular activity or eye posture. But these methods require focus and is often very difficult to operate wheelchairs with it. This is due the sensitivity of the EEG signal. Whenever the subject blinks, swallows, laughs, talks, or moves in any way, the corresponding EEG sample is rendered unusable, since the EEG consists of very tiny voltage potentials, that have to be amplified by a factor on the order of 10^4 but any noise contamination gets amplified, too, resulting in inaccurate commands. Furthermore, since a wheelchair driver constantly moves with the wheelchair he or she is sitting in, trying to compensate braking or turning, there would inevitably be a lot of contamination of signals [6].

Another alternative way used to control a wheelchair is the Tongue control method. The tongue control is inserted into the oral activity of the user and is practically invisible. This provides an aesthetic feel to the user. The high number of sensors and their special layout allows the system to be used in many different configurations and applications, such as typing on a computer, or for directional control by combining sensor signals to get joystick functionality. The embedded controller of the system coordinates which of the external devices the user is controlling and relays processed sensor information to the appropriate device [1].

People with severe disabilities and limited motor function have an option to control a wheelchair by the use of head movements. Devices such as head controlled joystick or head-movement interface, both mechanical, camera based, accelerometer based and based on infrared light [6].

People with Quadriplegia are unable to use hand-operated systems in controlling a wheelchair. Chin-operated joystick is a solution for people with such disabilities. Generally, a chin operated joystick is a type of position sensing joystick. Head movement is required in the form of neck flexion, extension and rotation. The ability to control the wheelchair for such users depends on their ability to use their head movement [2].

## 2.4 User issues with Powered and Autonomous wheelchairs

With both powered wheelchairs and autonomous wheelchairs developed in the past, users have different concerns with the technology. For example, power wheelchair users report being afraid to navigate in crowded spaces with their device. In addition, clinicians who prescribe wheelchairs report that some clients cannot use the wheelchair safely because of visual, motor and cognitive deficits. Users commented that they wanted to continue to do the tasks that they were currently physically able to carry out, and caregivers corroborated this finding [7].

Some participants used Autonomous Powered Wheelchairs in their daily lives for research with the user experience and faced problems. Participants did not want the it to replace their own physical abilities while using the wheelchair. For example, one PWC user stated: “So if I have this feature, of it driving itself, I’d become, hum, really, hum I’d rely on it too much more than I need to…So it could make me lazy and not enhance my mobility” (22 years old, male). Following a planned path was viewed by half the participants as being useful. Those that did not find it useful expressed that this was a task that everyone has to do, for example in a shopping mall, and that this was a task they could and wanted to continue to do by themselves [7].

# 3. Purpose

## 3.1 Our overall objective

The purpose of the project is to find the optimal distribution between user control and autonomy for different types of users. During our 8 week stay at Lund University we focused primarily on building the framework for a fully autonomous wheelchair that could be catered for heavily immobilized users. Our implementation can easily be extended to create semi autonomous wheelchairs for users that are less immobilized.

# 4. Method

## 4.1 Functions, Objectives and Constraints

Through iterative brainstorming, the following design design requirements were generated:

Functions

- ❏ Will operate in different environments such as:

- ❏ Indoors
- ❏ Outdoors
- ❏ Unknown area

- ❏ Will have different modes of autonomy that users can select from
- ❏ Will operate under a wide variety of speeds

Objectives

- ❏ Should be able to localize itself in a short span of time
- ❏ Should be able to estimate its pose in a short span of time

Constraints

- ❏ Must be easy and safe to operate
- ❏ Must avoid stationary obstacles while operating

Due to time constraints, some functions, objectives and constraints were omitted. In particular, we did not design the wheelchair for use in outdoor areas and unknown environments. Furthermore, we only designed for one mode of autonomy; fully autonomous mode.

## 4.2 Hardware Overview

The proposed system consists of the following hardware components:

1. Acer TravelMate P253-M (running Ubuntu 14.04)
2. Arduino Mega ADK
3. C400 Permobil PWC
4. Easy Rider Interface
5. Custom designed circuit (more details in section 4.1.1 below)
6. Xbox Kinect 360
7. Two 24 PPR (pulses per revolution) angular encoders (two backup LED encoders)
8. Biltema 12V 1.2Ah Lead Acid Battery (for Arduino Mega)
9. Biltema 12V 4.5Ah Lead Acid Battery (for Xbox Kinect)

## 4.3 System Overview

The Robot Operating System (ROS) running on a Ubuntu 14.04 was chosen as the operating system for the autonomous wheelchair. At the core of the system is the navigation stack. This stack creates a nodelet that take odometry information and sensor streams as input and outputs a velocity command to send to a mobile robot in order to get it to move to a desired location in a mapped environment. Odometry data will be collected by one encoder attached to each wheel, while sensor data will be provided by the Kinect. The system level diagram for SLAM with Gmapping is illustrated in figure 4.3.1 below. Bear in mind that this diagram does not contain all of the required nodes for the autonomous robot to operate. Instead, it describes the nodes that need to be configured at a high level.

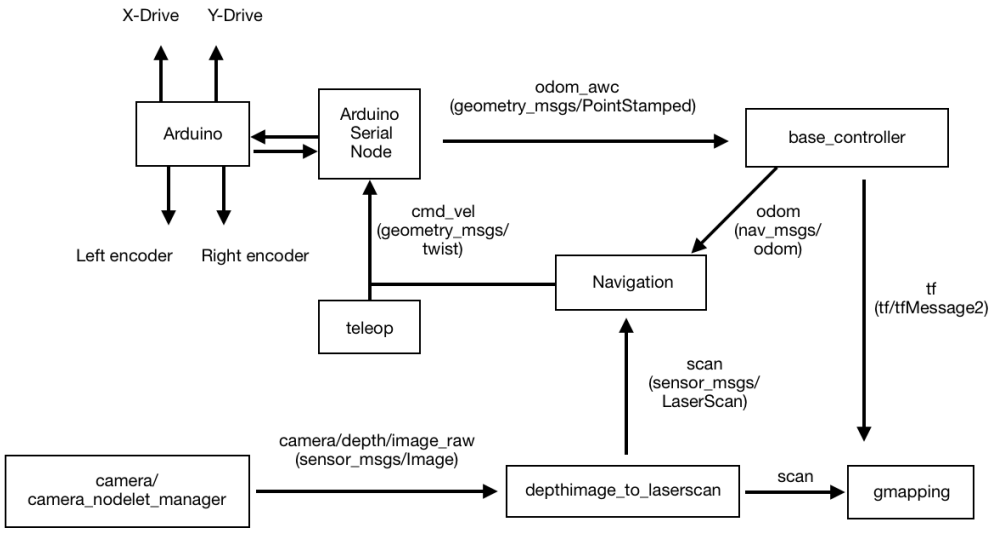



*Figure 4.3.1. System Overview Gmapping*

Each box represents a node in ROS. These boxes represent a computation process in the entire system. The connection between nodes/nodelets is given by a ROS topic, which is a pathway for communication between each node. The message type sent through the topic is indicated below each topic in brackets. The arrows indicate which node subscribes and publishes to different topics. An arrow pointed towards a box means that the node subscribes to the topic, while an arrow pointed away from the box means that the node publishes to a particular topic.

The map used by the navigation or move base node will be a 2D occupancy grid map. Depth images captured by the Kinect are first converted into laserscan data point. A combination of fake laser scans and odometry allows the Gmapping node to create a 2D map of the environment. The navigation node or move base node is responsible for the autonomous navigation of the wheelchair.

This system architecture was modified from Sunjik Cha's personal project titled "My Personal Robotic Companion" [8] as well as the Turtlebot meta package [9]. Much of the file structure and file names used in Cha's personal project were preserved, but most of the content had to be overwritten so that the code would work with our wheelchair.

In addition to Cha's system architecture, we also experimented with two other similar architectures: one with Hector Mapping [10] and another with Rtabmap [11]. Rtabmap without odometry was also used in our tests, however the results were inferior to those generated using wheel odometry. Overall, Gmapping [12] and Rtabmap were the two most promising SLAM techniques that we experimented with.

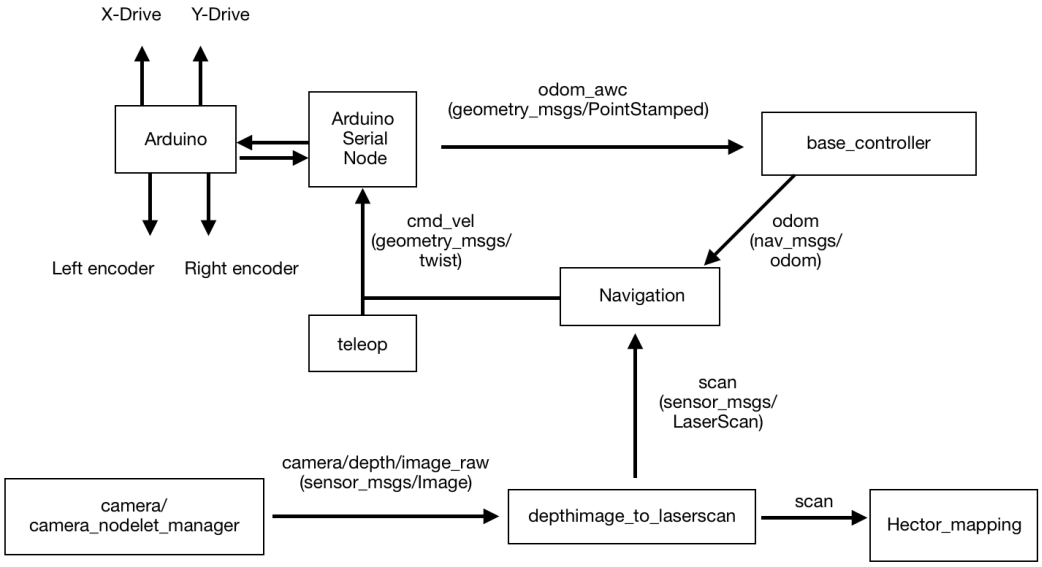



*Figure 4.3.2. Hector Mapping System Overview*

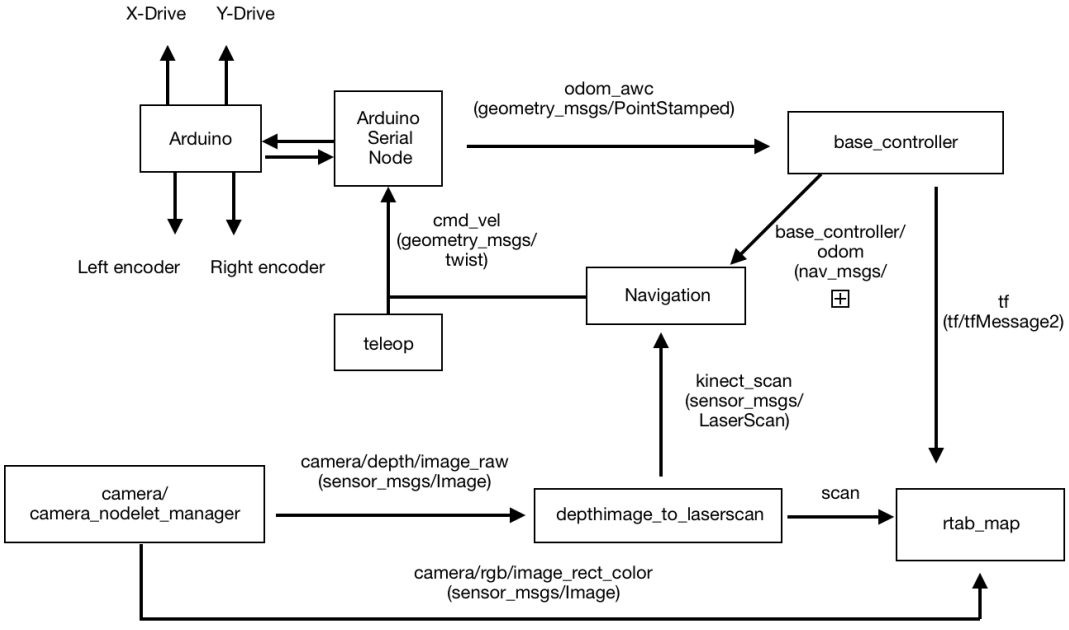



*Figure 4.3.3. Rtab mapping System Overview*

## 4.4 Hardware Configuration

### 4.4.1 Interfacing with Easy Rider

The Easy Rider Interface is an “integrated environment control system”[13]. It provides the wheelchair with additional functions such as connection to phones and alarm systems. Additionally, the Interface allows for easy use of alternative wheelchair feedback inputs such as chin, voice and alternative joystick control. In this project, we utilized the joystick control of the Easy Rider Interface to control the wheelchair.

A detailed pin assignment of the Easy Rider Interface is provided in Appendix A. The 8 pin DIN connector was chosen as a the main connection for interfacing between the wheelchair and the wheelchair circuitry.

The schematic for our custom made driver circuit for the Easy Rider Interface was created using Fritzing and is illustrated in figure 4.4.1 below. Initially, the Arduino Uno microcontroller was used, but due to lack of memory, the Arduino Mega was utilised. Note that only two of the operational amplifiers of the LM324N Quad Operational Amplifiers have been used. Using two non-inverting operational amplifier setups, the voltage at Pin 6 and Pin 7 of the 8 pin DIN connection could be controlled. The voltage at both of these pins hovered between 4V to 6V depending on the voltage coming out of Pin 10 and Pin 9 of the Arduino Mega.. 5V was the neutral position, while 4V and 6V dictated the direction the wheelchair would go along the X and Y axis of the wheelchair. Button 1 creates a short between Pin 4 and GND, which will turn the wheelchair ON/OFF. Button 2 creates a short between Pin 3 and GND, which acts as the function button in the Easy Rider interface.

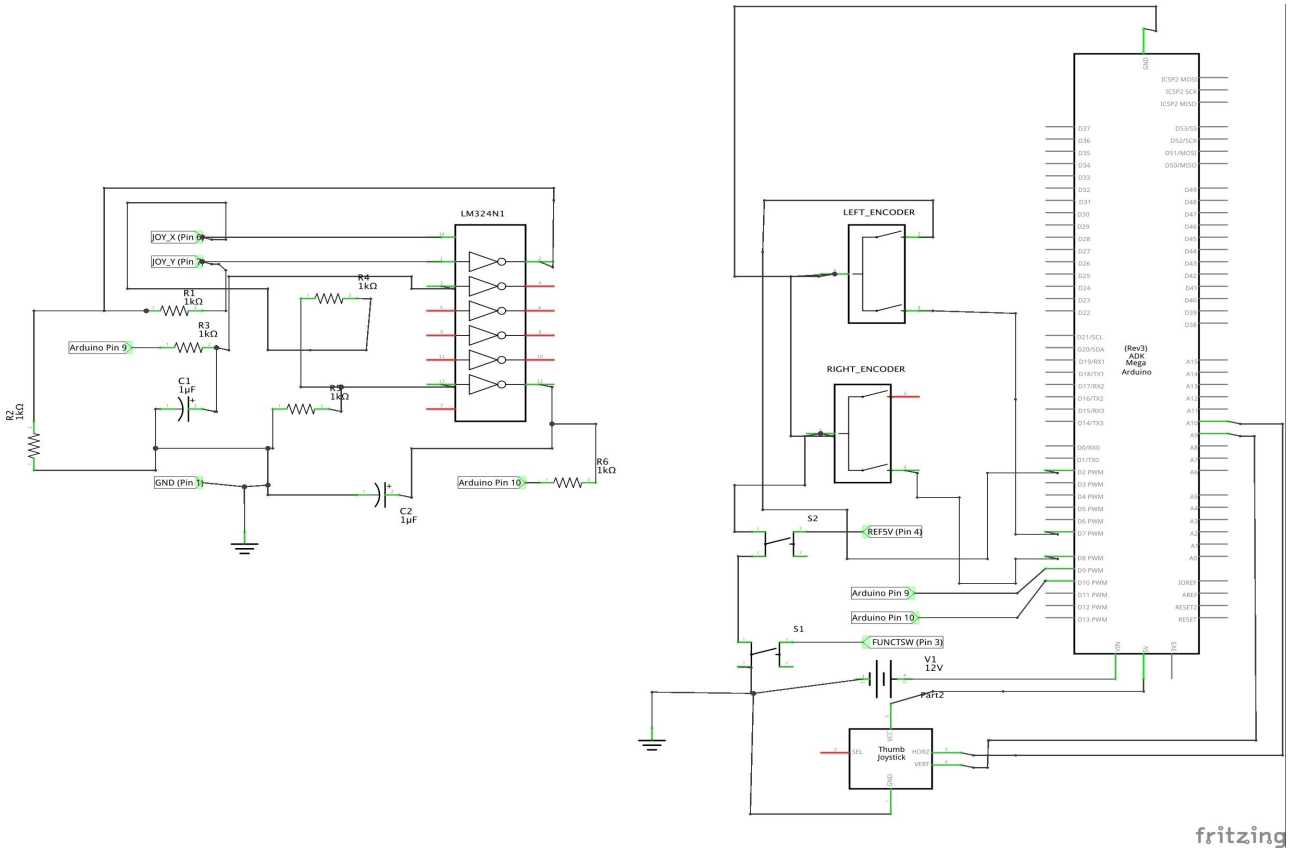



*Figure 4.4.1 Schematic of circuit design*

An external 12V battery was also plugged into the Arduino Mega to ensure sufficient power supply to the board. Removal of the 12V power supply often led to errors in which the Easy Rider read the joystick controls as out of neutral.

Our 8 pin DIN connector replaced the original 8 pin DIN connector for the Easy Rider joystick, so a joystick control was added to our circuit as a replacement.

### 4.4.2 Angular Encoders

Our initial setup consisted of two 24 PPR angular encoders that were hot glued onto the center of each wheel and held up by a steel frame. figure 4.4.2 below depicts our overall encoder setup.

*Figure 4.4.2. Encoder attachment on wheelchair*

After repeated usage, the glue slowly got worn out and the encoder completely detached from the hot glue as shown in figure 4.4.3 below.

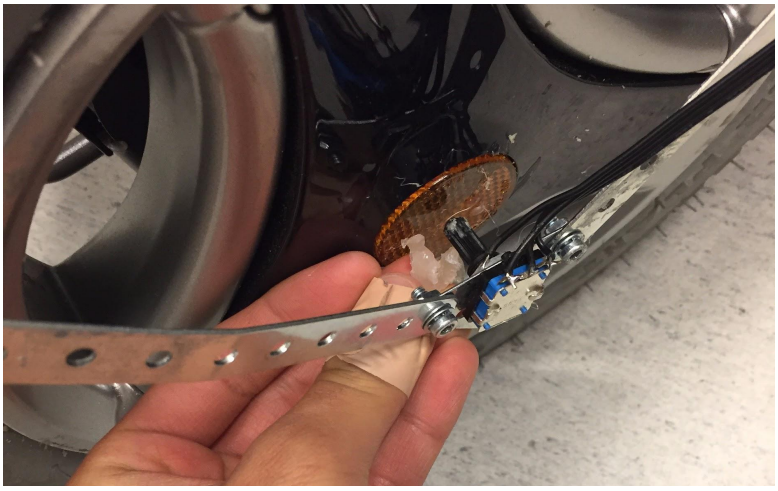


*Figure 4.4.3. Detachment of encoder*

Subsequently, a more robust solution for the encoder attachment was devised. The angular encoders were attached to a larger plastic casing using a set screw. The plastic casing was then glued onto the wheelchair's wheel using silicon glue and further reinforced with duct tape.

*Figure 4.4.4. Second iteration of angular encoder attachment*

A detailed implementation of the method of odometry update can be found in encoder_test.ino in Appendix B under the function UpdatePosition(). The function predicts the robot's new position based on the distance travelled by the left and right wheels, which is found by multiplying the number of encoder ticks by the circumference of the wheels and then dividing by the encoder pulses per revolution.

This type of odometry update is susceptible to errors when there is wheel slippage and/or uneven floors. Moreover, the caster wheels behind the wheelchair induced frictional forces on the wheelchair that were not accounted for in the odometry update.

### 4.4.2 Kinect Power Supply

The Kinect camera was powered by a portable 12V lithium ion battery as shown in figure 4.4.5 below. Since the Kinect was directly plugged into the battery supply, additional care had to be taken to ensure that the battery was supplying 12.0±0.5V to the Kinect camera. The original AC adapter for the Kinect power supply was cut off and the two power supply wires for the Kinect were attached to the terminals of the battery using alligator clips. The 1.2Ah 12V battery was initially used to power the Kinect. However, we later switched to using the 4.5Ah 12V battery for longer battery life.

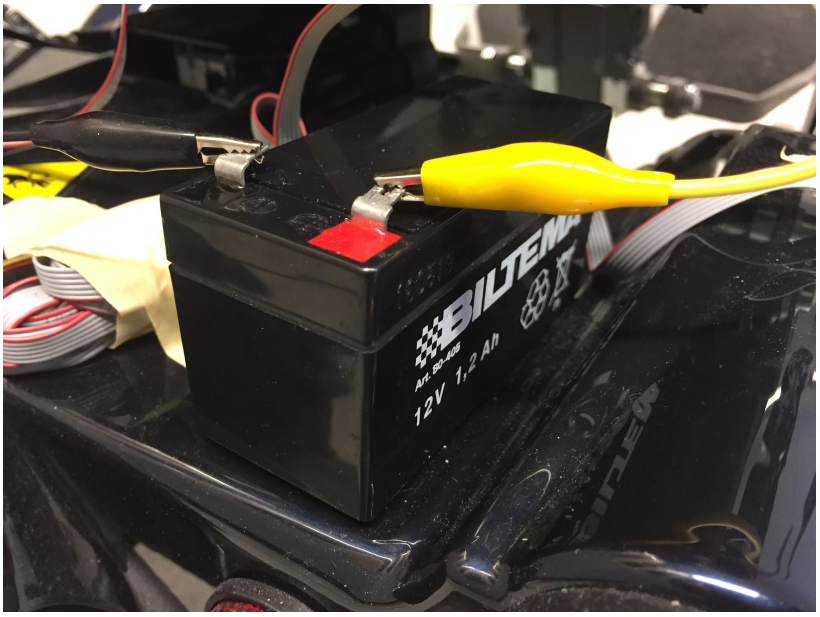



*Figure 4.4.5. Kinect Power Supply*

## 4.5 Software Configuration

### 4.5.1 Kinect Configuration

The freenect_launch package [15] was used as a driver to read Kinect data on the PC, while depthimage_to_laserscan package [16] was used to publish fake laserscan data computed using the Kinect's depth sensors.

### 4.5.2 Arduino Serial Node

The Arduino serial node takes input from the left and right wheel encoder and estimates the wheelchair's new position based on these inputs. The new position of the wheelchair is sent to the base controller node via the topic odom_awc. In addition, the serial node takes a velocity message from the topic cmd_vel and converts it into a corresponding analog output for the driver circuit. The serial communication between the Arduino and laptop was established with the help of the Rosserial package [17].

### 4.5.3 Teleop Node

The teleop node launches a window on the Linux computer that allows for the user to control the wheelchair using the keyboard. The user interface for keyboard control is illustrated in figure 4.5.1 below. This node was created for debugging velocity commands that were being fed into the Arduino serial node; the velocity commands from this node are of the same message type as those published by the move base node. Note that this node must not be operational while the move base node is in use.

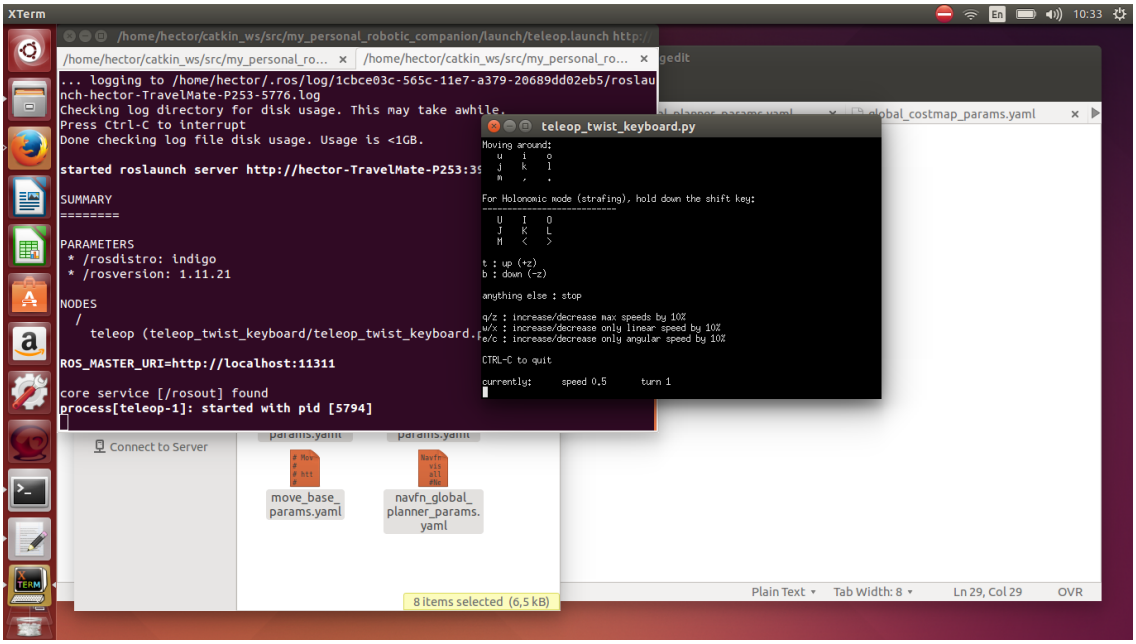



*Figure 4.5.1. Teleop interface*

### 4.5.3 Base Controller Node

Since the move base node does not take messages of the form geometry_msgs/PointStamped, the message published through the topic odom_awc had to be converted. The base controller node takes the message from the odom_awc topic and converts it into the form nav_msgs/Odometry where it will be published to the navigation nodelet via the odom topic. Furthermore, the base controller has been configured to publish the transform tree messages required for the navigation node and gmapping nodes. The transform tree (or TF) could not be published using the Arduino serial node due to dependency issues, which was the reason why the base controller node was created.

### 4.5.4 Configuration of the Navigation Stack (a.k.a move base node)

The figure below shows the required inputs of the navigation stack. Boxes labelled in blue are the custom configurations required for every robot, while boxes in gray have already been implemented in the ROS package. In a typical navigation configuration, the navigation nodelet takes sensor streams, odometry and transform tree inputs and provides a velocity command to the base controller of the robot. Our robot differed from this stack in that, the velocity commands are directly published to the Arduino serial node.

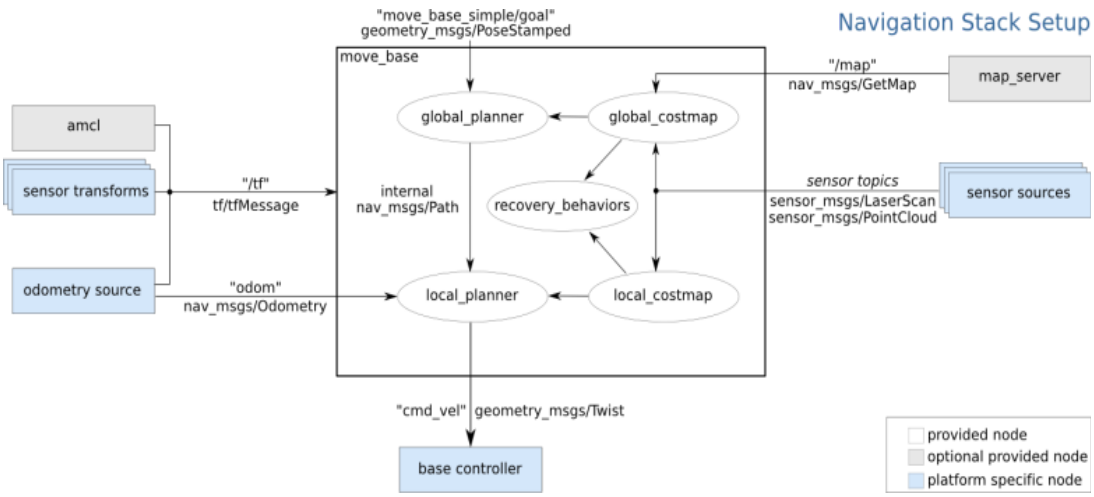



*Figure 4.5.2. Navigation Stack Configuration*

Four files containing various parameters were created for the configuration of this stack. All of these parameter files can be found in the directory ~/catkin_ws/src/my_personal_robotic_companion/params.

The costmap_common_params.yaml file defines the robot footprint, defines the inflation radius and directs the costmap to the sensor topic. A costmap is a modified 2D occupancy grid that the move base node uses for trajectory planning. The robot footprint defines how large of a box the robot takes up in meters under a two dimensional coordinate system. Inflation radius determines how much the robot footprint should be inflated in the costmap of the navigation stack.

The local_costmap_params.yaml and global_costmap_params.yaml file defines many parameters Most importantly, these files define the frame in which the local and global costmap will operate in as well as the update frequency of these costmaps.

The base_local_planner_params.yaml file defines the velocity and acceleration limits of the wheelchair.

## 4.6 Creating a Simulation

Software simulation was conducted in order to understand the ROS navigation setup and how the wheelchair behaves. Many different aspects of the wheelchair were considered in the simulations. The simulation helped us understand what the robot was sensing, how it was planning and then computing commands for the robot to execute. As the robot model was previously constructed in the URDF format, we had to change the code in the launch and world files in order to simulate the physical model. Gazebo and RViz were both used to simulate the model in different environments as Gazebo demonstrates a better physical model and Rviz is important for map making. The figure below demonstrates the wheelchair model in an empty

world. It shows the wheel encoders, kinect sensor and other physical features of the wheelchair model.

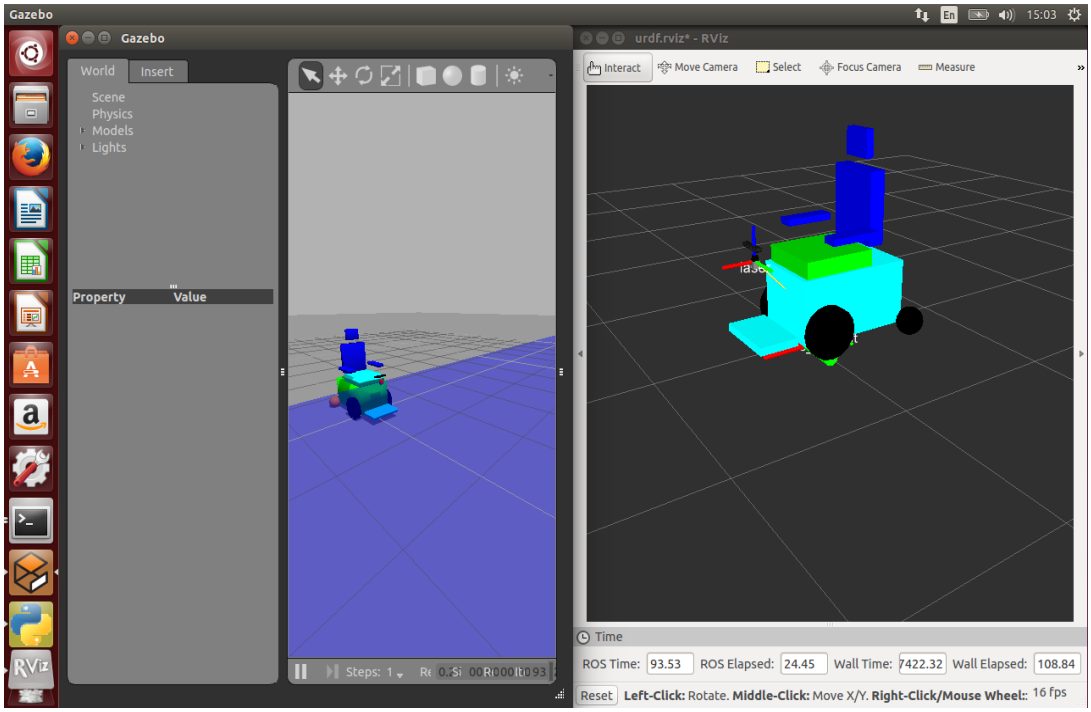



*Figure 4.6.1. Wheelchair robot model in Gazebo (left) and RViz (right)*

Different tests were run with the existing code but we were unable to move the wheelchair model at a sufficient speed which was required to map. The mybot model from the online tutorials was then modified to have the same physical dimensions as the body of the wheelchair [18]. The main body was modified along with the wheels by modifying the xacro file. The figure below shows the modified model in both Gazebo and RViz launched in the gmapping demo world which consists of five different obstacles. After launching it in Rviz, teleop was used to send velocity commands for the robot to move and create a map of the surrounding with the laser. After successfully creating the map, it was saved and relaunched with the wheelchair model in it. This time the '2D nav goal' option was chosen in Rviz to demonstrate autonomous movement while avoiding obstacles in a given map. The simulation successfully demonstrated a completely autonomous robot which localizes and moves to a desired location within the created map. It clearly demonstrated the ROS navigation package and how it works in real life.

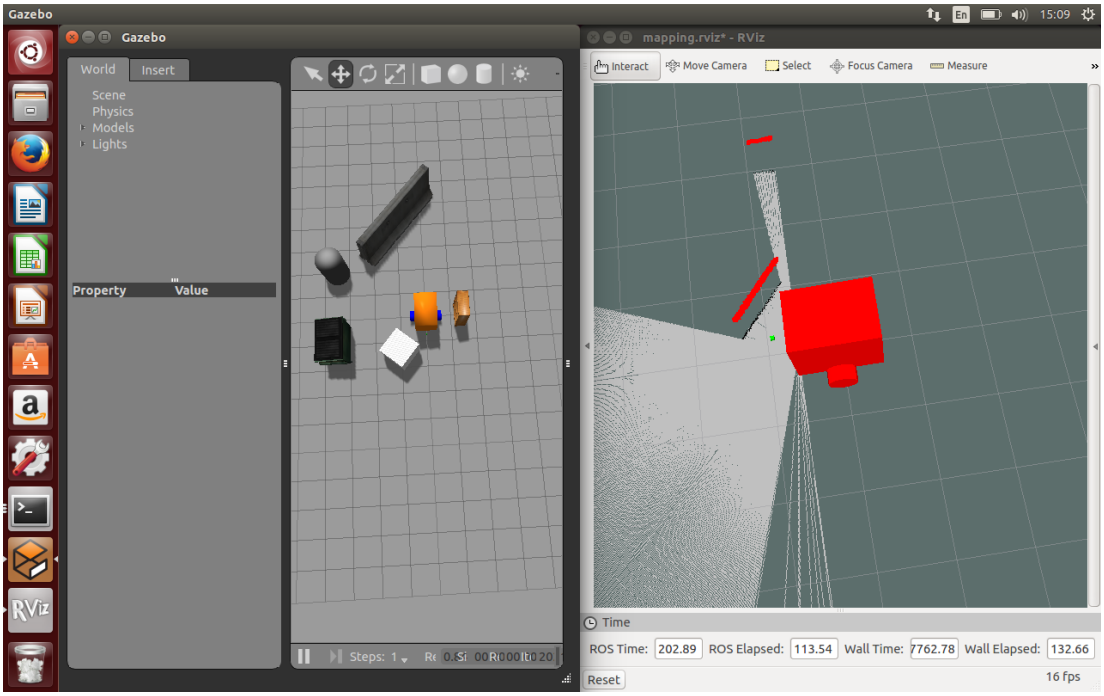



*Figure 4.6.2. Modified version of the wheelchair model in Gazebo (left) and RViz (right)*

## 4.7 Generating a Map

The three main mapping packages used in this project were Rtabmap, Hector Mapping and Gmapping. Both Rtabmap and Hector Mapping did not require odometry data and could be configured to only use sensor data from the Kinect. However, Rtabmap does have the option of incorporating Kinect sensor data with odometry data. Gmapping requires both odometry data and Kinect sensor data. Both Gmapping and Hector Mapping are laser-based SLAM approaches. As such, both mapping packages require that we transform the depth images from the kinect into laserscan data. Rtabmap is a SLAM based approach that relies on the RGB-D cameras. The required terminal commands for launching each type of mapping are outlined in Appendix C below.

## 4.8 Autonomous Navigation

The launch file instructions for autonomous navigation are shown in Appendix C. Once the user follows all of the instructions for “How to launch SLAM” outlined in Appendix C, an RViz window will pop up as illustrated in figure 4.9.1 below. The user will then have to provide a pose estimate to the move base node. This is done by selecting “2D Pose Estimate” button and selecting an approximate location and pose for the robot on the occupancy grid in the RViz window on the right. Red arrows will then appear around the general area where the user clicked; these are arrows represent particles used for the Adaptive Monte Carlo Localisation Method. Each of these particles represent a random position and orientation that the robot could

take on. As you move the robot around, many of the particles will start to die off and the robot gains a more accurate estimation of its location.

Once the robot has been localised, a navigation goal can be set. This is done by simply clicking on the “2D Nav Goal” button in RViz and specifying a location and final pose for the robot in the occupancy grid.

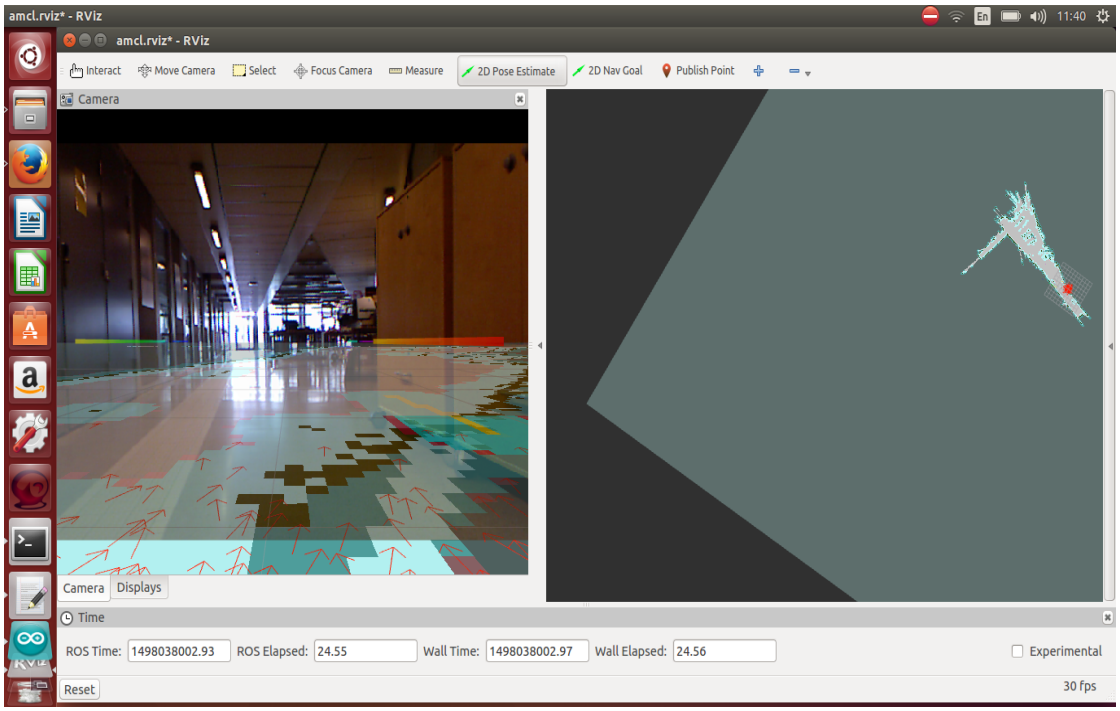



*Figure 4.9.1. Autonomous navigation test using Rviz command*

## 4.9 File Structure

The structure of our customised ROS package for the wheelchair is outlined below and it is located in the directory ~/catkin_ws/src/my_personal_robotic_companion. Unimportant files were omitted in the file structure.

my_personal_robotic_companion

- launch
    - robot_config.launch
    - gmapping.launch
    - rviz_gmapping.launch
    - amcl_demo.launch
    - rviz_amcl.launch
    - rtab_robot_config.launch
    - rtab_full.launch

        - rviz_rtab.launch
        - teleop.launch
        - includes
            - kinect_laser.launch
            - kinect_laser2.launch
            - amcl.launch
            - move_base.launch
            - urdf.launch
        - rviz
            - amcl.rviz
            - rtab.rviz
            - slam.rviz
- maps
    - map1.pgm
    - map1.yaml
- param
    - base_local_planner_params.yaml
    - costmap_common_params.yaml
    - global_costmap_params.yaml
    - local_costmap_params.yaml
- src
    - base_controller.cpp
- urdf
    - my_personal_robotic_companion.urdf
- CmakeLists.txt
- package.xml
- README.md

# 5. Results

## 5.1 Mapping Outcomes

Our initial attempts focused on using packages that created maps without odometry: Hector Mapping and Rtabmap. Hector Mapping created 2D occupancy grid maps, while Rtabmap only created 3D point cloud maps under default configurations (RGB-D SLAM with visual odometry). In order to utilise the navigation stack, a 2D occupancy grid must be fed into the move base node.

Hector Mapping performed very poorly in most of our initial attempts to map our environment. In our testing, we fixed the Kinect sensor to the wheelchair and drove it around different areas to see the map created by the package. Figure 5.1.1. below illustrates one of our many attempts at

mapping a corridor. Often times the Hector Mapping package would create two distinct regions when trying to map one area. This was most likely due to the limited field of view of the Kinect, which limited the scan matching capabilities of the Hector Mapping algorithm.

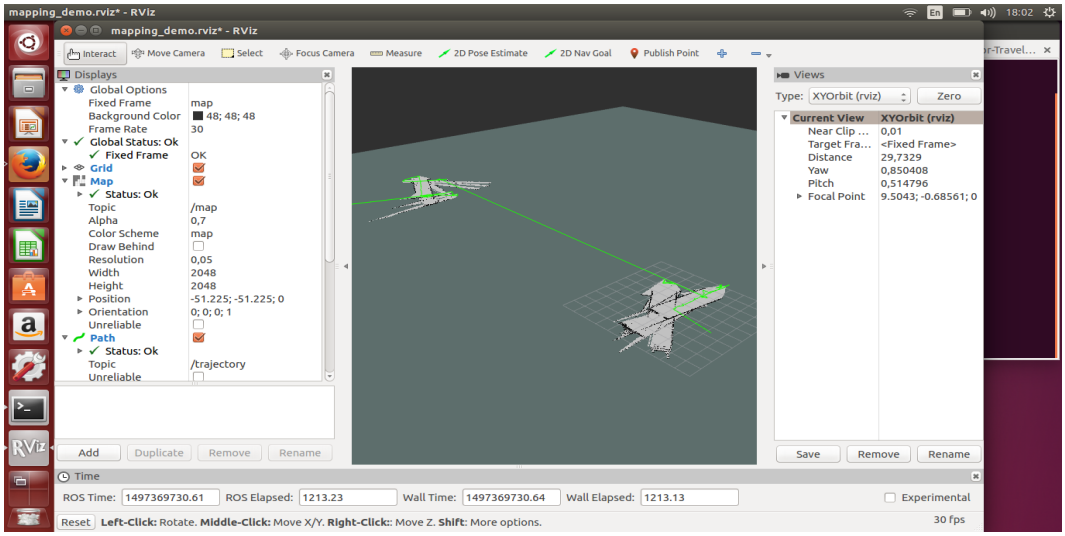



*Figure 5.1.1. Hector Mapping in narrow corridor*

Using a similar setup as above, we attempted to use Rtabmap to create a map of our environment. Figure 5.1.2 illustrates our attempt at generating a map of the IKDC lobby. Overall, the map generated was more accurate than that of Hector Mapping. However, there were certain areas in which the map ended up being slanted. In spite of its flaws, Rtabmap has shown to be extremely fast and effective at localising itself. During our testing, the robot was often able to determine its approximate location on the map almost instantly after resetting the visual odometry. Furthermore, the visual odometry node within the Rtabmap was decently effective at providing pose estimations of the robot.

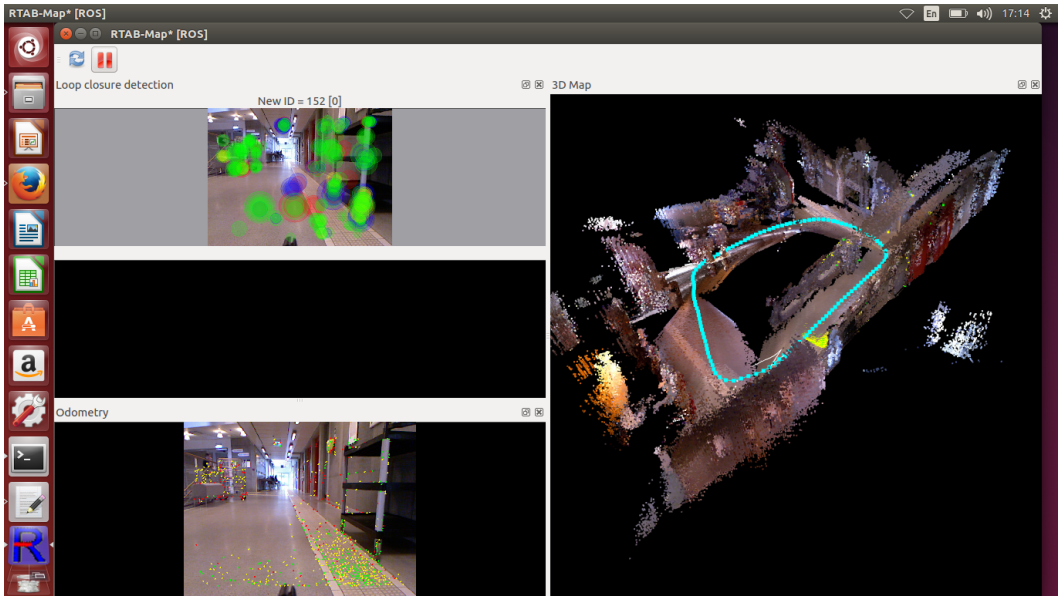



*Figure 5.1.2. Rtabmap mapping in IKDC lobby*

Having found little success in our attempts at creating odometry-free maps, we moved onto mapping packages that incorporated odometry data. The Gmapping package provided us with fairly accurate 2D grid maps that we were able to feed into the move base node. Our attempts at using Gmapping are shown in figure 5.1.3 and 5.1.4 below.

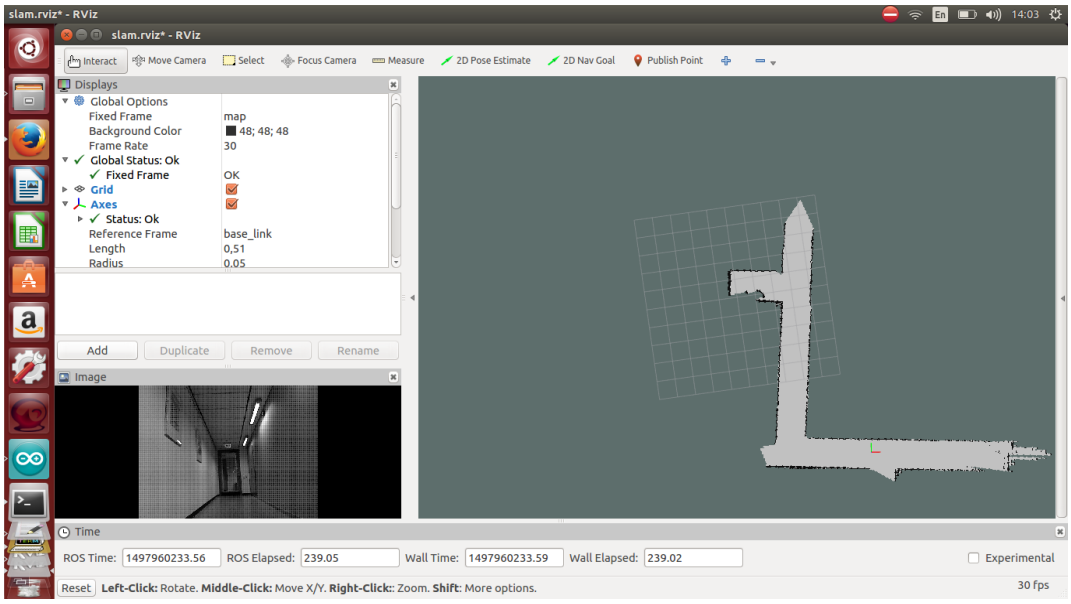



*Figure 5.1.3. Gmapping in a narrow corridor*

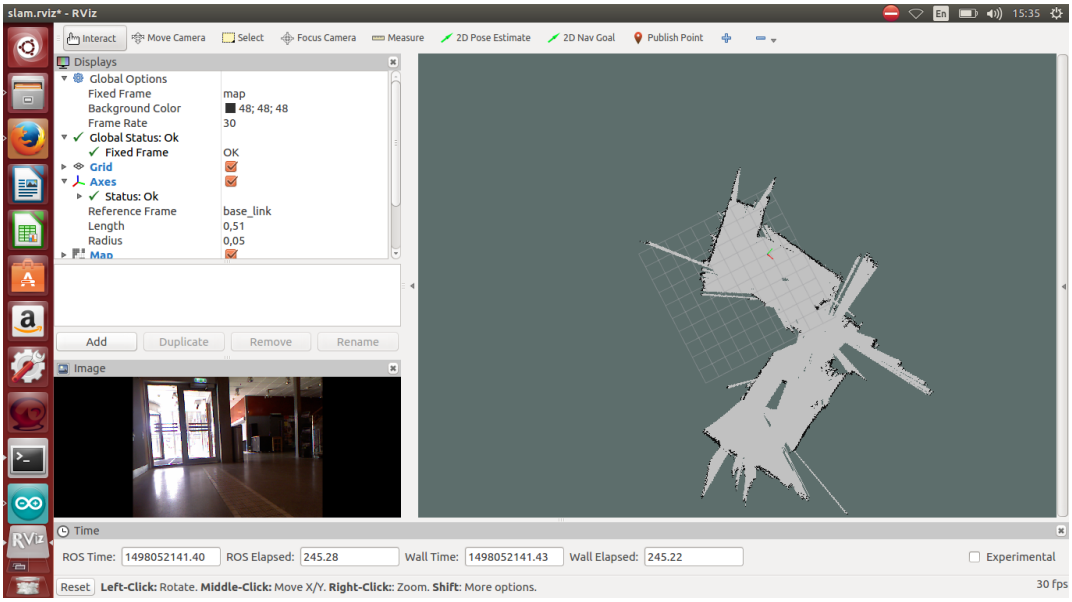



*Figure 5.1.4. Gmapping outside prototype lab*

Towards the end of the project, we were also able to use Rtab mapping with odometry information and fake laser scans. A basic overview of the system is given in figure 5.1.5 below. The rtabmap node accepts three streams of data: rgb images, fake laser scans and odometry data. Using this data, it then outputs a 3D map and a 2D occupancy grid.

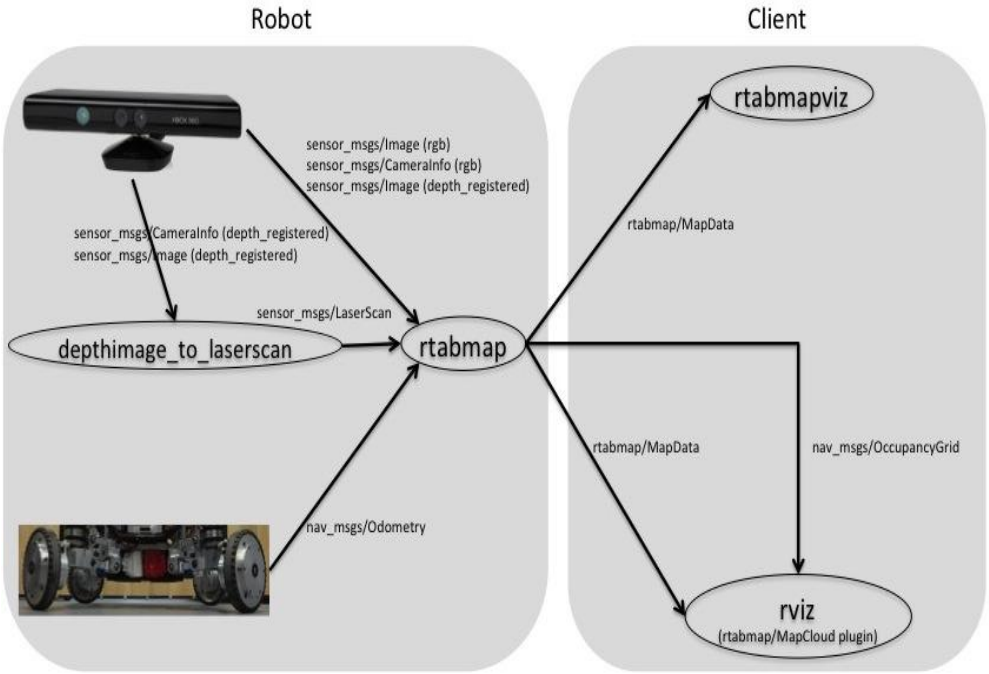



*Figure 5.1.5. Rtabmap with Kinect and Odometry [19]*

Rtabmap enables an additional level of complexity in localisation and mapping, which could provide more detailed and accurate maps in environments when compared with Hector Mapping and Gmapping. Figure 5.1.6 below illustrates an example of the 3D map and 2D map generated by Rtabmap. This image also showcases one issue with using laser scans for mapping; laser scans may not identify glass doors as obstacles or walls.

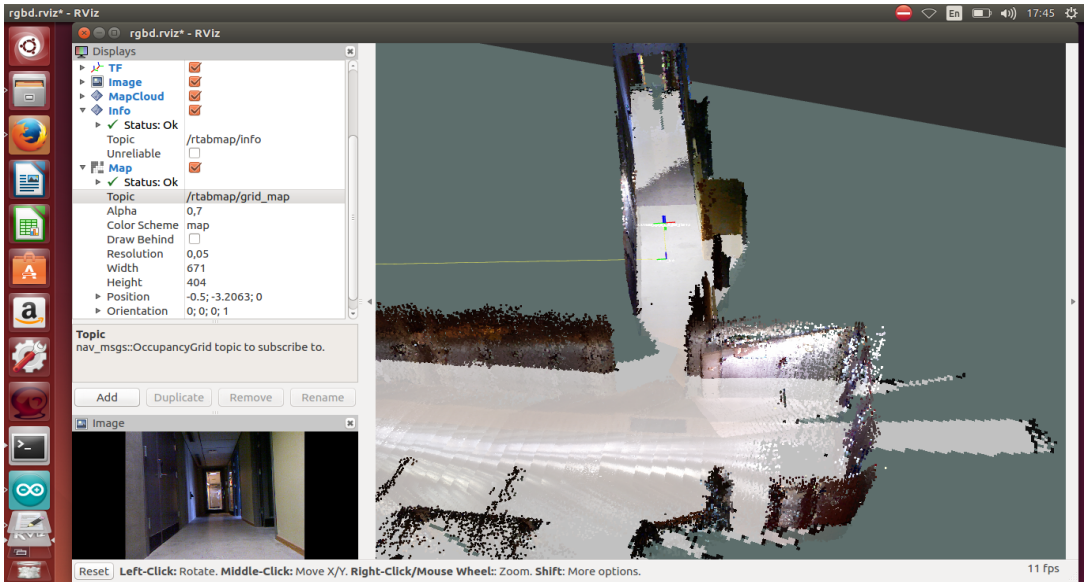



*Figure 5.1.6. Rtabmap with Kinect and Odometry*

## 5.2 Autonomous Navigation

Due to time constraints, the autonomous aspects of the wheelchair were not tested extensively. In particular, we were only able to have the wheelchair move autonomously towards a simple forward goal on RViz using a simple 2D occupancy grid. The parameters for the move base node were also not calibrated extensively. The wheelchair would also occasionally collide with walls. As such further calibration would should be done with the move base node. Additionally, future groups could also experiment with autonomous navigation using a 3D map and occupancy grid. The launch instructions for this type of navigation is outlined in the launch file instructions in Appendix C.

# 6. Discussion

## 6.1 Hardware system

We have attempted to use a Raspberry Pi 3 instead of the Acer TravelMate as the computer for our system. Unfortunately, the Pi 3 did not have sufficient processing power to run many of the visualizations that we used for this project.

## 6.2 Moving Ahead

### 6.2.1 UI for navigation

Our current system relies heavily on the use of terminal commands, but these commands may not be intuitive to users without a programming background. Future teams could work on desktop applications with custom-made graphical user interfaces that are more catered to users without programming experience. Touch screen interfaces could also be incorporated to control the wheelchair.

A smartphone could be used to navigate the wheelchair. Caretakers or wheelchair users could use their smartphones to gain control of the wheelchair for remote control.

### 6.2.2 Adding Modes of Autonomy

Three different modes of autonomy can be used to control the wheelchair: fully autonomous mode, semi autonomous mode and user control mode. The scope of our project focused primarily on autonomous navigation. Even so, our current system configuration can easily be modified to allow for a semi autonomous system. In fact, most of the changes that need to be made to enable a semi autonomous are in the file encoder.ino; this is the file for programming the Arduino.

In a semi autonomous navigation mode, the user would first input their final destination to allow the navigation stack to plan a global path to the user's final destination. Subsequently, the user will input their own velocity commands to the wheelchair through some form of analog signal input such as tongue, joystick or even chin control. The user's analog signal will create a corresponding velocity vector. If this vector deviates too far from the velocity vector published by the cmd_vel topic from the move base node then the Arduino will output the velocity command from the move base node. If the velocity vector generated is within a margin of error, then the Arduino will output the velocity command from the user. As such, only the hand_cmd function in the file needs to be modified.

## 6.3 Improving the System

In the following section we provide ways to improve our design, as we were not able to implement these changes due to time constraint with the project.

### 6.3.1 Using an IMU

Our current odometry updates are not entirely accurate. Sometimes 360° in place rotations on the wheelchair are registered as about 340°. These rotational errors accumulate over time and

distort the wheelchair's belief of its position. Such beliefs could negatively impact mapping and localisation results.

Figure 5.3.1 below best illustrates the need for an IMU. In this scenario, we drove the wheelchair in and out of a corridor twice. During the second time that we drove into the corridor, the map of the original corridor was overlapped with the map of another corridor. The reason for this issue was most likely due to inaccurate angular updates. An IMU would most likely provide a better approximation for the robot's rotational movements and prevent such an error.

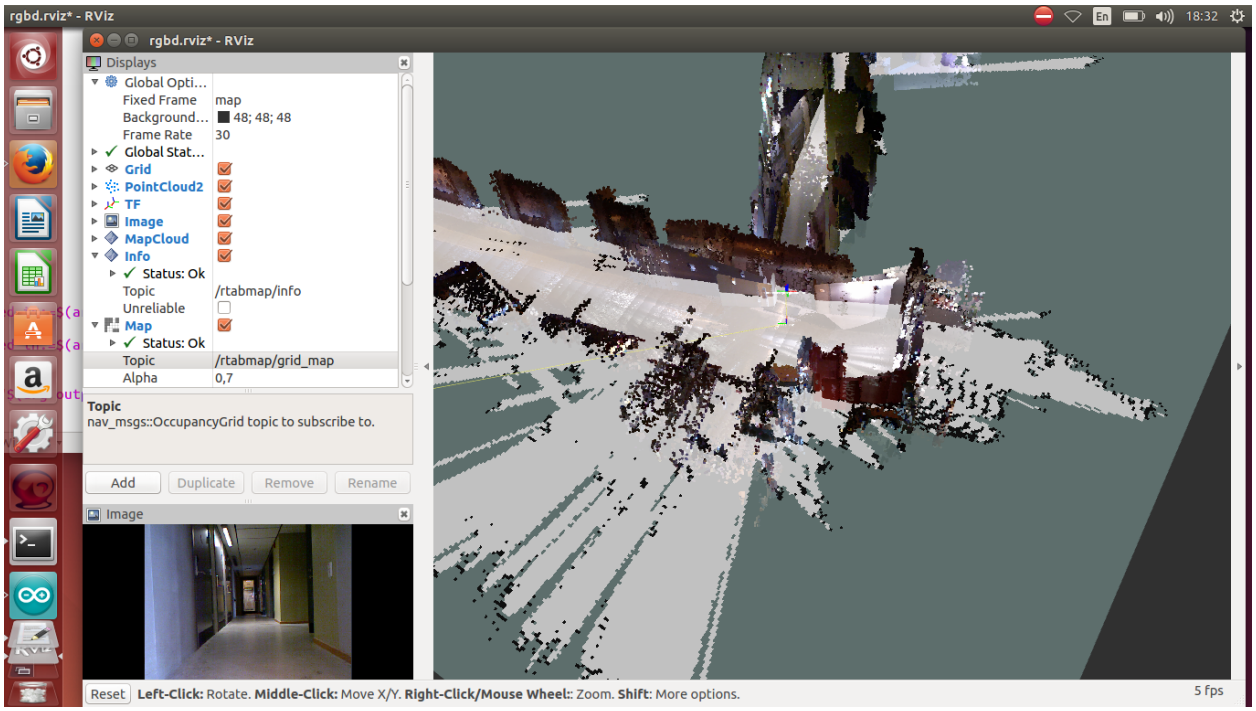



*Figure 5.3.1. Mapping error due to poor angular update*

An IMU (inertial measurement unit) could be incorporated to provide the robot with a more accurate odometry estimation. The ROS package robot_pose_ekf could be used to combine IMU and wheel odometry data to feed into Gmapping and the navigation stack. A new ROS topic may be created to send the IMU data from the Arduino to the base controller node, where the IMU data is processed alongside the odometry node. Alternatively, the IMU data could be processed with the odometry data in the Arduino and send out to the base_controller node via a topic of type nav_msgs/Odometry. Doing so could also provide a more accurate velocity update.

### 6.3.2 Incorporating Computer Vision

Reliance on simply the Kinect and odometry may not necessarily expose all of the potential dangers in a 2D map. For instance, manholes or staircases may not necessarily be detectable by laser scans. Computer vision could be used to identify these potential hazards and mark

these areas as occupied in the 2D occupancy grid. The vision_opencv package in ROS provides a bridge between the popular computer vision library OpenCV and ROS and could be a great starting point for such implementations.

### 6.3.3 Tuning the Parameters

In all of our mapping and navigation launch files, we used the default parameters for most nodes. The impact of these parameters on the robot's performance could be explored. For example, the depthimage_to_laserscan node has parameters that can change the timing between scans as well as the maximum and minimum range for the fake laser scan data. Future groups should first focus on tuning the move base parameters provided on the ROS wiki which can enable more effective path planning. In addition, the delay in the handle_cmd function of encoder.ino could be calibrated further for a more accurate conversion from the velocity commands from the move base node to the duration of the analogWrite signal on the X and Y pins of the Easy Rider.

### 6.3.4 Mechanical Improvements

The base for the Kinect does not provide enough stability and is only helpful for the testing stage of the project. For future use, we recommend a more stable platform that prevents rotation and can be installed at the same shaft for better detection of obstacles.

The circuits developed can be encased under the wheelchair to give it a more aesthetic look and prevent the user from tampering with the setup.

### 6.3.5 PID Controller

Currently our system employs an open loop control system for velocity control. As a result, there exists a large discrepancy between the velocity being published via the cmd_vel topic and the actual velocity travelled by the wheelchair. Allowing for a more accurate velocity control would most likely fix the poor performance of our move base node. In addition, a more accurate velocity control would enable the wheelchair to navigate narrow environments much more effectively.

A PID controller is a form of a closed loop control system that could potentially enable more precise velocity control. A second set of Arduino files has been constructed to utilize PID control. However, the gains of the proportional, derivative and integral terms have not been adjusted yet. Future groups could use a jack to elevate the wheelchair and monitor how close the wheelchair's velocity is to the setpoint, or target velocity, of the PID system using different parameters.

# 7. Conclusion

In conclusion, our project was an overall success. We were able to establish the hardware and software framework for autonomous navigation. Three different types of mapping have been set up on our robot and could easily be incorporated into future projects.

The next priority for future teams should be to tune the PID controller and navigation parameters for operational compliance. Once both of these parameters are tuned, many additional side project opportunities will arise from this project.

# 8. References


[1] Lund, Morten E. "Inductive Tongue Control Of Powered Wheelchairs". N.p., 2017. Web. 19 June 2017.

[2] Christensen, Henrik Vie, Garcia, Juan Carlos . "Infrared Non-Contact Head Sensor For Control Of Wheelchair Movements". N.p., 2017. Web. 19 June 2017.

[3] Guo. S, Cooper R.A, and Bonninger, M.L. "Development Of Power Wheelchair Chin Operated Force Sensing Joystick". *Ieeexplore.ieee.org*. N.p., 2017. Web. 19 June 2017.

[4] Fehr L, et al. "Adequacy Of Power Wheelchair Control Interfaces For Persons With Severe Disabilities: A Clinical Survey. - Pubmed - NCBI". *ncbi.nlm.nih.gov*. N.p., 2017. Web. 19 June 2017
.
[5] Simpson RC, et al. "How Many People Would Benefit From A Smart Wheelchair? - Pubmed - NCBI". *ncbi.nlm.nih.gov*. N.p., 2017. Web. 19 June 2017.

[6] Felzer, Torsten, and Rainer Nordmann. "Alternative Wheelchair Control". N.p., 2017. Web. 19 June 2017.

[7] Kairy, Dahlia et al. "Exploring Powered Wheelchair Users And Their Caregivers' Perspectives On Potential Intelligent Power Wheelchair Use: A Qualitative Study". N.p., 2017. Print.

[8] Cha, Sunjik. "My Personal Robotic Companion". *Sung's Blog*. WordPress.com , 2017. Web. 11 June 2017.

[9] Foote, Tully, and Melonee Wise. "Robots/Turtlebot". *ROS Wiki*. Open Source Robotics Foundation, 2017. Web. 16 June 2017.

[10] Kohlbrecher, Stefan, and Johannes Meyer. "Hector_Slam". *ROS Wiki*. Open Source Robotics Foundation, 2017. Web. 16 June 2017.

[11] Labbe, Mathieu. "Rtabmap". *ROS Wiki*. Open Source Robotics Foundation, 2017. Web. 14 June 2017.

[12] Rabaud, Vincent. "Gmapping". *ROS Wiki*. Open Source Robotics Foundation, 2017. Web. 15 June 2017.

[13] "Easy Rider". *HMC*. HMC International N.V., België, 2017. Web. 15 June 2017.

[14] Easy Rider- Installation Manual. 1st ed. Belgium: HMC International N.V., België, 2017. Print.

[15] Mihelich, Patrick, and Piyush Khandelwal. "Freenect_Launch". *Ros Wiki.* Open Source Robotics Foundation, 2017. Web. 21 June 2017.

[16] Rockey, Chad. "Depthimage_To_Laserscan". *ROS Wiki*. Open Source Robotics Foundation, 2017. Web. 21 June 2017.

[17] Bouchier, Paul and Mike Purvis, "Rosserial". *ROS Wiki.* Open Source Robotics Foundation, 2017. Web. 21 June 2017.

[18] Wang, Richard. "Simulating Robot Models for ROS". *Moorerobots.com*. N.p., 2017. Web. 19 June 2017. (Simulation)

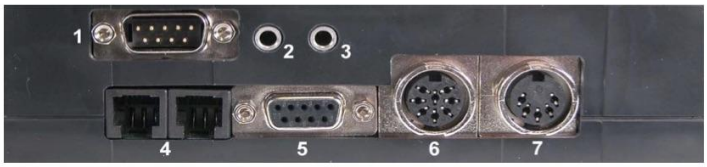
[19] "Setup RTAB-Map on Your Robot!", *ROS Wiki*, 2017. Open Source Robotics Foundation, 2017. Web. 12 June 2017.

[20] Ferguson, Michael and David Lu . "robot_pose_ekf". *ROS Wiki*. Open Source Robotics Foundation, 2017. Web. 16 June 2017.

# 9. Appendix

## 9.1 Appendix A: Easy Rider Interface Connections

**EASY RIDER INTERFACE CONNECTIONS**




| | | | | |
|---|---|---|---|---|
| **Number** | **1** | **Pin 1** | SW1 OUT/FORWARD | Max 0.5Amp 24V |
| **Name** | 5 SW OUT | **Pin 2** | SW2 OUT/BACK | Max 0.5Amp 24V |
| **Type** | SUBD 9P FEM. | **Pin 3** | SW3 OUT/LEFT | Max 0.5Amp 24V |
| | | **Pin 4** | SW4 OUT/RIGHT | Max 0.5Amp 24V |
| | | **Pin 5** | NOT CONNECTED | |
| | | **Pin 6** | FUNCTION OUT | Max 0.5Amp 24V |
| | | **Pin 7** | NOT CONNECTED | |
| | | **Pin 8** | COMMON | Max 1Amp 24V |
| | | **Pin 9** | NOT CONNECTED | |
| **Number** | **2** | **Pin 1** | BUTTON | 1K Input |
| **Name** | FUNCTION | **Pin 2** | GND | PTC Protected |
| **Type** | JACK 3.5mm | | | |
| **Number** | **3** | **Pin 1** | BUTTON | 1K Input |
| **Name** | ON/OFF | **Pin 2** | GND | PTC Protected |
| **Type** | JACK 3.5mm | | | |
| **Number** | **4** | **Pin 1** | GND | |
| **Name** | BUS | **Pin 2** | SCL | |
| **Type** | RJ-11 | **Pin 3** | GND | |
| | | **Pin 4** | SDA | |
| | | **Pin 5** | GND | |
| | | **Pin 6** | 24V SW | Max 0.5Amp |
| **Number** | **5** | **Pin 1** | SW1/FORWARD | 10K Input |
| **Name** | 5 SW IN | **Pin 2** | SW2/BACK | 10K Input |
| **Type** | SUBD 9P FEM. | **Pin 3** | SW3/LEFT | 10K Input |
| | | **Pin 4** | SW4/RIGHT | 10K Input |
| | | **Pin 5** | ON/OFF SW | 1K Input |
| | | **Pin 6** | FUNCTION SW | 1K Input |
| | | **Pin 7** | 24V CST | Max 50mAmp |
| | | **Pin 8** | GND | |
| | | **Pin 9** | 24V SW | Max 50mAmp |
| **Number** | **6** | **Pin 1** | GND | |
| **Name** | JOY IN | **Pin 2** | 24V SW | Max 50mAmp |
| **Type** | AUDIO DIN 8P | **Pin 3** | FUNCTION SW | 1K Input |
| | | **Pin 4** | REF 5V | 4K7 to 5V |
| | | **Pin 5** | 10V JOY SUPPL | Max 50mAmp |
| | | **Pin 6** | JOY X SIG. (DRIVE) | 5V +- 1V |
| | | **Pin 7** | JOY Y SIG. (DIR) | 5V +- 1V |
| | | **Pin 8** | JOY Z SIGNAL | 5V +- 1V |
| **Number** | **7** | **Pin 1** | GND | |
| **Name** | FUNCTION - ON/OFF | **Pin 2** | 24V SW | Max 50mAmp |
| **Type** | AUDIO DIN 5P | **Pin 3** | FUNCTION | 1K Input |
| | | **Pin 4** | ON/OFF | 1K Input |
| | | **Pin 5** | 24V CST | Max 100mAmp |

[14]

## 9.2 Appendix B: encoder_test.ino

```
*Creates node for updating odometry and detection of joystick
 *movement.
 *
 *  this node publishes odometry data and subscribes to joystick
 *  data
 *  DO NOT USE SERIAL MONITOR OR SERIAL PRINT ANYTHING WHEN CONNECTED TO
SERIAL NODE
 */

//Include Relevant ROS libraries
#include <ros.h>
#include <ros/time.h>
#include <tf/tf.h>
#include <tf/transform_broadcaster.h>
#include <geometry_msgs/Point.h>

#include <geometry_msgs/Twist.h>


//Include other header files
#include <Encoder.h>
#include <geometry_msgs/PointStamped.h>
#include <nav_msgs/Odometry.h>


//define input pins
const byte encLPin_A = 2;
const byte encLPin_B = 7;
const byte encRPin_A = 3;
const byte encRPin_B = 8;

const byte joyPin_x = A0;
const byte joyPin_y = A1;

//define output pins
const byte outPin_x = 10;
const byte outPin_y = 9;


//define constants (units of measurements are in meters)
```

```
const double WHEEL_DIAMETER  = 0.34;
const double PULSES_PER_REVOLUTION = 96.0;
const double AXLE_LENGTH = 0.55;

//Declare global variables
double x = 0.0;
double y = 0.0;
double theta = 0.0;
int x_vel = 0;
int y_vel = 0;


//double  L_ticks;
//double  R_ticks;
double dist_left;
double dist_right;
double left_ticks;
double right_ticks;
double s_mean;
double cos_current;
double sin_current;
double s_theta;
double right_minus_left;
double right_plus_left;
double MUL_COUNT;

long positionLeft  = 0;
long positionRight = 0;

char base_link[] = "/base_link";
char odom[] = "/odom";

int analogVal;
int analogOut;

int analogVal2;
int analogOut2;



// Define structures
struct position
{
```

```
  float x;          /* meter */
  float y;          /* meter */
  float theta;      /* radian (counterclockwise from x-axis) */
};


// instantiate struct
struct position current_position;


//Helper functions
void initialize_odometry(){
  current_position.x = 0.0;
  current_position.y = 0.0;
  current_position.theta = 0.0;
}

void handle_cmd(const geometry_msgs::Twist& cmd_msg){
//  digitalWrite(ledPin, HIGH-digitalRead(ledPin));   // blink the led
  x_vel = int(cmd_msg.linear.x * 10);
  y_vel = int(cmd_msg.angular.z * 10);


  if ((x_vel != 0) || (y_vel != 0)) {
          x_vel = map(x_vel,5,-5,103,152);
          x_vel = constrain(x_vel,103,152);
          y_vel = map(y_vel,5,-5,103,152);
          y_vel = constrain(y_vel, 103,152);

          analogWrite(outPin_x, x_vel);
          analogWrite(outPin_y, y_vel);

          //this delay can be calibrated
          delay(850);
  }
  //set them to neutral
  analogWrite(outPin_x, 127);
  analogWrite(outPin_y, 127);

}
```

```
//declare ROS objects
geometry_msgs::PointStamped pos_odom;
geometry_msgs::TransformStamped t;

ros::NodeHandle nh;
ros::Publisher Point_send("odom_awc", &pos_odom);

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

ros::Time current_time;

tf:: TransformBroadcaster broadcaster;

//initialise object Encoder
Encoder knobLeft(encLPin_A, encLPin_B);
Encoder knobRight(encRPin_A, encRPin_B);



void setup() {
  // put your setup code here, to run once:

  // initialize pins
  pinMode(outPin_x, OUTPUT);   // sets the pin as output
  pinMode(outPin_y, OUTPUT);   // sets the pin as output

  // Serial communication
  Serial.begin(57600);

  //Serial.println("Two Knobs Encoder Test:");
  initialize_odometry();
  MUL_COUNT = (PI*WHEEL_DIAMETER)/PULSES_PER_REVOLUTION;

  //initialize ROS objects
  nh.initNode();
  nh.advertise(Point_send);
  nh.subscribe(sub);

  // center joystick at reference
  delay(2000);
  //int cJoy = 127; //analogRead(joyPin_ref)/8; potentially change it
  int cJoyX = 127;
  int cJoyY = 127;
```

```
  delay(2000);
  analogWrite(outPin_x, cJoyX);
  analogWrite(outPin_y, cJoyY);

}

void loop() {
  // put your main code here, to run repeatedly:

  DetectJoystickMovement();

  //update current position and publish
  UpdatePosition();


  nh.spinOnce();
  delay(1);
}

void UpdatePosition() {
  /* sample the left and right encoder counts as close together */
  long newLeft, newRight;
  newLeft  = knobLeft.read();
  newRight = knobRight.read();
//  Serial.println("New data");
//  Serial.println(newLeft);
//  Serial.println(newRight);
//  Serial.println(positionLeft);
//  Serial.println(positionRight);
//
  // don't forget to initialise positionLeft and positionRight to 0
  if (newLeft != positionLeft || newRight != positionRight) {

          dist_right =  (newRight-positionRight) *  (double)MUL_COUNT;
          dist_left  =  (newLeft-positionLeft)  * (double)MUL_COUNT;

          /* and update last sampling for next time */
          positionLeft = newLeft;
          positionRight = newRight;

          right_minus_left = dist_right - dist_left;
          right_plus_left = dist_right + dist_left;
```

```
        s_mean=(right_plus_left)/2;


        /*************************/
        /* Calculate (x,y,theta) */
        /*************************/

        current_position.theta += right_minus_left / AXLE_LENGTH;
        current_position.x += s_mean * cos(current_position.theta);
        current_position.y += s_mean * sin(current_position.theta);


        /* Keep in the range -PI to +PI */
        while(current_position.theta > PI)
        current_position.theta -= (2.0*PI);
        while(current_position.theta < -PI)
        current_position.theta += (2.0*PI);

        pos_odom.header.frame_id = odom; //msg->header.frame_id;
        pos_odom.point.x = (current_position.x);
        pos_odom.point.y = (current_position.y);
        pos_odom.point.z = current_position.theta;

//      Serial.println("New Data");
//      Serial.println(current_position.x);
//      Serial.println(current_position.y);
//      Serial.println(current_position.theta);
//      Serial.println(dist_left);
//      Serial.println(dist_right);

  }
  Point_send.publish(&pos_odom);

}


void DetectJoystickMovement(){
  //int cJoy = 127;
  int cJoyX = 125;
  int cJoyY = 126;
```

```
  analogVal = analogRead(A0);
//  Serial.println("analogVal");
//  Serial.println(analogVal);

  analogVal = map(analogVal, 1024, 0, 412, 606);
  analogVal = constrain(analogVal, 412, 606);

  analogOut = analogVal / 4;
  analogWrite(outPin_x,analogOut);


  analogVal2 = analogRead(A1);
//  Serial.println("analogVal2");
//  Serial.println(analogVal2);

  analogVal2 = map(analogVal2, 0, 1024, 412, 606);
  analogVal2 = constrain(analogVal2, 412, 606);

  analogOut2 = analogVal2 / 4;
  analogWrite(outPin_y,analogOut2);

}
```

## 9.3 Appendix C: Launch File Instructions

Some Useful Commands in Ubuntu:
Switch window =
Alt Gr + Ctrl + arrow keys
New terminal window=
Ctrl + Alt + T
New terminal tab =
Ctrl + Shift + T
Kill terminal process (use this to kill the launch files)=
Ctrl + C

Useful Debugging Tools (Use these while the launch files are running to debug):
View TF Frames (this will create pdf in your current working directory) =
$ rosrun tf view_frames
$ evince frames.pdf

View all nodes graphically =
$rqt_graph

View current topics
$ rostopic list

View current nodes
$ rosnodes list

Launch File Instructions:
Before you launch please make sure of the following
- encoder_test.ino has been uploaded to the Arduino board.
- Kinect is connected to the 12V power supply.
- Kinect is connected to computer
- Arduino is connected to 12V power supply
- Arduino is connected to computer
- Make sure the wheelchair has been switched off.
- Make sure you source all your terminals.
How to source your terminal (do this whenever you open a new terminal):
$ cd ~/catkin_ws
$ source ./devel/setup.bash

How to launch Hector Mapping (this one's not very good with the Kinect):
Terminal 1:
$ cd ~/catkin_ws/src/my_personal_robotic_companion/launch/includes
$ roslaunch kinect_laser.launch

Terminal 2:
$ roslaunch hector_slam_launch tutorial.launch

How to launch Gmapping (mapping):
Terminal 1:
$ roslaunch my_personal_robotic_companion robot_config.launch

Terminal 2:
$ roslaunch my_personal_robotic_companion gmapping.launch

Terminal 3:
$roslaunch my_personal_robotic_companion rviz_gmapping.launch

Terminal 4: (optional, only if you want to control the chair with your keyboard)
$roslaunch my_personal_robotic_companion teleop.launch

Termianl 5: (save map after you're done)
$ cd ~/catkin_ws/src/my_personal_robotic_companion/maps
$ rosrun map_server map_saver -f <map_name>

How to launch Autonomous Navigation:
Terminal 1:
$ roslaunch my_personal_robotic_companion robot_config.launch

Terminal 2: (don't forget to change the path to the map file you want to use in the launch file)
$ roslaunch my_personal_robotic_companion amcl_demo.launch

Terminal 3: (Sometimes the window on the right will be blank in RViz. Just zoom out and refocus the camera.)
$roslaunch my_personal_robotic_companion rviz_launcher.launch

How to launch Rtabmap (without odometry):
Terminal 1:
$roslaunch freenect_launch freenect.launch depth_registration:=true

Terminal 2:
$roslaunch rtabmap_ros rtabmap.launch rtabmap_args:"--delete_db_on_start"
or you could launch Rtabmap standalone without ROS using
$ rtabmap

Rtabmap (with odometry):
Terminal 1:
$ roslaunch my_personal_robotic_companion rtab_robot_config.launch

Terminal 2:
$ roslaunch my_personal_robotic_companion full_rtab.launch rviz:=false rtabmap:=false arg:="--delete_db_on_start"
- This launch file takes multiple arguments. Above is simply an example of how you could launch the file. Setting rviz and rtabmap to false will stop the launch file from opening Rtabmap's default visualisation setups. By default this launch file will save the map database in the directory, ~/.ros/rtabmap.db. The arg statement simply deletes the existing database in this directory. You can localise your robot by using the argument, "localisation:=true".

Terminal 3:
$ roslaunch my_personal_robotic_companion rviz_rtab.launch
The move base node is already active, so you should be able to send 2D Nav goals in this RViz window.