NGEE ANN POLYTECHNIC

Diploma in Engineering Science

Year 3 Final Year Project
Final Report

Year 2017

Project Title:
Autonomous WheelChair (NP-08)

Student Names & IDs
Muhammad Mustaqeem - S10163795J
Ong Jia Jie - S10164682G
Teoh Yi Zheng - S10165158F
Yuen Sheng Hao - S10165027G

Supervisor Name(s)
Lim Yew Kin
Lim Ching Yang

Content

Abstract

This document is a manual that provides its readers a step-by-step instruction on how to start on this project, as well as a detailed analysis and explanation on the different technologies implemented in this project in order for the wheelchair to work autonomously. The main topics that will be covered in this document includes the programming and coding techniques to map out a location and transport the passenger from one location to another.

z This project makes use of a multitude of components and principles to allow the wheelchair to move autonomously and account for possible errors. Additionally, we will be using Ubuntu, a Linux Operating System, and the Robotics Operating System (ROS) as our programmer for this project.

1. Introduction

Hospitals utilise wheelchairs heavily for transporting patients with mobility problem between clinics and laboratories. And often porters are called upon to wheel the patients to the destinations. Our main focus of the project is to be able to program the wheelchair to operate autonomously to reduce workload.

In this project, our aim is to understand and do an in-depth analysis of the general components of our autonomous wheelchair, and be able to combine our knowledge to bring our project to completion. Additionally, through this project we will understand the uses of Ubuntu and ROS, and its application on robotics.

This guide will introduce one to the basics required in making an autonomous mobile vehicle. This includes:

- ROS
- Linux
- Customising an electric wheelchair
- Sensor functions
- And many more…

Tutorials will be available in this guide to help further enhance the knowledge in using ROS and its many functions. Relevant links will be provided to locate & install the application required to make the autonomous robot vehicle.

We hope that this guide will enhance the learning experience of anyone engaged in projects relevant to building an autonomous wheelchair.

2. Project Outline and Objectives

In this project, we are required to learn and understand how to use Ubuntu and the ROS to program the components of the wheelchair. At the same time, we had to experiment and understand the complex wiring of a given joystick-controlled wheelchair, so that we can rewire it using a switch circuit and a Arduino board to manipulate the clutch system. The understanding of Electrical, Mechanical, and Computer Engineering is essential as the basic understanding of electrical components, motors, and computer programming is the foundation of this project.

Our project will be consisting of various components such as the joystick-controlled wheelchair, the Sabertooth Motor Driver, and the Arduino board.

The objective of our project is to be able to program the wheelchair to move through commands from the user, whether its authorized from the computer or through voice command.

3. Autonomous Wheelchair (Documentation)

What is LINUX?

LINUX is an operating system. It is the software on a computer that enables applications and the computer operator to access the devices on the computer to perform desired functions. The operating system relays instructions to, for instance, the computer processor. The processor performs the instructed task, then sends the results back to the application via the operating system. Linux is very similar to the operating systems such as Windows and OS X.

For more information, checkout the website below:
https://www.linux.com/learn/new-user-guides/376-linux-is-everywhere-an-overview-of-the-linux-operating-system

What is UBUNTU?

Ubuntu is built on Debian's architecture and infrastructure, to provide Linux server, desktop, phone, tablet and TV operating systems. Ubuntu releases updated versions predictably every six months,and each release receives free support for nine months (eighteen months prior to 13.04)with security fixes, high-impact bug fixes and conservative, substantially beneficial low-risk bug fixes. The first release was in October 2004.

What is ROS?

ROS (Robot Operating Systems) is a Linux based software framework for operating robots. This framework uses the concept of packages, nodes, topics, messages and services. (Provides libraries and tools to help software developers create robot applications.) It provides hardware abstraction, devices drivers, libraries, visualizers, message-passing, package management and more.

For more details on ROS, please visit http://*wiki.ros.org*.

Requirements

Here is the list of software that we will be using for the robot:

- Windows
    - Unetbootin (Burning the ISO image of Ubuntu onto the flash drive)
    - Virtualbox (For learning & testing purposes)
- Ubuntu 14.04

- Python 2.7.3
- ROS-desktop-full (Indigo)

Installation

Ubuntu in Virtualbox

Virtualbox is a cross-platform virtualization application. It install on your existing operating system to operate another operating system at the same time. VirtualBox is deceptively simple yet also very powerful. It allows testing of the operating system, If something goes wrong (e.g. after installing misbehaving software or infecting the guest with a virus), one can easily switch back to a previous snapshot and avoid the need of frequent backups and restores. If you do not plan to install linux operating system on your computer, you can download the virtualbox to still be able to use linux.
We would first use Virtualbox to try out Ubuntu, a linux operating system (OS), in a virtual environment on top of Windows. (There are a few limitations with using Virtualbox like slow connection speeds and many other restrictions.)

Download the application file from this link:
https://www.virtualbox.org/wiki/Downloads
Guide to installing & using Virtualbox: (Everything you need to know is covered here)
http://www.makeuseof.com/tag/how-to-use-virtualbox/

Use the following during the setup: (Or anything you like)

Computer name: fyp-wheelchair Username: fyp-wheelchair P/S: fyp2017

Network Setting

http://cyaninfinite.com/tutorials/configuring-netapt-on-vb-tutorial/

VNC

http://cyaninfinite.com/tutorials/linux-vnc-guide/

Dual Booting Ubuntu (Linux)

[If you prefer to dual boot into Ubuntu instead later, read on. Or else, you can skip this part.] To do this we had to create a bootable Ubuntu 14.04 USB flash drive. You need at least 1GB of free memory in your flash drive.

1. Partition the hard drive on your computer (unless you have multiple drives)(80GB recommended)
2. Download the Ubuntu image driver(.iso) on your computer at http://releases.ubuntu.com/14.04/ (You can choose the suitable download for your specified operating system)
3. Download the usb installer from https://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/
4. Insert your flash drive
5. Once download is done open the application you have downloaded
6. In the select distribution box, Select “UBUNTU” in the select Version box, Select “version 14.04, 64 bits” (Most prefered)
7. Then at the type select the “USB drive” and choose the specific drive of your flash drive & click okay.

After doing the above, installation of the bootable ubuntu 14.04 will commence on your flash drive.

1. Insert the Flash drive into a computer you want to instal ubuntu on.
2. Make sure that the computer is Off when doing this, or you can just restart your computer.
3. During start-up of the computer, press F2 to enter the bios to change the boot sequence
4. Select the Boot tab. You will see the flash drive, make sure it is listed as number 1 on the list. You can do this by bringing the cursor on the Flash drive and press F6 to bring it up higher on the list.
5. Save the settings
6. It will immediately boot from the USB Driver, It will ask you to select a few options, Click Install ubuntu.

Check this video tutorial for step by step instructions for the installation of Ubuntu https://www.youtube.com/watch?v=hOz66FC0pWU

ROS

We will be using a terminal in install ROS.

First, we would setup your computer to accept software from packages.ros.org

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01FA116
```

Update your Debian package index to the latest:

```
$ sudo apt-get update
```

We will be installing the Full Desktop ROS, which will include: ROS, rqt, rviz, robot-generic libraries, 2D/3D simulators, navigation and 2D/3D perception.

```
$ sudo apt-get install ros-indigo-desktop-full
```

Before you can use ROS, you will need to initialize *rosdep*. *rosdep* enables you to easily install system dependencies for source you want to compile and is required to run some core components in ROS.(Do this when downloading new packages that causes errors such as “Invoking "make -j8 -l8" failed” during catkin_make)

```
$ sudo rosdep init
$ rosdep update
```

To make life easier, we would add this line so that the ROS environment variables are automatically added to your bash session every time a new shell is launched:

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

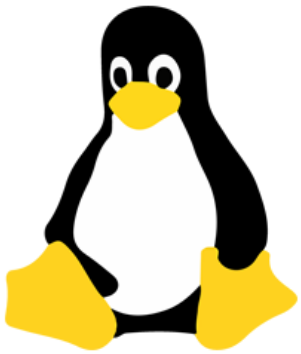
NOTE: Before starting any ros nodes you have to run **$ roscore**

References:

http://wiki.ros.org/indigo/Installation/Ubuntu

https://www.youtube.com/watch?v=TQ1Q2AUlTgw3r

BASICS



LINUX

In this chapter, we will explore the basics of navigating around Linux, followed by ROS.

Keyboard shortcuts

Making your life easier. (Applicable to most Linux distribution.)

Open a new terminal window.	***Ctrl + Alt + T***
Open a new terminal tab.	***Ctrl + Shift + T***
Kill a process.	***Ctrl + C***
Suspend an application.	***Ctrl + Z***
Closes the focused window.	***Alt + F4***
Permanently deletes a file without sending it to the Trash.	***Shift + Del***
Cycles between open applications.	***Alt + Tab***
Locks the screen.	***Ctrl + Alt + L***
Changing to tty (terminal) session.	***Ctrl + Alt + <F1-F6>***
Changing to xorg (GUI) session.	***Ctrl + Alt + F7***

More shortcuts here:

- http://lifehacker.com/5743814/become-a-command-line-ninja-with-these-time-saving-shortcuts
- http://www.howtogeek.com/howto/22283/four-ways-to-get-instant-access-to-a-terminal-in-linux/

EXTRAS

What does *tty* stands for?

Early user terminals connected to computers were electromechanical teleprinters or teletypewriters (TeleT`Ypewriter, TTY), and since then TTY has continued to be used as the name for the text-only console.

There are 6 virtual consoles in Ubuntu accessed by the keyboard shortcuts Ctrl+Alt+F1 to Ctrl+Alt+F6. You can move away from a text-only console (move the console to the background) by using the keyboard shortcut Ctrl+Alt+F7.

http://askubuntu.com/questions/481906/what-does-tty-stand-for

Terminals

It is a powerful tool for creating packages, compiling, running, etc. The terminal we are using are bash shell, which is the most common used in Linux distribution.

Learn how to use the terminal: http://www.howtogeek.com/140679/beginner-geek-how-to-start-using-the-linux-terminal/

Terminal Commands

Refer to Appendix A.

ROS

Setting up a workspace

Before using ROS, we will have to setup a workspace for our project & packages to reside in. We will be using catkin to create the workspace, which would be called catkin_ws. (Catkin is a build tool used by ROS, which is similar to CMake plus the concept of workspaces). A catkin workspace is a folder where you modify, build, and install catkin packages together all at once.

First start by creating a catkin workspace in our home directory:

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

```
$ catkin_init_workspace
```

After that, we would build the workspace (though it's empty):

```
$ cd ~/catkin_ws/
$ catkin_make
```

The catkin_make command is a convenience tool for working with catkin workspaces. If you look in your current directory you should now have a 'build' and 'devel' folder. Inside the 'devel' folder you can see that there are now several setup.*sh files.

Sourcing any of these files will overlay this workspace on top of your environment. Before continuing source your new setup.*sh file:

```
$ source devel/setup.bash
```

To make sure your workspace is properly overlayed by the setup script, make sure ROS_PACKAGE_PATH environment variable includes the directory you're in.

```
$ echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks
```

After this, the catkin_ws folder would be created and has been initialized successfully. This is very crucial steps as every programs MUST be created, stored inside this folder(catkin_ws/src), else the external programs are not able to link to internal operating file system(ROS).
DO NOT STORE PROGRAMS INSIDE OTHER FOLDERS EXCEPT catkin_ws/src.

http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
http://wiki.ros.org/catkin/workspaces
http://jbohren.com/articles/gentle-catkin-intro/

Creating a new package

The catkin_ws folder is used to store **packages** which contains the relevant programs and library folders required to run each of the individual hardware(i.e sensors,motors) of the robot.
The **executable file** is usually named as **xxx_node**. This file can either be a c++ or python program file.

A basic package should consist of 1 folder and 2 files.

```
CMakeLists.txt
Package.xml
Src (folder which contains the executable file)
```

Some hardware components require several executable file to be run at the same time for it to work. In order to do that, we would usually have to manually run the individual nodes. However, we can make use of the feature *roslaunch* built into ros to enable us to automatically run all the required nodes. In order to do this, we have to create a file called xxx.launch. This is usually placed inside

```
launch/xxx.launch
```

An example launch file is shown below

```
<launch> // to indicate that this is a launch file
  <arg name="joy_config" default="ps3" />
  <arg name="joy_dev" default="/dev/input/js0" />
        <arg name="config_filepath" default="$(find teleop_twist_joy)/config/$(arg joy_config).config.yaml" /> // setting the required arguments

   <node pkg="joy" type="joy_node" name="joy_node"> // package / node file / node file
    <param name="dev" value="$(arg joy_dev)" />
    <param name="deadzone" value="0.3" />
    <param name="autorepeat_rate" value="20" />
  </node> // setting required parameters | end of node

  <node pkg="teleop_twist_joy" name="teleop_twist_joy" type="teleop_node">
    <rosparam command="load" file="$(arg config_filepath)" />
  </node>
</launch> // indicate end of launch file
```

To run the launch file

```
$ roslaunch <pkg name> <xxx.launch>
```

NOTE: A launch file **must be placed** inside a folder which is recognised as a **package.**

Visualisation
To visualise the interactions between the various nodes, we will use *rqt_graph* to do so. This will allow us to see how the nodes subscribe or publish to the various topics.

```
$ rqt_graph
```

Commonly used ROS Commands
Refer to Appendix B.

Required Packages

ROS packages

We would download the relevant ros packages using apt-get. The following package will be installed system-wide.

Navigation

Gmapping
GMapping is a ROS package which contains ROS node called slam_gmapping which provides laser-based SLAM (Simultaneous Localisation and mapping).
Visit for more details: http://wiki.ros.org/gmapping
Used for.generating the map from the LIDAR data. (/scan topic)

```
$ sudo apt-get install ros-indigo-gmapping
```

After installing gmapping package, we would have to compile it before using it.

```
$ rosmake gmapping
```

To run this package, use the following command:

```
$ rosrun gmapping slam_gmapping scan:=base_scan
```

where *base_scan* will be your LIDAR topic. (E.g. If your LIDAR publishes a */scan* topic, the parameter would be *scan:=scan*)

Map server
Provides map_server ROS node that offers map data as ROS service and provides map_saver command to save generated maps. The same map is then used with RViz (3D Visualisation tool for ROS)
Visit for more details: http://wiki.ros.org/map_server
For saving the map generated from gmapping.

```
$ sudo apt-get install ros-indigo-map-server
```

To save the map generated from gmapping, run the following command:

```
$ rosrun map_server map_saver
```

AMCL (Adaptive Monte Carlo Localisation)
AMCL is a probabilistic localisation system for robots moving in 2D. It uses a particle filter to track the position of the robot against a known map. In this known map, the algorithm estimates the position and orientation of robot and senses the movement. This is all packaged in the the AMCL node for ROS.
Visit for more details: http://wiki.ros.org/amcl
Note: Only works with laser scans.
For localisation of robot.

```
$ sudo apt-get install ros-indigo-amcl
```

Move base
The move_base package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base. The move_base node links together a global and local planner to accomplish its global navigation task.
Visit for more details: http://wiki.ros.org/move_base
For moving the robot from one point to another.

```
$ sudo apt-get install ros-indigo-move-base
```

Web page server (Jetty)

As we need to cross platforms from Android to Linux Ubuntu communication, we have created a webpage for the Autonomous wheelchair so that this robot is able to receive commands from our html web page as well as the android interface. This webpage allows us to cancel the destination of the wheelchair, stop the wheelchair (emergency), move to certain destinations, etc.

The webpage has a two-way communication instead of one-sided as it receives commands from the android and sends commands to wheelchair. To create a webpage, you need to setup a server, html, javascript files to go online on a router instead of only using localhost. Bigger web pages require larger memory space. The Autonomous Wheelchair will be using jetty as the web server. In the jetty folder, it will store the html file and javascript file.

Basically, the html will load onto the web server, and the web server will be accessible from other devices such as the android which is connected to the server. Thus, establishing connection between the devices.

1. we need to learn how to write html, javascript.

To learn javascript,html,css, visit: https://www.w3schools.com/

2. Install rosbridge_suite

http://wiki.ros.org/rosbridge_suite

http://wiki.ros.org/rosbridge_suite/Tutorials/RunningRosbridge

3. Download ROS javascript library into web page folder
4. Test communications between the Rosbridge_server and the html
5. Once it runs and works, set up Jetty server to allow the web page to be accessed by android

Rosbridge

Rosbridge protocols is a specification for sending JSON based commands to ROS.

The specification is programming language and transport agnostic. The idea is that any language or transport that can send JSON can talk the rosbridge protocol and interact with ROS. The protocol covers subscribing and publishing topics, service calls, getting and setting params, and even compressing messages and more.

The rosbridge_suite package is a collection of packages that implement the rosbridge protocol and provides a WebSocket transport layer.

The packages include:

- rosbridge_library - The core rosbridge package. The rosbridge_library is responsible for taking the JSON string and sending the commands to ROS and vice versa.

- rosapi - Makes certain ROS actions accessible via service calls that are normally reserved for ROS client libraries. This includes getting and setting params, getting topics list, and more.
- rosbridge_server - While rosbridge_library provides the JSON<->ROS conversion, it leaves the transport layer to others. Rosbridge_server provides a WebSocket connection so browsers can "talk rosbridge." Roslibjs is a JavaScript library for the browser that can talk to ROS via rosbridge_server.

To install rosbridge which is available as a debian use following:

```
$Sudo apt-get install ros-indigo-rosbridge-server
```

Rosbridge 2.0 is fully catkinized and the build procedure is similar for other Catkin-based packages.

1. Install the rosbridge dependency rosauth

```
$sudo apt-get install ros-indigo-rosauth
```

2. Source ROS

```
$source /opt/ros/indigo/setup.bash
```

3. Create workspace(not needed if workspace already created)

```
$mkdir catkin_ws
```

4. Clone rosbridge suite

```
$mkdir ~/catkin_ws/src
$cd ~/catkin_ws/src
$git clone https://github.com/RobotWebTools/rosbridge_suite.git
```

5. Build rosbridge

```
$cd ~/catkin_ws
$catkin_make
```

6. Add the freshly installed rosbridge to your ROS path

```
$cd ~/ws
$source devel/setup.bash
```

Running Rosbridge

After installing ROS and rosbridge, you need to make sure your system is aware of the packages. To set up your environment for ROS and rosbridge:

```
$source /opt/ros/indigo/setup.bash
```

All that's left is to run rosbridge. To launch rosbridge and its packages like rosbridge_server and rosapi, a launch file is included in the install. To launch the file, run:

```
$roslaunch rosbridge_server rosbridge_websocket.launch
```

The default port that the rosbridge_server will run will be at port:9090
However, it can be configured by setting the ~/port param in ROS

Download ROS JavaScript library

Enter the webpage folder and store the html file inside it, also create a folder called js inside the webpage folder which will be used to store all javascript codes
Inside the js folder, create a text file and name it eventemitter2.min.js and go to the link: http://cdn.robotwebtools.org/EventEmitter2/current/eventemitter2.min.js and copy all the text and paste into the eventemitter2.min.js
Next, create another text file, naming it roslib.min.js, and copy all text from the link below and paste into the roslib.min.js file.
http://cdn.robotwebtools.org/roslibjs/current/roslib.min.js
In addition, after copying and pasting the js file, REMEMBER to change the src address in line 6 and 7 of the sample html accordingly to where you have saved the js file in.

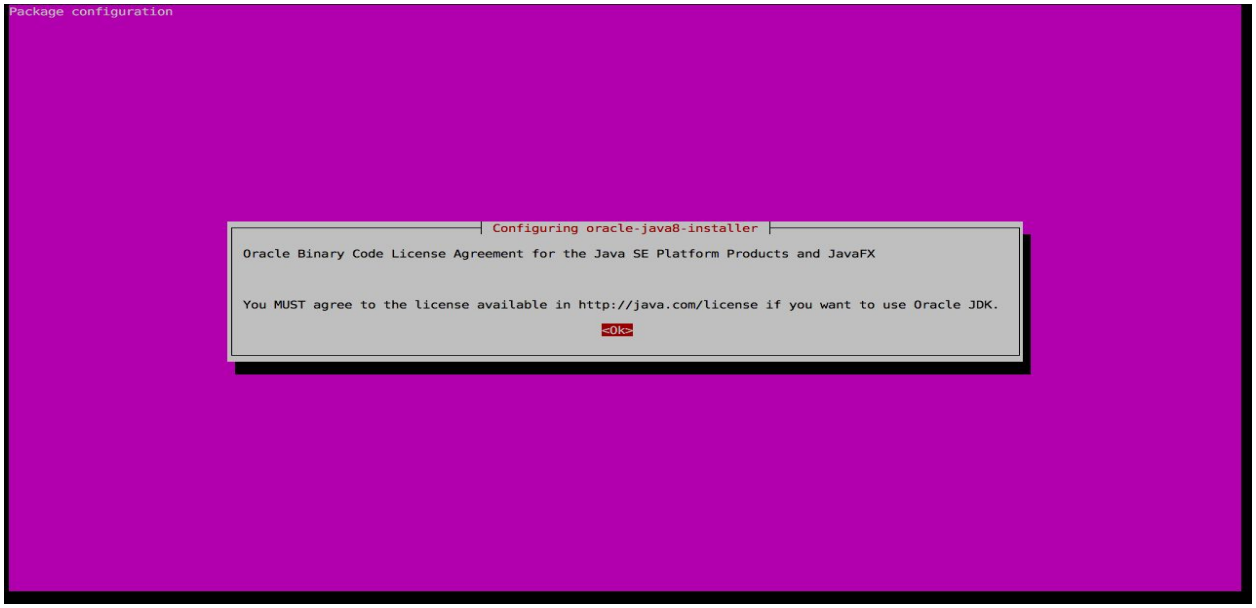
Jetty server setup

Before installing jetty, we need to install Java as jetty requires Java to run.

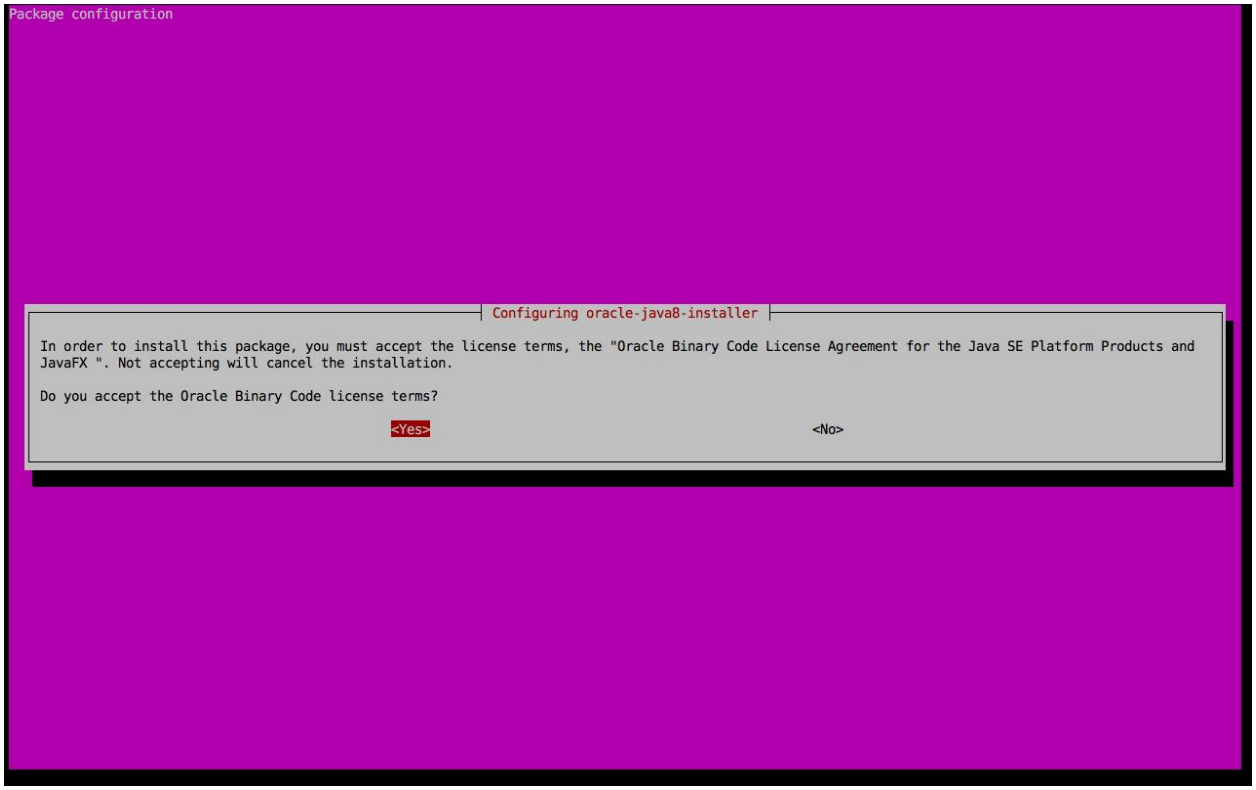
```
$sudo apt-get update
$sudo apt-get upgrade
$sudo add-apt-repository ppa:webupd8team/java
$sudo apt-get update
$sudo apt-get -y install oracle-java8-installer
```

Next we need to accept the license using:

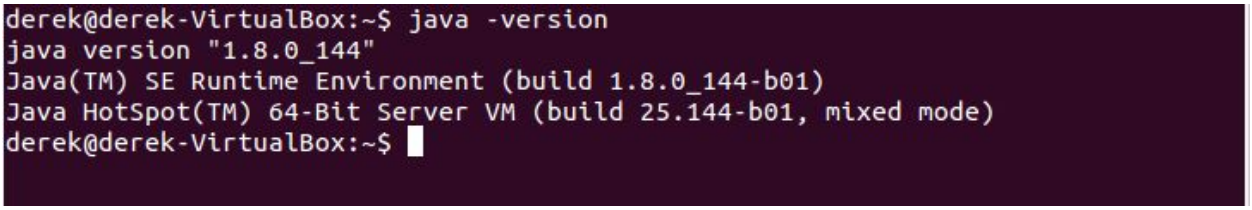
Package Configuration Choose OK



Accepting Oracle Binary Code License Terms. Choose Yes



After installing, running the java-version in a terminal should output something similar to:



Once Java is installed in Ubuntu, we can move on to installing Jetty

Jetty 9.3 Features

- HTTP2 support
- Application Layer Protocol Negotiation
- Server Name Indication (SNI) support for TLS/SSL negotiation. This enables jetty to use certificates with one main domain add multiple Subject Alternate Name.
- Targeted for Java 8. This change is due to the SNI extension on Java 8 API and HTTP specification need for TLS ciphers that are only available in Java 8.

Download the jetty 9.3.12 package using:

```
wget -c
http://repo1.maven.org/maven2/org/eclipse/jetty/jetty-distribution/9.3.12.v20160915/jetty-distri
bution-9.3.12.v20160915.tar.gz
```

Extract the jetty package:

```
tar xzf jetty-distribution-9.3.12.v20160915.tar.gz
```

Rename the directory to jetty and move it to /opt:

```
$ mv jetty-distribution-9.3.12.v20160915 jetty9

 $ sudo mv jetty9 /opt
```

We will create a user and group named jetty that will be used to run Jetty. Create the group first:

```
$ sudo addgroup --quiet --system jetty
```

Create the jetty system user.

```
$ sudo adduser --quiet --system --ingroup jetty --no-create-home --disabled-password jetty
```

Modify the /etc/passwd entry to change home and group for jetty user.

```
$ sudo usermod -c "Jetty 9" -d /opt/jetty9 -g jetty jetty
```

Change ownership of /opt/jetty9 directory to user jetty and group jetty.

```
$ sudo chown -R jetty:jetty /opt/jetty9
```

Change the permission of /opt/jetty9 directory.

```
$ sudo chmod u=rwx,g=rxs,o= /opt/jetty9
```

Configure Jetty 9

Create logs directory

We need to create directory for Jetty 9 to store logs:

```
$ sudo mkdir /var/log/jetty9
```

And change the ownership of the newly created directory to user jetty and group jetty:

```
$ sudo chown -R jetty:jetty /var/log/jetty9
```

Create a Default Configuration File for Jetty

We will create a default configuration file for jetty. This file will be read by jetty init script and also the crontab script. Create a new file /etc/default/jetty9 with content below:

Using

```
$sudo gedit /etc/default/jetty9
```

```
NO_START=0
VERBOSE=yes
JETTY_USER=jetty
```

```
JETTY_LOGS=/var/log/jetty9
JETTY_HOME=/opt/jetty9
JETTY_ARGS=jetty.port=9090
```

Log Files Cron

Now, let's create the crontab configuration file that will rotate log jetty9 log files. Create a new file on /etc/cron.daily/jetty9 with content below:

```
  #!/bin/shNAME=jetty9
   DEFAULT=/etc/default/$NAME# The following variables can be overwritten in
$DEFAULT# Default for number of days to keep old log files in /var/log/jetty/
   LOGFILE_DAYS=14# End of variables that can be overwritten in $DEFAULT#
overwrite settings from default file
   if [ -f "$DEFAULT" ]; then
         . "$DEFAULT"
   fi
if [ -d /var/log/$NAME ]; then find /var/log/$NAME/ -name *.log -mtime
+$LOGFILE_DAYS -print0 | xargs --no-run-if-empty -0 rm -- fi
```

Create an Init Script to Manage Jetty9 Service

The Jetty package comes with an init script. It's located in the bin directory with the name jetty.sh. We will create the init script to manage Jetty 9 service by creating symlink to that file.

```
$ sudo ln -sf /opt/jetty9/bin/jetty.sh /etc/init.d/jetty9
```

Make Jetty 9 Start on Boot

We'll make sure Jetty 9 running automatically on boot using update-rc.d.

```
$ sudo update-rc.d jetty9 defaults
```

After configuring jetty, we can start jetty service by using

```
$sudo service jetty9 start
```

Then you can check jetty service by using

```
$sudo service jetty9 status
```

It should be running now on port 9090. A Jetty screen should be shown with the available html file to go into.

To store the html,javascript into Jetty server, you can store the html file into /opt/jetty/webapps directory. In the webapps directory, create sub-folder named root and

place html in the root folder. When the localhost:<port> is accessed, a screen will appear where you can select which html will be opened.

After storing inside restart the jetty server:

```
$sudo service jetty9 restart
OR
$sudo service jetty9 stop
$sudo service jetty9 start
```

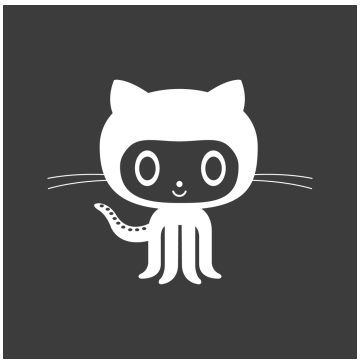
Additionally, you can check running port using

```
$ sudo netstat -lntp
```

Visit the link below for further details:

https://hostpresto.com/community/tutorials/how-to-install-jetty-9-on-ubuntu-14-04/

Github repositories



Github is where millions of developers use to build personal projects, support their businesses, and work together on open source (aka copyleft) technologies. Since many of the drivers for the robot has already been coded, we would be using them in our robot.

We will be using *git clone* to download the repository into the ROS workspace (catkin_ws), and it would look something like this:

~/catkin_ws/src $git clone (link with .git at the end)

[Remember to select the correct branch that corresponds to our ROS version - indigo/indigo-devel/master level]

Below are the following Github repository we will be making use of:

Joystick Drivers
Drivers to collect data from joystick controller and publish it
https://github.com/ros-drivers/joystick_drivers.git

Teleop Twist Joy (Replaces Husky teleop)
Programs to subscribe joystick position serial data and convert into x,y,z.
https://github.com/ros-teleop/teleop_twist_joy.git

SICK Tim571 - LIDAR that we are using
https://github.com/uos/sick_tim.git

Scan_tools(laser_scan_matcher) - Provides " Odometry" since we are not using an encoder
https://github.com/ccny-ros-pkg/scan_tools.git

Autonomous wheelchair

First, we have to test the hardware and the connections to the wheelchair

Items required

- **Motor Drive**r: Sabertooth 2x32
- **2 12V DC Motors**
- **Laboratory DC power supply**

Sabertooth

Sabertooth 2x32 Motor Driver

In this project, we decided to attempt to control the motor without the use of a encoder. The Kangaroo driver can be used together with the Sabertooth as an expansion board. When used together, the Sabertooth works with quadrature encoder or potentiometer feedback for speed or positioning control. Because it's self-tuning, plenty of time will not be wasted on getting the PID (Proportional-Integral-Derivative) coefficients dialed in.

Testing the sabertooth 2x32

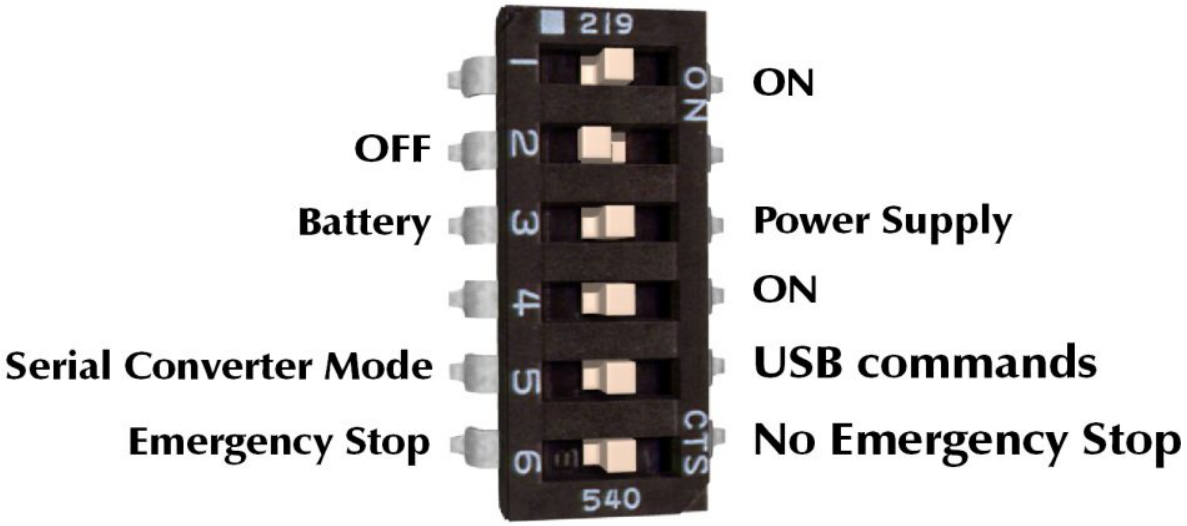
Since we do not have a motor controller to link the sabertooth to the computer, our computer will communicate with the sabertooth via usb mode(use a micro usb cable).

Use two 12V test motors to connect to the sabertooth and use a laboratory dc power supply to provide 12V to the main power input of the sabertooth.
Download Describe software on windows.
https://www.dimensionengineering.com/info/describe

Configure the sabertooth

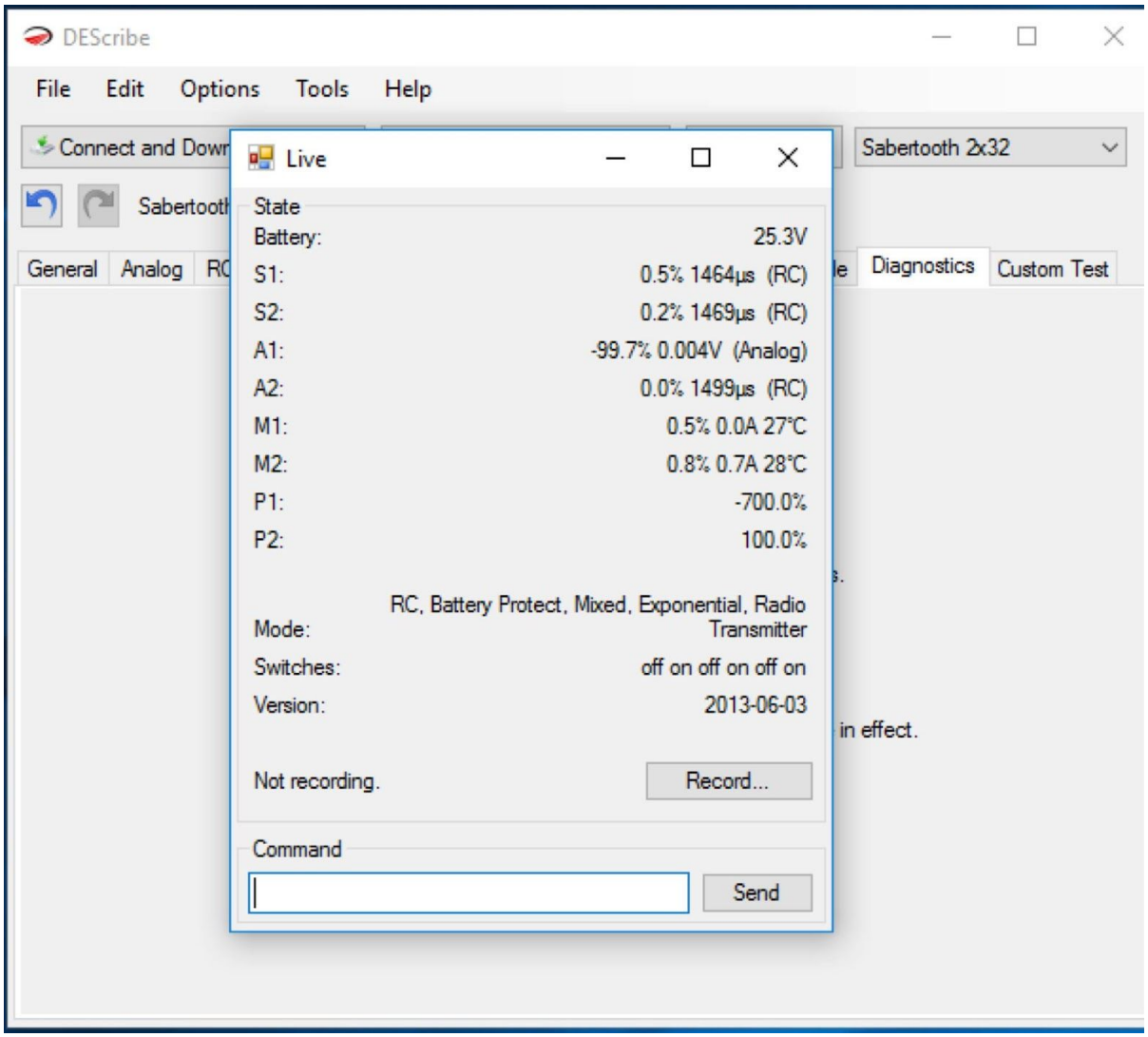
Use the default configurations on Describe software

DIP switch configuration



Read up on sabertooth configuration under USB Mode DIP Switch Settings
https://www.dimensionengineering.com/datasheets/Sabertooth2x32.pdf

In DEScribe, open diagnostics tab.
Click “show inputs and outputs”



Now enter “M1:2047”.(motor m1 will be activated. Test motor m2 also.)
Next, we have to wire the sabertooth to the wheelchair.

Basic commands

Outputs are controlled by sending the destination address, followed by a colon, followed by the power setting, followed by a newline. Most settings take commands from -2047 to 2047. Commands outside this range will be ignored. If you are using a terminal program to command the Sabertooth, remember that the command only takes effect when you press the Enter key.

Destination address	Description
M1	Motor 1
M2	Motor 2
MD	Drive channel. Both motors .Forward/Backwards in Mixed Mode
MT	Turn channel. Both motors. Right/Left in Mixed Mode
P1	Power output 1
P2	Power output 2
R1	Ramp rate motor 1
R2	Ramp rate motor 2
Q1	Auxiliary variable 1
Q2	Auxiliary variable 2

Example	Result
M1: 2047$_{\r\n}$	Motor 1 will go forward full speed.
M2: -1023$_{\r\n}$	Motor 2 will go backwards at half speed.
M1: 0$_{\r\n}$ M2: 0$_{\r\n}$	Motors 1 and 2 will stop.

MD: 0\r\n MT: -512\r\n	The robot will turn left at ¼ speed. Please note that to control using the mixed commands MD and MT, you must have sent both a turn and a drive command at some point. Once you have sent them both the first time, you only have to update the turn or drive commands as you desire, it is not necessary to always send them both.
P1: 2047\r\n	Power output 1 will be set to full power. Note that the power outputs are only controllable if they are set as additional output using the DEScribe software. Otherwise they will perform their normal voltage clamp or brake behavior and ignore serial input.
R2: 2047\r\n	This sets the speed ramping to the maximum amount. At this setting, it will take the Sabertooth approximately 8 seconds to go from a stop to full speed.
Q1: 1\r\n	By default the Q parameters control motor freewheeling. Setting a positive value will disable the motor channel. Freewheeling and shutdown are different. In a freewheel state the motor will act as though there is no power applied and the motor leads are floating. This makes the motor shaft easier to turn manually. In a shutdown state, the motor will act as if there is no power and the motor leads are connected together. This makes the motor shaft harder to turn manually. In user modes, the Q parameters are often used to change the operation of the program or activate special modes.

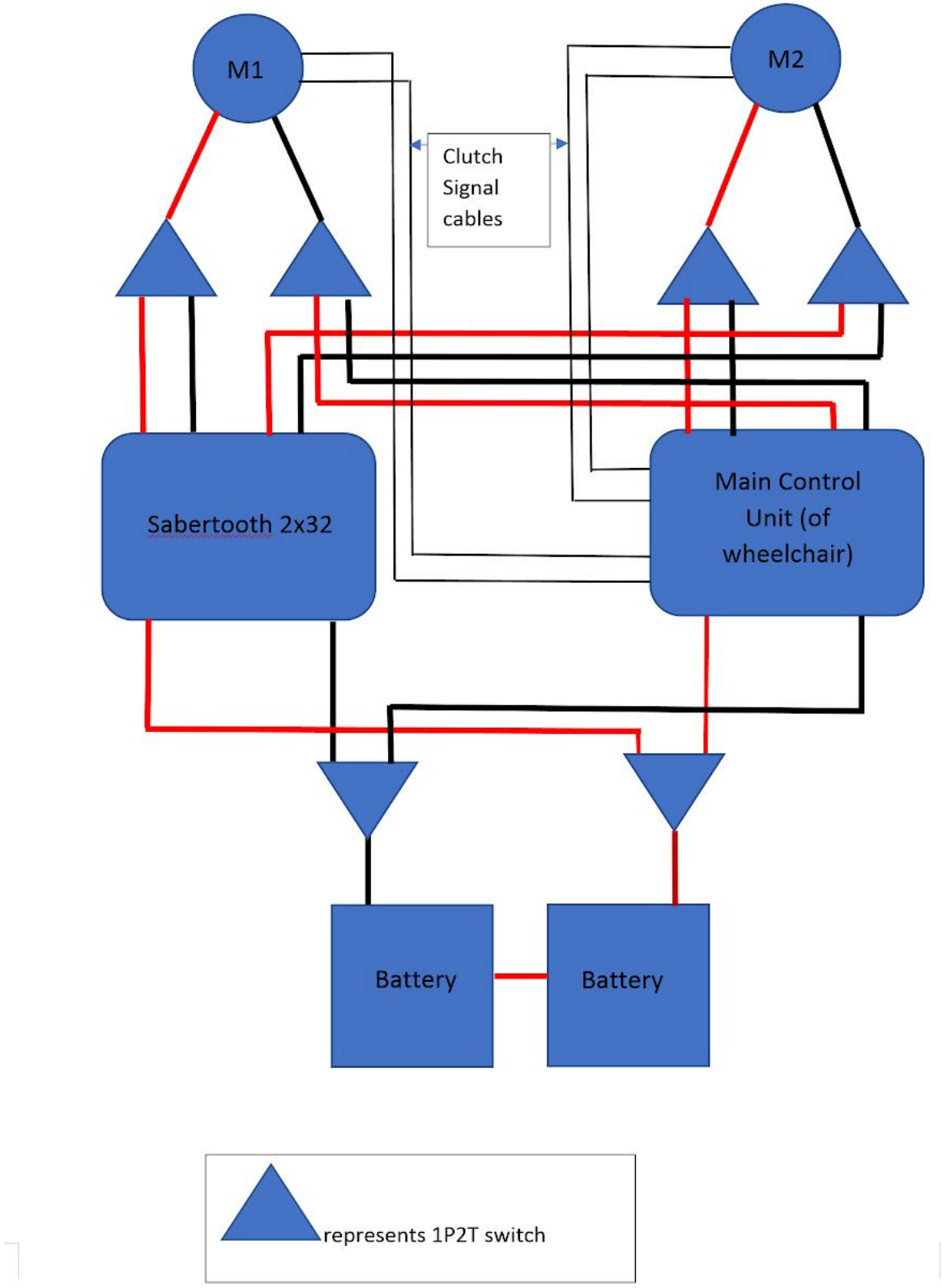
Gamepad controlled wheelchair

Items required

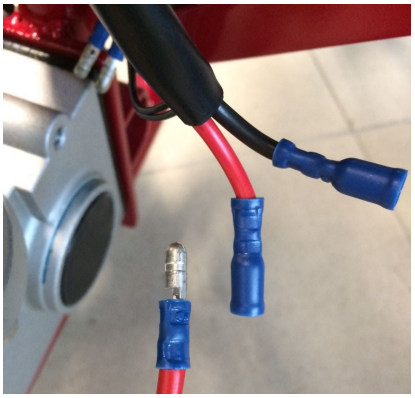
- **Motor Drive**r: Sabertooth 2x32
- **Joystick**: Logitech rumblepad700 Joystick Controller
- **Wire**: At least 15awg(smaller the value the thicker the wire)
- **Wire crimps**: According to wire thickness used
- **EW1200 Electrical Wheelchair**

Wheelchair we used

Wiring the wheelchair To be controlled with gamepad



Crimps can be used temporarily instead of 1P2T switches for the time-being to test the connections and allows us to easily manipulate the wiring without permanent soldering..



Controlling the wheelchair with Logitech gamepad

Packages required:

Joystick Drivers
Drivers to collect data from joystick controller and publish it
https://github.com/ros-drivers/joystick_drivers.git

Teleop Twist Joy (Replaces Husky teleop)
Programs to subscribe joystick position serial data and convert into x,y,z.
https://github.com/ros-teleop/teleop_twist_joy

Plow_motor_control_py:
https://github.com/USTseniordesignSNOWPLOW2016/ROSCODE
Extract the package plow_motor_control_py from the ROSCODE folder and into ~/catkin_ws/src/ as we will only be using that.

Don't forget to catkin_make and source after downloading.
Source command : $ source ./devel/setup.bash

Make a new launch file called **sabertooth.launch** and save it in one of your package folders in our case we created a new package called robot and saved it in there.
Paste this into your launch file:

```
<launch>
  <arg name="drive_speed" default="0.25" />
  <arg name="turn_speed" default="0.25" />
  <arg name="joy_dev" default="/dev/input/js6" />
  <arg name="cmd_topic" default="cmd_vel" />

  <node pkg="joy" type="joy_node" name="joy_node">
        <param name="dev" value="$(arg joy_dev)" />
        <param name="deadzone" value="0.3" />
  </node>

  <node pkg="teleop_twist_joy" type="teleop_node" name="teleop_node">
        <param name="turn_scale" value="$(arg turn_speed)" />
        <param name="drive_scale" value="$(arg drive_speed)" />
        <remap from="cmd_vel" to="$(arg cmd_topic)" />
  </node>



</launch>
```

Running the nodes

To run the launch file :

```
$ roslaunch robot sabertooth.launch
```

Chmod 777...

To run the python script for the sabertooth driver make sure you are in the right directory, in this case:

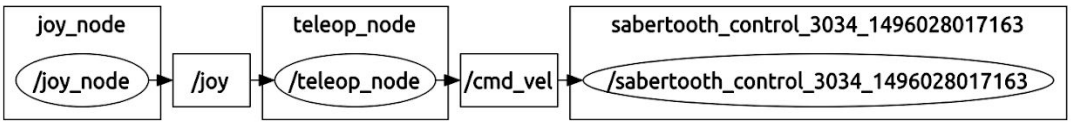
```
$ cd ~/catkin_ws/src/plow_motor_control_py/scripts/
```

```
$ sudo chmod +777 /dev/ttyACM0
```

And then run:

```
$ python sabertooth_drive.py
```

After running these the node graph should look like this:



To move the wheelchair around, hold down the A button and move the left joystick.

Mapping

LIDAR

LIDAR which means Light Detection and Ranging, is a remote sensing method which uses light in the form of a pulsed laser to measure distance to a certain target by illuminating it with the pulsed laser and measuring the reflected pulses using a sensor. These light pulses, in addition, to Hector_Mapping and GMapping generate precise, three-dimensional information about the shape of the surroundings. LIDAR systems enables us to examine both natural environment and man-made structures with accuracy, precision and flexibility.

Configuring the Lidar

To power the lidar, use a laboratory power supply to supply 24v to the SICK TIM LIDAR

Blue wire to negative and brown wire to positive

Connect Lidar to pc using the ethernet cable that comes with the lidar

Download Sopas ET for SICK TiM Lidar

https://www.sick.com/us/en/sopas-engineering-tool-v3/p/p367244

Run the program to find out what the ip-address is

The default ip-address should be 192.168.0.10

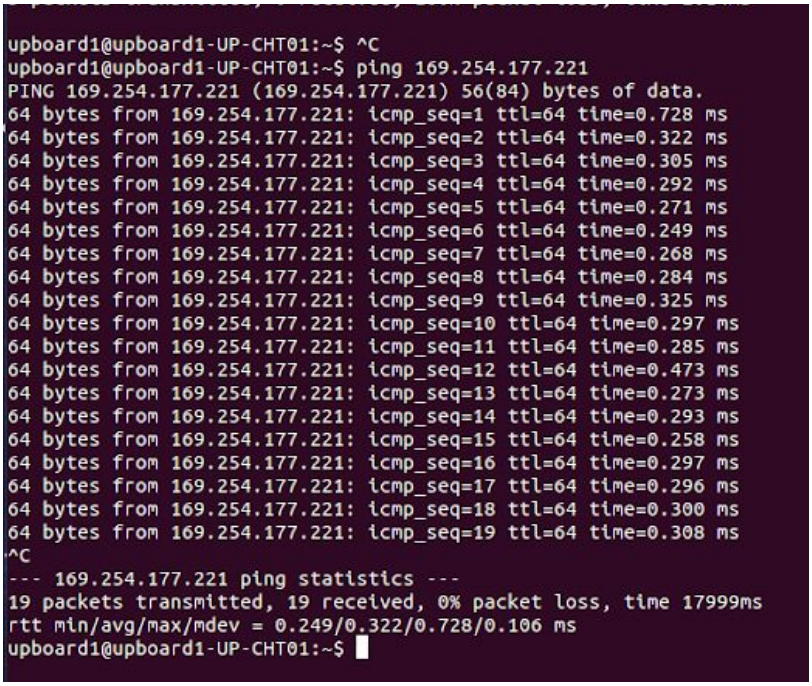
Testing the Lidar on ROS

Connect the lidar to the pc running
Go to **Edit connections** in ubuntu ---> **IPv4 Settings**
---> Change method to **Manual**
---> Set **Address** to **192.168.0.100** (IP address of LIDAR is 192.168.0.10 it cannot be the same so we use 100 instead)
---> Set **Netmask** to **255.255.255.0** ---> **Gateway** to **192.168.0.1**

Connect to wired connection 1 on network settings.
Enter into the terminal

$ ping (ip-address)

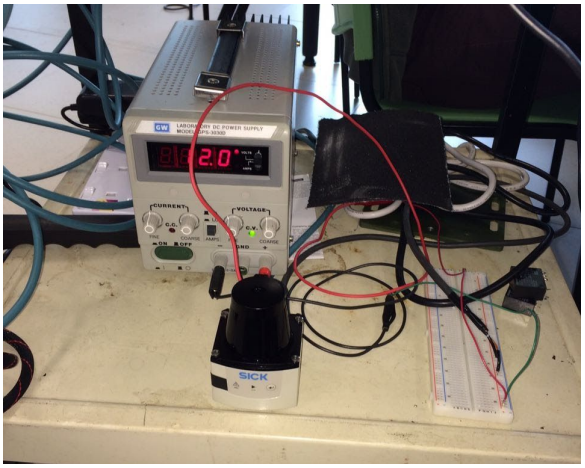
This should be the result (our LIDAR Ip address is 169.254.177.221) :



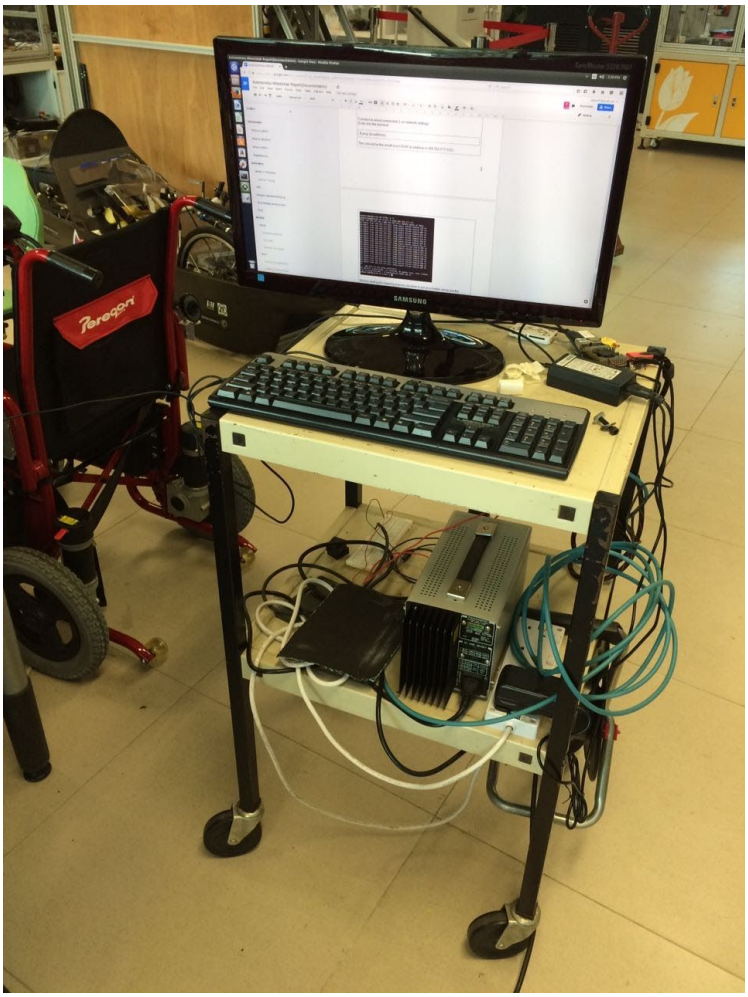
Before starting the mapping process, we have to set up a mobile unit to test the mapping process using the Lidar.

Things used

- **Trolley**(preferably 2 platforms)
- **Extension cable**(around 20m with 3 sockets)
- **Laboratory dc power supply**
- **Lidar**: SICK TIM 571 Lidar
- **PC**: Upboard cherry trail x5-Z8350 CPU 4gb ram
- **Monitor**
- **Keyboard**
- **Mouse**



The Lidar should be placed front and centered for easy configuration.



Now we can map the ASC by manually pushing the trolley around the ASC.

Gmapping

Create a launch file for gmapping named **gmapping.launch**
The available parameters are on the gmapping wiki page.

```
<launch>


<!-- GMAPPING -->
<node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
args="/scan">
   <param name="map_update_interval" value="5.0"/>
   <param name="maxUrange" value="20.0"/>
   <param name="delta" type="double" value="0.05"/>
   <param name="temporalUpdate" type="double" value="-1.0"/>
   <param name="particles" value="80"/>
   <param name="xmin" type="double" value="-5.0"/>
   <param name="xmax" type="double" value="5.0"/>
   <param name="ymin" type="double" value="-5.0"/>
   <param name="ymax" type="double" value="5.0"/>
   </node>-->

<!-- STATIC TRANSFORM PUBLISHER -->
<node pkg="tf" type="static_transform_publisher" name="Static_TF" args="0 0 0 0 0
0 1 base_link laser 100"/>

<!-- SCANMATCHER -->
            <node pkg="laser_scan_matcher" type="laser_scan_matcher_node"
name="scanmatcher">
   <param name="use_imu" type="bool" value="false"/>
   <param name="use_odom" type="bool" value="false"/>
   <param name="fixed_frame" type="string" value="odom"/>
   </node>

</launch>
```

The Static transform publisher transforms base_link to laser which is a required transform.

Mapping the ASC

Run the LIDAR launch file:

```
$ roslaunch sick_tim sick_tim571_2050101.launch
```

Run the Gmapping launch file:

```
$ roslaunch gmapping gmapping.launch
```

Run RVIZ

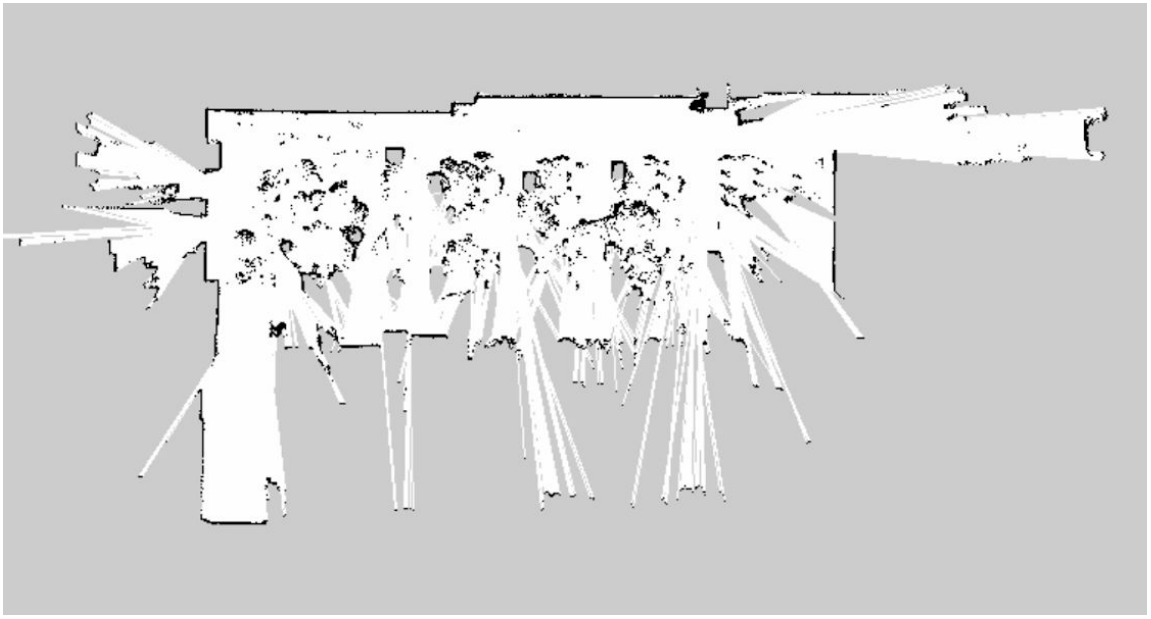
```
$ rviz
```

In RVIZ change fixed frame to /map for the map
If u change the fixed frame to /laser you can see the realtime laser scans.
click add ---> by topic and add /map
---> add tf

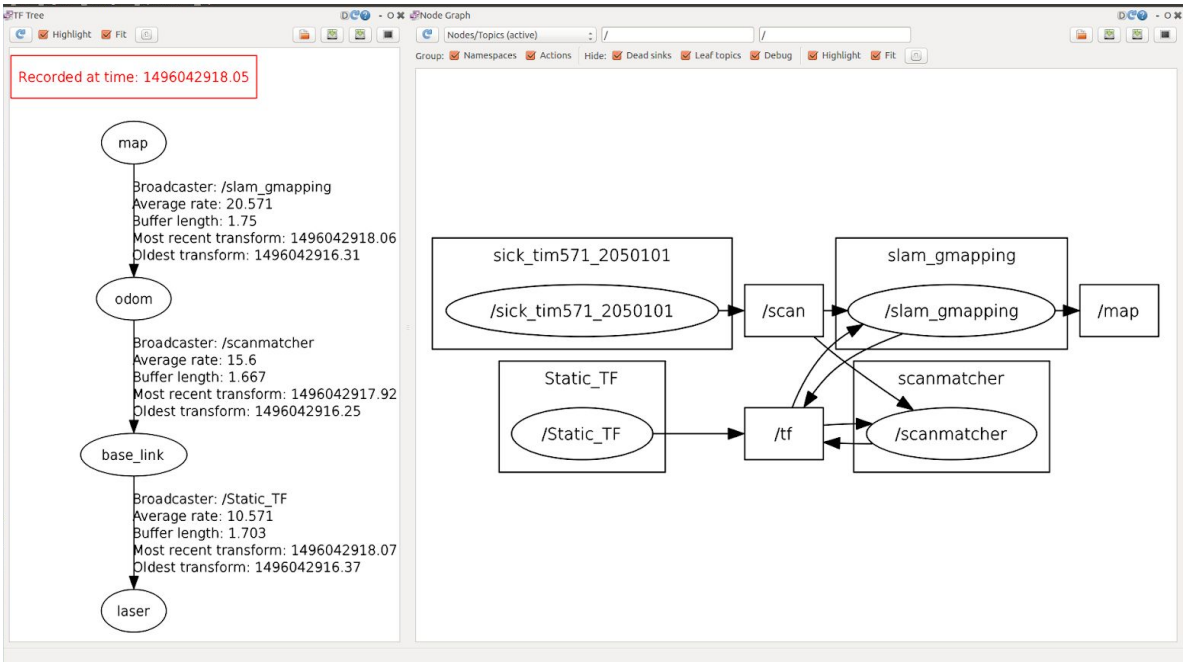
Move along a side of the room.
To save the map do:

```
$ rosrun map_server map_saver -f your_map_name(any name)
```

It should look something like this:



The TF tree and node graph should look like this when running the programs:



Use rosbag record to record your mapping around the room so it can be played back later on different parameter settings.

Create a directory for your bagfiles:

```
$ mkdir ~/bagfiles
$ cd ~/bagfiles
```

```
$ rosbag record -a
```

After recording you can filter out the /scan topic from the recording and playback the scan file in different settings.

```
$ rosbag filter my.bag(the name of your bagfile) scan.bag "topic == '/scan'"
```

For rosbag play to work properly u have to do:

```
$ rosparam set use_sim_time true
```

Rosbag play command

```
$ rosbag play (name of your file).bag
```

See the rosbag wiki for more specific commands
http://wiki.ros.org/rosbag/Tutorials/Recording%20and%20playing%20back%20data

Navigation

Localisation using amcl and a saved map

Make a new launch file called nav.launch.
Make sure to edit your map file name into the launch file(highlighted in red)

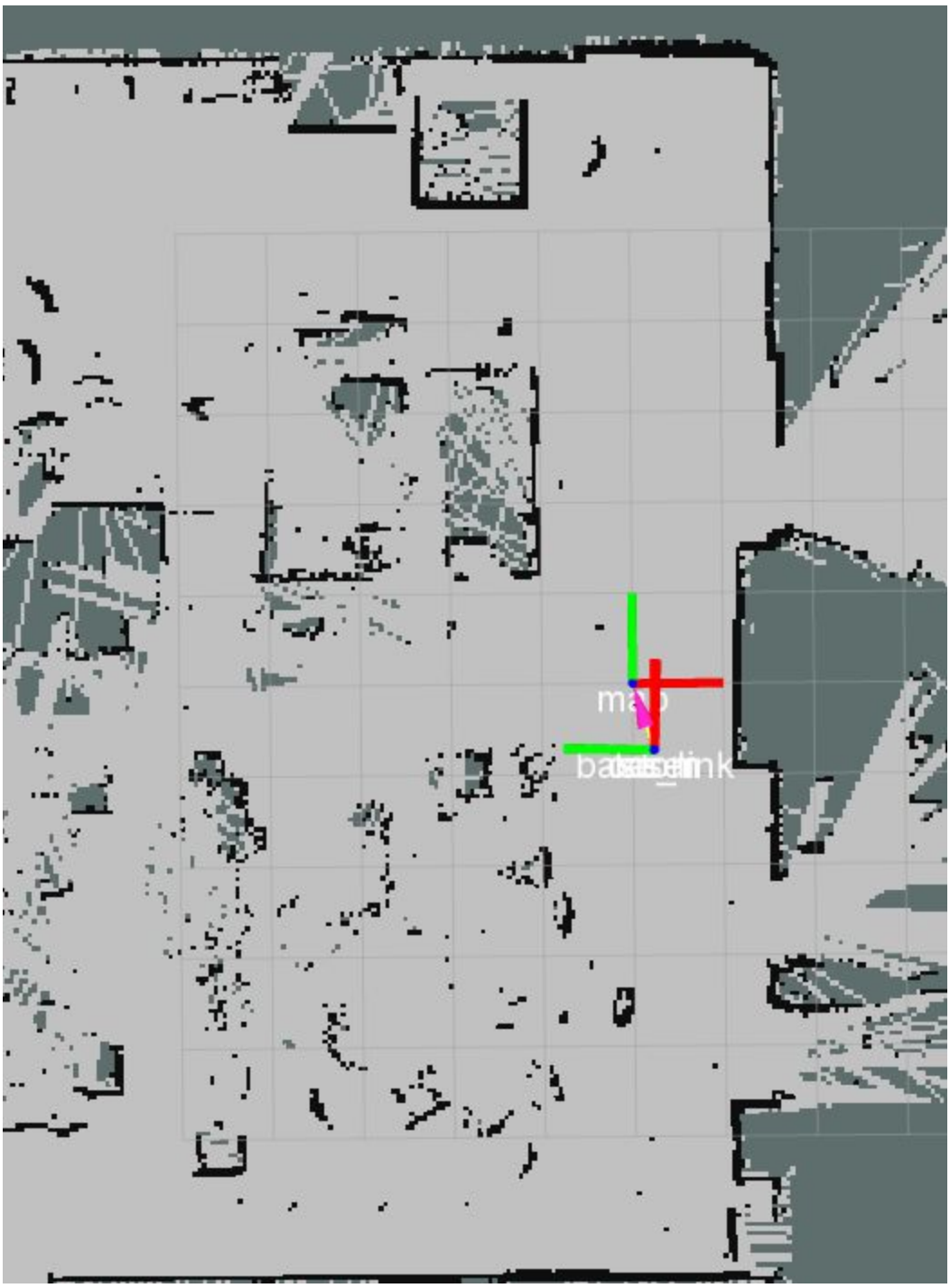
```
<arg name="map_file" default="$(find package_it_is_in)/your_map_name.yaml" />
    <node  pkg="map_server" type="map_server" name="map_server" args="$(arg map_file)" />



  <!-- AMCL -->
  <node pkg="amcl" type="amcl" name="amcl">
  <!--<param name="tf_broadcast" value="true" />
  <param name="base_frame_id" value="/base_link" />
  <param name="global_frame_id" value="map" />
  <param name="odom_frame_id" value="odom" />
  <param name="use_map_topic" value="true" />-->
  <remap from="scan" to="/scan"/>
  <param name="odom_model_type" value="omni" />
  <param name="transform_tolerance" value="0.2" />
  <param name="gui_publish_rate" value="-1.0"/>
  <param name="laser_max_beams" value="30"/>
  <param name="laser_max_range" value="29.5"/>
  <param name="min_particles" value="100"/>
  <param name="max_particles" value="5000"/>
  <param name="update_min_d" value="0.2"/>
  <param name="kld_err" value="0.01"/>
  <param name="kld_z" value="0.99"/>
  <param name="odom_alpha1" value="0.2"/>
  <param name="odom_alpha2" value="0.2"/>
  <param name="odom_alpha3" value="0.2"/>
  <param name="odom_alpha4" value="0.2"/>
  <param name="odom_alpha5" value="0.2"/>
  <param name="laser_z_hit" value="0.95"/>
  <param name="laser_z_short" value="0.1"/>
  <param name="laser_z_max" value="0.05"/>
  <param name="laser_z_rand" value="0.05"/>
  <param name="laser_sigma_hit" value="0.2"/>
  <param name="laser_lambda_short" value="0.1"/>
  <param name="laser_model_type" value="likelihood_field"/>
```

```
   <param name="laser_likelihood_max_dist" value="2.0"/>
   <param name="update_min_a" value="0.5"/>
   <param name="resample_interval" value="2"/>
   <param name="transform_tolerance" value="0.1"/>
   <param name="recovery_alpha_slow" value="0.0"/>
   <param name="recovery_alpha_fast" value="0.0"/>
   <param name="initial_pose_x" value="0.217"/>
   <param name="initial_pose_y" value="-0.729"/>
   <param name="initial_pose_a" value="1.559"/>
   </node>


   <!-- STATIC TRANSFORM PUBLISHER -->
    <node pkg="tf" type="static_transform_publisher" name="Static_TF" args="0 0 0
0 0 0 1 base_link laser 100"/>

<!-- SCANMATCHER -->
               <node    pkg="laser_scan_matcher"    type="laser_scan_matcher_node"
name="scanmatcher">
   <param name="use_imu" type="bool" value="false"/>
   <param name="use_odom" type="bool" value="false"/>
   <param name="fixed_frame" type="string" value="odom"/>
   </node>

</launch>
```

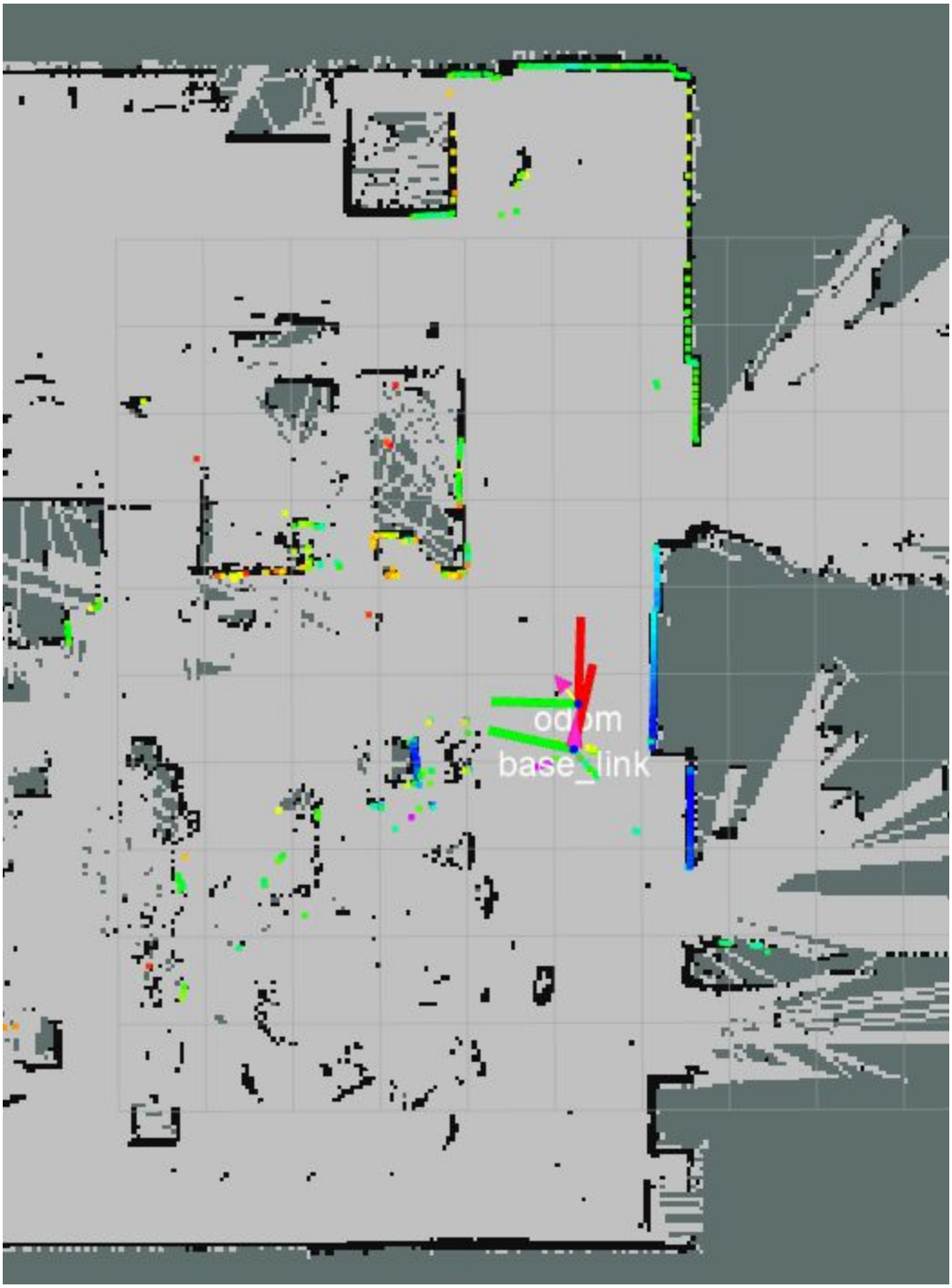
Run the launch file and open rviz to view your saved map.

Now we use the 2d pose estimate to position the robot at the correct point in the map. Mark out where you started mapping your map and bring your robot to that position before you run rviz.

After position it properly it should look something like this :



With the laser it should look like this :


odom
base_link

Navigation using RVIZ

For navigation we require the Navigation stack :

```
$ git clone https://github.com/ros-planning/navigation.git
```

Now create a new package called robot_config

```
$ catkin_create_pkg robot_config
```

Now go into robot_config and add a new folder called config

We need four configuration files for the move_base package which is what we are using for navigation.

Create a file named costmap_common_params.yaml:

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[0.0, 0.0], [0.75, 0.0], [0.75, 0.75],[0.0, 0.75]]
#robot_radius: ir_of_robot
inflation_radius: 0.6

observation_sources: laser_scan_sensor

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

***footprint** is what defines the size of your robot.
***Edit:** Footprint was edited to roughly fit the size of our wheelchair.

global_costmap_params.yaml :

```
global_costmap:
  global_frame: /map
  robot_base_frame: base_link
  update_frequency: 5.0
  static_map: true
```

Local_costmap_params.yaml :

```
local_costmap:
  global_frame: /map
  robot_base_frame: base_link
  update_frequency: 3.0
  publish_frequency: 1.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.25
```

Base_local_planner_params.yaml :

```
TrajectoryPlannerROS:
  max_vel_x: 0.45
  min_vel_x: 0.2
  max_vel_theta: 1.0
  min_vel_theta: 0.3
  min_in_place_vel_theta: 0.3

  acc_lim_theta: 0.15
  acc_lim_x: 2.5
  acc_lim_y: 2.5

  holonomic_robot: true
  y_vels: [-0.3, -0.1, 0.1, 0.3]

  yaw_goal_tolerance: 3.14
  xy_goal_tolerance: 0.3
  latch_xy_goal_tolerance: false

  escape_vel: -0.1
```

***Edit:** We changed the min/max vels to a more appropriate value.
***Edit2:** We added in goal tolerances so that the wheelchair does not go berserk when it can't find the exact location of the goal. This allows the wheelchair to recognise the goal easier.

Save all these files into **config**.

Giving priority

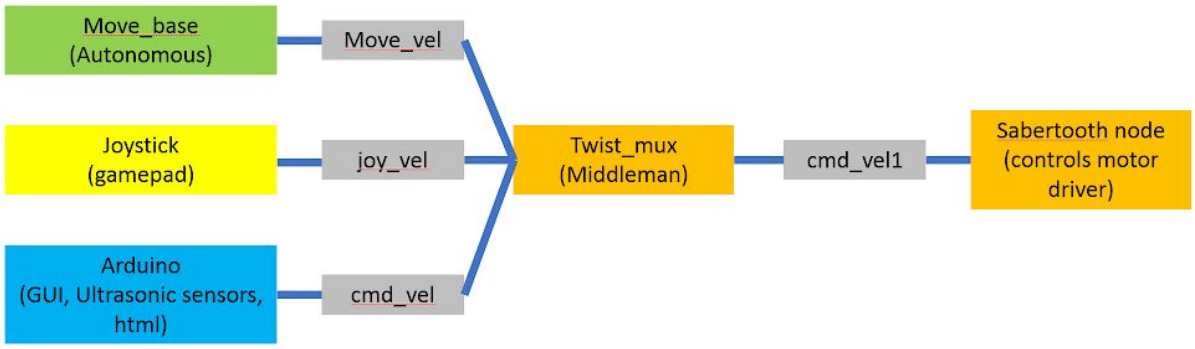
We used twist_mux package to set the priorities for the different topics publishing cmd_vel.

Priorities (highest to lowest) : Joystick
Emergency Stop
Move_base (Navigation Stack)

Package :

http://wiki.ros.org/twist_mux

$ git clone https://github.com/ros-teleop/twist_mux.git



Go into the config folder and open the **twist_mux_topics.yaml** file to edit it.
It should look like this :

```
# Input topics handled/muxed.
# For each topic:
# - name         : name identifier to select the topic
# - topic    : input topic of geometry_msgs::Twist type
# - timeout  : timeout in seconds to start discarding old messages, and use 0.0 speed instead
# - priority: priority in the range [0, 255]; the higher the more priority over other topics

topics:
-
  name : move_base
  topic   : move_vel
  timeout : 0.5
  priority: 10
-
```

```
  name : joystick
  topic   : joy_vel
  timeout : 0.5
  priority: 100
-
  name : arduino #and html
  topic   : cmd_vel
  timeout : 0.5
  priority: 50
-
  name : tablet
  topic   : tab_vel
  timeout : 0.5
  priority: 100
```

The bigger the number, the higher the priority.
Change the topic to whichever topic that particular node is publishing.
You can remap the topic name to a different name by adding this to your launch file under the node u need to remap.
For example (Joy) :

```
<node pkg="joy" type="joy_node" name="joy_node">
        <param name="dev" value="$(arg joy_dev)" />
        <param name="deadzone" value="0.3" />
  </node>

  <node pkg="teleop_twist_joy" type="teleop_node" name="teleop_node">
        <param name="turn_scale" value="$(arg turn_speed)" />
        <param name="drive_scale" value="$(arg drive_speed)" />
        <remap from="cmd_vel" to="joy_vel" />
  </node>
```

See the line in red

We modified our nav.launch file to include **sick_time, move_base, twist_mux** and **rviz**:

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

<launch>

  <arg name="drive_speed" default="0.25" />
  <arg name="turn_speed" default="0.25" />
  <arg name="joy_dev" default="/dev/input/js0" />
  <arg name="cmd_topic" default="cmd_vel" />
                        <arg            name="config_locks"                    default="$(find
twist_mux)/config/twist_mux_locks.yaml"/>
                              <arg              name="config_topics"             default="$(find
twist_mux)/config/twist_mux_topics.yaml"/>




  <!-- LIDAR -->
   <node name="sick_tim571_2050101" pkg="sick_tim" type="sick_tim551_2050001"
respawn="false" output="screen">
            <param name="range_max" type="double" value="25.0" />
          <param name="hostname" type="string" value="169.254.177.221" />
          <param name="port" type="string" value="2112" />
          <param name="timelimit" type="int" value="25" />
     <param name="min_ang" type="double" value="-1.55" />
     <param name="max_ang" type="double" value="1.55" />
  </node>




  <!-- JOYSTICK CONTROL -->
 <node pkg="joy" type="joy_node" name="joy_node">
          <param name="dev" value="$(arg joy_dev)" />
          <param name="deadzone" value="0.3" />
```

```
  </node>

  <node pkg="teleop_twist_joy" type="teleop_node" name="teleop_node">
        <param name="turn_scale" value="$(arg turn_speed)" />
        <param name="drive_scale" value="$(arg drive_speed)" />
        <remap from="cmd_vel" to="joy_vel" />
  </node>


  <node pkg="rviz" type="rviz" name="rviz"/>


        <!-- MAP SERVER -->
        <arg name="map_file" default="$(find robot)/mus1.yaml" />
        <node pkg="map_server" type="map_server" name="map_server"
args="$(arg map_file)" />


<node pkg="twist_mux" type="twist_mux" name="twist_mux" output="screen">

  <rosparam file="$(arg config_locks)"  command="load"/>
  <rosparam file="$(arg config_topics)" command="load"/>
</node>


        <!--(for mapping procedure)-->
        <!-- AMCL -->
        <node pkg="amcl" type="amcl" name="amcl">


        <!--<param name="tf_broadcast" value="true" />
        <param name="base_frame_id" value="/base_frame" />
        <param name="global_frame_id" value="/map" />
        <param name="odom_frame_id" value="/base_link" />-->


        <param name="use_map_topic" value="true" />
```

```
<remap from="scan" to="/scan" />
<param name="odom_model_type" value="omni" />
<param name="transform_tolerance" value="0.4" />
<param name="gui_publish_rate" value="-1.0"/>
<param name="laser_max_beams" value="30"/>
<param name="laser_max_range" value="29.5"/>
<param name="min_particles" value="100"/>
<param name="max_particles" value="5000"/>
<param name="update_min_d" value="0.2"/>
<param name="kld_err" value="0.01"/>
<param name="kld_z" value="0.99"/>
<param name="odom_alpha1" value="0.2"/>
<param name="odom_alpha2" value="0.2"/>
<param name="odom_alpha3" value="0.2"/>
<param name="odom_alpha4" value="0.2"/>
<param name="odom_alpha5" value="0.2"/>
<param name="laser_z_hit" value="0.95"/>
<param name="laser_z_short" value="0.1"/>
<param name="laser_z_max" value="0.05"/>
<param name="laser_z_rand" value="0.05"/>
<param name="laser_sigma_hit" value="0.2"/>
<param name="laser_lambda_short" value="0.1"/>
<param name="laser_model_type" value="likelihood_field"/>
<param name="laser_likelihood_max_dist" value="2.0"/>
<param name="update_min_a" value="0.5"/>
<param name="resample_interval" value="2"/>
<param name="transform_tolerance" value="0.1"/>
<param name="recovery_alpha_slow" value="0.0"/><!--modified here-->
<param name="recovery_alpha_fast" value="0.0"/>
</node>




<!-- STATIC TRANSFORM PUBLISHER -->
```

```
        <node pkg="tf" type="static_transform_publisher" name="Static_TF" args="0
0 0 3.142 0 0 base_link laser 100"/>




        <!-- SCANMATCHER -->
        <node pkg="laser_scan_matcher" type="laser_scan_matcher_node"
name="scanmatcher">
        <param name="use_imu" type="bool" value="true"/><!--change to help scan
matcher-->9
        <param name="use_odom" type="bool" value="false"/>
        <param name="fixed_frame" type="string" value="odom"/>
        </node>




        <!-- MOVE BASE -->
<node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
        <rosparam file="$(find robot_config)/config/costmap_common_params.yaml"
command="load" ns="global_costmap" />
        <rosparam file="$(find robot_config)/config/costmap_common_params.yaml"
command="load" ns="local_costmap" />
        <rosparam file="$(find robot_config)/config/local_costmap_params.yaml"
command="load" />
        <rosparam file="$(find robot_config)/config/global_costmap_params.yaml"
command="load" />
        <rosparam                                                    file="$(find
robot_config)/config/base_local_planner_params.yaml" command="load" />
        <param name="conservative_reset_dist" type="double" value="1.0" />
    <param name="recovery_behavior_enabled" type="bool" value="false"/>
    <param name="clearing_rotation_allowed" type="bool" value="false"/>
    <remap from="cmd_vel" to="move_vel" />
```

```
  </node>



        <!--GMAPPING
        <node pkg="gmapping" type="slam_gmapping" name="slam_gmapping"
args="/scan">
        <param name="map_update_interval" value="5.0"/>
        <param name="maxUrange" value="20.0"/>
        <param name="delta" type="double"       value="0.05"/>
        <param name="temporalUpdate" type="double" value="-1.0"/>
        <param name="particles" value="30"/>
        <param name="xmin" type="double" value="-5.0"/>
        <param name="xmax" type="double" value="5.0"/>
        <param name="ymin" type="double" value="-5.0"/>
        <param name="ymax" type="double" value="5.0"/>
    <remap from="/map" to="/stuff" />

        </node> -->




</launch>
```

Now we can run a test.
Launch nav.launch and everything should be working.
Now we can try to navigate my publishing a point from rviz.
After aligning the laser with the map click on **2d Nav Goal** and then click on a point on the map.(the direction the arrow points is the direction the robot will end up facing)
The wheelchair should move to that point somewhat smoothly and stop.
Now we move on to navigation with waypoints.

Navigation with waypoints

We will be using this package :

```
$ git clone https://github.com/jihoonl/waypoint.git
```

This is the package we used for creating waypoints on the saved map to navigate to.

The two main packages we are using from the waypoint package are :
waypoint_generator & waypoint_touring (remove these two folders from the main one and put into src)

Creating waypoints

- Launch everything used for navigation
- Run the waypoint_generator python script

```
$ cd ~/catkin_ws/src/waypoint_generator/scripts
$ python generator.py
```

Now in RVIZ we can use the 'publish point' feature to create our waypoints on the map. Click on publish point and then click on where you want the point to be on the map.

After creating all the points you need, end the waypoint_generator script and it will save the waypoints into a .txt file. This file can be found in the same directory as generator.py inside the scripts folder.

We now have to change this file to a .yaml file simply by renaming it (e.g. waypoints.yaml)
Move this file over to the waypoint_touring package, into the 'data' folder. Now go to the 'launch' folder in waypoint_touring and edit the run.launch file to launch the new .yaml file you just made. (The default should be example.yaml)

We are now ready to test the waypoint navigation. Run all the navigation processes, then in a new terminal launch the launch file.

```
$ roslaunch waypoint_touring run.launch
```

Upon launching this file the wheelchair should automatically start moving to the set waypoints.

Using other interfaces

We moved on to making a node to interact with external interfaces
and send commands to the wheelchair to control it.(in our case arduino and a html site)

We created a python script to listen to data being published by
arduino / html. (These two will be addressed in a different section below)

The data is being published on the topic /chatter
This is our code for **waypointlaunch.py** :

```
#!/usr/bin/env python
import rospy
import subprocess
import signal
import os
import time
import sys
from geometry_msgs.msg import Twist
from std_msgs.msg import String



def callback(data):

        global msg

        if data.data == "1":
        p = subprocess.Popen(["roslaunch", "waypoint_touring", "base.launch"])

        elif data.data == "2":
        p = subprocess.Popen(["roslaunch", "waypoint_touring", "cupboard.launch"])

        elif data.data == "3":
        p = subprocess.Popen(["roslaunch", "waypoint_touring", "door.launch"])

        elif data.data == "4":
        p = subprocess.Popen(["roslaunch", "waypoint_touring", "fish.launch"])

        elif data.data == "5":
        p = subprocess.Popen(["roslaunch", "waypoint_touring", "frontdoor.launch"])

        elif data.data == "6":
        p = subprocess.Popen(["roslaunch", "waypoint_touring", "workstation.launch"])
```

```
        elif data.data == "7":  #E-stop
        p = subprocess.Popen(["python", "cmdpub.py"],  preexec_fn=os.setsid)

        elif data.data == "8":  #cancel destination
        p = subprocess.Popen(["rostopic", "pub", "/move_base/cancel",
"actionlib_msgs/GoalID", "--", "{}"])

        elif data.data == "9":  #disengage E-stop
        p = subprocess.Popen(["pkill", "-9", "-f", "cmdpub.py"])

        elif data.data == "11":  #stop if path is blocked
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = 0
        pub.publish(msg)


        elif data.data == "12":  #Slow down
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0.25
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = 0
        pub.publish(msg)


        elif data.data == "13":  #Gradual Right
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0.4
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = -0.03
        pub.publish(msg)


        elif data.data == "14":  #Right
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = -0.2
```

```
        pub.publish(msg)


        elif data.data == "15":  #Gradual Left
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0.4
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = 0.03
        pub.publish(msg)


        elif data.data == "16":  #Left
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = -0.2
        pub.publish(msg)


        elif data.data == "17":  #Go straight
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0.3
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = 0


def listener():

        # In ROS, nodes are uniquely named. If two nodes with the same
        # node are launched, the previous one is kicked off. The
        # anonymous=True flag means that rospy will choose a unique
        # name for our 'listener' node so that multiple listeners can
        # run simultaneously.
        rospy.init_node('listener', anonymous=True)

        rospy.Subscriber("chatter", String, callback)

        # spin() simply keeps python from exiting until this node is stopped
        rospy.spin()

if __name__ == '__main__':
```

```
    listener()
```

Code explained

```
def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # node are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.

    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()
```

This creates a listener to subscribe to the topic /chatter

```
def callback(data):

    global msg

    if data.data == "1":
    p = subprocess.Popen(["roslaunch", "waypoint_touring", "base.launch"])

    elif data.data == "2":
    p = subprocess.Popen(["roslaunch", "waypoint_touring", "cupboard.launch"])

    elif data.data == "3":
    p = subprocess.Popen(["roslaunch", "waypoint_touring", "door.launch"])

    elif data.data == "4":
    p = subprocess.Popen(["roslaunch", "waypoint_touring", "fish.launch"])

    elif data.data == "5":
    p = subprocess.Popen(["roslaunch", "waypoint_touring", "frontdoor.launch"])

    elif data.data == "6":
    p = subprocess.Popen(["roslaunch", "waypoint_touring", "workstation.launch"])
```

All of these **If Else** statements are under the callback(data) function which is where we decide what to do with the data received by our listener.
We make use of the python subprocess module to launch our pre-created waypoint launch files (Refer to navigation with waypoints).

```
p = subprocess.Popen(["roslaunch", "waypoint_touring", "your_launch_file.launch"])
```

*Name of your launch file
This line of code basically executes this as it would when you type that into the terminal (it's just the command to launch a launch file)
This launches our launch file as a background process (so that the node can continue receiving other messages). As stated before the robot will automatically start moving once the waypoint file is launched

```
elif data.data == "7":  #E-stop
        p = subprocess.Popen(["python", "cmdpub.py"],  preexec_fn=os.setsid)
```

This opens another node we created called **cmdpub.py**(code of this node is below) which publishes cmd_vel = 0 in a loop until it is killed.

```
elif data.data == "9":  #disengage E-stop
        p = subprocess.Popen(["pkill", "-9", "-f", "cmdpub.py"])
```

We have a separate button to disengage the E-Stop.
This will send the command to kill the cmdpub.py node which will disable the E-Stop.

*note: The E-Stop has higher priority than move_base so the wheelchair will not move until it is disabled. But our joystick has the highest priority so we can move the wheelchair around using the joystick even with the E-stop engaged.

```
elif data.data == "8":  #cancel destination
        p = subprocess.Popen(["rostopic", "pub", "/move_base/cancel", "actionlib_msgs/GoalID", "--", "{}"])
```

This will send a command to cancel goals so the previously selected destination will be canceled. Cancelling the goals will end the waypoint launch file which was running in the background as well.

```
elif data.data == "11":  #stop if path is blocked
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = 0
        pub.publish(msg)
```

*Note : msg.linear.x controls the velocity of forward movement
msg.angular.z controls the angular movement

This part of the code is for the ultrasonic sensors. When they detect an obstacle they send different messages(numbers) to /chatter and depending on the number they do different things. We just created a simple publisher to publish cmd_vel. In this case, when the path is blocked it will publish 0 to cmd_vel to make it stop.

Cmdpub.py code : This is just a simple code to send cmd_vel = 0 in a loop

```
#!/usr/bin/env python

import serial
import rospy
from std_msgs.msg import String
from geometry_msgs.msg import Twist
import time


def talker():
   rospy.init_node('listener', anonymous=True)
   while not rospy.is_shutdown():
        global msg
        msg = Twist()
        pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
        msg.linear.x = 0
        msg.linear.y = 0
        msg.linear.z = 0
        msg.angular.z = 0
        pub.publish(msg)
        time.sleep(0.5)
```

```
if __name__ == '__main__':

    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

HTML

To use html with ros, we will need the following package :

```
$ git clone https://github.com/RobotWebTools/rosbridge_suite.git
```

Wiki page : http://wiki.ros.org/rosbridge_suite

We made a simple html interface that publishes data on the topic /chatter which is then received by the node we created(refer to previous section).
The interface looks like:

The code looks like this :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
```

```
<script type="text/javascript" src="http://cdn.robotwebtools.org/EventEmitter2/current/eventemitter2.min.js"></script>
<script type="text/javascript" src="http://cdn.robotwebtools.org/roslibjs/current/roslib.min.js"></script>

<script type="text/javascript" type="text/javascript">


/*
** Setup all GUI elements when the page is loaded.
 */
//Setting up Global Variables
// This function connects to the rosbridge server running on the local computer on port 9090
var rbServer = new ROSLIB.Ros({
        url : 'ws://172.16.22.60:9090'
  // url : 'ws://localhost:9090'
 });

 // This function is called upon the rosbridge connection event
 rbServer.on('connection', function() {
        // Write appropriate message to #feedback div when successfully connected to rosbridge
        var fbDiv = document.getElementById('feedback');
        fbDiv.innerHTML += "<p>Connected to websocket server.</p>";
 });

// This function is called when there is an error attempting to connect to rosbridge
rbServer.on('error', function(error) {
        // Write appropriate message to #feedback div upon error when attempting to connect to rosbridge
        var fbDiv = document.getElementById('feedback');
        fbDiv.innerHTML += "<p>Error connecting to websocket server.</p>";
});

// This function is called when the connection to rosbridge is closed
rbServer.on('close', function() {
        // Write appropriate message to #feedback div upon closing connection to rosbridge
        var fbDiv = document.getElementById('feedback');
        fbDiv.innerHTML += "<p>Connection to websocket server closed.</p>";
 });

var htmlTopic = new ROSLIB.Topic({
        ros : rbServer,
        name : 'chatter',
```

```
        messageType : 'std_msgs/String'
});

/*var listenerTopic = new ROSLIB.Topic({
        ros : rbServer,
        name : 'cmd_vel',
        messageType : 'geometry_msgs/Twist'
});


listenerTopic.subscribe(function(message) {
        console.log(message.g);
        var fbDiv = document.getElementById('feedback');
        fbDiv.innerHTML += 'Moving';
  });
*/
// These lines create a message that conforms to the structure of the Twist defined in our ROS installation
// It initalizes all properties to zero. They will be set to appropriate values before we publish this message.
var string = new ROSLIB.Message({
        data: '1'
});

var string2 = new ROSLIB.Message({
        data: '3'
});

var string3 = new ROSLIB.Message({

        data: '8'
});


var string4 = new ROSLIB.Message({
        data: '7'
});

var string5 = new ROSLIB.Message({
        data: '9'
});

function pubMessage() {


        htmlTopic.publish(string);
}
```

```
function pubMessage2() {


        htmlTopic.publish(string2);
}

function canceldest(i) {
  htmlTopic.publish(string3);
  if (i < 0) return;
  setTimeout(function () { canceldest(--i) }, 750);

}

function eStop() {


        htmlTopic.publish(string4);
}

function cont() {


        htmlTopic.publish(string5);
}


</script>
</head>



<body>
<form name="ctrlPanel">
<hl> Autonomous Wheelchair </hl> <br>

<button id="sendMsg" type="button" onclick="pubMessage()">Base</button>
<button id="sendMsg" type="button" onclick="pubMessage2()">Door</button>
<button id="sendMsg" type="button" onclick="canceldest(6)">Cancel
Destination</button>
<button id="sendMsg" type="button" onclick="eStop()">E-Stop</button>
<button id="sendMsg" type="button" onclick="cont()">Continue</button>

</form>

<div id="feedback"></div>
```

```
</body>
</html>
```

The buttons will publish different data when pressed.

```
var rbServer = new ROSLIB.Ros({
        url : 'ws://172.16.22.60:9090'
  // url : 'ws://localhost:9090'
 });
```

*Ip address

This part of the code is where you edit the ip address to match the target pc's ip address.

Using ***url : 'ws://localhost:9090'*** (which is currently commented out) *allows you to connect to the same pc you are using to run both rosbridge and the html script.*

Save this file as a .html file.

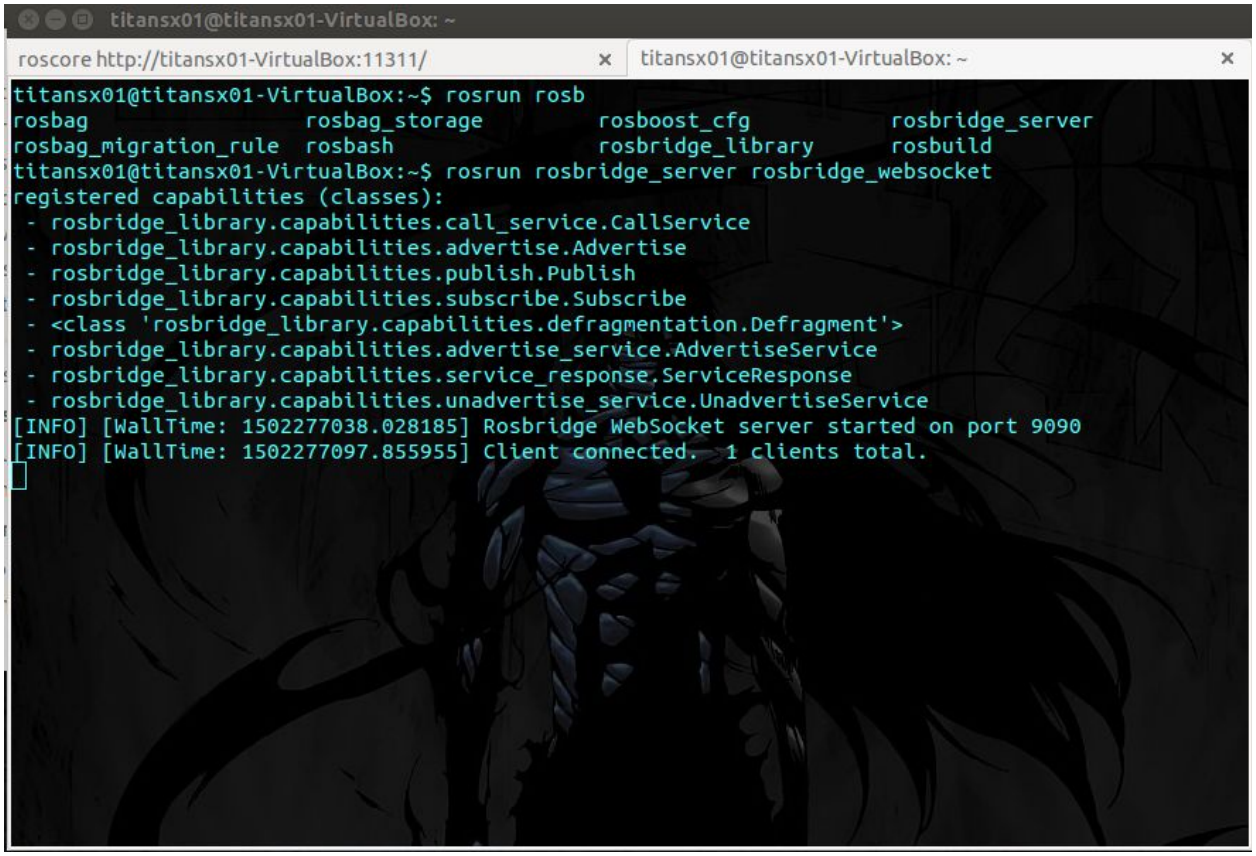
To find out the ip of your wirelessly connected pc, do:

```
$ ifconfig
```

To connect html to ros, run rosbridge_server :

```
$ rosrun rosbridge_server rosbridge_websocket
```

And then double click on the html file you just made and it should say that it has been connected on the terminal like this :



Now to check if the buttons work, do:

$ rostopic echo chatter

Now when u press one of the buttons it should echo the data published on the terminal.

Android UI

This UI is made on android studios. The UI will be linked to the web server from ROS and JSON files will be sent from androids studios.
The JSON files posted to web server are commands used for the wheelchair
Using voice recognition in the tablet to receive key words which are used for the commands of the wheelchair
Keywords: stop, cancel, continue, door,etc.

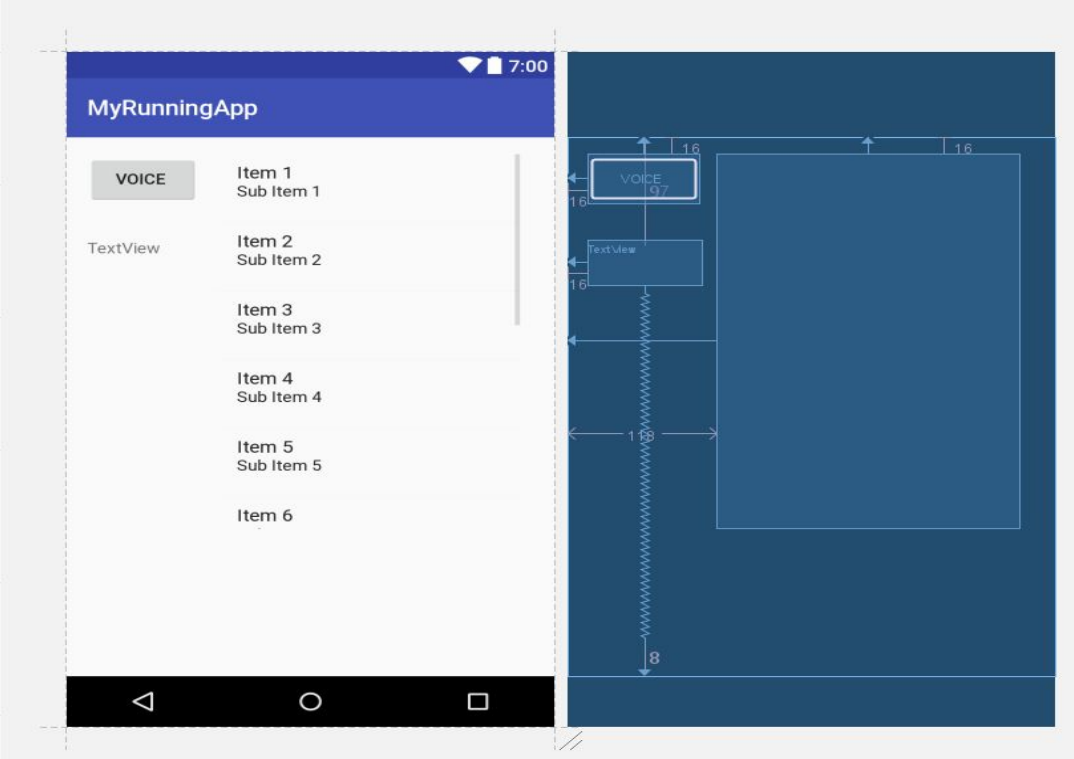
For this application, we have to use a button, textview and listview.
We add them by dragging these tools from the palette into the blueprint, which is on an xml file of your main activity in the design tab which is located near the bottom of the screen where you can select text or design tab in the xml file.
When these tools are dragged into the blueprint they need to be set up in the java file which is automatically created.

For our application, we start by coding out the android to have voice recognition.
Firstly, we use a button to activate voice recognition and will be set up as speakButton, a ListView is also used, which will display a list of possible matches for the speech that the android has recorded.
Once button and listview are added and declared, we need to declare an int variable for the voice recognition request using codes from line 31-33
Next, create an OnCreate Method and setup the button and listener and add a method voiceinputbuttons which will be set up later, referring to codes from line 35-41.
Remember to setContentView for the xml you are using.
Outside the OnCreate method, create the voiceinputbuttons method which will look similar to the codes in line 51-54. The id of your button and list in the code is referred from the id of your button and list in your xml file.

Now the voice recognition activity needs to be coded using codes from lines 56-63.
Once the voice recognition activity is coded, you need to implement the OnClickListener to your java. Code the OnClick method as done in line 65-68 and remember to set the on Click property of your button to the OnClick method in your java.
Once this is done, another method will be made. This method is made to produce a list of what the user could have possibly said. Using an if statement for a keyword you want to use, in this case, our wheelchair commands such as “stop”, “cancel” or “continue” are used in a multiple else if statements. “matches”, being the result of voice input, contains keyword coded, and activity can be prompt using a method, in this case we will send the command over to ROS in JSON format which the wheelchair will then receive and do

the following command. In addition, the textview will change to the keyword said, which will show the user which command has been heard and sent.
NOTE: The voice recognition will not work on a virtual emulator as the mic in the app does not detect the mic from your computer. Additionally, permission needs to be added in the app>manifests>AndroidManifest.xml. “android.permission.RECORD_AUDIO”

The next portion of the application will be the JSON object created and posted to the web server through ROS. Refer to the codes in the png from lines 115 onwards.
To code the application to post JSON files to web server, we must first create an Asynchronous method called SendPostRequest() in the MainActivity.java. The Asynchronous method operates independently of other processes and will be executed in the background.
Next, we need to define the URL in Asynchronous method and also initialize the JSON object and add data into JSONobject as a key value pair.
Now, we will use an URLConnection for HTTP used to send and receive data over the web server in ros and create a HTTPURLConnection by calling URL.openConnection() and casting the result to HTTPURLConnection and setting a connection timeout, method type and configured with setDoInput(true).
After that, we need to make a string return type method called postDataString(). This method is useful for encoding a string to be used in a query part of a URL.
Now, we will encode the URL string of JSONObject. This url string send the server to get the response. we get the response via getInputStream(). And read the response through StringBuffer object. Return the response string into onPostExcute() method.
Next, we call the SendPostRequest() method to send and receive data from server end. This data sent will be according to the keyword spoken and received through the voice recognition. If user’s speech matches the keyword, the SendPostRequest() method will be called in the SendCommand() method which is in turn called from the if else statements where the android checks for the user’s speech and the keyword to see if there is a match.
NOTE: permissions for internet access through the application must be stated in app>manifests>AndroidManifests.xml




```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="android.hmkcode.com.voiceactivate">
    <uses-permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="VoiceActivate"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity" android:label="MyRunningApp">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <action android:name="android.intent.action.SEND" />
                <action android:name="android.intent.action.DIAL"/>
                <action android:name="android.intent.action.SENDTO"/>
                <category android:name="android.intent.category.LAUNCHER" />
                <category android:name="com.google.android.voicesearch.SELF_NOTE" />
                <category android:name="android.intent.category.LAUNCHER" />
                <category android:name="com.google.android.gms.actions.RESERVE_TAXI_RESERVATION" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

```
public class MainActivity extends AppCompatActivity implements OnClickListener {

    public ListView mList;
    public Button speakButton;
    public static final int VOICE_RECOGNITION_REQUEST_CODE = 1234;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        speakButton = (Button) findViewById(R.id.btn_speak);
        speakButton.setOnClickListener(this);
        voiceinputbuttons();
    }

    public void informationMenu() {

        //startActivity(new Intent("android.intent.action.DIAL"));
        new SendPostRequest().execute();

    }

    public void voiceinputbuttons() {
        speakButton = (Button) findViewById(R.id.btn_speak);
        mList = (ListView) findViewById(R.id.list);
    }

    public void startVoiceRecognitionActivity() {
        Intent intent = new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
                "Speech Recognition");
        startActivityForResult(intent, VOICE_RECOGNITION_REQUEST_CODE);
    }
```

```
    public void onClick(View v) {
        // TODO Auto-generated method stub
        startVoiceRecognitionActivity();
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);

        if (requestCode == VOICE_RECOGNITION_REQUEST_CODE && resultCode == RESULT_OK) {
            // Fill the list view with the strings the recognizer thought it
            // could have heard
            ArrayList<String> matches = data.getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
            mList.setAdapter(new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, matches));
            // matches is the result of voice input. It is a list of what the
            // user possibly said.
            // Using an if statement for the keyword you want to use allows the
            // use of any activity if keywords match
            // it is possible to set up multiple keywords to use the same
            // activity so more than one word will allow the user
            // to use the activity (makes it so the user doesn't have to
            // memorize words from a list)
            // to use an activity from the voice input information simply use
            // the following format;
            // if (matches.contains("keyword here") { startActivity(new
            // Intent("name.of.manifest.ACTIVITY")

            if (matches.contains("stop")) {
                TextView displaytext = (TextView) findViewById(R.id.textView);
                displaytext.setText("stop");
                 informationMenu();
            }
            else if(matches.contains("door")){
                TextView displaytext = (TextView) findViewById(R.id.textView);
                displaytext.setText("door");
                informationMenu();
            }
```

```
            }
            else if(matches.contains("continue")){
                TextView displaytext = (TextView) findViewById(R.id.textView);
                displaytext.setText("continue");
                SendCommand();
            }
            else if(matches.contains("cancel")){
                TextView displaytext = (TextView) findViewById(R.id.textView);
                displaytext.setText("cancel");
                SendCommand();
            }

        }
    }
    public class SendPostRequest extends AsyncTask<String, Void, String> {

        TextView displaytext = (TextView) findViewById(R.id.textView);
        String ExtraMessage = displaytext.getText().toString();
        protected void onPreExecute(){}

        protected String doInBackground(String... arg0) {

            try {
                URL url = new URL("example.com:8050"); // here is your URL path
                JSONObject postDataParams = new JSONObject();
                TextView displaytext = (TextView) findViewById(R.id.textView);
                postDataParams.put("Data:",ExtraMessage);
                Log.e("params",postDataParams.toString());

                HttpURLConnection conn = (HttpURLConnection) url.openConnection();
                conn.setReadTimeout(15000 /* milliseconds */);
                conn.setConnectTimeout(15000 /* milliseconds */);
                conn.setRequestMethod("POST");
                conn.setDoInput(true);
                conn.setDoOutput(true);

                OutputStream os = conn.getOutputStream();
```

```
                BufferedWriter writer = new BufferedWriter(
                        new OutputStreamWriter(os, "UTF-8"));
                writer.write(getPostDataString(postDataParams));

                writer.flush();
                writer.close();
                os.close();

                int responseCode=conn.getResponseCode();

                if (responseCode == HttpsURLConnection.HTTP_OK) {

                    BufferedReader in=new BufferedReader(new
                            InputStreamReader(
                            conn.getInputStream()));

                    StringBuffer sb = new StringBuffer("");
                    String line="";

                    while((line = in.readLine()) != null) {

                        sb.append(line);
                        break;
                    }

                    in.close();
                    return sb.toString();

                }
                else {
                    return new String("false : "+responseCode);
                }
            }
            catch(Exception e){
                return new String("Exception: " + e.getMessage());
            }
```

```
172                 return new String("Exception: " + e.getMessage());
173             }
174
175         }
176
177         @Override
178         protected void onPostExecute(String result) {
179             Toast.makeText(getApplicationContext(), result,
180                     Toast.LENGTH_LONG).show();
181         }
182     }
183
184     public String getPostDataString(JSONObject params) throws Exception {
185
186         StringBuilder result = new StringBuilder();
187         boolean first = true;
188
189         Iterator<String> itr = params.keys();
190
191         while(itr.hasNext()){
192
193             String key= itr.next();
194             Object value = params.get(key);
195
196             if (first)
197                 first = false;
198             else
199                 result.append("&");
200
201             result.append(URLEncoder.encode(key, "UTF-8"));
202             result.append("=");
203             result.append(URLEncoder.encode(value.toString(), "UTF-8"));
204
205         }
206         return result.toString();
207     }
208
```

Rosserial and Arduino

Setup :http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup
Arduino IDE download : https://www.arduino.cc/en/Main/Software

```
$ sudo apt-get install ros-indigo-rosserial-arduino
```

OR

```
$ git clone https://github.com/ros-drivers/rosserial.git
```

We used sudo apt-get install.
For the rest of the setup follow the wiki page and there should be no problem.

For the people who are too lazy to check the wiki just do this :D **:**

```
$ cd ~/sketchbook/libraries
$ rm -rf ros_lib
$ rosrun rosserial_arduino make_libraries.py
```

Now you should have ros_lib under file -> examples in arduino IDE

To run rosserial open a new terminal and do:

```
$ rosrun rosserial_python serial_node.py /dev/ttyUSB0
```

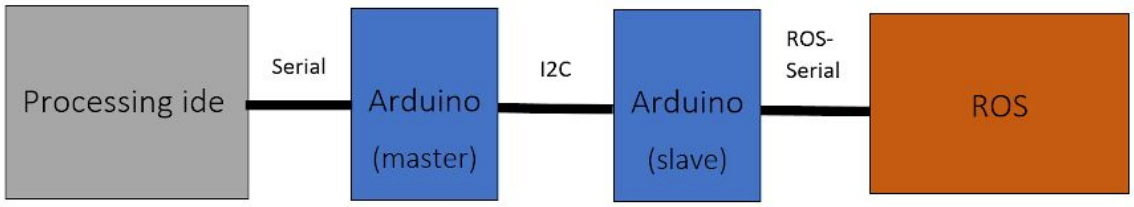
*This is the port your arduino is connected to. It can be ttyUSB or ttyACM

User interface

For this part, you have to download processing ide and arduino ide on a windows computer and arduino ide on ubuntu. You will also need the rosserial package.

Processing ide
This is a java program which we are using a for our user interface. The reason why we are using this is because we want to use an interface that is wired to our main processing unit so that we do not need to worry about network security and stability problems that might occur during use.

This interface is what the arduino ide is based on so it is used with arduino, it uses serial to communicate with the arduino board. But since we are also using rosserial to communicate with the arduino board we have a problem what the arduino board reads and writes using serial. So we decided to use 2 arduino boards, the master interfacing with processing and relaying the user input to the slave which sends the input to the ROS system via a node we created.




```
PImage bg;
PFont a;
ButtonB B_button;
ButtonE E_button;
ButtonC C_button;
ButtonD D_button;
ButtonF F_button;
ButtonFD FD_button;
ButtonW W_button;
ButtonCD CD_button;
ButtonCN CN_button;
ButtonH H_button;

import processing.serial.*;
Serial myPort;
String PT;
String myString=null;
int value, xy = 1, ptX, ptY;
float current_X, old_view_x=1080, myValueX, current_Y, old_view_y=380,
myValueY, old_X=2.56, old_Y=2.22, change_x, change_y, view_change_x,
view_change_y, sss=3.123;


void setup() {
  size(1315, 747);
  printArray(PFont.list());
  a = createFont("SourceCodePro-Regular.ttf", 28);
  textFont(a);
```

```
  bg = loadImage("points.jpg");
  E_button = new ButtonE("E-BRAKE", 120, 200, 200, 200);
  B_button = new ButtonB("BASE", 1080, 380, 100, 100);
  C_button = new ButtonC("CUPBOARD", 1050, 520, 100, 100);
  D_button = new ButtonD("DOOR", 1100, 130, 100, 100);
  F_button = new ButtonF("FISH", 960, 160, 100, 100);
  FD_button = new ButtonFD("FRONTDOOR", 440, 130, 100, 100);
  W_button = new ButtonW("WORKSTATION", 440, 520, 100, 100);
  CD_button = new ButtonCD("CANCEL DESTINATION", 357, 650, 600, 80);
  CN_button = new ButtonCN("CONTINUE NAV", 120, 450, 200, 80);
  H_button = new ButtonH("HORN", 130, 40, 180, 100);
  smooth();
  myPort = new Serial(this, "COM3", 57600);
  myPort.bufferUntil('\n');
  myString = myPort.readStringUntil('\n');
  myString = null;
}


void draw() {
  while(myPort.available()>0)
  {
    myString = myPort.readStringUntil('\n');
    if (myString != null) {

      myString = trim(myString);
      //value = int(myString);//test
      if(xy==1)
      {
        myValueX=float(myString);//int
        xy=0;
      }
      else
      {
        myValueY=float(myString);//int
        xy=1;
      }
      //println(myValueY);

    }
  }
  if (B_button.MouseIsOver()) {
  rect(1080, 380, 100, 100);
```

```
 //myPort.write('S');
 }
 else if (E_button.MouseIsOver()) {
 fill(250,120,0);
 rect(110, 190, 220, 220);
 }
 else if (C_button.MouseIsOver()) {
 rect(1050, 520, 100, 100);
 }
 else if (D_button.MouseIsOver()) {
 rect(1100, 130, 100, 100);
 }
 else if (F_button.MouseIsOver()) {
 rect(960, 160, 100, 100);
 }
 else if (FD_button.MouseIsOver()) {
 rect(440, 130, 100, 100);
 }
 else if (W_button.MouseIsOver()) {
 rect(440, 520, 100, 100);
 }
 else if (CD_button.MouseIsOver()) {
 rect(347, 645, 620, 90);
 }
 else if (CN_button.MouseIsOver()) {
 fill(150,120,250);
 rect(115, 445, 210, 90);
 }
 else if (H_button.MouseIsOver()) {
 fill(150,0,0);
 rect(125, 35, 190, 110);
 }

 else {
   // hide the square if the mouse cursor is not over the button
   background(bg);
 }
// background(bg);
 B_button.Draw();
 E_button.Draw();
 C_button.Draw();
 D_button.Draw();
 F_button.Draw();
```

```
  FD_button.Draw();
  W_button.Draw();
  CD_button.Draw();
  CN_button.Draw();
  H_button.Draw();

  //.....................x-coor
  change_x=old_X-myValueX;//in cordinate
  //println(old_X);
  view_change_x=change_x*37.3;//change in pixel
  current_X=old_view_x+view_change_x;
  ptX=Math.round(current_X);
  //println(ptX);
  old_view_x=current_X;
  old_X=myValueX;
  //.....................y-coor
  change_y=old_Y-myValueY;//in cordinate
  //println(old_Y);
  view_change_y=change_y*37.3;//change in pixel
  current_Y=old_view_y+view_change_y;
  ptY=Math.round(current_Y);
  //println(ptY);
  old_view_y=current_Y;
  old_Y=myValueY;
  //------------------point printed
  fill(0,255,0);
  ellipse(ptX+40,ptY+60,20,20);
}
void mousePressed()
{
  if (B_button.MouseIsOver()) {
    // print some text to the console pane if the button is clicked
    println("Clicked_BASE ");
    myPort.write('1');
    //println(clk++);
  }
  else if (C_button.MouseIsOver()) {
    // print some text to the console pane if the button is clicked
    println("Clicked_CUPBOARD ");
    myPort.write('2');
    //println(clk++);
  }
  else if (D_button.MouseIsOver()) {
```

```
  // print some text to the console pane if the button is clicked
  println("Clicked_DOOR ");
  myPort.write('3');
  //println(clk++);
}
  else if (F_button.MouseIsOver()) {
  // print some text to the console pane if the button is clicked
  println("Clicked_FISH ");
  myPort.write('4');
  //println(clk++);
}
  else if (FD_button.MouseIsOver()) {
  // print some text to the console pane if the button is clicked
  println("Clicked_FRONTDOOR ");
  myPort.write('5');//with Draft_AW
  //println(clk++);
}
  else if (W_button.MouseIsOver()) {
  // print some text to the console pane if the button is clicked
  println("Clicked_WORKSTATION ");
  myPort.write('6');
  //println(clk++);
}
  else if (E_button.MouseIsOver()) {
  // print some text to the console pane if the button is clicked
  println("Clicked_Ebrake ");
  myPort.write('7');
  //println(clk++);
}
  else if (CD_button.MouseIsOver()) {
  // print some text to the console pane if the button is clicked
  println("Clicked_cancel destination ");
  myPort.write('8');
  //println(clk++);
}
  else if (CN_button.MouseIsOver()) {
  // print some text to the console pane if the button is clicked
  println("Clicked_continue navigation ");
  myPort.write('9');
  //println(clk++);
}
  else if (H_button.MouseIsOver()) {
  // print some text to the console pane if the button is clicked
```

```
    println("Clicked_continue navigation ");
    myPort.write('a');
    //println(clk++);
  }
  else
  {
    myPort.write('0');
  }
}
```

Code explained

This is the load the map as the background image. The size of the interface should be the same as the size of the image.

```
bg = loadImage("points.jpg");
size(1315, 747);
```

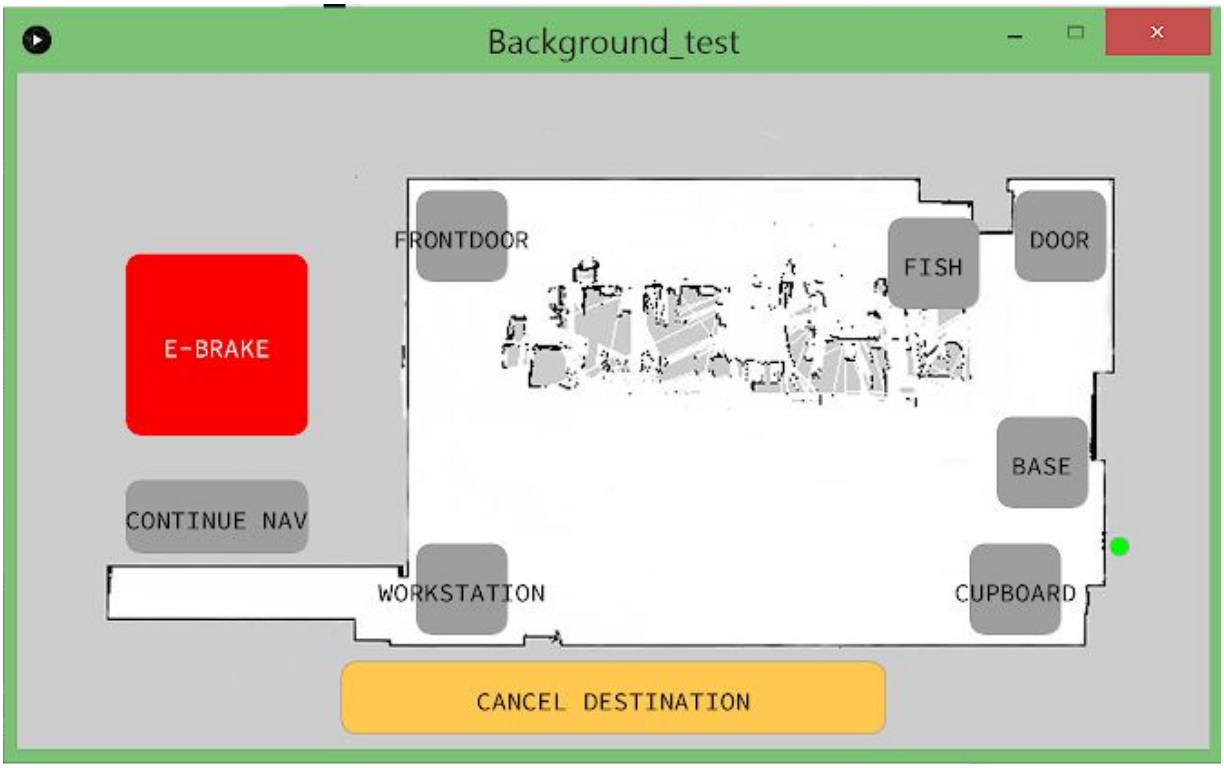
This is the procedure to define a button

```
ButtonE E_button;//define button

E_button = new ButtonE("E-BRAKE", 120, 200, 200, 200);//initialise button

E_button.Draw();// to print the button on the interface
```

This is the algorithm we created to convert the coordinates received from ros into the position at which the surrounding pixels will light up to show the real time position on the wheelchair.
The value 37.3 is gotten using interpolation of the coordinates of the points on the map and the ratio of change of the points on the background image which is the same map(This value differs according to how big the map is on the interface)

```
  //.................x-coor
  change_x=old_X-myValueX;//in cordinate
  //println(old_X);
```

```
view_change_x=change_x*37.3;//change in pixel
current_X=old_view_x+view_change_x;
ptX=Math.round(current_X);
//println(ptX);
old_view_x=current_X;
old_X=myValueX;
//.....................y-coor
change_y=old_Y-myValueY;//in cordinate
//println(old_Y);
view_change_y=change_y*37.3;//change in pixel
current_Y=old_view_y+view_change_y;
ptY=Math.round(current_Y);
//println(ptY);
old_view_y=current_Y;
old_Y=myValueY;
```



Added features: Ebrake and cancel destination

These buttons will publish different values to /chatter. Depending on which value is published, it will launch a different pre-made waypoint launch file. For this to work effectively, we use the node we created that listens to /chatter that determines which file to launch. (waypointlaunch.py - refer to previous section)

Connecting to Arduino using rosserial. Connect the arduino board to the computer, then open arduino IDE. Go to tools, and choose the correct board and select the port you want it connected to. Now open a new terminal

Now create a class for each button example...

```
class ButtonE {
  String label; // button label
  float x;      // top left corner x position
  float y;      // top left corner y position
  float w;      // width of button
  float h;      // height of button

  // constructor
  ButtonE(String labelB, float xpos, float ypos, float widthB, float heightB) {
    label = labelB;
    x = xpos;
    y = ypos;
    w = widthB;
    h = heightB;
  }

  void Draw() {
    fill(250,0,0);//RGB (bright red)
    stroke(255,0,0);
    rect(x, y, w, h, 20);
    textAlign(CENTER, CENTER);
    fill(255);
    text(label, x + (w / 2), y + (h / 2));

  }

  boolean MouseIsOver() {
    if (mouseX > x && mouseX < (x + w) && mouseY > y && mouseY < (y + h)) {
      return true;
    }
    return false;
  }
}
```

This is the load the map as the background image. The size of the interface should be the same as the size of the image.

```
ButtonE E_button;//define button

E_button = new ButtonE("E-BRAKE", 120, 200, 200, 200);//initialise button

E_button.Draw();// to print the button on the interface
```

Arduino (master)

In the master program, we need to receive the commands from the processing interface and relay the coordinates from the salve to the interface.

```
//MASTER...connect slave to processing
#include <Wire.h>

char destination, XY=1;
float x=2.22,y=2.56;
int c1;//x=2.22, y=2.56,c;//x and y coordinate respectively
void setup() {
 Wire.begin();
 Serial.begin(57600);
}

void loop() {
 if(Serial.available() > 0)
{
 destination = Serial.read();//receiving data from processing

 Wire.beginTransmission(8);
 Wire.write(destination);
 Wire.endTransmission();
}


//Serial.println("hi");
//delay(100);//testing
//Serial.println(y);//sending data
delay(2000);// update 5 times per second

//request coordinate from slave
```

```
Wire.requestFrom(8, 1);    // request 1 byte from slave device #8

while (Wire.available()) { // while

    c1 = Wire.read(); // receive a byte as character
    //Serial.println(c1);

    if(XY=1)
    {
     x=((float)c1)/(10.00);
     XY=0;
     Serial.println(x);

    }
    else
    {
     y=((float)c1)/10.00;
     XY=1;
     Serial.println(y);
    }
}

}
```

Sending the commands from processing to the slave.

```
Wire.beginTransmission(8);
Wire.write(destination);
Wire.endTransmission();
```

Since the coordinates are sent one after the other (x first then y), this algorithm is to receive and arrange the coordinates accordingly.

```
while (Wire.available()) { // while

    c1 = Wire.read(); // receive a byte as character
    //Serial.println(c1);

    if(XY=1)
```

```
    {
     x=((float)c1)/(10.00);
     XY=0;
     Serial.println(x);

    }
    else
    {
     y=((float)c1)/10.00;
     XY=1;
     Serial.println(y);
    }
}
```

Arduino (slave)

In the slave program, it needs to receive commands from the interface through the master to send to ROS as well as receive the coordinates from ROS to send to the master. Also this program has to receive feedback from the ultrasonic sensors and send commands to ROS to slow, stop turn as well as controlling the rear and turning indicators.
(Remember to download the ros.h library on machine running ros)

```
#include <Wire.h>
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Char.h>

ros::NodeHandle  nh;

std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg);


#define COMMANDREGISTER 0x00
#define RESULTREGISTER  0x02

#define CENTIMETERS 0x51

#define NO_OF_SENSORS 4

int SEQUENCE[] = {112, 115, 116, 123};//0x70,0x73,0x74,0x7B arrange in terms of
```

```
left to right

int reading = 0;
int ll,lf,rf,rr,lla,lfa,rfa,rra,llac=0,lfac=0,rfac=0,rrac=0;
String stringData;


char b[13]= "1";
char c[13]= "2";
char dd[13]= "3";
char f[13]= "4";
char fd[13]= "5";
char w[13]= "6";
char e[13]= "7";
char cd[13]= "8";
char cn[13]= "9";
char is[13]= "11";
char gs[13]= "12";
char ir[13]= "14";
char gr[13]= "13";
char il[13]= "16";
char gl[13]= "15";
char ss[13]= "17";
char tt[13]= "0";

float x, y, xc, yc;
char xy=1,a;

void messagetf( const std_msgs::Char& tfpub){

    a = tfpub.data;
}


ros::Subscriber<std_msgs::Char> sub("tfpub", &messagetf );

void setup()
{
  Serial.begin(57600);
  nh.initNode();
  nh.advertise(chatter);
  Wire.begin(8);
  Wire.onRequest(requestEvent);
```

```
  Wire.onReceive(receiveEvent);
  nh.subscribe(sub);
  pinMode(9,OUTPUT);//rear light indicator
  pinMode(10,OUTPUT);//right light indicator
  pinMode(11,OUTPUT);//left light indicator
  pinMode(12,OUTPUT);//Horn
}

long publisher_timer;

void loop()
{
if (millis() > publisher_timer || 1) {

   for (int i = 0; i < NO_OF_SENSORS; i++) {
     takeData(SEQUENCE[i]);
   }
   delay(70); //wait for reading

   readData(SEQUENCE[0]);
   //lf=700;
   stringData = String(reading);
   if(llac==0)
      {
        lla=800;
        if(reading<lla)
        {
          lla=reading;
        }
        llac++;
      }
      else if(llac<4)
      {
        if(reading<lla)
        {
          lla=reading;
        }
        llac++;
      }
      else
      {
       if(reading<lla)
        {
```

```
        lla=reading;
      }
      ll=lla;
      llac=0;
    }


for (int i = 1; i < NO_OF_SENSORS; i++) {
  readData(SEQUENCE[i]);
  stringData = ' ' + String(reading);
  if(i==1)
  {
    if(lfac==0)
    {
      lfa=800;
      if(reading<lfa)
      {
        lfa=reading;
      }
      lfac++;
    }
    else if(lfac<4)
    {
      if(reading<lfa)
      {
        lfa=reading;
      }
      lfac++;
    }
    else
    {
     if(reading<lfa)
      {
        lfa=reading;
      }
      lf=lfa;
      lfac=0;
    }
  }
  else if(i==2)
  {
    if(rfac==0)
    {
```

```
      rfa=800;
      if(reading<rfa)
      {
        rfa=reading;
      }
      rfac++;
    }
    else if(rfac<4)
    {
      if(reading<rfa)
      {
        rfa=reading;
      }
      rfac++;
    }
    else
    {
     if(reading<rfa)
      {
        rfa=reading;
      }
      rf=rfa;
      rfac=0;
    }
  }
  else if(i==3)
  {
    if(rrac==0)
    {
      rra=800;
      if(reading<rra)
      {
        rra=reading;
      }
      rrac++;
    }
    else if(rrac<4)
    {
      if(reading<rra)
      {
        rra=reading;
      }
      rrac++;
```

```
    }
    else
    {
     if(reading<rra)
      {
        rra=reading;
      }
      rr=rra;
      rrac=0;
    }
  }
}
if(rr<40&&rf<50&&ll<40&&lf<50)
{
  digitalWrite(9,LOW);
  digitalWrite(12,HIGH);
  str_msg.data = is;
  chatter.publish( &str_msg );
}
else if(rr<40&&rf<50&&ll<40)
{
  digitalWrite(9,LOW);
  digitalWrite(12,HIGH);
  str_msg.data = is;
  chatter.publish( &str_msg );
}
else if(rr<40&&ll<40&&lf<50)
{
  digitalWrite(9,LOW);
  digitalWrite(12,HIGH);
  str_msg.data = is;
  chatter.publish( &str_msg );
}
else if(lf<=50&&rf<=50)
{
  digitalWrite(9,LOW);
  digitalWrite(12,HIGH);
  str_msg.data = is;
  chatter.publish( &str_msg );

}
else if(lf<=100&&rf<=100)
{
```

```
  digitalWrite(9,LOW);
  digitalWrite(12,HIGH);
  delay(200);
  digitalWrite(9,HIGH);
  digitalWrite(12,LOW);
  delay(200);
  str_msg.data = gs;
  chatter.publish( &str_msg );

 }
 if(ll<40&&lf<=100)
 {//gradual right
  digitalWrite(12,HIGH);
  digitalWrite(9,LOW);
  digitalWrite(11,LOW);
  delay(300);
  digitalWrite(12,LOW);
  digitalWrite(11,HIGH);
  digitalWrite(9,HIGH);
  delay(300);
  str_msg.data = gr;
  chatter.publish( &str_msg );
 }
 else if(ll<=40&&lf>50)
 {//hard right
  digitalWrite(12,HIGH);
  digitalWrite(9,LOW);
  digitalWrite(11,LOW);
  delay(300);
  digitalWrite(12,LOW);
  digitalWrite(11,HIGH);
  digitalWrite(9,HIGH);
  delay(300);
  str_msg.data = ir;
  chatter.publish( &str_msg );
 }
 else if(ll<=40&&rf>50)
 {//hard right
  digitalWrite(12,HIGH);
  digitalWrite(9,LOW);
  digitalWrite(11,LOW);
  delay(300);
  digitalWrite(12,LOW);
```

```
 digitalWrite(11,HIGH);
 digitalWrite(9,HIGH);
 delay(300);
 str_msg.data = ir;
 chatter.publish( &str_msg );
}
else if(rr<40&&rf<=100)
{//gradual left
 digitalWrite(12,HIGH);
 digitalWrite(9,LOW);
 digitalWrite(11,LOW);
 delay(300);
 digitalWrite(12,LOW);
 digitalWrite(11,HIGH);
 digitalWrite(9,HIGH);
 delay(300);
 str_msg.data = gl;
 chatter.publish( &str_msg );
}
else if(rr<=40&&rf>50)
{//hard left
 digitalWrite(12,HIGH);
 digitalWrite(9,LOW);
 digitalWrite(11,LOW);
 delay(300);
 digitalWrite(12,LOW);
 digitalWrite(11,HIGH);
 digitalWrite(9,HIGH);
 delay(300);
 str_msg.data = il;
 chatter.publish( &str_msg );
}
   else if(rr<=40&&lf>50)
{//hard left
 digitalWrite(12,HIGH);
 digitalWrite(9,LOW);
 digitalWrite(11,LOW);
 delay(300);
 digitalWrite(12,LOW);
 digitalWrite(11,HIGH);
 digitalWrite(9,HIGH);
 delay(300);
 str_msg.data = il;
```

```
    chatter.publish( &str_msg );
   }
   else if(ll<40&&rr<40)
   {
    str_msg.data = ss;
    chatter.publish( &str_msg );  //(go straight, dont turn)
   }
   else
   {
     str_msg.data = tt;//send 0
     chatter.publish( &str_msg );
     //(resume navigation)
     digitalWrite(9,HIGH);
     digitalWrite(10,HIGH);
     digitalWrite(11,HIGH);
     digitalWrite(12,LOW);
   }
   publisher_timer = millis() + 4000; //publish once a second
  }
  nh.spinOnce();
  delay(200);
}
void receiveEvent(int howMany)//receiving command from arduino master
{
  int d = Wire.read();
  if(d==49)
  {
    str_msg.data = b;
    chatter.publish( &str_msg );
  }
  if(d==50)
  {
    str_msg.data = c;
    chatter.publish( &str_msg );
  }
  if(d==51)
  {
    str_msg.data = dd;
    chatter.publish( &str_msg );
  }
  if(d==52)
  {
    str_msg.data = f;
```

```
    chatter.publish( &str_msg );
  }
  if(d==53)
  {
    str_msg.data = fd;
    chatter.publish( &str_msg );
  }
  if(d==54)
  {
    str_msg.data = w;
    chatter.publish( &str_msg );
  }
  if(d==55)
  {
    str_msg.data = e;
    chatter.publish( &str_msg );
  }
  if(d==56)
  {
    str_msg.data = cd;
    chatter.publish( &str_msg );
  }
  if(d==57)
  {
    str_msg.data = cn;
    chatter.publish( &str_msg );
  }
  if(d==97)
  {
    //activate horn
    str_msg.data = b;
    chatter.publish( &str_msg );
  }

}
void requestEvent() {
  //get both coordinates from ROS
  //xc=x*100;//get 2dp in char form
  //yc=y*100;
  //Wire.write(); // respond with message of 4 bytes(float)
    char s[] = "c";
    sprintf(s,"%c",a);
    nh.loginfo(s);
```

```
    Wire.write(s);
}

void takeData(int address) {

  // step 1: instruct sensor to read echoes
  Wire.beginTransmission(address); // transmit to device #112 (0x70)

  // the address specified in the datasheet is 224 (0xE0)
  // but i2c adressing uses the high 7 bits so it's 112
    Wire.write(byte(COMMANDREGISTER));          // sets register pointer to the
command register (0x00)
    Wire.write(byte(CENTIMETERS));          // command sensor to measure in
"centimeters" (0x51)

  Wire.endTransmission();      // stop transmitting

}

void readData(int address) {

  // step 3: instruct sensor to return a particular echo reading
  Wire.beginTransmission(address); // transmit to device #112
    Wire.write(byte(RESULTREGISTER));          // sets register pointer to echo #1
register (0x02)
  Wire.endTransmission();      // stop transmitting

  // step 4: request reading from sensor
  Wire.requestFrom(address, 2);    // request 2 bytes from slave device #112

  // step 5: receive reading from sensor
  if (2 <= Wire.available()) { // if two bytes were received
    reading = Wire.read();  // receive high byte (overwrites previous reading)
    reading = reading << 8;    // shift high byte to be high 8 bits
    reading |= Wire.read(); // receive low byte as lower 8 bits
  }

}
```

Code explained

We created this algorithm to get the lowest out of 5 readings from the ultrasonic

sensors as we identified the sudden single jumps when a small object is very close to the ultrasonic sensors.

```
if(i==1)
    {
     if(lfac==0)
     {
       lfa=800;
       if(reading<lfa)
       {
         lfa=reading;
       }
       lfac++;
     }
     else if(lfac<4)
     {
       if(reading<lfa)
       {
         lfa=reading;
       }
       lfac++;
     }
     else
     {
      if(reading<lfa)
       {
         lfa=reading;
       }
       lf=lfa;
       lfac=0;
     }
```

This is to subscribe to the ros node that publishes the coordinates of the wheelchair and relay the values to the master arduino.

```
void messagetf( const std_msgs::Char& tfpub){

   a = tfpub.data;
}


ros::Subscriber<std_msgs::Char> sub("tfpub", &messagetf );
```

```
void requestEvent() {
```

```
  //get both coordinates from ROS
  //xc=x*100;//get 2dp in char form
  //yc=y*100;
  //Wire.write(); // respond with message of 4 bytes(float)
    char s[] = "c";
    sprintf(s,"%c",a);
    nh.loginfo(s);
    Wire.write(s);//sending the coordinates
}
```

'Wire.read()' is to receive commands from the master arduino via i2c communication

```
  int d = Wire.read();
  if(d==49)
  {
    str_msg.data = b;
    chatter.publish( &str_msg );
  }
```

Also we are using 4 ultrasonic sensors, 2 on the left side of the wheelchair. 1 facing left and the other forward and the other 2 on the right side. 1 of it facing the left while the other is facing forward.
When the ultrasonic sensors detect an obstacle, the arduino processor will send commands to the rosserial node which in turns publishes chatter which is received by newWaypoint.py which publishes cmd_vel to sabertooth. Cmd_vel has greater priority than move_base thus can control the wheelchair.

Subnode

This node is to receive the coordinates from tf and publish them as tfpub to be received by the arduino(slave)

```
#!/usr/bin/env python
import rospy
import math
import time
import tf
from std_msgs.msg import String
from std_msgs.msg import Char
from geometry_msgs.msg import Twist
from geometry_msgs.msg import TransformStamped
```

```
from itertools import cycle
myIterator = cycle(range(2))

def talker():
    pub = rospy.Publisher('tfpub', Char, queue_size=2)
    var=myIterator.next()
    if var:
          pub.publish(xrounded)

    else:
          pub.publish(yrounded)

    time.sleep(2)

if __name__ == '__main__':
 rospy.init_node('tf_listener')
 listener = tf.TransformListener()
 while not rospy.is_shutdown():
          try:
          (trans,rot) = listener.lookupTransform('/map', '/base_link', rospy.Time(0))
          x = trans[0]*10#
                    y = trans[1]*10#
          xrounded = round(x,0)
          yrounded = round(y,0)

          except (tf.LookupException, tf.ConnectivityException,
tf.ExtrapolationException):
          continue
           try:
          talker()
    except rospy.ROSInterruptException:
          pass

 rospy.spin()
```

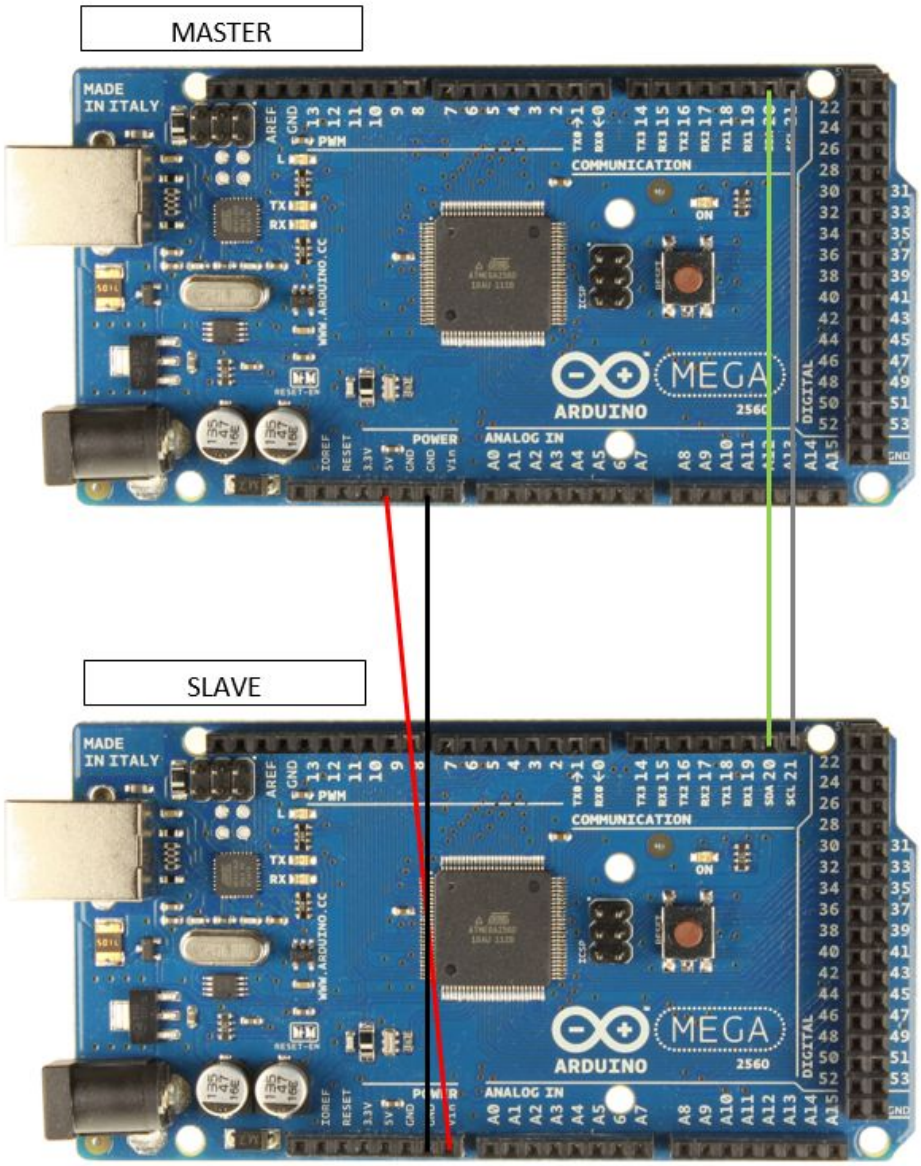
Code Explained:

```
rospy.init_node('tf_listener')
 listener = tf.TransformListener()
 while not rospy.is_shutdown():
          try:
          (trans,rot) = listener.lookupTransform('/map', '/base_link', rospy.Time(0))
          x = trans[0]*10#
```

```
                y = trans[1]*10#
        xrounded = round(x,0)
        yrounded = round(y,0)

        except (tf.LookupException, tf.ConnectivityException,
tf.ExtrapolationException):
        continue
         try:
        talker()
   except rospy.ROSInterruptException:
        pass
```

This listens to tf data, in this case the x and y coordinated of our wheelchair on the map. Values have been rounded up to the nearest whole number for simplicity's sake.
(then divided in processing to get 1d.p)

Arduino Wiring



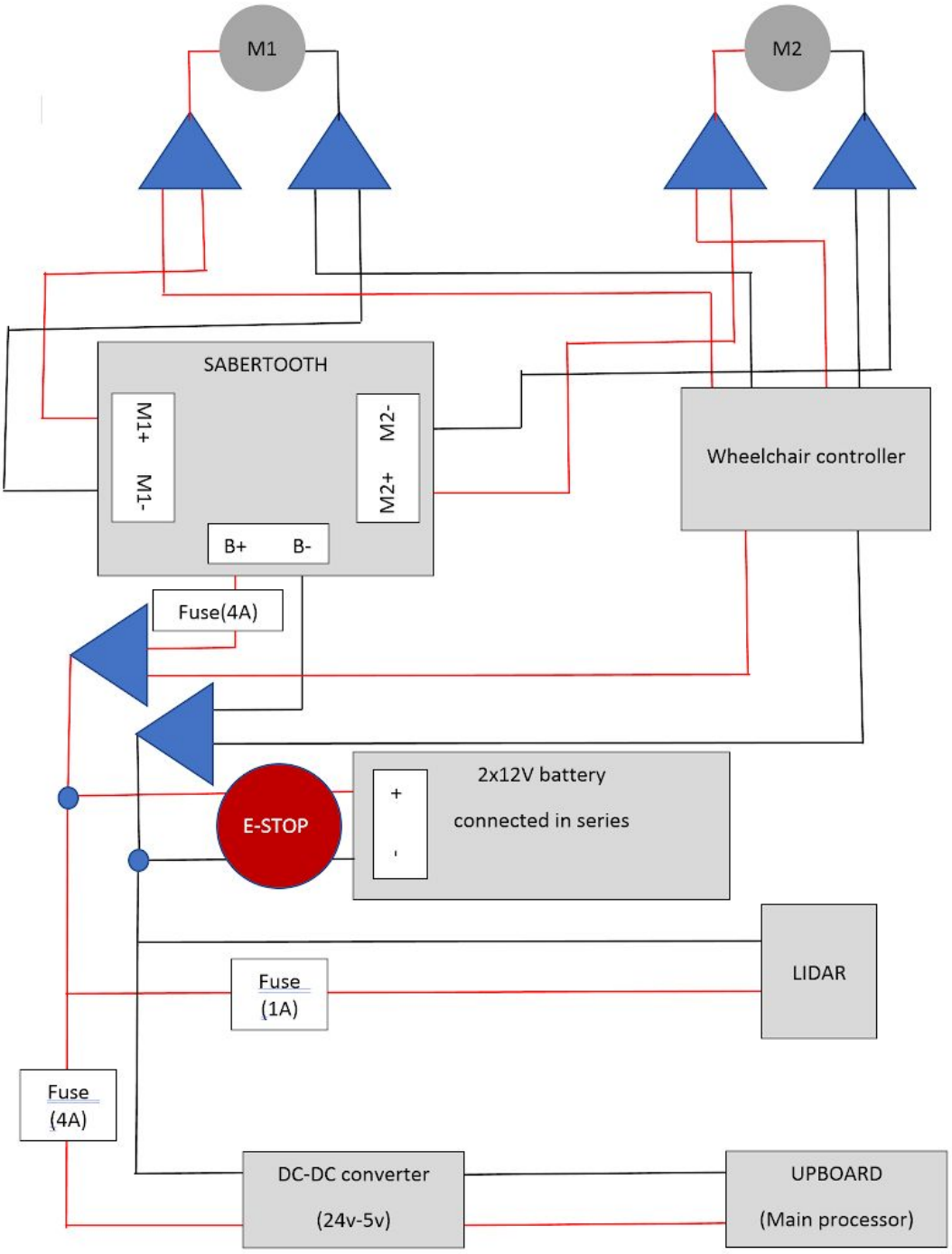
The master is to be connected to the windows device running the java(processing) interface, while the slave is to be connected to the UPboard. Since the power regulation of the usb ports on the UPboard is not very safe, it is recommended to power the slave arduino via the master(which is connected to a more stable power supply) by connecting the 5v on the master to Vin on the slave and connect the grounds on both arduino boards.

Additional connections

All the ultrasonic sensors to be connected to scl and sda.

All turning/rear indicators and horn should be connected with at least an 80ohm resistor preferably along pins 9-13 to the 5v power on the arduino board.

Final-wiring diagram



Blue triangle means 1P2T switches(one section of 4P2T switch)

Not depicted in this diagram, we included individual rocker switches for the LIDAR and UPboard.

Connection to 4P2T switches

1	4	7	10
2	5	8	11
3	6	9	12

1. Sabertooth M1A
2. Left motor+
3. Main L+
4. Sabertooth M1B
5. Left motor-
6. Main L-
7. Sabertooth M2A
8. Right motor-
9. Main R-
10. Sabertooth M2B
11. Right motor+
12. Main R+

1	4	7	10
2	5	8	11
3	6	9	12

1. Main control +
2. Batt +
3. Sabertooth B+
4. ...(unused)
5. ...
6. ...
7. ...
8. ...
9. ...
10. Main control -
11. Batt -
12. Sabertooth B-

Final running process

Ensure that the sabertooth, lidar, gamepad and arduino are connected.(Take note of the device name for the arduino, the gamepad and the sabertooth)

Launch the latest launch file made

```
$ roslaunch robot nav.launch
```

Run Sabertooth (in the correct directory)

```
$ cd ~/catkin_ws/src/robot/scripts
$ python sabertooth_drive.py
```

Run the waypoint node we made

```
$ cd ~/catkin_ws/src/robot/scripts
$ python waypoint.py
```

Run the subnode (this is the listener for x and y coordinates)

```
$ cd ~/catkin_ws/src/robot/scripts
$ python subnode.py
```

Run rosserial

```
$ rosrun rosserial_python serial_node.py /dev/ttyUSB0
```

*This is the port your arduino is connected to. It can be ttyUSB or ttyACM

Run rosbridge (if using html or android app)

```
$ roslaunch rosbridge_server rosbridge_websocket
```

With all of these running there should be no problems with the navigation.

4.Additional Software used

MIT App Inventor

MIT App inventor is a webpage where one can create projects which is made into an app for androids. A google account is required to save or access saved projects. The MIT App Inventor has several useful tools, such as a speech recognizer, clock, pedometer, location sensors etc. To create an app, you need to put in these tools in the designer tab of the inventor by adding buttons, labels and etc. Once all these tools are added, you have to put in the programs using the blocks tab. Through the MIT App Inventor, we are currently developing a user interface which will receive and send commands to the ROS Core and the wheelchair will act according to given commands. The app is currently made up of buttons which will allow sending and receiving of commands, a speech recognizer to take in voice commands and a tiny database(TinyDB) to store these commands and send to ROS Core through ROS Bridge.

Android Studios

Android Studio is the official Integrated Development Environment (IDE) for Android app development

Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

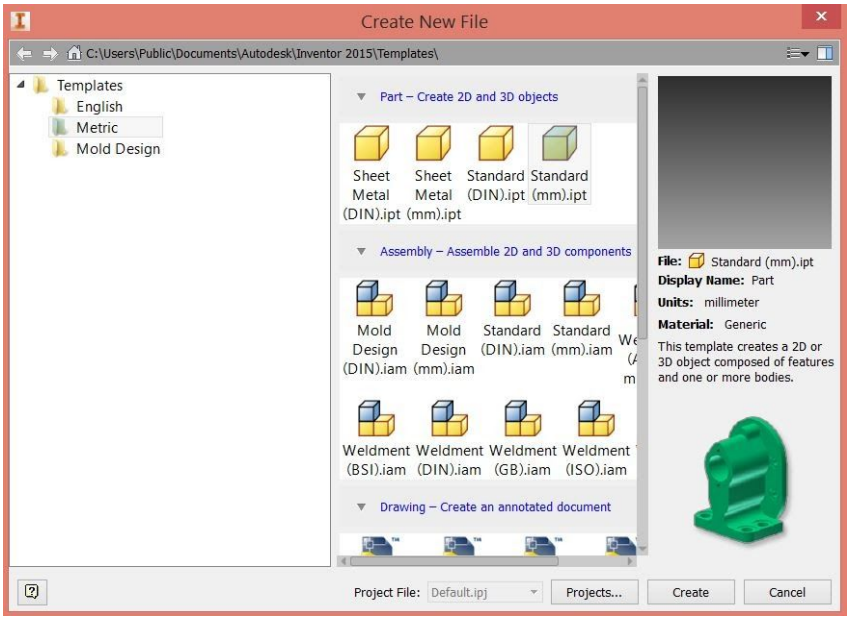
With the google cloud, we are able to access voice recognition in the android which is required in our application to allow sending of voice command.

Installation page: https://developer.android.com/studio/install.html

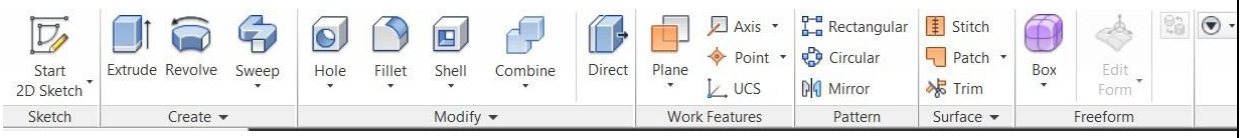
For more details, Visit:https://developer.android.com/studio/intro/index.html

Autodesk Inventor 2015

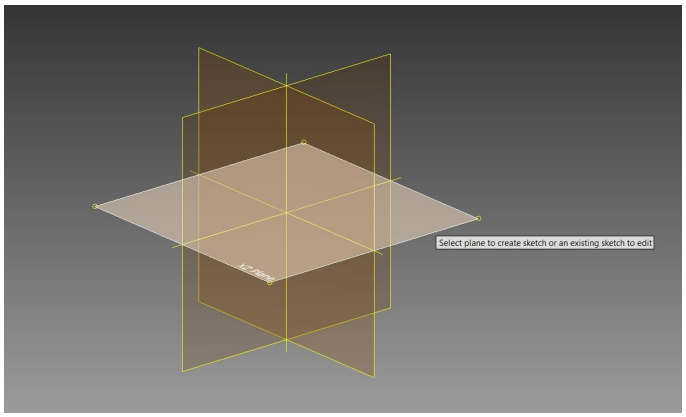
Autodesk Inventor is used to draw our mounting boards for our LIDAR and wiring for the wheelchair. When drawing the 3D images, one must always carefully measure the dimensions and consider some leeway for error, and also consider the amount of stress that may be exerted on the model once it is printed. The 3D drawing process will be briefly explained below:'



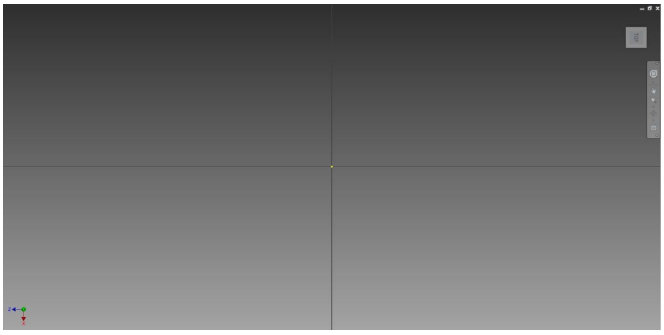
When you first start a 3D drawing, click on "New" and make sure that the Standard(mm).ipt template is selected, the click create at the bottom right.



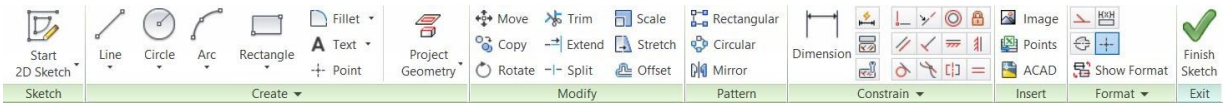
Once you have created the template, go to the top of the window and select “Start 2D Sketch”.



Once you have clicked on that, a figure with 3 planes. Select any plane to start drawing a 2D image to be turned into a 3D model.

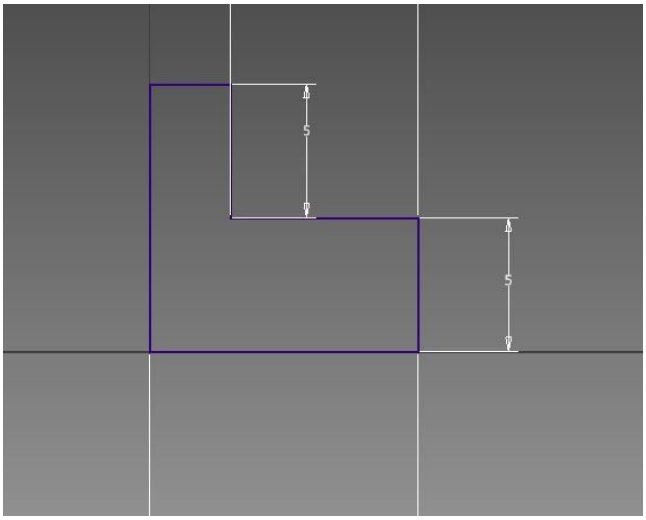


A plane with a X and Y axis will appear for the user to start drawing.

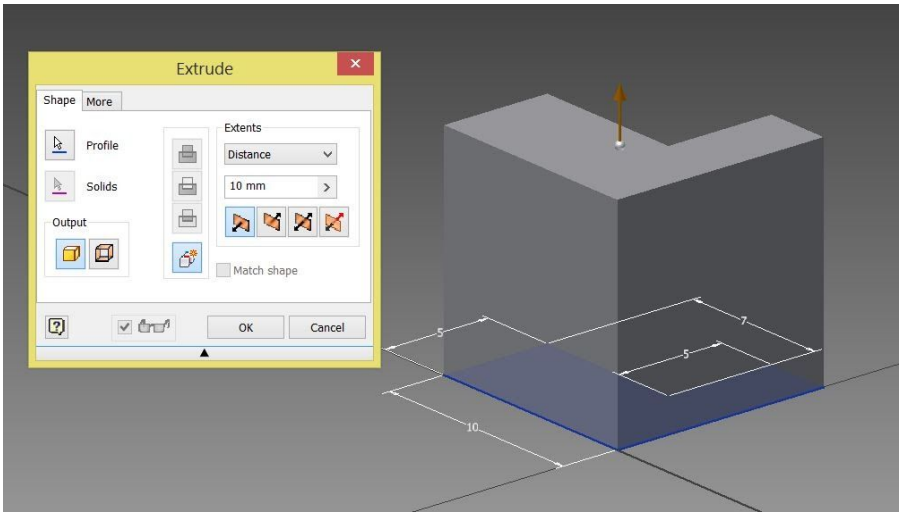


At the sketch tab, there are several tools that can be used to draw 2D image that will be converted to 3D later on.

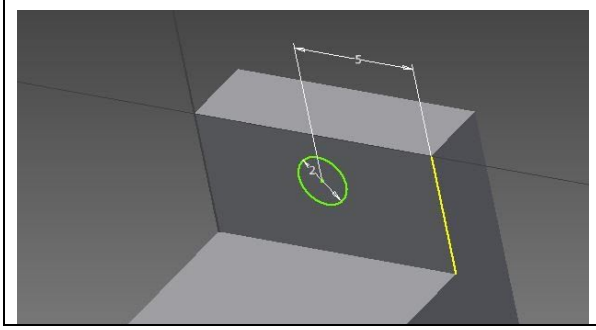
In this diagram, the line tool is used to draw a line, the angle and length of the line can be specified by clicking or pressing the “Tab” button.



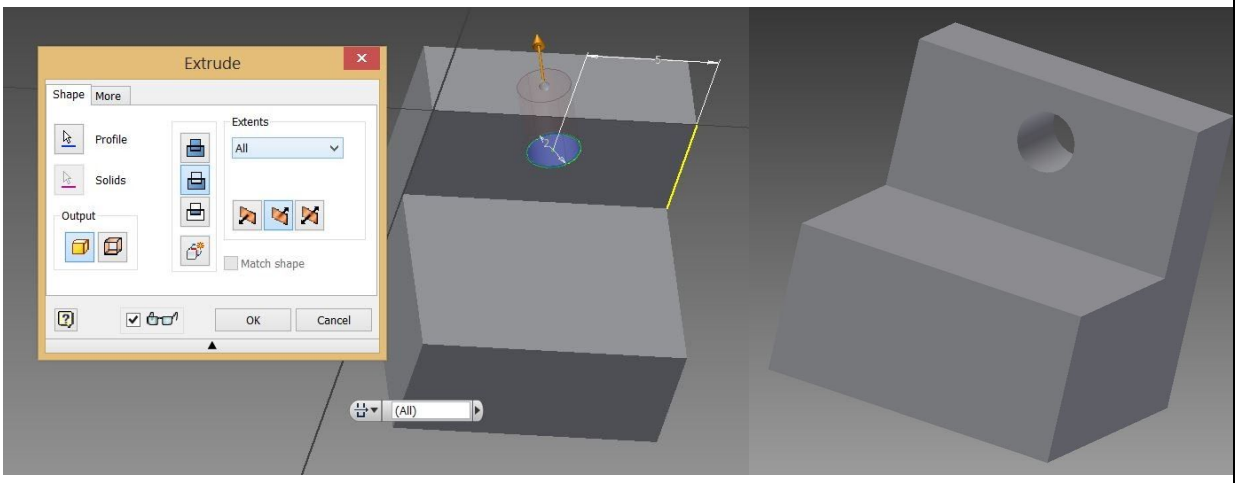
For demonstration purposes, a L bracket has been drawn using the line tool



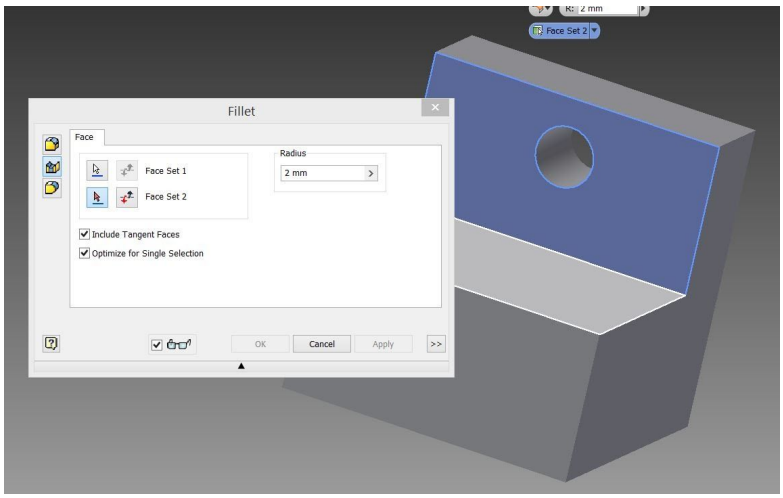
At the 3D Model tab, click on “Extrude” to create a 3d model extending outwards.



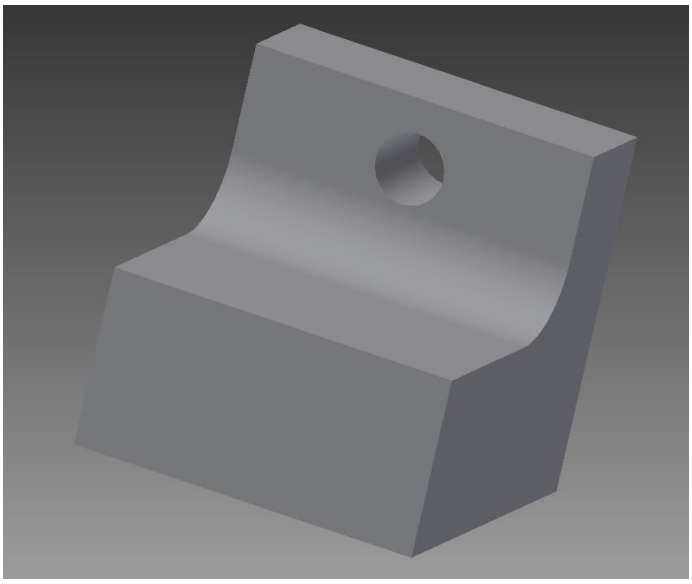
To demonstrate how to create a drill hole through the 3D Model, a circle of diameter 2mm is drawn on a surface.



Next, click on extrude once again and select the circle. Change the dropbox to All, and it should change to cutting mode, which will allow a drill hole to be cut out. The model on the right shows the hole after using the Extrude tool.



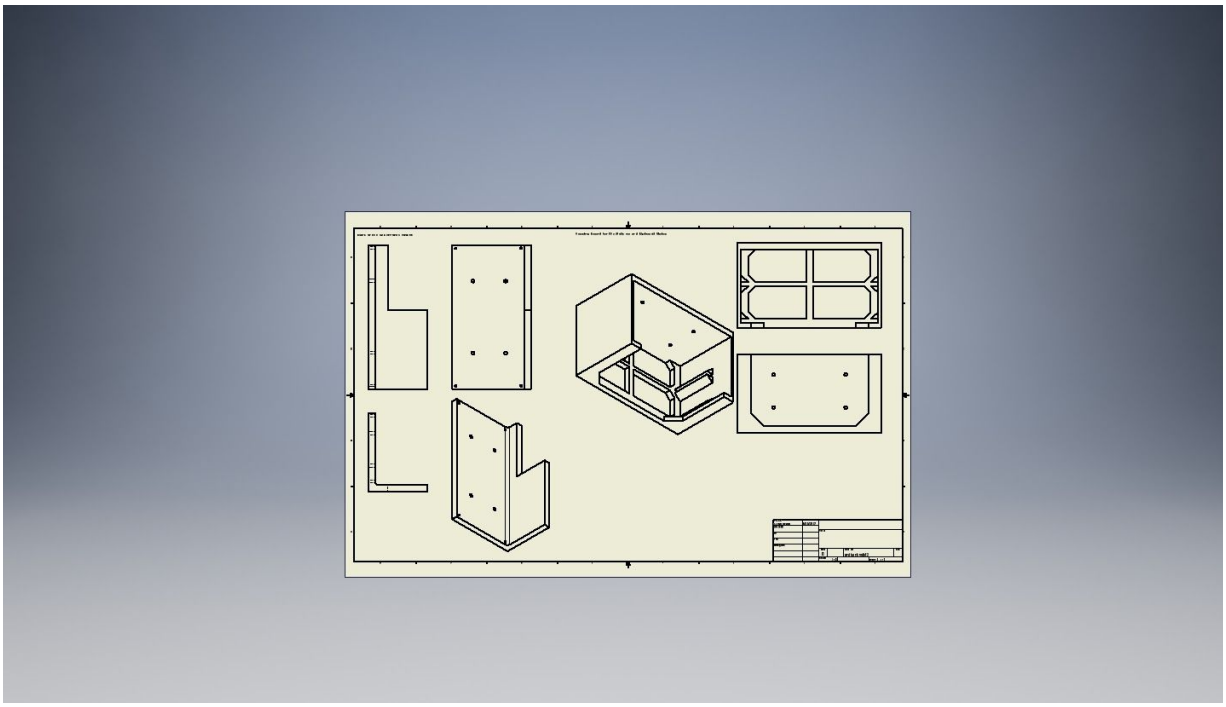
When building larger 3D models, it is important that you fillet 2 edges to reduce the amount of stress exerted on the printed boards. Using the Fillet tool, select the 2 edges for filleting, and enter a radius.



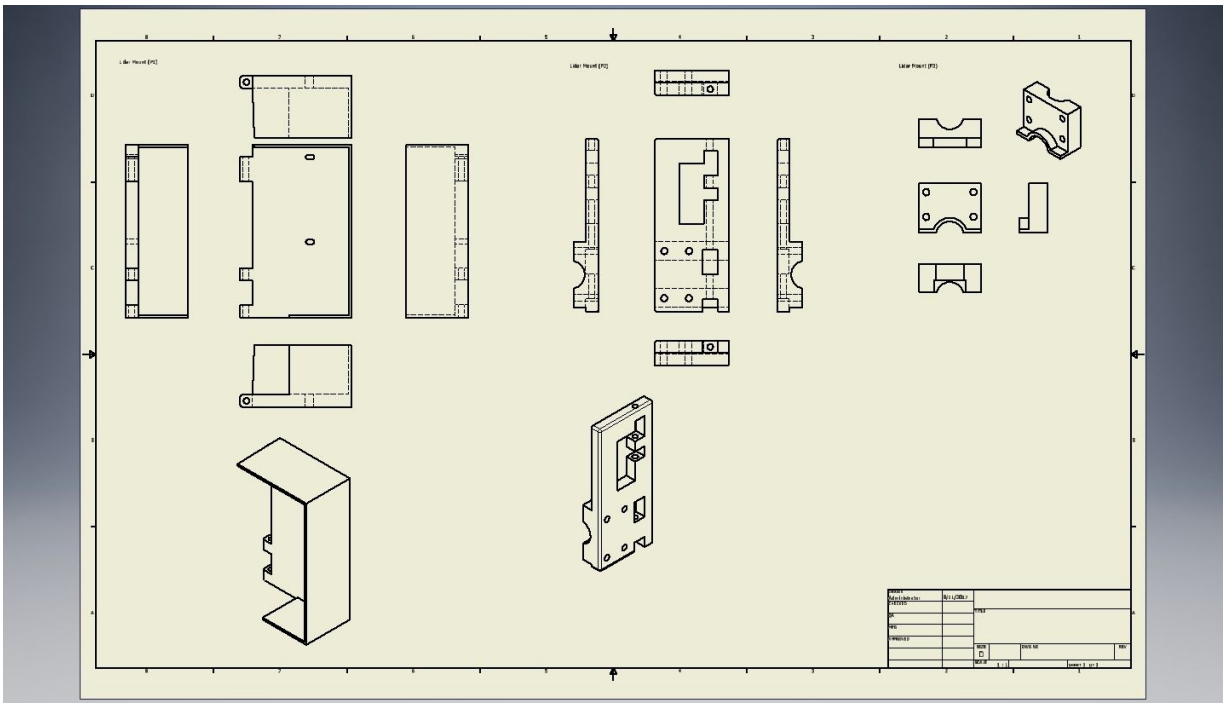
This should be the end product with a smoothened edge to act as an additional support to improve its durability and reduce fragility of protruding parts.

For reference, here are the orthographic projection drawings of the parts we printed.

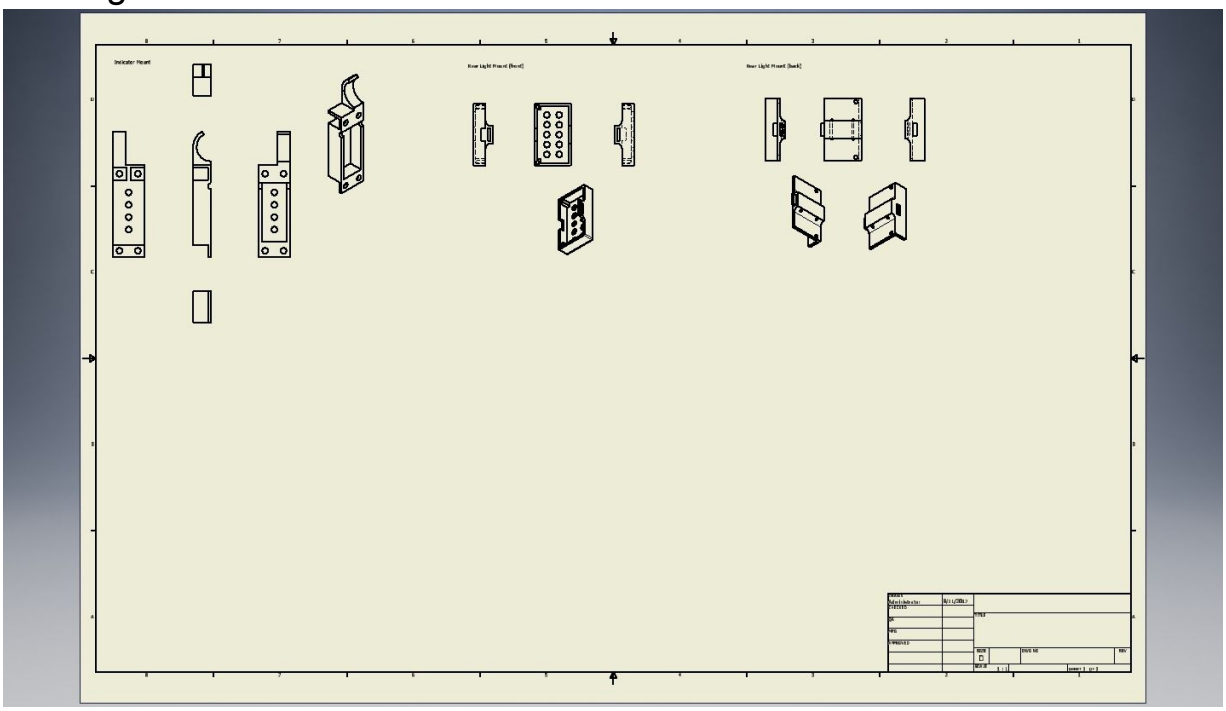
Sabertooth and Slot Mount



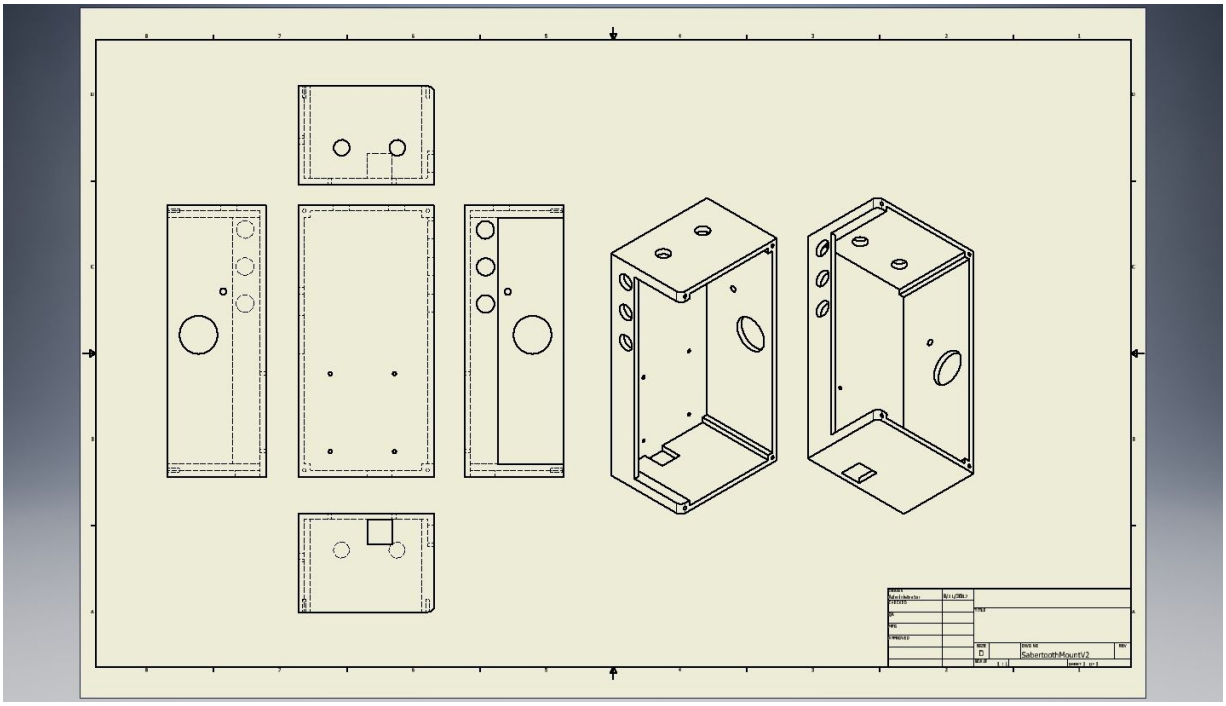
Lidar mounts



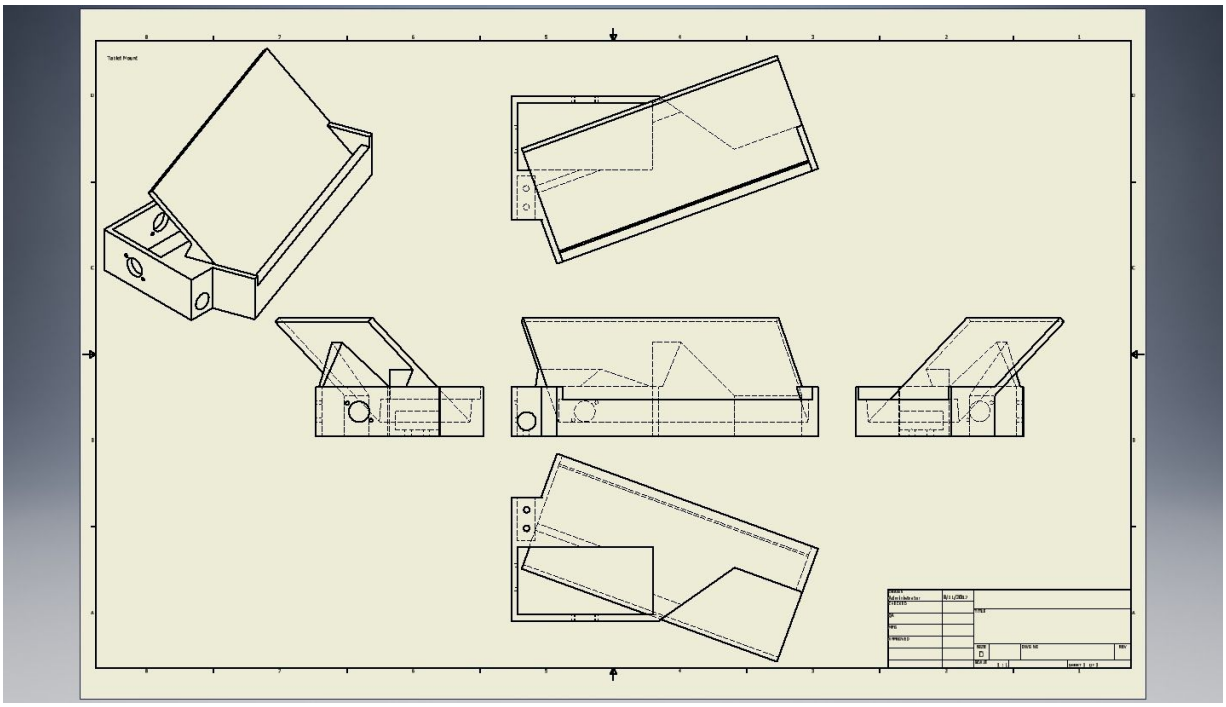
Rear light indicator mounts



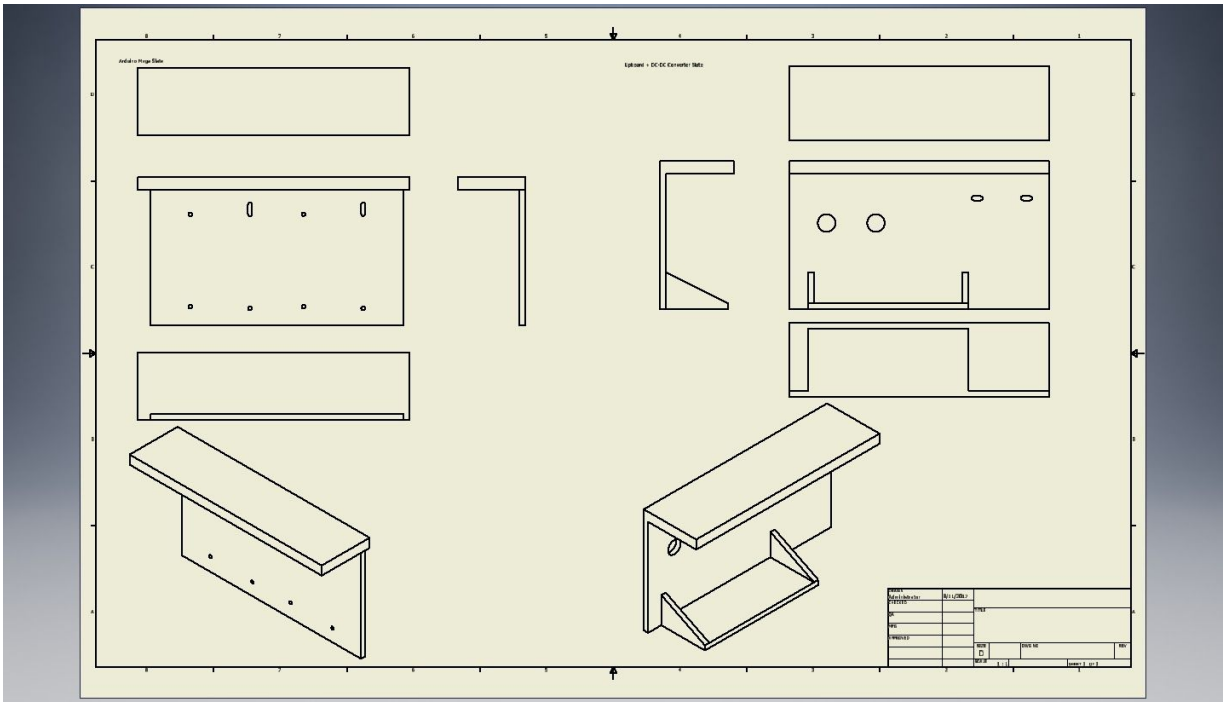
Sabertooth Mount



Tablet mounts



Slates



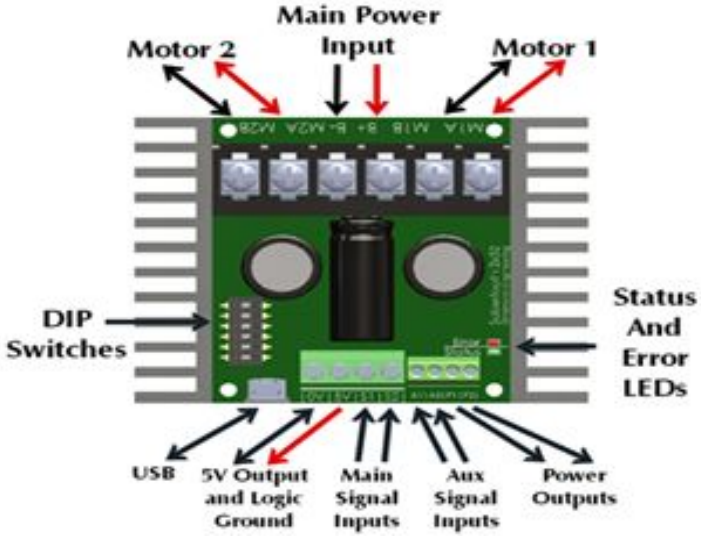
5.Components used

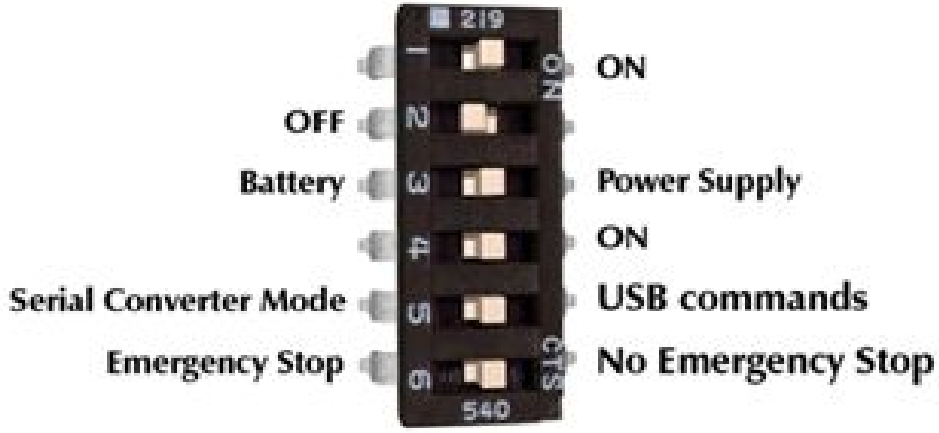
Sabertooth 2x32 Motor Controller

The Sabertooth 2x32 is a dual channel motor driver capable of supplying 32 amperes to to two motors, with peak currents up to 64 amperes per motor. There are several operating modes that can be set on the switches, such as radio control, analog, TTL serial, or USB inputs. It utilises regenerative drive and braking for efficient operation.

Additional information

The state of the driver can be monitored in real time using the USB port in any operating mode, making debugging your project faster and easier. Sabertooth 2x32 is more flexible, robust and powerful than previous motor drivers, while also being easier to use.





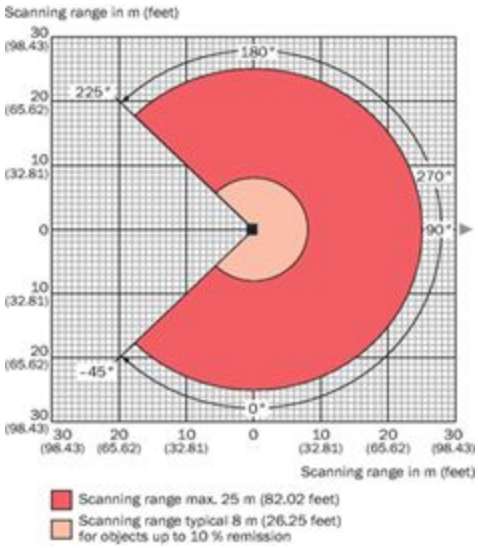
https://www.dimensionengineering.com/products/sabertooth2x32

SICK TiM 571 2D LiDAR

This is our LiDAR (Light detection and Ranging) sensor. It is used for Mapping using the GMapping packages in ROS and localisation of the wheelchair using AMCL node in ROS as the AMCL node only works with laser scans. It is electrically connected through one Ethernet connection with a 4 pin M12 Female Connector, Power and synchronisation connection using 5 pin M12 Male connector and a micro USB female connector type B.

Specifications:

- Ranging capabilities of 0.05m to maximum of 25m
- Aperture angle of 270 degrees
- Infrared Light Source of 850nm
- Scanning frequency of 15Hz
- Operating Voltage: 8V to 28V DC



https://www.sick.com/us/en/detection-and-ranging-solutions/2d-lidar-sensors/tim5xx/tim571-2050101/p/p412444

NKK SWITCHES S42F Toggle Switch

The S42F is a 4PDT non-illuminated medium/high capacity standard size panel-mount Toggle Switch with ON-None-ON switch function and quick connect switch terminals. It has PBT for flatted lever, chrome-plated brass for all other toggle, chrome-plated brass bushing and phenolic case with chrome-plating over zinc-plating steel case cover.

- 50000 Minimum operations of mechanical life
- 25000 Minimum operations of electrical life
- -10 to 70°C Operating temperature range
- Max current contact: 25A

- Contact DC nom: 30V

This switch is used to switch between Sabertooth control and the original control unit of the wheelchair. One switch is used to control the current from the battery to power the Sabertooth or the control unit, another switch is used to toggle the type of control between the 2 motors.

http://www.newark.com/nkk-switches/s42f/switch-toggle-4pdt-25a-125vac/dp/10X9483

Arduino Mega 2560

The Arduino Mega 2560 is a microcontroller board designed for robotic projects.
Technical Specifications:

- 256kB Flash Memory
- 54 Digital I/O pins
- 16 Analog Input pins
- Runs on 16MHz clock
- Operating voltage of 5V with maximum of 20V
- DC Current per I/O pin is 20mA
- DC Current per 3.3V pin is 50mA.

https://store.arduino.cc/usa/arduino-mega-2560-rev3

UpBoard Computer

The UpBoard computer have multiple features which make it available to different domains and products such as our autonomous wheelchair. It is compatible with android, linux, all windows 10 distributions which gives great flexibility.
Specifications:

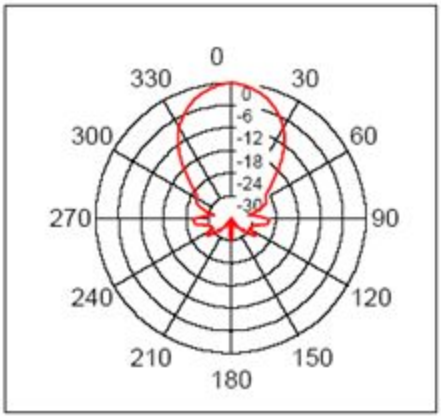
- Atom x5-Z8350 Quad-Core Processor
- 4GB RAM
- 64GB eMMC Storage Memory
- Operating Voltage: 5V
- Operating Current: 3A to 4A

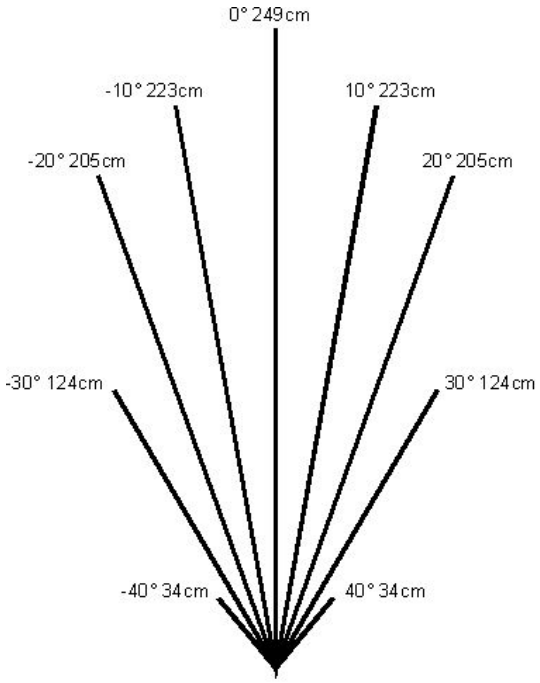
http://www.up-board.org/up/specifications/

SRF02 UltraSonic Sensors

The SRF02 UltraSonic Sensors is a single transducer ultrasonic rangefinder in a small footprint PCB which features both Serial interfaces and I2C. The serial interface is a standard TTL level UART format at 9600 baud,1 start, 2 stop and no parity bits, and may be connected directly to the serial ports on any microcontroller in this case the Arduino Mega 2560.

Its minimum measurement range is 15-16cm on a cool day and its maximum range is 249cm at 0 degrees.





https://www.robot-electronics.co.uk/htm/srf02tech.htm

DC-DC Converter

It is used to step down voltage from 24V to 5V for the Upboard Computer input.

Specifications:

- Input Voltage: 6V- 32V
- Output Voltage: 0.8V -28V (Adjustable)
- Output Current: 15A(maximum)

http://www.ebay.com.sg/itm/QS-1212CBA-DC-DC-Converter-Buck-Boost-Car-PC-Power-6-32V-to-0-8-28V-5V-150W/151409632826

3D Printed mounting board

We used AutoDesk Inventor on the PC to draw out a 3D sketch of a mounting board. This board will be used to connect to the side of the wheelchair as to help hold accessories such as an emergency button, 2 NKK Switch S42F toggle switches and the SaberTooth Motor Driver. This sketch is saved as a .stl file as the 3D Printer can only read .stl files. Once the 3D printer reads the .stl file, it will start printing. However, there are precautions to be made, such as breakaway support bridge option to be checked if there are holes to be printed vertically of the sketch.

HUAWEI Mediapad T3

We have decided to use the HUAWEI Mediapad T3 as it is a cheaper tablet as compared to the bigger brands such as Samsung
This tablet's specifications suits our needs of Android APK 7.0(nougat) with a least 2GB RAM, and Quad-Core processor, wifi connections Wi-Fi 802.11 b/g/n or a/b/g/n and available GPS.

Specifications:

Available as Wi-Fi only and Wi-Fi/LTE models

NETWORK	Technology	GSM / HSPA / LTE
LAUNCH	Announced	2017, April
	Status	Available. Released 2017, May
BODY	Dimensions	211.1 x 124.7 x 8 mm (8.31 x 4.91 x 0.31 in)
	Weight	350 g (12.35 oz)
	SIM	Yes
DISPLAY	Type	IPS LCD capacitive touchscreen, 16M colors
	Size	8.0 inches (~70.5% screen-to-body ratio)
	Resolution	800 x 1280 pixels (~189 ppi pixel density)
	Multitouch	Yes
		- EMUI 5.1
PLATFORM	OS	Android 7.0 (Nougat)
	Chipset	Qualcomm MSM8917 Snapdragon 425
	CPU	Quad-core 1.4 GHz Cortex-A53
	GPU	Adreno 308
MEMORY	Card slot	No
	Internal	16 GB, 2 GB RAM or 32 GB, 3 GB RAM
CAMERA	Primary	5 MP, autofocus
	Features	Geo-tagging
	Video	Yes
	Secondary	2 MP
SOUND	Alert types	Vibration; MP3, WAV ringtones
	Loudspeaker	Yes
	3.5mm jack	Yes
COMMS	WLAN	Wi-Fi 802.11 b/g/n or a/b/g/n, Wi-Fi Direct, hotspot
	Bluetooth	4.0, A2DP
	GPS	Yes, with A-GPS, GLONASS, BDS
	Radio	To be confirmed
	USB	microUSB 2.0
FEATURES	Sensors	Accelerometer
	Messaging	SMS(threaded view), MMS, Email, Push Mail, IM
	Browser	HTML5
	Java	No
		- MP3/WAV/Flac player - DivX/MP4/H.264 player - Document viewer
		- Document viewer - Photo/video editor
BATTERY		Non-removable Li-Ion 4800 mAh battery
MISC	Colors	Space Gray, Luxurious Gold
	Price	About 220 EUR

Disclaimer. We can not guarantee that the information on this page is 100% correct. Read more

For more details about the android, visit:
http://consumer.huawei.com/en/tablets/mediapad-t3/

6.Future development and Conclusion

Throughout this project, we were required to learn and implement various knowledge in the vast field of programming, from android to HTML and to ROS. Our journey through this project is of course not without obstacles, and we hope this documentation serves those who intend to replicate or carry on this project helpful. Of course, there is still room for improvement and we have thought out some further improvements and implementations we can add on to the Autonomous Wheelchair.

Our intended implementations and improvements include:

Hardware:

Attach rear-facing camera for convenience of user

Implement safety measures e.g stop if user gets up from wheelchair(using pressure detectors)

Improve the circuit to prevent and parasitic effects

Software:

Improve the web and android interface to he more efficient and user-friendly

Voice recognition that starts automatically instead of using a button

Voice recognition which can use multiple languages using a translator

7.Appendix

Appendix A: Terminal commands

Below are some common Terminal commands that are frequently used.

Unix/Linux Command Reference

FOSSwire.com

File Commands

ls – directory listing
ls -al – formatted listing with hidden files
cd *dir* - change directory to *dir*
cd – change to home
pwd – show current directory
mkdir *dir* – create a directory *dir*
rm *file* – delete *file*
rm -r *dir* – delete directory *dir*
rm -f *file* – force remove *file*
rm -rf *dir* – force remove directory *dir* *
cp *file1 file2* – copy *file1* to *file2*
cp -r *dir1 dir2* – copy *dir1* to *dir2*; create *dir2* if it doesn't exist
mv *file1 file2* – rename or move *file1* to *file2* if *file2* is an existing directory, moves *file1* into directory *file2*
ln -s *file link* – create symbolic link *link* to *file*
touch *file* – create or update *file*
cat > *file* – places standard input into *file*
more *file* – output the contents of *file*
head *file* – output the first 10 lines of *file*
tail *file* – output the last 10 lines of *file*
tail -f *file* – output the contents of *file* as it grows, starting with the last 10 lines

Process Management

ps – display your currently active processes
top – display all running processes
kill *pid* – kill process id *pid*
killall *proc* – kill all processes named *proc* *
bg – lists stopped or background jobs; resume a stopped job in the background
fg – brings the most recent job to foreground
fg *n* – brings job *n* to the foreground

File Permissions

chmod *octal file* – change the permissions of *file* to *octal*, which can be found separately for user, group, and world by adding:

- 4 – read (r)
- 2 – write (w)
- 1 – execute (x)

Examples:
chmod 777 – read, write, execute for all
chmod 755 – rwx for owner, rx for group and world
For more options, see **man chmod**.

SSH

ssh *user@host* – connect to *host* as *user*
ssh -p *port user@host* – connect to *host* on port *port* as *user*
ssh-copy-id *user@host* – add your key to *host* for *user* to enable a keyed or passwordless login

Searching

grep *pattern files* – search for *pattern* in *files*
grep -r *pattern dir* – search recursively for *pattern* in *dir*
command* | grep *pattern – search for *pattern* in the output of *command*
locate *file* – find all instances of *file*

System Info

date – show the current date and time
cal – show this month's calendar
uptime – show current uptime
w – display who is online
whoami – who you are logged in as
finger *user* – display information about *user*
uname -a – show kernel information
cat /proc/cpuinfo – cpu information
cat /proc/meminfo – memory information
man *command* – show the manual for *command*
df – show disk usage
du – show directory space usage
free – show memory and swap usage
whereis *app* – show possible locations of *app*
which *app* – show which *app* will be run by default

Compression

tar cf *file.tar files* – create a tar named *file.tar* containing *files*
tar xf *file.tar* – extract the files from *file.tar*
tar czf *file.tar.gz files* – create a tar with Gzip compression
tar xzf *file.tar.gz* – extract a tar using Gzip
tar cjf *file.tar.bz2* – create a tar with Bzip2 compression
tar xjf *file.tar.bz2* – extract a tar using Bzip2
gzip *file* – compresses *file* and renames it to *file.gz*
gzip -d *file.gz* – decompresses *file.gz* back to *file*

Network

ping *host* – ping *host* and output results
whois *domain* – get whois information for *domain*
dig *domain* – get DNS information for *domain*
dig -x *host* – reverse lookup *host*
wget *file* – download *file*
wget -c *file* – continue a stopped download

Installation

Install from source:
./configure
make
make install
dpkg -i *pkg.deb* – install a package (Debian)
rpm -Uvh *pkg.rpm* – install a package (RPM)

Shortcuts

Ctrl+C – halts the current command
Ctrl+Z – stops the current command, resume with **fg** in the foreground or **bg** in the background
Ctrl+D – log out of current session, similar to **exit**
Ctrl+W – erases one word in the current line
Ctrl+U – erases the whole line
Ctrl+R – type to bring up a recent command
!! - repeats the last command
exit – log out of current session

* use with extreme caution.

Appendix B: Using ROS commands

Before using any ROS commands or programmes setting up a server is necessary.

```
$ roscore
```

To eliminate the trouble to source devel/setup.bash every time you open up a terminal

```
$ sudo nano ~/.bashrc
```

add the following line into that file,

```
source ~/catkin_ws/devel/setup.bash
```

This is a tool that allows recording and playing back of ROS Topics.

```
$rosbag record <topic name> * Only 1 topic
$rosbag record -a *all topic*
```

If you want to see all possible usage of rosbag

```
$rosbag record -h *help*
```

This command allows you to change directories using a package name, stack name, or special location.

```
$roscd <package name>
```

This command allows you to creates a new ROS package with common package files:

- Manifest.xml
- CMakeLists.txt
- Makefile

```
$ catkin_create_pkg <package_name> <Package dependencies>
```

This command allows you to run an executable in an arbitrary package without having to cd (or roscd) there first.

```
$ rosrun <package_name> <executable>
```

This command is a tool for easily launching multiple ROS nodes locally and remotely. It works exactly like rosrun however letting you run multiple nodes in a launch file.

```
$ roslaunch <package_name> <launch_file>
```

Rostopic is a command-line tool for displaying debug information about ROS Topics, including publishers, subscribers, publishing rate, and ROS Messages.

```
$rostopic bw </topic_name>     display bandwidth used by topic
$rostopic echo </topic_name>  print messages to screen
$rostopic find </topic_name>  find topics by type
$rostopic hz </topic_name>    display publishing rate of topic
$rostopic info </topic_name>  print information about active topic
$rostopic list </topic_name>  print information about active topics
$rostopic pub </topic_name>   publish data to topic
$rostopic type </topic_name>  print topic type
```

Rosnode is a command-line tool for displaying debug information about ROS Nodes, including publications, subscriptions and connections.

```
$rosnode info <node_name>          print information about node
$rosnode kill <node_name>          kill a running node
$rosnode list <node_name>          list active nodes
$rosnode machine <node_name>  list nodes running on a particular machine or list machines
$rosnode ping <node_name>          test connectivity to node
$rosnode cleanup <node_name>      purge registration information of unreachable nodes
```

More details can be found at http://wiki.ros.org

Appendix C: Android Studios Keyboard Shortcuts

Description	Windows/Linux	Mac
General		
Save all	Control + S	Command + S
Synchronize	Control + Alt + Y	Command + Option + Y
Maximize/minimize editor	Control + Shift + F12	Control + Command + F12
Add to favorites	Alt + Shift + F	Option + Shift + F
Inspect current file with current profile	Alt + Shift + I	Option + Shift + I
Quick switch scheme	Control + ` (backquote)	Control + ` (backquote)
Open settings dialogue	Control + Alt + S	Command + , (comma)
Open project structure dialog	Control + Alt + Shift + S	Command + ; (semicolon)
Switch between tabs and tool window	Control + Tab	Control + Tab
Navigating and Searching Within Studio		
Search everything (including code and menus)	Press Shift twice	Press Shift twice
Find	Control + F	Command + F
Find next	F3	Command + G
Find previous	Shift + F3	Command + Shift + G
Replace	Control + R	Command + R
Find action	Control + Shift + A	Command + Shift + A
Search by symbol name	Control + Alt + Shift + N	Command + Option + O
Find class	Control + N	Command + O
Find file (instead of class)	Control + Shift + N	Command + Shift + O

Find in path	Control + Shift + F	Command + Shift + F
Open file structure pop-up	Control + F12	Command + F12
Navigate between open editor tabs	Alt + Right/Left Arrow	Control + Right/Left Arrow
Jump to source	F4 / Control + Enter	F4 / Command + Down Arrow
Open current editor tab in new window	Shift + F4	Shift + F4
Recently opened files pop-up	Control + E	Command + E
Recently edited files pop-up	Control + Shift + E	Command + Shift + E
Go to last edit location	Control + Shift + Backspace	Command + Shift + Backspace
Close active editor tab	Control + F4	Command + W
Return to editor window from a tool window	Esc	Esc
Hide active or last active tool window	Shift + Esc	Shift + Esc
Go to line	Control + G	Command + L
Open type hierarchy	Control + H	Control + H
Open method hierarchy	Control + Shift + H	Command + Shift + H
Open call hierarchy	Control + Alt + H	Control + Option + H
Writing Code		
Generate code (getters, setters, constructors, hashCode/equals, toString, new file, new class)	Alt + Insert	Command + N
Override methods	Control + O	Control + O
Implement methods	Control + I	Control + I
Surround with (if...else / try...catch / etc.)	Control + Alt + T	Command + Option + T

Delete line at caret	Control + Y	Command + Backspace
Collapse/expand current code block	Control + minus/plus	Command + minus/plus
Collapse/expand all code blocks	Control + Shift + minus/plus	Command + Shift + minus/plus
Duplicate current line or selection	Control + D	Command + D
Basic code completion	Control + Space	Control + Space
Smart code completion (filters the list of methods and variables by expected type)	Control + Shift + Space	Control + Shift + Space
Complete statement	Control + Shift + Enter	Command + Shift + Enter
Quick documentation lookup	Control + Q	Control + J
Show parameters for selected method	Control + P	Command + P
Go to declaration (directly)	Control + B or Control + Click	Command + B or Command + Click
Go to implementations	Control + Alt + B	Command + Alt + B
Go to super-method/super-class	Control + U	Command + U
Open quick definition lookup	Control + Shift + I	Command + Y
Toggle project tool window visibility	Alt + 1	Command + 1
Toggle bookmark	F11	F3
Toggle bookmark with mnemonic	Control + F11	Option + F3
Comment/uncomment with line comment	Control + /	Command + /
Comment/uncomment with block comment	Control + Shift + /	Command + Shift + /
Select successively increasing code blocks	Control + W	Option + Up
Decrease current selection to previous state	Control + Shift + W	Option + Down

Move to code block start	Control + [	Option + Command + [
Move to code block end	Control +]	Option + Command +]
Select to the code block start	Control + Shift + [	Option + Command + Shift + [
Select to the code block end	Control + Shift +]	Option + Command + Shift +]
Delete to end of word	Control + Delete	Option + Delete
Delete to start of word	Control + Backspace	Option + Backspace
Optimize imports	Control + Alt + O	Control + Option + O
Project quick fix (show intention actions and quick fixes)	Alt + Enter	Option + Enter
Reformat code	Control + Alt + L	Command + Option + L
Auto-indent lines	Control + Alt + I	Control + Option + I
Indent/unindent lines	Tab/Shift + Tab	Tab/Shift + Tab
Smart line join	Control + Shift + J	Control + Shift + J
Smart line split	Control + Enter	Command + Enter
Start new line	Shift + Enter	Shift + Enter
Next/previous highlighted error	F2 / Shift + F2	F2 / Shift + F2
Build and Run		
Build	Control + F9	Command + F9
Build and run	Shift + F10	Control + R
Apply changes (with Instant Run)	Control + F10	Control + Command + R
Debugging		
Debug	Shift + F9	Control + D
Step over	F8	F8
Step into	F7	F7
Smart step into	Shift + F7	Shift + F7
Step out	Shift + F8	Shift + F8
Run to cursor	Alt + F9	Option + F9
Evaluate expression	Alt + F8	Option + F8
Resume program	F9	Command + Option + R
Toggle breakpoint	Control + F8	Command + F8
View breakpoints	Control + Shift + F8	Command + Shift + F8
Refactoring		
Copy	F5	F5
Move	F6	F6
Safe delete	Alt + Delete	Command + Delete
Rename	Shift + F6	Shift + F6
Change signature	Control + F6	Command + F6
Inline	Control + Alt + N	Command + Option + N
Extract method	Control + Alt + M	Command + Option + M
Extract variable	Control + Alt + V	Command + Option + V
Extract field	Control + Alt + F	Command + Option + F
Extract constant	Control + Alt + C	Command + Option + C
Extract parameter	Control + Alt + P	Command + Option + P
Version Control / Local History		
Commit project to VCS	Control + K	Command + K
Update project from VCS	Control + T	Command + T
View recent changes	Alt + Shift + C	Option + Shift + C
Open VCS popup	Alt + ` (backquote)	Control + V