

**ELEE4033/4034**

**Robotics and Mechatronic Systems Engineering**

**Senior Capstone Design II**

Winter 2018

(Term II, 2017-2018)

**Final Report**

Mingyuan Shen    T02133255

Yuchen Luo        T02133244

Hao Lan            T02133236

Diwen Miao        T02133246

Wuxin Shen        T02133315

Instructor: Dr. Chaomin Luo

Department of Electrical and Computer Engineering

College of Engineering and Science

University of Detroit Mercy

Date: April 24, 2018

# Outlines:

<b>Product's Name:</b>	<b>2</b>
<b>Motivation</b>	<b>2</b>
<b>Introduction</b>	<b>2</b>
<b>Hardware Used and Some Specification</b>	<b>3</b>
<b>Hardware Logic</b>	<b>3</b>
<b>Additional Hardware we need to add</b>	<b>7</b>
<b>Software logic</b>	<b>8</b>
<b>Simulation</b>	<b>13</b>
<b>Addition software model we need to add</b>	<b>15</b>
<b>Problems encountered so far</b>	<b>15</b>
<b>The work we done</b>	<b>21</b>
<b>Plan in nest stage</b>	<b>23</b>
<b>Summary</b>	<b>23</b>
<b>Reference</b>	<b>23</b>

**ELEE4033/4034**

**Robotics and Mechatronic Systems Engineering**

**Senior Capstone Design II**

**Final Report**

### **Product Name**

Blind Guiding Robot——'RoRo'

### **Motivation**

As a vulnerable group, the blind, need society to give them more attention and care. Guide dogs are significant to the blind. But in real life, guide dogs are not suitable to help the blinds in many occasions. For example, some places do not allow pets to enter, or some people are allergic to hair. Moreover, the number of guide dogs is very limited, and training is really difficult. For these reasons, the number of the guide dog is far less than the actual needs. Therefore, we hope to use the knowledge we learn to design a guide robot to help the blind and solve this dilemma.

### **Introduction**

This guide robot can lead the owner by using a built-in map to find the target, avoid the obstacles in the process no matter they are in map or not and prompt the owner. The owner can set the target point by voice and also give other commands by voice to control the robot and get the feedback. And by this way the robot can lead the blind to the target with avoiding the static or moving obstacles in the way.

## **Timetable and Package list**

Our project started from the October 2017 and ended at April 2018.

General timetable is listed as following:

RosAria: Oct. 2017

RosAria Guest: Oct. 2017

Gmapping with matlab: Nov. 2017

Urg\_node: Feb. 2018

SLAM\_Gmapping: Feb. 2018

p3dx\_Navigation: Feb. 2018

pioneer\_bringup (Delated)

p3dx\_model: Mar. 2018

hunter (Own build): Mar. 2018

hunter\_navigation\_goal (Own build): Mar. 2018

iai\_Kinect2: Mar. 2018

libfreenet: Mar. 2018

pocketphoenix: Mar. 2018

rbx1: Apr. 2018

sound\_play: Apr. 2018

NiTE-Linux-x64-2.2: Apr. 2018

## **Effort Division**

Students Name	Main working section	Percentage of Job Effort
Minguan Shen	Software	25%
Yuchen Luo	Software	25%
Hao Lan	Software	25%
Diwen Miao	Hardware	25%
Wuxin Shen	Hardware	25%

### **Function Achievement**

- Mapping (Hokuyo)
- Path planning (Local/Global)
- Obstacle avoiding (Expected/Unexpected)
- Goals Sending (Relative/Absolute)
- Voice Command Recognition
- Sound Feedback

### **Function Test**

- Mapping(Kinect)
- Object tracking
- Skeleton tracking

## **Hardware Section**

### **Overlook of the Robot**



Figure-1 Hardware Overlook

### **Hardware We Used and Some Specification**

- Pioneer 3-dx robot
  - Voltage: 12 V
- HOKUYO UST-10LX Lidar
  - Voltage: 12 V
  - Range: 10 m, 270 degrees
- Kinect V2
  - Voltage: 12V
  - Range: 0.8-3 m, 100 degrees
- Intel NUC
  - Voltage: 19 V
- WR-DC\_to\_DC-USB200 converter
- Earphone
- Guide Stick

## **Hardware Logic**

As shown in the figure 2, the only power supply of the robot is the 12V DC source, batteries. Kinect and the Hokuyo require a 12V DC power supply so that we can directly connect them to the batteries. We will talk the details later. The NUC require a 19V DC source. So that is why we need a DC to DC converter here.

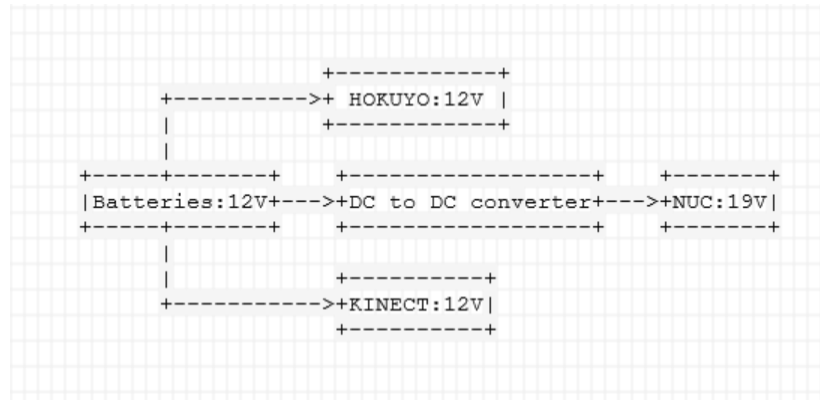


Figure-2 Flow Chart of the Power Supply

### 3D Printing Modules

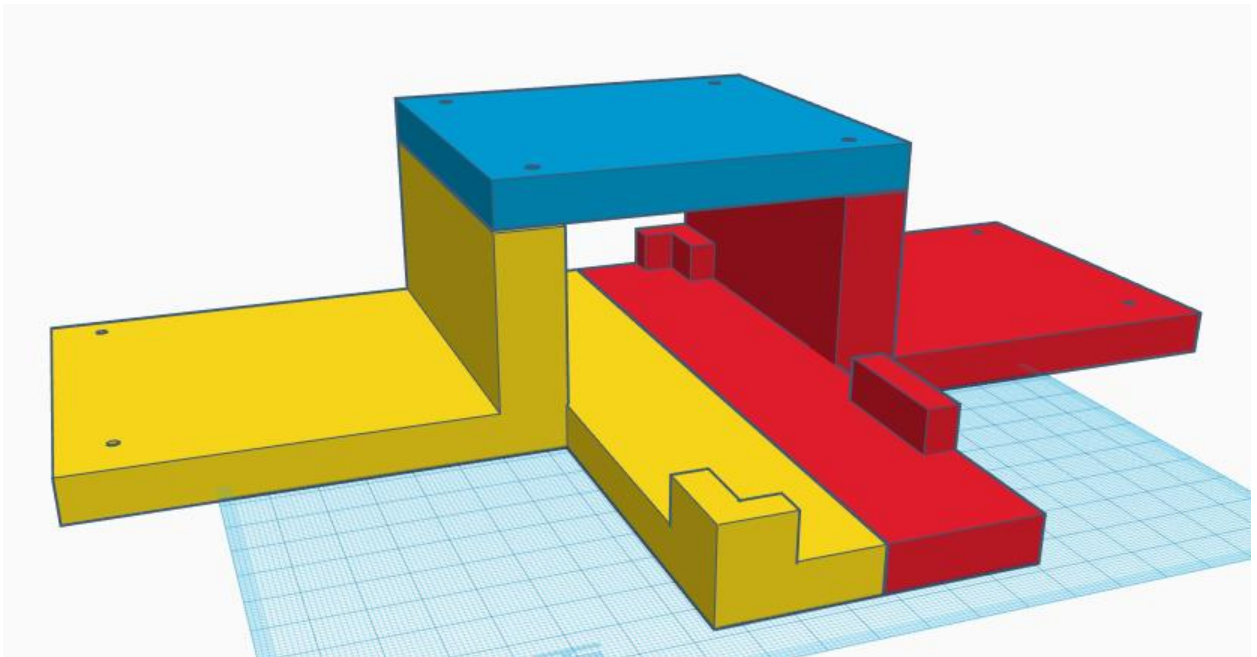


Figure-3 Module for Fixing the Converter and NUC



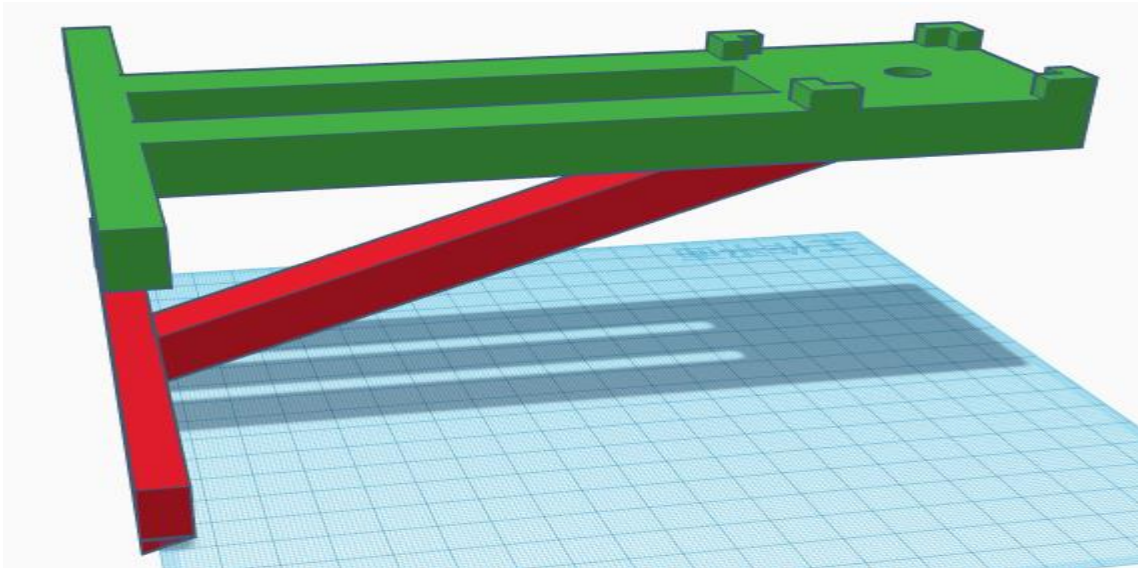


Figure-4 Module for Fixing the HOKUYO

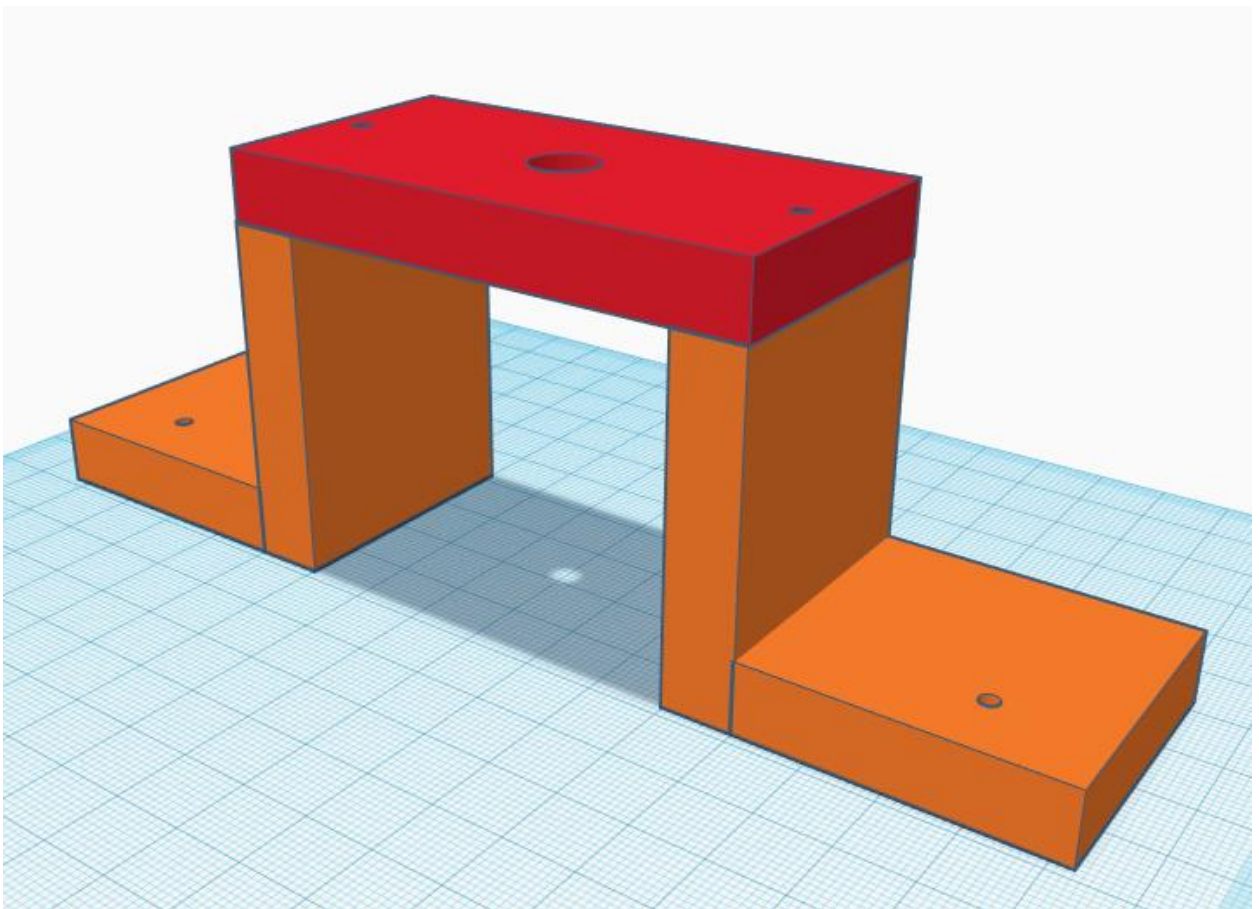


Figure-5 Module for Fixing Kinect

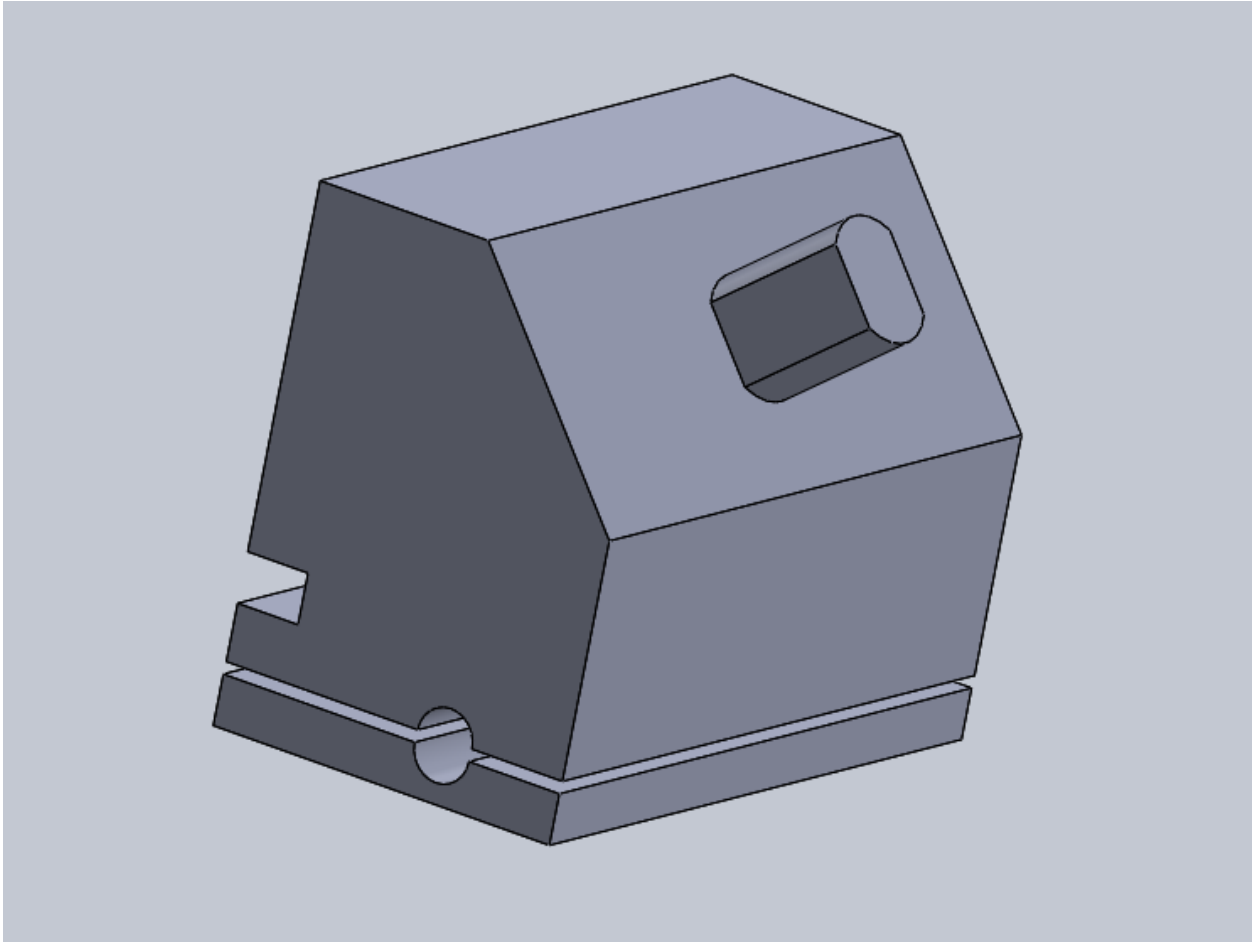


Figure-6 Module for Fixing the Holder

## **Power Supply**

### **1. DC-DC USB power supply**

The DC-DC USB is a small but powerful DC-DC powerful DC-DC power supply designed to power a wide variety of devices.

This converter can help us to connect Hokuyo and Pioneer 3. The battery of Pioneer 3 can supply 12V voltage to all devices; however, the voltage requirement of NUC is 19V, so we cannot just connect them together. Therefore, DC-DC converter is very important. This unit has a wide input

range (6-34V) and it can provide a tightly regulated output ranging from 6 to 24V (default set to 12V).

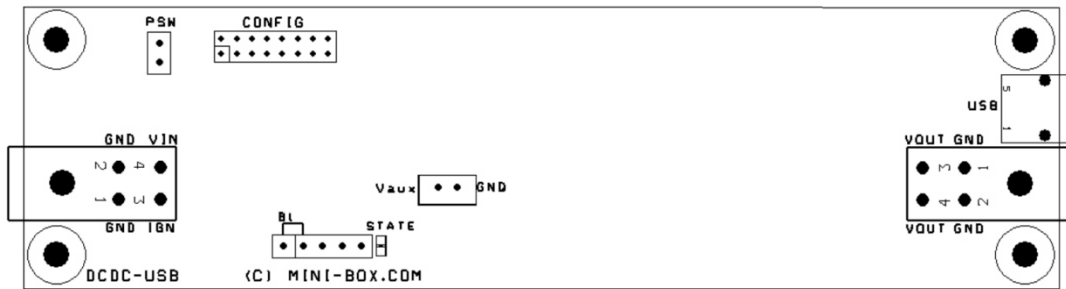


Figure-7 DC-DC-USB layout

To achieve the target, we choose jumper to do this work which is a configuration header is the most important header in this board. It is divided in 3 sections:

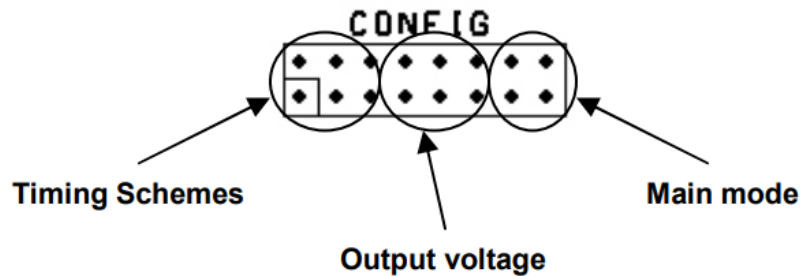


Figure-8 Headers

By default, the DC-DC USB module provides regulated 12 V output. If we want other voltage levels, we can change output voltages by setting jumpers 2,3 and 4. Jumper setting form is as following:

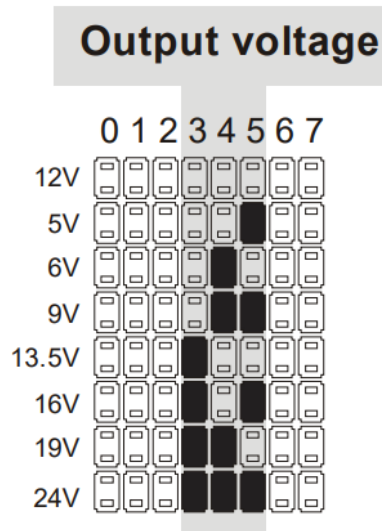


Figure-9 Output Configuration

In this way, the battery of Pioneer 3 is finally able to provide power to NUC.

**There are two disadvantages:**

- a. The empty space inside the robot is obviously not enough to contain converter and NUC. As a result, I decided to put converter inside the robot.
- b. The temperature near the battery of robot is so high that may damage the converter, so I fixed the converter on the corner.

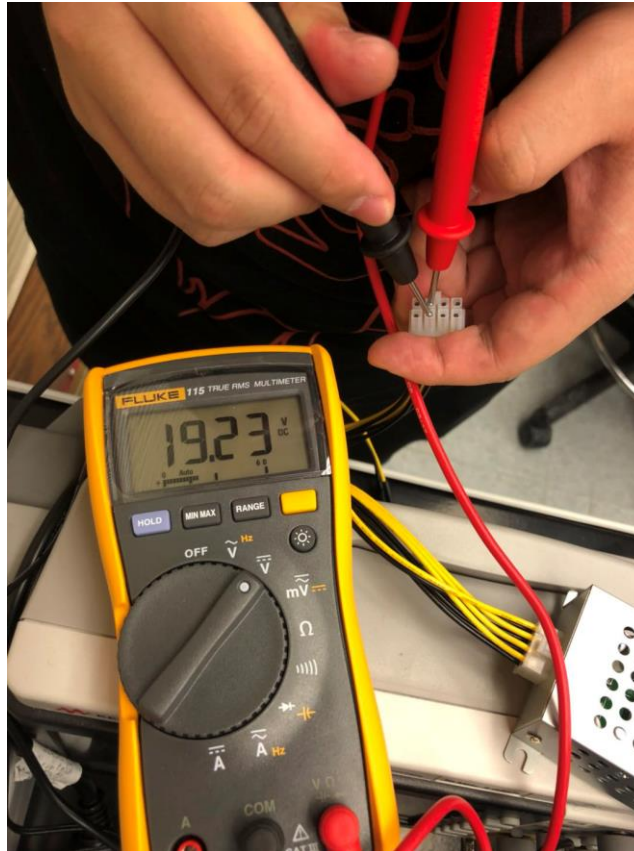


Figure-10 Checking the Output Voltage

## 2. Connect the DC-DC converter to pioneer

The input port of DC-DC converter should connect two cables: one is 12V, the other one is GROUND. After searching, we found that the red cable is 12V and the black one is GROUND. We cut the cable then reveal the copper wire. Then shape the copper wire to a circle and use welding gun to reinforce the circle. After the preparatory work, we can set the screw spike through this circle, then set the screw spike on battery board.

Two screw spikes set on 12V and GROUND respectively (The first right and the second left), and make sure the red cable(12V) connect to the 12V of battery, the black cable (GROUND) connect to the GROUND of battery.



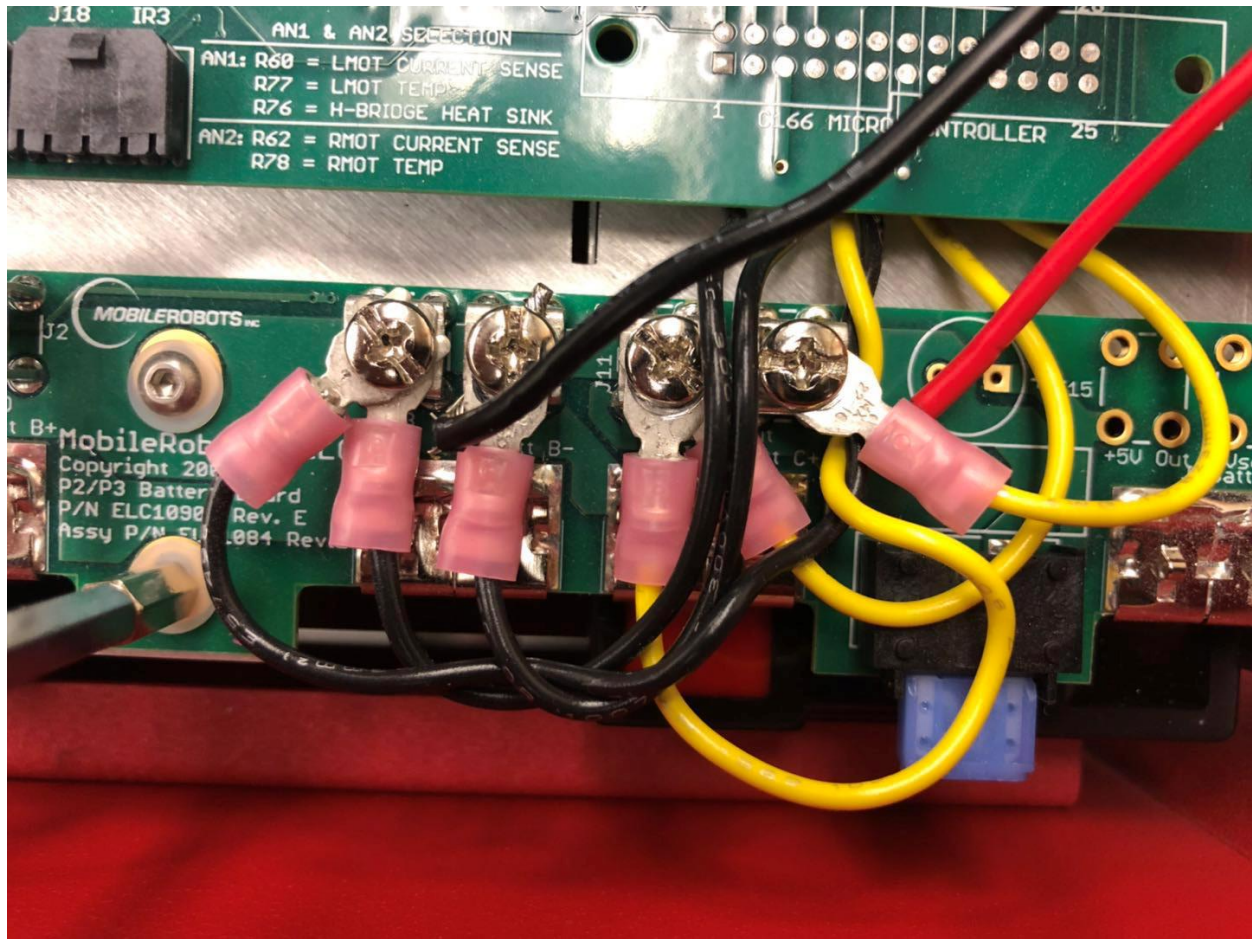


Figure-11 The Connection between Pioneer and Converter

### 3. Connect Hokuyo and Kinect to Pioneer 3 and computer

The model number of our Hokuyo is UST-10LX, whose power source is 12V or 24V, so we don't need to use DC-DC converter again. However, we still need to reform the cable of Hokuyo because the pioneer doesn't have a suitable port to connect Hokuyo, and supply voltage.

To achieve this target, firstly we should figure out the color of each lead wire, signal name and function.

Just as shown on table-1.

Color	Signal	Function	Description	AWG
Red	COM Input+	Input	COM Input+	28
Gray	COM Output-	Input	COM Output-	28
Light blue	Input	Input	IP reset input	28
Orange	Output	Output	Synchronous Output	28
Brown	+Vin	Power	Power supply: DC 12V/24V	28
Blue	-Vin	Power	Power supply: 0V	28

Table-1 Information of six wires

As we can see, to use battery of Pioneer to supply power, we should use Brown wire and Blue wire.

But there is a new question: the lead wire of Hokuyo is so slender that any violent action may destroy it. As a result, we soldered the lead wire and a stronger wire together, then connect the wire and Pioneer using AUX PWR (Auxiliary Power Outlet)

In conclusion, this is the method of connection:

Brown wire of Hokuyo (Kinect)-->Red strong wire-->12V of AUX PWR (optional equip)

Blue wire of Hokuyo (Kinect)-->Black strong wire-->GND of AUX PWR

Finally, the Pioneer can provide Hokuyo power. Then we connect the other port to NUC via Ethernet port.

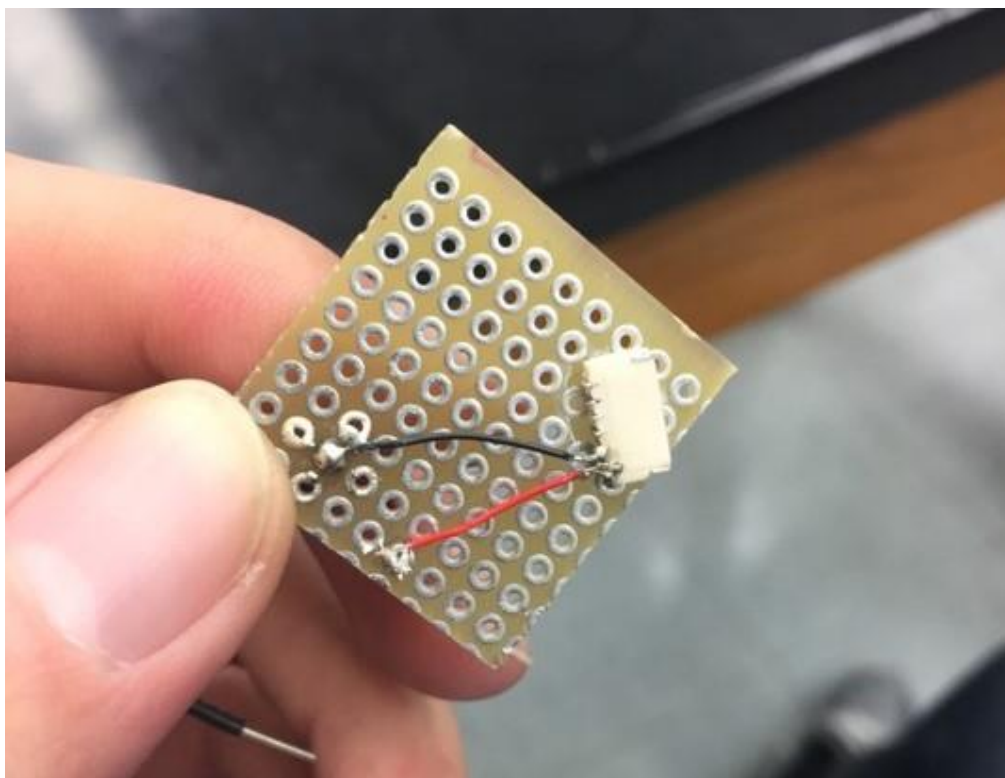


Figure-12 Solder the Connector

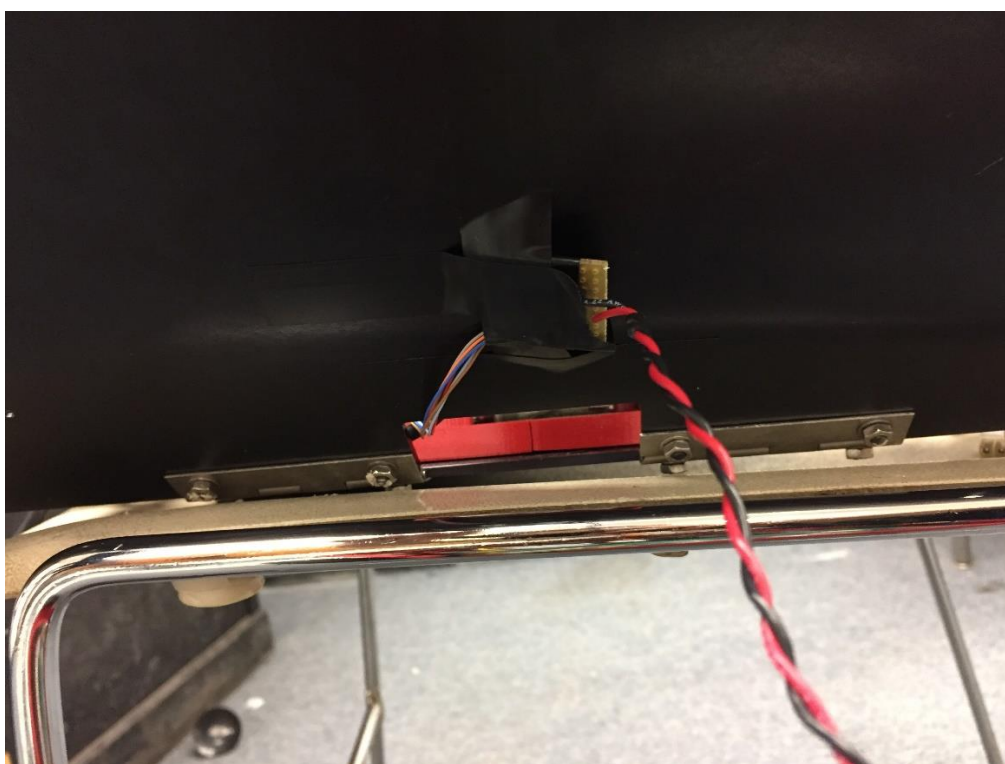


Figure-13 Stick Connector to the Top Plate



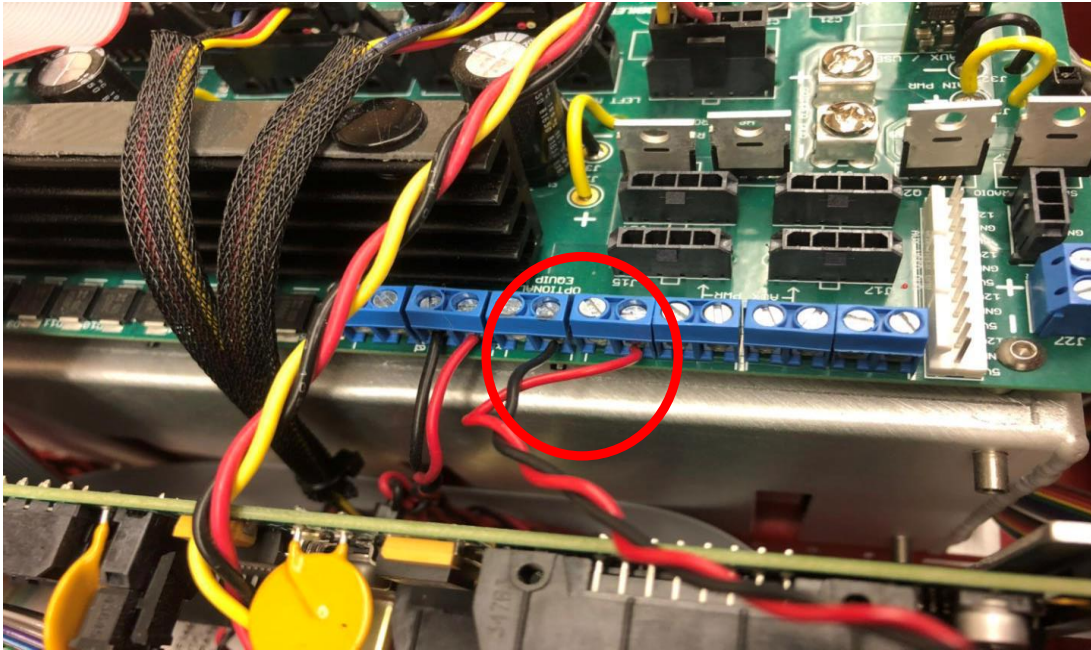


Figure-14 HOKUYO Connector Connection

The connection of the Hokuyo and the Kinect is similar. We cut the head of the joint of the Hokuyo (Kinect) and welding them with the batteries directly.

The Kinect wire structure is shown below. Since there is no accurate information about it, we used a multimeter to find out.

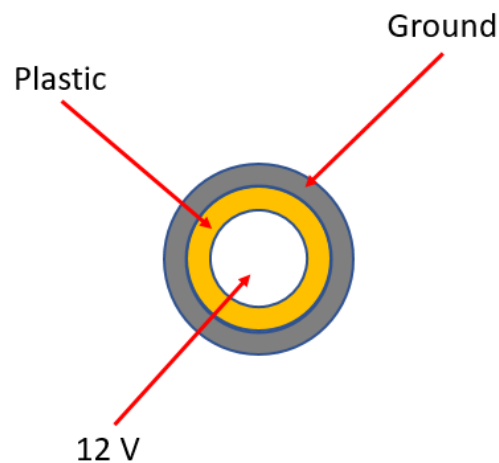


Figure-13 Cable Structure

We collected wires, stuck them together, and insert them into the correct power supply position on the Pioneer board.

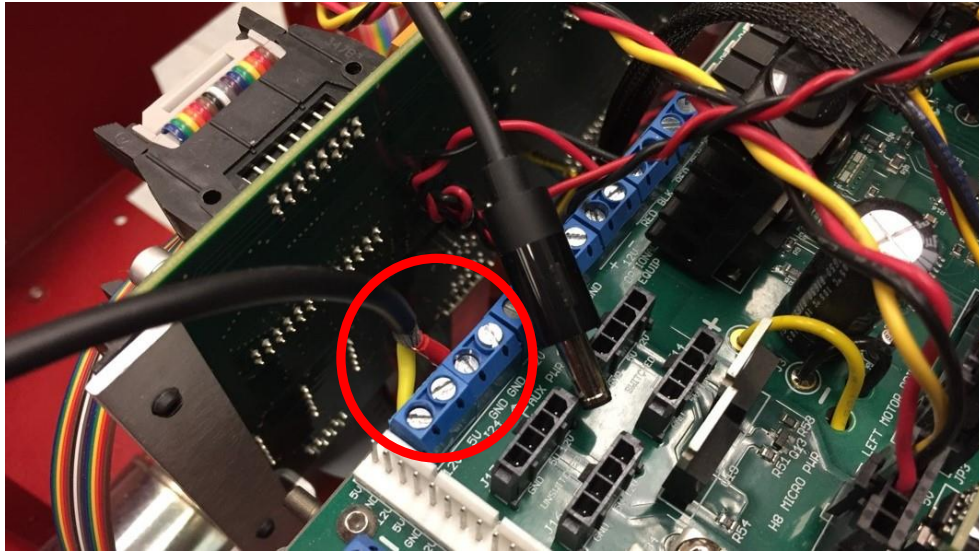


Figure-15 Kinect Power Supply Connection

## Holder

We bought a walking stick as the basic part of our handle. In addition, we need the stick to be able to change direction for the users' convenience. So we combined the stick with a u joint, connecting them with screws on the u joint.



Figure-16 The Hole on the Stick & the U-joint



Figure-17 Combined Holder

We also designed a 3D module to fix the stick on the Pioneer. The 3D module is shown in the “3D Printing Module” above. The way of connection is shown below.

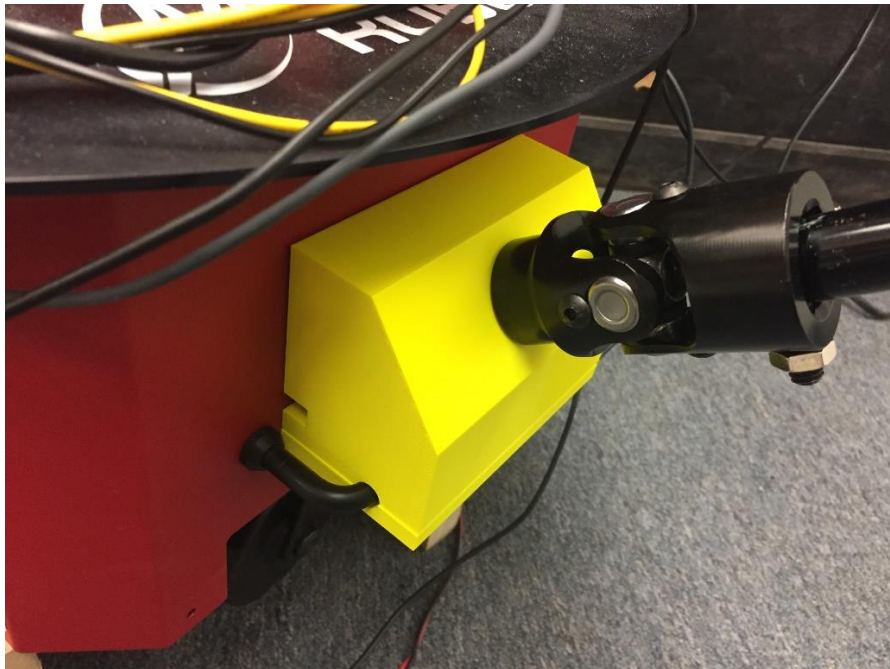


Figure-18 3D Module Connection

## **Hardware Manual**

### **1. 3D Printing Modules**

All 3D printing modules have obvious screw stabilization. you can install or uninstall those modules by simply screwing them on or off.

All the screw positions can be seen in the modules pictures above.

### **2. Cable Connections**

All the cable connections are shown in the picture clearly with some indications. You can find them inside the Pioneer boards. Use a screwdriver to loosen screws and connect or disconnect those cables.

### **3. Holder**

The 3D module connection of the holder is fixed on the Pioneer by 4 screws on the bottom of the 3D model. It can be screwed off with a screwdriver.

The u-joint is inserted into the corresponding part of the 3D model very tightly. It is stabilized by itself. Disconnecting them may need a powerful tool to smash the 3D model from the u-joint side.

The stick and u-joint are connected by screws. A local screw on the u-joint is screwed on the hole drilled at the end of the stick.

Another screw is used to stabilize the stick. Screw off those screws to disconnect the stick and u-joint.

## Software Section

All of our project is based on the Ubuntu operating system which is a Linux distribution based on the Debian architecture. And the software program we wrote to control the robot is under the ROS (Robot Operating System). ROS is a flexible framework for writing robot software which is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust behavior across a wide variety of robotic platforms.

The navigation work can be generally divided into two parts, global and local which depends on the source of the information. It generally includes mapping, path planning, localization, obstacle avoiding, etc.

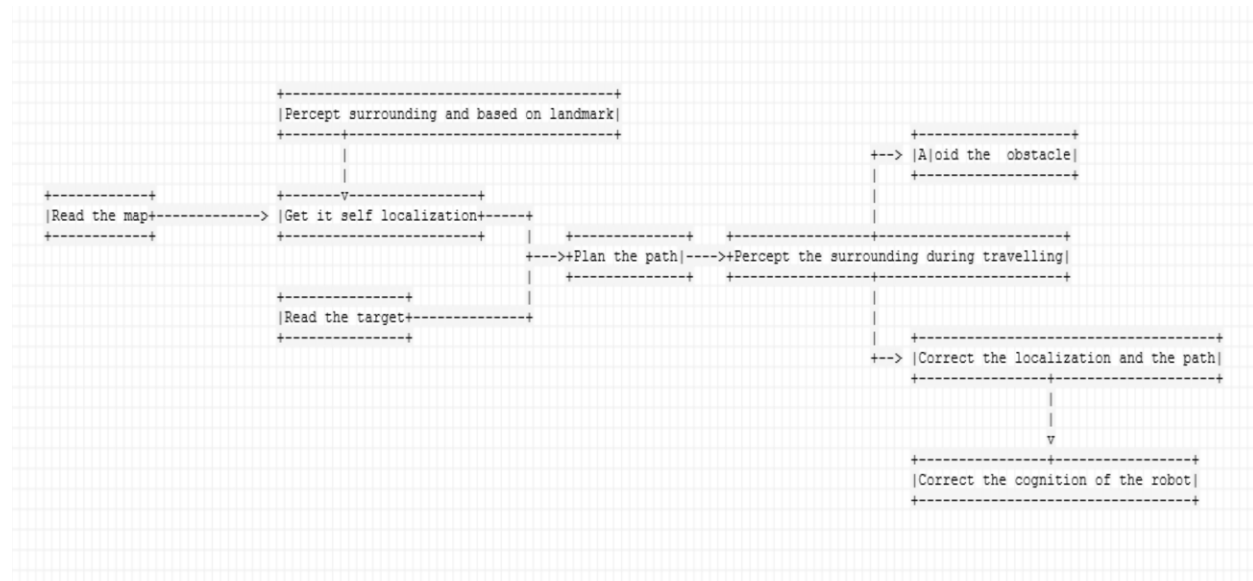


Figure-19 General Logic of Navigation Software Design

- **ROS Interface**

The ROS interface we used to control the robot is the RosAria. It's a node provides a ROS interface for most mobile robots which supported by Adept



Mobile Robot's open source ARIA library. Information from the robot base, and velocity and acceleration control, is implemented via a RosAria node, which publishes topics providing data received from the robot's embedded controller by ARIA, and sets desired velocity, acceleration and other commands in ARIA when new commands are received from command topics. It is the most basic node for the mobile robot. And at any time when we hope to connect the robot with the speed or odom data, we will need it.

The algorithms we used are generally as following:

- **Mapping: Gmapping (SLAM)**

In robotic mapping and navigation, simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. While this initially appears to be a chicken-and-egg problem there are several algorithms known for solving it, at least approximately, in tractable time for certain environments. Popular approximate solution methods include the particle filter, extended Kalman filter, and GraphSLAM. SLAM algorithms are tailored to the available resources, hence not aimed at perfection, but at operational compliance. However, it is the most popular method used in the robotics area. The data source can be either lidar or camera. So, we test both the HOKUYO and Kinect for building map. If we just hope to build a 2-D map, HOKUYO is a better choice. Because the max range for the Kinect is only about 2 meters and it is greatly shorter than the Hokuyo.

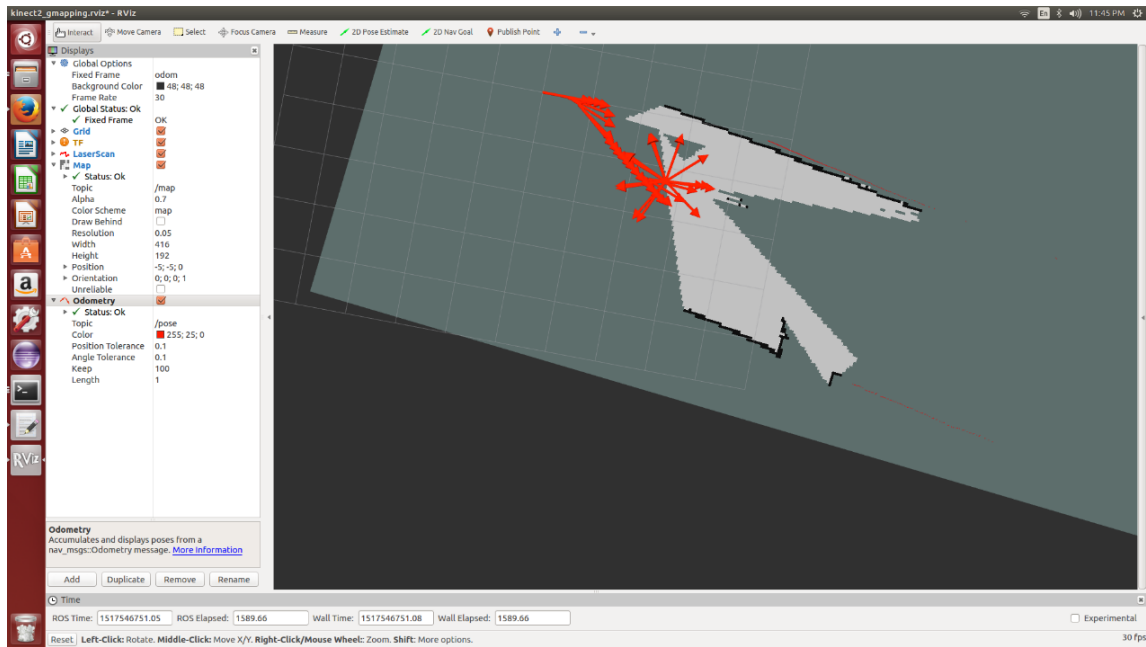


Figure-20 Mapping Process

- **Localization: Landmark and Probability (SLAM)**

Because the indoor robot can't use the GPS for localization, without other device, the best way we thought to achieve it is the landmark and the probability. As I mentioned above, though it is not perfect, it is the best choice now. The ROS itself has a package called AMCL which is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map. With the updating of the Odom data, it can work accurate enough. However, the initial localization is still the problem. In my test, without the strict limitation of the testing environment, ex. big clear space, obvious obstacles and so on, it is really hard for the robot get the correct initial position. Flowing command is used for the robot reset its localization.

\$ rosservice call /global\_localization

And related parameter such as particle numbers can be found in the  
Home/catkin\_ws/src/p3dx\_navigation/launch/amcl.launch

Ideally, the more particles it has, the accurate it will be. But concern its low accuracy, I don't think it is a good idea to keep using it.

- **Path planning: A\***

A\* is an informed search algorithm which we have learned both in Senior Design and AMR course. It's a best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node.

At each iteration of its main loop, A\* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. Specifically, A\* selects the path that minimizes

$$f(n) = g(n) + h(n)$$

Where  $n$  is the last node on the path,  $g(n)$  is the cost of the path from the start node to  $n$ , and  $h(n)$  is a heuristic that estimates the cost of the cheapest path from  $n$  to the goal. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.



- **Obstacle Avoiding: Inflation**

Based on A\*, obstacles avoiding used the inflation method.

As we learned, the inflation should be concerned with the radius of the robot.

And it can change from the

Home/catkin\_ws/src/p3dx\_navigation/config/p3dx\_rosaria/costmap\_common\_params.yaml

We use 0.4 here, to make sure that there is enough space for robot avoid the obstacle and can go through most of gap.

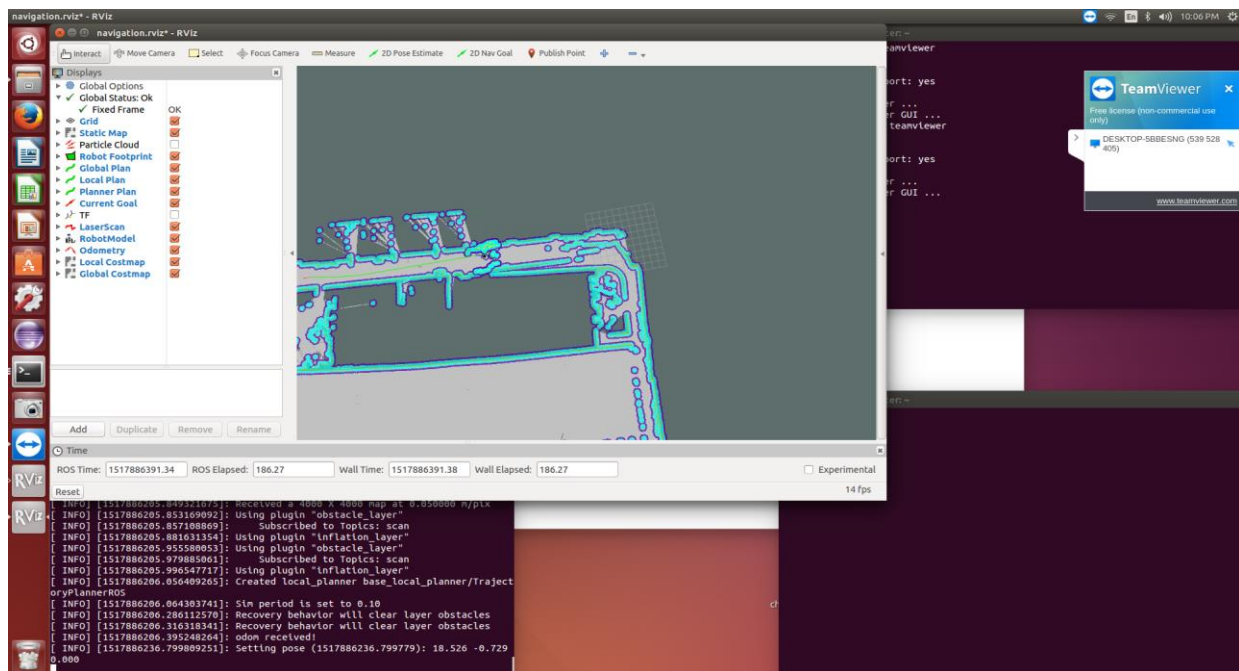


Figure-21 Blue Areas are the Inflation and the Green Line is the Planned Path

The following figure gives a clearly introduction of the nodes that a robot need to finish a navigation work. The sensor source can be either HOKUYO or KINECT in our project. And the base controller actually controls the robot's

movement. The transfer frame of our robot is in figure 11. Match these two figures can help us check our work.

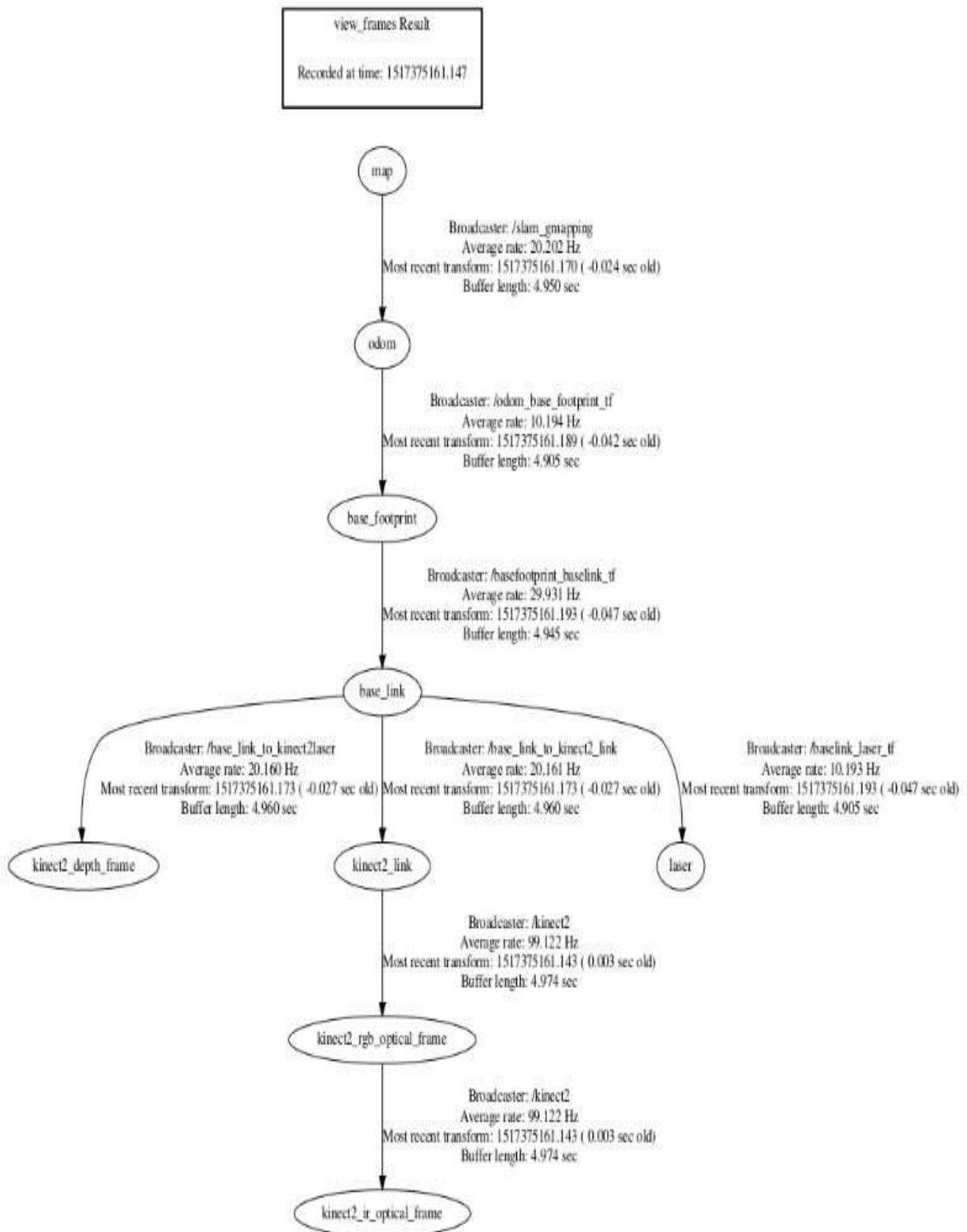
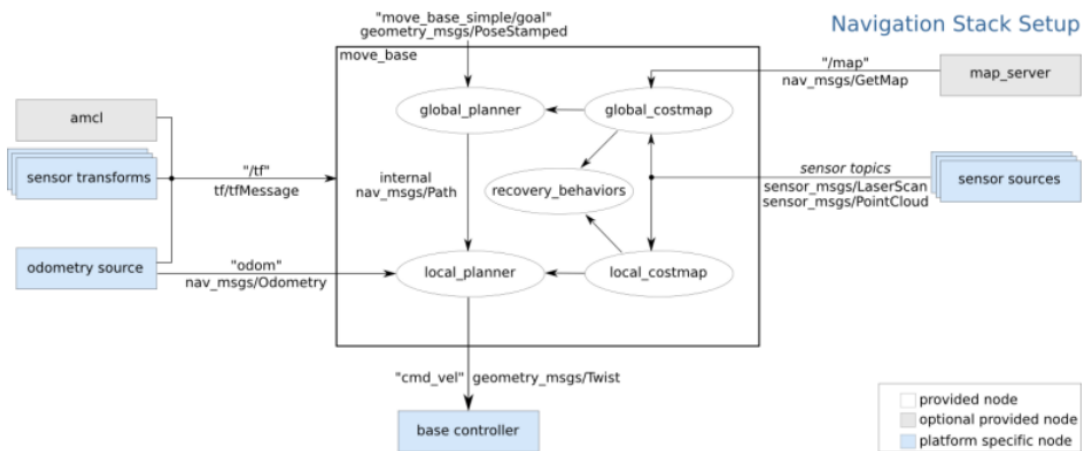


Figure-22 Data Transfer Frame Tree



The navigation stack assumes that the robot is configured in a particular manner in order to run. The diagram above shows

Figure-23 Logic Between Nodes of Navigation Part

## Simulation

We are using RViz for 2-D simulation now. Frame and robot model have been added into the simulation and related parameter have been set. So, we can test the robot in the RVIZ before we actually run it on the ground to keep safe and save time.

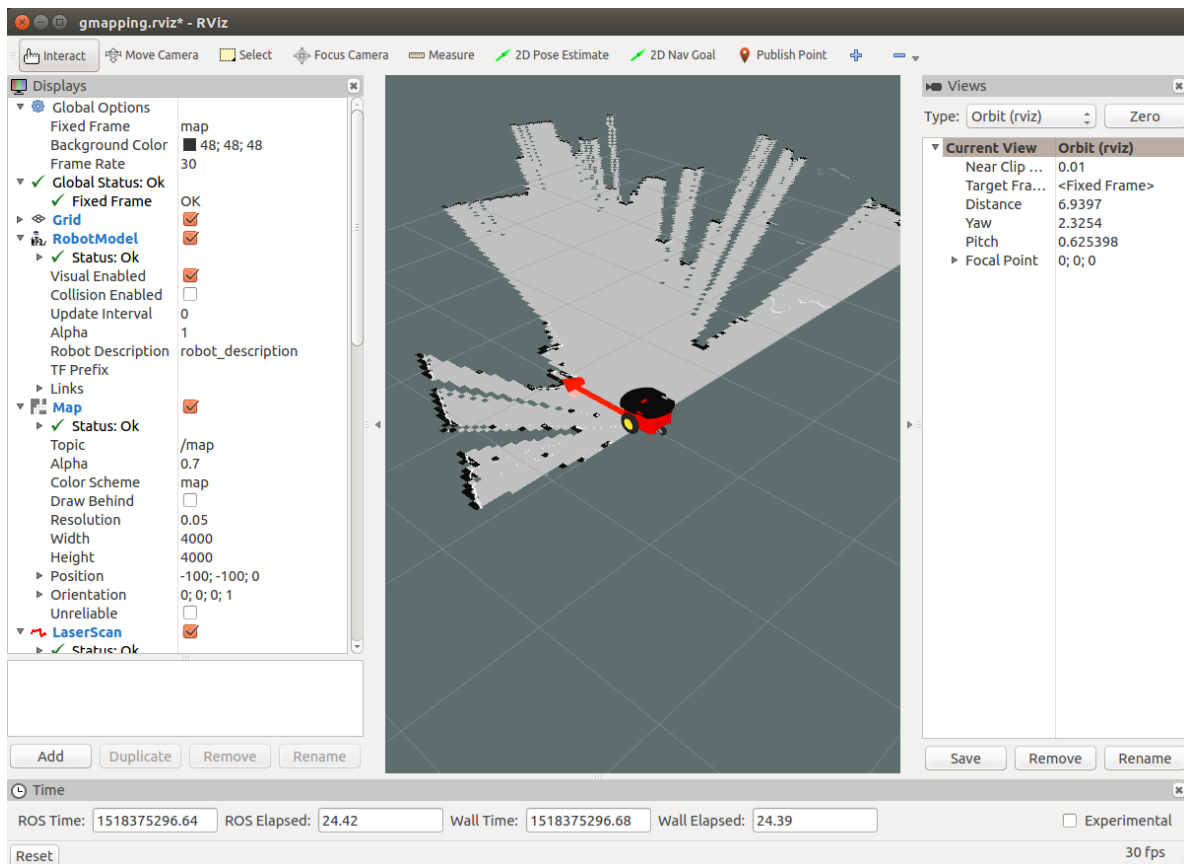


Figure-24 Simulation of Mapping

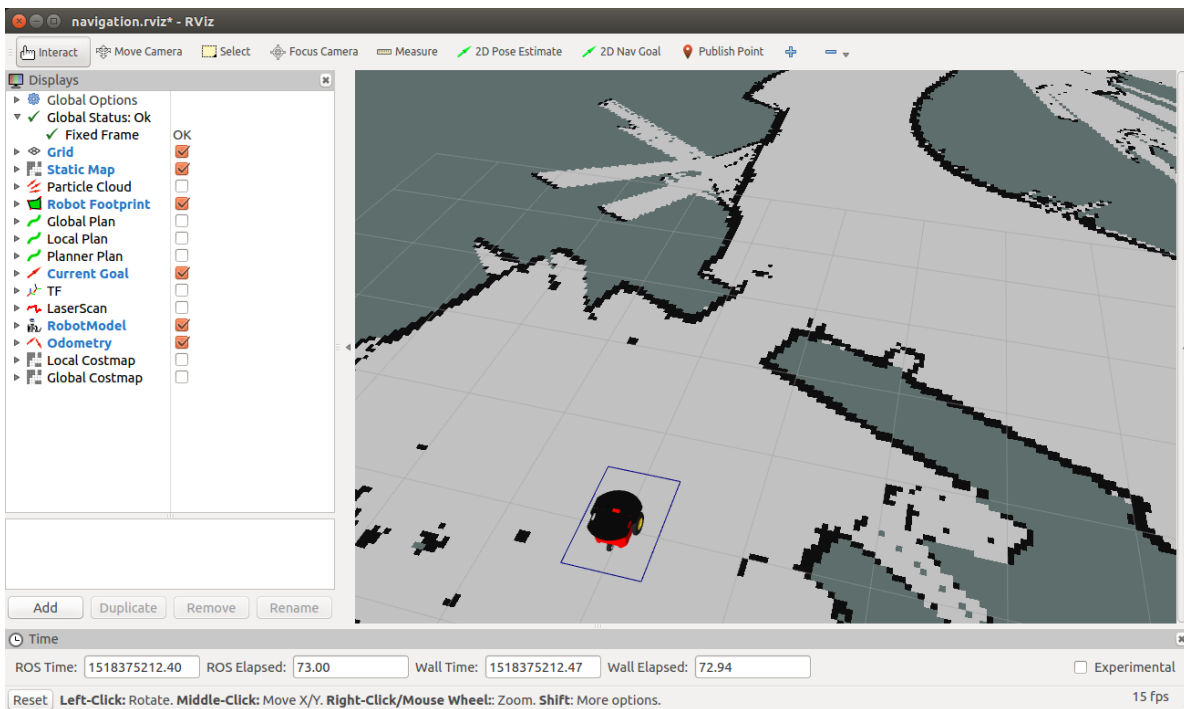


Figure-25 Simulation of Navigation

## **Goal Sending**

Instead of the original way which need us to click to set the goal, we hope to use a more accurate way, based on the coordinates. As what we have contacted in the mapping part, the robot has its own local frame named as the 'base\_link'. The origin of that coordinates will always be on the center of the robot and the axis are shown as the following figure.

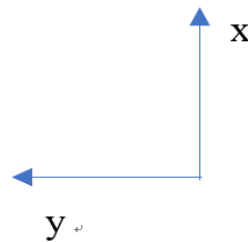


Figure-26 Axis and their direction

With the local frame we can send the relative position to the robot as the goal based on the move\_base. However, most of time what we need the robot to go is the position in the real world. So, we will need a global frame to send the absolute position to robot which named as the 'map'. The map frame was build when we started mapping. And the origin is the start position of the mapping. The axis is same as the moment of starting mapping's robot local frame's axis.

## **Voice Recognition**

For the blind ones, the best way to connect with the robot should be the voice. For our design, voice recognition is the core. Because the NUC doesn't have the

microphone and it is really hard to find an earphone with the microphone and has the drive on the Linux system. So, we use the Kinect as the voice input device.

We use an important package here called ‘pocketphinx’ which is one of the most popular one in this area. You can choose the language model that which kind of language you hope and download from the following website and it allow us to build our own library:

<https://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/>

And also, a Chinese company called Xunfei has the similar technology. We can build the voice library by either of them and then we can call the function that recognize the voice we record in the library. The limit is that to make the recognition’s accuracy, we would better to use the word with about three or four syllables. And as the same time, it will be influenced by the accent.

With the technology of the voice recognition, we can use the voice command as the input to control the robot. Programming logic is as following:

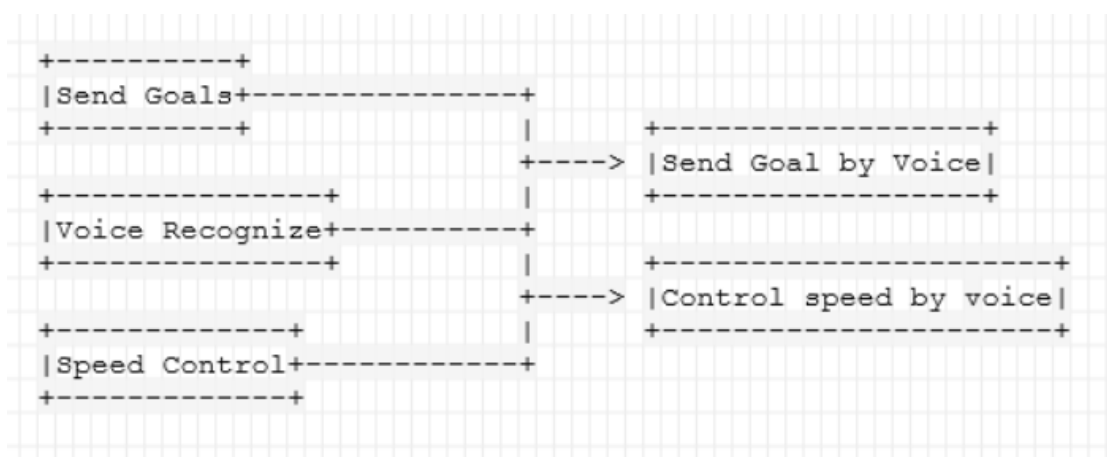


Figure-27 Function with voice command

## Voice Feedback

As a blind people, without seeing, how can he now the robot gets the correct command or not? So, the feedback is important and it can't be just print on the screen. For a ubuntu system and the ROS, the best choice to let it speak is using the package 'sound\_play'. It allows you play the sound you have recorded before or read the certain characters. Opposite to the recognition, sound\_play gives you a new choice of output. And what we need to do is connect it with the certain input commands.

## Image processing

To fully use the Kinect, we also work on the image processing. Depth detecting, color recognition and skeleton tracking are known as the three most important function of the Kinect. With downloading the drives and the related libraries, we successfully get the depth data and the color data and tracking the skeleton.

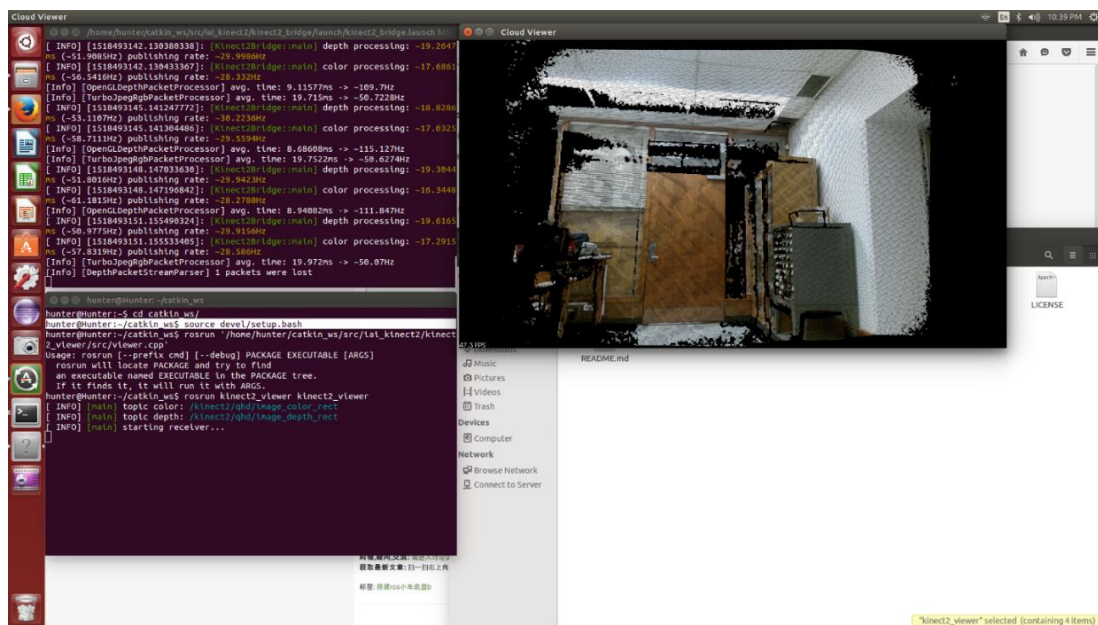


Figure-28 Reading Depth and Color Data from the Kinect



With the data of these, and the tutorial in the RBX1, we worked out the object\_tracker node which allows us to use the mouse to choose an obstacle and let the robot move follow it. However, because we have to use the Kinect to collect the voice and it only has about 100 degrees detecting area. We have to let the Kinect face to the people who is following the robot. Concerning this, we abandon this function and start to work on the skeleton tracking.

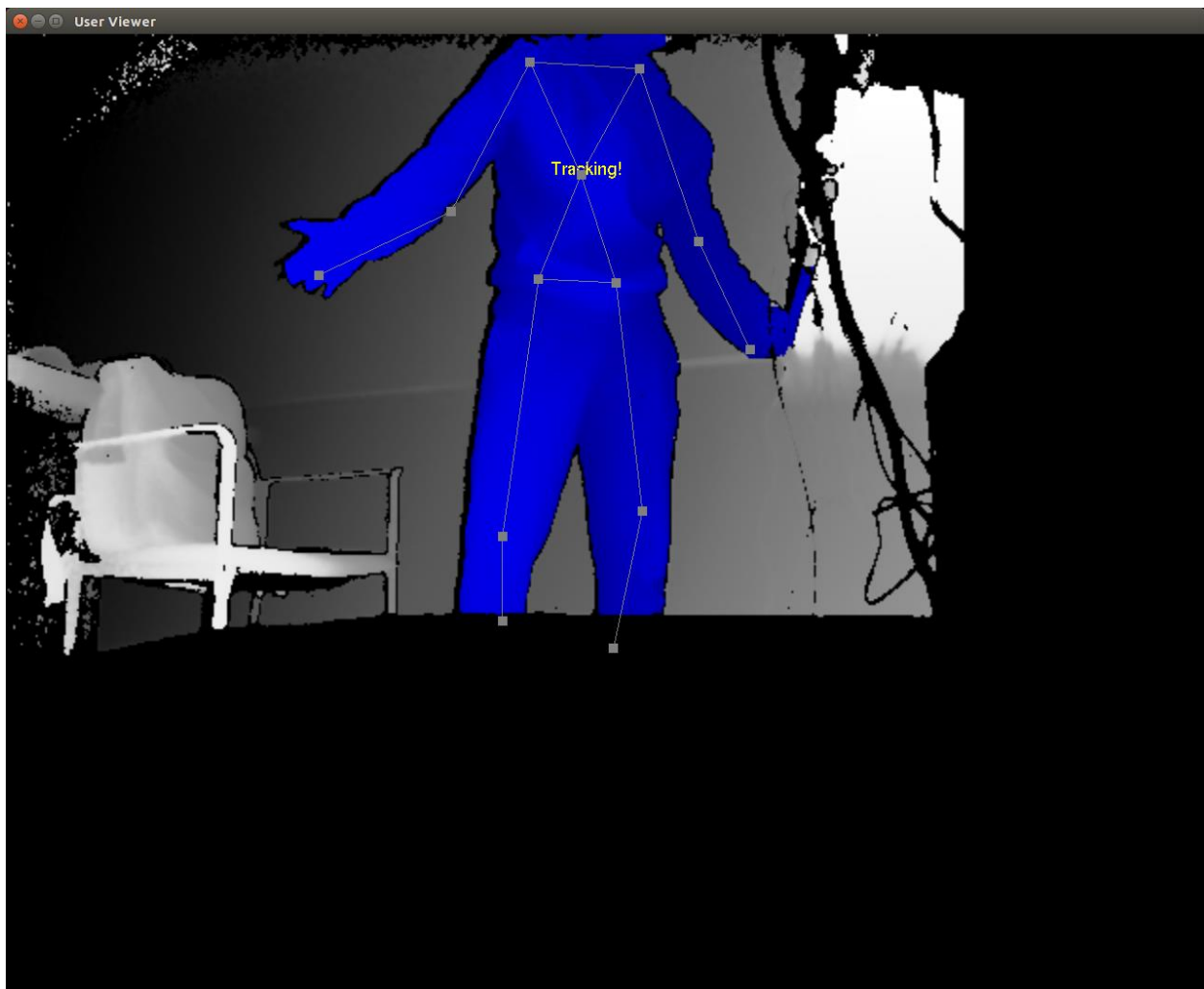


Figure-29 Skeleton Tracker

The problem of this part is, most of the algorithm and method here to deal the skeleton data to recognize the gesture is base on the Windows SDK. The source that can work on the Ubuntu with ROS is really rare. And I have to admit that writing an algorithm of that in the such limited time is beyond my ability. So we didn't finish this part with the pity.

### **Trouble Shooting**

Most of the problems we met are caused by unfamiliar with the ROS. So, the further we go, the lees problem we met. Because the process of solving problem is the best way of learning. For the concerning of being professional, I just list some technical problem here.

- **Map-updating error**

When we use the Pioneer \_bringup package which suggested by the TA, hope to use the Hokuyo lidar to do the gmapping, it's very possible that get in trouble with map updating. The typical wrong map is like following:

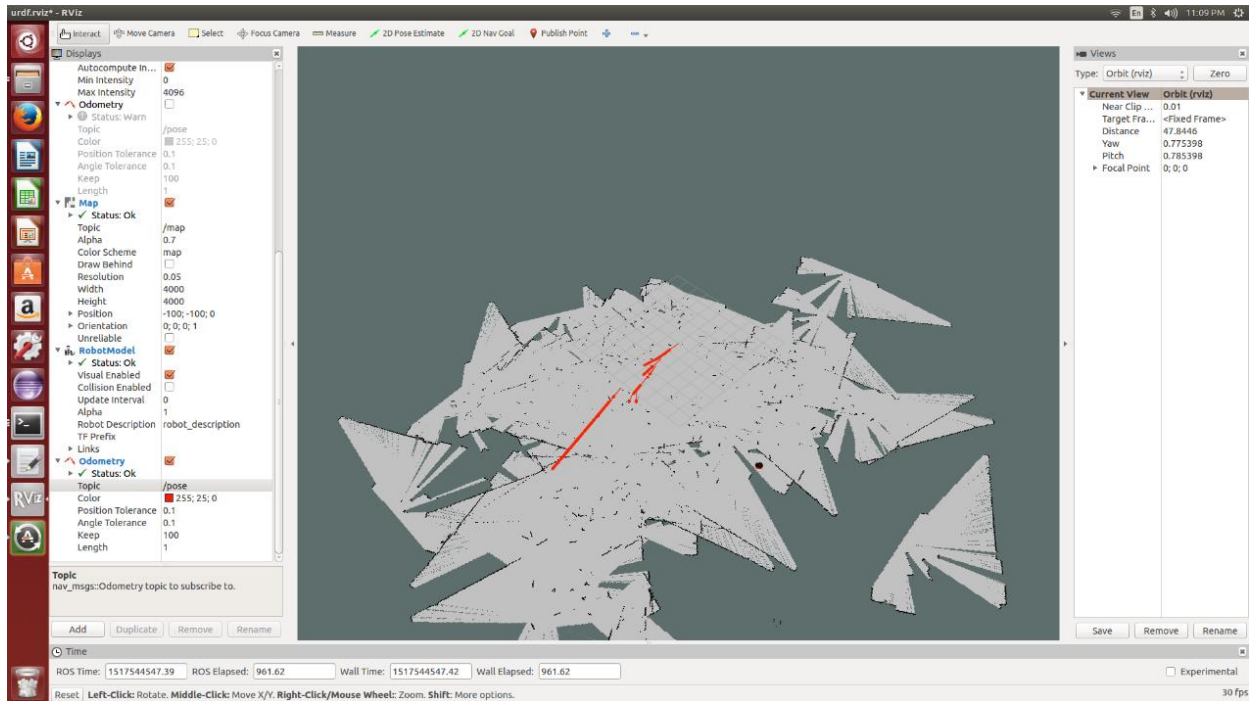


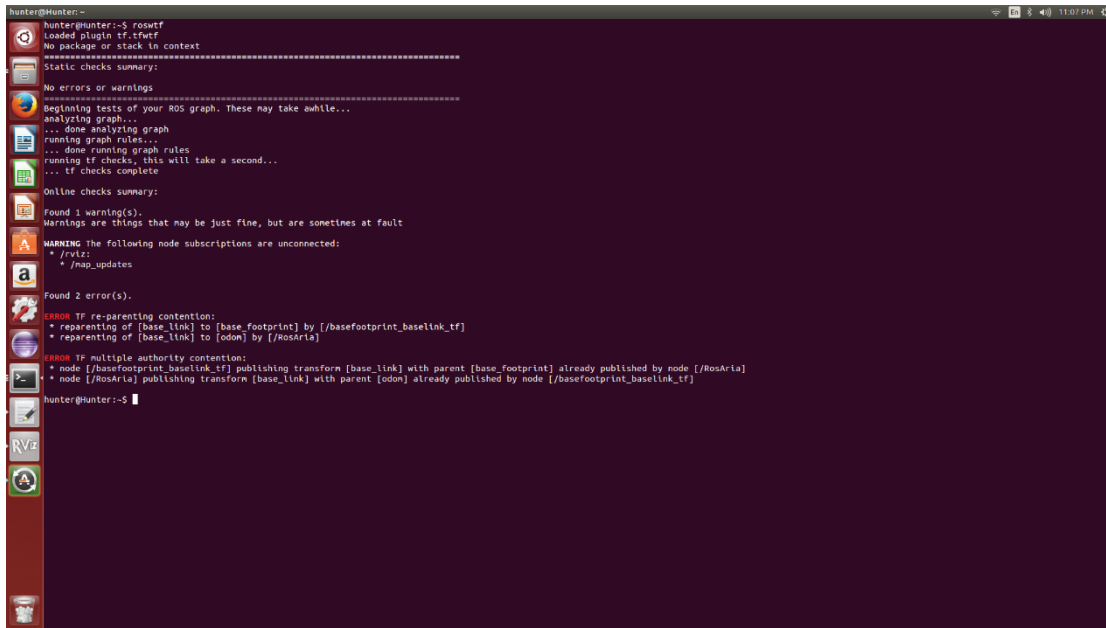
Figure-30 Typical Wrong Map

It's easy to find that there is something wrong with the odom's data transfer. If you open the tf and the robot model in the RVIZ you may find that the frame is shifting erratically. But not keep with the robot at all.

And there is an important command here we use to find out where we can solve this is:

`$ rosrtf`

By this command, you can let the ROS check itself to find if there anything wrong. Especially the message transfer and the relation with the publisher and subscriber.



```
hunter@Hunter:~$ rosrun tf_tutorial tf_tutorial
loaded plugin tf_tutorial
No package or stack in context
=====
Static checks summary:
=====
No errors or warnings
=====
Beginning tests of your ROS graph. These may take awhile...
analyzing graph...
... done analyzing graph
running graph rules...
... done running graph rules
running tf checks, this will take a second...
... tf checks complete

Online checks summary:
=====
Found 1 warning(s).
Warnings are things that may be just fine, but are sometimes at fault
WARNING: The following node subscriptions are unconnected:
* /rviz;
* /map_updates

Found 2 error(s).
=====
ERROR: TF re-parenting contention:
* reparenting of [base_link] to [base_footprint] by [/basefootprint_base_link_tf]
* reparenting of [base_link] to [odom] by [/RosAria]

ERROR: TF multiple authority contention:
* node [/basefootprint_base_link_tf] publishing transform [base_link] with parent [base_footprint] already published by node [/RosAria]
* node [/RosAria] publishing transform [base_link] with parent [odom] already published by node [/basefootprint_base_link_tf]

hunter@Hunter:~$
```

Figure-31 Result of Check

You may find that the problem is caused by the reparenting connection. That is why when you check the tf frame tree everything is good but map updating is wrong.

So, what we need to do now is just delete the redundant tf. You can delete either one actually, but my advice is keep the ARIA's. Because you will use this package in many places. And concerned this, the pioneer\_bringup package has no meaning. So, from here, with the confidence I accelerate during solve this problem, I built up our own package and write our own source code.

- **Wheel encoder error**

After a big test of hole map of the second floor, we found that there still some error in our wheel encoder. The error accelerates. That means the longer the distance is, the more obvious the error's influence is.

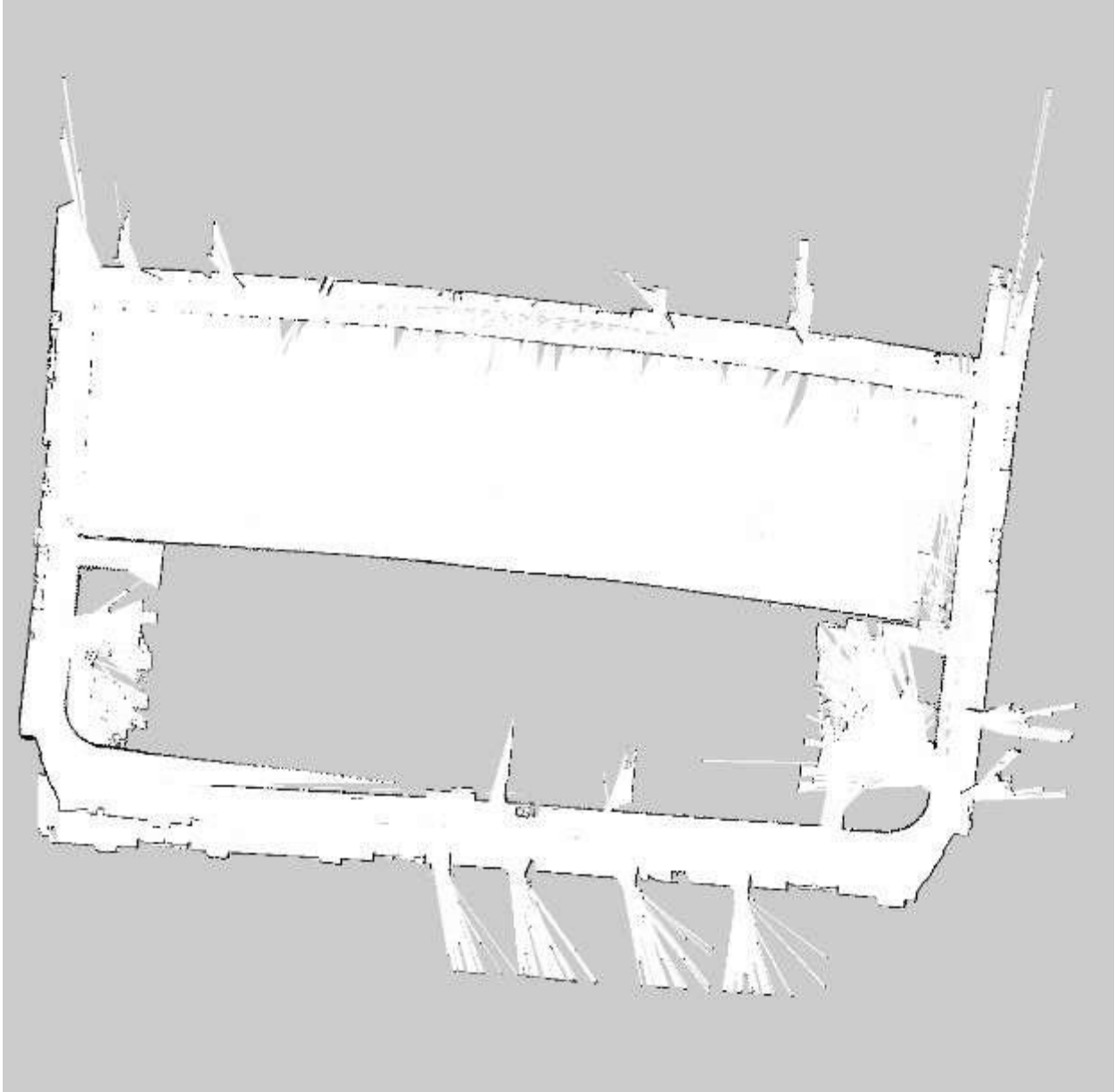


Figure-32 Testing Mapping

In this map, you may find that the right-down corner isn't overlap perfectly. That is caused by the error. The Kalman filter has already be used. But it still need to adjust and test. However, because the demo place is not so big. This small error didn't influence its performance there.

- **HOKUYO's position**

This is the problem we find during the demo test. Because our HOKUYO is too high, some lower obstacle will be almost ignored. However, if move the HOKUYO down, we have to make sure that there won't be any collision. Or the HOKUYO may not safe.

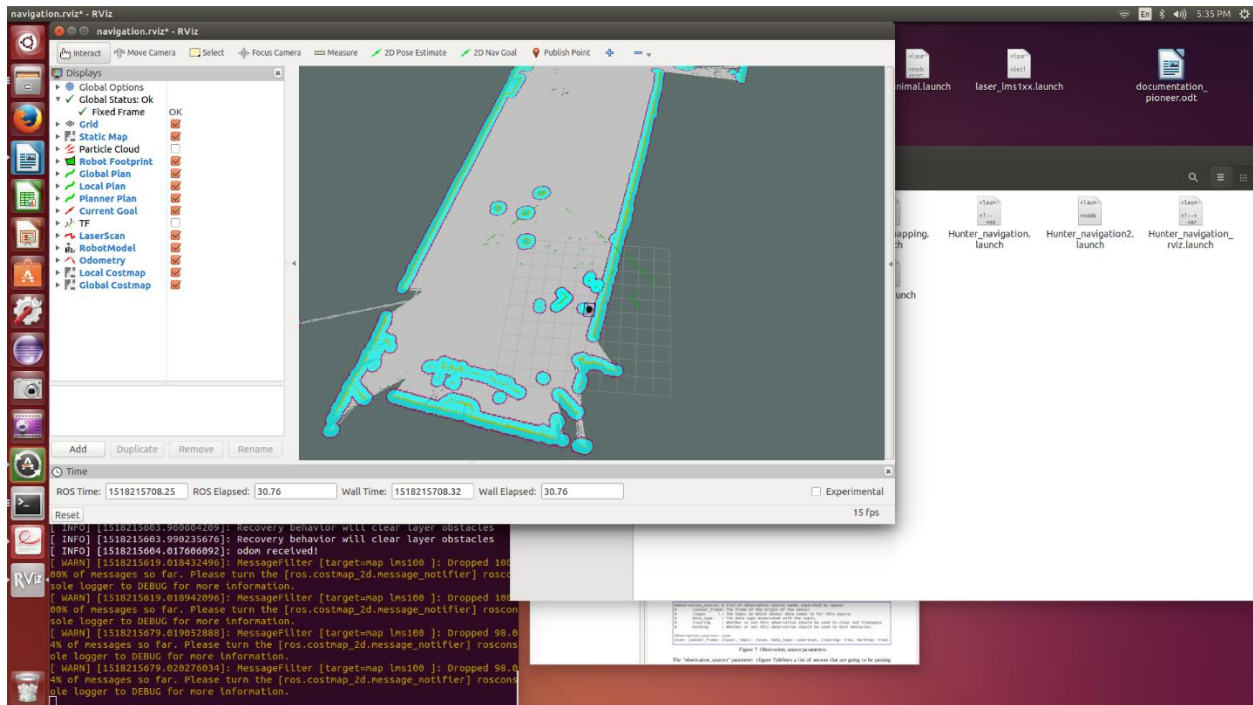


Figure-33 Demo Map, the Target Signature is Lower than the HOKUYO

- **Target's situation (Pending)**

This is also a problem I find during the demo test. If there is a gap too near the target and other situation to make the target can't or hard to be achieve. The robot will confuse. This is a really hard part for us to solve. What we can do is just try our best to improve the robot's performance in different situation during navigation. But if the environment is too severe, we can do nothing. So, the best way is limit the environment.

- **Wake-up localization problem/ Initial localization (Pending)**

Be famous as one of the three localization problem, the wake-up localization problem also known as kidnapped robot problem which refers to a situation where a robot is carried to an arbitrary location and put to operation, and the robot must localize itself without any prior knowledge.

The easiest way is set the start point of the navigation same as the start point of mapping. Or click to give it as uncertain initial localization. As I mentioned above, I am trying to use the AMCL which provided by ROS to solve this based on probabilistic localization. But the result is still not so good. The accuracy rate is only about 10% and it related to the map. The more the signatures which can be worked as landmarks are and the clearer the map is, the higher the accuracy rate will be.

- **Combine nodes with python and C++**

As I mentioned above, when we working on the programming. A lot of time what we need to do is combine the input and the output to a new program which can work on the ROS. For a project, we need to combine the nodes to make it as a whole thing. To achieve the same function, the less nodes we need, the better it is. Some can be combine in the launch file, but if we hope to change the logic of the input and the output, we need to write a new node. He problem here is, a truly big part of the source of the nodes we can find as the reference are written with Python. And none of us has contact with the python before. When we write a node, the language we will choose is obviously C++. So, what is difficult is to find out the corresponding word, the title file, the

library of the Python in the C++. However, it is really meaningful that it can simplify the codes and the commands.

## **Software Manual**

All the command need to be run in the workspace, our workspace is:

`~/catkin_ws`

And you'd better run the following command before use run the ros nodes or packages:

`. devel/setup.bash`

### 1. Gmapping:

`$ roslaunch hunter Hunter_gmapping.launch`

`$ roslaunch hunter Hunter_gmapping_rviz.launch`

Save the map:

`$ rosrun map_server map_server -f <name>`

Keyboard Control:

`$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py`

Voice Control:

`$ roslaunch hunter_navigatino_goal voice_cmd.launch`

### 2. Navigation:

`$ roslaunch hunter Hunter_navigation.launch`

`$ roslaunch hunter Hunter_navigation_rviz.launch`

Reset localization:

`$ rosservice call /global_localization`



### 3. Send goal

Directly send goal:

```
$ roslaunch hunter_navigation_goals
```

```
Hunter_navigation_goals.launch
```

Send goal by voice control:

```
$ roslaunch hunter_navigation_goals SendGoal.launch
```

### 4. Voice feedback

```
$ roslaunch '/home/hunter/catkin_ws/src/pioneer3at_ETSIDI  
/pioneer_utils/voice_audio/launch/voice_cmd.launch
```

Check output:

```
$ rostopic echo recognizer/output
```

### 5. Voice recognize

```
$ rosrunc pocketsphinx recognizer.py
```

### 6. Kinect test

```
$ cd catkin_ws/src/libfreenect2/build/bin
```

```
$ . Pronekt
```

### 7. Skeleton tracker

```
$ cd NiTE-Linux-x64-2.2/Samples/Bin
```

```
$ . UserViewer
```

## **Comparing with the proposal**

When we finished these two semesters work and look back to our proposal, we have to admit that some of ideas and thoughts at that time is really naïve. Without the deep understanding of the robot area, we don't have a clearly cognition of the difficulties' degree of each parts of work. However, luckily, at least we know that what is possible to work out and what is almost impossible.

We thought that we finished at least 90% of our main purpose and some parts of additional purpose. The main function: navigation, voice control and the sound feedback have already achieved successfully.

Shake feedback and the force control are abandoned because of the limited of the guiding stack and the lack of the sensor.

We have a try of the 3-D mapping and it is not so hard actually. But with the limit of the Kinect's detecting degrees. We have to abandoned this part. And which mean we have abandoned that distinguishing the height can allow the human to pass or not.

So, in our opinion, though some detail functions we haven't worked out, we never deviate from our original way.

## **Conclusion**

What we are proud of is, we are always clear that what we need to do, what is useful for our design and what should we focus on at the current time. Senior Design is not just a simple course that teach you the knowledge. But it throws you into the actual conditions to let you use whatever you have learned to explore. During this process, the problems will occur from time to time. But if you spend time on it and try to solve it by yourself. You will understand it much deeper than just what you have been taught or what you have read. Trouble

shooting is painful. But it is the best way to learn. Always remember to make your hands dirty.

## **Reference**

ROS: [www.ros.org/about-ros/](http://www.ros.org/about-ros/)

RosARIA: <http://wiki.ros.org/ROSARIA>

A\*: [https://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

SLAM:

[https://en.wikipedia.org/wiki/Simultaneous\\_localization\\_and\\_mapping](https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping)

AMCL: <http://wiki.ros.org/amcl>

Kidnapped robot problem:

[https://en.wikipedia.org/wiki/Kidnapped\\_robot\\_problem](https://en.wikipedia.org/wiki/Kidnapped_robot_problem)