Project Report II

Team 4

Table of Contents

Design Objectives and Constraints:

The aim of this project is to develop a service robot for autonomous delivery of food. We aim to control the robot with a joystick remotely, through wireless communication, from a base station, for this phase of the project. The final phase of the project aims to accomplish complete autonomy with robust design of navigation system, path planning and safety critical features.

Design Objectives/Decisions Breakdown

In order to achieve our phase 1 goals, firstly we need to develop the sensor suite to perceive the environment around us and supporting framework to mount it on the robot platform. Our sensor suite includes SICK LMS LiDAR and Intel RealSense camera for point cloud data and RGB-D data. The LiDAR is mounted on robot chassis with high quality Velcro, and the camera is mounted on top of the LiDAR with Velcro. We use a hobby servo for achieving pan tilt for the camera. Multiple sensors increase our environment perception capabilities, thereby increasing coverage and improving prediction accuracies by decreasing uncertainty.

Next we need to develop the communication pipeline for remotely accessing the robot from our base station. This means we would need some sort of computing environment to execute the decisions made by us. We also need to actuate our motors using some form of embedded system. Communication from the base station to the robot can be made using an integrated Wi-Fi module or an external modem. Several on board batteries would be needed for powering our hardware and voltage regulation may be needed for power management. Appropriate choice for operating system must be made.

To that extent, our computing platform comprises of an on board PC, the INTEL NUC, integrated with a wireless communication module. We use the Arduino UNO integrated with the Polulu motor driver to control/actuate our robot motors. Power management system includes 3 LiPo batteries to power the robot motors, LiDAR and NUC respectively. Total power consumption is xxx. A xxV-xxV DC-DC convertor is used to power the Arduino and Polulu motor driver shield. The software architecture is governed by Robot Operating System (ROS) framework running on Ubuntu 16.04 installed on the NUC, which has lots of support for autonomous applications. For this project phase we use a joystick to navigate the robot, remotely from the base station.

We next develop the enclosure for aesthetic purposes, covering our hardware from the human view. All the hardware components mentioned above are mounted to the robot chassis using high quality Velcro and wire management/routing is performed with zip-ties, to ensure easy and reliable debugging of faults. The enclosure framework is cubical in shape and developed using rectangular aluminum tubing. The enclosure casing is made out of light weight plywood board to reduce self-weight of the robot. A plastic payload basket with anti-slip cup holders, is Velcro-ed to the top of the enclosure to hold the food items as needed. Roughly the bot weighs xxx.

Some of the safety critical features of this robot are: manual switch to disconnect power to the motors through a relay in advent of communication errors or human errors through software. We may incorporate a battery voltage monitoring circuit to monitor the voltage of the on board battery, thereby conducting in-time replacement. We may use an E-stop if needed depending on criticality of our application needs. We may later on add a temperature sensor to monitor the inside temperature, some form of water sprinkler

system to avoid fire hazards due to overheating. Since the robot is sufficiently ventilated due to our enclosure housing we need not bother much about heating/thermal issues for the electronic components on board.

We would also like to thank the CMS lab for providing us the support and equipment to make this project a success.

Design Constraints:

Some of the design constraints we imposed were:

- Given that we are developing a service bot, we wanted it to be small and compact in nature and so the dimensions for the enclosure casing were chosen accordingly.
- Light weight components were selected to ensure minimum self-weight of the robot.
- Sensors were only used to visualization. No perception was implemented in terms of object detection and object tracking.
- Navigation capabilities were restricted to joystick control.
- Smooth movement needs to be tuned by reducing large acceleration and deceleration.
- Feedback control remains to be implemented.
- Real time communication is not critical.

System Design:

In this section we elaborate on the design philosophy behind our service robot. The type of role the robot is going to play is integral to the robot design. Our task was to develop a robot that could serve food and drinks to humans. Keeping this in mind we designed a model that can co-exist in a human environment. Both electronic and hardware designs emphasize practical and safe operations. The following figure illustrates a high level block representation of the entire system.

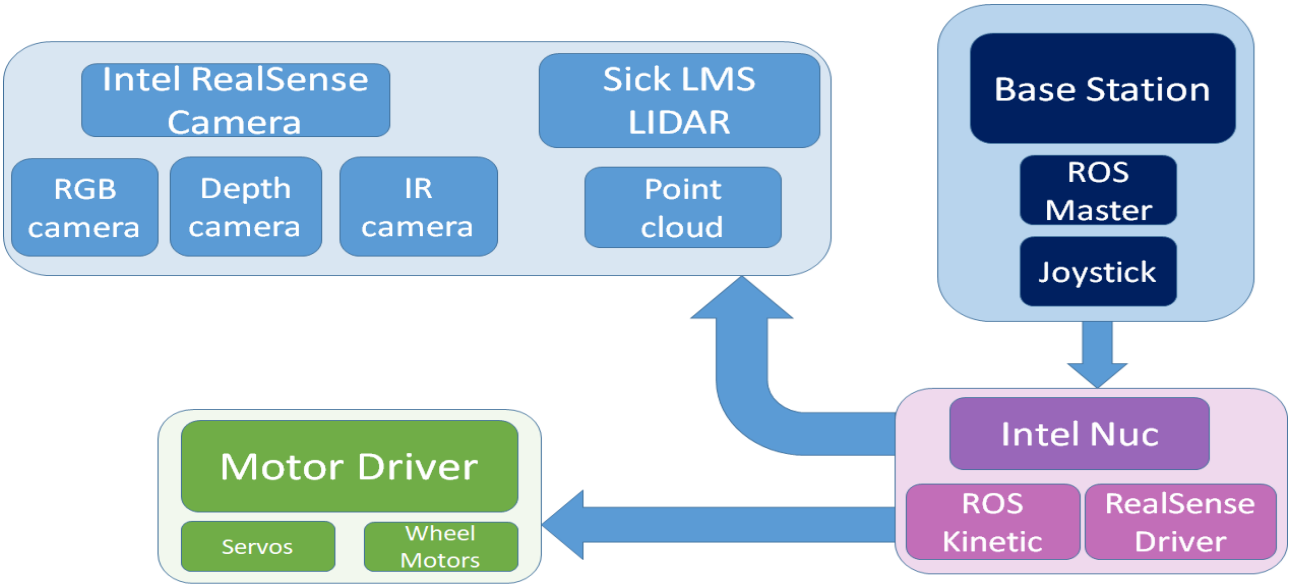


Figure 1: High level system representation

Electronic System Design:

Our electrical system comprises of sensors such as Intel RealSense and SICK LMS LIDAR. Actuation of the robot is through the use of a VNH5019 dual motor driver. Control of the robot is established using Logitech Joystick through the basestation communicating via Wi-Fi with the on board Intel NUC.

The communication mediums are USB for LIDAR, Intel RealSense and Arduino to the Intel NUC. While the Intel NUC communicates with the Base Station via Wi-Fi. The electronic system can broadly be divided into three modules

1. Central Operations Module
2. Sensor Module
3. Motor Actuation Module

Central Operations Module:

This module consists of the Intel NUC that is the brain of the entire robot operation. The Intel NUC controls sensor and actuation modules via connection to camera, LIDAR and motor driver. The module also consists of the Base Station which receives the ROS URI and allows remote operation of robot via Wi-Fi. The joystick is connected to the Base Station and controls the robot motion through input received form the Intel RealSense camera feed.

Sensor Module:

The sensor module comprises of Intel RealSense camera that is capable of providing RGB, RGB-Depth, IR and IMU information using ROS wrapper. It also consists of the SICK LMS LIDAR that provides point cloud information at a Field Of View of 180deg. The sensor module are the eyes of the robot and responsible for assisting in human control of robot trajectory.

Motor Actuation Module:

The motor actuation module consists of the VNH5019 dual motor driver that is capable of controlling two motors simultaneously. The VNH5019 is a shield which sits on to an Arduino and this Arduino can receive serial commands from the Intel Nuc to drive the motors at required speed and hence direction. We have used the driver to control 4 motors using the common output connections to operate two motors with two terminals and the other two terminals to control the other pair of motors.

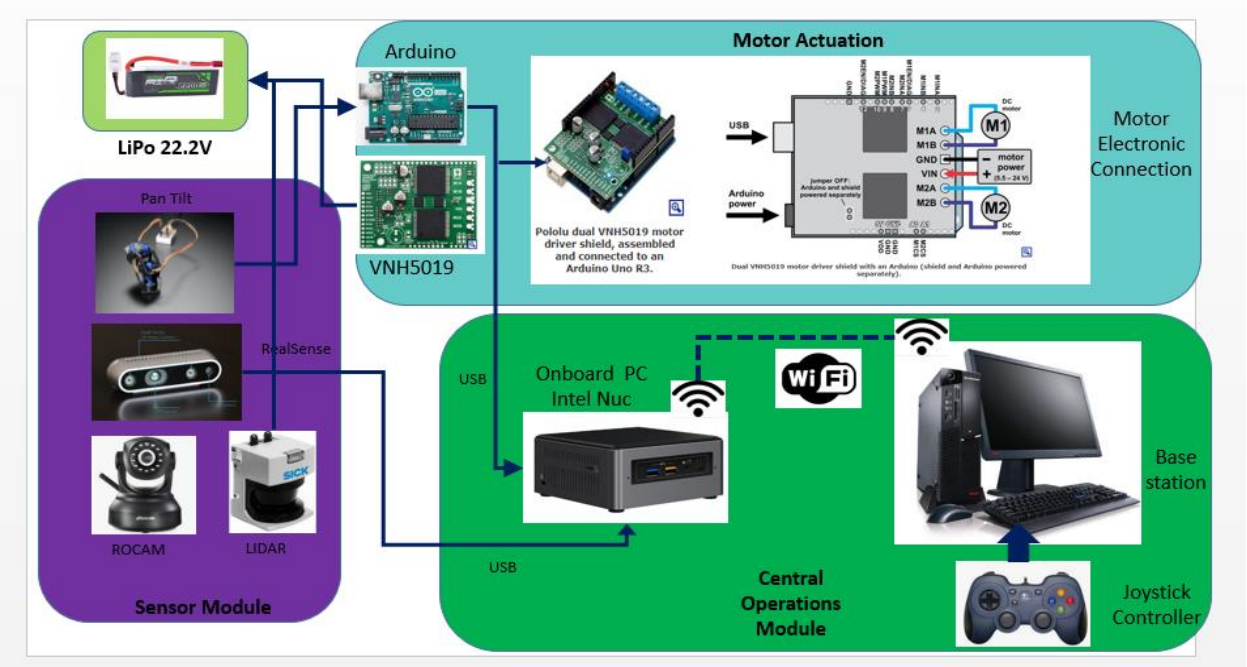


Figure 2: Electronic circuit connection based on the three key modules of the robot

The system is operated by 3 LiPO batteries whose connections are given in the figure in the next page. The connections showcase the power inputs of each of the components and also mode of communication between different systems.

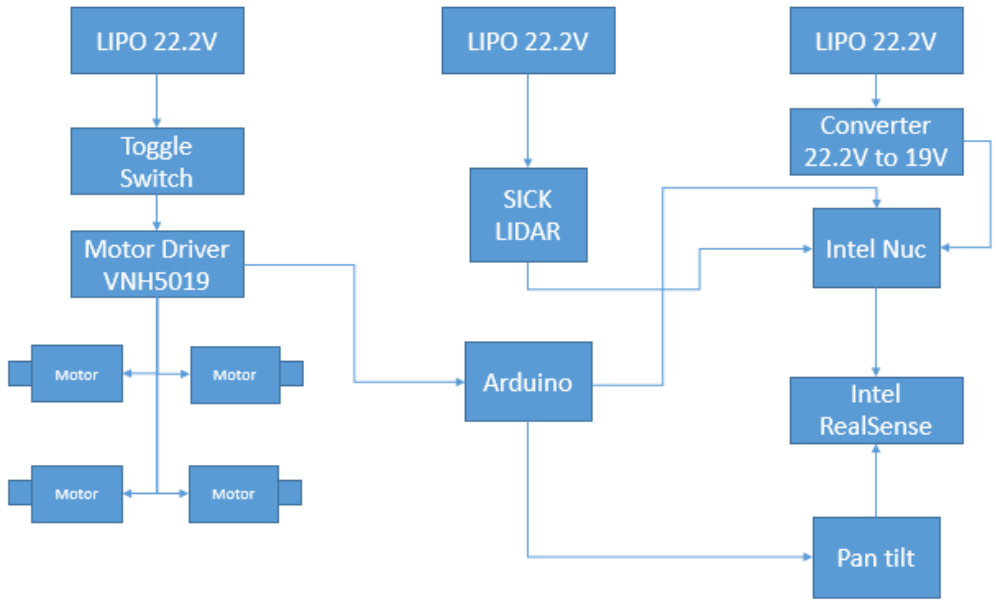


Figure 3: Power connection wiring diagram to each component

Hardware Design:

Hardware design of the robot is not only of aesthetic importance but also is a barometer for reliability and robustness. Our robot has been designed keeping in mind the prime factors that make a service robot more desirable. The key factors we have addressed with our design are accessibility, strength, weight and functionality.

The key components of the hardware consist of the outer frame of the bot which is made of light weight and durable hollow aluminum rods. The frame is held to the base via angle brackets and Velcro. The frame is covered with mixed plywood which is also light bud sturdy. The plywood is designed to enhance appearance. The height of the frame is such that a person sitting on a chair can easily reach for the required drink or food item. The front of the frame also has painted eyes as research shows that humans respond better to robots with such features.

The top of the robot is where we have place a serving tray. The tray has been compartmentalized to contain two cup holders that can easily fit cups and mugs with handles whilst not spilling drinks and a section for additional food items.

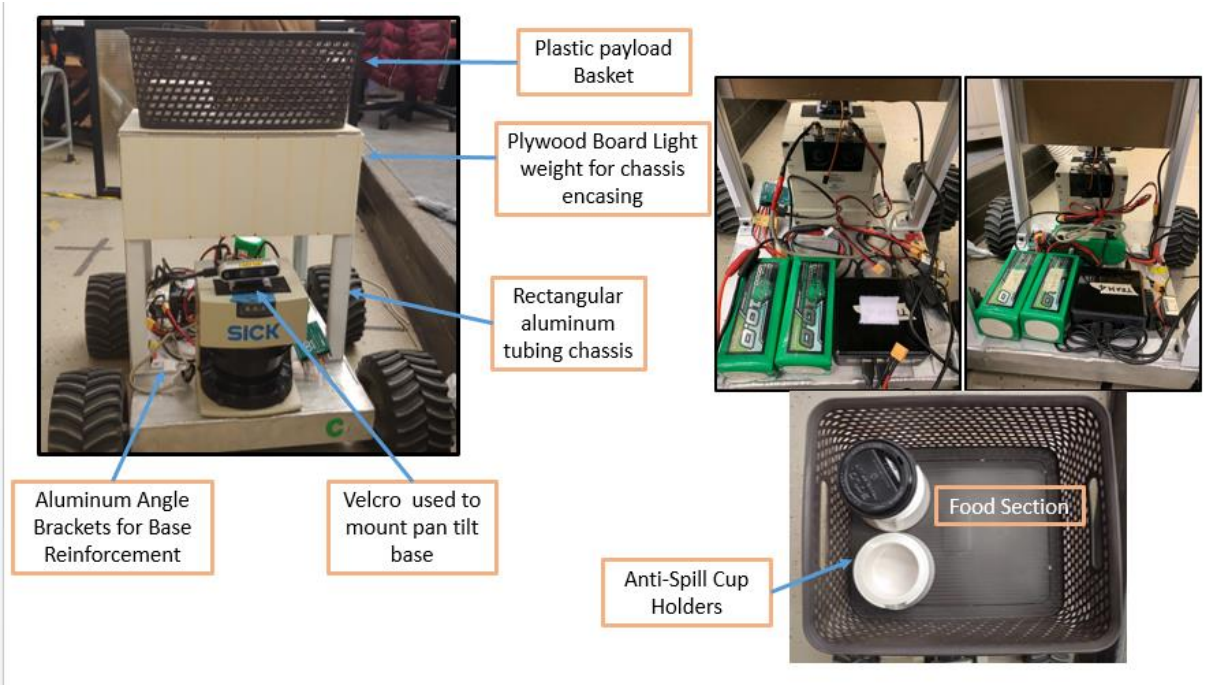


Figure 4: Hardware components

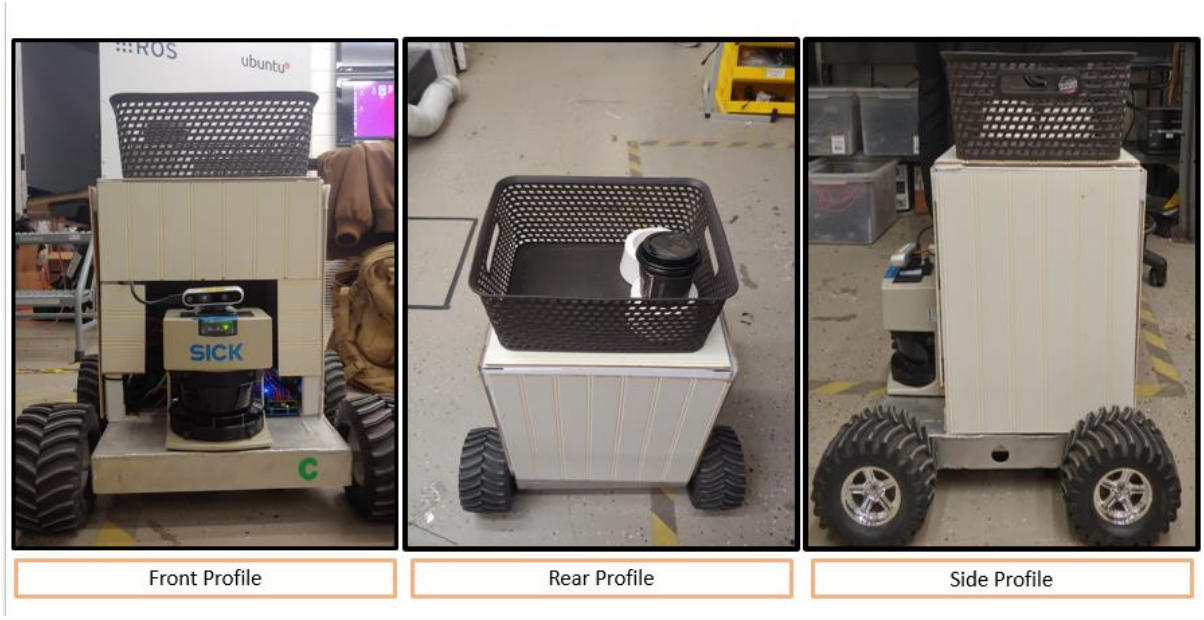


Figure 5: Profile views of complete robot

SLAM

2D – SLAM:

The Hector SLAM fits perfectly to our purpose of mapping. The uniqueness and choice of Hector SLAM is its ability to provide great indoor maps without the need of odometric information exclusively based on LIDAR data. SLAM in general follows the following sequence of steps in order to decipher a map of the environment.

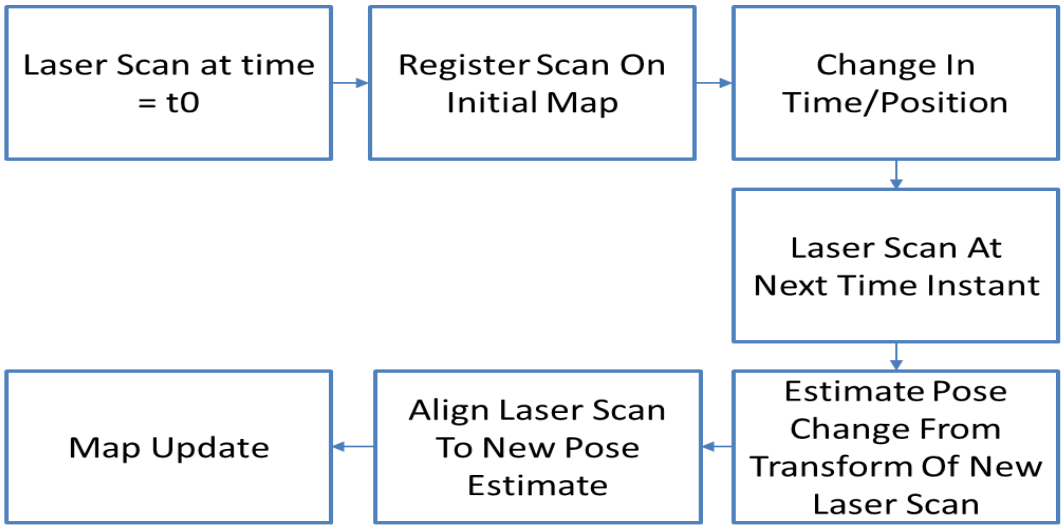


Figure 6: SLAM sequence diagram

Hector SLAM relies on occupancy grid map technique. When we place the UGV in the occupancy grid the LIDAR head needs to be ported to the cell values. We maintain a separate database of the laser

measurements corresponding to each cell values. The measurement is assigned a value of 1 if LIDAR ray endpoint is found else we assign it 0.

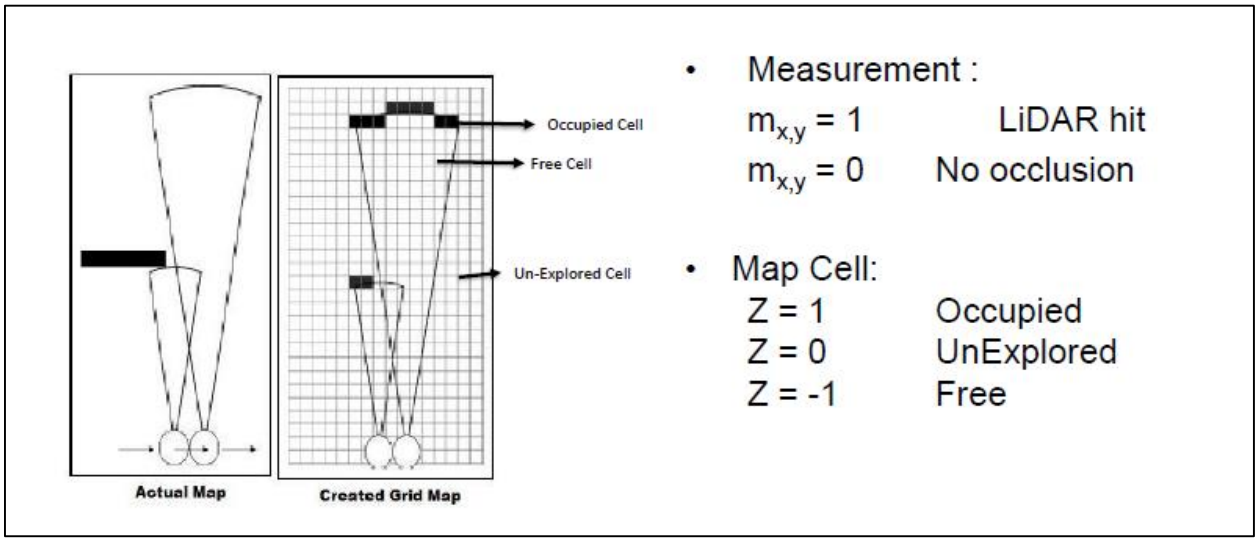


Figure 7: Occupancy grid map under the hood of Hector SLAM

Based on the above figure we can establish a measurement model of the Hector SLAM as

$p(\mathrm{Z} \mid \mathrm{m}_{x,y})$;

$$\log odd\_occ := \log \frac{p(z=1|m_{x,y}=1)}{p(z=1|m_{x,y}=0)} \quad \log odd\_free := \log \frac{p(z=-1|m_{x,y}=0)}{p(z=-1|m_{x,y}=1)}$$

Log Probability for occupied cells Log Probability for free cells

Our implementation of Hector SLAM was done using a ROS package by TU Darmstadt, where we received information from ROS topics that correspond to the Hokuyo Node that we implemented as part of the one of the Home Works. This laser point cloud information is later used by the Hector SLAM package to create a map of the environment. To establish a clear map required the robot to move in relatively slower speeds. As can be seen in Figure 6 the map on the right shows the clarity of the hector slam map with respect to the map assigned for our project on the right.

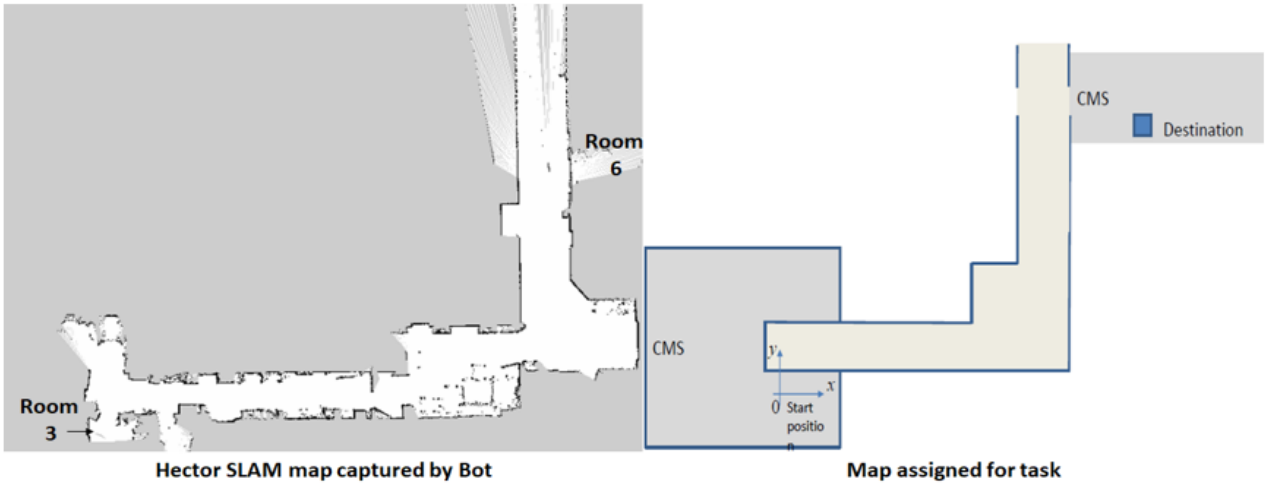


Figure 8: Hector SLAM map generated by bot (left) and map assigned for task (right)

3D SLAM:

In order to create a 3D SLAM of the entire basement region assigned. The package RTABMAP was utilized based on Intel Realsense d435 is the sensor information. Real Time Appearance Based Mapping (RTAB-Map) approach starts a 2d occupancy grid and 3d octomap of the environment. A process called Loop Closure is used to determine whether a robot has seen a location before or not. RTAB-Map is optimized for large scale and long term SLAM by using multiple strategies to allow for loop closure to be done in real time.

RTAB-Map (Real Time Appearance Based Mapping) [4] is a graph based SLAM approach. Appearance based SLAM means that the algorithm uses data collected from vision sensors to localize the robot and map the environment. In appearance based methods, a process called Loop Closure is used to determine whether the robot has seen a location before. As the robot travels to new areas in its environment, the map is expanded and the number of images that each new image must be compared to increases. This causes the loop closure to take longer with the complexity increasing linearly. RTAB-Map is optimized for large scale and long term SLAM by using multiple strategies to allow for loop closure to be done in real time.

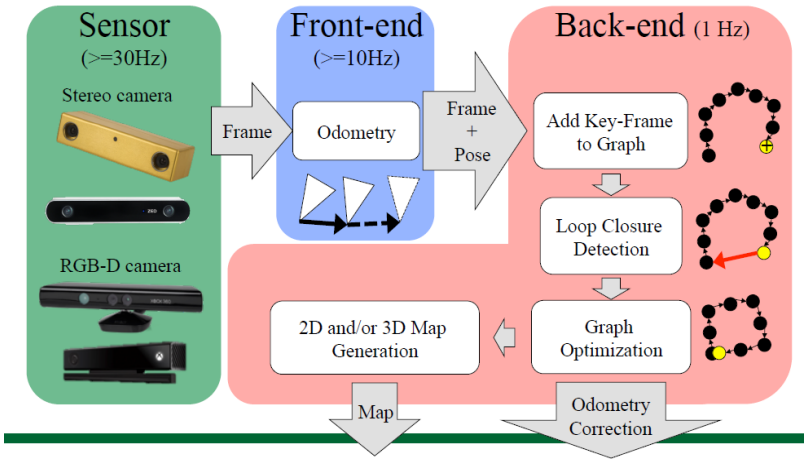


Figure 9: RTABMAP flow and module diagram*

The front end of graph SLAM looks at how to construct the graph, using the odometry and sensory measurements collected by the robot. This includes interpreting sensory data, creating the graph and continuing to add nodes and edges to it as the robot traverses the environment. The front end of graph SLAM also has the challenge of solving the data association problem. In simpler terms, this means accurately identifying whether features in the environment have been previously seen.

The back end of graph SLAM is where the magic happens. The input to the back end is the completed graph with all of the constraints and the output is the most probable configuration of robot poses and map features. The back end is an optimization process that takes all of the constraints and find the system configuration that produces the smallest error. The back end is a lot more consistent across applications.

The front end and the back end can be complete in succession or can be performed iteratively, with a back end feeding an updated graph to the front end for further processing.

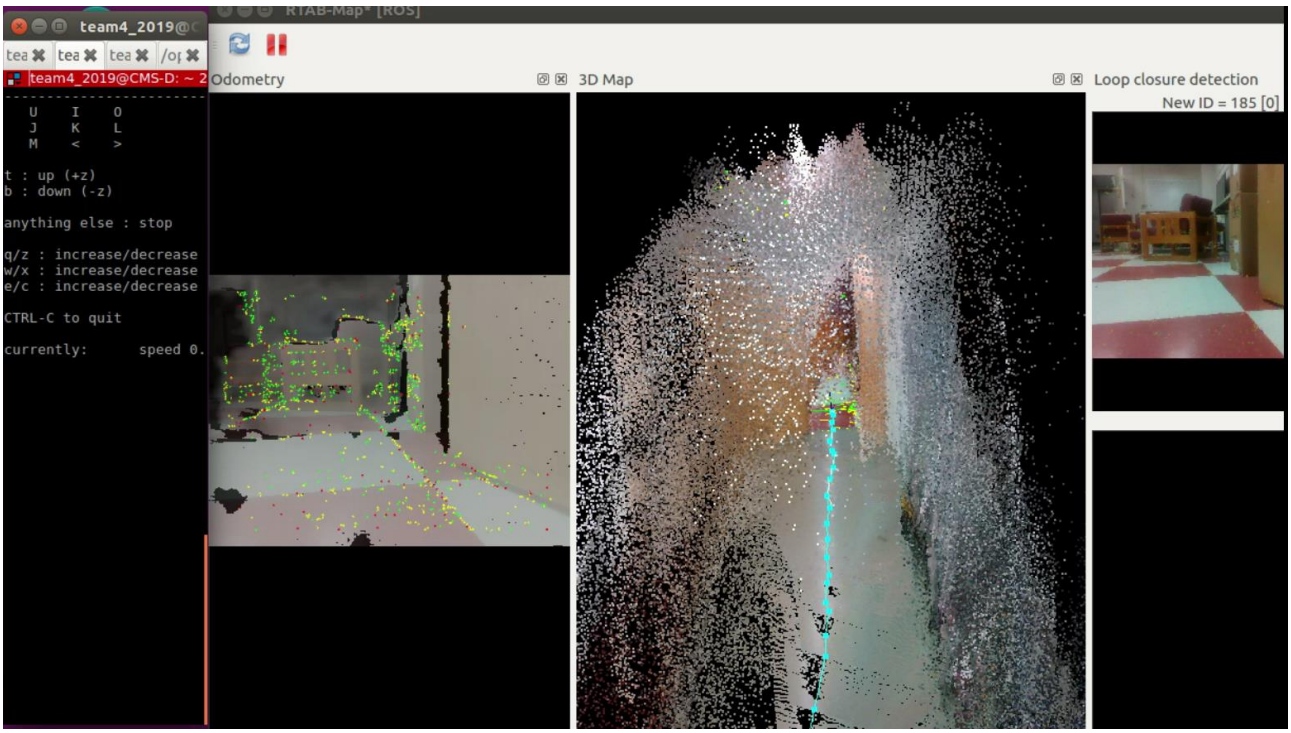
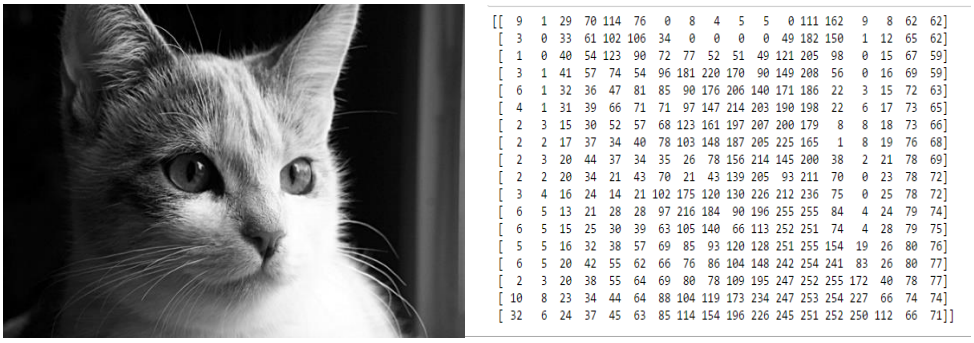


Figure 10: 3D Map generate using RTABMap

Perception

Perception is an ability to become aware of the surrounding by using sensing. For achieving perception, we made use of find_object_2d package. The robot has an ability to detect the save objects by using camera image. To understand the elaborate working of the package it is necessary to know how images store information. When we click a picture, the computer sees it as a combination of numbers between 0 to 255. In the case of black and white picture one picture is one number between 0 to 255. 0 being complete black and 255 being complete white. The RGB image pixel stores three values for Red, Green and Blue: all values between 0 to 255.



```
[[  9   1  29  70 114  76   0   8   4   5   5   0 111 162   9   8  62  62]
 [  3   0  33  61 102 106  34   0   0   0   0  49 182 150   1  12  65  62]
 [  1   0  40  54 123  90  72  77  52  51  49 121 205  98   0  15  67  59]
 [  3   1  41  57  74  54  96 181 220 170  90 149 208  56   0  16  69  59]
 [  6   1  32  36  47  81  85  90 176 206 140 171 186  22   3  15  72  63]
 [  4   1  31  39  66  71  71  97 147 214 203 190 198  22   6  17  73  65]
 [  2   3  15  30  52  57  68 123 161 197 207 200 179   8   8  18  73  66]
 [  2   2  17  37  34  40  78 103 148 187 205 225 165   1   8  19  76  68]
 [  2   3  20  44  37  34  35  26  78 156 214 145 200  38   2  21  78  69]
 [  2   2  20  34  21  43  70  21  43 139 205  93 211  70   0  23  78  72]
 [  3   4  16  24  14  21 102 175 120 130 226 212 236  75   0  25  78  72]
 [  6   5  13  21  28  28  97 216 184  90 196 255 255  84   4  24  79  74]
 [  6   5  15  25  30  39  63 105 140  66 113 252 251  74   4  28  79  75]
 [  5   5  16  32  38  57  69  85  93 120 128 251 255 154  19  26  80  76]
 [  6   5  20  42  55  62  66  76  86 104 148 242 254 241  83  26  80  77]
 [  2   3  20  38  55  64  69  80  78 109 195 247 252 255 172  40  78  77]
 [ 10   8  23  34  44  64  88 104 119 173 234 247 253 254 227  66  74  74]
 [ 32   6  24  37  45  63  85 114 154 196 226 245 251 252 250 112  66  71]]
```

Figure 11: Image representation in matrix of pixel values

Descriptors or keypoints are the information which stores these pixel values additionally the location of the pixel itself in the picture. So whenever the exact combination of the keypoints are found, it is able to create a box around the object. For our tests we saved two images and achieved the boxes created around the frames.

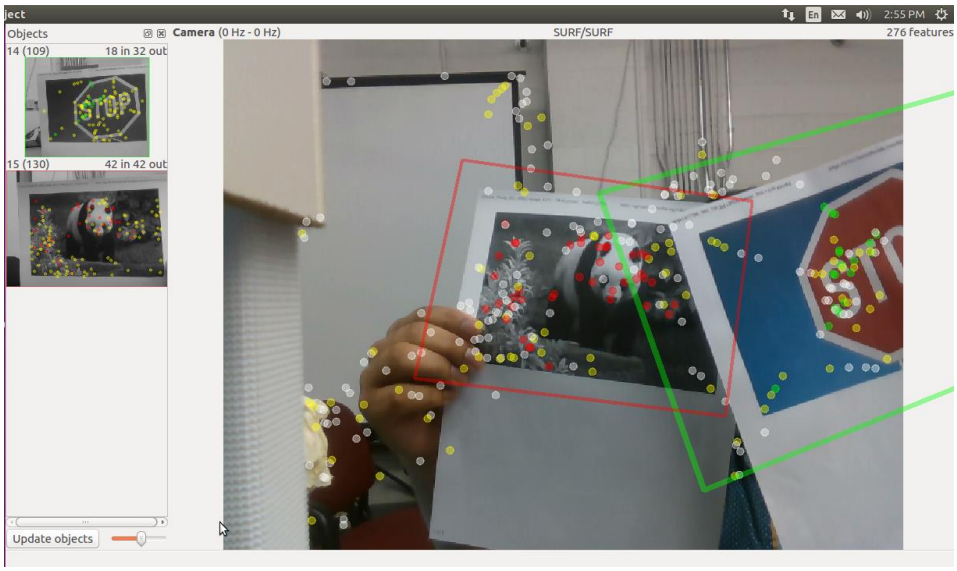


Figure 12: Detecting 2 different images using findobject_2d

The package works best with the textured frames. By textured it means, more variation of colors, edges, more linings, brightness and shapes. The QR codes can be used with the package as well. The final test was with the person sitting on the chair. The image topic used here is /camera/color/image_raw.

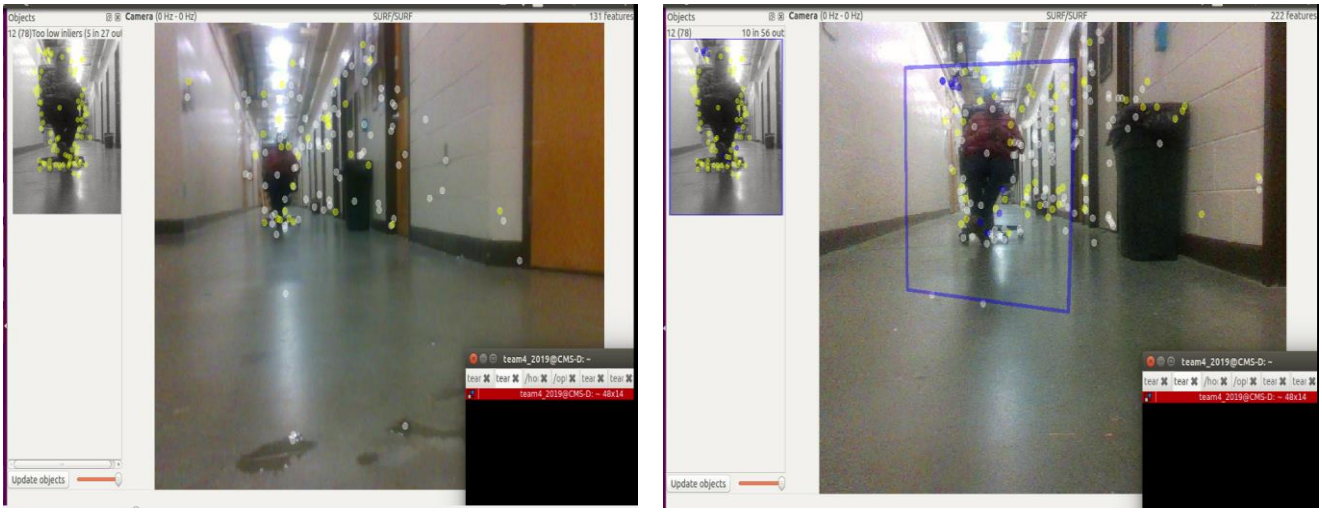


Figure 13: Recognizing human on chair

to run it with RVIZ give the following commands:

```
$ roscore
$ rosrun find_object_2d find_object_2d image:=/camera/color/image_raw
$ rosrun find_object_2d print_objects_detected image:=/camera/color/image_raw
$ rqt_image_view
# In RVIZ select /image_with_object topic to check the image with the box.
```

One thing to take in consideration is in RVIZ it will only show the picture frame when the object is in the screen. Otherwise the frame will display the message “NO IMAGE”. The proposed idea for target detection for service robot would be to take pictures of the customers while seated on the table and save them as objects followed by feeding the pictures back to the find_object package. The /image_with_object topic data can also be used in the object avoidance package.

Autonomy

Achieving autonomy is a challenging process. The work involved in setting up autonomy is heavily reliant on the navigation stack of ros. This is a package that works with different launch files and configuration files to enable them to work in sync with each other.

Navigation stack is a set of algorithms that use the sensors of the robot and the odometry, and allows control of the robot using a standard message. It can move the robot without problems (for example, without crashing or getting stuck in some location, or getting lost) to another position. To enable our robot we had to start by tuning configuration files and write some nodes to use the stack.

The robot had to first satisfy some requirements before navigation stack could be implemented:

• The navigation stack can only handle a differential drive and holonomic wheeled robots. The shape of the robot must be either a square or a rectangle.

• It requires that the robot publishes information about the relationships between all the joints and sensors' position.

• The robot must send messages with linear and angular velocities.

• A planar laser must be on the robot to create the map and localization. Alternatively, you can generate something equivalent to several lasers or a sonar, or you can project the values to the ground if they are mounted in another place on the robot.

The team was able to achieve almost all parts of the navigation stack configuration except for completing the tf publisher file. The figure below illustrates the parts that we successfully integrated

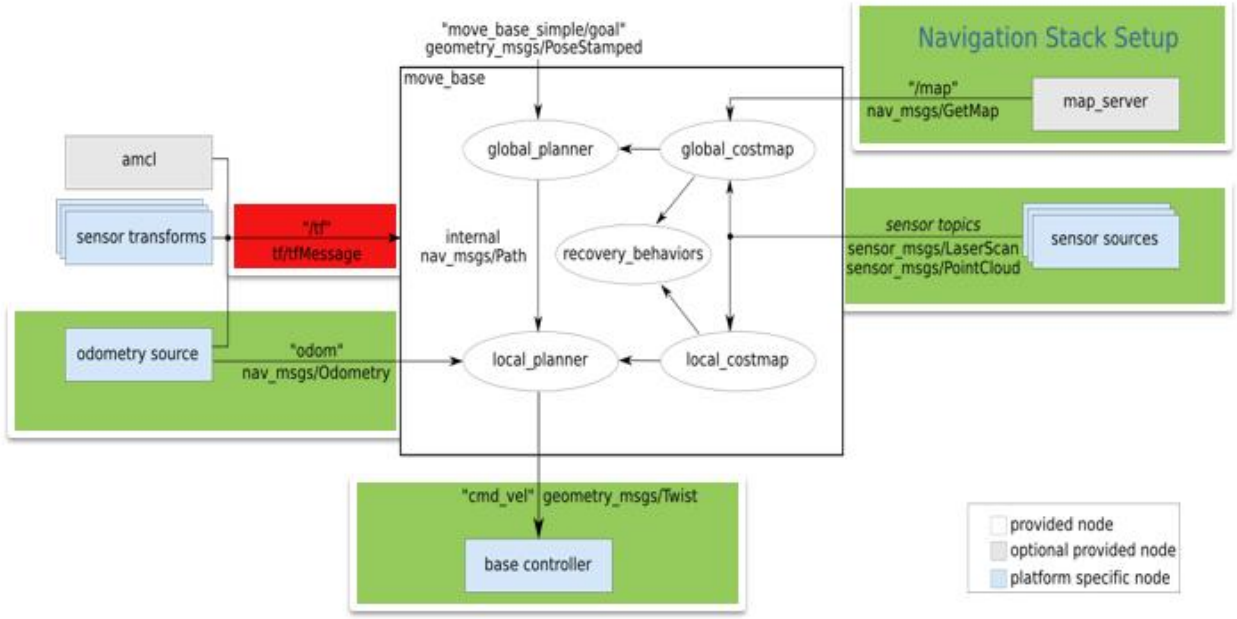


Figure 14: Segments of Navigation stack completed (marked in green) and incomplete part (marked in red)

Semi-Autonomous Operation:

As a next best scenario the team decided to resort to make the bot semi-autonomous by programming its movement to the environment specifically. In this endeavor we faced significant challenges that allowed us to understand the attributes apart from software that can present a hurdle.

- Wheel slippage
- Bearing control

- Uneven surface
- Wheel alignment

Despite these challenges we were able to get the bot to go a distance after which we resumed control from base station. The figure in the next page shows the route that was traversed by our bot during our attempts. Video footage of our attempts were presented in slide 16 of our presentation. We also have extensive video footage showing our runs that reach a great distance.

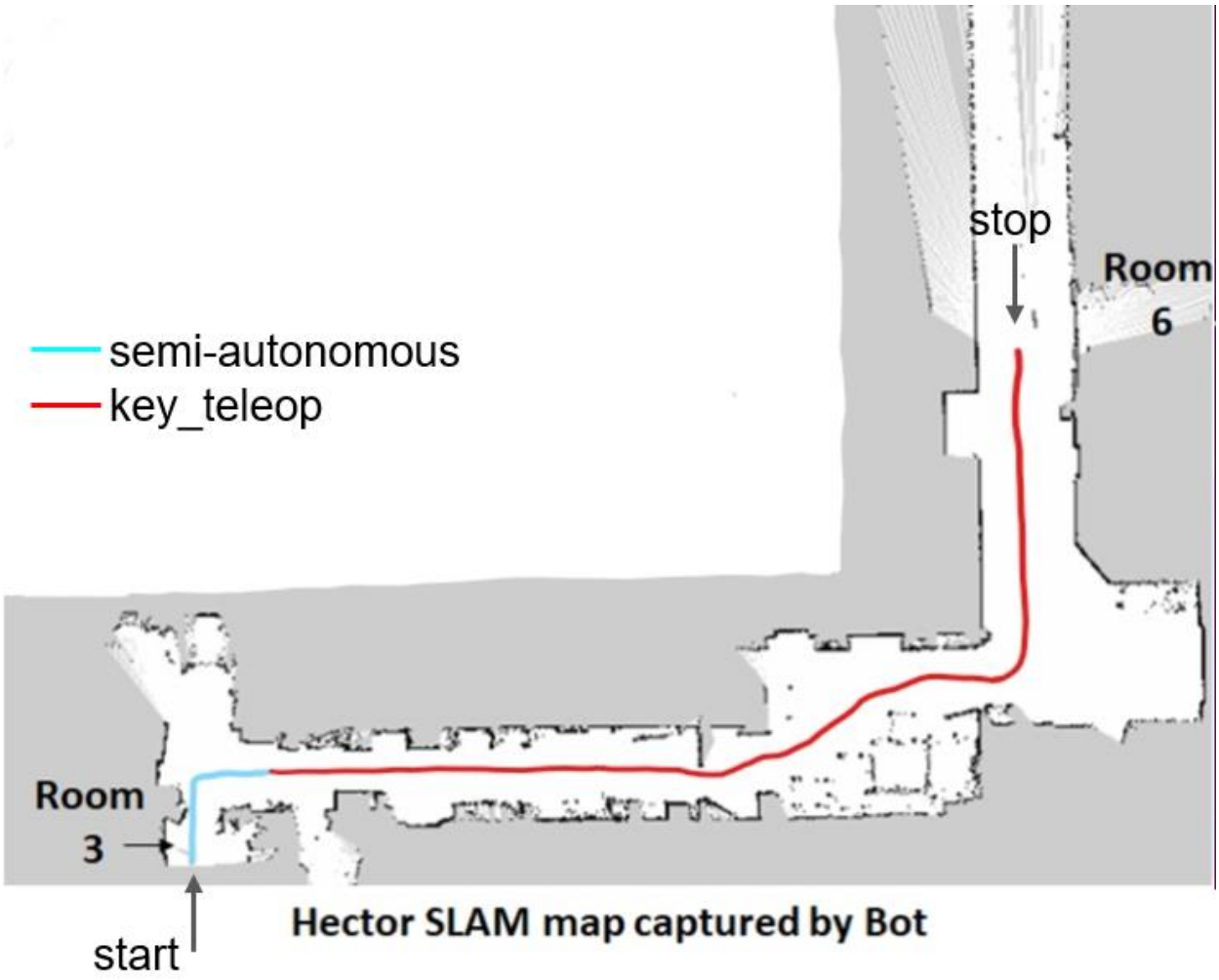


Figure 15: Semi-autonomous route the robot performed

Software System

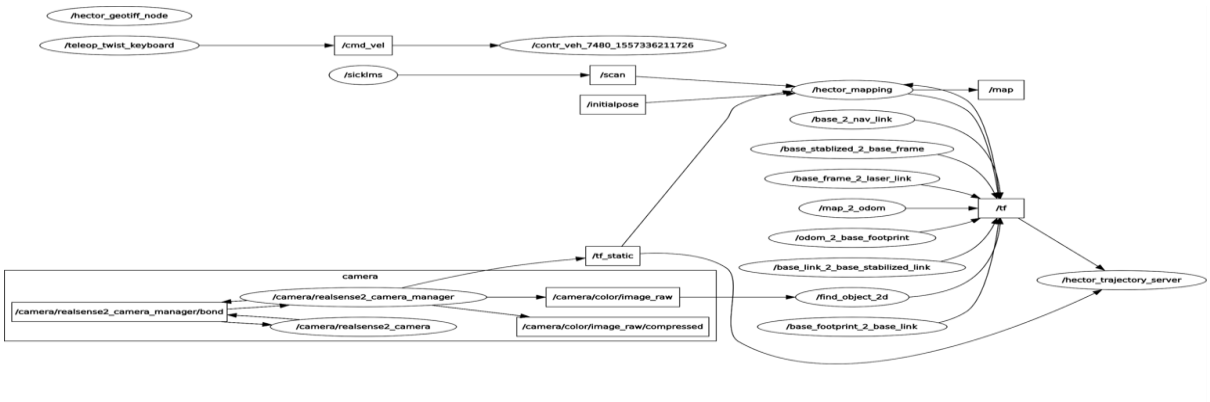


Figure 16: Software diagram

We have three major components that run all the software needed for running this robot. The Base Station is a fixed Computer which acts as a command station which we use to pass commands to our robot and display the information we get back from the robot such as the sensor data and camera feed. The On-board PC is the PC which we fix on the Robot. This PC is the 'brain' of the robot which controls all its motion and acts as a bridge between the base station and the sensors. It also acts as a bridge between the control messages received from the base station and the Arduino. Finally, we have the Arduino which controls the motors based on the instructions received from the on-board pc.

The onboard pc acts the master pc for our setup it runs 4 ros nodes as shown in the figure . the Sick LMS and the realsense nodes are mainly responsible for receiving the data from the lidar and the realsense camera respectively and publishing them as rostopics for further processing. While the contr_veh node is responsible for accepting the control messages from the base station and converting them into instructions that the contr_veh_joy program running on Arduino which controls the motors can understand. On the base station the hector_slam is responsible for reading the rostopics published by the sicklms and using that to make a 2d map of the environment. We also use the find_object_2d package to interface with the realsense camera for object detection and perception in the robot. While the key node passes on the instructions it receives from the keyboard to the onboard pc in the form of control messages.

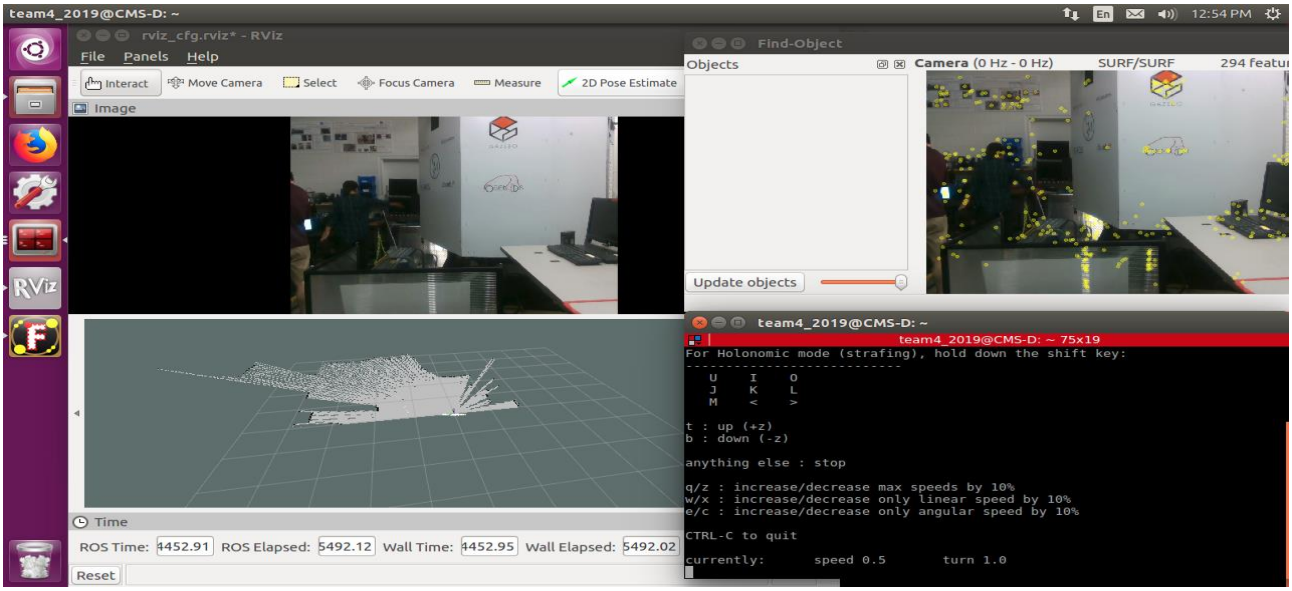


Figure 17: GUI created for Demo

User Manual:

Step 1: Initialing the base station (Only needs to be done once, ignore if already setup)

Since our entire package runs on ros we assume ros is already installed on the base station if it is not please install it as shown here : http://wiki.ros.org/ROS/Installation (robot designed and tested on ROS Kinetic other versions may or may not work)

1. Install Joy package (ignore if already installed) as shown in http://wiki.ros.org/joy/Tutorials/ConfiguringALinuxJoystick

2. Install the hector_slam package (ignore if already installed) as shown in http://wiki.ros.org/hector_slam

3. Install the find_object_2d package (ignore if already installed) as shown in http://wiki.ros.org/find_object_2d

4. open the .bashrc file and paste the following commands at the bottom export

ROS_MASTER_URI=http://192.168.1.11:11311 export ROS_IP=*your ip address*

Step 2: Starting the base station

1. Open a terminal and run the key_teleop node : rosrun teleop_twist_keyboard teleop_twist_keyboard.py

2. Open a new terminal and run the hector_slam : roslaunch hector_slam_example hector_hokuyo.launch

3. Open a new terminal and run the find_object_2d package: rosrun find_object_2d find_object_2d image:=image_raw

Step 3: Starting the Robot

Switch on the Robot and the motors.

Experimental Validation:

This section involves the summary of the tests performed from the first day of the course to the demo presentation day

Step 1: RO-CAMS and Realsense Camera:

We started to work with ROCAMS using ROS. With joy_teleop and key_teleop packages. ROCAMS have two degree of rotations: Horizontal and vertical. The first success was after operation it with joy_teleop node for which video can be found on the slides (Video 2, 3 on slide #8). Due to security reasons the cameras were replaced by Realsense Cameras. Which has both RGB and depth for each pixel. The output of the cameras can be seen using RVIZ. The link to install Realsense wrapper for ROS is https://github.com/intel-ros/realsense.

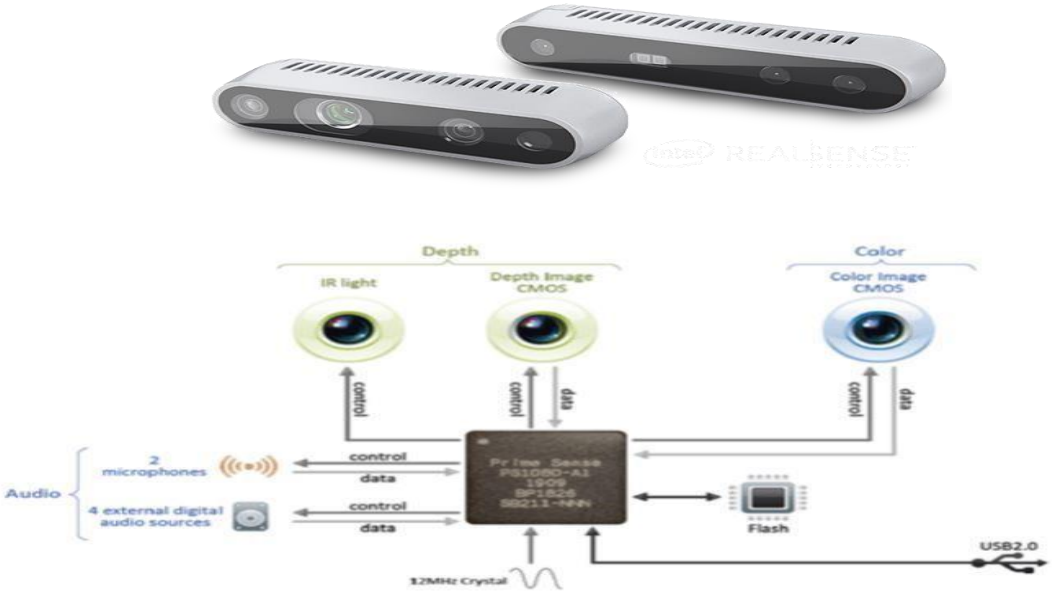


Figure 18: Intel Real Sense

Step 2: Working with Arduino

The Arduino works as a control of the motors. As the onboard PC sends the command according to joy__node,

The Arduino controls the vehicle signals. We used Arduino Uno and Arduino_serial packages for ROS.

(link:http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup)Before communication with ROS we tried burning the motor controls in the Arduino code itself to check the working of motors and signals(Video 1 on slide # 9).

Step 3: LIDAR

Sicktoolbox_wrapper can be installed from this link http://wiki.ros.org/sicktoolbox_wrapper. On ROS using RVIZ we can see the making of point cloud with the LIDAR. Which will be helpful in making the simultaneous localization and mapping (Video 1 in slide #8). We then use this Lidar with the Hector_Slam package to be abke to build a 3d map of the environment.

Step 4: Combining.

The goal of the project was making an easy to use Service Robot.
We wrote a script so that all the components and sensors will initialize on the boot up of the onboard PC. As out ROS master is running on the onboard PC, as soon as it boots up, it starts to publish the rostopics to the connected base station. ROS has a mechanism to prevent users from messing with root directory called listserv. To find the workaround we used a tutorial on this link
(https://bash.cyberciti.biz/guide//etc/init.d). The final working of the Robot to go from point a to point b is shown in Video 3 on slide #9. The last and very important step was to setup the communication between base station and the onboard PC using ROS_MASTER. The
communication was written in the ~/.bashrc script. (Video 2 on slide #9)

Step 5: Assembling the components.

Making the inner arrangement of the robot neater and accessible the Velcro is used.
The Side and front profile of the model can be seen in the pictures.

Figure 19: Assembled bot

Conclusion

Conclusively, we are able to achieve the target detection using Realsense camera and create a box around the detected object. Additionally, we can achieve 2D and 3D maps of the area. For the navigation stack, we got the odometry data from the rtabmap, but weren't able to connect it to base_link via transformation (tf) script.

Future Work

We intend to work towards following goals:

1) Complete Tf transform to achieve autonomy. the script involves using odometry data received from the camera to connect to base_link of the robot.

2) Achieving object avoidance. The robot should be able to detect the barrier in it is path and be able to find a workaround to avoid collision.

3) Enabling sensor fusion using Bayesian robotics concepts .Acknowledgements

Acknowledgement

We would like to thank Professor Dr. Furukawa for providing us the opportunity to explore theory through experimentation. This course has been stimulating and highly engaging.