Final Project

Team #2

Contributors: Japnit Sethi and Arthur Pawlica

Executive Summary:

The Course of Experimental Robotics ME 5874 under Prof. Tomonari Furukawa has introduced to Team #2 and other such teams to the open source Robotic Operating System and its application in the robotic forte.

In the Spring 2019, Team 2 was provided with the project challenge of transforming an unmanned ground vehicle to a catering service robot that delivers a cup full of liquid in the fastest and safest way possible without spilling it during delivery. Traditionally, food service is completely reliable on human labor, resulting in high labor costs and sometimes inefficiency in delivery. With our eventual autonomous system, each delivery assignment would be categorised uniquely and greatly reduce the inefficiency and replace the human labour cost with low maintenance cost.

For the Final Project, Team 2 applied knowledge from mechanical design, system integration, establishing wireless communication, object detection, SLAM and autonomous navigation through an open source platform like ROS to develop a system that would meet all customer needs across the service robot industry and achieve necessary target specifications. Knowing that an autonomous service robot exists in the market and costs over forty thousand dollars **[1]**, the team was able to design and build a less expensive product than with a smaller budget. The team has been able to complete the detailed design of the service robot with some future recommendations and targets to achieve after the completion of the course. This report discusses the design objective, determination of target specifications, mechanical and software design and finally the validation phase to validate the target specification set for various components in our system.

Table of Contents

I. Introduction

We are surrounded by convenience appliances like dishwashers, vacuum cleaners, and refrigerators (just to name a few) that improve our standard of living. There is still plenty of room for improvement to improve where there remain many tasks to automate such as: refrigerator delivering the required item stored in it, fetching the newspaper. Many of these tasks seem unnecessary but a lot of people would agree to have a personal attendant to help assist in such tasks.

Thinking about it in a broader perspective, majority of us who can handle the day-to-day activities do not think about commercial things being autonomous whereas if we think about the disabled or the elderly, it offers a much more relevance and human service costs much more in that scenario where it can be replaced by service robots.

Some of the cons that we can think about are that these robots can be considered invasive, and that they are only useful in certain predefined conditions. For example during perception, the images that are not taught to the robot can cause some difficulties in things like recognition.

Currently, caregivers in elderly homes and servers in restaurants, the cost of caregivers is high, and is going to keep on increasing,and with time the shortage of quality and quantity of these amenities would just decrease. Making such tasks autonomous would help support the cause and helpful in future to prevent such deficiencies in the service sector.

Based on all the limitations that we are facing currently and the potential future ones there is a need for system that provides the functions of an autonomous personal robot that helps the disabled or elderly to remain independent as long as possible and provide the much needed boost in the service sector is the need of the hour.**[2]**

Design Objectives

Team 2 began by evaluating customer needs from the project details provided by Prof. Furukawa. This served as an outline of the essential requirements for the completion of the semester project , as well as standards of what potential consumers would expect. These customer needs were then converted into target specifications for the robot. Below are the requirements for the final project:

Site Conditions:

- Tiled floors with stable horizontal surface and some potential turns for the robot to take

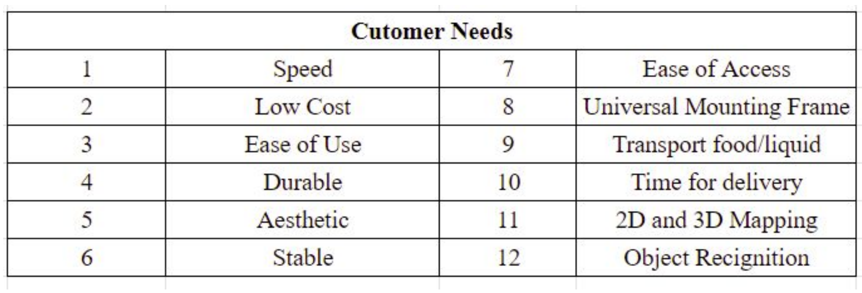
Goals:

- Build a strong metallic frame capable of holding a cup of water weighing at least 16 oz
- Networked ROS on onboard computer and base station
- Image feed from camera for visual feedback
- Object Recognition
- SLAM (2D and 3D mapping capability)
- Navigation using 2D map

Customer Needs

Using these provided project goals and site constraints, we were able to develop the following customer needs:

Table 1: Customer Needs



Cutomer Needs			
1	Speed	7	Ease of Access
2	Low Cost	8	Universal Mounting Frame
3	Ease of Use	9	Transport food/liquid
4	Durable	10	Time for delivery
5	Aesthetic	11	2D and 3D Mapping
6	Stable	12	Object Recignition

Design Constraints

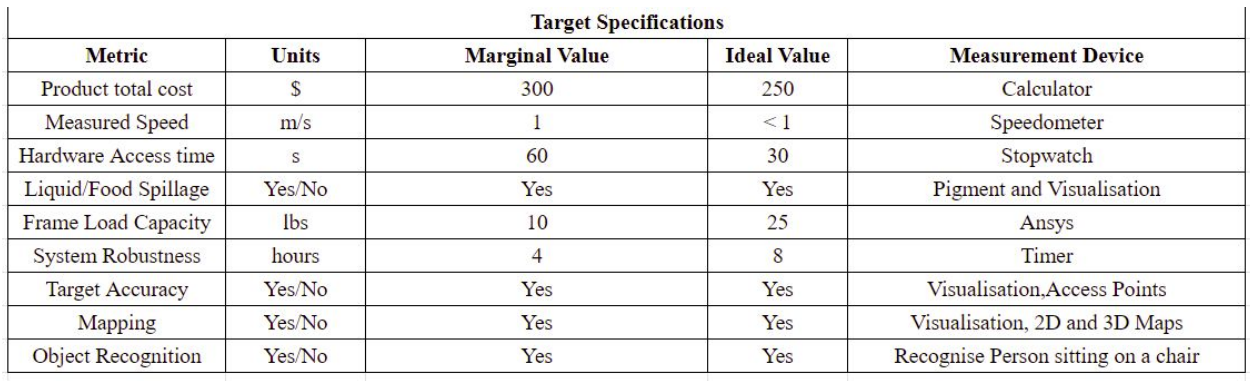
The next task was to translate these customer needs into measurable engineering characteristics that we could use to quantify our progress. The developed engineering characteristics were found to be:

1. Product total cost
2. Measured Speed
3. Hardware Access time
4. Liquid/Food Spillage
5. Frame load capacity
6. System Robustness
7. Target Accuracy
8. Mapping
9. Object Recognition

Target Specifications

Using research of similar designs and known properties, the Team 2 then determined appropriate marginal and ideal values of the engineering characteristics. The marginal and ideal values were used to compare our planned design to the characteristics of current solutions on the market. The marginal and ideal values will also give us a way to validate our design during and after the build stage. Values such as the speed, cost, and hardware access time were inferred from current solutions with our final product properties in mind, while the Frame load capacity, and food spillage are component restraints that would have to conform to given our solutions. Object Recognition and Mapping are important Metric taking the next step towards autonomous navigation which are both passed or failed based on a visual perception from the person operating the UGV.

Table 2. Target Specifications

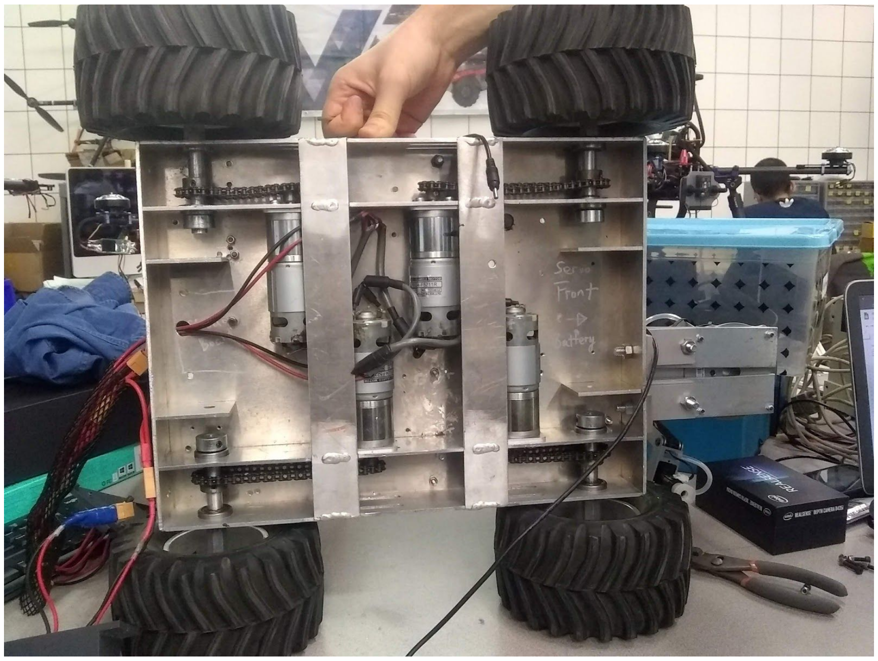


Target Specifications

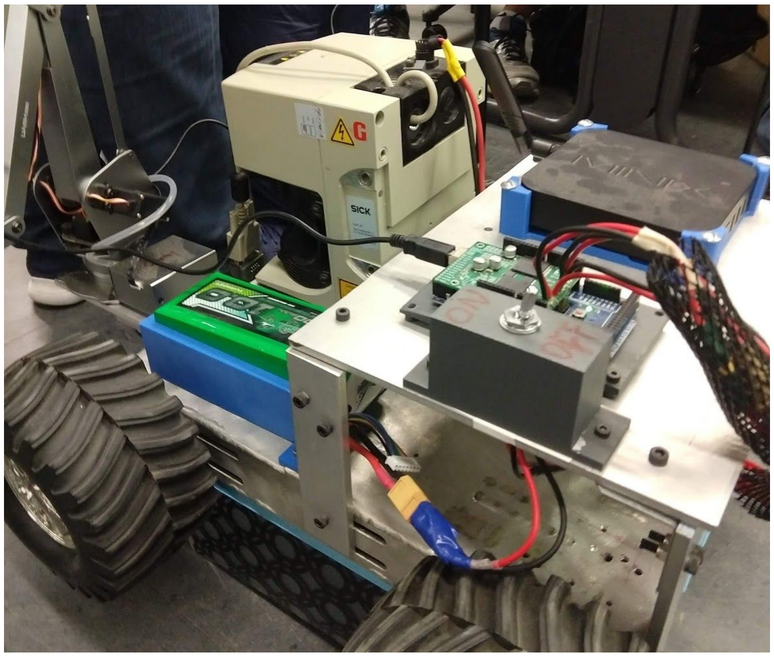
Metric	Units	Marginal Value	Ideal Value	Measurement Device
Product total cost	$	300	250	Calculator
Measured Speed	m/s	1	< 1	Speedometer
Hardware Access time	s	60	30	Stopwatch
Liquid/Food Spillage	Yes/No	Yes	Yes	Pigment and Visualisation
Frame Load Capacity	lbs	10	25	Ansys
System Robustness	hours	4	8	Timer
Target Accuracy	Yes/No	Yes	Yes	Visualisation,Access Points
Mapping	Yes/No	Yes	Yes	Visualisation, 2D and 3D Maps
Object Recognition	Yes/No	Yes	Yes	Recognise Person sitting on a chair

Mechanical System

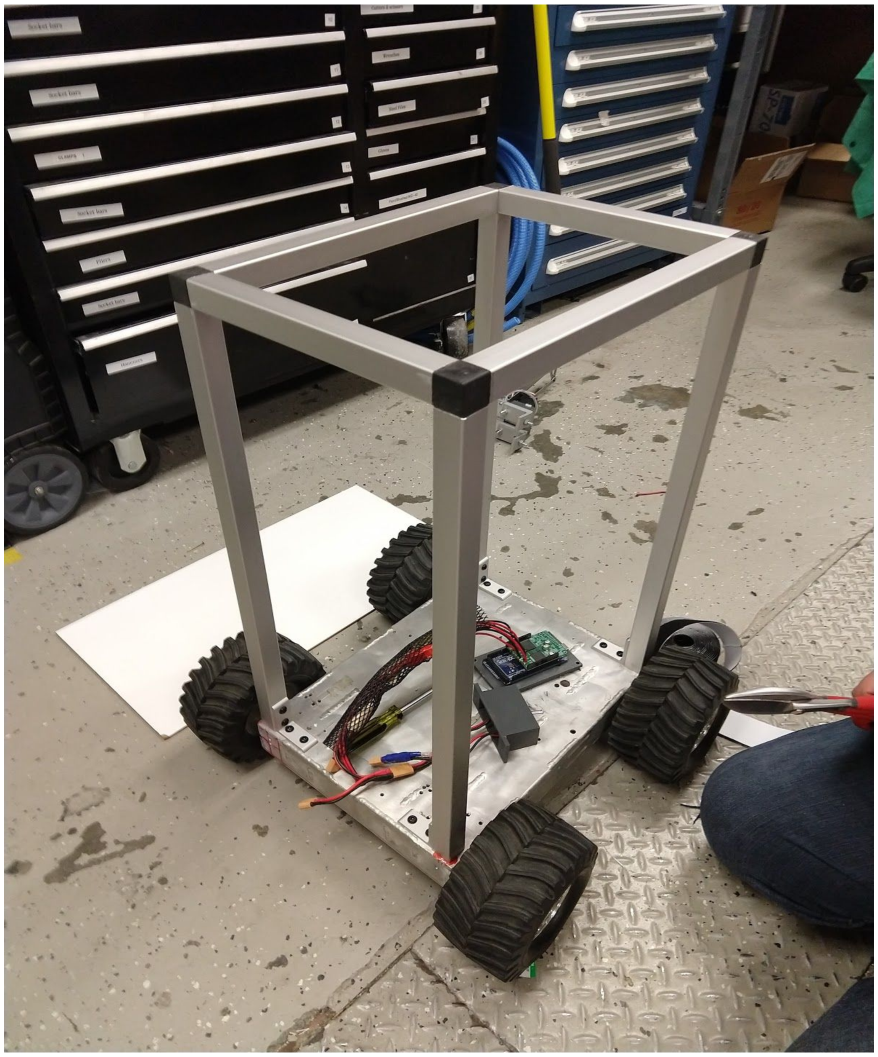
The robot has a fairly straightforward mechanical system. The base of the robot serves as the platform to which everything else is mounted, and is constructed of aluminum sheet. On its underside, it houses 4 motors that drive the wheels via chain, as seen below.



When the heavy duty base was provided to us, most components were mounted directly on the top of the base and on a shelf mounted to the base, as pictured here.

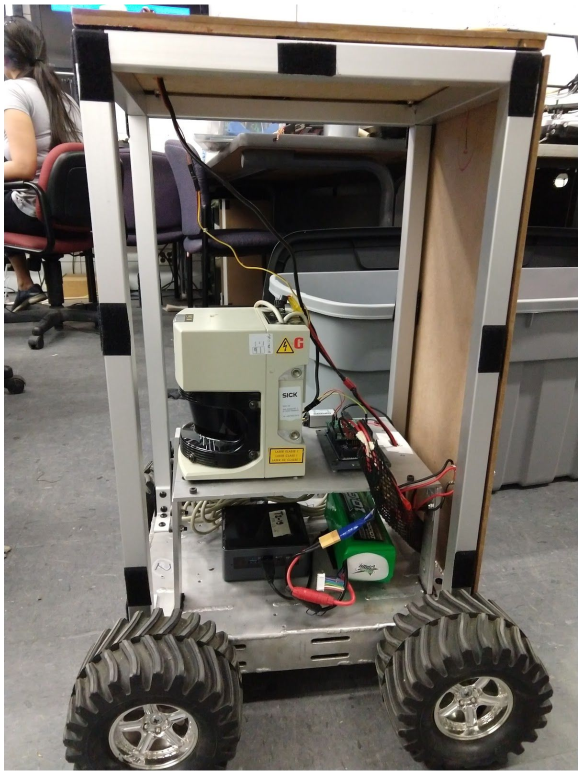


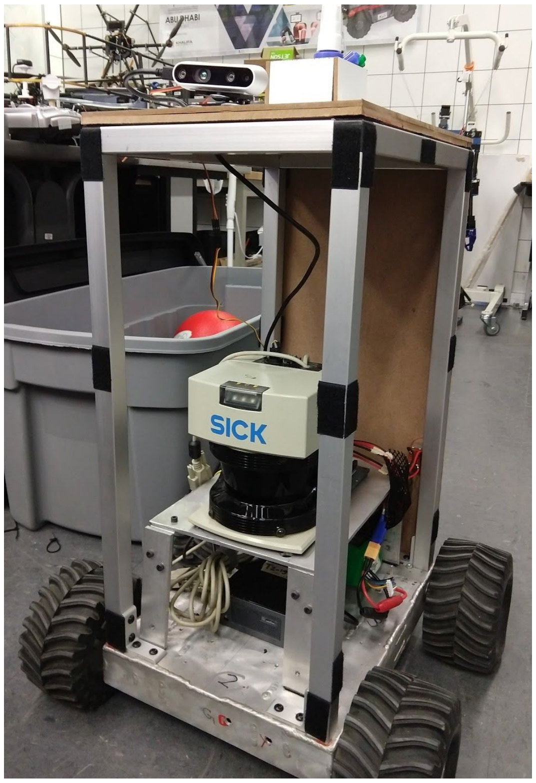
The group removed these components to allow for the placement of a frame to give the robot some height as well as a surface to place drinks/food on. The frame was constructed with 1 in^2 aluminum tubing with 1/16 inch wall thickness joined together by plastic, press-fit 3-way elbows. L-brackets were used to securely bolt the frame to the base of the robot.



Side and top panels for the frame were cut from fibreboard with a whiteboard coating. This material was chosen for its clean aesthetic, low cost, and ease of cleaning. This also allows people to draw and write on it, which could be used as a feedback system for a service robot, to improve the visual appearance of the robot, or even just to provide an entertaining way for users to interact with the service robot and grow accustomed to it.

The group chose to fasten the panels to the frame using high strength hook-and-loop fastener (commonly known as Velcro). This was done primarily for ease of access, as the panels can be quickly removed and reattached should any internal components need to be accessed. Additionally, the robot is not expected to bear any loads or encounter any resistance that could remove the panels, so the group felt that bolting the panels to the frame would not serve a practical purpose. In the future, quick release clamps may be substituted to provide a more secure hold, should it be required.



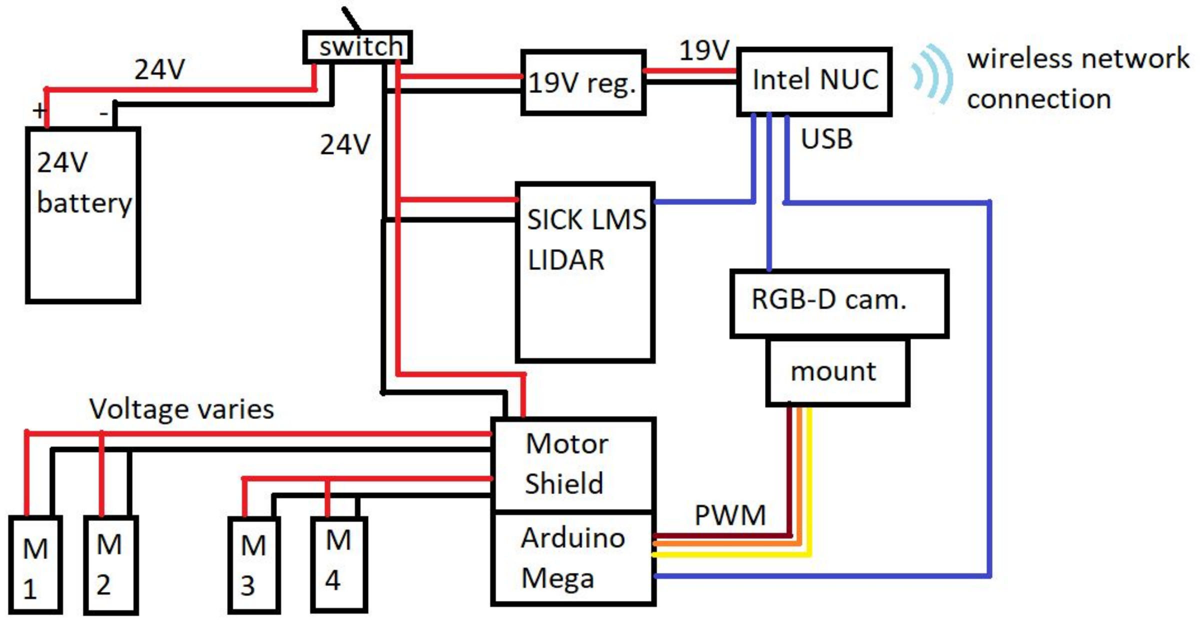


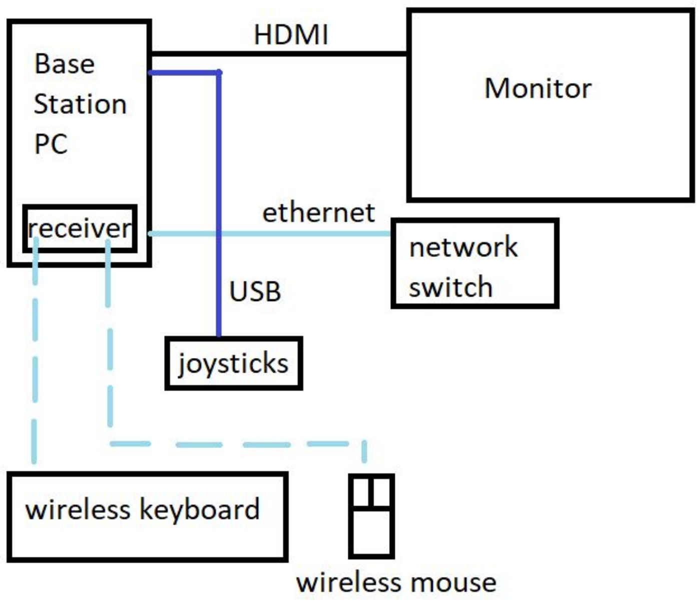
The necessary components that were previously removed were replaced, along with a few new additions. To provide more mounting space, the group chose to reuse the shelf that had been previously mounted on the robot, only moved to fit inside the frame. On top of this shelf sits the SICK LIDAR, a 19 volt DC/DC converter, which allows the PC to be powered via the battery and the Arduino Mega + Pololu Dual VNH5019 Motor Driver Shield, which is the motor control components of the robot, respectively. Underneath the shelf, the battery pack adheres to the base via a large strip of hook-and-loop fastener, which is more than sufficient to keep it secured. Also under the shelf is the Intel NUC PC that serves as the brain of the robot

An Intel RealSense D435i camera was mounted in a servo-controlled pan/tilt mechanism (designed for the CMU PixyCam and adapted for the D435i) on the top front of the robot to provide visual feedback. In the future, this may be moved to the rear top of the robot such that the cargo could be visually monitored as well.

Electrical System

Several changes to the robot electrical system were made for the final project as design objectives were added. The primary power source for the robot and its components is a 10,000 mAh 22.2V LiPo battery pack (when fully charged, this pack usually runs hot at about 24V). The battery terminals are wired to a global power switch that turns the power on/off for the entire system. From the switch, power is routed to a 19V DC/DC converter, the SICK LMS LIDAR, and the Pololu Dual VNH5019 Motor Driver Shield. The 19V output from the converter is used to power the onboard Intel NUC PC, which in turn powers the RealSense D435i camera and Arduino Mega through their respective USB connections. The RealSense camera is mounted on a panning mechanism powered by an RC servo motor, which is controlled via PWM to the Arduino. A third USB connection exists between the LIDAR and the onboard PC over which the laser scan data is sent. The two outputs from the motor driver shield are connected to the motors in the following configuration: one output connects to both motors on the left side, and the other connects to both motors on the right side. This essentially treats each side as if it were a single actuator, which is why the group opted to use a tank-type drive system (described later in the User Manual). Lastly, the NUC PC is connected to the lab WiFi network wirelessly (this is how it can communicate with the base station). The images below show the onboard electrical layout as well as the base station layout.





The base station layout is straightforward and has not been changed since Project 1 was completed. The base station is connected via ethernet to the lab network switch, which allows it to communicate with the onboard PC. A wireless mouse and keyboard are used for ROS commands, and the PC is connected to a monitor with an HDMI cable. Lastly, the joysticks that are used to control the robot movement are connected to the base station PC via USB.

Software System

Robot Operating System, or ROS, serves as the software platform for the robot. To begin, ROS was installed on both the base station PC and the onboard PC. For Project 1, the most recent version of ROS, Melodic, was used. However, the design objectives for the final project utilized some ROS packages that were either not supported or not fully supported on Melodic, so the team reinstalled ROS, this time using an older distribution (Kinetic). Both of the PCs had ROS configured to run across both machines, with the base station chosen to run the ROS master (this enables ROS nodes to locate and communicate with each other across machines). Some of the ROS nodes run on the base station PC and some on the onboard PC, depending on their function and physical proximity to the PC. There is no “correct” configuration in this regard, as networked machines are only running one roscore; the specific configuration used will depend on the specific project objectives and functionality. In this case, most nodes were chosen to run on the onboard PC because that is where the relevant hardware is located. In the future, the onboard PC may be chosen as the ROS master, which would allow many of the systems on the robot to continue running in the event of loss-of-signal between the PC and the base station.

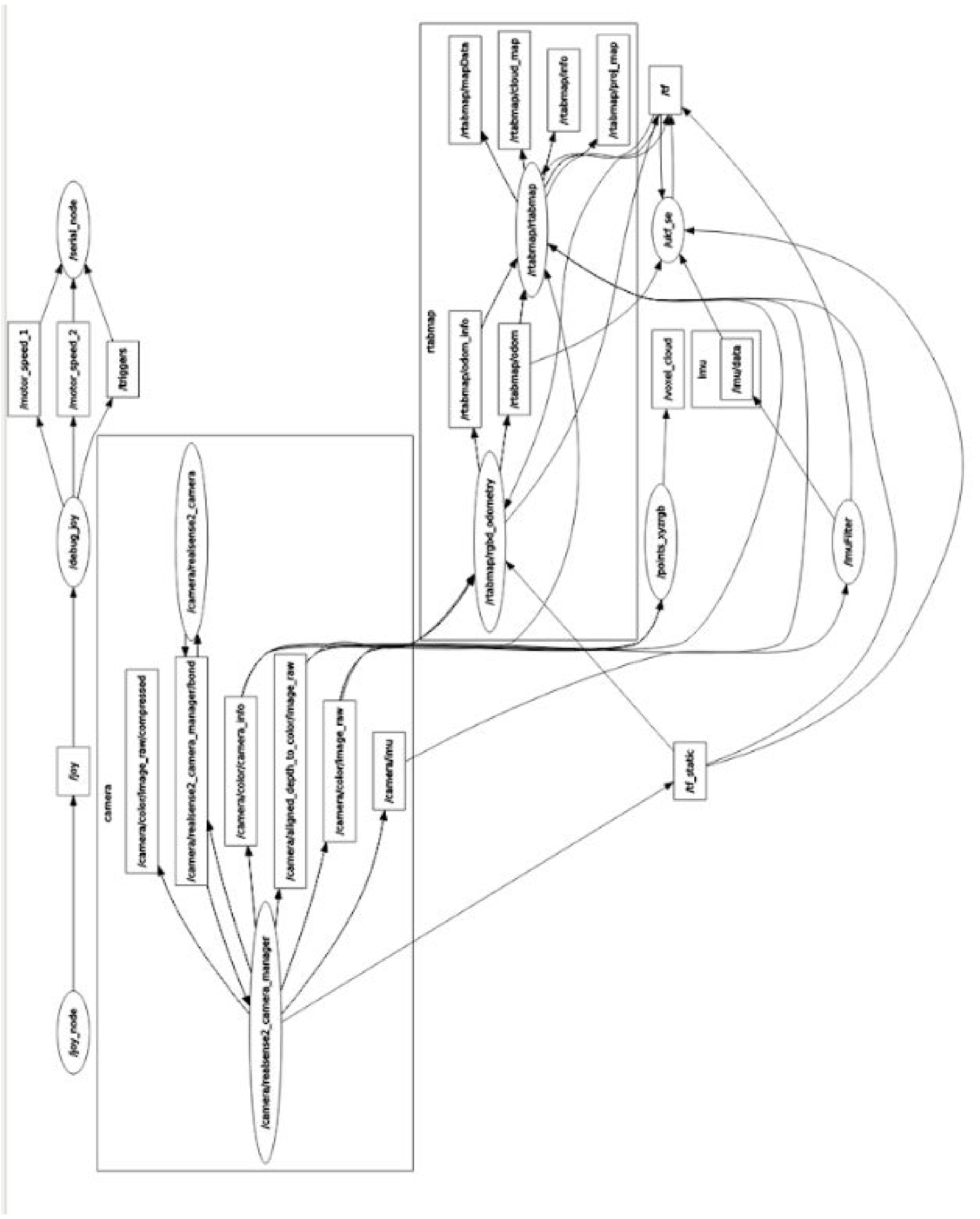
The following nodes run on the base station PC: roscore, joy_node, and RViz. roscore is a collection of nodes that form the prerequisites that a ROS system needs in order to run. The joy_node is part of the ROS joy package, and reads input from the gamepad connected to the base station. The data containing the state of all buttons, triggers, and joysticks is published as a joy message across the /joy topic. RViz is used to display a camera feed and mapping display on the base station.

The onboard PC runs the bulk of the nodes used. The debug_joy node was written by the group and is subscribed to the /joy topic. It reads the specified parameters from the /joy node and converts them to a format that the Arduino code is prepared to recognize; in this case, it scales and converts joystick data (float between 0 and 1) to an integer between 1 and 100. It does this for both the right and left joystick, and then publishes these integers across the topics /motor_speed_1 and /motor_speed_2, respectively. The debug_joy node also scales and converts values from the left and right triggers to create a target position for the camera pan servo, which is published across the topic /triggers. The onboard PC also runs the serial_node from the rosserial_arduino package, which enables the Arduino Mega to interface with ROS and subscribe and publish to ROS topics. The serial_node is subscribed to the /motor_speed_1, /motor_speed_2, and /triggers topics and sends the data contained in the integer messages to the Arduino Mega via serial communication. Using the Arduino libraries provided by rosserial and the motor shield manufacturer, the group wrote an Arduino program that subscribes to these topics through the serial_node. This program reads the joystick commands and sets motor speed through the motor shield and servo position accordingly.

Also running on the onboard PC is a collection of nodes that utilize the Intel RealSense camera. A roslaunch file within the realsense2_camera package (opensource_tracking.launch) is used to launch a few other launch files which do a few different things. First, the rs_camera.launch file is used to start the required nodes that publish several types of camera feed and IMU data across different topics (RGB, depth, infrared, accelerometer, gyroscope). The RGB image is used for a typical camera feed for visual feedback and user navigation. Second, an IMU filter is started to filter the IMU data from the RealSense so that it can be used in mapping (imu_filter_node from the package imu_filter_madgwick). Third, RTAB-Map, an RBG-D SLAM software is launched with rtabmap.launch. This launch file allows for visual 3D mapping of the environment based on camera feeds and odometry from the IMU. RTAB-Map also utilizes RViz to display the map in real time as it is formed.

The onboard PC also runs the sicklms node for the SICK LMS LIDAR, if the user chooses to utilize it. However, the SICK LMS LIDAR is a relatively old model, and the RealSense camera can also be used for 2D mapping. This negates the need for a cumbersome LIDAR that quickly consumes battery power. At the same time, the LIDAR can be used as a secondary system running alongside the RealSense to verify the results, or as a backup in case the camera experiences a failure. It is up to the user to decide whether or not to use the LIDAR.

A schematic provided by the rqt_graph script is included below as a visual aid. The rqt_graph script was run on the ROS master, and n__ prefixes in front of a node denote that the node is running on a networked machine and not the master.


/joy_node
/joy
/debug_joy
/motor_speed_1
/motor_speed_2
/triggers
/serial_node
camera
/camera/realsense2_camera_manager
/camera/color/image_raw/compressed
/camera/realsense2_camera_manager/bond
/camera/realsense2_camera
/camera/color/camera_info
/camera/aligned_depth_to_color/image_raw
/camera/color/image_raw
/camera/imu
rtabmap
/rtabmap/rgbd_odometry
/rtabmap/odom_info
/rtabmap/odom
/rtabmap/rtabmap
/rtabmap/mapData
/rtabmap/cloud_map
/rtabmap/info
/rtabmap/proj_map
/tf
/ukf_se
/points_xyzrgb
/voxel_cloud
imu
/imu/data
/imufilter
/tf_static

User Manual

Disclaimer: this manual includes sections on system setup and system operation. System operation is relatively straightforward, and does not require any specialized knowledge. Currently, the system setup requires the user to have basic knowledge of ROS and some experience using the linux terminal. Compared to Project 1, many of the nodes that run have been condensed into launch files to lessen the amount of setup required.

Setup

1. Begin by making sure all required cables are appropriately connected, i.e. battery is connected, onboard PC power is plugged in, RealSense camera is connected to onboard PC, gamepad connected to base station, etc. Make sure the onboard PC is connected to a monitor via HDMI, this will be necessary to configure ROS.
2. Turn on the base station PC (and boot into Ubuntu if this is not the default), flip the power switch on the robot, and turn on the onboard PC. Log into both machines.
3. On the base station, using the terminal, set the ROS_MASTER_URI to the address and port of the base station (i.e. $ ROS_MASTER_URI=http://team2_2019:11311, change the IP as required). Also on the base station, set the ROS_IP parameter to be the IP of the base station (i.e. $ ROS_IP=192.168.1.24). Note: every time a new terminal window is opened to be used for ROS, remember to source the setup.bash file located in your catkin workspace (in this case, $ source ~/catkin_ws/devel/setup.bash).
4. On the onboard PC, use the terminal to set the ROS_MASTER_URI the same as it was set up on the base station. This allows the onboard PC to recognize the base station as the master. Also on the onboard PC, set the ROS_IP to match the IP of the onboard PC. For more detailed information on these parameters and networking in ROS, visit the ROS Multiple Machines and ROS Network Setup tutorials.
5. In the terminal on the base station, run roscore ($ roscore) to start the ROS master. The network communication between the two machines can be easily verified through the use of rostopic; run ($ rostopic list) in the terminal on the onboard PC to verify that the machine is seeing topics published by the master.

6. Once ROS is running and communication has been verified, a permission needs to be set to utilize joystick input. On the base station, open a new terminal and set the appropriate permission for the joystick input ($ sudo chmod a+rw /dev/input/jsX) where X is the specific joystick (usually 0, but not always).
7. Now several nodes will be started. On the base station, set the joystick parameter for the joy node ($ rosparam set joy_node/dev "/dev/input/jsX") where X is the specified joystick, then start the joy node ($ rosrun joy joy_node).
8. On the onboard PC, open a new terminal and repeat step 4 (this must be done every time a new terminal on the onboard PC will be used for ROS). Start the debug_joy node ($ rosrun network_cam joystick.py).
9. On the onboard PC, open a new terminal and run the rosserial node ($ rosrun rosserial_python serial_node.py /dev/ttyACM0).
10. On the onboard PC, open a new terminal and start the camera, mapping, and IMU filter nodes. ($ roslaunch realsense2_camera opensource_tracking.launch).
11. (optional) On the onboard PC, open a new terminal and set the following parameters for the SICK LIDAR: ($ rosparam set sicklms/port /dev/ttyUSB0) and ($ rosparam set sicklms/baud 38400), then start the LIDAR node ($ rosrun sicktoolbox_wrapper sicklms).
12. Disconnect the HDMI cable from the onboard PC and replace any side panels that were removed to access the components. If the LIDAR is being used, do not replace the front panel, as this would obstruct the view.
13. On the base station, run the GUI using RViz ($ rviz), add a visualizer, and select the desired camera topic for visual feedback (a compressed RGB image is ideal to save bandwidth). If desired, the 3D map being constructed can be displayed by adding PointCloud2 topics. The robot is now ready to operate.

Operation

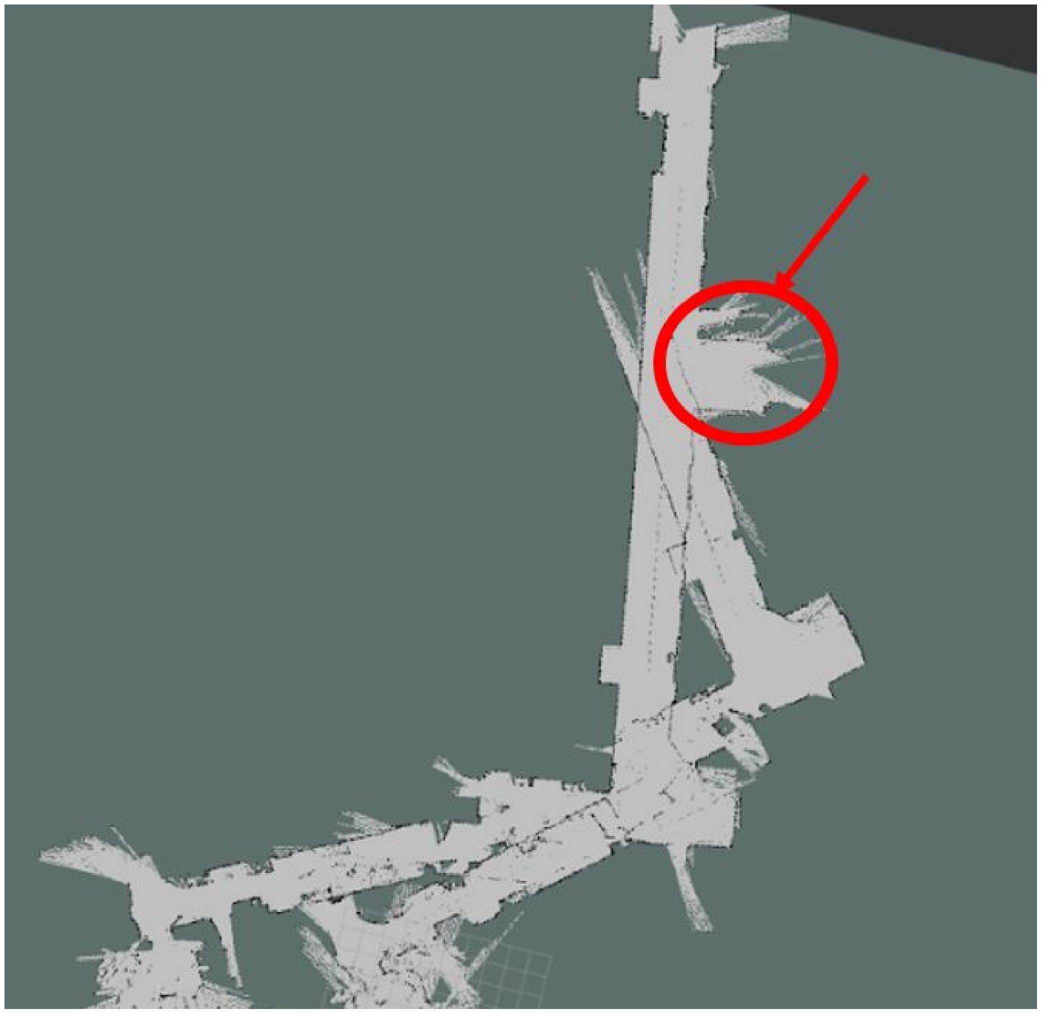
The robot currently has very straightforward driving controls, as it uses a tank drive system: the left and right joysticks control the left and right side motors of the robot, respectively. This type of system was chosen because of its intuitiveness and ease of learning. To move the robot straight forward, both joysticks must be pushed forward by the same amount. To turn, run one side faster than the other (or run the opposite side backwards to rotate in place). To pan the camera, use the left and right triggers on the gamepad (right trigger pans right, left trigger pans left).

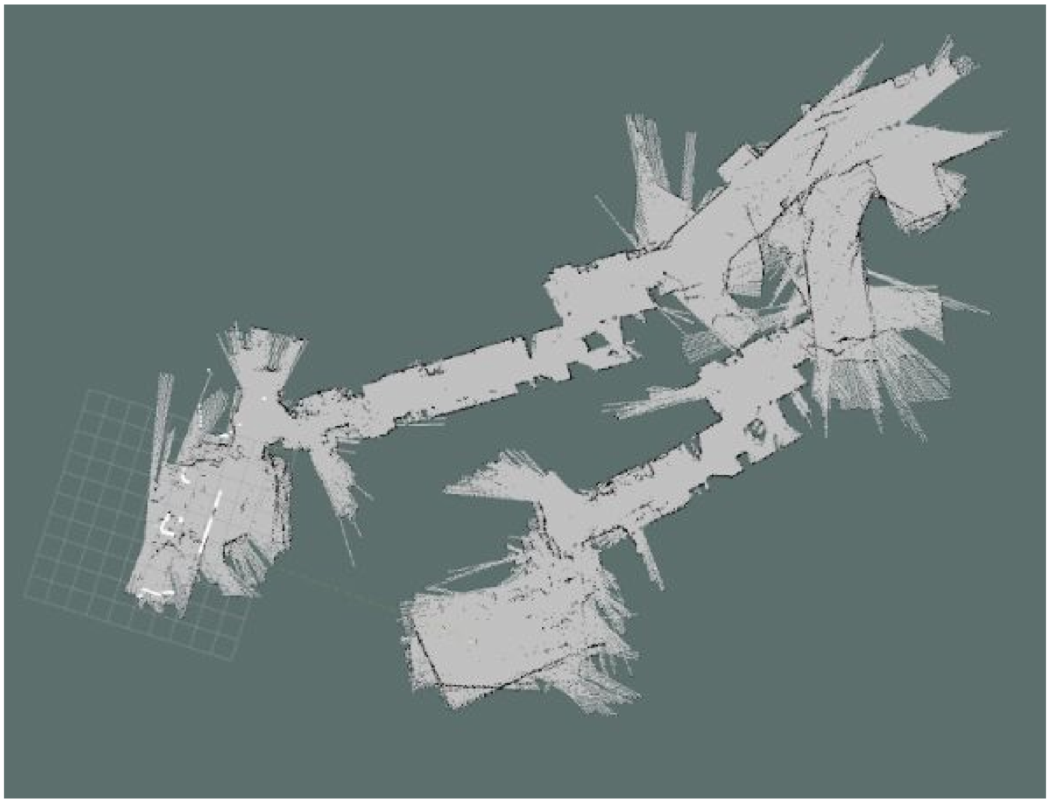
Experimental Validation

Most experimental validation was completed as we tackled each task (LIDAR mapping, 3D mapping, etc.) Unfortunately, on scheduled demo day (May 8), when setting up the system, we encountered an error with the RealSense concerning a version mismatch between the RealSense package and the SDK libraries it uses, and as a consequence we were unable to demo our system. On May 9 we reinstalled the relevant software and updated the camera firmware and got the system functioning correctly. This will be demonstrated on May 13.

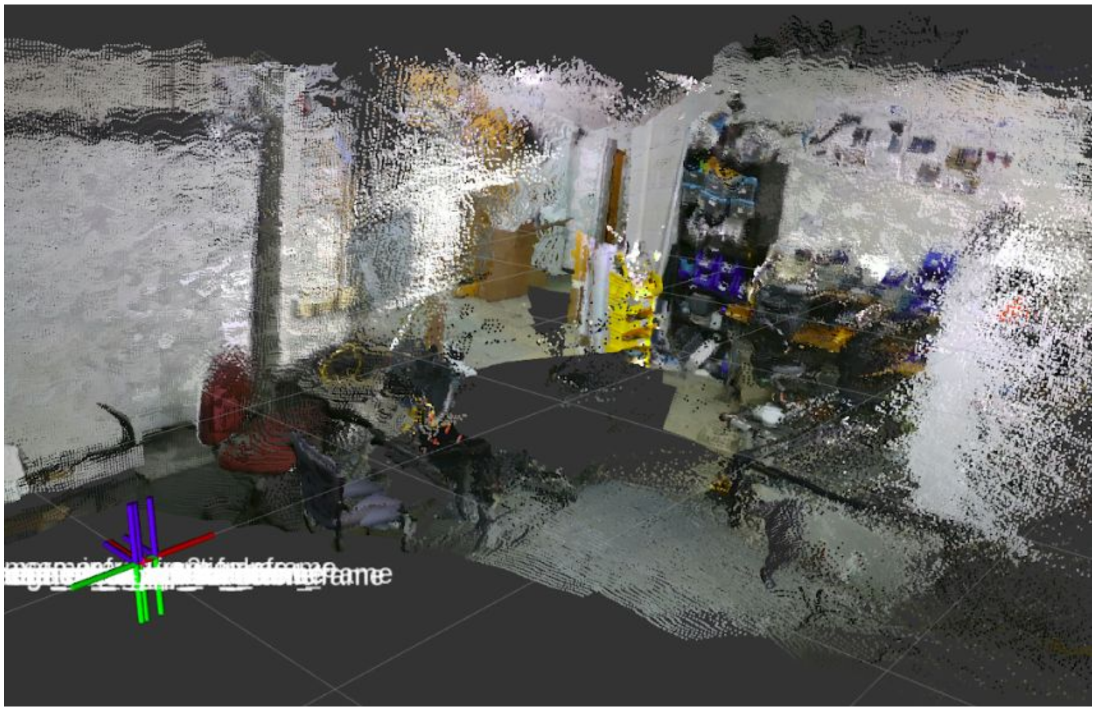
The specified task was to start in the main CMS lab room, drive out of the door, down the hallway, take a left turn and go into the main hallway and into the CMS room onto the right side where a "person" (read: robot) was waiting to receive the cargo of one drink (this was changed from the original path depicted in the image below due to wireless signal strength limits). This was to be done by a user navigating the robot from the base station, through the 2D map of the Randolph basement generated through Hector Mapping and Lidar and send feedback in form arrows directing the final destination of the robot while its traversing, the robot is also required to have place recognition capabilities for which 3D map through Rtabmap is useful and to recognise the person the liquid is to served.

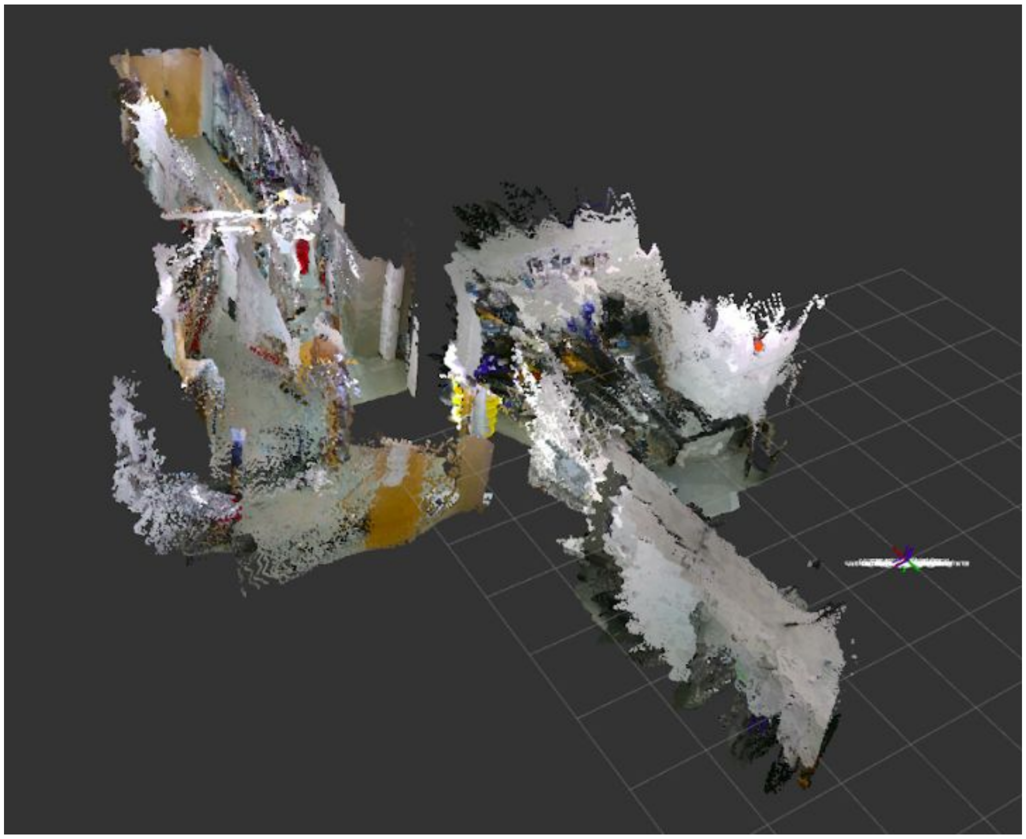
Below you can see all the figures of the continuous project validation that we have been doing throughout the semester, where the first two images represent 2D map of the Randolph basement with hector mapping and SICK LIDAR where you can see the Error marked is the big machining area on the side of the hall, which we are thinking occurs because of the rapid change in distance between the LIDAR and the next wall (since the room is huge and the enclosing wall is farther away compared to other walls in the basement), that the LIDAR loses track of its original position it started mapping from and thus shift the orientation of whole map at an angle. This may potentially be improved by playing around with parameters in the hector mapping script, slowing down our pace of mapping the Randolph basement.





The following two images are of the 3D point cloud created with rtabmap and rgbd realsense camera. The only problem we faced while creating the 3D point cloud was that if the camera was rotated rather quickly the camera's coordinate would lose track of the coordinate of reference and if we keep it moving at that instance, in a very short time (a few seconds) it becomes unlikely that the camera can reorient itself.





Right from the beginning, the group encountered significant delay and low frame rate in the camera feed. However, the robot was still able to be driven from the start position into the hallway and turned 90 degrees to move down the hallway. As the robot was moving down the hallway, the camera feed became more delayed; just outside of the destination room, the image feed was updating very infrequently, about once every 20-30 seconds. Because of this, there was no visual feedback for the user, and the robot was unable to make it to the destination. This is believed to be because of poor network signal strength resulting from an outdated router, which will be updated soon with a modern replacement. Barring the issues arising from signal strength, the robot performed within the groups expectations and did not encounter any other issues.

Even though this was not accounted for in the design, as we expected we would have good signal strength, it does bring the issue of signal strength in the operating environment to attention. Service robots may be desired in locations where the wireless network signal is weak or spotty, and a service robot would be more robust if it could operate in a wider range of conditions, such as a weak network signal. This is an important consideration that the group plans to take into account in future.

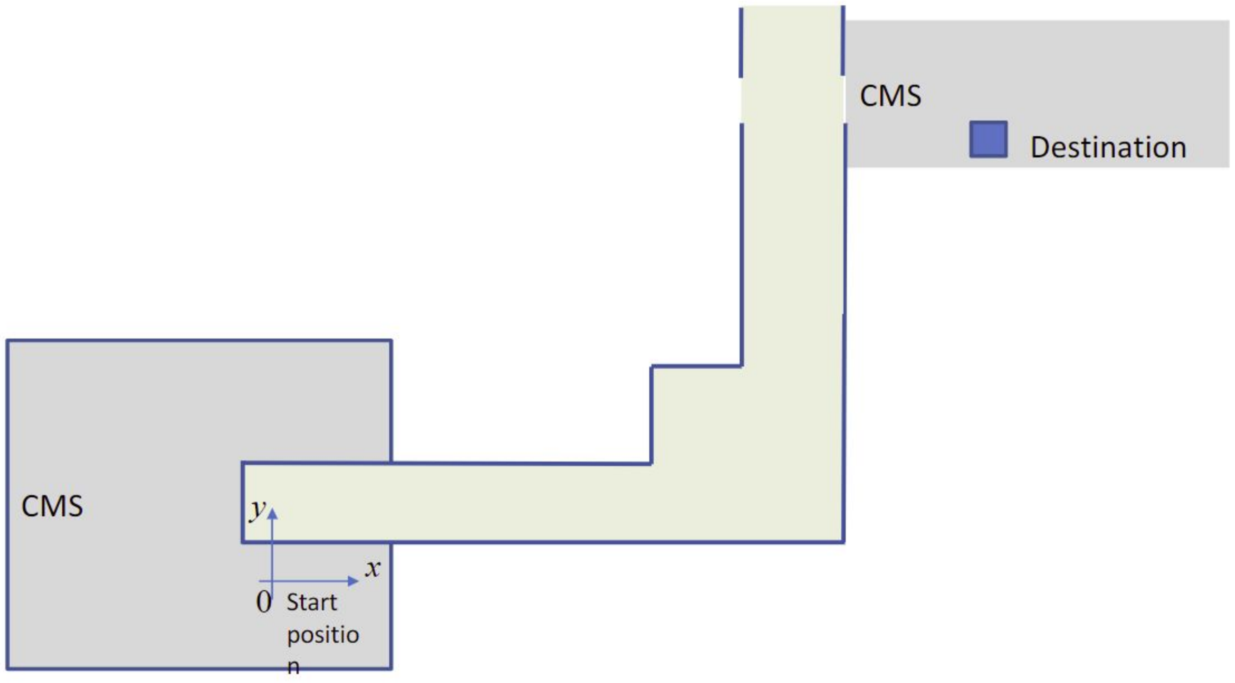


Figure: Original Path for Final Demo

Conclusion

Throughout this course Team 2 started by identifying the customer needs for the project of service robot, some of which were specified by Dr. Furukawa. Using these requirements, the team developed target specifications with which the design needed to satisfy in order to meet customer needs. Finally, the team has completed the mechanical assembly of the system, including all major components, functional elements, and device selection, have setup a communication network between the base station and on-board PC, implemented drive control of UGV using joystick for Project 1 which is continued to be used for the Final project with addition of 2D mapping of Randolph basement using Hector and Lidar, 3D mapping with RTAB-Map and RGB-D camera and finally created a navigation package using navigation stack

Future Work

Going Forward, Team 2 plans to refine the mechanical design, work on aesthetic aspect of the robot to bring it closer to a consumer product, and implement the linear actuator assembly to control height of the catering top. In addition to the mechanical design, the group will also work to finish the rqt GUI running on the base station; it will be expanded to allow for control of the robot and the RealSense camera, while implementing more user-friendly inputs to follow, another important addition is to make the robot fully autonomous by integration of the navigation stack and object recognition so that the robot can reach and serve the final objective without any human interaction.

References

1. https://ozrobotics.com/shop/promobot-v-4/?gclid=Cj0KCQjwkIzlBRDzARIsABgXqV_oclAvRfQblTVamFmym7-NP2fY5ESSH6SNEDYDDeUlwK8OipKMUssaAiTAEALw_wcB
2. https://patents.google.com/patent/US8359122B2/en