Senior Design 2

ROS-enabled Robot – Project Report

Ilya Busaev – UCA Department of Physics & Astronomy

Project Mentor: Dr. William Slaton

Date: May 7, 2021

Table of Contents

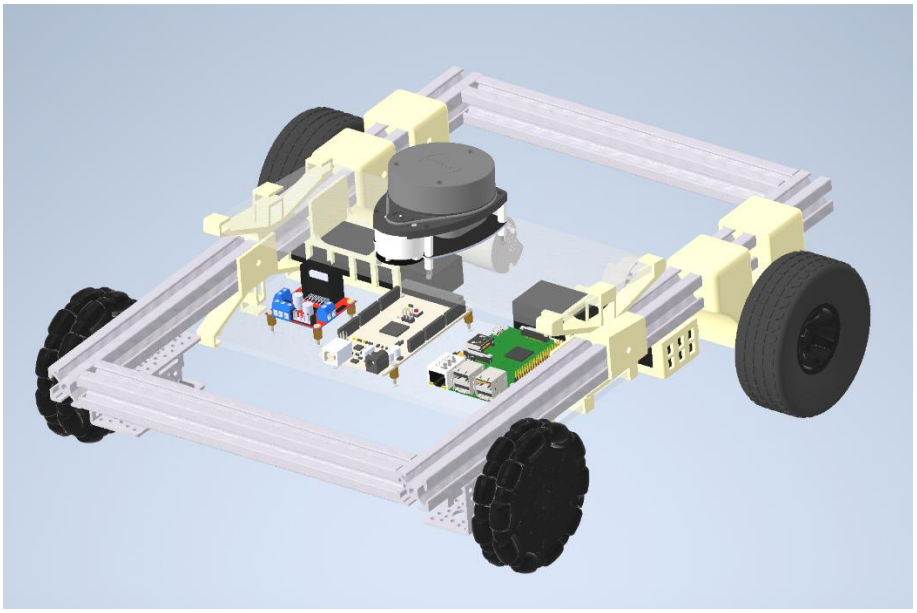
Goal

The goal in the beginning of Spring 2021 semester was to modify the previous Senior Design group's model and successfully launch ROS-enabled robot to autonomously navigate in an outdoor environment participating in NRC-AVC (National Robotics Challenge – Autonomous Vehicle Challenge) competition. However, due to COVID-19 and time limitation, the team was unable to travel to the competition's location and successfully reach the goal, therefore current state of the robot's progress is at 70%, leaving 30% for the next group of Engineering Physics students.

Executive Summary

The first part of Spring 2021 semester was spent on modifying previous model's design and developing hardware to decrease robot's weight, size, while improving its mobility and stabilizing its movement. The second part of the semester mainly focused on establishing communication between components, ROS implementation, and software development.

This report describes current state of the robot as well as work that has been done to get to that point. Moreover, it outlines procedures for building, modifying, and running the robot to make a smooth transition for next year's Senior Design team to build upon current model. There is a GitHub repository that contains ROS packages, CAD files, codes, and other useful documents. The link to that repository will be listed in "*Appendix A*" section.



Result/Analysis

Competition Setup

"For the Autonomous Vehicle Challenge each team will design and build a vehicle to navigate an obstacle course. A successful run is one where the vehicle navigates around the 4 waypoints (yellow stanchions) and crosses the finish line in under 5 minutes. Additionally, bonus points are given for completing special tasks during a run. Once earned, bonus points cannot be taken away. Teams can score bonus points even if they do not complete a successful run". Taken from NRC-AVC competition manual.

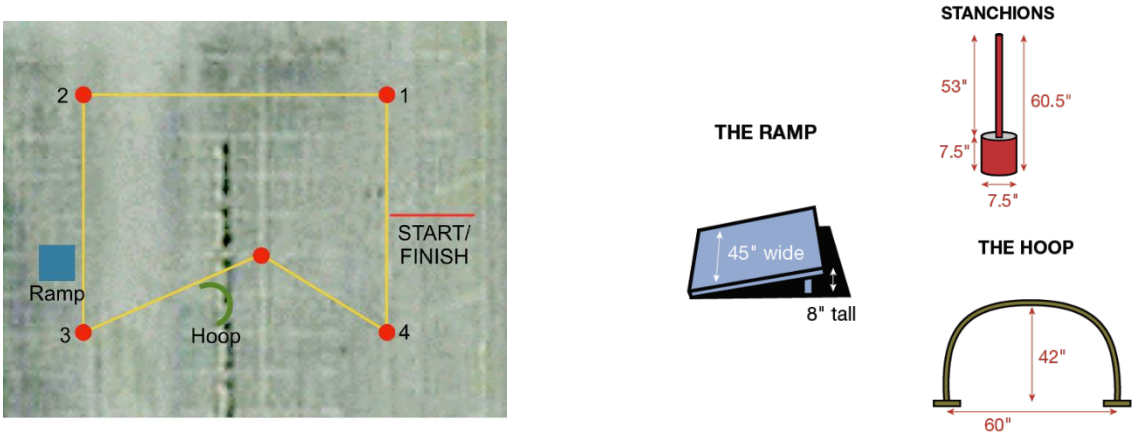


Figure 1: NRC-AVC competition setup

Dimensions of stanchions, the hoop, and the ramp can be check in the competition manual found on the thenrc.org website. The competition manual should be taken from the current year of competition for most recent updates. While all details can be found in the competition manual, there are a few that need to be highlighted:

- Competition will take place in Marion, Ohio on a parking lot area where there is expected to be a rough pavement surface.
- There is a 5-min limit to pass the entire course track from the start to the end line.
- Robot's dimensions are within 24" x 24" x 24" space.
- Physical input has to activate robot's motion (no external laptop participation)

Next year team is expected to compete in this competition, however, it will be determined by Physics & Astronomy Department officers.

Hardware

Previous & Current Design

Previous model of the Senior Design was completely remodified (shown on the left), however, most of the body parts were used to create this model that is shown on the right of the figure below.

Previous design's drawback were due to its heavy weight and instability when moving around. Heavy and chaotic in movements front caster wheel has been replaced with two omni-wheels that will be described in the next section in more detail.

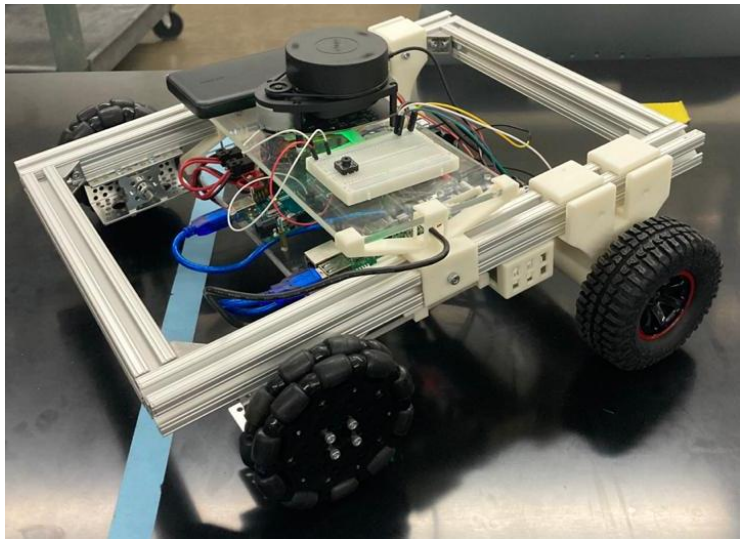


Figure 2: Previous & Current Designs

Dimensions of the previous design, comparing to current design are as follows:

- Previous: 18” x 16” x 12”
- Current: 16” x 16” x 7”

Last dimensions is the representation of height. The reason behind it is because we want to make sure the Lidar scanner that is mounted on top of the robot records a base of a stanchion rather than a poll of a stanchion, since the base is much wider and its height is 7.5” so Lidar Unit can easily scan the entire surface of the stanchion (the concept of Lidar scanning will be described later in the report). Looking at the figure 1, red stanchion’s base height is 7.5” whereas height from the ground to Lidar’s sensor is just about 7 inches which gives plenty of space for the Lidar to scan the surface properly.

With a lighter weight, current robot has the ability of being controlled accurately and therefore navigating in a proper manner.

Wheels

There were several design ideas throughout the semester that the team was considering, however, the final design implementation of two rear wheels of an RC car, and two front omni-wheels were utilized. Both rear and front wheels were substituted with new ones comparing to the previous model, because of the diameter of previous wheels.

Rear tire wheels, diameter of 103 mm, were chosen due to potentially rough surface of the pavement at a parking lot. These tires are good for many outdoor surfaces, including the pavement at the parking lot. Due to weight of the robot and its components, rear tires tend to deform slightly when leaving the robot on the ground for a long time, therefore wooden blocks would serve good as supports.

Front wheels were determined to be omni-wheels. The reason for that is omni-wheels have 360 degree rotating rollers mounted on the outer diameter of wheels, allowing an assembly to perform translational and rotational motion. This is suitable for turning left-right and driving forward-backward on most surfaces. This structure also makes the design least expensive and more stable than with a caster wheel at the front. Two omni-wheels were attached together (on the figure below there’s only one) for a better contact with surface, because the rollers on a single unit have significant space in between them so it doesn’t always touch the ground when in translational motion.

Figure 3: Front Omni-wheels

Motors & Motor Driver

There are two motors that serve as the driving force for the robot. GoBilda 5202 Yellow Jacket Series Planetary Gear motors. With no-load current of 0.25A and stall current of 9.2A, under current conditions the resulting current was found to be around 3.5A.

With weight of 420g both motors result in just about 2 lb providing a stable sand reliable base for the robot. These motors are secured in the motor holders that were 3D printed in the engineering lab at UCA. These motor holders were design by the previous team and serve as a strong base support for the wheels and motors. These can be observed on figure 2 on page 5.

Current Stall Load: 9.2 Amps * 12 Volts = 110 watts

Current No-Load: 0.25 Amps * 12 Volts = 3 watts

With XY-160D motor driver there is a 12V supplied voltage capability for Motors to run properly. This dual motor driver allows to have both motors hooked up and operating at proper specifications. Motor driver is already connected to the Arduino board pins, however, those can be reconfigured in a desired manner. The spec sheet of this motor driver is located in the GitHub repository of Senior Design class (link to this repository is in the **Appendix A**) section.

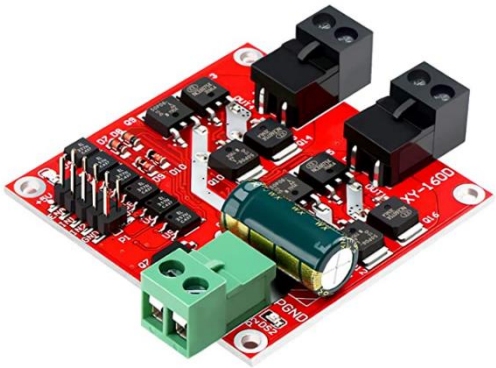


Figure 4: Motors & Motor Driver

Arduino & Raspberry Pi, Ubuntu Laptop

Raspberry Pi Model 3B was used in the design, however, this microcontroller is a personal item so it will have to be replaced with a new one and all necessary programs and packages will have to be installed for it to work properly. First of all, Raspberry Pi has Ubuntu Server 18.05 LTS installed on it, which is a distribution of Linux. This can be installed through a flashed USB or SD card with an Ubuntu image on it (instructions are all over the internet, so nothing special has to be done). This Ubuntu server doesn't have a desktop environment which significantly helps RPi to process information quicker due to its low RAM. The board is mounted onto a plexiglass plate that is attached to the bottom of them aluminum extruded base of the robot.

Figure 5: Raspberry Pi 3B

Moreover, as the main operating Ubuntu laptop a personal laptop was used, therefore a new laptop will have to be used for this project and all necessary packages will have to be reinstalled, including ROS and all subsequent packages.

Ubuntu Laptop is serving with the same purposes as Raspberry Pi, except it provides more gui tools for visualizations and simulations. By 'ssh'-ing into Raspberry Pi we are able to see the scanning and navigating processes while the robot is moving. SSH is a command that allows to access a computer through another machine and operate virtually on that another machine.

Lastly, Arduino Mega 2560 was used as a main hardware communication microprocessor. Arduino is a better hardware communicating microcontroller which already has established mechanisms and algorithms to control motors and encoders. With Arduino IDE installed onto a machine (in our case it is an Ubuntu Laptop) we are able to compile a code necessary for communication and recording of data. This will be described in details in Software section.

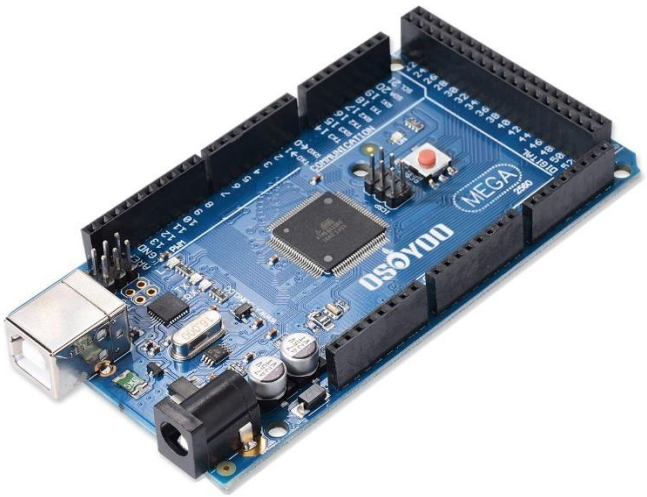


Figure 6: Arduino Mega 2560

Lidar & Lidar Plate Holders

Lidar is a device that uses a laser measurement system to determine the distance from the object and plot the environment that the robot is currently in. This device allows the robot to create a map and store it in Raspberry Pi memory for it to navigate around a specified course track. Lidar is mounted on top of the robot at a height of around 7 inches. For the scanner to take proper measurements it has to be above all components without any objects blocking its scanning field.

Lidar A1 scanner was used for this robot, with a range of 12 meters which is enough to determine a 6 meter-radius range scanning field. This unit is mounted on a plexiglass plate that

is attached to an aluminum extruded base through lidar plate holders, designed in a software modelling application – AutoDesk Inventor.

Lidar Plate Holders are shown on the left picture of figure 7. They are designed to complement an inclined robot's surface that arose due to the height difference between motor holders and front wheel assembly plates. The angle resulted to be around 6 degrees for the lidar to be close to perfectly-horizontal. Lidar Plate Holders designed in AutoDesk Inventor and printed out using a 3D Printer in the engineering lab at UCA.

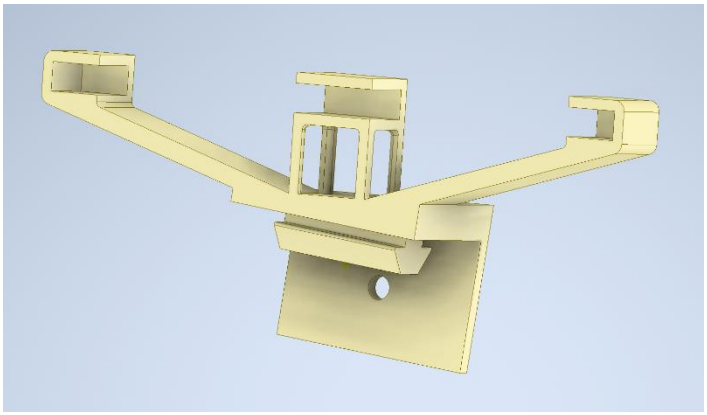


Figure 7: Lidar Scanner & Lidar Plate Holders & Robot Side-view

Power Supplies

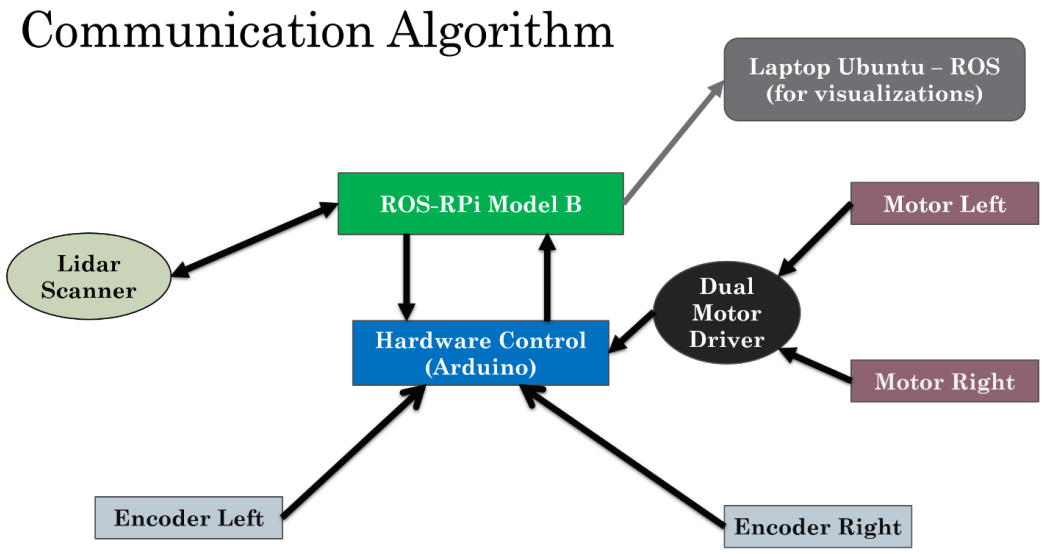
There are two power supplies that are necessary to provide enough current for motors, Raspberry Pi, Motor Driver, and Arduino. Two 12V batteries connected in parallel provide 12V voltage and sufficient current to drive motors through a Motor controller board (motor driver).

Anker Power Bank of 10,000 mAh capacity is enough to supply Raspberry Pi model 3B and Arduino Mega 2560 boards.

Communication Concept

Microcontroller boards use a publish-subscribe communication type to send and receive information from each other and communicate to actuators and sensors. Usually the sensors in a robot produce a stream of data at a regular interval (laser readings, ticks from wheel encoders) and you want the robot to react in some way to the information coming from the sensors whenever new information is available. In that sense, a publisher/subscriber communication pattern comes quite handy. A node publishes the information from a sensor whenever a new reading is available and any node interested in those updates can subscribe to the sensor topic.

Through such communicating protocol sensors and actuators are able to send and receive messages allowing to perform such operations. Raspberry Pi 'talks' to Arduino through a so called *rosserial protocol* that is a way to connect both boards and allow them to communicate, sending and receiving messages from subscribed topics.



Software

Motors & Encoders

While Encoders communicate directly to Arduino, Motors communicate through XY-160D motor driver, which supplies enough voltage and provides enough current to motors. There is a basic code for driving motors that I have compiled, however, there's not one for encoders with established communication.

This code is located in the GitHub repository, named "Motors_Test_Code". It allows the motors to drive in a desired direction with user established speed by varying the speed factor. The code is very simple, allowing to test the ability of motors to drive the robot. This setup doesn't account for publishing and subscribing the odometry data.

Arduino-ROS

Arduino is communicating with ROS using *rosserial_protocol*. Basically, this protocol is utilized for wrapping standard ROS messages, topics, and services over a serial port or network socket. To read more on rosserial, follow the link http://wiki.ros.org/rosserial.

To install rosserial node on Arduino and Ubuntu follow the steps below:

Note: rosserial protocol is already installed on currently used Arduino board, however, if the board needs to be replaced, this would be a guide for reinstalling it.

Installing rosserial packages on Ubuntu

We can install the rosserial packages on Ubuntu using the following commands:
1. Installing the rosserial package binaries using apt-get:

- In Melodic:
  $ sudo apt-get install ros-melodic-rosserial
  ros-melodic-rosserial-arduino ros-melodic-rosserial-server

2. For installing the rosserial_client library called ros_lib in Arduino, we have to download the latest Arduino IDE for Linux 32/64 bit. Following is the link for downloading Arduino IDE:
https://www.arduino.cc/en/main/software

Here we download the Linux 64 bit version and copy the Arduino IDE folder to the Ubuntu desktop.

3. Arduino requires JAVA runtime support to run it. If it is not installed, we can install it using the following command:

$ sudo apt-get install java-common

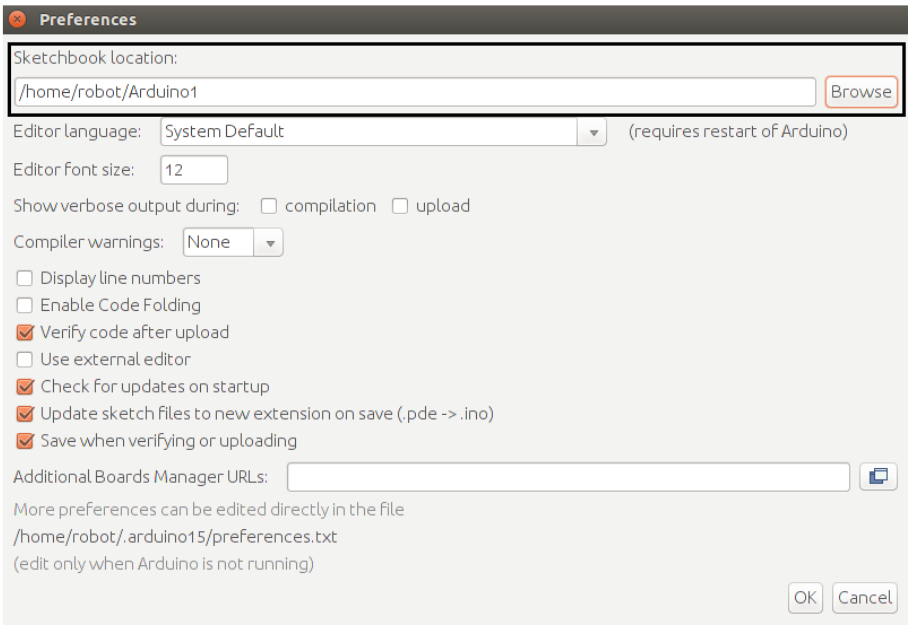
4. After installing JAVA runtime, we can switch the arduino folder using the following command:

$ cd ~/Desktop/arduino-1.6.5
(depending on the version that you install the name of the folder will be different)

5. Start Arduino using the following command:

$ Arduino

6. Go to **File** | **Preference** for configuring the sketchbook folder of Arduino. Arduino IDE stores the sketches to this location. We created a folder called Arduino1 in the user home folder and set this folder as the sketchbook location.



7. We can see a folder called libraries inside the Arduino1 folder. Switch to this folder using the following command:

$ cd ~/Arduino1/libraries/

If there is no libraries folder, we can create a new one.

8. After switching into this folder, we can generate ros_lib using a script called make_libraries.py, which is present inside the rosserial_arduino package. ros_lib is rosserial_client for Arduino, which provides the ROS client APIs inside an Arduino IDE environment.

$ rosrun rosserial_arduino make_libraries.py

rosserial_arduino is ROS client for arduino which can communicate using UART and can publish topics, services, TF, and such others like a ROS node. The make_libraries.py script will generate a wrapper of the ROS messages and services which optimized for Arduino data types. These ROS messages and services will convert into Arduino C/C++ code equivalent, as shown next:

• Conversion of ROS messages:

ros_package_name/msg/Test.msg --> ros_package_name::Test

• Conversion of ROS services:

ros_package_name/srv/Foo.srv --> ros_package_name::Foo

For example, if we include #include <std_msgs/UInt16.h> , we can instantiate the std_msgs::UInt16 number.

If the script make_libraries.py works fine, a folder called ros_lib will generate inside the libraries folder. Restart the Arduino IDE and we will see ros_lib examples inside the sketchbook folder.

There might be an issue with **Uploading** the code onto your Arduino board. There might be an error message displayed in Arduino IDE, something like this:

avrdude: ser_open(): can't open device "/dev/ttyACM0": Permission denied

If above in the previous step happened, resolve it by typing in the new terminal on Ubuntu laptop:

$ *sudo chmod a+rw /dev/ttyACM0*

Restart Arduino IDE and upload your code again. Should work fine.

Step-by-Step for Publishing Odometry:

After connecting Arduino to RPi or Laptop always type the following command in the terminal:

$ *sudo chmod a+rw /dev/ttyACM0*

Upload Code onto Arduino (if it is not there) which is located in the *libraries* of Arduino IDE, called "Odometry_Publisher"

Run roscore:

$ *roscore*

Run *serial node* (it is already installed in Arduino and Ubuntu Laptop and/or RPi):

$ *rosrun rosserial_python serial_node.py /dev/ttyACM0*

Start Rviz and "Add" **TF.** You will see a robot (if robot URDF is not uploaded you will not see robot's model but the movement will still occur) moving in a circle.

Lidar-ROS

Lidar is connected to the Raspberry Pi model 3B or Ubuntu laptop. Basically, Lidar has to be connected to a machine with Ubuntu and ROS installed on it. There is a package called rplidar_ros that is located in the GitHub repository that contains all necessary packages for the Lidar unit to navigate properly. There is another set of nodes called hector_slam that will have to be installed on Ubuntu laptop for navigating purposes. This package can be installed from the following github repository https://github.com/tu-darmstadt-ros-pkg/hector_slam.

One notable issue: a USB-to-microUSB cable was incompatible and not powerful enough to power the Lidar, therefore a communication issue arised. Undervoltage was caused by a USB-

to-MicroUSB cable. Replacing the cable fixed the issue and allowed a computer to read rplidar at a USB port. Make sure a proper cable is used when working with Lidar and ROS machine.

There are a couple of issues that arose along the way while installing those packages. After uploading these github repositories, catkin_make command will have to be performed in order to **build** these packages and link all components of those for the system to read it properly. As outlined in many instructional pages catkin_make command should build all of the packages that are inside the src folder of catkin workspace folder. However, due to an unknown at this moment configuration issue, the system only builds one package at a time so in order to solve this problem, perform the following command in the terminal window and it should resolve the issue and build all necessary packages individually:

$ catkin_make --only-pkg-with-deps hector_slam

(if desired "hector_slam" can be changed to a package name that needs to be build)

To make the scanning simulation perform the following:

1) Connect RPlidar with an extension USB cord (black cable located on the rolling cart)

2) Open up a Terminal Window on a Laptop:

   - Type in a terminal window:
     $ *sudo chmod 666 /dev/ttyUSB0*

   - Type in:
     $ *roslaunch rplidar_ros rplidar.launch*

   - Open NEW terminal and Type in:
     $ *roslaunch hector_slam_launch tutorial.launch*

(This opens-up Rviz and starts mapping out the environment, space around it)

On the figure below is a snapshot of the simulation and map that this Lidar is creating, visualized in rViz.

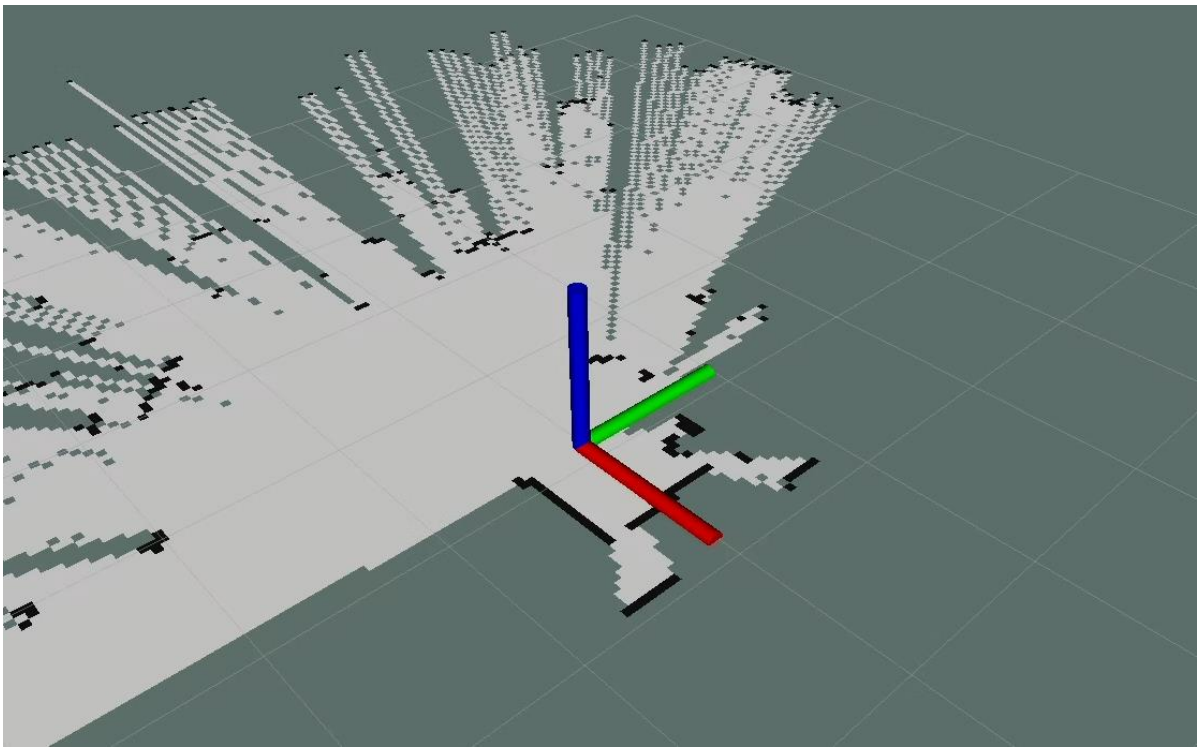


Figure 8: Scanning & Mapping Simulation

Next Steps

Current robot's model is a stable and mobile structure, which allows to obtain accurate scanning data and leave motion of the robot predictable and reliable. However, there are several modification that can be done in order to make the robot perform better. Substituting aluminum extruded with plastic or any other lighter material bars would significantly decrease robot's weight but will still provide a solid structure. Rear wheels may be upgraded more rigid ones due to wheels' tires deformation when robot is stationary for long periods of time.

As for software, there's plenty of work that this year's team have achieved. There's a code in the GitHub repository in Arduino sketchbook called "EncoderCode-Unfinished" which was compiled combining Nox robot's code and a code from James Bruton's github page (all githubs and codes are referenced).

There are also test codes for driving motors and publishing odometry messages that are located in the github repository. A code for publishing odometry establishes communication between Arduino and a ROS machine, where it is an Ubuntu Laptop or a RPi board.

All CAD files are located in the GitHub repository in order to recreate any of robot's components that may potentially break or wear out or in order to modify anything.

Conclusion

This year has brought us a lot of uncertainties due to COVID-19, but we, as a team, managed to operate and achieve goals under these conditions and completed most of the project. The progress this semester with this robot allows the team of the next year to have an advantage of already working and built robot's model with established basic communication between components. This communication will have to be looked thoroughly into in order to have the units communicate between each other properly. With started but unfinished code for Arduino to publish and subscribe to necessary topics, more work will have to be performed in order to achieve desired goal for this project – autonomous navigation in an outdoor environment.

All files are located in GitHub repository (the link is below in Appendix A) rather than including all files in this report, which is a more organized manner.

Appendix A

All codes can be found in the GitHub repository, but all of them should already be installed on Arduino. The Link to GitHub page of the project is below:

https://github.com/UCASeniorDesign/ROS-Autonomous-Spring-2021

References

[1] “Mastering ROS for Robotics Programming”, Lentin Joseph

PDF, ISBN 978-1-78355-179-8

[2] “Learning ROS for Robotics Programming”, Aaron Martinez, Enrique Fernandez

PDF, ISBN 978-1-78216-144-8

[3] “Nox Robot”, ROS Wiki page

http://wiki.ros.org/Robots/Nox

[4] “Rosserial Node”, ROS Wiki page

http://wiki.ros.org/rosserial

[5] “XRobots, James Bruton”, GitHub Repository

https://github.com/XRobots

[6] “Arduino with ROS”, ROS Wiki page

http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup