

Autonomous Wheelchair

**Department of Robotics & Mechatronics Engineering**

**Critical Design Review**

Max Bronson, Denny Mannakulathil, Oscar Montealegre

Dr. Chan Ham, Prof. Razvan Voicu

Mechatronics System Design- Fall 2021

Mechatronics Engineering

Kennesaw State University

10/11/2021

# Executive Summary

This report is used to summarize the development of the Automated Wheelchair project, and the major work conducted over the Fall 2021 semester. The system implements Python, C++, and ROS to process data provided from both a Velodyne VLP-16 LiDAR Module & a Microsoft Kinect V2. Using this processed data, serial commands are sent to an Arduino MEGA 2560 which sends a variable voltage to the electric motors on the rear wheels of the wheelchair. The system is to be capable of point-to-point navigation without human intervention, although the user can provide direction commands through the use of a joystick. The system implements object avoidance to prevent collision with obstacles that may pass through the optimal path calculated by the wheelchair.

**Table of Contents**

[Executive Summary 2](#_Toc89202969)

[1. Introduction 7](#_Toc89202970)

[1.1. Project Overview - Montealegre 7](#_Toc89202971)

[1.2. Major Objectives & Developments - Montealegre 7](#_Toc89202972)

[1.3. Team Members & Major Responsibilities - Mannakulathil 7](#_Toc89202973)

[2. System Requirements & Specifications 8](#_Toc89202974)

[2.1. Technical Requirements & Specifications - Montealegre 8](#_Toc89202975)

[2.1.1. Dimensions 9](#_Toc89202976)

[2.2. Functional Features - Montealegre 9](#_Toc89202977)

[2.3. Minimum Success Criteria - Bronson 9](#_Toc89202978)

[2.4. Design Verification Plan - Bronson 10](#_Toc89202979)

[3. System Overview 12](#_Toc89202980)

[3.1. Configuration: Block Diagram & Description – Montealegre 12](#_Toc89202981)

[3.2. Major Subsystems & Components - Montealegre 13](#_Toc89202982)

[3.3. Trade Studies - Montealegre 14](#_Toc89202983)

[3.3.1. Microcontrollers 14](#_Toc89202984)

[3.3.2. Sensors 15](#_Toc89202985)

[3.3.3. Motors 15](#_Toc89202986)

[3.3.4. Sensor Battery 16](#_Toc89202987)

[3.3.5. Wheelchair Frame 17](#_Toc89202988)

[4. Prototype Development 17](#_Toc89202989)

[4.1. Major Works - Montealegre 17](#_Toc89202990)

[4.1.1. Major Parts 18](#_Toc89202991)

[4.2. Hardware - Mannakulathil 18](#_Toc89202992)

[4.2.1. CAD of Sensor Mounts 18](#_Toc89202993)

[4.2.2. CAD of Sensors Electrical Box 22](#_Toc89202994)

[4.2.3. CAD of Miscellaneous Parts 24](#_Toc89202995)

[4.2.4. Circuit Diagrams - Bronson 25](#_Toc89202996)

[4.3. Software - Bronson & Montealegre 26](#_Toc89202997)

[4.4. Budget – Mannakulathil 27](#_Toc89202998)

[4.4.1. Budget 27](#_Toc89202999)

[4.4.2. Bill of Materials (BOM) 28](#_Toc89203000)

[4.5. Project Schedule - Bronson 29](#_Toc89203001)

[4.6. Analysis – Bronson & Montealegre 29](#_Toc89203002)

[4.6.1. Operation Time and Power Draw 29](#_Toc89203003)

[4.6.2. Testing Path-planning 29](#_Toc89203004)

[4.6.3. Optimization of Obstacle Detection & Avoidance 30](#_Toc89203005)

[4.6.4. Sensor Data Accuracy Check 30](#_Toc89203006)

[4.7. Navigation in a Virtual Environment- Bronson 31](#_Toc89203007)

[4.8. Testing - Bronson 32](#_Toc89203008)

[4.8.1. Sensor Calibration Testing 32](#_Toc89203009)

[4.8.2. Motor Response Testing 32](#_Toc89203010)

[4.8.3. Obstacle Avoidance Testing 32](#_Toc89203011)

[5. Conclusion 32](#_Toc89203012)

[5.1. Achievements & Lessons Learned - Mannakulathil 32](#_Toc89203013)

[5.2. Future Improvements & Optimizations - Bronson 33](#_Toc89203014)

[5.3. Team Chart/Assignments - Mannakulathil 34](#_Toc89203015)

[References - Montealegre 34](#_Toc89203016)

[Resources - Mannakulathil 35](#_Toc89203017)

[Appendices - Bronson 36](#_Toc89203018)

[Appendix A: Velodyne VLP-16 36](#_Toc89203019)

[Appendix B: Microsoft Kinect V2 38](#_Toc89203020)

[Appendix C: Arduino 2560 39](#_Toc89203021)

[Appendix D: Bioenno Lithium-Ion Battery 40](#_Toc89203022)

[Appendix E: Project Drawings 41](#_Toc89203023)

[Appendix F: Electrical Diagrams 45](#_Toc89203024)

[Appendix G: Project Programming Scripts 46](#_Toc89203025)

[Appendix H: Team Contributions Table 63](#_Toc89203026)

**Table of Figures**

[**Figure 1:** System Block Diagram 12](#_Toc89203027)

[**Figure 2:** Drawing of LiDAR Mount Design 19](#_Toc89203028)

[**Figure 3:** Drawing of Kinect Mount Design 20](#_Toc89203029)

[**Figure 4:** Drawing of Handlebar Mount Design 21](#_Toc89203030)

[**Figure 5:** Drawing of Sensors Assemblage Electrical Box 22](#_Toc89203031)

[**Figure 6:** Drawing of Sensor Assemblage Electrical Box Lid 23](#_Toc89203032)

[**Figure 7**: Drawing of Joystick Mount 24](#_Toc89203033)

[**Figure 8:** Drawing of Arduino Housing Lid 25](#_Toc89203034)

[**Figure 9:** Digital Signals Circuit Diagram 26](#_Toc89203035)

[**Figure 10:** Sensor Power Circuit Diagram 26](#_Toc89203036)

[**Figure 11:** Gantt Chart of Project Schedule 29](#_Toc89203037)

[**Figure 12:** Visualization of Localization & Mapping 31](#_Toc89203038)

[**Figure 13:** VLP-16 Data Sheet 1 36](#_Toc89203039)

[**Figure 14:** VLP-16 Data Sheet 2 37](#_Toc89203040)

[**Figure 15:** Microsoft Kinect V2 Rendering 38](#_Toc89203041)

[**Figure 16:** Microsoft Kinect V2 Data Sheet 38](#_Toc89203042)

[**Figure 17:** Arduino 2560 Rev3 Wiring Diagram 1 39](#_Toc89203043)

[**Figure 18:** Arduino 2560 Rev3 Wiring Diagram 2 39](#_Toc89203044)

[**Figure 19:** Arduino 2560 Rev3 Technical Specifications 40](#_Toc89203045)

[**Figure 20:** Properties of Bioenno Battery 40](#_Toc89203046)

[**Figure 21:** Drawing of LiDAR Mount Design 41](#_Toc89203047)

[**Figure 22:** Drawing of Kinect Mount Design 42](#_Toc89203048)

[**Figure 23:** Drawing of Handlebar Mount Design 43](#_Toc89203049)

[**Figure 24:** Drawing of Sensors Assemblage Electrical Box 43](#_Toc89203050)

[**Figure 25:** Drawing of Sensor Assemblage Electrical Box Lid 44](#_Toc89203051)

[**Figure 26**: Drawing of Joystick Mount 44](#_Toc89203052)

[**Figure 27:** Drawing of Arduino Housing Lid 45](#_Toc89203053)

[**Figure 28:** Digital Signals Circuit Diagram 45](#_Toc89203054)

[**Figure 29:** Sensor Power Circuit Diagram 46](#_Toc89203055)

**List of Tables**

[**Table 1:** Wheelchair Dimensions 9](#_Toc89203056)

[**Table 2:** Verification Approach & Plan 11](#_Toc89203057)

[**Table 3:** Linux Machine Trade Study 14](#_Toc89203058)

[**Table 4:** LiDAR Sensor Trade Study 15](#_Toc89203059)

[**Table 5:** Sensor Battery Trade Study 16](#_Toc89203060)

[**Table 6:** Wheelchair Trade Study 17](#_Toc89203061)

[**Table 7:** Major Parts 18](#_Toc89203062)

[**Table 8:** Project Budget 27](#_Toc89203063)

[**Table 9:** Bill of Materials 28](#_Toc89203064)

[**Table 10:** Available Resources 35](#_Toc89203065)

[**Table 11:** Team Contributions 63](#_Toc89203066)

# Introduction

## 1.1. Project Overview - Montealegre

The Autonomous Wheelchair is a system that utilizes Python, C++, and ROS programs to autonomously perform point-to-point navigation while dynamically avoiding static and mobile obstacles. The system implements a Velodyne VLP-16 as its LiDAR sensor & a Microsoft Kinect V2 as its RGB & Infrared sensors; utilized to collect data about the wheelchair’s environment. Data processing, generation of serial commands and other computations are performed by a Huawei Matebook X Pro. These serial commands are then transferred to an Arduino Mega 2560, which then translates the commands into corresponding voltages, effectively operating the motors.

## 1.2. Major Objectives & Developments - Montealegre

The general objective of this project is to develop a system that allows a user to autonomously navigate through the world, with the target destination as the only required input from the user.

Throughout its development, the “Autonomous Wheelchair” project has required mechanical, electrical & software developments, this includes:

* The 3D design and assembly of the sensors assemblage’s mounts & electrical box, the digital logic assemblage’s electrical box, and a few other miscellaneous parts
* The attachment of an additional battery onto the wheelchair’s structure, and the wiring that allows the transfer of power to the sensors; this wiring utilizes a 20 amps fuse, a 24V → 12V power converters & three SPST switches
* The utilization of a purchased mount to attach the laptop to the wheelchair; this mount is composed of a set of links that attach the laptop platform to a c-clamp.
* The digital logic assemblage (Arduino Mega & wheelchair motors)
* The development of a central python navigation script, as well as a SLAM & path planning, an obstacle detection & avoidance script, and an Arduino script. Also, the interconnection of all programs, which allows us to launch all programs in unison, in order to perform all features.

## 1.3. Team Members & Major Responsibilities - Mannakulathil

Max Bronson has been the team leader throughout the project, as well as one of the programming leads. He oversees the LiDAR interfacing, ROS, and Arduino Programming. Denny Mannakulathil oversees 3D design, which makes him responsible for the modeling, drawing, and 3D printing of all parts needed. Oscar Montealegre is the other programming lead, focusing on Kinect Interfacing, object identification & avoidance.

# System Requirements & Specifications

The following section shows the physical composition of the system, as well as the functions and capabilities of the wheelchair:

## 2.1. Technical Requirements & Specifications - Montealegre

Based on the input of a target destination, the wheelchair is capable of autonomous navigation to points at any distance from its current location, moving an average pace of at least 3 mph. The wheelchair will begin planning a path to the desired destination while simultaneously moving toward such desired destination. The wheelchair is capable of maneuvering around corners, obstacles and through entryways on the planned path by modifying its originally planned path until reaching the desired destination.

### 2.1.1. Dimensions

The following table provides a dimensional analysis of the system upon completion:

**Table 1:** Wheelchair Dimensions

|  |  |
| --- | --- |
| **Part** | **Model** |
| Seat Height | 19 in. |
| Seat Depth | 19 in. |
| Back Height | 36 in. |
| Width | 23 in. |
| Rear Wheel Radius | 4.75 in. |
| Front Wheel Radius | 3 in. |
| Wheelbase | 19 in. |

## 2.2. Functional Features - Montealegre

Navigation from the initial point is accomplished autonomously and without any user intervention. Throughout the wheelchair’s journey to the target destination, the path-planning algorithm is capable of adjustment as new data about the environment is collected by the sensors. The system implements SLAM to both map its environment and localize its current location in a virtual representation of the world. The obstacle avoidance algorithm can detect both stationary and moving obstacles; if an object is detected, the wheelchair stops and waits for the object to be removed from its path. The user is also able to provide directional input through the joystick on the right armrest.

## 2.3. Minimum Success Criteria - Bronson

For the system to be considered a success, a series of requirements must be achieved. The wheelchair must navigate to a target destination with an error of less than or equal to 4 feet. This means that the wheelchair must be less than 4 feet from the target when navigation is complete. Secondly, the wheelchair must avoid walls and stationary objects by not coming closer than 2 feet to the obstacle. The wheelchair must be capable of coming to a stop whenever an obstacle crosses the planned path, near the wheelchair’s current location. Finally, the wheelchair is to be able to navigate to a point at least 50 feet away at an average rate greater than 3 mph.

## 2.4. Design Verification Plan - Bronson

To ensure the wheelchair system functions properly, a series of tests and inspections must be conducted; these are to be conducted throughout development and before every operation of the system. Below is an outline of these tests and instructions on how to properly inspect the system to ensure successful operation:

Proper testing of the system prior to running includes:

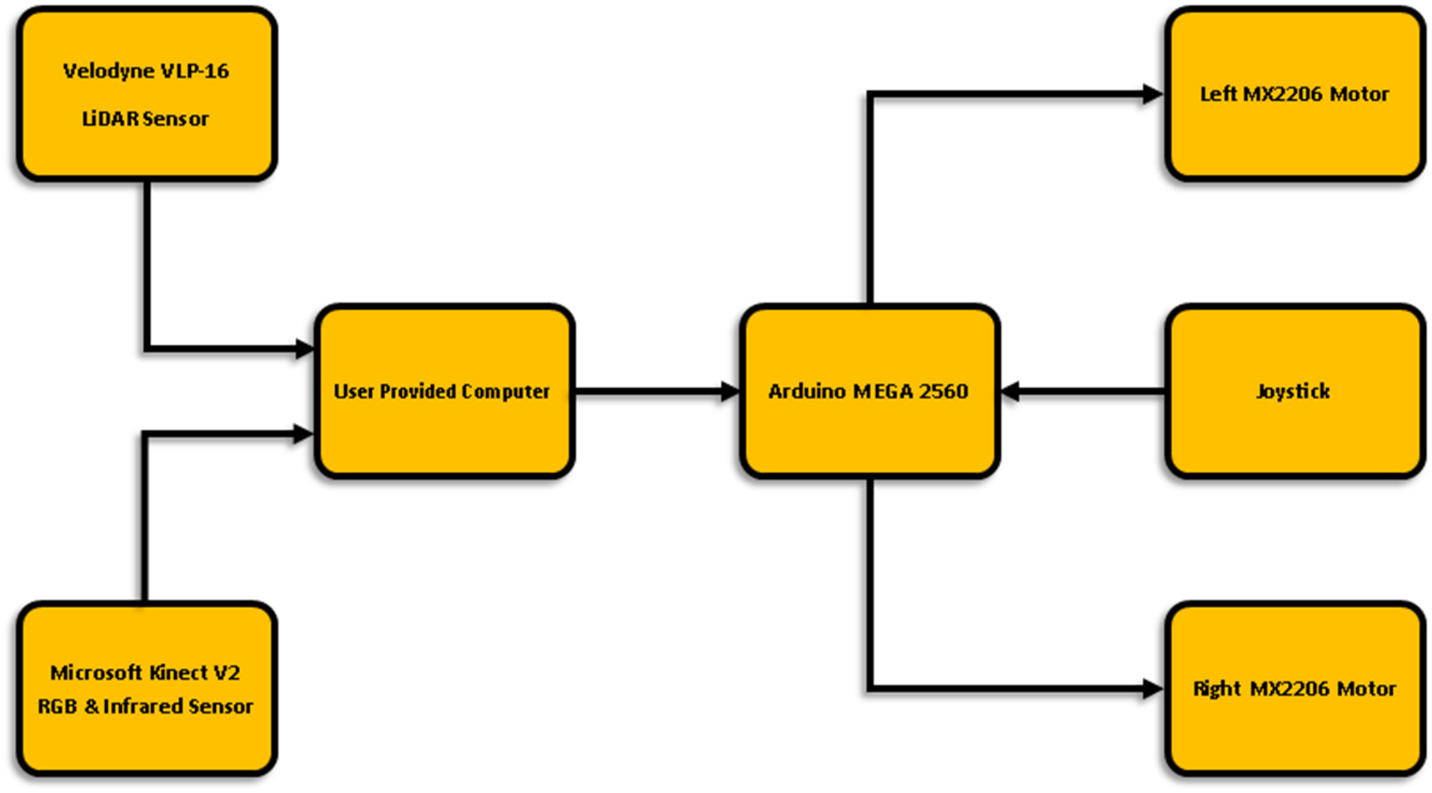
1. Connecting the sensors and Arduino to the Linux machine
2. Verifying that the brakes on the motors are in the locked position (The wheelchair will not power on if the breaks are unlocked)
3. Verify all power switches are in the on position
4. Launching of terminals and initializing the ROS workspace
5. Launching the object detection script and monitoring the terminal for proper output when an obstacle is detected
6. Launching the Navigation script and ensuring the ROS launch file is functioning with all necessary costmaps this will also begin the wheelchairs startup procedure controlled by the Arduino’s onboard program
7. Verifying that the Navigation script and object detection script are communicating by monitoring the terminal outputs.
8. Testing the manual navigation by moving the joystick
9. Waiting for the terminal message stating that it is ready for a target location
10. Selecting a target and looking for the green path generated by ROS

**Table 2:** Verification Approach & Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Systems** | **Analysis** | **Model** | **Simulation** | **Testing** |
| Mechanical | ✔ | ✔ | ✔ | ✔ |
| Motors |  |  |  | ✔ |
| Batteries | ✔ |  |  | ✔ |
| Microcontroller | ✔ |  |  | ✔ |
| Sensors | ✔ |  | ✔ | ✔ |

# System Overview

## 3.1. Configuration: Block Diagram & Description – Montealegre

**Figure 1:** System Block Diagram

The sensors assemblage is composed of the Velodyne VLP-16 & the Microsoft Kinect V2. The Velodyne VLP-16's LiDAR capabilities are utilized in this project to perform the SLAM & path-planning procedures. The Microsoft Kinect V2’s color & depth perception capabilities are utilized in this project to perform obstacle detection & avoidance. The raw feeds of both sensors are transferred to the host laptop. Since the SLAM & path-planning program utilizes ROS Noetic, the host laptop can only run Ubuntu 20.04. The raw feeds from both sensors are then processed, this entails interpreting the LiDAR sensor’s raw feed to produce a map of the wheelchair’s environment. This map is utilized to define the wheelchair’s location relative to the map produced, this is performed throughout the wheelchair’s path-planning procedures. The map produced is also utilized to plan a path that is within the constraints of the map, and also to adjust the planned path as the wheelchair follows such planned path. While the SLAM & path-planning procedures are performed, the host computer also processes the raw feed from the Kinect V2’s RGB & infrared sensors. The RGB sensor’s feed is utilized to detect & label obstacles. The infrared sensor’s feed is utilized to determine the distance to any obstacle in front of the wheelchair system, if these distance values are within a threshold distance, the wheelchair system comes to a stop and waits for the obstacle to be removed from the planned path. While the wheelchair system is not moving, the user is also able to utilize the joystick to maneuver around any obstacles that cannot be moved out of the planned path. In order to follow any planned path, the host laptop first sends a set of corresponding serial commands to the Arduino MEGA 2560 (the main component of the digital logic assemblage), which the Arduino translates to corresponding electric voltages that are sent to the electric motors, causing the wheelchair system to move along the planned path.

## 3.2. Major Subsystems & Components - Montealegre

The major subsystems of the wheelchair system, and their corresponding components, are:

* The computation assemblies
  + The laptop assemblage
    - Laptop mount
    - Host laptop
    - USB C adapter, which enables data transfer from the sensors, and power feed to the Arduino
  + The digital logic assemblage
    - Digital logic assemblage electrical box & its lid
    - Cable that feeds power to Arduino
    - Arduino MEGA 2560
    - Breadboard
    - Wiring from Arduino to breadboard, within breadboard, and from breadboard to motors
    - Resistors, capacitors, and relay
* The sensor assemblage
  + Sensors’ mounts
  + Velodyne VLP-16
  + Microsoft Kinect V2
  + Sensor assemblage electrical box
  + Wiring that transfers power from Bioenno battery to sensors
    - 20 amps fuse
    - A 24V → 12V power converters
    - Three SPST switches
* The batteries
  + Wheelchair’s OEM battery
  + Additional Bioenno battery
* The motors
  + Wheelchair’s two OEM motors (MX2206)
* Wheelchair’s structure
  + Aluminum frame
  + Fabric seat

## 3.3. Trade Studies - Montealegre

### 3.3.1. Microcontrollers

The team had to select a microcontroller to orchestrate the operation of the two electric motors, based on the computation of the sensors’ data by the host laptop. The team agreed to retain the Arduino Mega 2560. This is due to its 54 digital I/O pins (of which 15 provide PWM output) giving us the freedom to implement as many features as desired. Another advantage of the Arduino Mega, it provides a clock speed of 16 MHz. For the computation of the sensors’ feeds and the control of the Arduino Mega, the team vetted the set up implemented by the ‘Intelligent Wheelchair’ team (the team who worked on the wheelchair project last semester); such set up was composed of a Raspberry Pi 3.0 B & a 7” touchscreen attached to the wheelchair by a mount. The team then performed a trade study to compare the capabilities & features provided by the forementioned setup to the capabilities & features provided by a host laptop, the criteria used for such examination was processor speed, usability, memory, and form factor:

**Table 3:** Linux Machine Trade Study

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Weight** | **Huawei Matebook X Pro** | | **Raspberry Pi 3.0** | |
| **Rating** | **Weighted** | **Rating** | **Weighted** |
| **Processor Speed** | 0.5 | 4 | 2.0 | 3 | 1.5 |
| **RAM** | 0.2 | 4 | 0.8 | 2 | 0.4 |
| **Usability** | 0.1 | 5 | 0.5 | 3 | 0.3 |
| **Form Factor** | 0.2 | 2 | 0.4 | 5 | 1 |
| **Total** | 1 |  | 3.7 |  | 3.2 |

We prioritized processor speed & RAM because the wheelchair’s user might often find themselves in a situation that requires intensive data analysis and near immediate reaction by the computer, such as when performing path planning & needing to avoid obstacles above & below the floor’s surface. We also prefer the host laptop’s usability & form factor, the current laptop also allows the touchscreen of the laptop to be used as a user interface.

### 3.3.2. Sensors

The team performed a trade study to compare various LiDAR sensors offered by our mentor, Chris. The criteria used for such examination were range, sample frequency, samples per period, size, weight, and cost:

**Table 4:** LiDAR Sensor Trade Study

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Weight** | **Velodyne VLP-16** | | **RPLiDAR A2** | |
| **Rating** | **Weighted** | **Rating** | **Weighted** |
| **Range** | 0.3 | 5 | 1.5 | 1 | 0.3 |
| **Samples / period** | 0.3 | 4 | 1.2 | 2 | 0.6 |
| **Size & Weight** | 0.25 | 3 | 0.75 | 4 | 1 |
| **Cost** | 0.15 | 1 | 0.15 | 5 | 0.75 |
| **Total** | 1 |  | 3.6 |  | 2.65 |

The team chose the Velodyne Puck (VLP-16) due to its greater range of visualization, its greater number of samples per period of rotation, and despite its slightly greater size, the Velodyne Puck is still very versatile for our intents & purposes. The only criteria in which the RPLiDAR A2 greatly outperforms the Velodyne Puck is in cost, with an average price comparison of 1:10 when purchased new.

### 3.3.3. Motors

Since the structure of our project is an electric chair, we chose to retain the electric motors originally assembled with the wheelchair. The motors on the wheelchair can move a user as well as the additional weight of the components we are attaching to the wheelchair for our project so there was no need to replace them with a more powerful alternative.

### 3.3.4. Sensor Battery

In order to power all the additional components being implemented throughout our project, the team realized the need to utilize an additional battery. The team performed a trade study to compare various batteries offered by our mentor, Chris. The criteria used for such examination were weight, charge capacity (Amp-Hours), physical size and cost:

**Table 5:** Sensor Battery Trade Study

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **Weight** | **Bioenno 12V** | | **Eco-Worthy 12V** | |
| **Rating** | **Weighted** | **Rating** | **Weighted** |
| **Processor Speed** | 0.2 | 3 | 0.6 | 4 | 0.8 |
| **Memory** | 0.1 | 3 | 0.3 | 3 | 0.3 |
| **Usability** | 0.2 | 4 | 0.8 | 2 | 0.4 |
| **Form Factor** | 0.5 | 5 | 2.5 | 2 | 1 |
| **Total** | 1 |  | 4.2 |  | 2.5 |

The team preferred the Bioenno 12V battery because although it is a bit heavier, it provides a better size, shape and connector facilitating the attachment of the battery to the wheelchair structure and the implementation of the battery into the electrical system, while still providing as much charge capacity.

### 3.3.5. Wheelchair Frame

The team performed a trade study to compare the current system’s structure with the structure of other electric wheelchairs available in the market. The criteria used for such examination were weight, footprint, motor interface and cost:

**Table 6:** Wheelchair Trade Study

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Criteria** | **Weight** | **Yurob Portable** | | **Electra 7** | | **Falcon Portable** | |
| **Rating** | **Weighted** | **Rating** | **Weighted** | **Rating** | **Weighted** |
| **Processor Speed** | 0.2 | 4 | 0.8 | 2 | 0.4 | 2 | 0.4 |
| **Memory** | 0.2 | 3 | 0.6 | 2 | 0.4 | 2 | 0.4 |
| **Usability** | 0.3 | 4 | 1.2 | 3 | 0.9 | 3 | 0.9 |
| **Form Factor** | 0.3 | 5 | 1.5 | 2 | 0.6 | 3 | 0.9 |
| **Total** | 1 |  | 4.1 |  | 2.3 |  | 2.6 |

The team preferred to retain the current structure due to its aluminum alloy composition, which makes it very lightweight while still being very rigid. Retaining the current structure also saves the team what would likely be over $1000. The only disadvantage the team noticed is the lack of comfort of the textile used for the user to sit on, we therefore added some cushioning to alleviate the situation.

# Prototype Development

## 4.1. Major Works - Montealegre

The Automated Wheelchair consists of three major categories of work: Mechanical, Electrical and Programming. The mechanical system is now complete; this includes the 3D design and assembling of the sensors assemblage’s mounts & electrical box, the digital logic assemblage’s electrical box, and all other miscellaneous parts. A mount was purchased to attach the laptop to the wheelchair; this mount is composed of a set of links that attaches the laptop platform to a c-clamp. The electrical system is also now complete; this includes the digital logic assemblage (Arduino Mega & wheelchair motors), as well as the power and digital signals for the addition of the VLP-16 and Kinect. Lastly, the programming is 99% complete. The central python script, as well as the ROS launch, obstacle avoidance & Arduino scripts are complete; The union of all programs, which allows us to launch all programs in unison in order to perform all features, is now complete. All programs continue to be optimized and debugged through onsite testing.

### 4.1.1. Major Parts

**Table 7:** Major Parts

|  |  |
| --- | --- |
| **Part** | **Model** |
| Wheelchair Frame/Structure | Yurob Portable Electric Wheelchair |
| Microcontroller | Arduino Mega 2560 Rev3 |
| Sensors | Velodyne VLP-16, Microsoft Kinect V2 |
| Linux Machine | Huawei Matebook X Pro |
| Battery | Bioenno Power 12V Lithium-Ion Battery |

## 4.2. Hardware - Mannakulathil

### 4.2.1. CAD of Sensor Mounts

Prior to each mount being 3D printed it was modeled in Solidworks, to ensure correct dimensions. This allowed for simulations of how the parts would fit together when printed and assembled.

Diagram

Description automatically generated

**Figure 2:** Drawing of LiDAR Mount Design

Diagram, schematic

Description automatically generated

**Figure 3:** Drawing of Kinect Mount Design

Diagram, engineering drawing

Description automatically generated

**Figure 4:** Drawing of Handlebar Mount Design

### 4.2.2. CAD of Sensors Electrical Box

Diagram, engineering drawing

Description automatically generated

**Figure 5:** Drawing of Sensors Assemblage Electrical Box

Diagram, engineering drawing

Description automatically generated

**Figure 6:** Drawing of Sensor Assemblage Electrical Box Lid

### 4.2.3. CAD of Miscellaneous Parts

Diagram, engineering drawing

Description automatically generated

**Figure 7**: Drawing of Joystick Mount

Diagram, engineering drawing

Description automatically generated

**Figure 8:** Drawing of Arduino Housing Lid

### 4.2.4. Circuit Diagrams - Bronson

The wiring of the wheelchair system has been split into two drawings. The first drawing shows the digital signals from the Arduino to the wheelchair itself. This system is connected to the laptop using the Arduino’s integrated USB connection. The second drawing is the power diagram of the sensors and battery being added to the current system:

Diagram, schematic

Description automatically generated

**Figure 9:** Digital Signals Circuit Diagram

Diagram

Description automatically generated

**Figure 10:** Sensor Power Circuit Diagram

## 4.3. Software - Bronson & Montealegre

**4.3.1 SLAM & Path Planning**

The mapping and navigation capability is split between three programs scripts. The first is the ROS launch file. This file is responsible for configuring ROS and with the necessary packages. Inside ROS the first thing that must be done is configuring the interface to communicate with the Velodyne LiDAR module. This is done using the Velodyne published ROS package that allows for the processing of data into a usable LaserScan format. After configuring the sensor interface, a SLAM package needs to be chosen and given the necessary variables. The slam package used is Hector\_SLAM, however this is only used for mapping, localization is being done separately. Hector\_Mapping generates a map that is fed to the visual tool rViz. The last thing done in the ROS launch file is the frame matching to maintain accurate localization in the map.

The next script is the Python navigation script. This is used to subscribe to outputs from ROS and process them into serial commands that are sent to the Arduino. The script translates velocity commands, when to shift into and out of autonomous mode, and an interruption from the object detection script that is sent when an obstacle is caught in the frame. All this data is processed into commands and sent to the Arduino over a designated serial line.

The final script is the Arduino code written in C++. The Arduino is responsible for the startup sequence of the wheelchair and the sending of commands to the motors. This script receives serial commands from the Python navigation script and translates them into velocity commands for the wheelchair’s motors. It also handles where to source the navigation commands depending on if it is in autonomous or manual mode. While in manual mode, this script also receives and processes direction commands from the joystick mounted on the wheelchair.

**4.3.2 Object Detection & Avoidance**

Obstacle detection & avoidance are forms of computer vision. They entail image processing, the processing of a visual sensor's (most commonly an RGB sensor paired with an infrared sensor, or simply a set of three RGB sensors) image feed.

The program developed for this project begins by establishing a connection with the Kinect through its serial number. A frame listener is then established, and such frames begin being transferred to the computer, during this process the frames also begin being designated as RGB frames and depth frames. We then indicate the location of the configuration, .weights & .names files as parameters to the function used for comparing the contents of each frame to the forementioned files. In a while loop each new frame is designated to the same variable; at this point, each frame can be processed as desired with the use of the variables just established. We then ask the program to surround any objects detected in the RGB frames with a box, along with the object's name and the confidence of the detection performed. The location of these boxes is then passed to a function that processes the depth frames, inside the location of these boxes, the function "indexes" objects based on their distance from the Kinect. If objects detected are within a threshold distance, the program will generate serial commands in order to bring the wheelchair to a stop until the object moves away from the planned path.

## 4.4. Budget – Mannakulathil

### 4.4.1. Budget

Below is an estimate of the actual cost to the team taking into account the parts already owned by Kennesaw State University.

**Table 8:** Project Budget

|  |  |
| --- | --- |
| **Item** | **Price ($)** |
| Sensor Adapters | $30 |
| Battery Mount Hardware | $40 |
| Sensor Mount Hardware | $30 |
| Miscellaneous Parts/Materials | $50 |
| **Total (with contingency)** | **$150** |

### 4.4.2. Bill of Materials (BOM)

Below is a detailed Bill of Materials estimating the cost required to build the Automated Wheelchair without considering any already owned parts.

**Table 9:** Bill of Materials

|  |  |  |  |
| --- | --- | --- | --- |
| **Description** | **Quantity** | **Cost per Unit** | **Total Cost** |
| Wheelchair Structure | 1 | $1000 | $1000 |
| Velodyne VLP-16 | 1 | $4000 | $4000 |
| Microsoft Kinect V2 | 1 | $40 | $40 |
| Bioenno Battery | 1 | $193 | $193 |
| Mounting Hardware | 1 | $53 | $53 |
| Arduino MEGA 2560 | 1 | $40 | $40 |
| Wiring Multipack | 1 | $16 | $16 |
| Resistors Multipack | 1 | $12 | $12 |
| Host Laptop | 1 | $1500 | $1500 |
| Bread Board | 1 | $5 | $5 |
| **Total** |  |  | **$6860** |

## 4.5. Project Schedule - Bronson

**Graphical user interface, application, table, Excel

Description automatically generated**

**Figure 11:** Gantt Chart of Project Schedule

## 4.6. Analysis – Bronson & Montealegre

### 4.6.1. Operation Time and Power Draw

The system will need to be tested for operation time while all sensors are running, and the wheelchair is moving. Upon assemblage the wheelchair will run until the batteries are drained. Using the run time data, an estimated power draw will be calculated.

This testing was conducted by setting all sensors to their maximum power draw and allowing the system to run continuously for as long as possible. At the end of the test the system had run uninterrupted for 5 hours and was still capable of completing all functions necessary. These results far exceeded the necessary runtime for any functions anticipated.

### 4.6.2. Testing Path-planning

Testing and analysis of the path-planning algorithms will be crucial in developing a successful system. This will be done through both physical tests and ROS simulation; the team will extensively navigate the wheelchair throughout hallways and any other areas possible. The testing & analysis will focus on ways to optimize the system’s path-planning capabilities, as well as confirmation of user comfort and safety.

Testing of path-planning was conducted in rViz upon the wheelchair being capable of following a given path. The focus was on two aspects of functionality, firstly the feasibility of the path being planned, and secondly the ability of the wheelchair to accurately follow the desired path. Using properly configured costmaps and tuning the speeds of the wheelchair have resulted in a product that both generates safe plans and follows those closely enough for the system to be trusted.

### 4.6.3. Optimization of Obstacle Detection & Avoidance

Throughout the development of the obstacle detection program, proficiency tests were carried out to confirm whether obstacles were being detected, and whether the obstacles detected were labeled correctly. We discovered that obstacles were being detected as often as they were placed in front of the RGB sensor, but a few deficiencies were encountered:

As the reference files that the program uses to generate, encase, and label a detection may not contain sufficient reference to correctly identify all obstacles in the world; whenever an obstacle was detected, it wasn’t always labeled correctly, sometimes causing a workbench to be labeled as a dining table.

Also, sometimes the program displayed multiple detections where only one obstacle was present. This obligated the team to change the functions used to generate, encase and label detections, as well as the parameters and their data types.

Throughout the development of the obstacle avoidance program, proficiency tests were also carried out to confirm that the program would be able to provide the desired feature of depth perception. Once the program was complete, we encountered one deficiency:

Although the team was under the belief that the program was using SI units, the distance values used by the program were not using either imperial or SI units. Therefore, whenever threshold values to execute a navigation action were set, we noticed that the distances in our real world were different. This required a different approach to the processing of the raw feed from the infrared sensor; effectively setting the units used by the program to millimeters.

### 4.6.4. Sensor Data Accuracy Check

Manual analysis of the data received from both the Kinect and the LiDAR sensor will be conducted to ensure there are no unexpected objects or gaps in the readings taken by the sensors. There will also be inspections of the data received as it is represented in rViz; this representation provides a straightforward way to ensure that the data is accurately representing the wheelchair’s environment.

The sensor data was a visual check comparing the world around the wheelchair to the world generated in rViz. The system reliably creates an accurate representation of its surroundings. A couple of shortcomings within the system are that the LiDAR sensor does not detect windows and it encounters some instability while turning at high rotation rates. While these are notable concerns to be addressed in the future, the desired capabilities of the wheelchair are met despite them.

## 4.7. Navigation in a Virtual Environment- Bronson

Using rViz and the data from the sensors, a virtual environment can be created that closely matches the real world around the wheelchair. Within this virtual environment, the expected operations of the wheelchair can be closely monitored and compared to their equivalents in the real world. This means that as the system operates in the real world, it can be optimized to better match the expected functionality shown in the virtual environment. An example of this is shown in the rViz screenshot below. This screenshot shows how the virtual world is attempting to match the changing real-world environment as the wheelchair moves. A screenshot of a computer

Description automatically generated with medium confidence

**Figure 12:** Visualization of Localization & Mapping

## 4.8. Testing - Bronson

### 4.8.1. Sensor Calibration Testing

Testing of the calibration before operation will be crucial in creating a consistent and reliable system. Both the LiDAR sensor and Kinect can be recalibrated, should the data received be inaccurate or unpredictable. This testing will be done by placing the sensors in a consistent start position and confirming accurate data. Should the data not be as expected, recalibration will be conducted, and the test will be restarted.

### 4.8.2. Motor Response Testing

Manual commands have been sent from the joystick on the wheelchair to ensure the motors respond correctly when told to maneuver forward, back, and turn. Forced commands from the Arduino have confirmed proper communication between the microcontroller and the motors.

### 4.8.3. Obstacle Avoidance Testing

After its development, intensive testing has been conducted in order to redefine and optimize the functionalities the team wants the obstacle detection & avoidance program to have. This has entailed editing the functions, as well as their parameters and threshold values, utilized by the program in order to achieve such desired functionalities. The main desired functionality is ensuring that the wheelchair does not collide with obstacles on the wheelchair’s path, as this is crucial in preventing injury or damage to the user, the obstacle and the wheelchair.

# Conclusion

This section covers available resources, project schedule, and team responsibilities.

## 5.1. Achievements & Lessons Learned - Mannakulathil

From personal experience, we would like to recommend, never 3D print a part fully filled in, always apply an infill pattern. Also, always confirm the measurements (preferably with a caliper) of any physical parts that your 3D designs will interact with, this will keep you from wasting materials and time during the printing of a part that will need to be modified. Learned how to solder an electrical system together and test it for shorts, faulty contacts, and polarity. Learned how the engineering process works and how responsibility is distributed during projects. Expanded understanding of ROS and how it interacts with other programming languages. This includes writing launch files, creating configuration files for packages, and learning how rospy allows for python scripts to interact with different ROS topics and nodes. Learned about different options for path planning and world mapping. This required in-depth analysis of the capabilities of the different options and an examination of the plausibility of implementing each one. Expanded our capabilities in Python and Arduino C. The additional libraries needed in Python to work with ROS files in meaningful ways expanded the general understanding of program structure in Python. Interacting with serial lines and processing commands on an Arduino required that we gain an understanding of the different ports of Arduinos and how they are configured. Learned about object detection in Python using a Kinect’s sensors, which expanded our understanding of the overall structure of image processing and its application into computer vision. The book expanded the team’s knowledge of the engineering processes entailed during the development of an engineering project.

## 5.2. Future Improvements & Optimizations - Bronson

Throughout the development of the project there have been several opportunities for both expansion and improvement. The first of these expansions in capability would be to implement voice commands, this would allow further impaired users to easily utilize the system. Using the public Amazon Alexa voice recognition algorithm and the microphone array on the Microsoft Kinect, voice commands could be an added functionality.

While this project implements the rViz integrated path planning algorithm set, there are many alternatives that have different advantages.

A GUI that allows the user to select from a variety of path planning options based on the current need could be implemented to improve the dynamic ability of the wheelchair to meet the users' needs.

The current wiring of the digital logic assemblage is not user friendly. It is in a place that interferes with the comfort of the user when they sit while also being at risk of dislodging connections if bumped. A reworking of this circuit would create a more comfortable user experience while also improving the reliability of the system.

Even with functional obstacle avoidance there is always room for improvement. Expanding the types of obstacles that the wheelchair recognizes and avoids would create a safer system and prevent any unexpected collisions due to a failure to notice an obstacle on the path. Specifically, the system could be improved in a way that allows it to recognize sudden changes in elevation that might result in either tipping or uncontrollable acceleration of the wheelchair.

The wheelchair has a lot of weight on the back due to the added sensors and battery. This causes it to be unstable and affects the predictability of movement. Adding weight to the front of the wheelchair would stabilize it and help limit unexpected movement of the wheelchair as it navigates.

## 5.3. Team Chart/Assignments - Mannakulathil

Max Bronson is the team leader, as well as one of the programming leads in charge of the LiDAR interfacing, ROS, and Arduino Programming. Oscar Montealegre is the other programming lead, focusing on Kinect Interfacing, Object Identification, and ROS programming. Denny Mannakulathil is in charge of 3D design, which makes him responsible for the modeling, drawing, and 3D printing of all parts needed.

# References - Montealegre

[LIDAR-based autonomous wheelchair](https://ieeexplore.ieee.org/document/7894082)

[Autonomous wheelchair design reference](https://journals.sagepub.com/doi/10.5772/55477)

[Python reference #1](https://www.youtube.com/watch?v=XCKWZAtKTnU&list=PLGs0VKk2DiYzguDvh5xk2XoX9V1VKP5Hv)

[3D design reference](https://www.tinkercad.com/things/eF5YDip8WDO-lidar-scanse)

[SolidWorks reference](https://help.solidworks.com/2018/english/SolidWorks/sldworks/HIDD_OPTIONS_IMPORT_VRML_2.htm)

[D\* lite reference](https://github.com/Sollimann/Dstar-lite-pathplanner)

<https://ieeexplore.ieee.org/document/7894082>

[ROS tutorial for beginners](https://youtube.com/playlist?list=PLk51HrKSBQ8-jTgD0qgRp1vmQeVSJ5SQC)

[OpenCV with use of python reference #1](https://www.youtube.com/playlist?list=PLS1QulWo1RIa7D1O6skqDQ-JZ1GGHKK-K)

[OpenCV with use of python reference #2](https://www.youtube.com/watch?v=oXlwWbU8l2o)

[ROS reference #1](https://www.youtube.com/playlist?list=PLk51HrKSBQ8-jTgD0qgRp1vmQeVSJ5SQC)

[ROS reference #2](https://www.youtube.com/playlist?list=PL8dDSKArO2-m7hAjOgqL5uV75aZW6cqE5)

[ROS reference #3](https://www.youtube.com/playlist?list=PLJNGprAk4DF5PY0kB866fEZfz6zMLJTF8)

[Arduino reference](https://www.youtube.com/playlist?list=PLGs0VKk2DiYw-L-RibttcvK-WBZm8WLEP)

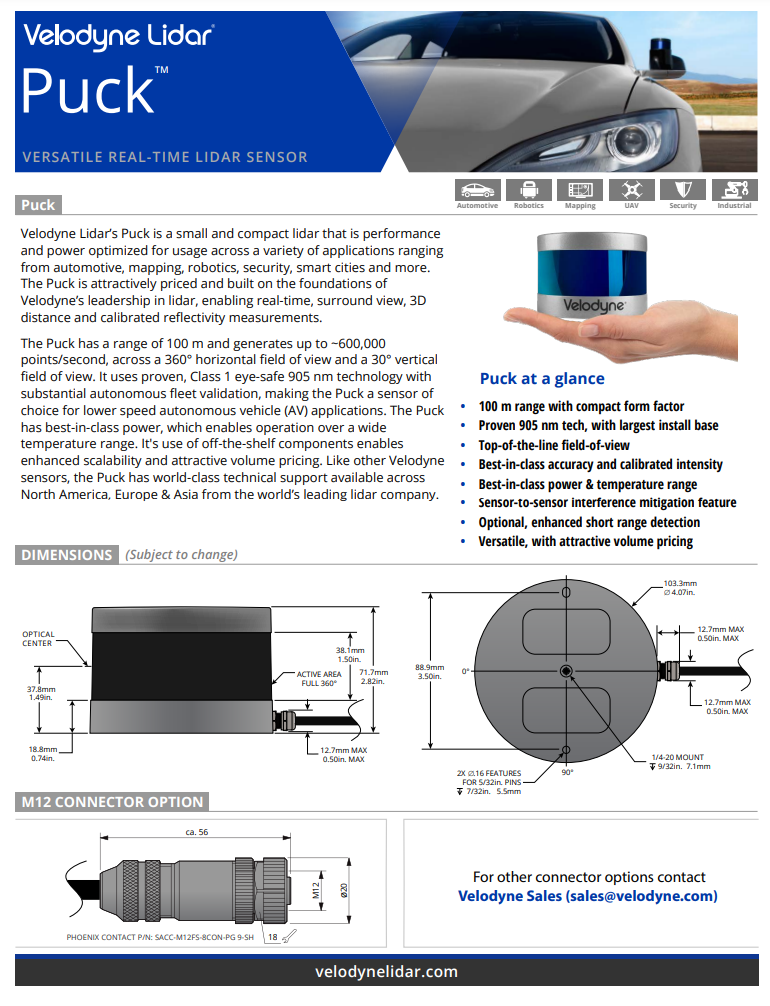
# Resources - Mannakulathil

**Table 10:** Available Resources

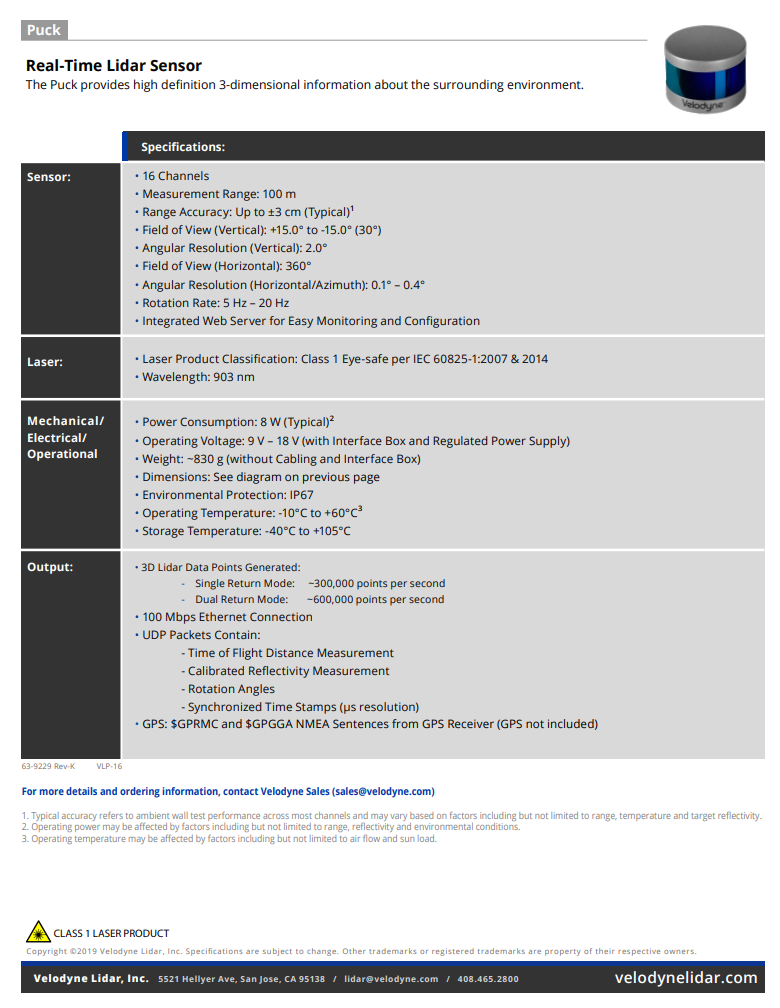
|  |  |
| --- | --- |
| **Hardware** | **Faculty** |
| * Velodyne VLP-16 | * Chris Voicu – Master of All |
| * Microsoft Kinect V2 | * Muhammad Hassan Tanveer – ROS |
| * Arduino MEGA 2560 | * Ying Wang – Obstacle Avoidance |
| * Huawei Matebook X Pro |  |
| * Wheelchair Skeletal Structure |  |
| **Software** | **Facility** |
| * Solidworks | * MTRE Lab – Q-118 |
| * AutoCAD Electrical | * K-Space – Q-308 |
| * Visual Studio Code |  |

# Appendices - Bronson

## Appendix A: Velodyne VLP-16

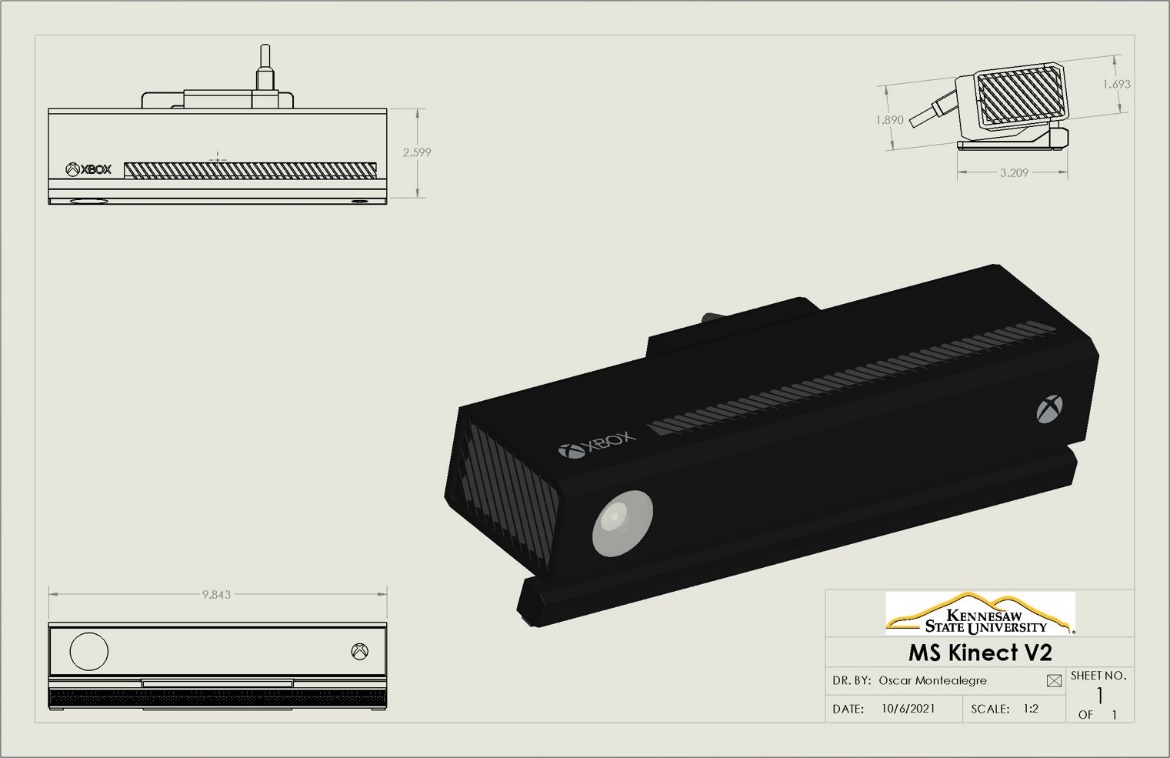


**Figure 13:** VLP-16 Data Sheet 1

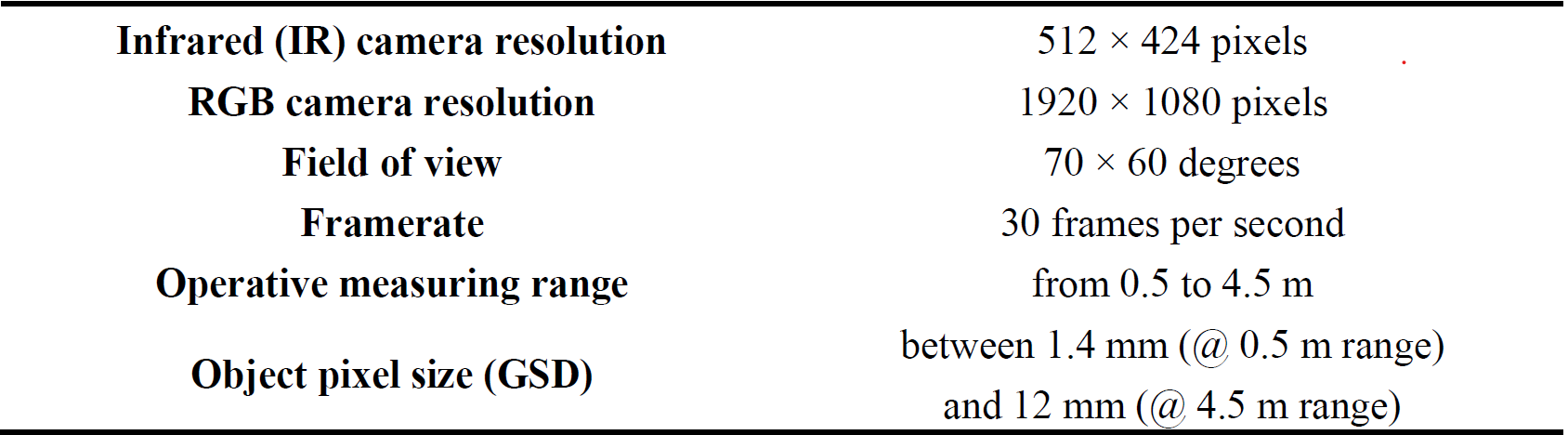


**Figure 14:** VLP-16 Data Sheet 2

## Appendix B: Microsoft Kinect V2

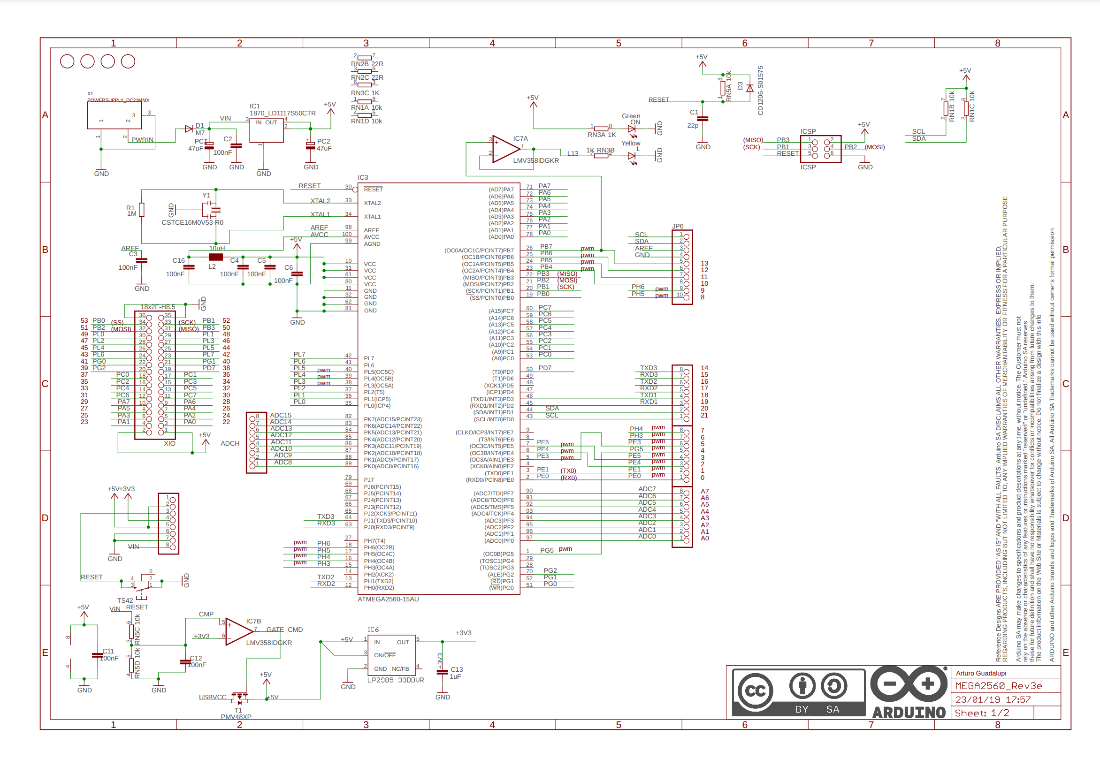


**Figure 15:** Microsoft Kinect V2 Rendering

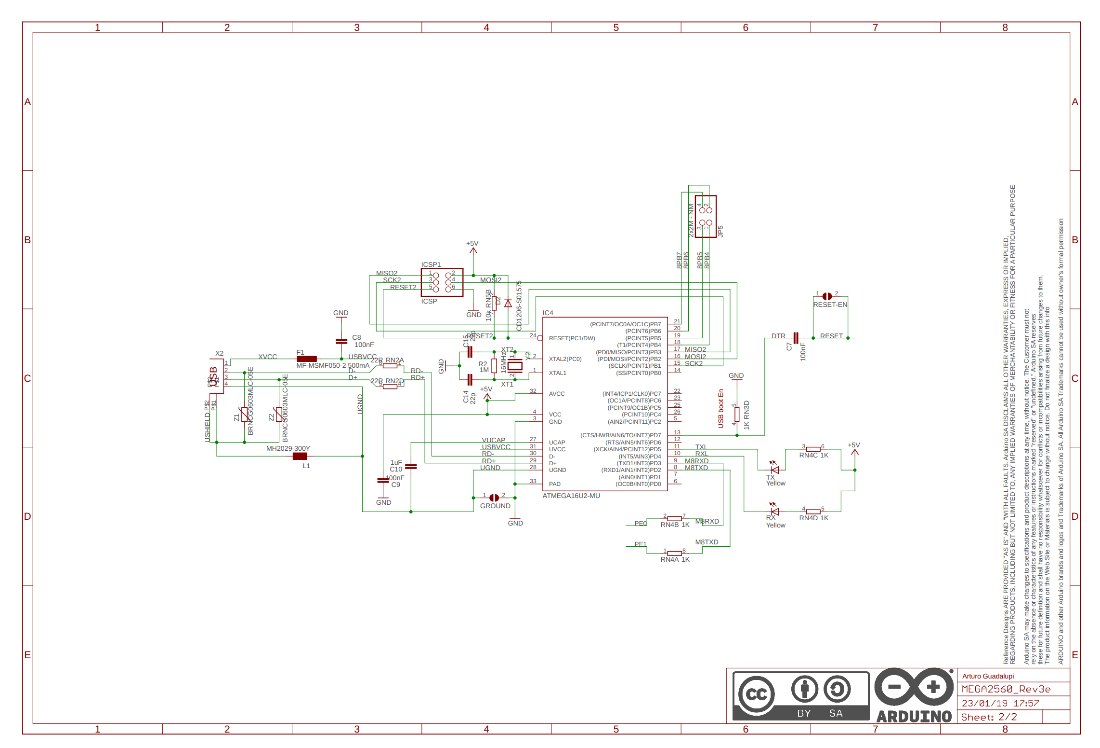


**Figure 16:** Microsoft Kinect V2 Data Sheet

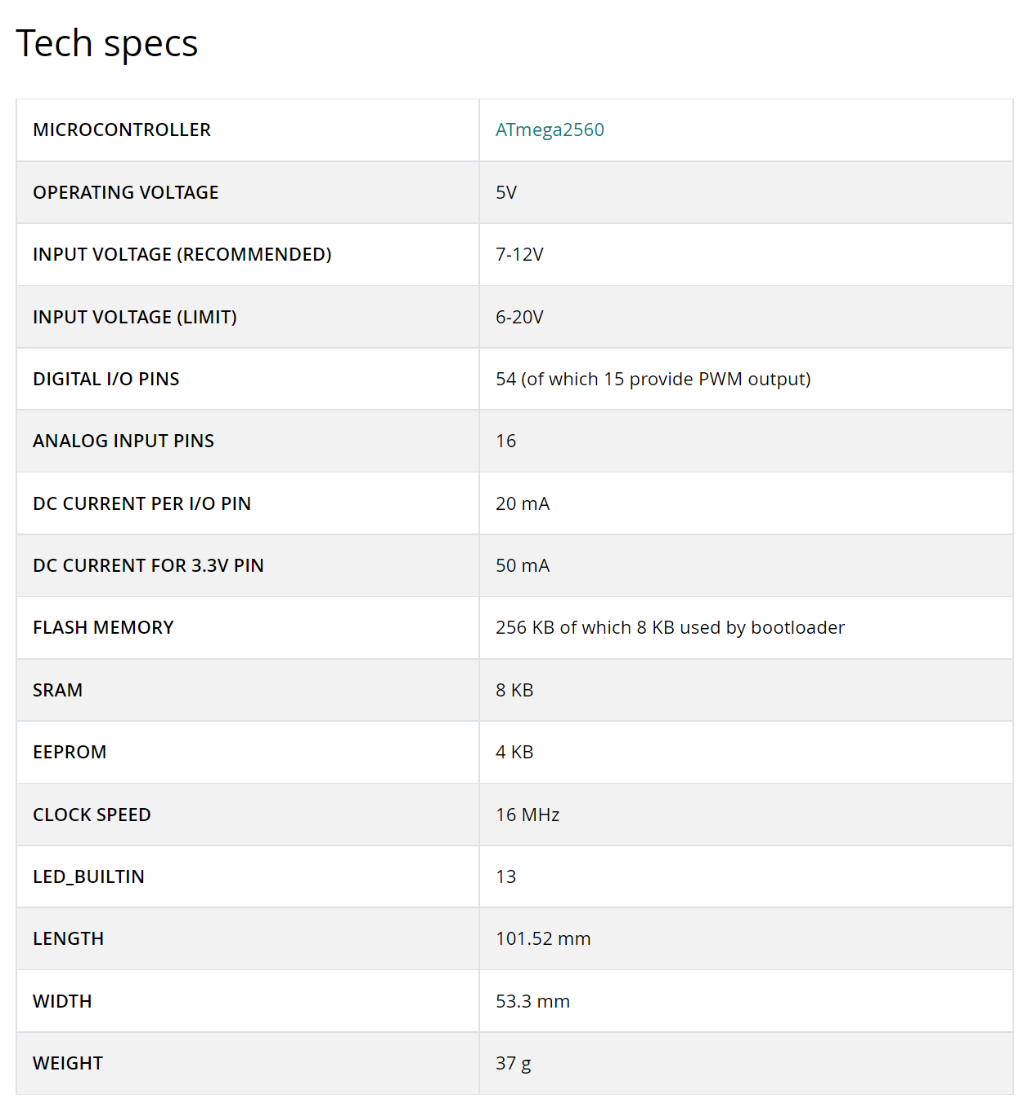
## Appendix C: Arduino 2560



**Figure 17:** Arduino 2560 Rev3 Wiring Diagram 1

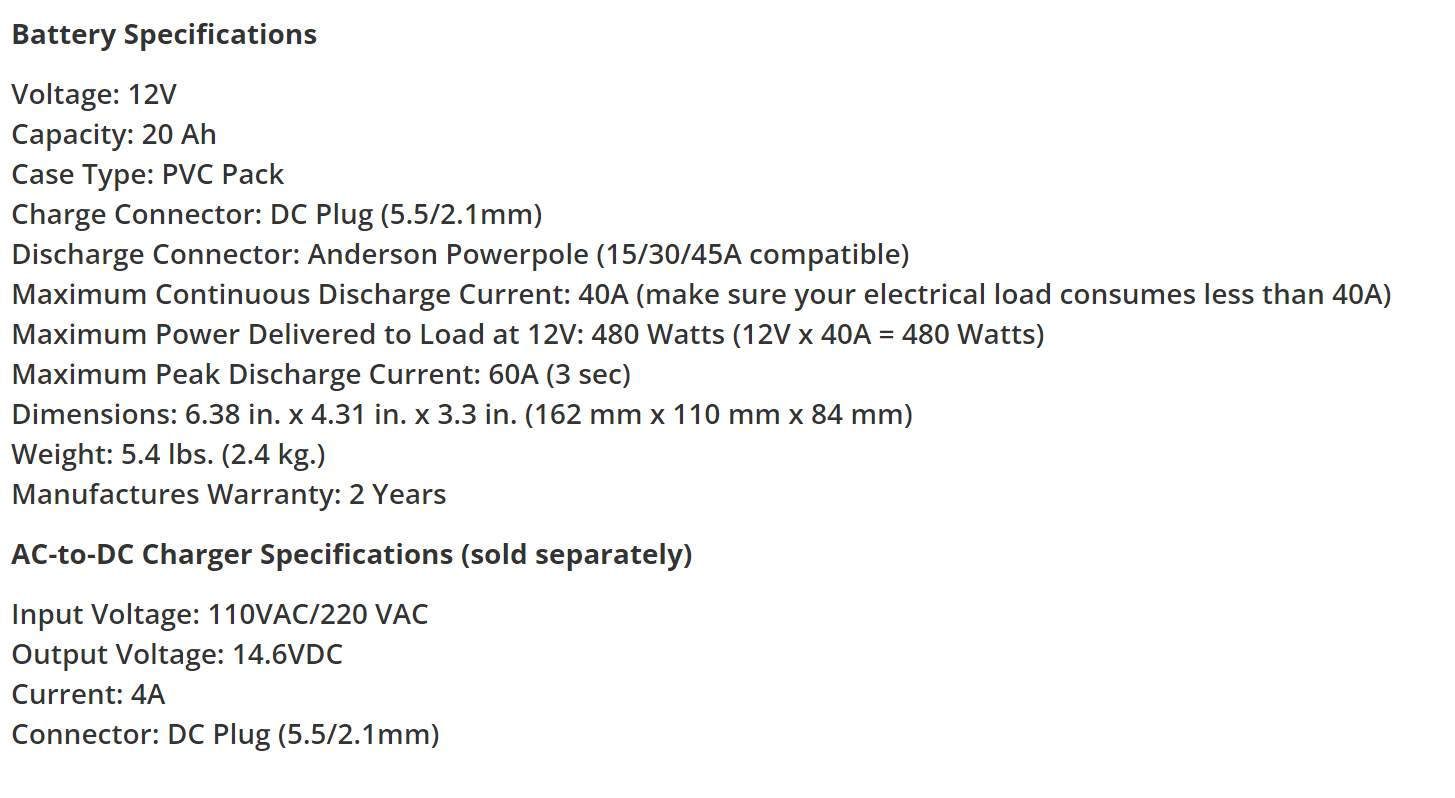


**Figure 18:** Arduino 2560 Rev3 Wiring Diagram 2



**Figure 19:** Arduino 2560 Rev3 Technical Specifications

## Appendix D: Bioenno Lithium-Ion Battery



**Figure 20:** Properties of Bioenno Battery

## Appendix E: Project Drawings

Diagram

Description automatically generated

**Figure 21:** Drawing of LiDAR Mount Design

Diagram, schematic

Description automatically generated

**Figure 22:** Drawing of Kinect Mount Design

Diagram, engineering drawing

Description automatically generated

**Figure 23:** Drawing of Handlebar Mount Design

Diagram, engineering drawing

Description automatically generated

**Figure 24:** Drawing of Sensors Assemblage Electrical Box

Diagram, engineering drawing

Description automatically generated

**Figure 25:** Drawing of Sensor Assemblage Electrical Box Lid

Diagram, engineering drawing

Description automatically generated

**Figure 26**: Drawing of Joystick Mount

Diagram, engineering drawing

Description automatically generated

**Figure 27:** Drawing of Arduino Housing Lid

## Appendix F: Electrical Diagrams

Diagram, schematic

Description automatically generated

**Figure 28:** Digital Signals Circuit Diagram

Diagram

Description automatically generated

**Figure 29:** Sensor Power Circuit Diagram

## Appendix G: Project Programming Scripts

ROS Launch File:

<?xml version="1.0"?>

<launch>

<!-- Velodyne PointCloud Launchers -->

<!-- declare arguments with default values -->

<arg name="calibration" default="$(find velodyne\_pointcloud)/params/VLP16db.yaml"/>

<arg name="device\_ip" default="192.168.1.201" />

<arg name="frame\_id" default="velodyne" />

<arg name="manager" default="$(arg frame\_id)\_nodelet\_manager" />

<arg name="max\_range" default="130.0" />

<arg name="min\_range" default="1" />

<arg name="pcap" default="" />

<arg name="port" default="2368" />

<arg name="read\_fast" default="false" />

<arg name="read\_once" default="false" />

<arg name="repeat\_delay" default="0.0" />

<arg name="rpm" default="1200.0" />

<arg name="gps\_time" default="false" />

<arg name="pcap\_time" default="false" />

<arg name="cut\_angle" default="-0.01" />

<arg name="timestamp\_first\_packet" default="false" />

<arg name="laserscan\_ring" default="-1" />

<arg name="laserscan\_resolution" default="0.007" />

<arg name="organize\_cloud" default="false" />

<!-- start nodelet manager and driver nodelets -->

<include file="$(find velodyne\_driver)/launch/nodelet\_manager.launch">

<arg name="device\_ip" value="$(arg device\_ip)"/>

<arg name="frame\_id" value="$(arg frame\_id)"/>

<arg name="manager" value="$(arg manager)" />

<arg name="model" value="VLP16"/>

<arg name="pcap" value="$(arg pcap)"/>

<arg name="port" value="$(arg port)"/>

<arg name="read\_fast" value="$(arg read\_fast)"/>

<arg name="read\_once" value="$(arg read\_once)"/>

<arg name="repeat\_delay" value="$(arg repeat\_delay)"/>

<arg name="rpm" value="$(arg rpm)"/>

<arg name="gps\_time" value="$(arg gps\_time)"/>

<arg name="pcap\_time" value="$(arg pcap\_time)"/>

<arg name="cut\_angle" value="$(arg cut\_angle)"/>

<arg name="timestamp\_first\_packet" value="$(arg timestamp\_first\_packet)"/>

</include>

<!-- start transform nodelet -->

<include file="$(find velodyne\_pointcloud)/launch/transform\_nodelet.launch">

<arg name="model" value="VLP16"/>

<arg name="calibration" value="$(arg calibration)"/>

<arg name="manager" value="$(arg manager)" />

<arg name="fixed\_frame" value="" />

<arg name="target\_frame" value="" />

<arg name="max\_range" value="$(arg max\_range)"/>

<arg name="min\_range" value="$(arg min\_range)"/>

<arg name="organize\_cloud" value="$(arg organize\_cloud)"/>

</include>

<!-- start laserscan nodelet -->

<include file="$(find velodyne\_pointcloud)/launch/laserscan\_nodelet.launch">

<arg name="manager" value="$(arg manager)" />

<arg name="ring" value="$(arg laserscan\_ring)"/>

<arg name="resolution" value="$(arg laserscan\_resolution)"/>

</include>

<!-- Hector\_SLAM Configuration -->

<arg name="tf\_map\_scanmatch\_transform\_frame\_name" default="base\_link"/>

<arg name="base\_frame" default="base\_footprint"/>

<arg name="odom\_frame" default="odom"/>

<arg name="pub\_map\_odom\_transform" default="false"/>

<arg name="scan\_subscriber\_queue\_size" default="500"/>

<arg name="scan\_topic" default="scan"/>

<arg name="map\_size" default="1500"/>

<node pkg="hector\_mapping" type="hector\_mapping" name="hector\_mapping" output="screen">

<!-- Frame names -->

<param name="map\_frame" value="map" />

<param name="base\_frame" value="$(arg base\_frame)" />

<!-- <param name="odom\_frame" value="$(arg odom\_frame)" /> -->

<!-- Tf use -->

<param name="use\_tf\_scan\_transformation" value="false"/>

<param name="use\_tf\_pose\_start\_estimate" value="false"/>

<param name="pub\_map\_odom\_transform" value="false"/>

<!-- Map size / start point -->

<param name="map\_resolution" value=".05"/>

<param name="map\_size" value="2048"/>

<param name="map\_start\_x" value="0.5"/>

<param name="map\_start\_y" value="0.5" />

<param name="map\_multi\_res\_levels" value="2" />

<!-- Map update parameters -->

<param name="update\_factor\_free" value="0.4"/>

<param name="update\_factor\_occupied" value="0.9" />

<param name="map\_update\_distance\_thresh" value=".6"/>

<param name="map\_update\_angle\_thresh" value="0.9" />

<param name="laser\_min\_dist" value = "1" />

<param name="laser\_max\_dist" value = "130.0" />

<param name="laser\_z\_min\_value" value = "-1" />

<param name="laser\_z\_max\_value" value = "1" />

<param name="map\_pub\_period" value = "0.5" />

<param name="advertise\_map\_service" value="true"/>

<param name="tf\_map\_scanmatch\_transform\_frame\_name" value="$(arg tf\_map\_scanmatch\_transform\_frame\_name)" />

<param name="pub\_map\_scanmatch\_transform" value="false" />

</node>

<!-- Advertising config -->

<node pkg="tf" type="static\_transform\_publisher" name="map\_base\_footprint" args="0 0 0 0 0 0 map base\_footprint 100"/>

<!-- <node pkg="tf" type="static\_transform\_publisher" name="footprint\_to\_frame" args="0 0 0 0 0 0 base\_footprint base\_frame 100"/> -->

<node pkg="tf" type="static\_transform\_publisher" name="base\_link\_2\_velodyne" args="0 0 0 0 0 0 base\_link velodyne 5"/>

<node pkg="laser\_scan\_matcher" type="laser\_scan\_matcher\_node" name="laser\_scan\_matcher\_node" output="screen">

<param name="base\_frame" value="base\_link"/>

<param name="fixed\_frame" value="map"/>

<param name="use\_alpha\_beta" value="false"/>

<param name="use\_odom" value="false"/>

<param name="use\_imu" value="true"/>

<param name="max\_iterations" value="100"/>

<param name="publish\_pose" value="true"/>

<param name="publish\_tf" value="true"/>

<param name="use\_vel" value="false"/>

</node>

<!-- <node name="costmap\_node" pkg="costmap\_2d" type="costmap\_2d\_node">

<rosparam file="$(find navigation)/config/global\_costmap\_params.yaml" command="load" ns="costmap" />

</node> -->

<arg name="no\_static\_map" default="false"/>

<arg name="base\_global\_planner" default="navfn/NavfnROS"/>

<!-- <arg name="base\_local\_planner" default="dwa\_local\_planner/DWAPlannerROS"/> -->

<arg name="base\_local\_planner" default="base\_local\_planner/TrajectoryPlannerROS"/>

<node pkg="move\_base" type="move\_base" respawn="false" name="move\_base" output="screen">

<param name="base\_global\_planner" value="$(arg base\_global\_planner)"/>

<param name="base\_local\_planner" value="$(arg base\_local\_planner)"/>

<rosparam file="$(find navigation)/config/planner.yaml" command="load"/>

<!-- observation sources located in costmap\_common.yaml -->

<rosparam file="$(find navigation)/config/costmap\_common.yaml" command="load" ns="global\_costmap" />

<rosparam file="$(find navigation)/config/costmap\_common.yaml" command="load" ns="local\_costmap" />

<!-- local costmap, needs size -->

<rosparam file="$(find navigation)/config/costmap\_local.yaml" command="load" ns="local\_costmap" />

<param name="local\_costmap/width" value="10.0"/>

<param name="local\_costmap/height" value="10.0"/>

<!-- static global costmap, static map provides size -->

<rosparam file="$(find navigation)/config/costmap\_global\_static.yaml" command="load" ns="global\_costmap" unless="$(arg no\_static\_map)"/>

<!-- global costmap with laser, for odom\_navigation\_demo -->

<rosparam file="$(find navigation)/config/costmap\_global\_laser.yaml" command="load" ns="global\_costmap" if="$(arg no\_static\_map)"/>

<param name="global\_costmap/width" value="100.0" if="$(arg no\_static\_map)"/>

<param name="global\_costmap/height" value="100.0" if="$(arg no\_static\_map)"/>

</node>

<!-- OLD MOVE BASE CONFIG -->

<!-- <node pkg="move\_base" type="move\_base" respawn="false" name="move\_base" output="screen">

<rosparam file="$(find navigation)/config/costmap\_common\_params.yaml" command="load" ns="global\_costmap"/>

<rosparam file="$(find navigation)/config/costmap\_common\_params.yaml" command="load" ns="local\_costmap"/>

<rosparam file="$(find navigation)/config/local\_costmap\_params.yaml" command="load"/>

<rosparam file="$(find navigation)/config/global\_costmap\_params.yaml" command="load"/>

<rosparam file="$(find navigation)/config/trajectory\_planner.yaml" command="load"/>

<rosparam file="$(find navigation)/config/move\_base\_params.yaml" command="load"/>

</node> -->

<!-- Launch RVIZ Using Velodyne Configuration -->

<node pkg="rviz" type="rviz" name="rviz" args="-d /home/max/Documents/catkin\_ws/rviz\_config\_Hector\_SLAM.rviz"/>

</launch>

Python Navigation Script:

#!/usr/bin/env python3

import os

import time

import rospy

import serial

import pickle

import actionlib

import subprocess as sp

import multiprocessing as mp

from std\_msgs.msg import String

from geometry\_msgs.msg import PoseStamped,Twist, PoseWithCovarianceStamped, Pose2D

from actionlib\_msgs.msg import GoalStatusArray

from move\_base\_msgs.msg import MoveBaseAction

# Launch ROS process

def ROSProcess():

sp.run('roslaunch navigation hector\_map.launch', shell = True, check = True, stdout = sp.PIPE, stderr = sp.STDOUT)

# Establish serial communication with external device

def setupComPort(comPort):

serialPort = serial.Serial(port = comPort, baudrate = 9600, bytesize=8, timeout=2, stopbits=serial.STOPBITS\_ONE)

return serialPort

# Create necessary global variables

COM = setupComPort("/dev/ttyACM0")

serialCounter = 0

cancelBool = False

freeze = "False"

# Clear the map on rViz (NOT CURRENTLY IN USE, IF USED ALLOW TIME FOR MAP TO REPOPULATE)

def clearMap():

print("Clearing MAP")

clear\_publisher = rospy.Publisher("syscommand", String, queue\_size=5)

msg = "reset"

clear\_publisher.publish(msg)

# send a new target goal to rViz (NOT CURRENTLY IN USE)

def setGoal(msg):

goal\_publisher = rospy.Publisher("move\_base\_simple/goal", PoseStamped, queue\_size=5)

goal = PoseStamped()

if msg.pose != goal.pose:

writeCommand(COM, 'a')

goal.header.seq = 1

goal.header.frame\_id = "map"

goal.header.stamp = rospy.Time.now()

goal.pose = msg.pose

goal\_publisher.publish(goal)

time.sleep(2)

# Translate the desired command and assign it the proper numeric value

def translateCommands(target):

global COM

lineA = float(target.linear.x)

lineB = float(target.angular.z)

if lineA> 0:

lineA = lineA+200

elif lineA< 0:

lineA = lineA+100

elif lineA == 0:

lineA = 135

if lineB> 0:

lineB = lineB+160

elif lineB< 0:

lineB = lineB+110

elif lineB == 0:

lineB = lineB+135

lineA = 'A' + str(int(lineA))

lineB = 'B' + str(int(lineB))

print('x = ',target.linear.x,'a = ', lineA)

print('y = ',target.angular.z,'b = ', lineB)

writeCommand(COM, lineA)

writeCommand(COM, lineB)

# Format the desired command and send it over the open COM port

def writeCommand(comPort, strvar):

comPort.write(str.encode(strvar + '\*'))

# Translate and send velocity commands received from rViz, flush serial line every 25 messages sent to prevent overloading

def navCommandsReceived(poses):

global COM

global freeze

global serialCounter

if freeze == "False":

translateCommands(poses)

if serialCounter == 25:

COM.flushInput()

COM.flushOutput()

serialCounter = 0

serialCounter = serialCounter+1

# When new target goal is reached, send an 'a' command to put the wheelchair in autonomous mode

def newGoalReceived(target):

global COM

global cancelBool

cancelBool = True

writeCommand(COM,'a')

time.sleep(.5)

writeCommand(COM,'a')

# When Target location is reach send a DONE command and clear the goal from Rviz

def targetReached(status):

global COM

global cancelBool

if status.status\_list != []:

if status.status\_list[0].status == 3 and cancelBool == True:

print('Target reached')

writeCommand(COM, 'DONE')

move\_base = actionlib.SimpleActionClient('/servicebot/move\_base', MoveBaseAction)

move\_base.cancel\_all\_goals()

cancelBool = False

# Check the camera output for Wheelchair Freeze command

def checkCamera(pose):

global freeze

global COM

if os.path.getsize("/home/max/shared.pkl") > 0:

fp = open("/home/max/shared.pkl", "rb")

freeze = pickle.load(fp)

if freeze == "True":

print('checkCamera ', freeze)

stopWheelchair()

# Send Stop command to wheelchair

def stopWheelchair():

global COM

writeCommand(COM, 'A135')

writeCommand(COM, 'B135')

# Looping listener for ROS Topics

def listener():

global freeze

rospy.init\_node('listener',anonymous=True)

rospy.Subscriber('/pose2D', Pose2D, checkCamera)

rospy.Subscriber('/move\_base\_simple/goal', PoseStamped, newGoalReceived)

rospy.Subscriber('/cmd\_vel', Twist, navCommandsReceived)

rospy.Subscriber('/move\_base/status', GoalStatusArray, targetReached)

rospy.spin()

# Launch ROS and rVIZ, start listener process

def main():

p = mp.Process(target=ROSProcess)

p.start()

time.sleep(10)

l = mp.Process(target=listener)

l.start()

# sp.run('mark3.py', shell = True, check = True, stdout = sp.PIPE, stderr = sp.STDOUT)

time.sleep(5)

print('Ready for target location')

p.join()

l.join()

if \_\_name\_\_ == '\_\_main\_\_':

print('Start navigation script')

main()

Object Detection Script:

#!/usr/bin/env python3

import numpy as np

import cv2

import scipy.misc

import signal

import pyfreenect2

from numpy import testing, uint16

import pickle

from functions import \*

from pylibfreenect2 import Freenect2, SyncMultiFrameListener

from pylibfreenect2 import FrameType, Registration, Frame

try:

from pylibfreenect2 import OpenGLPacketPipeline

pipeline = OpenGLPacketPipeline()

except:

from pylibfreenect2 import CpuPacketPipeline

pipeline = CpuPacketPipeline()

def sigint\_handler(signum, frame):

print("Got SIGINT, shutting down...")

quit()

def nothing(x):

pass

def pretty\_depth(depth):

depth = depth.astype(np.uint8)

return depth

signal.signal(signal.SIGINT, sigint\_handler)

fn = Freenect2()

num\_devices = fn.enumerateDevices()

serial = fn.getDeviceSerialNumber(0)

device = fn.openDevice(serial, pipeline=pipeline)

registration = Registration(device.getIrCameraParams(),device.getColorCameraParams())

undistorted = Frame(512, 424, 4)

registered = Frame(512, 424, 4)

listener = SyncMultiFrameListener(FrameType.Color | FrameType.Ir | FrameType.Depth)

device.setColorFrameListener(listener)

device.setIrAndDepthFrameListener(listener)

device.start()

bigdepth = Frame(1920, 1082, 4)

color\_depth\_map = np.zeros((424, 512), np.int32).ravel()

kernel = np.ones((5, 5), np.uint8)

thres = 0.5

while 1:

if listener.hasNewFrame():

frames = listener.waitForNewFrame()

color = frames["color"]

ir = frames["ir"]

depth = frames["depth"]

registration.apply(color, depth, undistorted, registered,bigdepth=bigdepth,color\_depth\_map=color\_depth\_map)

# #get kinect input\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# dst = pretty\_depth(cv2.resize(depth.asarray(),(int(512), int(428))))

depth = (depth.asarray()).astype(uint16)

depth = depth.reshape(424,512)

dst = depth

cv2.imshow("Depth", dst)

# #rectangular border (improved edge detection + closed contours)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

cv2.rectangle(dst,(0,0),(1920,1080),(40,100,0),2)

# #defined points approach #

spac = 30

(rows,cols)=dst.shape

# print(dst.shape)

shared = str("False")

counter = 0

for i in range(int(rows)):

for j in range(int(cols)):

if i>250 and i < 300:

if j>150 and j< 400:

if dst[i,j]>100:

if ((dst[i,j]<=2200)):

counter = counter+1

if counter> 500:

print('distance = ',dst[i,j], 'row = ', i, 'column = ', j)

shared = str("True")

break

with open("/home/max/shared.pkl","wb") as f:

print(shared)

pickle.dump(shared, f)

listener.release(frames)

key = cv2.waitKey(delay=1)

if key == ord('q'):

break

device.stop()

Arduino Script:

#define joyA 9

#define joyB 10

const int powerPin = 4;

char data = '\*';

int line\_A = 130;

int line\_B = 130;

bool inAuto = false;

void setup() {

pinMode(powerPin,OUTPUT);

Serial.begin(9600);

}

void startupProcedure() {

stop();

delay(300);

digitalWrite (powerPin,HIGH);

delay(100);

digitalWrite (powerPin,LOW);

delay(2500);

}

void move(int lineA, int lineB){

analogWrite(joyA,lineA);

analogWrite(joyB,lineB);

}

void stop(){

move(140,140);

}

void readingSerial(){

command = Serial.readStringUntil('\*')

if command == 'a'{

inAuto = true;

}

if command == "DONE"{

stop();

if command[0] == 'A'{

command.remove(0,1);

move(command, 130);

}

if command[0] == 'B'{

command.remove(0,1);

move(130, command);

}

}

}

void manualMode() {

if inAuto == false{

int xPosition = analogRead(A0);

line\_A = xPosition/8.0+80;

int yPosition = analogRead(A1);

line\_B = yPosition/4.6+40;

SW\_state = digitalRead(2);

delay(1);

move(line\_A,line\_B);

}

}

void loop(){

readingSerial();

manualMode();

}

## Appendix H: Team Contributions Table

**Table 11:** Team Contributions

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment/Team Member** | **Hardware** | **Software** | **Electrical** |
| **Max Bronson** | 15% | 50% | 60% |
| **Denny Mannakulathil** | 60% | 0% | 10% |
| **Oscar Montealegre** | 25% | 50% | 30% |