Clarkson University
ECE Department
10 Clarkson Ave, Potsdam, NY
EE 416/464 Computer Engineering Senior Design

# PACMAN

**P**ath **A**ssembling **C**artographer, **M**apping under **A**utonomous **N**avigation

## Phase II Detailed System Design

*Very Good Robotics*

Max Cutugno, Avery Oefinger, Adam Romlein

Fall 2019

## Table of Contents

# 1. Introduction

The PACMAN is an autonomous robot for surveying and path mapping. Described within this document is the detailed system design for the complete architecture. This system design will give a detailed description of all the major components in both software and hardware as well as the purpose of each.

## 1.1. Purpose

The proposed PACMAN is a self-driving ground robot capable of two main areas of expertise: indoors and outdoors. Outdoors, the robot is capable of following a path, specifically paved sidewalks, and returning real-world coordinates of that path back to a ground station (such as a remote PC). These real-world coordinates can be saved and distributed to other systems that require the absolute position of pathways. Such systems include autonomous lawnmowers, autonomous snow removers, autonomous delivery robots, and others. Indoors, the robot is capable of traversing hallways and generating a 3D-Model of the interior of the building (provided in a BIM-compatible format).

## 1.2. Scope

The scope of this project is on the outdoor capabilities of PACMAN. As a mandatory baseline, the system must be able to operate autonomously outside on these pathways and return these valuable real-world coordinates back to the operator. The detailed design outlined below explains how this will be accomplished.

## 1.3. Abbreviations and Acronyms

| | |
|---|---|
| PACMAN | Path Assembling Cartographer, Mapping under Autonomous Navigation |
| ROS | Robot Operating System |
| GPS | Global Positioning System |
| IEEE | Institute of Electrical and Electronics Engineers |
| BIM | Building Information Modeling |
| USB | Universal Serial Bus |
| TCP/IP | Transmission Control Protocol/Internet Protocol |

| | |
|---|---|
| DoD | Department of Defense |
| IMU | Inertial Measurement Unit |
| GMM | Gaussian Mixture Model |
| LiDAR | Light Detection And Ranging |
| UI | User Interface |
| GUI | Graphical User Interface |
| SSH | Secure Socket Shell |
| PC | Personal Computer |
| RC | Radio Control |
| PWM | Pulse-Width Modulation |
| GPIO | General-Purpose Input/Output |
| E-stop | Emergency Stop |
| GNSS | Global Navigation Satellite Systems |
| RTK | Real Time Kinematics |
| SSI | Synchronous Serial Interface |
| URL | Uniform Resource Locator |
| RPM | Revolutions Per Minute |
| FIFO | First In First Out (Queue) |
| URDF | Universal Robotic Description Format (XML file format) |
| SDF | Simulation Description Format |
| TB | Terabyte |
| GB | Gigabyte |
| NVMe | Non-Volatile Memory Host Controller Interface Specification |
| SSD | Solid-State Drive |

### 1.4. Definitions

**1.4.1.** Shall - The specification is mandatory to consider the final product successful.

**1.4.2.** Could - The specification is an optional addition to the final product.

**1.4.3.** Will - A promise of future work to be done.

**1.4.4.** System - The robot called PACMAN.

**1.4.5.** Floating - a digital or analog voltage line that is not connected to a source or ground.

**1.4.6.** Users - Informed persons operating PACMAN and / or its peripherals.

### 1.5. Standards

**1.5.1.** USB 2.0 type A & B

**1.5.2.** USB 3.0 type A & C

**1.5.3.** TCP/IP (Ethernet)

**1.5.4.** IEEE 802.11 (Wifi)

## 2. References

The above definitions and the below system description shall be defined and used in conjunction with the following documents.

**2.1.** IEEE Std. 830-1998 Recommended Practice for Software Requirements Specification

**2.2.** IEEE Std. 1233 Guide for Developing System Requirements Specifications

**2.3.** DoD Data Item Description for System/Subsystem Specification

**2.4.** Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." *ICRA workshop on open source software*. Vol. 3, No. 3.2, 2009.

**2.5.** ROS Drivers, *video_stream_opencv*, (2019), GitHub repository, https://github.com/ros-drivers/video_stream_opencv

**2.6.** Scene Parsing through ADE20K Dataset. B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso and A. Torralba. Computer Vision and Pattern Recognition

(CVPR), 2017. (http://people.csail.mit.edu/bzhou/publication/scene-parse-camera-ready.pdf)

**2.7.** Semantic Understanding of Scenes through ADE20K Dataset. B. Zhou, H. Zhao, X. Puig, T. Xiao, S. Fidler, A. Barriuso and A. Torralba. International Journal on Computer Vision (IJCV), 2018. (https://arxiv.org/pdf/1608.05442.pdf)

**2.8.** Ouster LiDAR, *ouster_example*, (2019), GitHub repository, https://github.com/ouster-lidar/ouster_example

**2.9.** Autonomous Systems Lab at ETH Zurich, *ethz_piksi_ros,* (2019), GitHub repository, https://github.com/ethz-asl/ethz_piksi_ros

**2.10.** Osrf, "SDF Specification," *sdf format*. [Online]. Available: http://sdformat.org/spec. [Accessed: 04-Nov-2019].

**2.11.** Wiki.ros.org. (2019). *urdf/XML - ROS Wiki*. [online] Available at: http://wiki.ros.org/urdf/XML [Accessed 4 Nov. 2019].

**2.12.** Gazebosim.org. (2019). *Gazebo*. [online] Available at: http://gazebosim.org/ [Accessed 4 Nov. 2019].

**2.13.** Adam Romlein, *pacman_ws*. [online] Available at: https://github.com/romleiaj/pacman_ws [Accessed 4 Nov. 2019].

**2.14.** Docs.blender.org. (2019). *Blender 2.80 Reference Manual — Blender Manual*. [online] Available at: https://docs.blender.org/manual/en/latest/ [Accessed 4 Nov. 2019].

## 3. Detailed Design for Phase I Hardware

The following sections describes the hardware design and functionality for the PACMAN vehicle. All the specified hardware units/modules are not designed by *Very Good Robotics* and are purchased from either a manufacturer or vendor. Please refer to the user manual of any of these units for more information.

### 3.1. System Power Design

Figure 3.1 below shows the power connections to each component along with switch logic and failsafes.

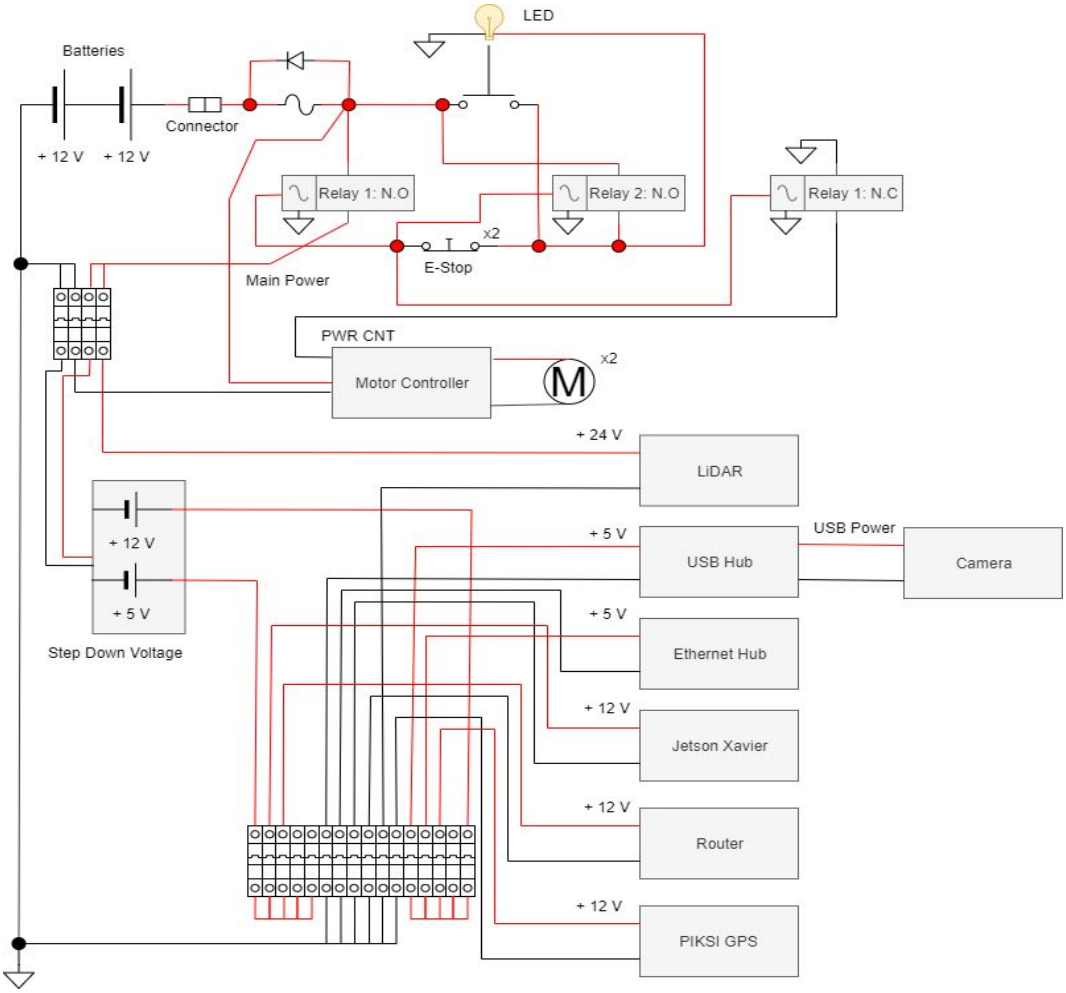



Figure 3.1. *System Power Block Diagram*

The system is powered by two 12V 46Ah batteries in series. These batteries provide a DC 24V supply to components such as the motor controller, step-down voltage block, and LiDAR. The other components are powered by 19, 12 or 5 volts through the step-down voltage block.

#### 3.1.1. Relay and Switch Logic

The system is designed to be turned on by a push button and shut off by one of two E-stops. By having a power button that does not latch reduces the risk of having a false start; when an E-stop is pressed the E-stop must be released and the push button must be pressed again to restart the system. If the start

button was a latching operation, after the E-stop was pressed and then released without toggling off the start button the system would start right away, perhaps unexpectedly and resume motion.

There are two relays used in the system. Relay one has two mutually exclusive switches, one normally open and the other normally closed. The normally open gate of relay 1 is used to provide power to the main system. The normally closed gate of relay 1 is used to ground the “power control” line of the motor controller that will normally prevent the motors of the system to keep running. When the “power control” line of the motor controller is floating the wheels of the motor controller are free to move. The normally open gate of relay 2 is used for the push button and E-stop control logic.

While not in use the batteries should always be disconnected from the system. When the batteries are connected the system is off and the relays are not powered. When the push button is pressed relay 2 is powered and stays powered after the push button is released. The output of relay 2 passes through 2 E-stops in series and then connects to the power of both relays 1 and 2. Thus, after the push button is pressed and the E-stops are not engaged both relays will be powered. When the E-stops are pressed both relays lose power.

### 3.1.2. RoboteQ Warnings and Failsafes

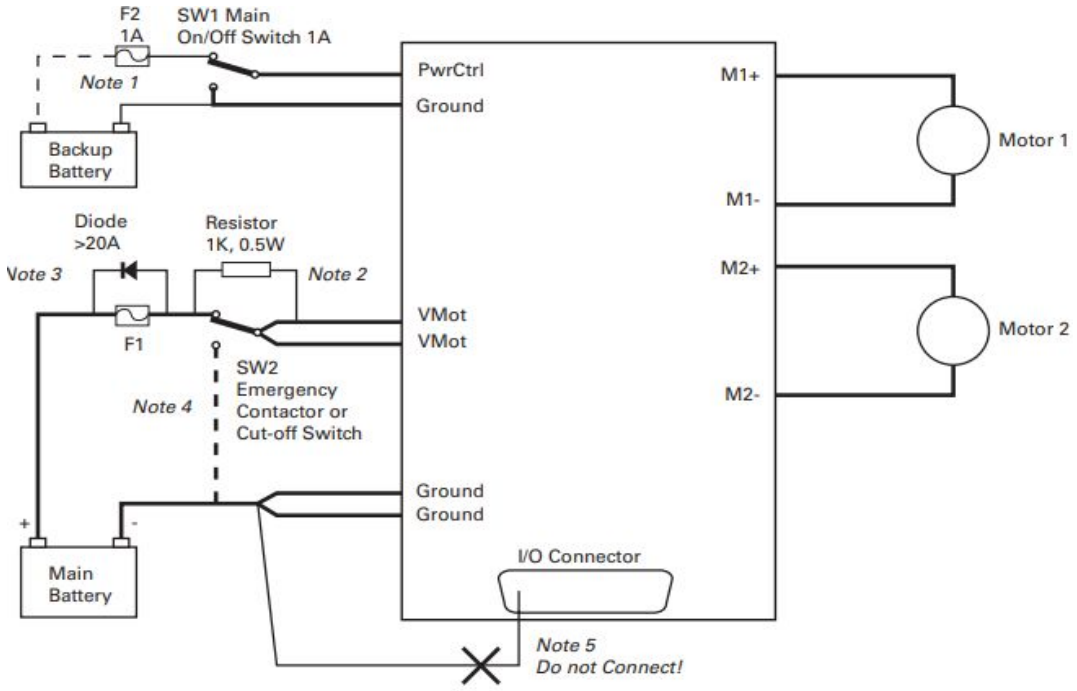



Figure 3.2. *Recommended Configuration for Motor Controller from RoboteQ*

Figure 3.2 above is used as a reference for the development of the power diagram. There are serious cases where the component or system could be damaged if certain precautions and failsafes are not implemented. The most important precautions of Figure 2 are notes 1, 2, 3 & 4.

**From RoboteQ User Manual:**

*Note 1*

> "The controller will stop functioning when the main battery voltage drops below 7V. To ensure motor operation with weak or discharged batteries, connect a second battery to the Power Control wire/terminal via the SW1 switch. This battery will only power the controller's internal logic. The motors will continue to be powered by the main battery while the main battery voltage is higher than the secondary battery voltage."

*Note 2*

> "Insert a 1K, 0.5W resistor across the SW2 Emergency Switch. This will cause the controller's internal capacitors to slowly charge and maintain the full battery voltage by the time the SW2 switch is turned on and thus eliminate damaging arcing to take place inside the switch. Make sure that the controller is turned Off with the Power Control wire grounded while the SW2

> switch is off. The controller's capacitors will not charge if the Power Control wire is left floating and arcing will then occur when the Emergency switch is turned on."

*Notes 3 & 4*

> "Voltage generated by motors rotating while not powered by the controller can cause serious damage even if the controller is Off or disconnected. This protection is highly recommended in any application where high motion inertia exists or when motors can be made to rotate by towing or pushing (vehicle parking). • Use the main SW1 switch on the Power Control wire/terminal to turn Off and keep Off the controller. • Insert a high-current diode (Digikey P/N 10A01CT-ND) to ensure a return path to the battery in case the fuse is blown. Smaller diodes are acceptable as long as their single pulse current rating is > 20 Amp. • Optionally use a Single Pole, Dual Throw switch for SW2 to ground the controller power input when OFF. If a SPDT switch cannot be used, then consider extending the diode across the fuse and the switch SW2."

There is no backup battery supply in the system and we will therefore monitor the battery voltage input via the motor controller. If the battery voltage ever gets close to this 7 volt threshold due to low battery levels the compute node will command the motors to stop moving.

To prevent arcing in the initial connection, the internal capacitors of the motor controller must be charged before full power is supplied to the motor controller. Without overcomplicating the design the resistor from note 2 is not added to the design and instead the motor controller is always powered while the batteries are connected to the system. Since the arcing is only concerned with damage to the plug-in connection to the batteries, and our connection is industrial-grade, it is an acceptable design. The batteries must be disconnected from the system once the system is no longer in use.

Notes 3 and 4 mention how if the wheels for the system are still moving while the motor controller is off the electric motors can source current and potentially damage the system. This case is likely to happen if the system's E-stop is pressed while moving. To avoid potential damage a diode rated for over 20 amps feeds back current to the batteries in case the breaker flips, so this feedback current has a path back to the battery.

**WARNING** - There are 4 MOSFETs used to control the motors. The RoboteQ user manual mentions how it is possible for any of the MOSFETs to fail. Bellow in figure 3 shows all the possible combinations of shorted MOSFET switches.

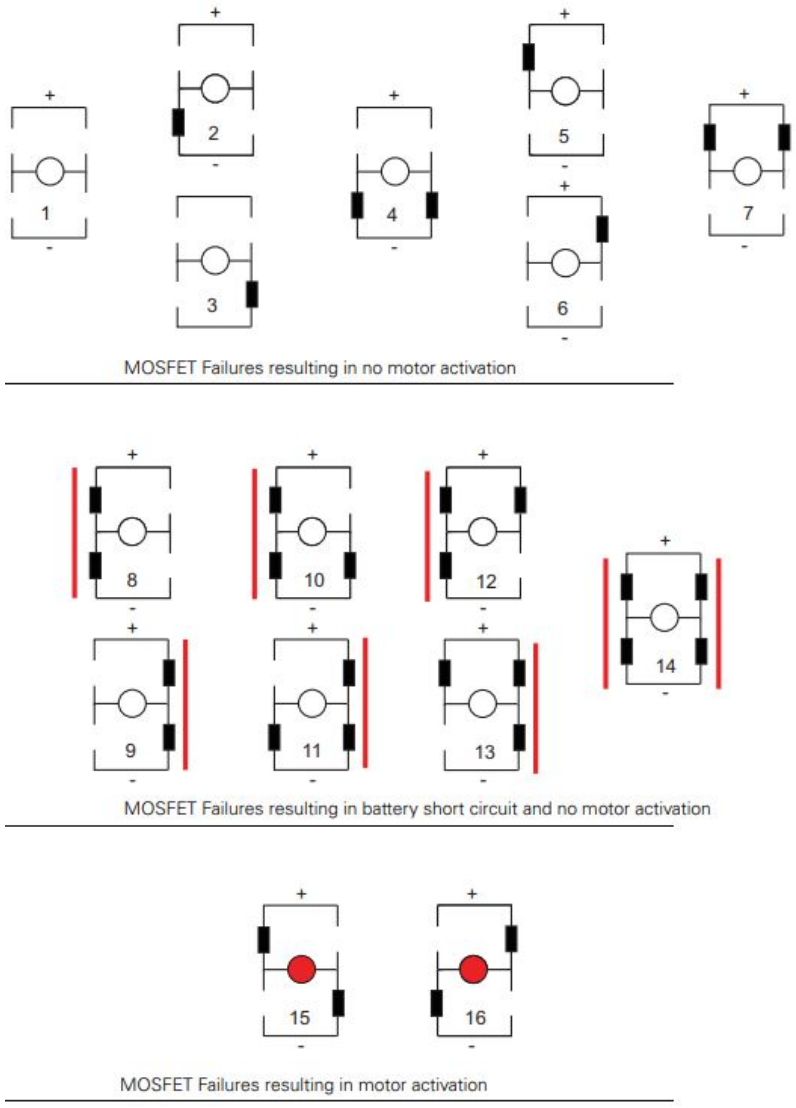



Figure 3.3. *Motor Controller MOSFET Switch Failure Cases*

Figure 3 above shows how there are 2 cases of 16 where the motor controller fails in an active state. In this state the motors would not be able to be stopped if an E-stop is pressed.

Table 3.1 below shows the specified values for all the hardware components operation (voltage and current).

| Hardware Component | Voltage (volts) | Current (amps) | Notes |
|---|---|---|---|
| LiDar | 22-26 | Max: 1 | 14-20 W (22 W peak at startup) |
| USB Hub | 5 | 2.5 | |
| Ethernet Hub | 5 | 2 | |
| Jetson Xavier | 9-19 | 3 | 10-30 W |
| Router<br>Cradlepoint BR600 | 9-18 | 1.5 | Idle<br>typical=350mA@12V (4.3W)<br>worst=700mA@12V (8.3W)<br>Tx/Rx<br>typical=600mA@12V (7.2W)<br>worst=1200mA@12V (14.4W) |
| PIKSI Duro GPS | 12-24 | 1 | 12 W |
| Camera<br>Logitech C920 | 5 | Less than 0.5 | Powered via USB |
| RoboteQ Motor Controller<br>Model: MDC2230 | Max: 35 | Max: 60 per channel | |
| DC/DC Step Down Voltage | Input: 24<br>Output: 12 | Output: 10 | 120 W<br>Waterproof |
| DC/DC Step Down Voltage | Input: 24<br>Output 5 | Output: 10 | 60 W<br>Waterproof |
| DC/DC Step Down Voltag | Input: 24<br>Output: 19 | Output: 5 | Waterproof |
| Push Button | Used at 24 | - | |
| LED | 24 | - | 0.7 W |
| Relay 1 | 24 | - | 200 A current passthrough |
| Relay 2 | 24 | - | 200 A current passthrough |
| E-Stop | Used at 24 | - | Low current passthrough |
| Power Chair Wheel Base<br>Vendor: Golden Tech.<br>Model: Compass HD | 24 | Max: 80 per motor | 0.3-0.8 ohms |
| Breaker | Max: 42 | 120 | |
| Battery | Supply 24 | - | 46 Ah |
| Encoders | IP: 4.5-5.5<br>OP: 2 | IP: 27-85 mA<br>OP: 0.05-0.25 mA | Powered by Motor Controller |

Table 3.1. *Component Voltage and Current Rating*

### 3.2. System Hardware Components

Figure 3.4 above shows the hardware connection of each main component to one another. The two types of physical connections are Ethernet and USB.

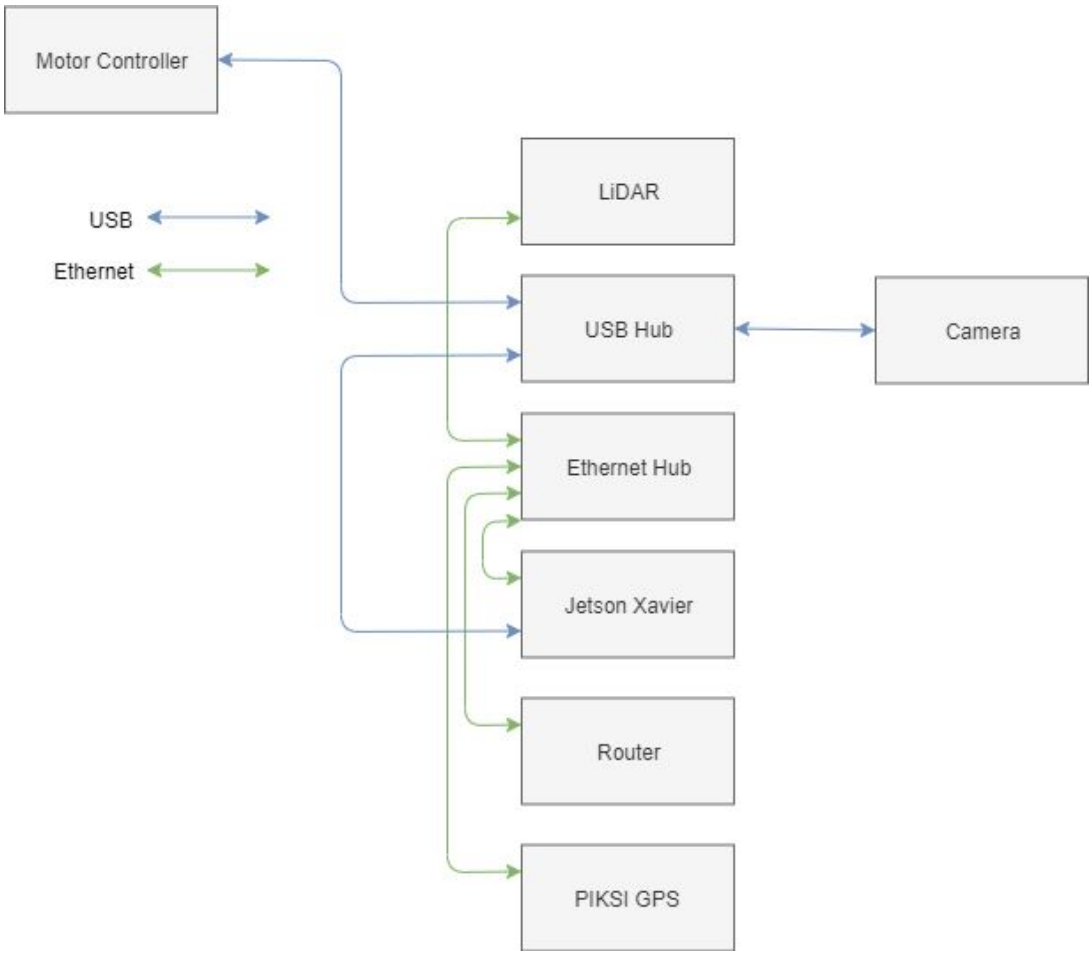



Figure 3.4. *System Hardware Block Diagram*

Table 3.2 below expands on the standard sub-types and the data rates of each hardware and protocol used.

| Hardware Component | Communication Standard | Data Rate (Gbps) |
|---|---|---|
| LiDAR | Ethernet, RJ45 Port | 10 |
| USB Hub | None | - |
| Ethernet Hub | None | - |
| Jetson Xavier | Ethernet, | 10 |

| | | |
|---|---|---|
| | USB 3.1 | 10 |
| Router | Ethernet,<br>Cellular 3G/4G/LTE | 10<br>50 Mbps |
| PIKSI GPS | Ethernet | 10 |
| Camera | USB 2.0 | 0.48 |
| RoboteQ Motor Controller | USB | Not Specified in Manual |

Table 3.2. *System Hardware Communication Standards and Data Rates*

A detailed description of every component shown in the system hardware block diagram in Figure 3.4 is given below.

### 3.2.1. Vehicle Computer

- **Functional Specification**

The Vehicle Computer hardware *Jetson Xavier* (vended by Nvidia) is used to collect and process data from the peripherals and sensors (GPS receiver, LiDAR sensor, motor controller, camera, base computer commands) and react to control the motor control (and therefore controlling the movement of the vehicle). It is the personified brain of the vehicle; its OS is a modified Linux Debian distribution and it runs the sensor suite by using a ROS framework. Three upgrades were made to the stock Xavier. The first in Figure 3.5, was a 1 TB NVMe V-NAND SSD.

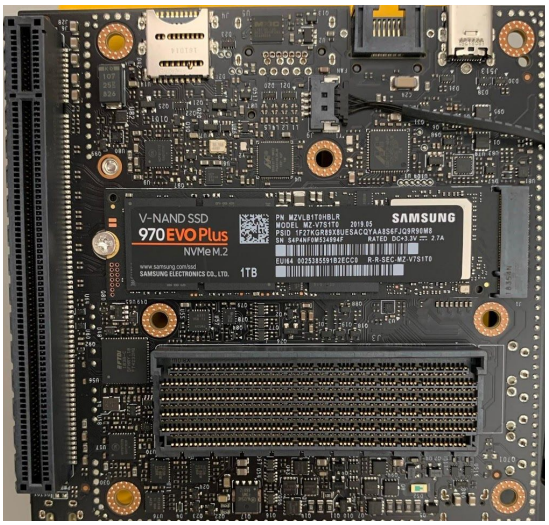



Figure 3.5. *Samsung 970 Evo Plus Installed on the Underside of Xavier*

This allows for an extremely high data rate (~3GB/s) for doing large data collects with a high volume of information. Since the Xavier does not come with Bluetooth or WiFi by default, the second upgrade required an additional external chip to be installed (see Figure 3.6).


Figure 3.6. *Intel 8265 Wireless Card*

This enabled the bluetooth controller to connect to the Xavier, as well as the Xavier to connect to any WiFi networks. The final upgrade was a 3D printed upgrade. Default, the Xavier does not come with any mounting options or support for the antennae required for the WiFi card. So, a mounting option was printed, in process shown in Figure 3.7.

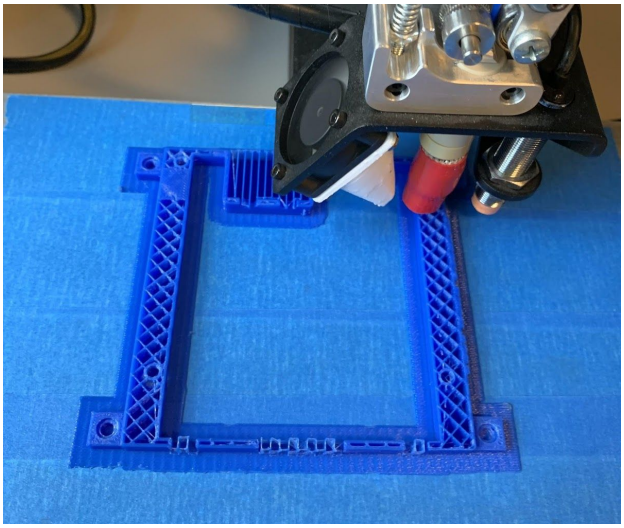


Figure 3.7. *3D Printing Mounting Block and Antennae Support for Xavier*

- **Signals**
  - Ethernet: This is used to communicate with the cellular network, local network, GPS Receiver, and LiDAR sensor (please see the Sections 3.2.2, 3.2.4, and 3.2.6 to refer to these devices).

There is only one ethernet port for the Jetson, therefore a 4-port ethernet hub is connected to the Jetson through an ethernet cable so all ethernet peripherals can be connected.

- USB 3.0: This is used to communicate with the Motor Controller and Camera. There are only two USB ports, therefore a 4-port USB hub is connected to the Jetson so that both the Motor Controller, Camera, and other USB peripherals can be connected at once.

#### 3.2.2. GPS Receiver

- **Functional Specification**

The GPS Receiver hardware *Duro* (vended by Swift Navigation) is responsible for contriving GPS/IMU data of the robot with centimeter accuracy through Dual-Frequency RTK GNSS; this is done using trilateration techniques, and precise coordinates are obtained using kinematics calculations. The GPS receiver is a critical component to accumulating physical world coordinates (consisting of latitude and longitude values) of its location. These points are streamed to the Jetson Xavier at a maximum rate shown in the table above (Table 3.2). The hardware is composed of two main components: the Duro Evaluation kit that processes raw antenna data and the GNSS Antenna itself. These two components are connected through a GNSS cable. The *Duro* is a ruggedized version of their *Piksi* module. When referring to the computation node, these terms are used interchangeably.

- **Signals**

- Ethernet: The *Duro* has an ethernet port that sends GPS/IMU data using TCP/IP to the Jetson Xavier. This data, when sent to the Jetson, is processed again through a ROS driver described in Section 4.9.

#### 3.2.3. Motor Controller

- **Functional Specification**

This device is a brushed DC Motor Controller *Roboteq MDC2230* that is responsible for setting and regulating the speed of the two high torque motors that control the movement of the vehicle. Furthermore, this hardware provides information concerning the power consumption, battery life, and encoder information from the high torque motors. The Jetson Xavier sets angular velocity and receives status messages from this device. This motor control has several modes of operation. In our case, the closed loop control is used for this application. This is because the motor encoders capture motor data, which can then be used to adjust the current status of the control (hence closed loop).

- **Signals**

- USB (Serial): The *Roboteq* motor controller is configured and regulated by commands within a MicroBasic or C++ script sent over serial communication through USB from the Jetson. This script is used through the ROS driver described in the next section (Section 4.8).

- Motor Driver: The *Roboteq* motor controller has two channels that send PWM signals from the controller to each motor. Channel one has Mot1(+) and Mot1(-) that are respectively connected to the positive and negative rails of the 1st motor (see Figure 3.8). Channel two has Mot2(+) and Mot2(-) that are respectively connected to the positive and negative rails of the 2nd motor. This allows proper control over the current supplied to each motor, which effectively controls each wheel given an arbitrary angular velocity.

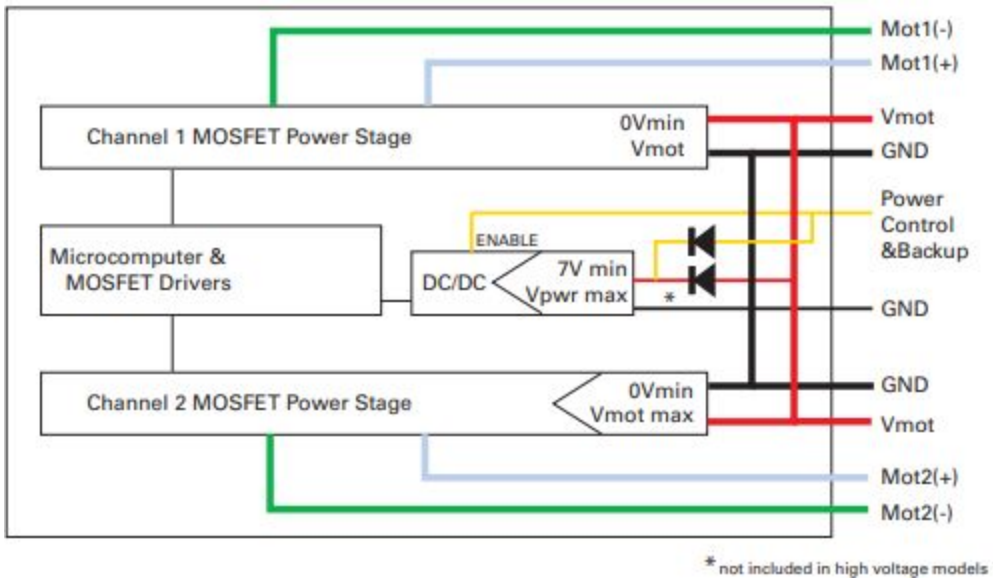



Figure 3.8. *Motor Controller Power Signals*

- Encoder Inputs: The *Roboteq* motor controller has I/O ports ENC1a/b to ENC2a/b for the encoder inputs. SSI (with resolution up to 16 bits) is the protocol used for communication between the encoders and the motor control. The types of information that is communicated through these ports are:
    - Speed or position feedback
    - Safety Stop
    - Emergency stop
    - Motor Stop (deadman switch)
    - Invert motor direction

- Forward or reverse limit switch
- Run MicroBasic Script
- Load Home counter

#### 3.2.4. LiDAR Sensor

- **Functional Specification**

The hardware *OS1-16* (vended by Ouster) is a LiDAR sensor that sends acquires mobile LiDAR data in a 360º horizontal view and approximately 20º vertical view from exposed surfaces within the external environment around the vehicle. In addition to LiDAR acquisition, this sensor is also capable of collecting IMU data of the vehicle. Both data are streamed to the Jetson Xavier to be further processed. It is worth noting that this device has a ROS driver within the Jetson to interpret the LiDAR data sent; the LiDAR data allows the vehicle to have a sense of 3D geometry of its surroundings (see section 4.3).

- **Signals**

- Ethernet (TCP/IP protocol): The LiDAR communicates to the Jetson through ethernet cable using TCP/IP protocol for status messages.
- Ethernet (UDP protocol): The LiDAR Sensor sends UDP data packets (12608 Bytes in length) on port 7502 for LiDAR data to Jetson.
- Ethernet (UDP protocol): The LiDAR Sensor sends UDP data packets (48 Bytes in length) on port 7503 for IMU data to Jetson.

#### 3.2.5. Camera

- **Functional Specification**

The camera is a *Logitech HD Pro Webcam C920* is that captures RGB images at a specified rate in front of the vehicle and sends this data to the Jetson. This sensory device gives the vehicle vision; image segmentation (processed within the Jetson) takes these images to follow a path (see section 4.2). This camera is UHD (2304 x 1296) with 20-step autofocus feature.

- **Signals**

- USB 2.0: connects to Jetson Xavier to stream image data to a ROS driver to be further processed (see Section 4.1).

#### 3.2.6. WiFi Router

- **Functional Specification**

The Wifi Router is a *COR IBR600 Series Cradlepoint* that is simultaneously a 3G/4G (cellular) modem and a WiFi router. This router is mounted to the vehicle to allow the Jetson to have access to the internet via the cellular network. Moreover, the Jetson is accessed via a remote computer through ssh, or the base station can connect to the local network onboard the system to be able to view ROS services and messages with the graphical user interface.

- **Signals**
  - 10/100 Ethernet (LAN): This connects to the Jetson Xavier through an ethernet cable (see ethernet speeds in Table 3.2) The Jetson is assigned a static IP through the router of 198.168.1.10. The I/O options to the Cradlepoint are shown in Figure 3.9.

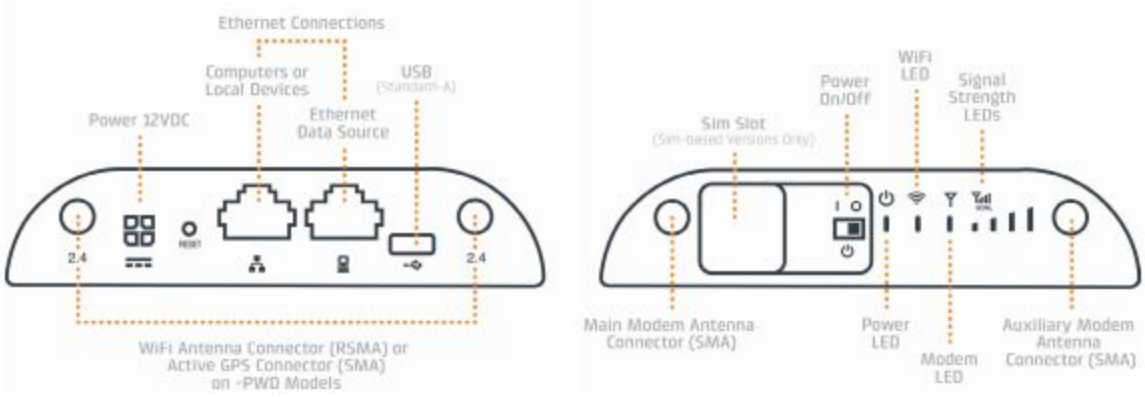



Figure 3.9. *Cradlepoint Router IO*

### 3.2.7. Motor Encoders

- **Functional Specification**

Two *E3 Optical Kit Encoders* (vended by US Digital) are used to provide rotary data of both high torque motors to the motor controller. More specifically, it is used to measure the RPM of the motor shaft. This RPM data is output to the motor controller and is passed through via USB to the vehicle computer. The motor controller uses the RPM data as closed-loop feedback for the motor control.

- **Signals**
  - 2 Pin Digital output: the encoder has two pins for two channels that send quadrature square wave outputs which encode the rotary position of the train motor. These pins are connected to the encoder inputs of the motor controller and processed through ROS (see Section 4.8).

### 3.2.8. Vehicle Chassis

- **Functional Specification**

The chassis is is taken from the bottom of a *Golden Technologies* Power Chair (Golden Compass HD, Model ID: GP620). It is the primary physical frame that holds both 12V batteries, the left and right high torque motors, the left and right drive wheels, and the four additional smaller wheels with casters. Additional hardware has been mounted on the plywood beams and surfaces on top of the base.

- **Joints**
  - Right and left Motorized Joints: These are horizontal revolutionary joints that connect the two high torque motors to the left and right motorized middle wheels. Wheel centers are placed 1.74 feet away from each other. The middle wheels are 9.7 inches in diameter.
  - Front wheels and casters: The front casters each have vertical revolutionary joints (which are attached to the chassis) that are placed approximately 1.74 feet away from each other and allow the front wheels to turn. The small wheels are attached to the caster with horizontal revolutionary joints. These wheels are 5.7 inches in diameter.
  - Back wheels and casters: Similar the front casters, they are vertical revolutionary joints that are connected to the 5.7 inch diameter wheels through horizontal revolutionary joints, however are placed 1.32 feet away from each other.

#### 3.2.9. High Torque Motors

- **Functional Specification**

The high torque motors are attached to the wheels and base frame that came with the chair. The motors are used to precisely control the movement of the vehicle and are connected to the motor controller. Since there is a motor for each the left and right wheel, the vehicle relies on a differential drive configuration.

- **Signals**
  - PWM cables: Each motor has two wires that are used to receive PWM signals from within the motor controller to control the power supplied to the motor.
  - Brake leads: Each motor has 2 wires (ground and a 10V lead) that can be used to disengage the high torque motors in case of power failure or emergency stop. When > 10V is applied between these leads, the motors are engaged, otherwise the motors are disengaged.

## 4. Detailed Design for Phase II Software

ROS [2.4] is an open-source robotics meta-operating system, acting as a middleware between hardware and software, as well as Python and C++. It provides multiple useful layers of abstraction, and uses a publish / subscribe model popular in robotics. All development for PACMAN is based off this

architecture, and all following descriptions of each module are described using ROS nomenclature, with some definitions outlined below.

| Concept | Definition |
|---|---|
| Nodes | An atomic process, supported in C++ or Python, that performs computation. |
| Messages | A strictly typed data structure to pass data between nodes. |
| Topics | A human-readable string representing a specific ROS URL in its internal server to publish and subscribe to messages. |

Table 4.1. *ROS Concepts and Definitions*

The following diagram (Figure 4.1) outlines the major software modules that will be implemented in the design. It is outlined in a ROS-like format, with ROS-specific messages interfacing in between each module, illustrated as a ROS node.

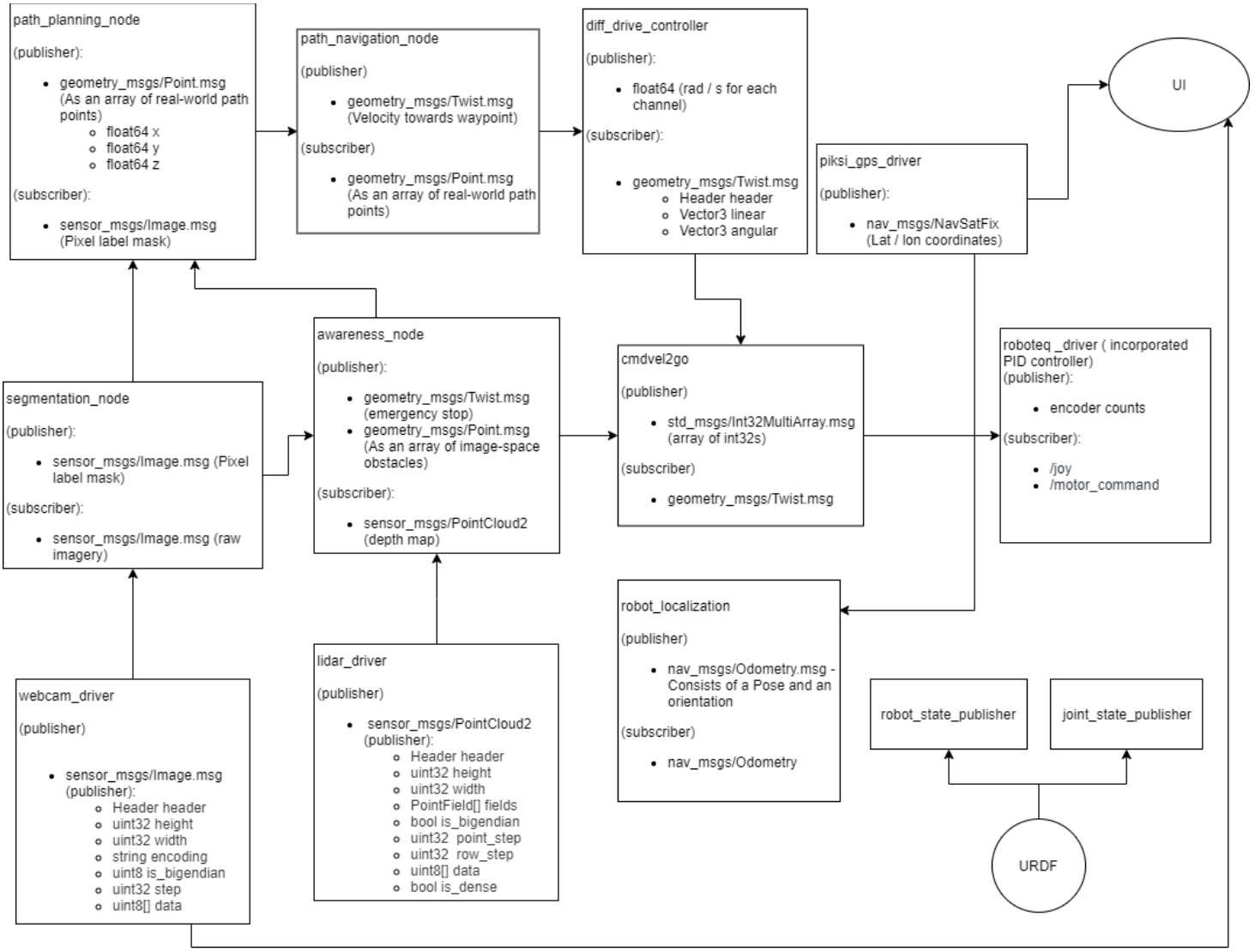



Figure 4.1. *ROS Node Diagram of Software Architecture*

A high-level functional description of each node and the input / output to each node is detailed here.

### 4.1. Webcam Driver

- **Functional Specification**

This driver [2.5] will be responsible for interfacing with the connected camera and translating the information into the ROS ecosystem. Based on the USB standard used for our webcam, the driver will determine the width, height, depth, and encoding of the pixels in the image. This will then be used to fill the Image message defined in ROS, and published to a topic of type image raw.

- **Input / Output Nodes**

This node only has an output, the Image message containing raw camera data. This imagery will be directly transferred to the Segmentation node (4.2), and the UI to be displayed. If higher framerate processing is required, this may be sent to a GMM node as well.

- **Messages**
  - sensor_msgs/Image.msg
- **Topics**
  - webcam/image_raw

### 4.2. Semantic Segmentation

- **Functional Specification**

This node will take in raw imagery from the webcam, and run a Semantic Segmentation algorithm produced by MIT [2.6, 2.7] on this imagery. This is computationally intensive and takes several seconds, usually operating at 1-3Hz. Originally, this node produced a label for each pixel in the image, from a choice of 150 categories. The input is a standard image as shown in Figure 4.2 (left).

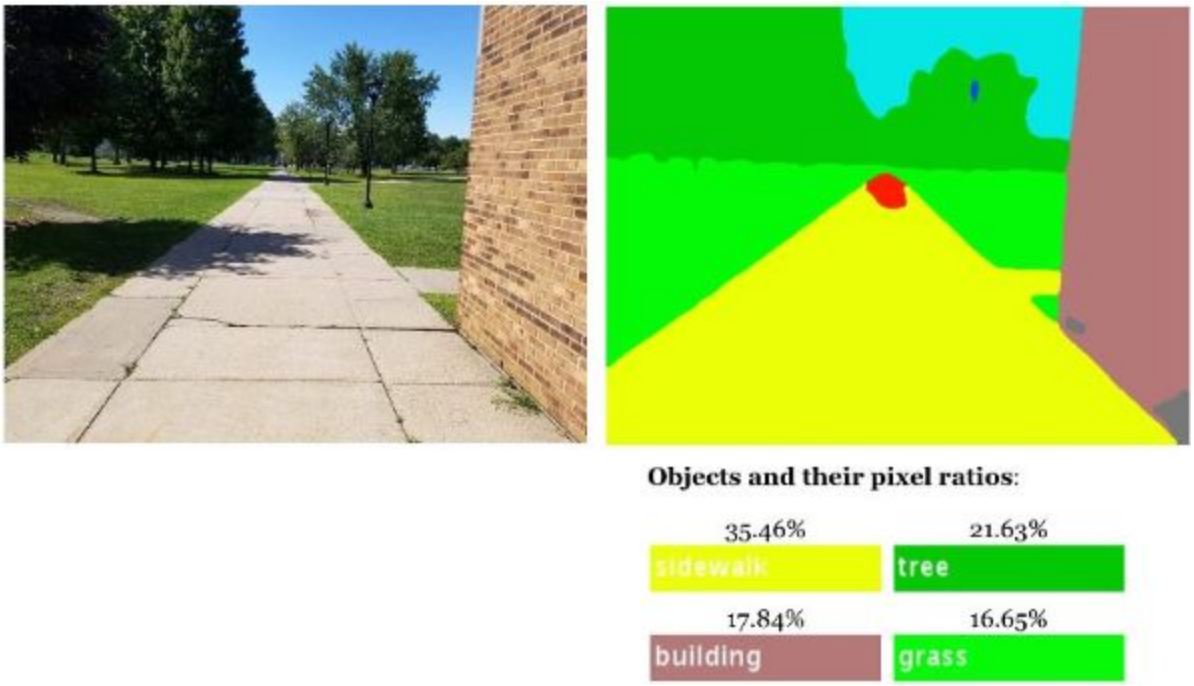



Figure 4.2. *Input (left) and Output (right) of Semantic Segmentation*

The output shown in Figure 4.2 (right) is every pixel labeled with the corresponding best guess. The top 4 categories and their percentage of the image is shown as well, on the very bottom right. The categories PACMAN is interested in is limited to the path-like ones and people, which consists of 3

main user-defined categories: Diveable, Person, and Other. Since we have such a limited scope out of the 150 classes, the output was adjusted from the strict pixel-label classification. The “Driveable” categories’ (sidewalk, path, runway, road, floor) probabilities (carefully extracted from the network) were summed and displayed in the green channel of an empty image. The “Person” category was placed in the red channel, and the blue channel was left blank. The result is shown in Figure 4.3.

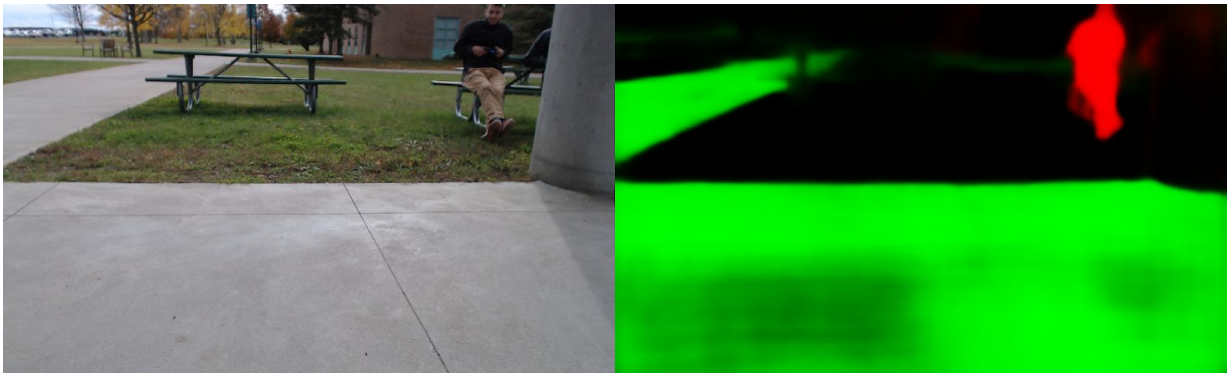


Figure 4.3. *(left) Input Image (right) Output Image: Drivable = Green and Person = Red*

This visualization much more clearly shows the areas which look reasonable to drive in, or have a high probability, and the areas that don’t.

- **Input / Output Nodes**

The input to this node will be the Webcam Driver node (4.1). The output will be directed to both the Path Planning (4.5) and Awareness (4.4) nodes for a better semantic understanding of the world in front of the system.

- **Messages**
  - sensor_msgs/image_raw
  - Custom message for pixel label mask
- **Topics**
  - webcam/image_raw
  - webcam/image_segmented

### 4.3. LiDAR Driver

- **Functional Specification**

This driver [2.8] will serve as an interface between the raw Ouster LiDAR data and the ROS ecosystem. It will determine the encoding and data ordering, and translate this into a ROS-compatible format. An example data visualization of this LiDAR data is shown in Figure 4.3.

Figure 4.3. *Example LiDAR Point Cloud Data*

An IMU is also onboard the LiDAR, which acts as a source of odometry..

- **Input / Output Nodes**

This node serves as an output-only node, sending the depth information to both the UI node (4.10) to be displayed, and also the Awareness node (4.4), to determine if any objects are too close. The IMU is connected to the localization node (4.12).

- **Messages**
  - sensor_msgs/PointCloud2
  - sensor_msgs/Imu
  - nav_msgs/Odometry
- **Topics**
  - os1_cloud_node/points
  - os1_cloud_node/imu

### 4.4. Path Planning

- **Functional Specification**

This node deals with processing the segmented image into a real-world calibrated path. A calculated homography will be applied to the image to get a real-world position of each pixel. The homography is

created through a user input calibration method. The user individually identifies four corner points on the left 2D image and corresponds them to four chosen points on the right grid. The four points on the right grid represent the real world distances from the camera to the chosen features on the left 2D image.

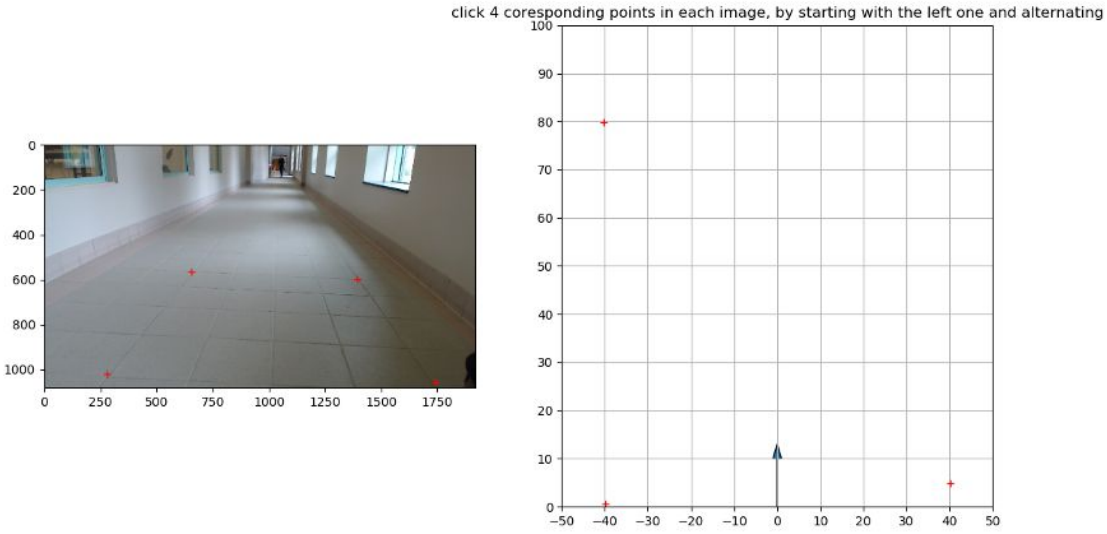



Figure 4.4 *Example Camera Calibration Method*

Once the calibration is complete the segmented images are transferred into a probability map shown in figure 4.5. In the figure below white indicates a high probability of drivable path while black is the opposite.

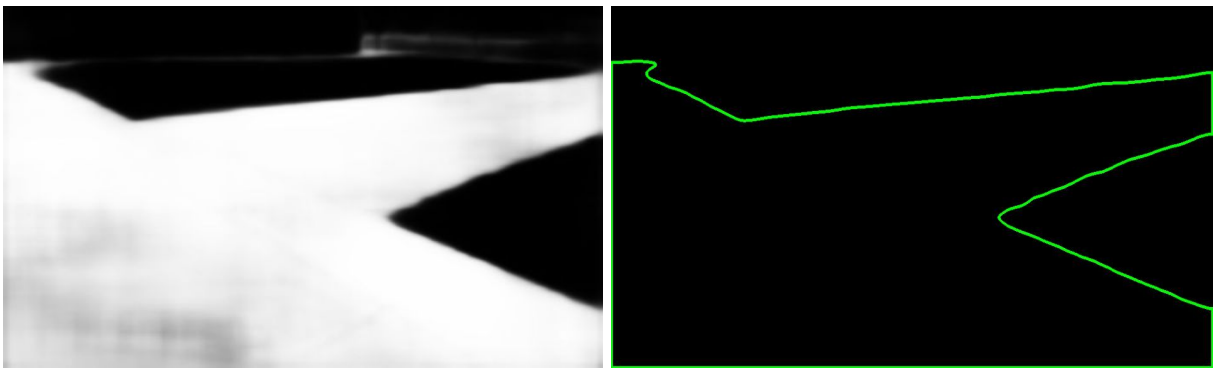


Figure 4.5 *Segmentation to Path Contour Creation*

Once a probability map has been created it is transferred into a path contour, shown in the right of figure 4.5. From this path contour and created homography a best-guess for the path will be generated based on Dijkstra's shortest path algorithm. This will output a series of (X,Y) coordinates (assuming a constant Z, or level driving ground) consisting of the best path for the system to take.

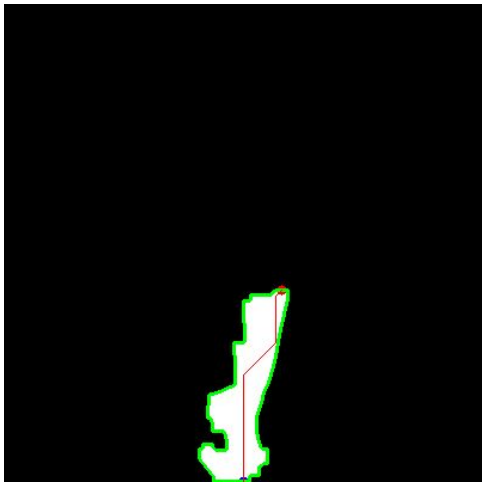


Figure 4.6 *Dijkstra's Algorithm of Path Contour Image*

- **Input / Output Nodes**

This node will take input from the Segmentation node (4.2) for path planning, and Awareness node (4.4) for object avoidance, and the output velocity will feed directly into the Differential Drive Controller (4.7).

- **Messages**
  - Custom message for pixel label mask
  - geometry_msgs/Point array
- **Topics**
  - awareness/near_objects
  - webcam/image_segmented
  - path/real_world_xyz

### 4.5. Path Navigation

- **Functional Specification**

The Path Navigation node receives arrays of points at a time. These points are waypoints designating the desired location of the robot. A "point filter" function is used on the incoming waypoints. The point filtering process is designed to prevent latency issues. Once the camera takes a picture and is segmented, segmentation takes a few seconds after the photo was taken. While segmentation is running the robot is expected to keep traveling based on know knowledge of where the path is. Once segmentation produces an array of waypoints for the Path Navigation node there will already be points in the array that have been passed. Thus the point filtering process looks at the current position of the robot and filters out points that have already been passed. The array of valid waypoints is put into a

FIFO queue and the path node handles each point at a time. When the queue is emptied the robot stops at the last location. An internal PID controller is implemented to handle transitioning from point to point. A series of published velocity command are produced for the differential driver.

- **Input / Output Nodes**

This navigation node will pull the path from the Path Planning node (4.4). The output velocity vector will go directly into the Differential Drive Controller node (4.6).

- **Messages**
  - pacman.msg/pointArray.msg (custom message)
  - geometry_msgs/Twist
  - nav_msgs/Odometry.msg
- **Topics**
  - /cmd_vel
  - /odom
  - /path_points

### 4.6. Differential Drive Controller

- **Functional Specification**

This node provides a layer of abstraction between controlling a general velocity that the system should go and the specific angular velocity that each wheel should spin. A generic velocity vector should be input, with a specific output from -1000 to 1000 (“go units”, see 4.7) output to each motor.

- **Input / Output Nodes**

This node will take input velocities from multiple sources including a bluetooth controller, the autonomous driving mode, or other manual input. The output is directly to the Roboteq Motor Controller Driver (4.7). This output is determined by 2 kinematics equations (one for each wheel) and the physical dimensions of the system itself.

- **Messages**
  - geometry_msgs/Twist
  - Int32MultiArray for motor commands.
- **Topics**
  - /cmd_vel
  - /roboteq_motor_command

### 4.7. Roboteq Motor Controller Driver

- **Functional Specification**

This driver will act as the interface between the proprietary Roboteq software and the robot ecosystem. It must be able to communicate serially to the controller and pull off the correct data from fault flags to motor speed, as well as publish the necessary commands to keep the wheels spinning at the specified velocity subscribed to. This required writing in C++ using the provided API by Roboteq. This translation publishes a unit of speed, from -1000 to 1000, termed as “go units”. Our atomic node takes in an array of two values, one for each motor, in this “go unit” scale and outputs it directly to the motors. There is an emergency stop function constantly checking the joystick input to check if the stop button, ‘X’, is pressed. The system will not move when this button is pressed, only pressing the ‘O’ button will release the E-stop. The two status functions read the encoder value at a rate of 50 Hz from each wheel. The encoders are a part of a closed loop PID control system. The gains are experientially adjusted and tested.

- **Input / Output Nodes**

The angular velocity will come from the manual controller, or from the Differential Drive Controller (4.6) for the usual manual and autonomous mode commands.

- **Messages**
  - Int32 for each encoder value
  - Int32MultiArray for motor commands
- **Topics**
  - /roboteq_motor_command
  - /roboteq/left_encoders
  - /roboteq/right_encoders
  - /joy

### 4.8. Bluetooth Controller Driver

- **Functional Specification**

This driver interfaces to a bluetooth ps4 controller allowing the user to manually control robot. The left joystick of the controller controls the linear and angular velocity of the robot; forward and backward control linear velocity while left and right control angular velocity. However, moving the joystick on its own will not move the robot. In tandem, the LB or LT buttons on the bluetooth controller must be pressed for velocity messages to be sent to the robot. The LT button is the slow speed mode and will let

the robot travel at a max speed of 0.4 m/s. Pressing the LB button will allow the robot to go at a top speed of 1 m/s. These speed limit values are adjustable in the ps4 configuration file. The manual controller also allows the user to activate a virtual emergency stop to the motor controller. If the (X) button is pressed the emergency stop is enabled. When the emergency stop is enabled no command velocities are registered by the motor controller and the robot just drifts to a stop. The emergency stop can be disabled by pressing the (O) button on the controller. If the manual controller ever loses connection the driver will give the motor controller a zero command velocity (active braking).

- **Input / Output Nodes**

The input to the bluetooth controller driver is the ps4 controller and thus does not receive any input for other nodes. The output message of the node is a cmd_vel message that the twist mux node (4.9) is subscribed to.

- **Messages**
  - sensor_msgs/Joy
  - geometry_msgs/Twist
- **Topics**
  - /joy
  - /joystick/cmd_vel

### 4.9. Twist Mux Node

- **Functional Specification**

The twist_mux node takes potentially 255 cmd_vel inputs and prioritises them from 255 to 1 (the larger number has a greater priority). The priorities and input cmd_vel topics can be configured in the twist_mux_topics.yaml configuration file. The priority of the manual controller is set to 255 while the priority of the autonomous mode, path navigation node, is set to 1.

- **Input / Output Nodes**

There are currently two input cmd_vel messages that the twist mux node is subscribed to: from bluetooth controller driver (4.8) and from path navigation node (4.5). The output cmd_vel message is sent to the RoboteQ motor controller driver (4.7).

- **Messages**
  - geometry_msgs/Twist
- **Topics**
  - joystick/cmd_vel
  - pathnav/cmd_vel

- /cmd_vel

### 4.10. Swiftnav Duro / Piksi Driver

- **Functional Specification**

This driver [2.9] will act as an interface between the proprietary Swiftnav software and the ROS ecosystem. The GPS coordinates must be able to be published to a topic, with the corrections over WiFi from the base station occurring through the same node. In order to receive these corrections out in the field when the system is on a cellular network, a VPN was setup to act as a middleman between the system and the base station. The IMU onboard the GPS will act as a source of odometry.

- **Input / Output Nodes**

No nodes will flow into the GPS. It's output feeds directly into the UI (4.11) node for visualization and mapping. These coordinates will be accumulated over the course of a mapping run, and assembled into a map of the path at the end. The IMU output is directed towards the localization node (4.12).

- **Messages**
  - sensor_msgs/Imu
  - nav_msgs/navsatbestfix
- **Topics**
  - /piksi/wifi_corrections
  - /piksi/navsatbestfix
  - /piksi/imu

### 4.11. User Interface (UI)

- **Functional Specification**

The user interface is a means to view sensor information and robot status messages. The GUI will have an emergency stop button that will toggle the activation of the motor controller. The GUI will also show motor driver statuses such as voltage, current, and temperature of the motor driver. On the right half of the GUI will be a tabbed view of four windows: camera & segmentation, gps map, LiDAR, and path creation. The camera & segmentation tab will show the live camera feed as well as the camera segmentation processed every couple seconds. The gps map tab will show an overhead gps image of the robots surroundings. As the robot moves on the path the odometry point of the robot will be plotted onto the map. The LiDAR tab will show the real time generated point clouds from the robot. The path creation tab will show the intended path for the robot inside the created contour image (referenced in Figure 4.6).

- **Input / Output Nodes**

All sensor nodes will have a connection to the UI node. The output will return back to the Differential Drive Controller (4.6) in the form of an emergency stop command.

- **Messages**
  - sensor_msgs/Image
  - sensor_msgs/PointCloud2
  - geometry_msgs/Twist
  - Custom message for pixel label mask
  - Float64 for lat/lon coordinates
  - Return messages containing fault flags, motor current, temperature, motor speed, etc.
- **Topics**
  - duro/best_navsatfix
  - roboteq/controller_status
  - /cmd_vel
  - os1_cloud_node/points
  - os1_cloud_node/imu
  - webcam/image_raw
  - webcam/image_segmented

### 4.12. Robot Localization

- **Functional Specification**

This node also takes an associated covariance matrix and boolean matrix with each input source of odometry, IMU, or Pose, and outputs a weighted, fused odometry using an extended kalman filter. This allows for an accurate autonomy mode, and for creating well-defined and oriented maps of the sidewalks.

- **Input / Output Nodes**

This localization node takes input from all sources of position, velocity, and acceleration. This includes the IMUs aboard the Duro and the LiDAR, the motor encoders, and the GPS lat/lon received via the Duro. Outputs this fused odometry to both the path planning and path navigation nodes, as well as any post-processing for 3D map building.

- **Messages**
  - nav_msgs/NavSatFix
  - sensor_msgs/Imu

- nav_msgs/Odom

- **Topics**

- /piksi/imu
- /os1_cloud_node/imu
- /odometry/fused
- /odom
- /motor/odom

## 5. Simulation

This section details the simulation aspect of the PACMAN vehicle. Simulation of PACMAN is critical for testing the motor control stack outlined in the previous section, as well as the autonomous path navigating behavior. It is also worth noting that this section also discusses the robot-state-controller and the joint-state-controller for mapping relative positional sensor data to a global positional basis. This description of the simulation setup and procedure references SDF and URDF format extensively; for more information please refer to the specifications in references 2.10 and 2.11 respectively.

### 5.1. SDF File

The SDF[2.10] model (contained within the git repository[2.13]) was used to describe the geometry, physics, joints, links, and actuators of the vehicle presented for simulation. There are five links in total that make up the PACMAN simulation model described within the SDF file. Three out of the five used collada mesh files to describe their geometry and texture; each were created and UV wrapped in Blender[2.14]. There are two visible links that have collision physics around their mesh file: the right wheel link, the left wheel link. The chassis link is the third visible link but has no collision physics (therefore the simulation can focus on the physics described by just the right and left wheel links). It is worth noting that the outer four wheels and caster joints physics are ignored as they are not actuated and prohibited the actuated left and right wheels to move forward realistically in many instances. The right and left wheel link are connected via joints to the chassis (seen in Figure 5.1). There are additionally two invisible ball joints (with collisions enabled) that prohibited the chassis to rotating around the left and right wheel joints. The right and left wheels of the simulated model are attached to actuators using the differential drive plugin (see Gazebo plugins at reference 2.12).

**Wheel Link (Left or Right)**

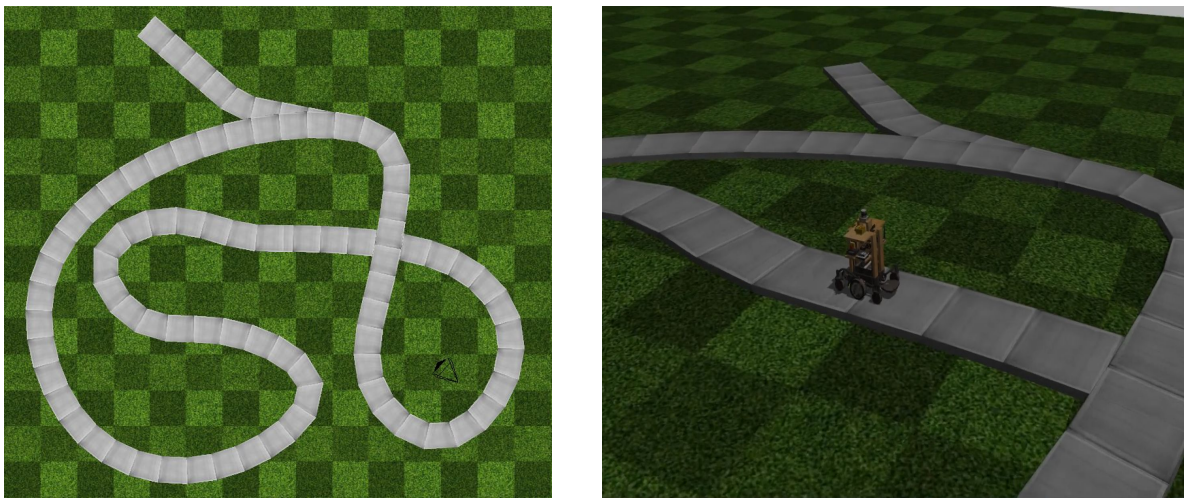
**Chassis Link**

**Assembled Links**

Figure 5.1. *Visible Links for SDF model*

## 5.2. Gazebo Simulation

Using the SDF file described in the previous section, a Gazebo Simulation (see reference 2.12) can be carried out. Figure 5.2 shows a simulated path for the pacman vehicle to follow. This path was created with common obstacles the vehicle may encounter when running realistically. Figure 5.2 also shows the PACMAN simulated model following this sidewalk path using pre-generated sequence of points along the path that are fed into the path navigation node (see section 4.6).



**Custom Sidewalk Path** **PACMAN Simulation Following Path**

Figure 5.2. *Sidewalk Path for running PACMAN in simulation (left) and PACMAN following the sidewalk path within the simulation*

### 5.3. URDF File

The URDF[2.11] model (contained within the git repository[2.13]) was used as a description that specificies all sensor positions relative to the base of the vehicle. This file has everything the SDF file describes, however it is used primarily for configuring relative positions of one link to another. This description additionally has a unique static link (connected to the “base_link”) for the LiDAR sensor, GPS sensor, camera, all four casters, and all four caster wheels; the chassis link described within the SDF file is renamed to “base_link” in the URDF so the robot-state-publisher can properly compute transforms. Moreover, an artificial invisible static link “base_footprint” is created to specify the position of the ground right below the “base_link”. The robot-state-controller and joint-state-controller are API ROS nodes that parse the URDF description for the pacman model and continuously generates map/transforms for each link and joint based on the positions specified within the URDF file through a “/tf” topic. The joint-state-controller additionally publishers joint data and subscribes to command messages to control each joint.

**Datasheets & Manuals**


1. RoboteQ MDC2230 Motor Controller User Manual
2. Jetson AGX Xavier User Manual
3. Cradlepoint BR600 User Manual
4. Piksi Duro GPS User Manual
5. Logitech C920 User Manual
6. Step-Down Voltage 24/12&5 V MH50T2405-12
7. Optical Kit Encoder Data Sheet
8. Ouster OS1-16 LiDAR User Manual
9. DUB-H4 USB 2.0 Hub
10. DAP-1522 4-Port Ethernet Switch