3D ROAD SURFACE MESHING
WITH LIDAR

by

Nathan Obert

A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Science

Committee:

_____	Dr. Zoran Duric, Thesis Director
_____	Dr. Jyh-Ming Lien, Committee Member
_____	Dr. Duminda Wijesekera, Committee Member
_____	Dr. Sanjeev Setia, Chairman, Department of Computer Science
_____	Dr. Kenneth S. Ball, Dean, Volgenau School of Engineering
Date: _____	Spring Semester 2018 George Mason University Fairfax, VA

3d Road Surface Meshing with LIDAR

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Nathan Obert
Bachelor of Science
George Mason University, 2015

Director: Dr. Zoran Duric, Associate Professor
Department of Computer Science

Spring Semester 2018
George Mason University
Fairfax, VA

Copyright © 2018 by Nathan Obert
All Rights Reserved

Dedication

I dedicate this thesis to my wife Kimberly Obert for putting up with all these years of college.

Acknowledgments

I would like to thank the Computer Science professors at George Mason University, and my employer Armedia LLC who made this possible.

Table of Contents

List of Tables

Table Page

List of Figures

Abstract

3D ROAD SURFACE MESHING WITH LIDAR

Nathan Obert

George Mason University, 2018

Thesis Director: Dr. Zoran Duric

The primary goal of this research is to create 3d maps of George Mason Fairfax roads. These maps will be used in the 3d driving simulator. Towards this goal I have decided to use Google Cartographer, which uses Lidar and Inertial Measurement Unit (IMU) data. The Cartographer has been developed and applied for slam inside buildings. In the original application Lidar and IMU units were worn in a backpack. Because of that Cartographer is adjusted for abundance of vertical features and relatively slow moving speeds.

I have collected Lidar, IMU, HD video, and GPS data for the major GMU campus roads and several parking structures and lots over four days. I have used two different Lidars (VLP-16 and VLP-64) mounted on two different vehicles. I have written the needed drivers to integrate GPS data collection with VLP-16 data. I have fabricated mounts for VLP-16 and reconfigured VLP-64 mounts for these collections. I have processed the data using Cartographer in Robot Operating System (ROS). The collected data was archived on GMU servers for future teaching and research uses.

In our application the Lidar and units were placed on a moving vehicle which travels at several times the speed of a walking person. In addition, the vertical features were more

struct accurate point clouds for some roads and parking lots and structures, while in several cases more parameter tuning may be needed to make it work.

Chapter 1: Introduction

The primary goal of this research is to create 3d maps of George Mason University (GMU) Fairfax roads. These maps will be used in the 3d driving simulator. Towards this goal I have decided to use Google Cartographer, which uses Lidar and Inertial Measurement Unit (IMU) data. The Cartographer has been developed and applied for slam inside buildings. In the original application Lidar and IMU units were worn in a backpack. Because of that Cartographer is adjusted for abundance of vertical features and relatively slow moving speeds.

I have collected Lidar, IMU, HD video, and Global Positioning System (GPS) data for the major GMU campus roads and several parking structures and lots over four days. I have used two different Lidars (VLP-16 and VLP-64) mounted on two different vehicles. I have written the needed drivers to integrate GPS data collection with VLP-16 data. I have fabricated mounts for VLP-16 and reconfigured VLP-64 mounts for these collections. I have processed the data using Cartographer in Robot Operating System (ROS). The collected data was archived on GMU servers for future teaching and research uses.

George Mason University needs 3d road data and a method to collect 3d road data. The data will be used in researching motion planning, localization, navigation, obstacle avoidance, pedestrian detection, road sign detection, and many other challenges with autonomous cars.

To implement a method of creating 3d road data, and collect data I procured any equipment required for this project, either by reusing existing equipment or purchasing additional equipment as detailed in the following sections. I fabricated mounting equipment for the sensors, and performed software integration. Data was collected and offline slam was used to build maps. After the maps were constructed I performed additional research. Conversion between point cloud formats to aid in integration with the driving simulator.

Export and manipulation of GPS for visualizing the paths on Google Maps, and input into Graph Slam. Lastly a small amount of research was done to see if surface reconstruction could be performed on the point clouds.

1.1 Overview of Cartographer SLAM

When movement occurs two point clouds are created. One for where you were, and for where you now are. Local Slam takes the two point clouds from the Lidar and attempts to consolidate them into one single map by placing the data on a series of submaps or grid points. Since orientation and velocity might have changed between the two sensor measurements the IMU is used with rigid transformations and rotations to orient the data on a grid together. Smoothing by solving least squares is also performed to assist with connecting the data together.

Local Slam can be performed as data is collected know as online Slam, or after the data is collected which is know as offline Slam. As noted in Google's paper, offline Slam typically runs around 5 times faster than real time. If 50 minutes of data is collected, it should be processed offline in around 10 minutes. Through the research all the data was processed afterwards with offline Slam.

The challenge with slam is connecting data together that is not collected in adjacent time. For example when data is collected when you drive around a block, local slam can easily identify frame by frame how to connect the data. It is global slam that recognized the last frame connects with the first frame. Essentially any use case that involves traveling near or over an area already traveled is resolved by global slam which is also referred to as loop closure.

The map being constructed is a 3d point cloud that is pushed into the slam plane (2d) that is being traveled to create a road map of obstacles. A road map is an electric map that aids with navigation by marking where obstacles are located and how to travel from one place to another. The map is broken into submaps or grids. Local slam connected the grids together to the best of its ability however small errors occur in sensors called drift that

slow cause the grids to no longer be properly aligned for loop closure. Google Cartographer addresses global slam by solving a non-linear least squares problem using branch and bound search to attempt to connect the submaps of the data together. When global slam is ran the submaps are adjusted to connect matching features together.

1.2 Overview of Data Collection

The data is collected with all the sensors by driving around George Mason University. I used two different Lidars. I collected the data by driving around GMU Fairfax Campus. The Velodyne VLP-16, 16 laser Lidar, was mounted on my Jeep Wrangler. The Velodyne VLP-64, 64 laser Lidar, was mounted on a Golf Cart. The Golf Cart was used because GMU's Motion and Shape Computing Group already had mount fabricated for VLP-64 and Golf Cart.

Lidar is one of the main sensors used in this project. It is composed of several (16 or 64) spinning lasers that provide a 3d point cloud of the immediate surrounding area. Each laser provides one "ring" corresponding to a slice of the visual field. Figure 2.4 is an example the data that comes out of Lidar. The figure is vertically alined which is sub optimal for SLAM.

When Lidar is mounted vertically the lidar would see sides, top and bottom, but would be unable to see in front and behind. Unfortunately the sky is beyond the range of the lidar, and the bottom is occluded with the vehicle which leaves only the sides of the vehicle available. The road itself would not be visible or mapped. This is not optimal for SLAM.

When mounted horizontally the lidar would see the front, sides, and behind, but would be unable to see below and above. They sky is out of range, and below is occluded so immediately this is optimal since more regions are visible to the lidar. When 3d road mapping, there is zero need to see the sky. The actual road would be captured by what is in front and behind the lidar.

The Velodyne VLP-16 is one of the Lidar units used. It is very low resolution with only 16 lasers. The VLP-16 for example might have 6 different rings of distances for objects on

the ground plane, and 10 rings for objects on vertical surfaces. There could be multiple meters between these data points, which leads features being missed due to the sparse nature of the data. For example when near a building the lidar might see the bottom of a window frame, but if the vehicle moves even a foot or two left or right with a second pass, it may see below or above the window frame being unable to match features. Even a slight variation in timing or pathing could lead to a different set of features which would cause huge issues with recognition.

The VLP-64 is high resolution with 64 lasers that also was used. Since the 64 is much higher resolution even if timing or pathing is different there is a significantly higher probability that the same features will be detected, however possibly with a different laser since the point cloud data is four times as dense. The VLP-64 produces over 1 million vectors (points) per second.

Inertial Measurement Unit (IMU) is the other main sensor I used. The IMU provides three accurate measurements: (i)orientation— a quaternion (X,Y,Z,W)—which corresponds to the heading direction with respect to the gravity direction, (ii) angular Velocity (radians per second) around X, Y, Z provide the rotation information, and (iii) Linear Acceleration (meters per second squared) in X, Y, and Z directions.

I also used Global Position System (GPS), and High Definition (HD) Camera and synchronized them with Lidar and IMU in all data captures. These two additional sensors were used for ground truth for troubleshooting and evaluation. GPS is not used within Google Cartographer. The GPS unit provided timestamps for the Lidar.

Chapter 2: Background and Related Work

2.1 Related Work

The basis of this work is Google Cartographer[4]. In their work the authors utilized Robot Operating System (ROS). They improved on multi-resolution scan matching [5] by adding branch and bound method and made exhaustive matching practical. An earlier approach to global loop closure provided a probabilistically-motivated scan-matching algorithm which produced higher quality and more robust results at the cost of additional computation time [6]. The idea of submaps used in Cartographer can be traced back to [7]; this work introduced spatially-aware data structure that enabled the cost of a map update to be proportional to the impact of any loop closures, resulting in better average case performance than naive method. Cartographer does local slam by doing rigid transformations to place each frame of the Lidar into the same overall grid of submaps. Ceres Scan Matcher is used to smooth data as cartographer adds. Google Cartographer performs the branch-and-bound algorithmic framework to find exact solutions to loop closure [8].

ROS has been a popular tool for researching Simultaneous Localization And Mapping (SLAM). ROS was originally developed at Standford's Robotics Lab and is the ground work for many of the most referenced SLAM research. SLAM has been a popular research topic within robotics and there are many related worked. "Map-Based Precision Vehicle Localization in Urban Environments"[9] is one of earlier works followed up by "Robust Vehicle Localization in Urban Environments Using Probabilistic Maps"[10] also by the same authors. "Towards fully autonomous driving: Systems and algorithms"[11] providers further details.

I reviewed the 27 road datasets listed in "When to use what data set for your self-driving car algorithm: An overview of publicly available driving datasets"[12]. 19 of the data sets

do not contain lidar. Of the other 8 data sets reviewed may of the datasets documented problems and issues with their own data, or were in a format that we could not easily use with alot of development. In the end I decided it was easier to simply record my own data with George Mason University's equipment.

"An evaluation of 2D SLAM techniques available in Robot Operating"[13] provides a significant overview of the major SLAM techniques offered using ROS, however it does not include Google's more recent contribution to SLAM.

2.2 Cartographer

Cartographer provides Local Slam and Global Slam to ROS. The lidar makes a series of 3d point cloud frames that represent the world at each location the vehicle is at. Local Slam uses the rigid transformation given by Eq. 2.1

$$T_\xi p = \begin{pmatrix} \cos\xi_\theta & -\sin\xi_\theta \\ \sin\xi_\theta & \cos\xi_\theta \end{pmatrix} p + \begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix} = R_\xi p + t_\xi \tag{2.1}$$

where ξ_θ is angular rotation about vertical axis and t_ξ is the translation between the frames. The transformations allow the frames to be organized in world coordinates. Since sensors are not perfect additional smoothing is done to optimize the matching. Optimization is provided by Ceres Scan matching shown in Eq. 2.2.

$$\underset{\xi}{\operatorname{argmin}} \sum_{k=1}^{K} (1 - M_{smooth}(T_\xi h_k))^2 \tag{2.2}$$

Submaps are matched to a world grid presented in Fig. 2.1 Local slam is not sufficient for mapping. Global slam or loop closure resolves matching frames that are not temporally aligned. For instance if the car travels in a circle, the start and end connect, however the data that connects is not in order. The same applies to a use case of mapping the left and

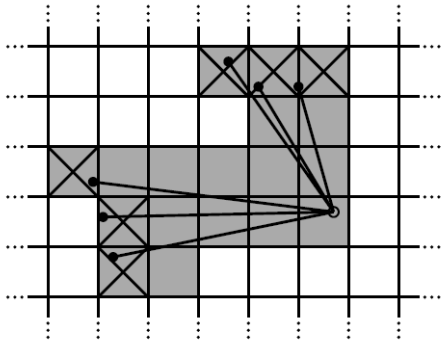


Figure 2.1: Cartographer Scan Matcher[4]

right side of the map, and then having the center connect the two sides. Global slam within cartographer is a optimized search problem. The submaps are organized in a tree, and branch and bound is used to search for submaps that connect. The search utilizes depth first search to find the leaf nodes of the tree which represent submaps as seen in Fig. 2.2

Once slam is completed, cartographer provides an utility to export map as a point cloud or raster images of the point cloud.

2.3 Platform: Hardware and Software

A Jeep Wrangler and Golf Cart was used to mount the sensors on. The Jeep Wrangler was used with the Velodyne VLP-16, and the Golf Cart was used with the Velodyne VLP-64.

The two lidar sensors used was a low resolution lidar from Velodyne the VLP-16 with 16 lasers. The second lidar was a high resolution lidar sensor also from Velodyne the VLP-64 with 64 lasers. Both Velodyne units had a Garmin GPS 18x LVC attached onto them for satellite timestamps and GPS data.

The Inertial Measurement Unit (IMU) used was by CH Robotics, model UM7. A Logitech 1080p HD USB Web Camera was used to collect ground truth information.

The computer processing hardware was a Mid 2012 Macbook Pro, with 2.7 GHz Intel

Algorithm 3 DFS branch and bound scan matcher for (BBS)

$best\_score \leftarrow score\_threshold$
Compute and memorize a score for each element in $\mathcal{C}_0$.
Initialize a stack $\mathcal{C}$ with $\mathcal{C}_0$ sorted by score, the maximum score at the top.
while $\mathcal{C}$ is not empty **do**
  Pop c from the stack $\mathcal{C}$.
  if $score(c) > best\_score$ **then**
    if c is a leaf node **then**
      $match \leftarrow \xi_c$
      $best\_score \leftarrow score(c)$
    else
      Branch: Split c into nodes $\mathcal{C}_c$.
      Compute and memorize a score for each element in $\mathcal{C}_c$.
      Push $\mathcal{C}_c$ onto the stack $\mathcal{C}$, sorted by score, the maximum score last.
    end if
  end if
end while
return $best\_score$ and $match$ when set.

Figure 2.2: Cartographer Branch and Bound[4]

Core i7, 16 GB of ram, NVIDIA GeForce GT, with Solid state internal hard drive. Cartographer is memory intense when building point clouds. A large amount of memory and solid state for swap file is optimal. Additionally I had 4 TB of external storage for the large sensor data files.

The software used was "Ubuntu 16.04.3 LTS (Xenial Xerus)", with "Robot Operating System (ROS) Kinetic Kame". Make sure the combination of software used matches Google's website, when this research was done Luner Logger (the newer version of ROS) was not supported.

For clarity: Python 2.7.12, GCC 5.4.0, Google Cartographer 0.2.0, Google ROS Cartographer Integration 0.2.0, Velodyne ROS Integration 1.2.0, ROS USB_CAM package 0.3.5, and Meshlab 1.3.2[14] were used in this research.

2.4 Overview of Hardware and Software Integration

When selecting hardware sensors do the most amount of research and verification that the hardware and software have drivers that are fully compatible to reduce the amount of development required. The Velodyne for example has drivers that feed directly into ROS. The GPS unit however supplied with the Velodyne does not have drivers. Out of the box the only benefit of the GPS is providing accurate timestamps on the Velodyne packets. I had to write our own driver to read the GPS on the Velodyne and feed it into ROS. The python source code to the driver is in the appendix.

Robot Operating System which is the core of this thesis, provides a listing on line of which sensors are compatible with it. I highly recommend you first download the drivers and attempt to compile them prior to purchasing any hardware. Many of the drivers depends on old antiquated libraries, and many not function with this specific version of ROS. Also verify the drivers provide all the necessary information within the sensor topics. Not all the open source drivers are equally developed to provide what is required.

CH Robotics UM-7 was selected for the IMU because it would properly compile, was low cost, the ROS integration was commercially supported by the manufacturer and provided all the sensor data. Not all the sensors integrations provide all these features.

2.5 Background of Datasets

The initial datasets used were provided by Google[1]. The indoor museum dataset was used largely to verify all the software was properly installed and functional. Map GMU[2] and KITTI[3] datasets were existing outdoor datasets that we used, both with some shortcomings documented in experiments section. The last three datasets I collected at George

Mason University.

2.5.1 Cartographer Backpack Deutsches Museum [1]

Google[1] supplied sample indoor data for 2D Slam collected with two VLP-16 (Horizontal and Vertical) and IMU from a backpack mounted GPS. This dataset worked flawlessly with Google's software that was already tuned to use this data. The horizontal VLP-16 provided the data for optimal matching since it was on the same plan as the map. The vertical VLP-16 provided the additional point cloud information for accurate reconstruction of the room that was outside of the view frame of the horizontal LIDAR.

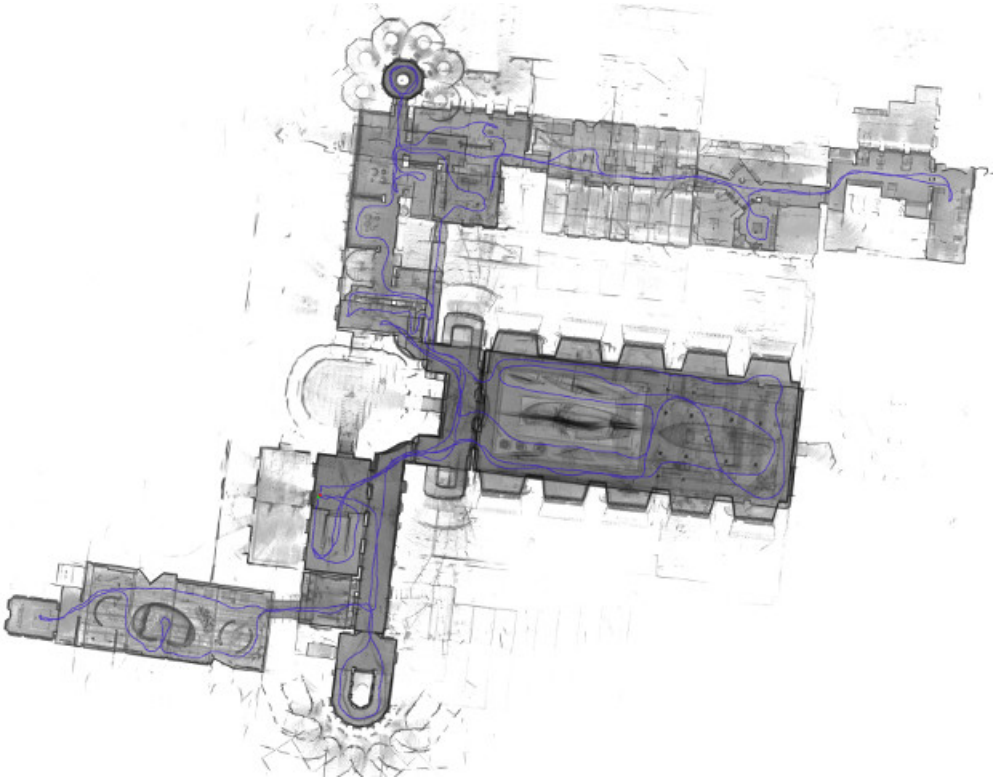


Figure 2.3: Google Sample Data[1]

Google[1] supplied sample indoor data for 3D SLAM collected with two VLP-16 (Horizontal and Vertical) and IMU from a backpack mounted GPS. This dataset also worked

flawlessly with Google's software was already tuned to use this data.

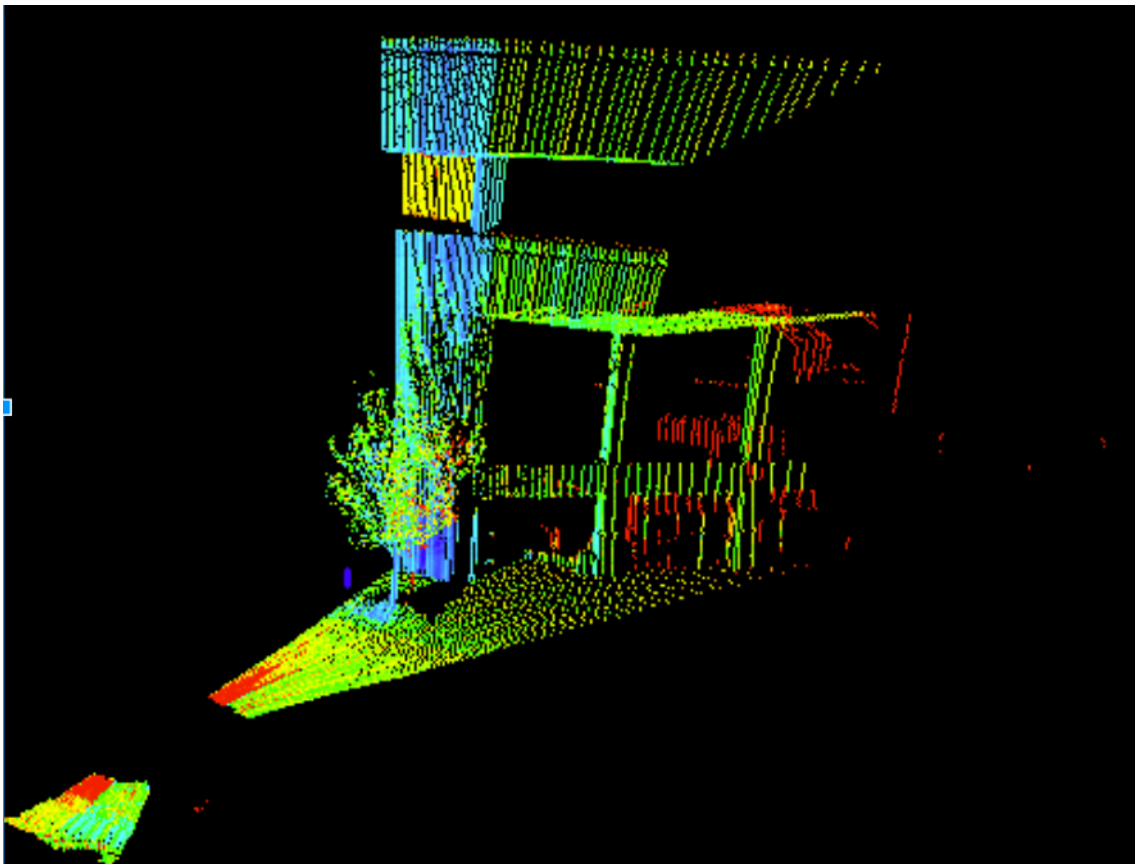


Figure 2.4: Example of no road data in MapGMU[2]

2.5.2 Map GMU [2]

The Map GMU Dataset was collected with the same Velodyne VLP-64 hardware, however a previous semester by a different student with different research. The research involved mapping buildings on campus, and the Velodyne was vertically aligned to capture the top of buildings. Additionally the Map GMU research team fitted a golf cart with a mounting bracket for the Velodyne that I used to collect data.

I had access to previous data[2] and software collected by former students at George Mason University. The previous data was collected with Velodyne VLP-64. The only

data usable was pcap lidar itself. The GPS collected was not synchronized or usable, no IMU data, or ground truth such as camera data. The data presented numerous challenges. The first challenge is the Velodyne was oriented to scan vertically instead of horizontally. This is sub-optimal for 3d road mapping as stated in other papers, however optimal for their previous thesis of mapping buildings. To clarify, it is more optimal to match an existing horizontal plane to another horizontal plan of data. The most extreme sub-optimal design would be to attempt to match a horizontal plan (the road map you have) to a new vertical plan of data. The very small intersection of the two planes would be the only area the matcher could use to attempt to connect the frames together. With the VLP-64 for example only the lasers pointed the ground that intersected with the ground plane could be used. In contrast to the hundreds of thousands of points that could be used when aligned horizontally as each laser sweeps across the entire plane.

The previously collected Velodyne data also has significant noise in it. The front of the golf cart was in the view frame of the lidar. This problem was compounded by the splash noise reflected off the white surface of the golf cart. To correct this, we removed the quadrant of the view frame with all the noise, however by doing so, it eliminated 99 percent of the actual road we were attempting to map, leaving only a view looking directly to the sides. Since the goal of the project is to map the horizontal surface of the ground plane, and not the vertical components, having the scanner oriented to only capture components that are not from the road, and vertical components was not feasible.

I did however try to load the data through Cartographer using correlative scan matching without IMU. Unfortunately cartographer attempts match horizontal segments together without the IMU, which was not optimal since the lidar was mounted vertical and most of the area it would have matched had to to be removed due to reflective noise.

New data was collected using Robot Operating System (ROS). Lidar and IMU specifically was captured for Cartographer. Additionally GPS, and HD Video was captured for ground truth. All the data was captured in a ROS datafile called a "BAG" file that contains all the datastreams at topics that are synchronized, timestamped and can be replayed. The

data can easily be accessed through a python API or the ROS applications provided. The python API allows the data to easily be exported into any format with a very small amount of code. I wrote the code necessary to export the GPS out of the bag format for example to perform GPS Smoothing and export the paths to Google Maps for visualization.

2.5.3 The KITTI Dataset [3]

The KITTI Dataset is a well known dataset on the internet. The key feature of the KITTI dataset is the data has been manually labeled. Every pedestrian, street sign, car, or other item within their datasets have been labeled which makes the dataset very popular for computer vision. Their dataset also includes IMU and lidar which makes it feasible to run through cartographer to see what kind of results can be produced.

2.6 Background of Data Conversion

Most of the data conversion was done for producing ground truth visualization or assisting with comparing to other methods outside the scope of this thesis.

Data conversion from proprietary library formats to PLY format, to OBJ format, from point cloud to mesh textures. Normals and poisson mesh reconstruction[15] was used verify feature work could use my data to build surfaces from point clouds. GPS Data was converted from $GPRMC to Degrees Minutes Seconds (DMS), and Degrees format for Google Maps visualization.

Chapter 3: Data Collection

Data Collection at GMU starts of mounting the sensors on the Jeep or Golf Cart, using a fabricated mount. Once the sensors are mounted, they need to be calibrated. The lidar must be aligned with the IMU, and the gyroscope on the IMU should be zeroed out to a flat surface the vehicle is parked on. GPS and HD Camera as also mounted, however they are only used for ground truth with this research. Once the sensors are setup and calibrated, data can be recorded by driving around GMU. Afterwards the data will go through offline slam to produce a point cloud of the area driven. Additionally an utility I wrote can be ran to export the GPS to Google maps to visualize where the data was collected.

3.1 Calibration

With all our data collection calibration took a substantial amount of time and in some cases was not a precise as desired. I used a 1 yard construction level on the ground to attempt to locate a flat surface. I next zeroed out the gyroscope on the IMU.

I discovered most roads are crowned and even parking lots are typically intentionally tilted for rain water to drain off. This provides challenges for finding level ground to calibrate the IMU. Additionally even after calibration when reviewing data, surfaces that might appear flat in HD video will most likely have a slight tilt due to the way roads that are constructed. With the Golf Cart I discovered even most side walks have some type of tilt. Fortunately ROS uses a URDF file to define the sensor orientations allowing for tuning the pitch, yaw, and roll after recording the data should the IMU not be perfectly level at calibration.

I experienced numerous problems while collecting data from either interference, cables unhooking or even sensors slightly shifting. Its very important to first do a dry run with a

small subset of the data collected before wasting hours collecting data that cannot be used. Additionally as many checks as possible that things are working should be done along the way, to spot check and correct any changes that might have occurred while collecting data, such was wiring coming unhooked, or sensors not responding.

As seen in the Fig. 3.1, I used a level during calibration to insure when I zeroed out the gyroscopes they were in the most level spot in a parking lot I could find.

Figure 3.1: Jeep Wranger with Velodyne Bumper Mounted being calibrated

One issue I encountered was our IMU would sometimes start reporting rotation while I was not moving. The easiest way to detect this problem was to monitor the orientation being reported in ROS using RVIZ when data collection starts and I was not moving. If the orientation is moving while the vehicle is not moving obviously the sensor is malfunctioning. The

easiest correction I determined was unplug the IMU and plug it back in and start a new data collection. The root cause to why the IMU sensor would start "spinning", and why unplugging it would reset it was never isolated.

Figure 3.2: Jeep Wranger with Velodyne Roof Mounted

3.2 Global Position System Smoothing

I experimented with GPS Smoothing, since I am using a consumer grade Garmin GPS unit. I developed a program that can export the GPS out of the ROS Bag format into Degrees, Degrees Minutes Seconds (DMS), and NMEA 0183 $GPRMC formats. Additionally the program can do a full GPS export (dense) or a sparse export of only changes in GPS. Once

the data is exported in CSV format, I can run it through MatLab or any external tool to clean up any data points I feel need to be corrected. The program developed can then be utilized to inject the new GPS stream into the ROS Bag file either replacing the original topic stream or as a new topic stream. The data will be sequenced in the same order, unless temporal smoothing was done to the spreadsheet, in which case, the GPS positions will be inserted at their new respective positions.

This smoothing will allow us to correct any errors in the GPS I identify to insure with Graph SLAM the data is constrained to the correct position on the map.

3.3 Global Position System Analysis

I also used our GPS program to transfer the GPS data into Google Maps. This allowed us to visually inspect the GPS. Review of the data showed the GPS data was slightly bias to the right, which is explainable by driving on the right hand side of the road and having the GPS mounted on the right hand side of the vehicle.

Although Google Cartographer does not use GPS, this bias can easily be compensated for within ROS by insuring the frame_id within the Uniform Robot Description Format (URDF) file in ROS has the proper translation and rotations on all the sensors. Examples of the data exported to Google Maps can be seen within prior figures.

I detected the GPS Data from the Velodyne/GPS hardware was substantially more dense than I expected. After careful review I determined the Velodyne/Garmin hardware was returning 12 duplicate GPS entries. I altered the code to sub sample the data to only return data when the GPS changed.

3.4 Data Collection 240 degrees, VLP-16

I collected data with the VLP-16 lidar mounted on the bumper of a Jeep Wrangler. I collected 3d Lidar, IMU, GPS, and 1080p video. All the data was collected on the same computer, the GPS was hard wired to the lidar for ultra precise timing. All the data was

recorded into ROS Bag format allowing it to be synchronized and easily replayed. Depending on which inputs are used, the other inputs would be used as ground truth. For example Cartographer used IMU and lidar, leaving the GPS and camera data as ground truth. The datasets are displayed on Fig. 3.5 through Fig. 3.8. Specifically the region around the hub, police station, and Sandy Creek parking deck were collected with the 240 degrees mount.

Figure 3.3: GPS Shenandoah Parking Deck

3.5 Data Collection 360 degrees, VLP-16

I constructed a small tower on the jeep, to elevate the lidar's view above the jeep. I recorded 360 degree lidar at George Mason with the same equipment as collecting 240 degree data, see Fig. 3.2. The datasets are displayed on Fig. 3.5 through Fig. 3.8. All the locations except the sandy creek parking deck were collected with the 360 degrees mount.

Figure 3.4: GPS West Campus Parking Lot

3.6 Data Collection 360 degrees, VLP-64

I used a Golf Cart(Fig. 3.9) with a horizontally mounted VLP-64 Lidar to also collect data. See Fig. 3.10. The datasets collected are the following:

- parkingbuilding.bag - GMU parking office
- pond.bag - Road around Mason pond
- police.bag - Road around Mason police Station
- police2.bag - Road around Mason police Station
- hub2.bag - Road near "The Hub" as Mason
- patriotcircle.bag - Patriot Circle Dr.
- parkinggarage.bag - first floor of parking garage
- ikes.bag - Near "Ikes" restaurant
- sandycreek.bag - Sand Creek Lane around parking structure

All of the datasets are road mesh data requiring loop closure to complete.

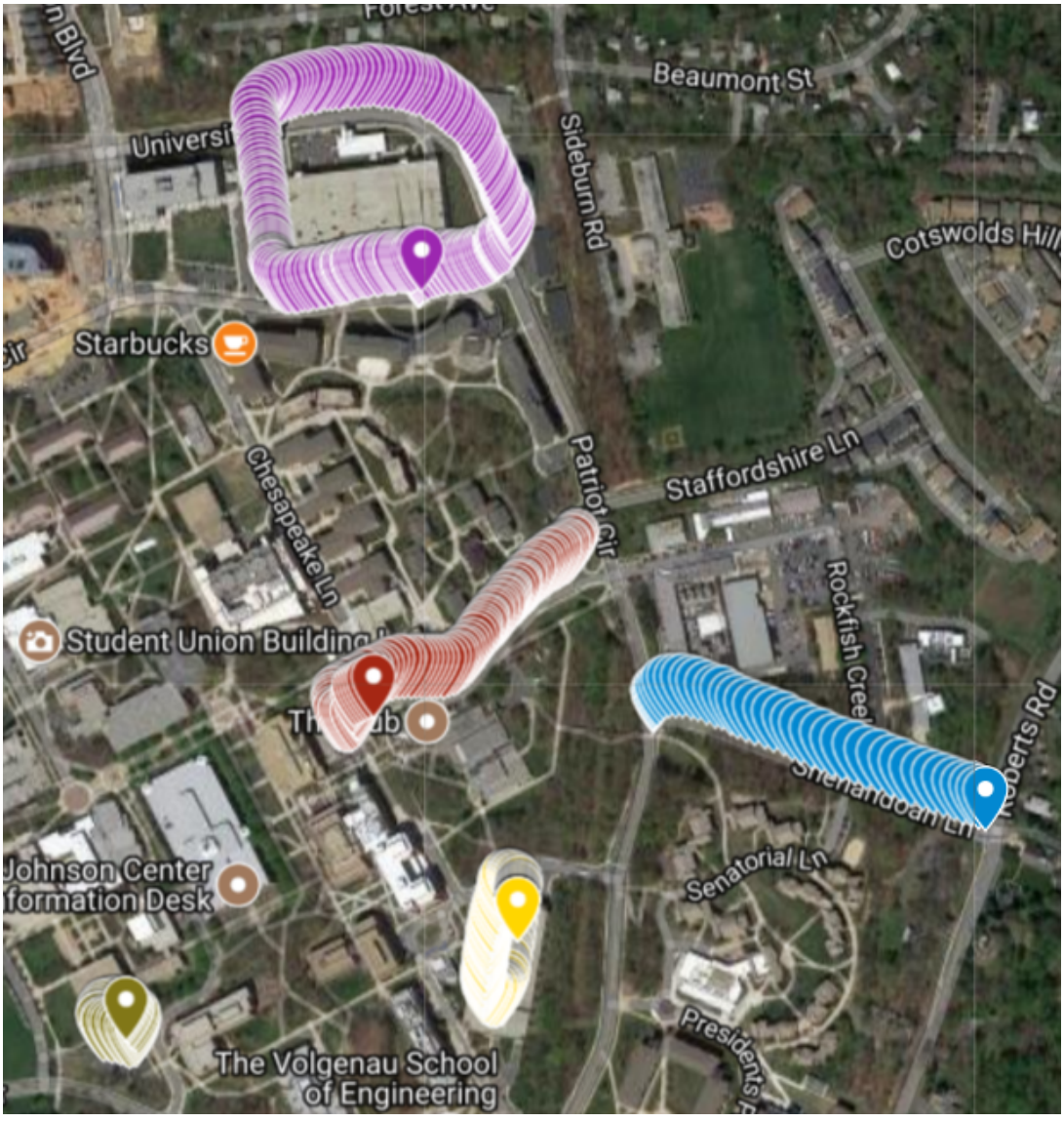


Figure 3.5: Data Collected with Velodyne VLP-16 1

Figure 3.6: Data Collected with Velodyne VLP-16 2

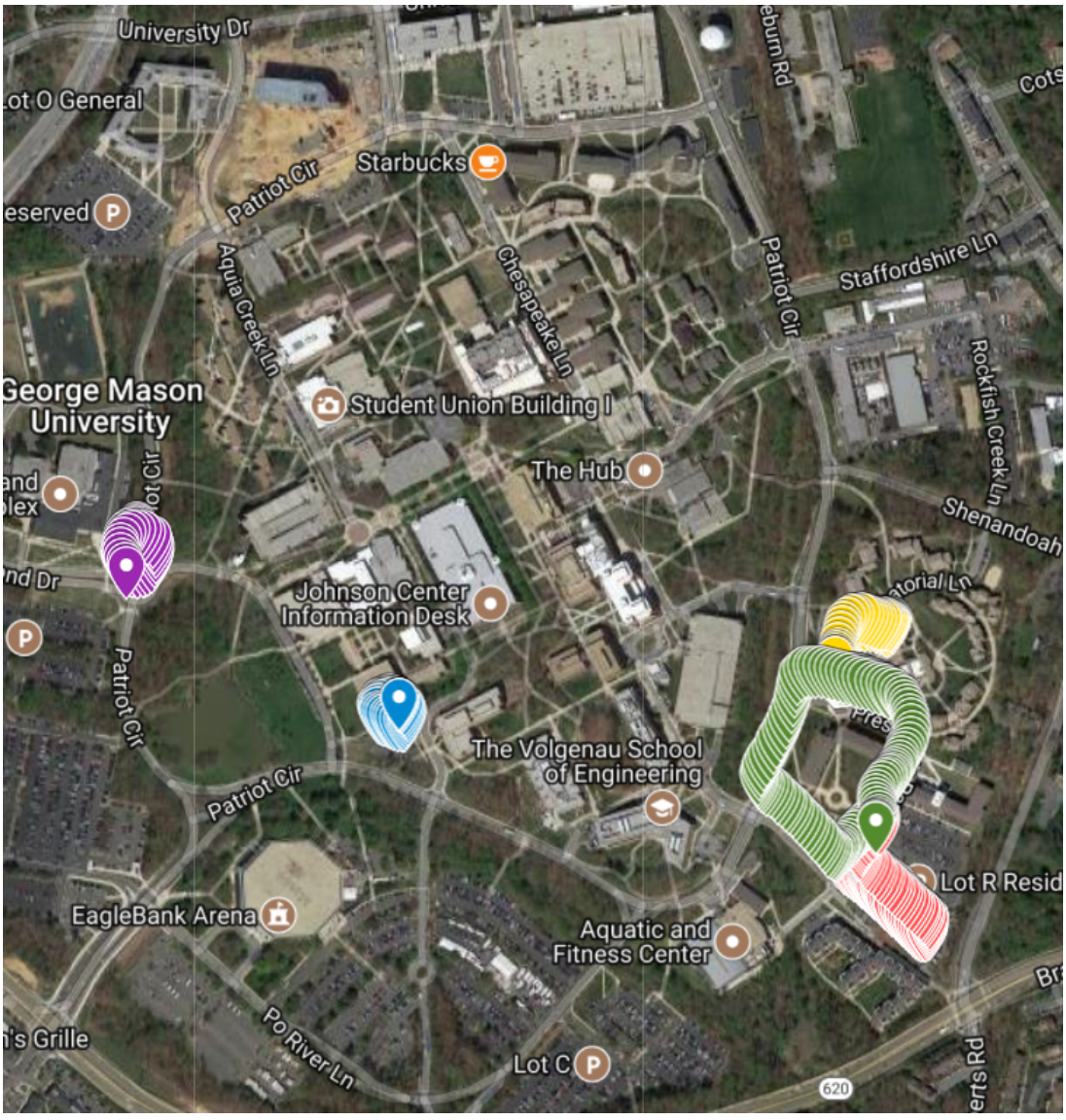


Figure 3.7: Data Collected with Velodyne VLP-16 3

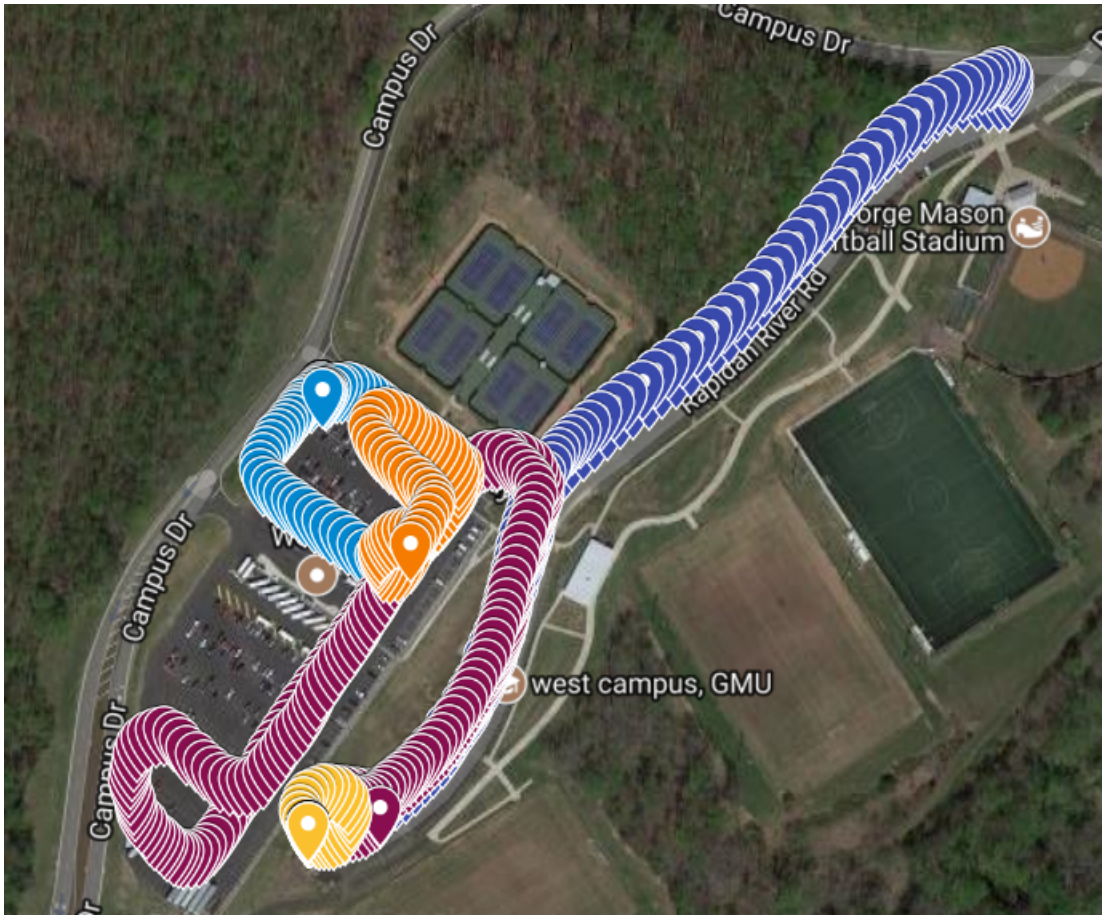


Figure 3.8: Data Collected with Velodyne VLP-16 4

Figure 3.9: VLP-64 mounted on Golf Cart

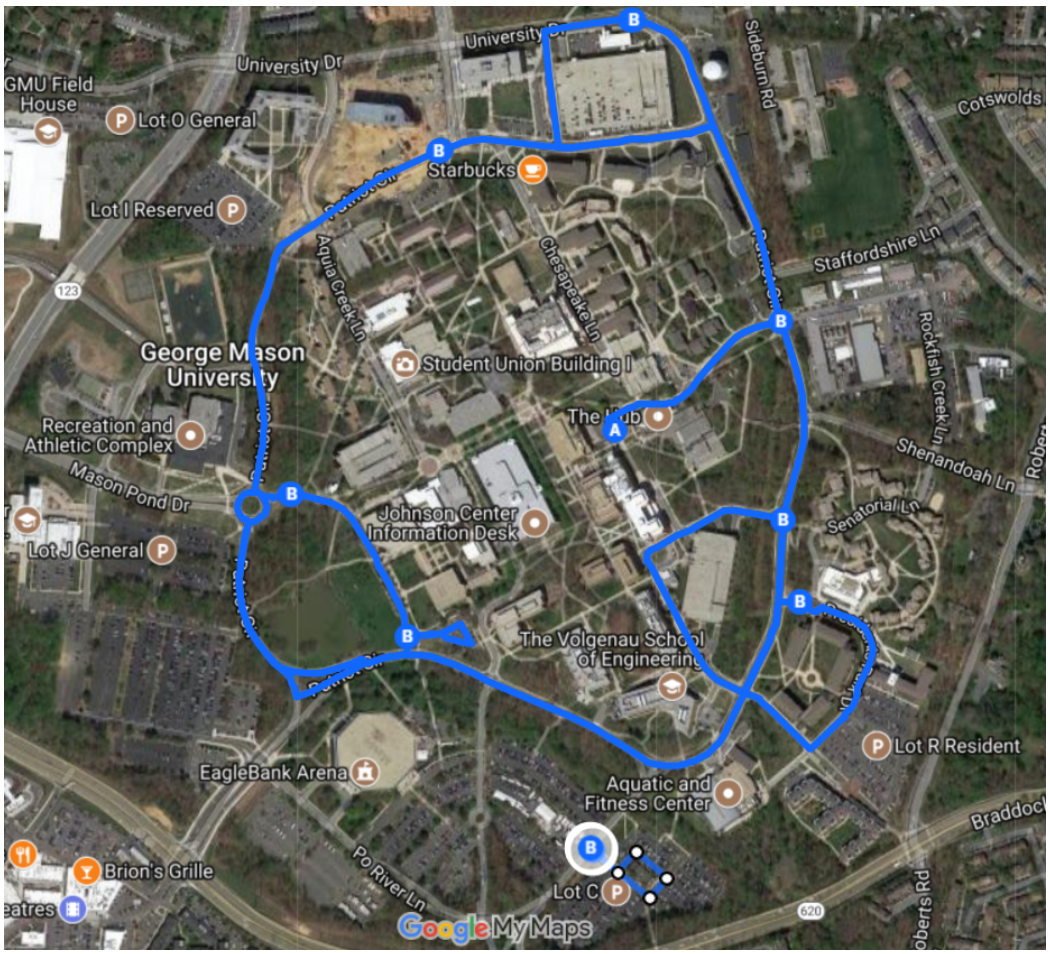


Figure 3.10: Data Collected with Velodyne VLP-64

Chapter 4: Experiments

Experiments were conducted on the Map GMU, KITTI, and collected data sets. Each dataset provides its own challenges. The Map GMU data failed because it was vertically aligned, without an IMU and did not contain any road data. The KITTI data only was successful with their trivial data. The data I collected was the most successful of all the experiments.

4.1 Map GMU[2]

The existing lidar data from Map GMU[2] was the first outdoor dataset used. The data had the golf cart in every frame of the data and reflection noise since the surface of the golf cart is white and reflective. I modified the open source Velodyne drivers for ROS exclude this region from the data within the PCAP files previously captured. I attempted to use "Online Corrective Scan Matching" which is the mode that attempts to match the frames of data without an IMU, or with faulty IMU data. The path of the golf cart was a square, ending where it started at. The result in cartographer however was a straight line. Without the IMU, and with the vertical mounted Lidar that could only see the sides, cartographer could not match enough features to see turns.

4.2 The KITTI Dataset[3]

The small simple datasets from KITI[3] processed with Google Cartographer successfully. In some cases however the dataset was so large I could not export the point cloud data out of Cartographer.

The large complex datasets from KITTI[3] did not have as good results with Google Cartographer. Specifically I could not get loop closure. Portions of the maps were accurate,

while other portions were very poor. I inquired with the Google authors for feedback on why the well known KITTI[3] dataset had problems with Google Cartographer. The first issue they pointed out was the dataset did not start at a complete stop. Since the IMU returns acceleration the starting speed must be zero. The second issue they pointed out is it appears the sensors were down sampled. Google expects very fast IMU times where the software knows exactly where the lidar is as the data comes in and sweeps across the plane. Down sampling to a complete sweep with one imu data point vastly reduces the capability of their algorithm.

Trivial KITTI[3] data like a car moving straight down a road for 10 seconds processed and converted to point cloud very well. Very complex datasets requiring global slam however performed substantially worse as displayed in the figure.

Figure 4.1: KITTI Dataset[3]

4.3 SLAM Tuning

The default parameters were not sufficient for the data processing and the parameters were updated. See the appendix for the exact settings used.

IMU drift was identified as a problem after processing some of the problematic datasets from the VLP-16 and VLP-64 data collections. To re-mediate IMU drift I fine tuned the IMU translation/rotation matrix within the Uniform Robot Description Format (URDF) with -0.1125 radian pitch.

I experimented with the accumulator to determine what differences there were with how much data was accumulated in batches going into SLAM. Our results indicate that with the VLP-16 three or four accumulations (full spins) provided optimal results with our dataset. Fewer spins, down to even one spin provided slightly lesser results, while going much beyond four spins drastically decreased the results.

I experimented with ceres scan matcher, this allows computer vision and matching to have a stronger weight in the cost formulas within slam than the IMU. The ceres scan matcher is typically very helpful with high resolution lidar with poor IMU. Since the VLP-16 is low resolution, the ceres scan matcher significantly decreased the results. Our conclusion was that with low resolution lidar, the IMU is necessary for constraining the data.

4.4 Data Collected roof mounted Velodyne VLP-16 (240 degrees)

Using Google Cartographer with VLP-16 I had good results with local trajectory builder, and loop closure, however the low resolution of VLP-16 against high resolution outdoor environment provided poor results for scan matching and global trajectory builder. This allowed in limited cases in optimal conditions to collect small loop closures of outside data, like Shenandoah Parking Deck, and Rivanna River Way.

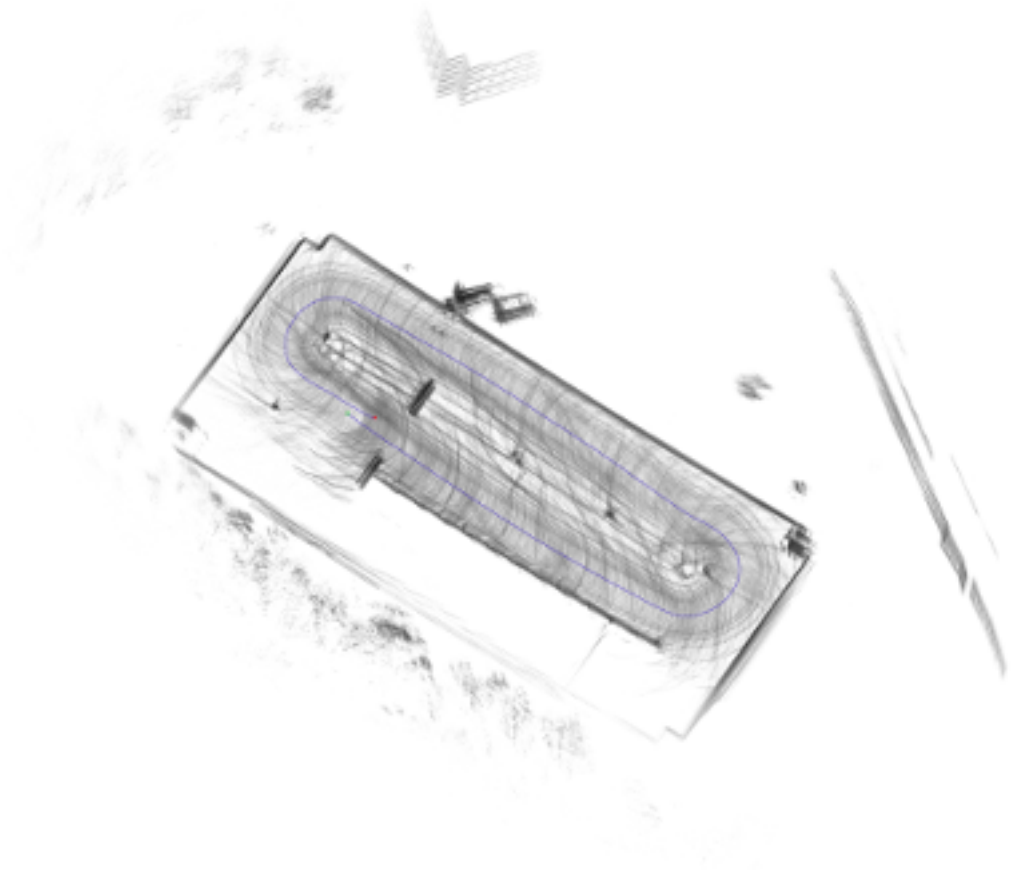


Figure 4.2: Point Cloud Shenandoah Parking Deck

4.5 Data Collected roof mounted Velodyne VLP-16 (360 degrees)

I moved the lidar from the bumper to the roof on a small tower, to increase the view frame from 240 degrees to 360 degrees. I found it much easier for local trajectory builder to perform loop closure, however I still had the same problem with poor results on scan matching and global trajectory builder in some locations. Our results indicate maximizing the area on the plan being matched is the key to good matching results with SLAM. Obviously a large view plan from 240 to 360 degrees adds a significant increase in incoming data that can be matched to the existing maps horizontal plane.

West Campus Parking Lot, and Parking Building in Lot C are examples of loop closure with the 360 degree dataset using Google Cartographer.

4.6 Data Collected from Golf Cart with Velodyne VLP-64 (360 degrees)

I collected data with a Golf Cart that had been already configured for the LVP-64 Lidar. Previous researchers had mounted the lidar vertically to capture buildings, I mounted it horizontally to capture roads. The vertical orientation from the previous students data collection was sub-optimal for road mapping since matching only occurred where the orthogonal data from the vertical lidar touched the horizontal plan of the road map. In contrast to our existing data collected on VLP-16, I obviously cannot increase the number of degrees, but I did vastly increase the resolution. By having 64 lasers instead of 16 lasers, there was four times the resolution, which provides four times as many features for matching. The increased resolution vastly improved the results. Some areas that I could not get slam to map correctly instantly were mapped correctly with the increased resolution.

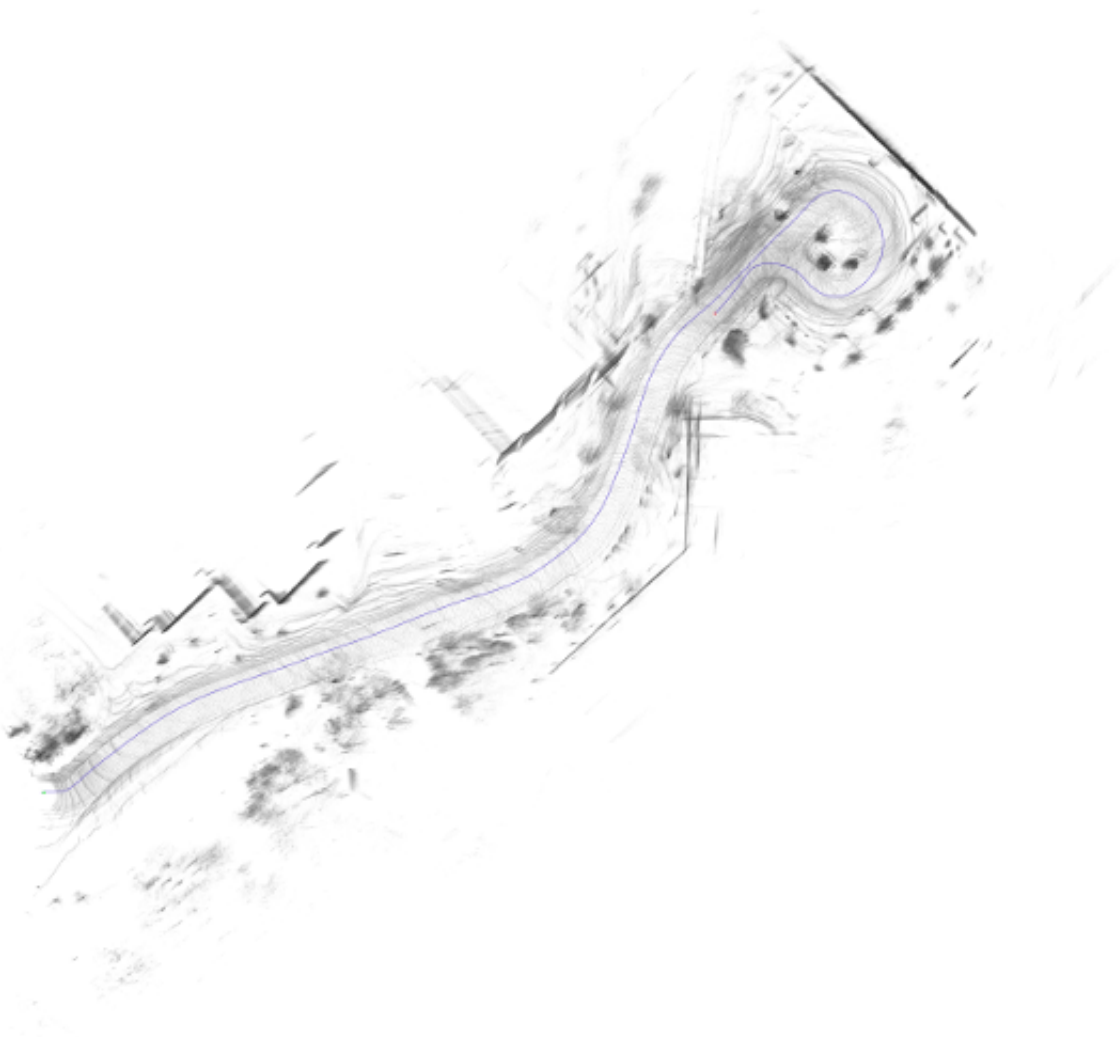


Figure 4.3: Point Cloud Rivanna River Way

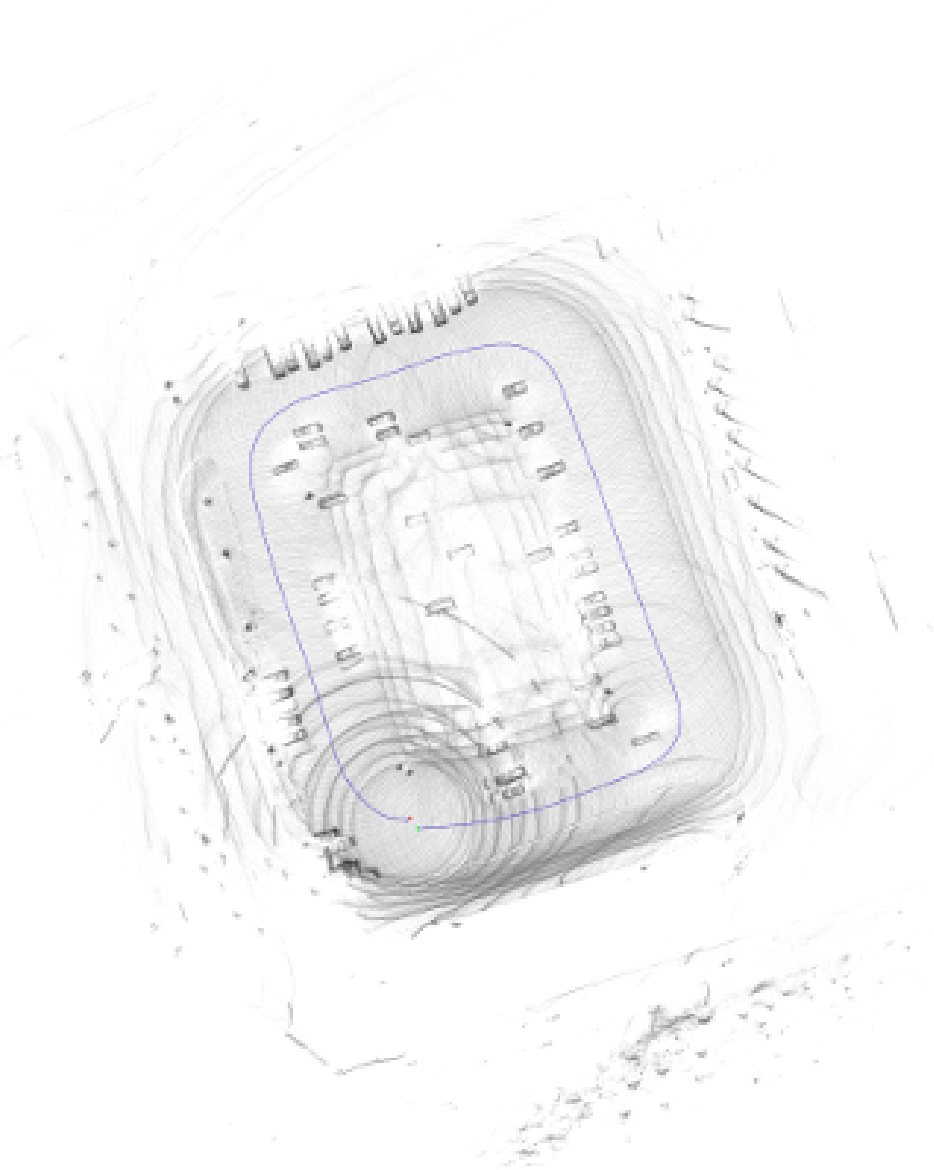


Figure 4.4: Top View Point Cloud West Campus Parking Lot

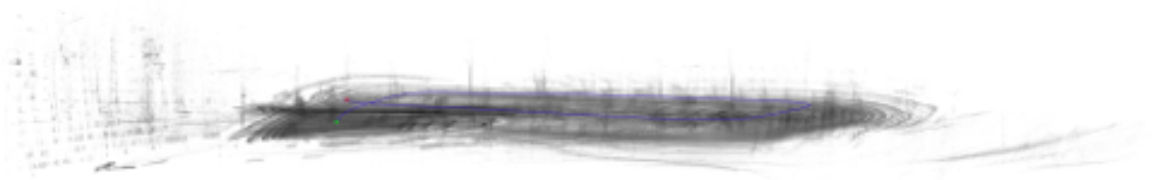


Figure 4.5: Side View Point Cloud West Campus Parking Lot

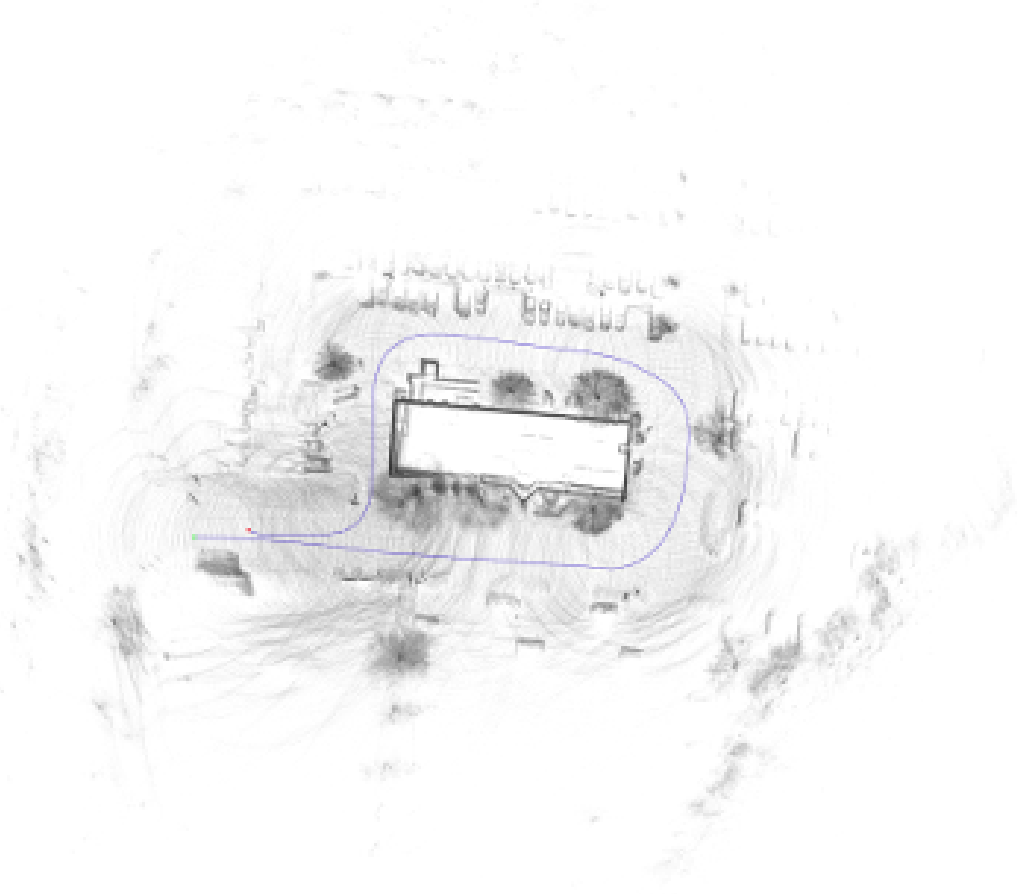


Figure 4.6: Point Cloud Parking Building in Lot C

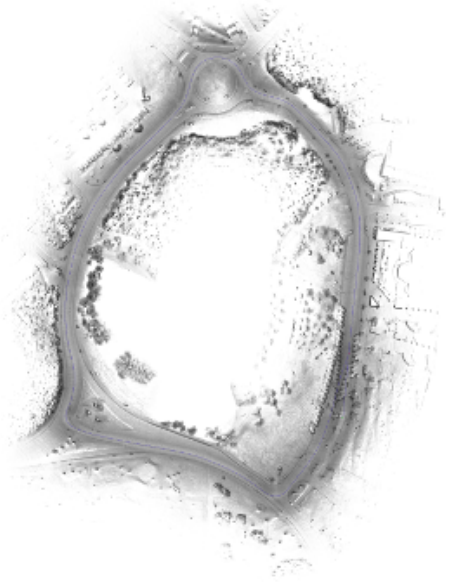
4.7 Results

With the VLP-16, The two configurations, I had overall better results with 360 degrees both for quality of the point cloud returned, and the overall number of point clouds produced. I inferred that the more points on the horizontal 'slam plane' the better Google Cartographer could perform local and global slam.

The VLP-64 however, provided better data than the VLP-16 with four times the resolution. The Mason Pond dataset will be carefully examined below using Google as ground truth. The four images supplied for comparison are a Satellite Image provided by Google, a Road Map provided by Google, the Cartographer Output from using the VLP-64 Lidar, as well as the road map superimposed on top of the scaled and rotated Cartographer output for careful examination.

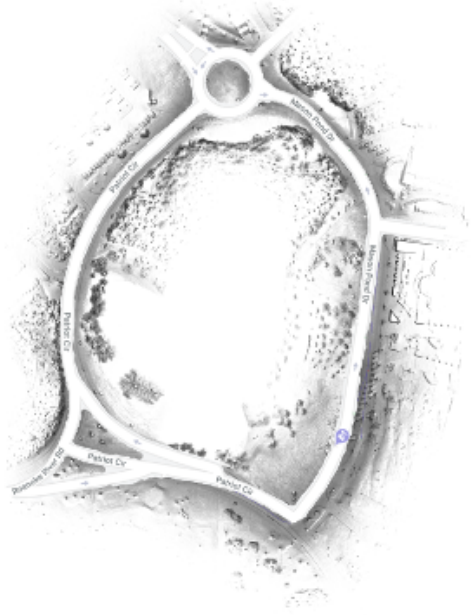
The superimposed image shows the road mesh is where it belongs on the map provided by Google.

Satellite Image from Google



Cartographer Output using VLP-64 Lidar

Road Map from Google



Cartographer Output rotated, scaled, and superimposed on Google Road Map

Figure 4.7: Results from Mason Pond

Chapter 5: Discussion

Figures 5.1, 5.2, and 5.3 display simple loop closure using the low resolution VLP-16 lidar with cartographer. The vehicle drove in a circular direction with the starting and ending in the same area on George Mason west campus. The data is rather simple with a large hill within the view frame of the lidar in nearly every frame. Although there are very few strong square corner features in this dataset it demonstrates cartographer's ability to perform local slam and loop closure even with a low resolution lidar.

Figures 5.4, 5.5, and 5.6 display loop closure with a slightly more complex dataset using the low resolution VLP-16 lidar with cartographer. The vehicle drove roughly in a square. The distinct circles seen in the lower left corner show the distance the lidar can see. In this example the loop is larger than the distance of the lidar, displaying that the algorithm functions outdoors even when landmarks are outside of its view. Its my belief this dataset worked well with the low resolution lidar because the cars in the parking lot provided a feature rich environment.

Figures 5.7 and 5.8 display loop closure with an even more complex dataset using the low resolution VLP-16. The vehicle drove around a building that was physically impossible to see around. The low resolution lidar with cartographer performed well in this environment also. Its my belief the building itself and the cars in the parking lot provided a strong set of features to aid in local and global slam being successful.

Figures 5.9, 5.10, and 5.11 display a problematic dataset with the low resolution VLP-16 lidar. This is a "round about" or "traffic circle" located on Patriot circle. This dataset does not contain many strong features for SLAM to utilize and the end result looks more like a cork screw shape rather than a circle. Its my strong suspicion if more features were present in the dataset the VLP-16 lidar with cartographer would have operated well in this setting as well.

Figures 5.12, 5.13, and 5.14 display the exact same problematic "round about" however with the data collected with the high resolution VLP-64 lidar. This "round about" or "traffic circle" from patriot circle is clearly visible. Although there are other structures present in the point cloud, the same structures were also present with the low resolution lidar. These three figures are a subset of the larger loop closure of George Mason Pond presented in the results section. As clearly visible in the images, local and global slam were successful with the higher resolution lidar when fewer features were available.

Figure 5.15 displays a closeup in meshlab of the point cloud produced of the "round about" featured in the previous images with the VLP-64 Lidar. The details of the road are very clearly present within the point cloud.

The conclusion I have draw is that as the number of strong features are reduced, the resolution of the lidar needs to increase for quality results.

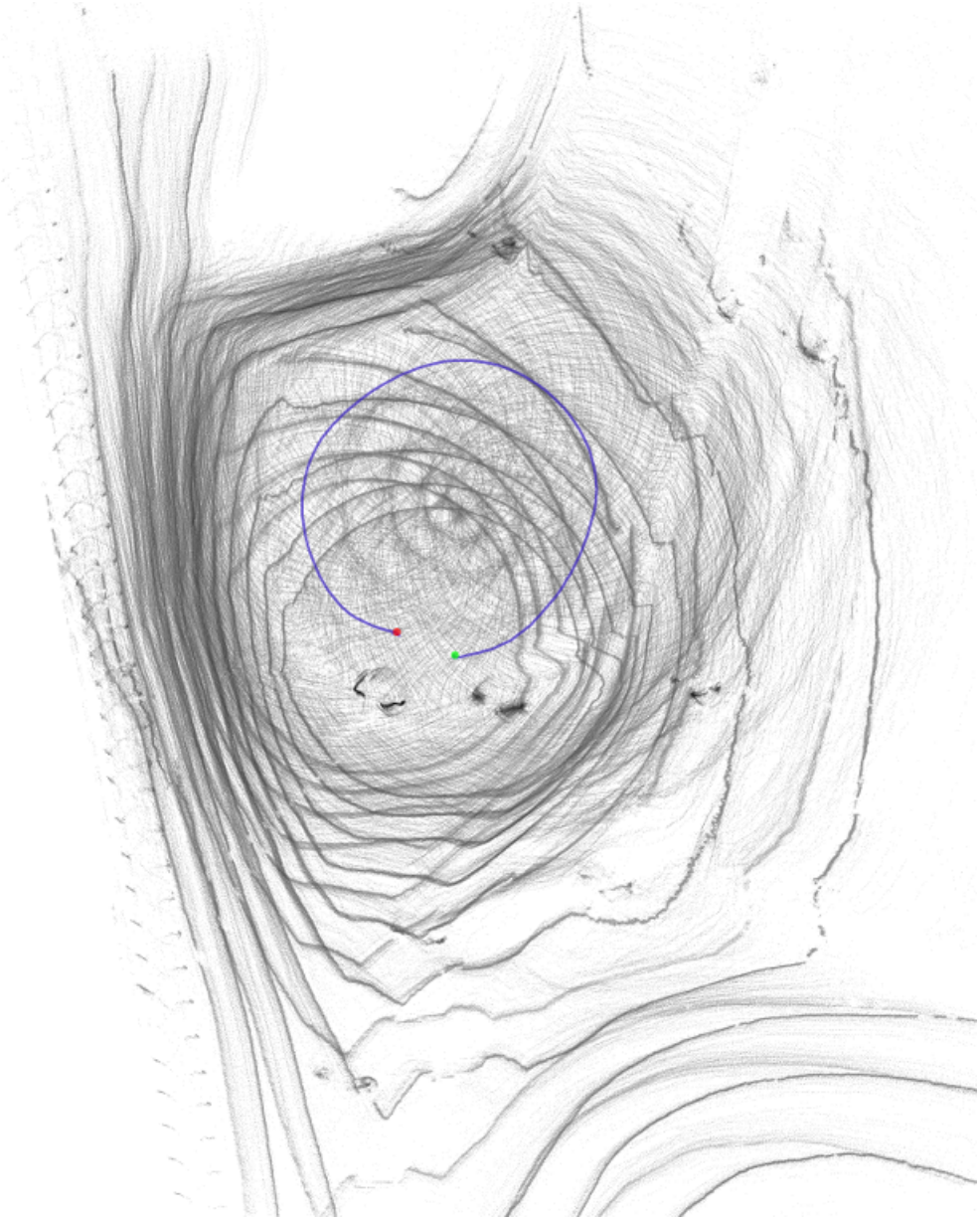


Figure 5.1: VLP-16 XY Simple Loop Closure

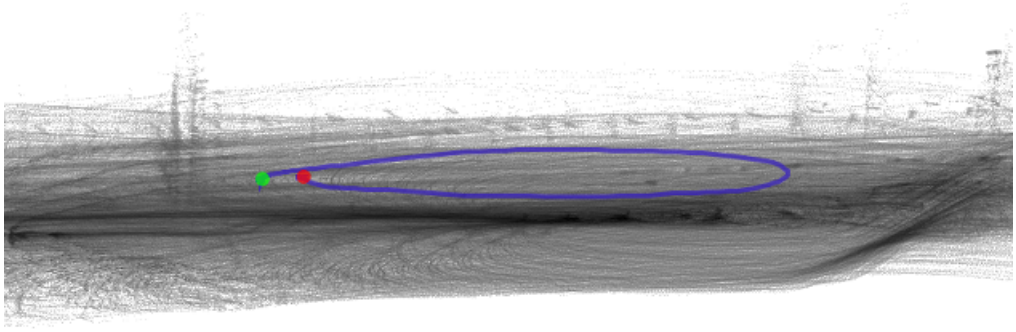


Figure 5.2: VLP-16 XZ Simple Loop Closure

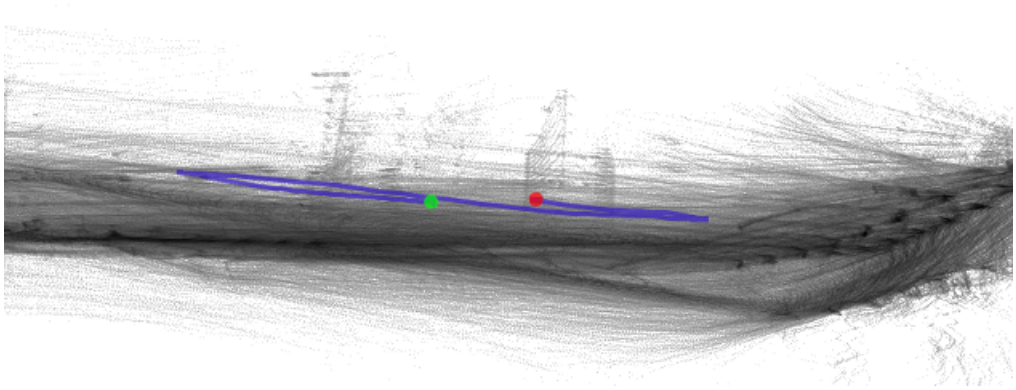


Figure 5.3: VLP-16 YZ Simple Loop Closure

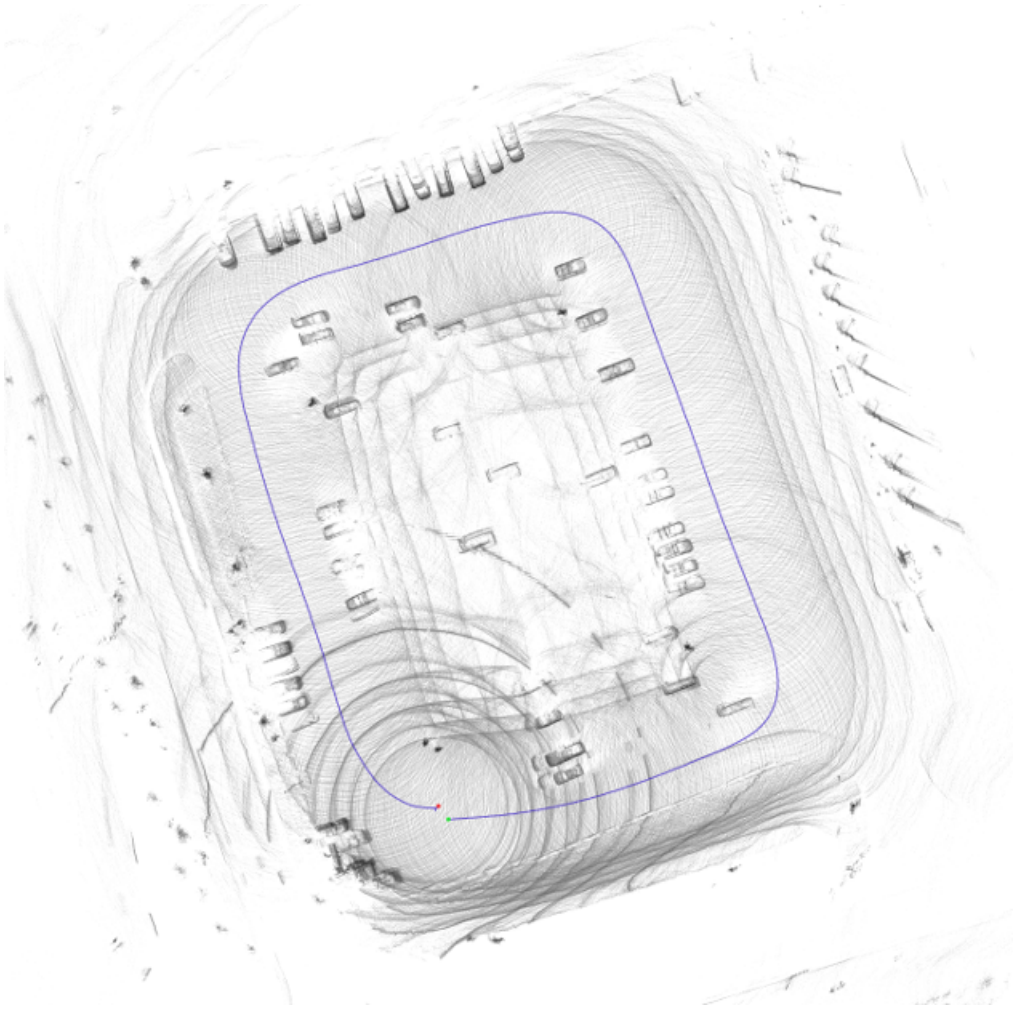


Figure 5.4: VLP-16 XY Loop Closure Around Parking Lot

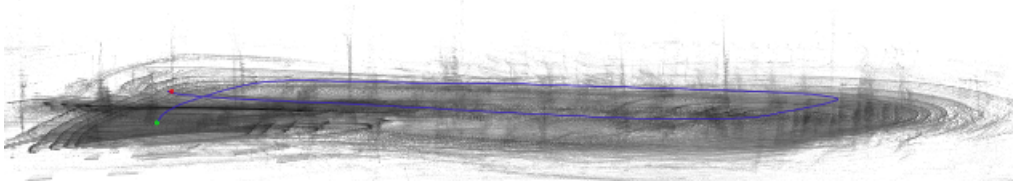


Figure 5.5: VLP-16 XZ Loop Closure Around Parking Lot

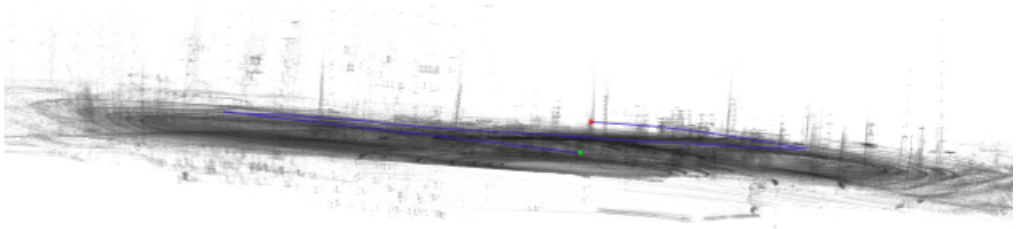


Figure 5.6: VLP-16 YZ Loop Closure Around Parking Lot

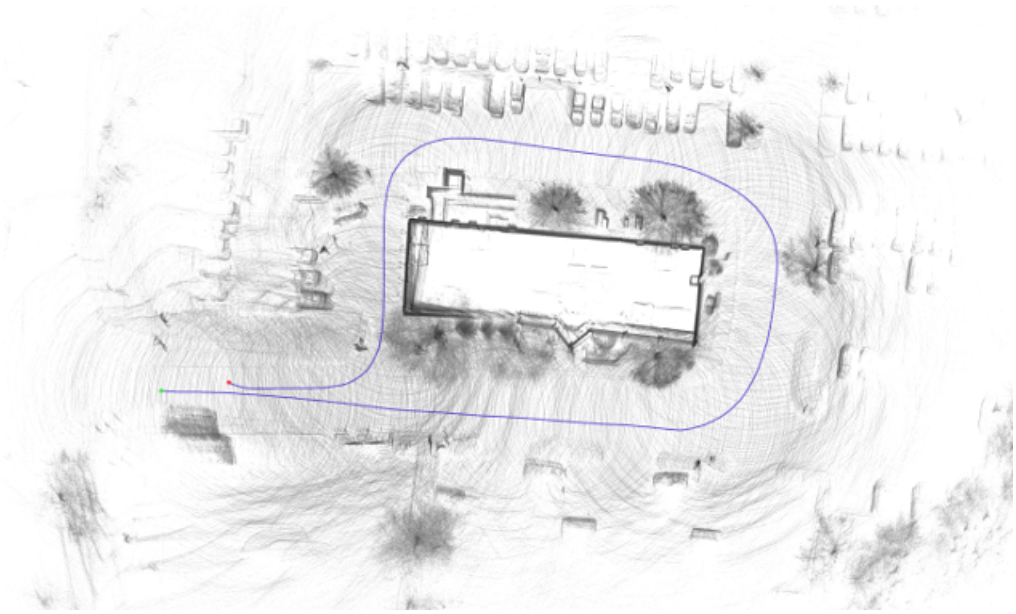


Figure 5.7: VLP-16 XY Loop Closure Around Parking Building

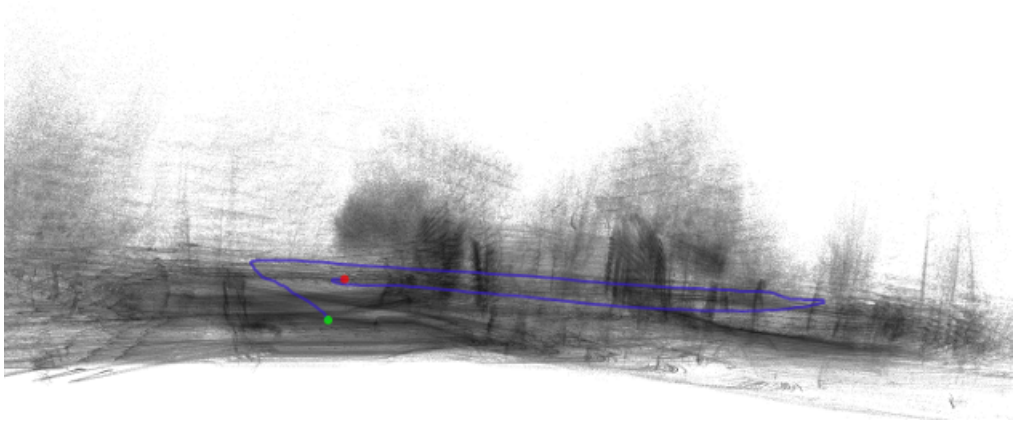


Figure 5.8: VLP-16 XZ Loop Closure Around Parking Building

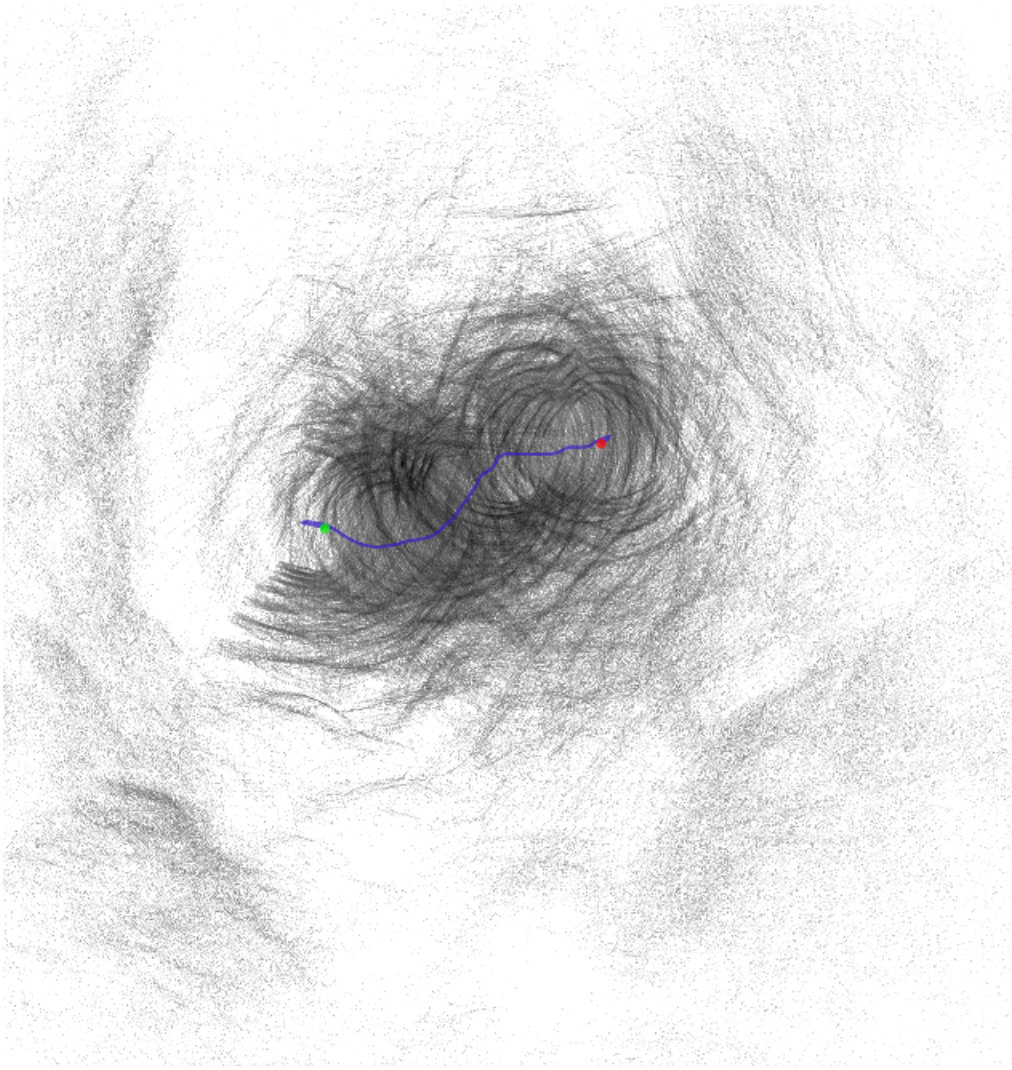


Figure 5.9: VLP-16 XY round about Mason Pond

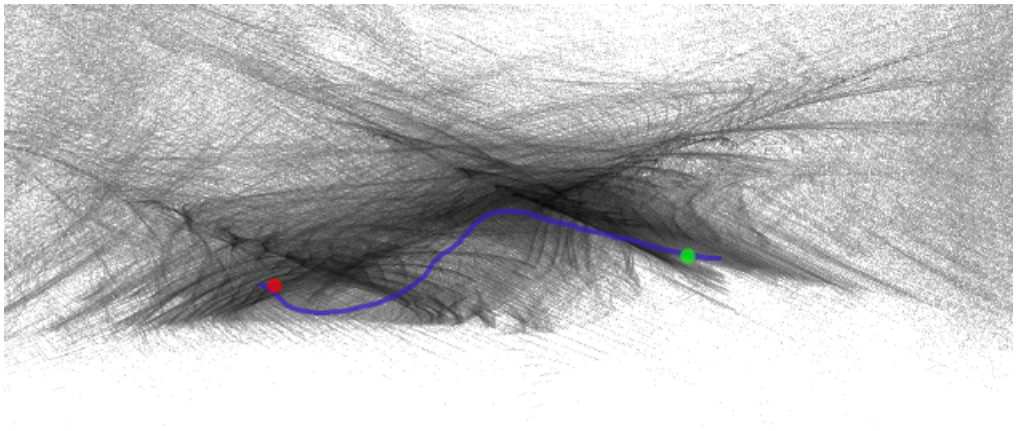


Figure 5.10: VLP-16 YZ round about Mason Pond

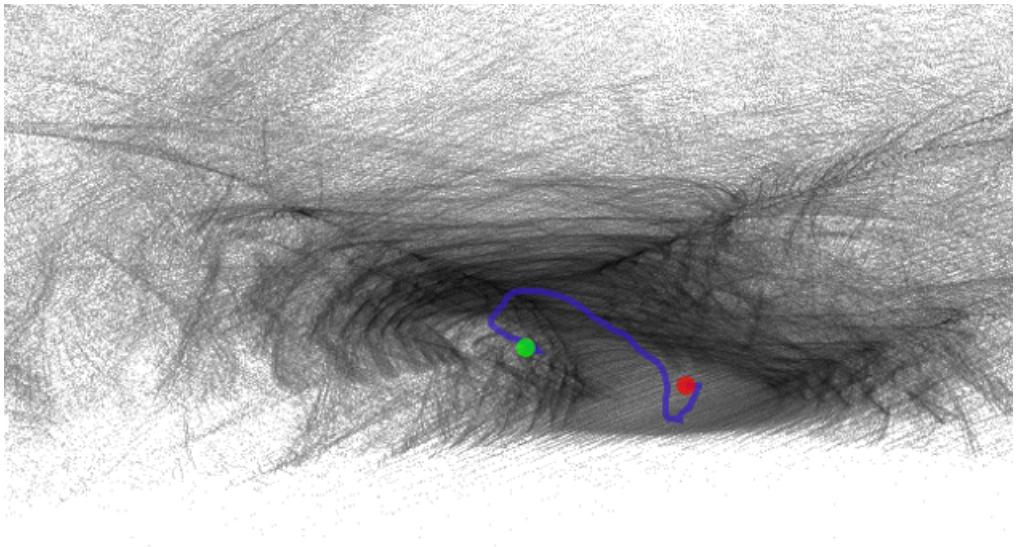


Figure 5.11: VLP-16 XZ round about Mason Pond

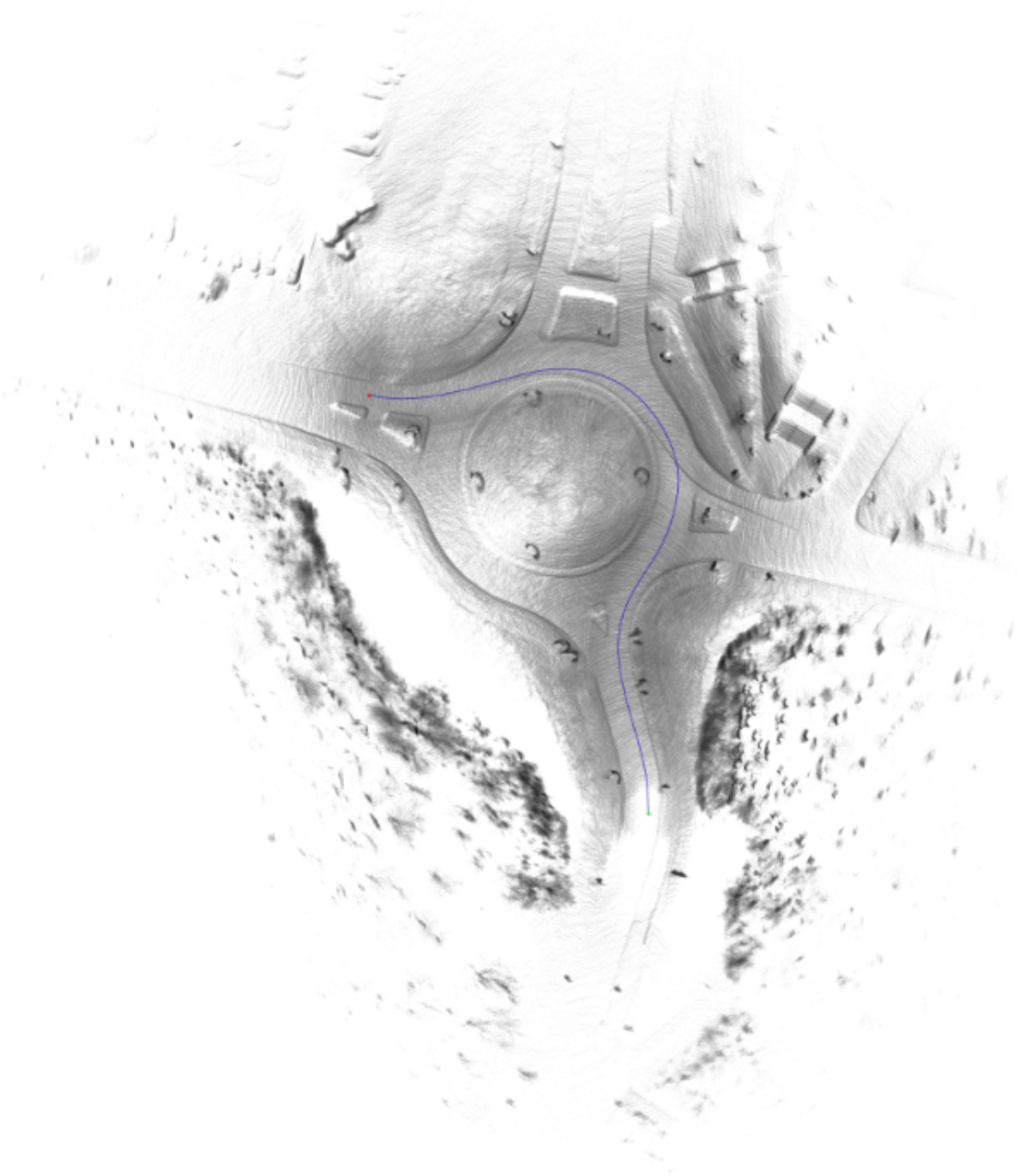


Figure 5.12: VLP-64 XY round about Mason Pond

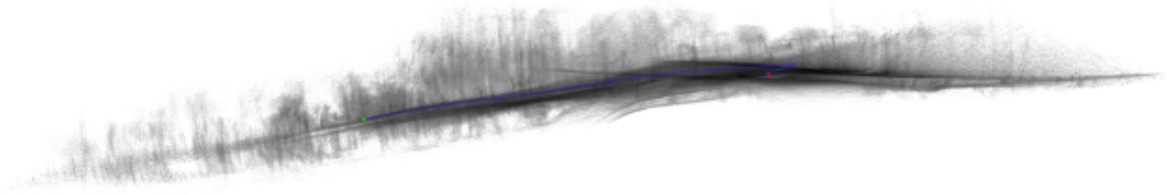


Figure 5.13: VLP-64 XZ round about Mason Pond

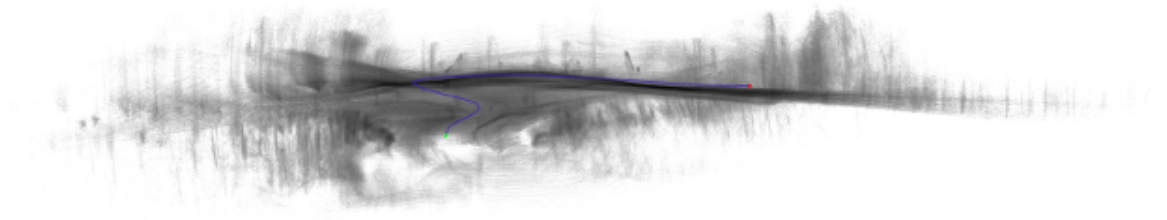


Figure 5.14: VLP-64 YZ round about Mason Pond

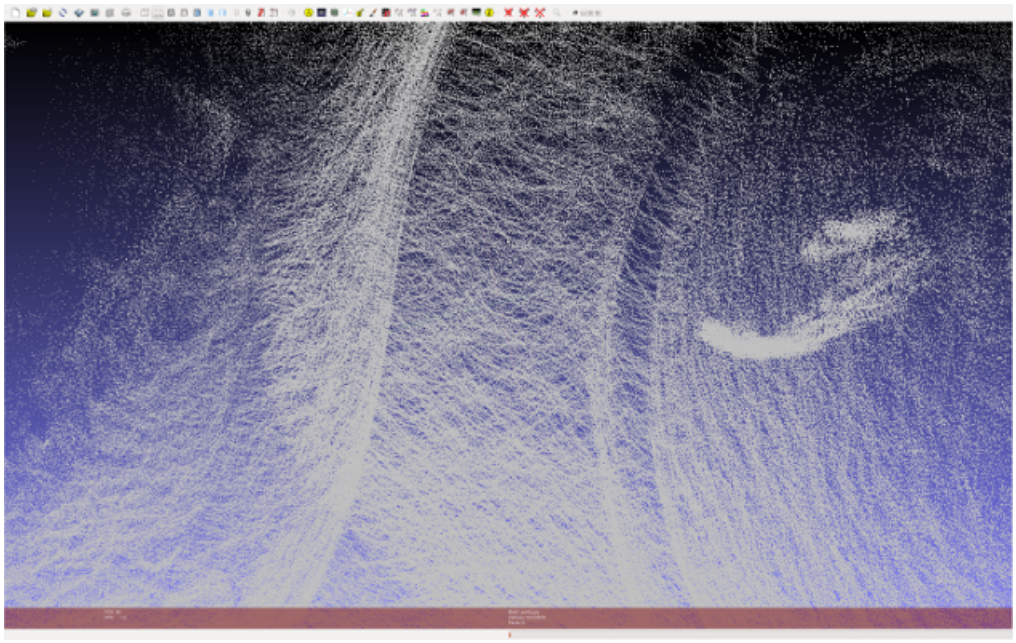


Figure 5.15: VLP-64 closeup round about Mason Pond

Chapter 6: Conclusion

Cartographer works quite well outdoors as well with optimal hardware even though it is designed for indoor SLAM where GPS is not available. With lesser hardware like the Velodyne VLP-16, cartographer has difficulties matching surfaces with poor features. The circle presented on the top of the Mason Pond data for instance is mostly grass without any solid structures or strong features around it. Cartographer had no problem with the high resolution data from the VLP-64, however we captured this area multiple times with the lower resolution VLP-16 and had very poor results. In contrast however cartographer worked very well with low resolution lidar like the VLP-16 in feature reach environments and indoor like environments like the parking garage with flat solid walls with strong ridge features.

The shortcomings of cartographer in the outdoor environment in the future might be be overcome if GPS is added, allowing problematic areas to be constrained using GPS for global slam. Essentially making cartographer a hybrid with graph slam.

Appendix A: Glossary of Terminology

- DMS: Degrees, Minutes, Seconds
- $GPRMC: GPS sentence within NMEA 0183 international standard
- GPS: Global Positioning System
- GMU: George Mason University
- HD: High Definition
- IMU: Inertial Measurement Unit
- LIDAR: Light Imaging, Detection, And Ranging
- LTS: Long Term Support
- NMEA: National Marine Electronics Association
- OBJ: Wavefront Object File Format
- PLY: Polygon File Format
- ROS: Robot Operating System
- SLAM: Simultaneous Localization And Mapping
- URDF: Uniform Robot Description Format
- USB: Universal Serial Bus

Appendix B: Installation

B.1 Ubuntu Installation

The installation of Ubuntu was fairly typical. Although most of these settings make nearly no difference in the end, I am documenting them for clarity. Insure the installation is of Xenial Xerus (16.04), and there is sufficient disk space allocated.

1. Download Updates while Installing Ubuntu
2. Install third-party software for graphics and Wi-Fi hardware, Flash, MP3, and other media
3. Select: Erase disk and install Ubuntu
4. Select: Use LVM with the new Ubuntu installation
5. New York
6. English (US)
7. English (US) - English (Macintosh)
8. Your name: user
9. computer name: cs799
10. pick a username: user
11. Select: Log in automatically

B.2 Linux Configuration

Immediately after installing Linux, some basic configuration is necessary.

Update the security file to allow for frequent easy access of sudo. emphsudo vi /etc/sudoers

```
user    ALL= NOPASSWD : ALL
```

Install SSH, Create a few directories that will be needed later, and get the system to start patching itself.

```
sudo apt-get -y install openssh-server
sudo apt-get install -y vim vim-gnome vim-gnome-py2
sudo mkdir -p /opt/crontabs
sudo chown  -R user:user /opt
touch /opt/crontabs/update.sh
chmod a+x /opt/crontabs/update.sh
vim /opt/crontabs/update.sh
```

Create a bash file for system updates.

/opt/crontabs/update.sh

```
#!/bin/bash
sudo apt-get -y update
sudo apt-get -y upgrade
sudo apt-get -y dist-upgrade
```

Update the cron so it will automatically perform updates *sudo crontab -e*

Crontab Listing:

```
0 2 * * * /opt/crontabs/update.sh > /dev/null 2>&1
```

Properly tune screen settings, using Gnome Desktop. This will allow for collecting data without the screen locking.

1. Brightness & Lock:

2. Turn screen off when inactive for: Never

3. Lock: OFF

B.3 Install Cartographer Libraries

Install the Cartographer Prerequisites, and compile the library by itself. The library would allow you the ability to write c++ code directly against the cartographer libraries. This will insure the system has what it needs to run Cartographer, and can be troubleshot by itself. Perform the following Linux commands[16]:

```
cd /opt
# Install all prerequists
sudo apt-get install -y cmake g++ git google-mock libboost-all-dev libcairo2-dev
sudo apt-get install -y libeigen3-dev libgflags-dev libgoogle-glog-dev
sudo apt-get install -y liblua5.2-dev libprotobuf-dev libsuitesparse-dev
sudo apt-get install -y ninja-build protobuf-compiler python-sphinx
# Verify Cartographer (Libraries) will compile and built without ROS
# Build and install Ceres.
git clone https://ceres-solver.googlesource.com/ceres-solver
cd ceres-solver
mkdir build
cd build
cmake .. -G Ninja
ninja
```

```
ninja test
sudo ninja install

# Build and install Cartographer.
cd /opt
git clone https://github.com/googlecartographer/cartographer
cd cartographer
mkdir build
cd build
cmake .. -G Ninja
ninja
ninja test
sudo ninja install
```

B.4 Install Robot Operating System

Installation of Robot Operating System Kinetic Kane is required prior to installing the Cartographer Integration. Below are the Linux commands I used to install ROS.[17]

Note: Make sure you use Kinetic Kane verison of ROS, do not attempt to use other versions of ROS without verifying compatibility with Cartographer documentation. Currently the newer version of ROS "Luner Logger" is not supported by Cartographer.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(
    lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.
    list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80
    --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
sudo apt-get update
sudo apt-get install -y ros-kinetic-desktop-full
```

```
sudo rosdep init
rosdep update
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source /opt/ros/kinetic/setup.bash
sudo apt-get -y install python-rosinstall python-rosinstall-
    generator python-wstool build-essential
```

B.5 Install ROS Cartographer Integration

Now that Cartographer can be compiled, and ROS is installed, installation of the integration can be done. Use the following Linux commands to build and deploy ROS/Cartgrapher Integration: [1]

```
cd /opt
# Install wstool and rosdep.
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build

# Create a new workspace in 'catkin_ws'.
mkdir catkin_ws
cd catkin_ws
wstool init src

# Merge the cartographer_ros.rosinstall file and fetch code for
    dependencies.
wstool merge -t src https://raw.githubusercontent.com/
    googlecartographer/cartographer_ros/master/cartographer_ros.
    rosinstall
wstool update -t src
```

```
14
15 # Install deb dependencies.
16 # The command 'sudo rosdep init' will print an error if you have
       already
17 # executed it since installing ROS. This error can be ignored.
18 sudo rosdep init
19 rosdep update
20 rosdep install --from-paths src --ignore-src --rosdistro=${
       ROS_DISTRO} -y
21
22 # Build and install.
23 catkin_make_isolated --install --use-ninja
24 source install_isolated/setup.bash
25 echo "source /opt/catkin_ws/install_isolated/setup.bash" >> ~/.
       bashrc
```

B.6 Velodyne Driver Installation

In order to interface directly with the hardware of the Velodyne (or read pcap files previously captured with Velodyne), it is necessary to compile and deploy Veloydne drivers. The following Linux commands will compile and deploy Velodyne drivers [18].

```
1 mkdir -p /opt/catkin_ws/src
2 cd /opt/catkin_ws/src
3 git clone https://github.com/ros-drivers/velodyne.git
4 cd ..
5 sudo rosdep init
6 rosdep update
```

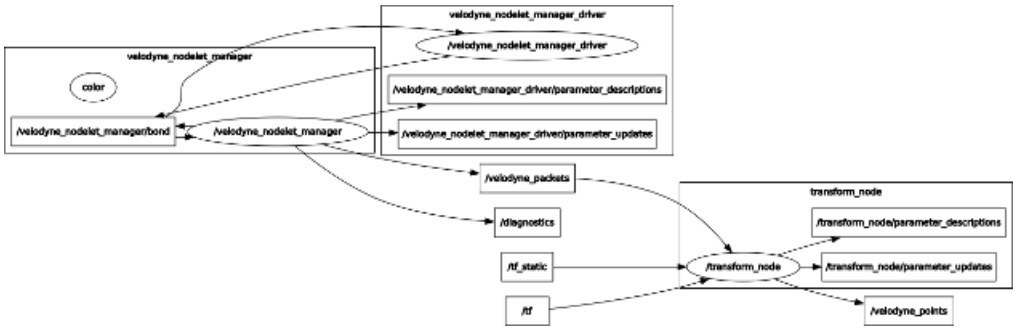


Figure B.1: Velodyne ROS Interaction

```
7  rosdep install --from-paths src --ignore-src --rosdistro=${
       ROS_DISTRO} -y
8
9  # Build and install.
10 catkin_make_isolated --install --use-ninja
11 source install_isolated/setup.bash
```

Appendix C: Code

C.1 GPS Driver for Velodyne

```
1  #!/usr/bin/env python
2  # -*- coding: iso-8859-15 -*-
3
4  import sys
5  import rospy
6  import socket
7  from std_msgs.msg import String
8  from std_msgs.msg import Header
9  from socket import *
10
11 # create ros publisher
12 rospy.init_node("telemetry_node")
13 pub = rospy.Publisher("telemetry", String, queue_size=10)
14
15 # connect to velodyne
16 UDP_IP = "192.168.1.2"
17 UDP_PORT = 8308
18 sock = socket(AF_INET, SOCK_DGRAM)
19 sock.bind((UDP_IP, UDP_PORT))
20
21 # loop forever
22 while True:
23         try:
24                 data, addr = sock.recvfrom(512)
```

```
25                      nmea = data[204:278]
26                      #print nmea
27                      pub.publish(String(nmea))
28              except:
29                      print "Exception: Reconnecting..."
30                      sock.close()
31                      sock.bind((UDP_IP, UDP_PORT))
32
33  # we dont need to close port since we loop forever!
```

C.2 GPS Export to Google Maps and GPS Smoothing

```
1   #!/usr/bin/python
2   #
3
4   import sys
5   import rospy
6   import rosbag
7   from std_msgs.msg import Int32, String
8
9   ##########################################################################
10  # How to Use.
11  #
12  # Step #1, choose format you wish you use GPS:
13  #
14  # DMS: Degrees, Minutes, Seconds
15  # Degrees: Degrees.decimal    [preferred by google]
```

```
16 # $GRPRMC: Long standardized gps string from hardware with
       additional information
17 #
18 # Set True to whichever formats you desire (you can pick multiple
       )
19 #
20 # options to export csv
21 writeToCsvDMS = False
22 writeToCsvDegrees = False
23 writeToCsvGPRMC = False
24 spareGps = False
25
26 # Note, sparseGPS will only provide you changes in GPS, instead
       of every gps entry in file
27
28 # Step #2, run the program
29 #
30 # ./rewritebag.py <provide bagfilename>
31 #
32 # example:  rewritebag.py in1.bag
33 #
34 ## depending on options these files will appear ######
35 #
36 # -rw-rw-r-- 1 nathan nathan   2329 Jan 31 22:25 in1.bag-DMS-
       sparse.csv
37 # -rw-rw-r-- 1 nathan nathan 320231 Jan 31 22:24 in1.bag-DMS-
       dense.csv
```

```
38 # -rw-rw-r-- 1 nathan nathan    2490 Jan 31 22:25 in1.bag-degrees-
       sparse.csv
39 # -rw-rw-r-- 1 nathan nathan 342634 Jan 31 22:24 in1.bag-degrees-
       dense.csv
40 # -rw-rw-r-- 1 nathan nathan    4134 Jan 31 22:24 in1.bag-GPRMC-
       sparse.csv
41 # -rw-rw-r-- 1 nathan nathan 568372 Jan 31 22:24 in1.bag-GPRMC-
       dense.csv
42 #
43
44 # Step #3, perform any smoothing desired to the csv file in the
       format you desire
45 #
46 #
47
48 # Step #4, create a new bag file with updated "smoothed" gps
49 #
50 # set all three options below most likely to True
51
52 # options to write bag file
53 readInput = False
54 # readInput tells the program to read in the CSV file you are
       providing
55
56 writeIgnoreSourceGPS = False
57 # writeIgnoreSourceGPS tells program to discard gps from original
        bag file
58
```

```
59 writeToBag = False
60 # writeToBag tells program to write a new bagfile named <original
       name>-updated.bag
61
62 # then run the program
63 #
64 # ./rewritebag.py <provide original bag filename> <provide one of
       above csv files>
65 #
66 #
67 # note, in the csv file:
68 # column #1 is the timestamp,
69 # column #2 is the name of the topic
70 # column #3 is the gps
71
72 # the program will use whatever topic name you choose, so if you
       want original gps and new gps
73 # most likely you will want to change the topic name in the
       spreadsheet unless you want the
74 # gps mixed
75
76 # additionally the new topic will have the data in the exact
       string format from the spreadsheet
77 ##################################################
78 # nathan@www:~/IdeaProjects/cs699-gps$ rosbag info test-west-
       parking.bag
79 # path:        test-west-parking.bag
80 # version:     2.0
```

```
81 # duration:      5:04s (304s)
82 # start:         Jan 20 2018 19:08:08.58 (1516475288.58)
83 # end:           Jan 20 2018 19:13:12.91 (1516475592.91)
84 # size:          10.1 GB
85 # messages:      56783
86 # compression:   none [4819/4819 chunks]
87 # types:         sensor_msgs/Image        [060021388200
       f6f0f447d0fcd9c64743]
88 #                sensor_msgs/Imu          [6
       a62c6daae103f4ff57a132d6f95cec2]
89 #                sensor_msgs/PointCloud2  [1158
       d486dd51d683ce2f1be655c3c181]
90 #                std_msgs/String          [992
       ce8a1687cec8c8bd883ec73ca41d1]
91 # topics:        /imu/data                5779 msgs    : sensor_msgs/
       Imu
92 #                /telemetry              41933 msgs    : std_msgs/
       String
93 #                /usb_cam/image_raw       3039 msgs    : sensor_msgs/
       Image
94 #                /velodyne_points         6032 msgs    : sensor_msgs/
       PointCloud2
95 #
96 ################################################
97 # http://aprs.gids.nl/nmea/#rmc
98 # $GPRMC
99 #
100 # Recommended minimum specific GPS/Transit data
```

```
#
# eg1. $GPRMC,081836,A,3751.65,S,14507.36,E
    ,000.0,360.0,130998,011.3,E*62
# eg2. $GPRMC,225446,A,4916.45,N,12311.12,W
    ,000.5,054.7,191194,020.3,E*68
#
#
# 225446        Time of fix 22:54:46 UTC
# A             Navigation receiver warning A = OK, V = warning
# 4916.45,N     Latitude 49 deg. 16.45 min North
# 12311.12,W    Longitude 123 deg. 11.12 min West
# 000.5         Speed over ground, Knots
# 054.7         Course Made Good, True
# 191194        Date of fix  19 November 1994
# 020.3,E       Magnetic variation 20.3 deg East
# *68           mandatory checksum
#
#
# eg3. $GPRMC,220516,A,5133.82,N,00042.24,W
    ,173.8,231.8,130694,004.2,W*70
# 1    2    3    4    5     6    7    8      9     10  11 12
#
#
# 1    220516       Time Stamp
# 2    A            validity - A-ok, V-invalid
# 3    5133.82      current Latitude
# 4    N            North/South
# 5    00042.24     current Longitude
```

```
126 # 6    W            East/West
127 # 7    173.8        Speed in knots
128 # 8    231.8        True course
129 # 9    130694       Date Stamp
130 # 10   004.2        Variation
131 # 11   W            East/West
132 # 12   *70          checksum
133 #
134 #
135 # eg4. $GPRMC,hhmmss.ss,A,llll.ll,a,yyyyy.yy,a,x.x,x.x,ddmmyy,x.x
        ,a*hh
136 # 1     = UTC of position fix
137 # 2     = Data status (V=navigation receiver warning)
138 # 3     = Latitude of fix
139 # 4     = N or S
140 # 5     = Longitude of fix
141 # 6     = E or W
142 # 7     = Speed over ground in knots
143 # 8     = Track made good in degrees True
144 # 9     = UT date
145 # 10    = Magnetic variation degrees (Easterly var. subtracts from
          true course)
146 # 11    = E or W
147 # 12    = Checksum
148 #
149 ###################################################
150
151 # Debugging options
```

```
152 debug = False
153
154
155 # http://en.proft.me/2015/09/20/converting-latitude-and-longitude
    -decimal-values-p/
156 def dms2dd(degrees, minutes, seconds, direction):
157     dd = float(degrees) + float(minutes)/60 + float(seconds)
    /(60*60);
158     if direction == 'S' or direction == 'W':
159         dd *= -1
160     return dd;
161
162
163 # first argument is name of output file
164 basefilename = sys.argv[1]
165
166 # filenames
167 readBagFileName = basefilename # + '.bag'
168 writeBagFileName =  basefilename + '-updated.bag'
169
170 if spareGps:
171     writeCsvGPRMCFilename = basefilename + '-GPRMC-sparse.csv'
172     writeCsvDMSFilename = basefilename + '-DMS-sparse.csv'
173     writeCsvDegreesFilename = basefilename + '-degrees-sparse.csv
    ,
174 else:
175     writeCsvGPRMCFilename = basefilename + '-GPRMC-dense.csv'
176     writeCsvDMSFilename = basefilename + '-DMS-dense.csv'
```

```
177     writeCsvDegreesFilename = basefilename + '-degrees-dense.csv'
178
179
180
181 if readInput:
182     # second argument is name of input file
183     inputFilename = sys.argv[2]
184     inputFile = open(inputFilename)
185
186
187 readBagFile = rosbag.Bag(readBagFileName)
188
189 if writeToBag:
190     writeBagFile = rosbag.Bag(writeBagFileName, 'w')
191
192 if writeToCsvGPRMC:
193     writeCsvGPRMCFile = open(writeCsvGPRMCFilename, 'w')
194
195 if writeToCsvDMS:
196     writeCsvDMSFile = open(writeCsvDMSFilename, 'w')
197
198 if writeToCsvDegrees:
199     writeCsvDegreesFile = open(writeCsvDegreesFilename, 'w')
200
201 lastGps = ""
202 nextLine = ""
203
204 if readInput:
```

```
205
206         # First Line from input file
207         nextLine = inputFile.readline()
208         nextList = nextLine.split("\"")
209
210         if nextLine != "":
211             nextTopic = str(nextList[3])
212
213             nextMsg = String()
214             nextMsg.data = nextList[5]
215
216             nextTimestamp = rospy.rostime.Time()
217             nextTimestamp.secs = int(nextList[1][0:10])
218             nextTimestamp.nsecs = int(nextList[1][10:])
219
220 # Loop through bag file
221 for topic, msg, timestamp in readBagFile.read_messages():
222
223     # if re-writing with input gps
224     if nextLine != "" and  readInput and writeToBag:
225
226         # for every message in the bag file, insert any prior
    messages from gps input file
227         while nextLine != "" and (nextTimestamp.secs < timestamp.
    secs or (nextTimestamp.secs == timestamp.secs and
    nextTimestamp.nsecs <= timestamp.nsecs)):
228             writeBagFile.write(nextTopic, nextMsg, nextTimestamp)
229
```

```
230             # Next Line from input file
231             nextLine = inputFile.readline()
232             nextList = nextLine.split("\"")
233
234             if nextLine != "":
235                 nextTopic = str(nextList[3])
236
237                 nextMsg = String()
238                 nextMsg.data = nextList[5]
239
240                 nextTimestamp = rospy.rostime.Time()
241                 nextTimestamp.secs = int(nextList[1][0:10])
242                 nextTimestamp.nsecs = int(nextList[1][10:])
243
244
245     #### format from $GPRMC ##############################
246     # 3849.8998,N,07719.5674,W
247     #
248     ### format for input into google ####################
249     #
250     # 38 49.8987, -77 19.5685
251     # [- or blank][degree][space][minutes.decimal minutes],[space
        ][- or blank][degree][space][minutes.decimal minutes]
252     ######################################################
253
254     ######################################################
255     # http://static.garmincdn.com/pumac/GPS_18x_Tech_Specs.pdf
256     #
```

```
257         # Latitude ,    ddmm.mmmm format for GPS 18x PC/LVC;  ddmm.
        mmmmm format for GPS 18x-5Hz (leading zeros must be
        transmitted)
258         # Longitude , dddmm.mmmm format for GPS 18x PC/LVC; dddmm.
        mmmmm format for GPS 18x-5Hz (leading zeros must be
        transmitted)
259         #
260         ############################################################
261
262         if topic == '/telemetry':
263             GPRMC = str(msg)[11:len(str(msg))-1]
264             GPRMCList = GPRMC.split(",")
265
266
267             # Change from N/S W/E to + or -
268             if GPRMCList[4] == 'N':
269                 latSign = ""
270             else:
271                 latSign = "-"
272
273             # Latitude ,  ddmm.mmmmm format for GPS 18x-5Hz (leading
        zeros must be transmitted)
274             Latdir = GPRMCList[4]
275             Latdd = int(GPRMCList[3][0:2])
276             Latmm = float("0" + GPRMCList[3][2:9])
277             Latss = 0
278
279             if GPRMCList[6] == 'W':
```

```
280             longSign = "-"
281         else:
282             longSign = ""
283
284         # Longitude, dddmm.mmmmm format for GPS 18x-5Hz (leading
    zeros must be transmitted)
285         Longdir = GPRMCList[6]
286         Longdd = int(GPRMCList[5][0:3])
287         Longmm = float("0" + GPRMCList[5][3:10])
288         Longss = 0
289
290
291
292         # Reformat GPS String [to google format]
293         DMS = str(latSign) + str(Latdd) + " " + str(Latmm) + ","
    + str(longSign) + str(Longdd) + " " + str(Longmm)
294
295         LatDegrees = dms2dd(Latdd, Latmm, Latss, Latdir)
296         LongDegress = dms2dd(Longdd, Longmm, Longss, Longdir)
297
298         Degress = str(LatDegrees) + "," + str(LongDegress)
299
300
301         # CSV Format for Degrees
302         CsvDegrees= "\"" + str(timestamp) + "\",\"" + str(topic)
    + "\",\"" + Degress + "\""
303
304         # CSV Format for DMS
```

```
305         CsvDMS = "\"" + str(timestamp) + "\",\"" + str(topic) + "
    \",\"" + DMS + "\""
306
307         # CSV Format for $GPRMC
308         CsvGPRMC = "\"" + str(timestamp) + "\",\"" + str(topic) +
     "\",\"" + GPRMC + "\""
309
310
311         if spareGps == False or lastGps != DMS:
312
313             if debug:
314                 print Degress
315
316             if writeToCsvGPRMC:
317                 writeCsvGPRMCFile.write(CsvGPRMC + "\n")
318
319             if writeToCsvDMS:
320                 writeCsvDMSFile.write(CsvDMS + "\n")
321
322             if writeToCsvDegrees:
323                 writeCsvDegreesFile.write(CsvDegrees + "\n")
324
325             lastGps = DMS
326
327     # write to new output file
328     if writeToBag:
329         if topic != '/telemetry' or writeIgnoreSourceGPS == False
    :
```

```
330                 writeBagFile.write(topic, msg, timestamp)
331
332 # write any gps points that exist after data in bag file
333 if nextLine != "" and  readInput and writeToBag:
334
335     # for every message in the bag file, insert any prior
    messages from gps input file
336     while nextLine != "" and (nextTimestamp.secs < timestamp.secs
     or (nextTimestamp.secs == timestamp.secs and nextTimestamp.
    nsecs <= timestamp.nsecs)):
337         writeBagFile.write(nextTopic, nextMsg, nextTimestamp)
338
339         # Next Line from input file
340         nextLine = inputFile.readline()
341         nextList = nextLine.split("\"")
342
343         if nextLine != "":
344             nextTopic = str(nextList[3])
345
346             nextMsg = String()
347             nextMsg.data = nextList[5]
348
349             nextTimestamp = rospy.rostime.Time()
350             nextTimestamp.secs = int(nextList[1][0:10])
351             nextTimestamp.nsecs = int(nextList[1][10:])
352
353 # close bag files
354 readBagFile.close()
```

```
355
356 if writeToBag:
357     writeBagFile.close()
358 if writeToCsvDMS:
359     writeCsvDMSFile.close()
360 if writeToCsvDegrees:
361     writeCsvDegreesFile.close()
362
363 if writeToCsvGPRMC:
364     writeCsvGPRMCFile.close()
```

Appendix D: Configuration

D.1 Cartographer Main LUA Configuration

This is the main Cartographer configuration file. scans_per_accumulation should be set to a value between 2 and 10 for the velodyne. optimize_every_n_scans should also be set to a low number to have global slam performed frequently.

```
include "map_builder.lua"
include "trajectory_builder.lua"

options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  published_frame = "base_link",
  odom_frame = "odom",
  provide_odom_frame = true,
  use_odometry = false,
  num_laser_scans = 0,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
  num_point_clouds = 1,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
}
```

```
22
23 TRAJECTORY_BUILDER_3D.scans_per_accumulation = 2
24
25 -- default 5
26 -- TRAJECTORY_BUILDER_3D.ceres_scan_matcher.
       occupied_space_weight_0 = 1
27 -- TRAJECTORY_BUILDER_3D.ceres_scan_matcher.
       occupied_space_weight_1 = 1
28 -- TRAJECTORY_BUILDER_3D.ceres_scan_matcher.translation_weight =
       1
29
30 -- TRAJECTORY_BUILDER_2D.scans_per_accumulation = 2
31
32 -- TRAJECTORY_BUILDER_3D.submaps.number_range_data = 160
33
34
35 MAP_BUILDER.use_trajectory_builder_3d = true
36 -- MAP_BUILDER.use_trajectory_builder_2d = true
37 MAP_BUILDER.num_background_threads = 7
38 SPARSE_POSE_GRAPH.optimization_problem.huber_scale = 5e2
39 -- SPARSE_POSE_GRAPH.optimization_problem.acceleration_weight = 1
       e3
40 -- SPARSE_POSE_GRAPH.optimization_problem.rotation_weight = 3e5
41 SPARSE_POSE_GRAPH.optimize_every_n_scans = 8
42 SPARSE_POSE_GRAPH.constraint_builder.sampling_ratio = 0.03
43 SPARSE_POSE_GRAPH.optimization_problem.ceres_solver_options.
       max_num_iterations = 20
44 SPARSE_POSE_GRAPH.constraint_builder.min_score = 0.62
```

```
45 TRAJECTORY_BUILDER_3D.use_online_correlative_scan_matching =
       false
46 return options
```

D.2 Unified Robot Description Format (URDF)

```
1
2  <robot name="jeep">
3    <material name="orange">
4      <color rgba="1.0 0.5 0.2 1" />
5    </material>
6    <material name="gray">
7      <color rgba="0.2 0.2 0.2 1" />
8    </material>
9
10
11   <link name="usb_cam_link">
12     <visual>
13       <origin xyz="0.0 0.0 0.0" />
14       <geometry>
15         <cylinder length="0.25" radius="0.5" />
16       </geometry>
17       <material name="gray" />
18     </visual>
19   </link>
20
21   <link name="lidar_link">
22     <visual>
```

```
23        <origin xyz="0.0 0.0 0.0" />
24        <geometry>
25          <cylinder length="0.25" radius="0.5" />
26        </geometry>
27        <material name="gray" />
28      </visual>
29    </link>
30
31
32    <link name="imu_link">
33      <visual>
34        <origin xyz="0.0 0.9 0.0" />
35        <geometry>
36          <cylinder length="0.25" radius="0.5" />
37        </geometry>
38        <material name="gray" />
39      </visual>
40    </link>
41
42
43    <link name="base_link" />
44
45
46    <joint name="usb_cam_link_joint" type="fixed">
47      <parent link="base_link" />
48      <child link="usb_cam_link" />
49      <origin xyz="0.0 0.0 0.0" rpy="0.0 0.0 0.0" />
50    </joint>
```

```
51
52    <joint name="lidar_link_joint" type="fixed">
53      <parent link="base_link" />
54      <child link="lidar_link" />
55      <origin xyz="0.0 0.0 0.9" rpy="0.0 0.0 0.0" />
56    </joint>
57
58    <joint name="imu_link_joint" type="fixed">
59      <parent link="base_link" />
60      <child link="imu_link" />
61      <origin xyz="0.0 0.0 0.0" rpy="-0.16 0.0 0.0" />
62    </joint>
63
64  </robot>
```

D.3 Launch Configuration for High Definition Camera

```
1  <launch>
2    <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="
       screen" >
3      <param name="video_device" value="/dev/video0" />
4      <param name="image_width" value="1280" />
5      <param name="image_height" value="720" />
6      <param name="pixel_format" value="yuyv" />
7      <param name="camera_frame_id" value="usb_cam" />
8      <param name="io_method" value="mmap"/>
9    </node>
```

```
10    <node name="image_view" pkg="image_view" type="image_view"
       respawn="false" output="screen">
11      <remap from="image" to="/usb_cam/image_raw"/>
12      <param name="autosize" value="true" />
13    </node>
14  </launch>
```

D.4 Launch Configuration for VLP-16 and other Sensors

```
1  <launch>
2
3  <!-- **** Start: Robot Description * -->
4    <param name="robot_description" textfile="$(find gmu)/urdf/jeep
       .urdf" />
5
6    <!-- **** Start: Robot Description * -->
7    <node name="robot_state_publisher" pkg="robot_state_publisher"
       type="robot_state_publisher" />
8
9    <!-- USB CAMERA -->
10   <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="
       screen" >
11      <param name="video_device" value="/dev/video0" />
12      <param name="image_width" value="1280" />
13      <param name="image_height" value="720" />
14      <param name="pixel_format" value="yuyv" />
15      <param name="camera_frame_id" value="usb_cam_link" />
16      <param name="io_method" value="mmap"/>
```

```
17    </node>
18  <!--
19    <node name="image_view" pkg="image_view" type="image_view"
       respawn="false" output="screen">
20      <remap from="image" to="/usb_cam/image_raw"/>
21      <param name="autosize" value="true" />
22    </node>
23
24  -->
25   <!-- IMU -->
26   <node name="imu" pkg="um7" type="um7_driver" output="screen">
27      <param name="port" value="/dev/ttyUSB0" />
28      <param name="baud" value="115200" />
29      <param name="frame_id" value="imu_link" />
30      <param name="mag_updates" value="true" />
31      <param name="quat_mode" value="true" />
32      <param name="zero_gyros" value="true" />
33      <param name="covariance" value="0 0 0 0 0 0 0 0 0" />
34  </node>
35
36  <!-- declare arguments with default values -->
37    <arg name="calibration" default="$(find velodyne_pointcloud)/
       params/VLP16db.yaml"/>
38    <arg name="device_ip" default="192.168.1.201" />
39    <arg name="frame_id" default="lidar_link" />
40    <arg name="manager" default="$(arg frame_id)_nodelet_manager"
       />
41    <arg name="max_range" default="130.0" />
```

```
42  <arg name="min_range" default="0.4" />
43  <arg name="pcap" default="" />
44  <arg name="port" default="2368" />
45  <arg name="read_fast" default="false" />
46  <arg name="read_once" default="false" />
47  <arg name="repeat_delay" default="0.0" />
48  <arg name="rpm" default="600.0" />
49
50  <!-- start nodelet manager and driver nodelets -->
51  <include file="$(find velodyne_driver)/launch/nodelet_manager.
     launch">
52    <arg name="device_ip" value="$(arg device_ip)"/>
53    <arg name="frame_id" value="$(arg frame_id)"/>
54    <arg name="manager" value="$(arg manager)" />
55    <arg name="model" value="VLP16"/>
56    <arg name="pcap" value="$(arg pcap)"/>
57    <arg name="port" value="$(arg port)"/>
58    <arg name="read_fast" value="$(arg read_fast)"/>
59    <arg name="read_once" value="$(arg read_once)"/>
60    <arg name="repeat_delay" value="$(arg repeat_delay)"/>
61    <arg name="rpm" value="$(arg rpm)"/>
62  </include>
63
64  <!-- start cloud nodelet -->
65  <include file="$(find velodyne_pointcloud)/launch/cloud_nodelet
     .launch">
66    <arg name="calibration" value="$(arg calibration)"/>
67    <arg name="manager" value="$(arg manager)" />
```

```
68      <arg name="max_range" value="$(arg max_range)"/>
69      <arg name="min_range" value="$(arg min_range)"/>
70    </include>
71
72    <node pkg="nodelet" type="nodelet" name="$(arg manager)
        _transform" args="load velodyne_pointcloud/TransformNodelet $(
        arg manager)" >
73      <param name="calibration" value="$(arg calibration)"/>
74      <param name="frame_id" value="lidar_link"/>
75      <param name="max_range" value="$(arg max_range)"/>
76      <param name="min_range" value="$(arg min_range)"/>
77    </node>
78
79  <node pkg="rosbag" type="record" name="record_data"
80          args="record -o /home/user/record/gmu /imu/data /
       velodyne_points /usb_cam/image_raw /telemetry" />
81 </launch>
```

D.5 Launch Configuration for VLP-64 and other Sensors

```
1 <launch>
2
3 <!-- **** Start: Robot Description * -->
4   <param name="robot_description" textfile="$(find gmu)/urdf/jeep
      .urdf" />
5
6   <!-- **** Start: Robot Description * -->
```

```
7    <node name="robot_state_publisher" pkg="robot_state_publisher"
       type="robot_state_publisher" />
8
9    <!-- USB CAMERA -->
10   <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="
       screen" >
11     <param name="video_device" value="/dev/video0" />
12     <param name="image_width" value="1280" />
13     <param name="image_height" value="720" />
14     <param name="pixel_format" value="yuyv" />
15     <param name="camera_frame_id" value="usb_cam_link" />
16     <param name="io_method" value="mmap"/>
17   </node>
18 <!--
19   <node name="image_view" pkg="image_view" type="image_view"
       respawn="false" output="screen">
20     <remap from="image" to="/usb_cam/image_raw"/>
21     <param name="autosize" value="true" />
22   </node>
23
24 -->
25  <!-- IMU -->
26  <node name="imu" pkg="um7" type="um7_driver" output="screen">
27     <param name="port" value="/dev/ttyUSB0" />
28     <param name="baud" value="115200" />
29     <param name="frame_id" value="imu_link" />
30     <param name="mag_updates" value="true" />
31     <param name="quat_mode" value="true" />
```

```
32      <param name="zero_gyros" value="true" />
33      <param name="covariance" value="0 0 0 0 0 0 0 0 0 0" />
34  </node>
35
36  <!-- declare arguments with default values -->
37    <arg name="calibration" default="$(find velodyne_pointcloud)/
        params/64e_utexas.yaml"/>
38    <arg name="device_ip" default="192.168.3.43" />
39    <arg name="frame_id" default="lidar_link" />
40    <arg name="manager" default="$(arg frame_id)_nodelet_manager"
        />
41    <arg name="max_range" default="130.0" />
42    <arg name="min_range" default="4.0" />
43    <arg name="pcap" default="" />
44    <arg name="port" default="2368" />
45    <arg name="read_fast" default="false" />
46    <arg name="read_once" default="false" />
47    <arg name="repeat_delay" default="0.0" />
48    <arg name="rpm" default="600.0" />
49
50    <!-- start nodelet manager and driver nodelets -->
51    <include file="$(find velodyne_driver)/launch/nodelet_manager.
        launch">
52      <arg name="device_ip" value="$(arg device_ip)"/>
53      <arg name="frame_id" value="$(arg frame_id)"/>
54      <arg name="manager" value="$(arg manager)" />
55      <arg name="model" value="VLP16"/>
56      <arg name="pcap" value="$(arg pcap)"/>
```

```
57      <arg name="port" value="$(arg port)"/>
58      <arg name="read_fast" value="$(arg read_fast)"/>
59      <arg name="read_once" value="$(arg read_once)"/>
60      <arg name="repeat_delay" value="$(arg repeat_delay)"/>
61      <arg name="rpm" value="$(arg rpm)"/>
62    </include>
63
64    <!-- start cloud nodelet -->
65    <include file="$(find velodyne_pointcloud)/launch/cloud_nodelet
        .launch">
66      <arg name="calibration" value="$(arg calibration)"/>
67      <arg name="manager" value="$(arg manager)" />
68      <arg name="max_range" value="$(arg max_range)"/>
69      <arg name="min_range" value="$(arg min_range)"/>
70    </include>
71
72    <node pkg="nodelet" type="nodelet" name="$(arg manager)
        _transform" args="load velodyne_pointcloud/TransformNodelet $(
        arg manager)" >
73      <param name="calibration" value="$(arg calibration)"/>
74      <param name="frame_id" value="lidar_link"/>
75      <param name="max_range" value="$(arg max_range)"/>
76      <param name="min_range" value="$(arg min_range)"/>
77    </node>
78
79  <node pkg="rosbag" type="record" name="record_data"
80          args="record -o /media/user/easystore/record/gmu /imu/data
         /velodyne_points /usb_cam/image_raw /telemetry" />
```

```
81 </launch>
```

Appendix E: Demonstrations

E.1 Download Demonstrations from Google

The Google Team provides sample data to verify the installation is working properly, below are the commands necessary to download the sample data.[1]

```
1 wget -P /opt/Downloads https://storage.googleapis.com/
    cartographer-public-data/bags/backpack_2d/
    cartographer_paper_deutsches_museum.bag
2 wget -P /opt/Downloads https://storage.googleapis.com/
    cartographer-public-data/bags/backpack_2d/b2
    -2016-04-05-14-44-52.bag
3 wget -P /opt/Downloads https://storage.googleapis.com/
    cartographer-public-data/bags/backpack_2d/b2
    -2016-04-27-12-31-41.bag
4 wget -P /opt/Downloads https://storage.googleapis.com/
    cartographer-public-data/bags/backpack_3d/b3
    -2016-04-05-14-14-00.bag
5 wget -P /opt/Downloads https://storage.googleapis.com/
    cartographer-public-data/bags/revo_lds/
    cartographer_paper_revo_lds.bag
6 wget -P /opt/Downloads https://storage.googleapis.com/
    cartographer-public-data/bags/pr2/2011-09-15-08-32-46.bag
7 wget -P /opt/Downloads https://storage.googleapis.com/
    cartographer-public-data/bags/taurob_tracker/
    taurob_tracker_simulation.bag
```

E.2 Google SLAM Demonstrations

Below is a series of different demonstrations to utilize the sample data and watch the system map different environments to verify the system is operational.[1]

```
# Launch the 2D backpack demo.
roslaunch cartographer_ros demo_backpack_2d.launch bag_filename
    :=/opt/Downloads/cartographer_paper_deutsches_museum.bag
# Generate the map: Run the next command, wait until
    cartographer_offline_node finishes.
roslaunch cartographer_ros offline_backpack_2d.launch
    bag_filenames:=/opt/Downloads/b2-2016-04-05-14-44-52.bag
# Run pure localization:
roslaunch cartographer_ros demo_backpack_2d_localization.launch \
    bag_filename:=/opt/Downloads/b2-2016-04-27-12-31-41.bag \
    map_filename:=/opt/Downloads/b2-2016-04-05-14-44-52.bag.
    pbstream
# Launch the 3D backpack demo.
roslaunch cartographer_ros demo_backpack_3d.launch bag_filename
    :=/opt/Downloads/bag/b3-2016-04-05-14-14-00.bag
# Launch the Revo LDS demo.
roslaunch cartographer_ros demo_revo_lds.launch bag_filename:=/
    opt/Downloads/cartographer_paper_revo_lds.bag
# Launch the PR2 demo.
roslaunch cartographer_ros demo_pr2.launch bag_filename:=/opt/
    Downloads/2011-09-15-08-32-46.bag
# Launch the Taurob Tracker demo.
roslaunch cartographer_ros demo_taurob_tracker.launch
    bag_filename:=/opt/Downloads/taurob_track
```

This demonstrations can be visualized through ROS tools such as rviz, and data topics can be monitored through rosecho.

E.3 Visualize existing 'Map GMU' data

Open 5 seperate linux termnial windows and execute the following commands in order. [19]

Download, Unzip, and Convert the xml configuration file into yaml format.

```
1
2 # install 7zip to decompress files from GMU
3 sudo apt install p7zip-full
4
5 # Download files from GMU
6 mkdir -p /opt/Downloads/pcap/gmu
7 cd /opt/Downloads/pcap/gmu
8 wget http://masc.cs.gmu.edu/wiki/uploads/MapGMU/velodyne-calib.
    xml
9 wget http://masc.cs.gmu.edu/wiki/uploads/MapGMU/Nottoway-Annex-
    track5-fast-1loop.7z
10 7z e Nottoway-Annex-track5-fast-1loop.7z
11
12 # Convert xml file into yaml file
13 rosrun velodyne_pointcloud gen_calibration.py velodyne-calib.xml
```

Velodyne pcap file to bag file conversion: [20]

The following steps can be performed to convert an existing pcap file in raw Velodyne format into ROS BAG format.[11]

```
1 # install 7zip to decompress files from GMU
2 sudo apt install p7zip-full
3 # steps to convert
```

```
roscore &
rosrun velodyne_driver velodyne_node _model:=VLP16 _pcap:=/opt/
    Downloads/Nottoway-Annex-track2-slow.pcap _read_once:=true &
rosrun rosbag record -O your_vlp16_070815.bag /velodyne_packets
```

E.4 PLY to OBJ Conversion

Cartographer exports point cloud in Polygon File Format (PLY), in x,y,z binary Little Endian. The end format of the data needs to be on Wavefront OBJ format. The task to convert from PLY to OBJ became trivial when we discovered Point Cloud Library (PCL) provides a library that does this conversion. We did find however there are limitations on the library and it will stack overflow if the PLY file size exceeds the stack size allocated in the source code of the PCL Library.

An example of conversion between file formats is presented for Shenandoah Parking Deck.

E.5 Poisson Surface Reconstruction

The next conversion was to move from a point cloud representation to a surface representation.

Using Meshlab[14] we cleaned the noise from 3d point clouds. We computed normals for point sets, using 10 neighbors. With the normals to the points I then performed surface Poisson reconstruction[15] of the Point Cloud.

The immediate discovery with Poisson reconstruction is it will connect all the points and essential make caves. Surface reconstruction will require hand editing the point clouds and either segmenting each object for Poisson reconstruction or removing all the background and noise to only reconstruct the road surface. I verified the point clouds created were compatible with surface reconstruction techniques such as Poisson.

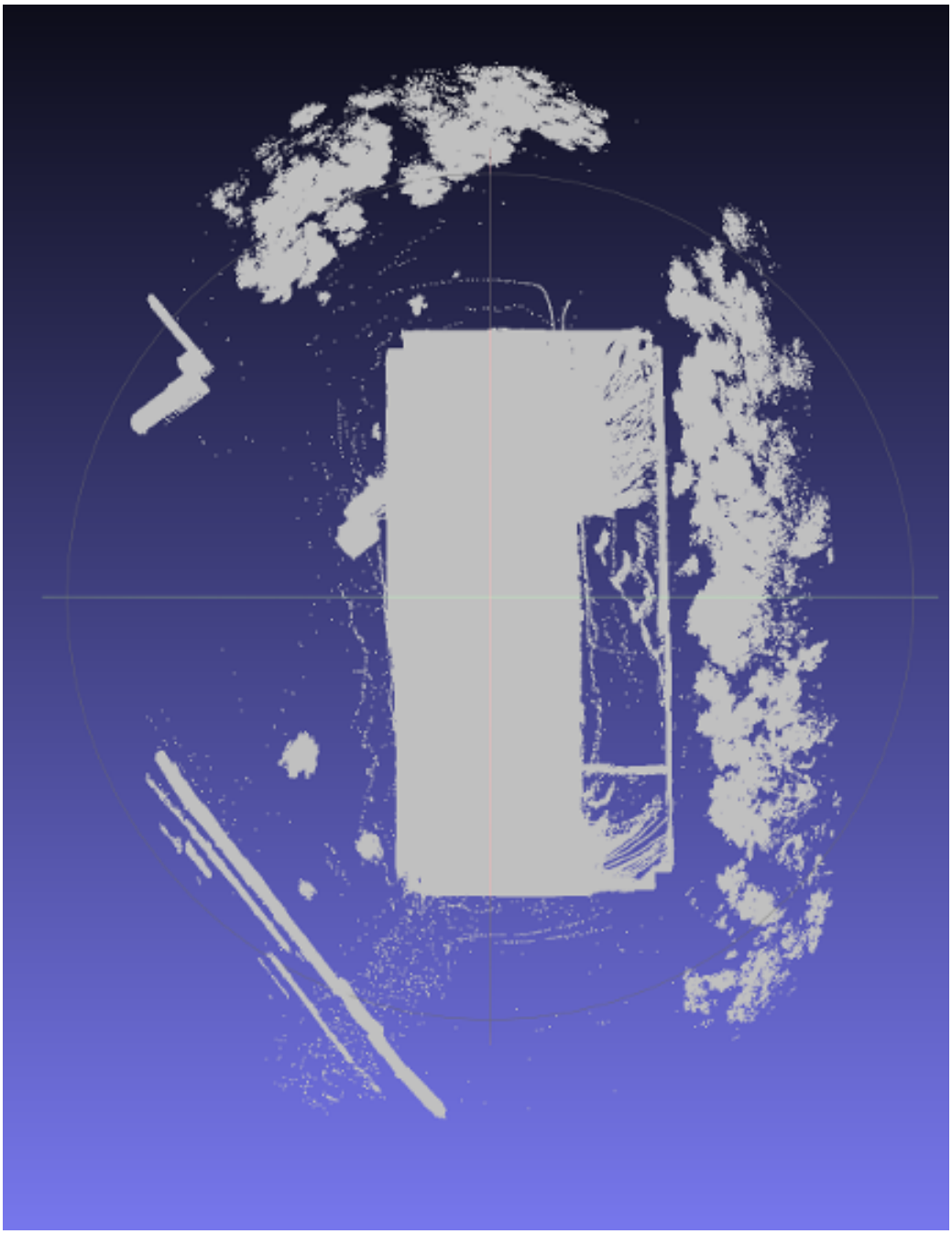


Figure E.1: PLY Shenandoah Parking Deck

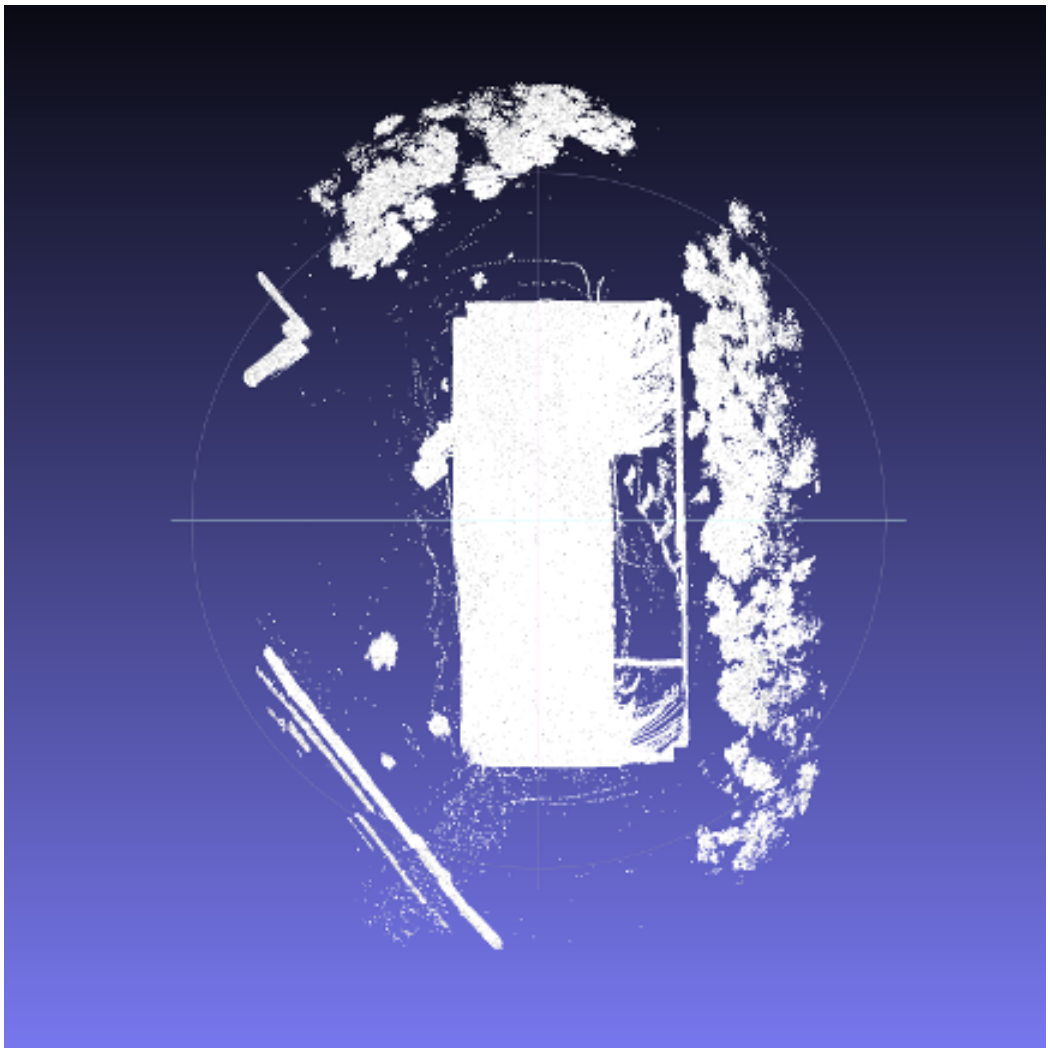


Figure E.2: OBJ Shenandoah Parking Deck

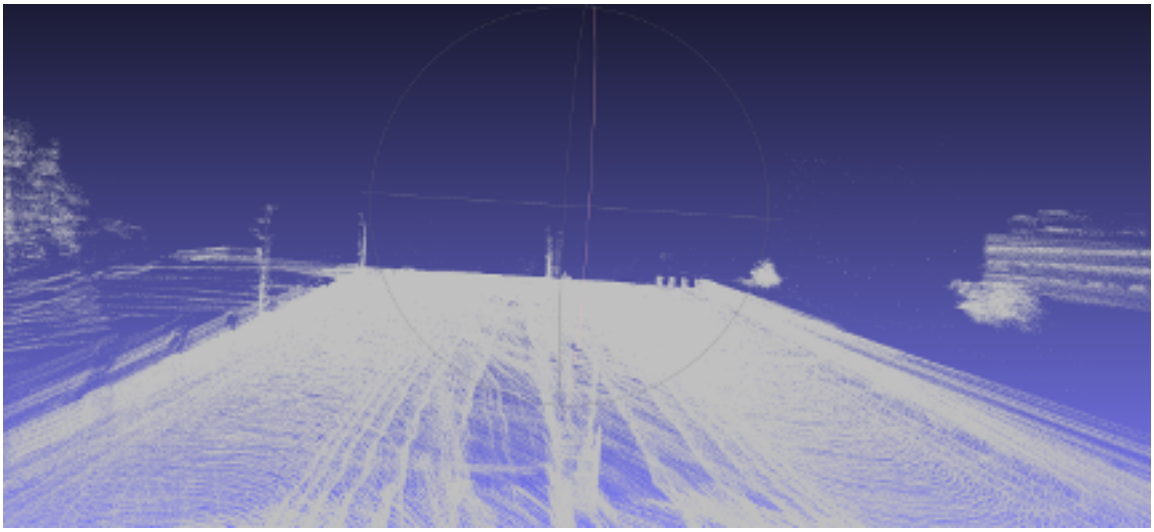


Figure E.3: Before Poisson Shenandoah Parking Deck

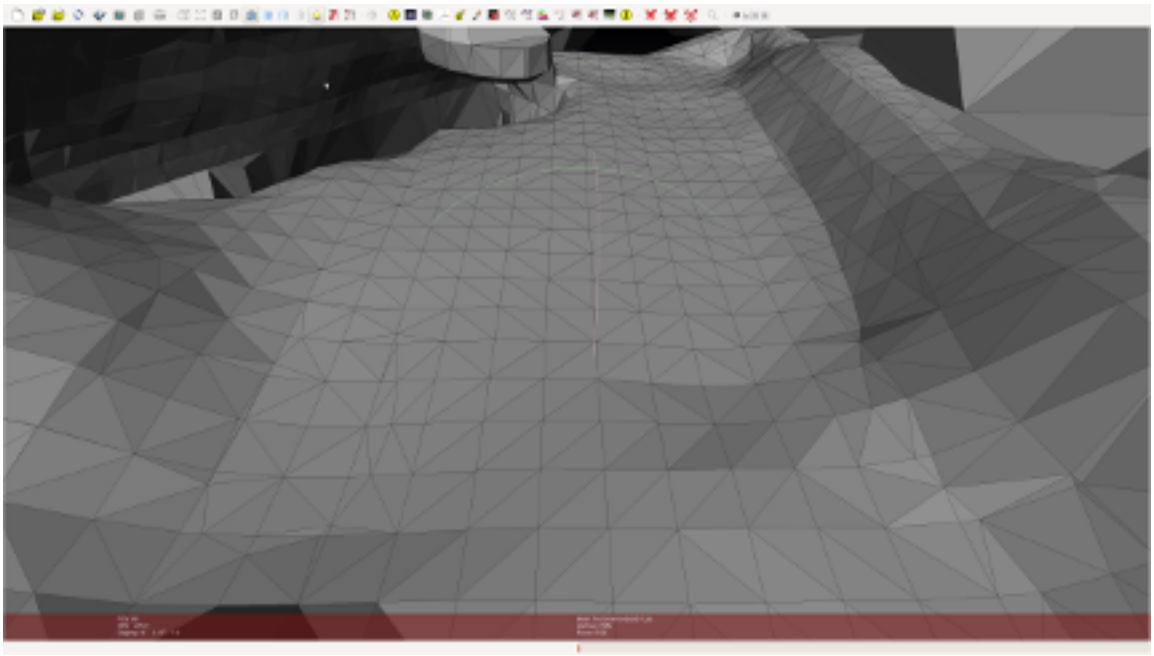


Figure E.4: After Poisson Shenandoah Parking Deck

Bibliography

Bibliography

[1] T. C. Authors. (2017) Cartographer ros integration. [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/

[2] e. a. Yu-Ta Tsai. (2017) Map gmu. [Online]. Available: http://masc.cs.gmu.edu/wiki/MapGMU

[3] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[4] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 1271–1278.

[5] E. Olson, "M3rsm: Many-to-many multi-resolution scan matching," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5815–5821.

[6] E. B. Olson, "Real-time correlative scan matching," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 4387–4393.

[7] J. Strom and E. Olson, "Occupancy grid rasterization in large environments for teams of robots," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 4271–4276.

[8] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell, "Branch-and-bound algorithms," *Discret. Optim.*, vol. 19, no. C, pp. 79–102, Feb. 2016. [Online]. Available: http://dx.doi.org/10.1016/j.disopt.2016.01.005

[9] J. Levinson and S. Thrun, "Map-based precision vehicle localization in urban environments," in *Robotics: Science and Systems*, 2007.

[10] ——, "Robust vehicle localization in urban environments using probabilistic maps," in *International Conference on Robotics and Automation*, 2010.

[11] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, June 2011, pp. 163–168.

[12] H. Yin and C. Berger, "When to use what data set for your self-driving car algorithm: An overview of publicly available driving datasets," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, Oct 2017, pp. 1–8.

[13] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2d slam techniques available in robot operating system," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Oct 2013, pp. 1–6.

[14] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, "MeshLab: an Open-Source Mesh Processing Tool," in *Eurographics Italian Chapter Conference*, V. Scarano, R. D. Chiara, and U. Erra, Eds. The Eurographics Association, 2008.

[15] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, ser. SGP '06. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, pp. 61–70. [Online]. Available: http://dl.acm.org/citation.cfm?id=1281957.1281965

[16] Google. (2017) Cartographer. [Online]. Available: https://google-cartographer.readthedocs.io/en/latest/

[17] ROS. (2017) Ubuntu install of ros kinetic. [Online]. Available: http://wiki.ros.org/kinetic/Installation/Ubuntu

[18] ——. (2017) Ros answers. [Online]. Available: https://answers.ros.org/question/226594/how-do-i-build-ros-vlp16-velodyne-driver-for-indigo-using-catkin/

[19] ——. (2017) velodyne_pointcloud. [Online]. Available: http://wiki.ros.org/velodyne_pointcloud

[20] ——. (2017) Ros answers. [Online]. Available: https://answers.ros.org/question/213080/convert-raw-velodyne-vlp16-pcap-to-bagfile/

Curriculum Vitae

Nathan C. Obert was been employeed at fortune 500 and government agencies for over two decades. He returned to school and received his Bachelor of Science from George Mason University in 2015. He currently is supporting the Joint Chiefs of Staff, US Senate and New York State Government.