Real-Time Semantic Mapping of Tree Topology Using Deep Learning and Multi-Sensor Factor Graph

by

Rakshith Vishwanatha

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved March 2022 by the
Graduate Supervisory Committee:

Jnaneshwar Das, Chair
Roberta Martin
Heather Throop
Wenlong Zhang
Reza Ehsani

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

Physical and structural tree measurements are applied in forestry, precision agriculture and conservation for various reasons. Since measuring tree properties manually is tedious, measurements from only a small subset of trees present in a forest, agricultural land or survey site are often used. Utilizing robotics to autonomously estimate physical tree dimensions would speed up the measurement or data collection process and allow for a much larger set of trees to be used in studies. In turn, this would allow studies to make more generalizable inferences about areas with trees.

To this end, this thesis focuses on developing a system that generates a semantic representation of the topology of a tree in real-time. The first part describes a simulation environment and a real-world sensor suite to develop and test the tree mapping pipeline proposed in this thesis. The second part presents details of the proposed tree mapping pipeline. Stage one of the mapping pipeline utilizes a deep learning network to detect woody and cylindrical portions of a tree like trunks and branches based on popular semantic segmentation networks. Stage two of the pipeline proposes an algorithm to separate the detected portions of a tree into individual trunk and branch segments. The third stage implements an optimization algorithm to represent each segment parametrically as a cylinder. The fourth stage formulates a multi-sensor factor graph to incrementally integrate and optimize the semantic tree map while also fusing two forms of odometry. Finally, results from all the stages of the tree mapping pipeline using simulation and real-world data are presented. With these implementations, this thesis provides an end-to-end system to estimate tree topology through semantic representations for forestry and precision agriculture applications.

DEDICATION

To ever-curios minds and those who inspired me to have one.

ACKNOWLEDGEMENTS

First off, I would like to thank my advisor Dr. Jnaneshwar Das for providing his continuous support and guidance during the entire duration of this project. He helped me get exposure to a vast array of topics from an algorithmic, hardware and software standpoint. Next, I would like to thank Dr. Roberta Martin, Dr. Heather Throop and Dr. Reza Ehsani for providing insights on what aspects of this project would be of importance from a forestry, ecology and precision agriculture perspective. Working on a project with useful applications helped me stay motivated and energetic during this research work. I would also like to thank Dr. Wenlong Zhang for firstly allowing me to work at his lab during my first year at Arizona State University and secondly for taking the interest to serve on this committee. The experience I got while working at Dr. Zhang's Robotics and Intelligent SystEms (RISE) Lab in the first year helped me make quick progress at Dr. Das' Distributed Robotic Exploration and Mapping Systems (DREAMS) Lab. Finally, I would like to thank all the lab members for creating an amazing environment to work in. A.L.G. Prasad, Zhiang Chen, Dr. Luiza Aparecido, Devin Keating, Darwin Mick, Abdullah Masud and others kept the day-to-day work at the lab enjoyable and helped immensely in proposing this thesis topic and having technical discussions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Thesis Statement

Real-time semantic mapping of tree topology enables fast and precise estimation of tree characteristics for use in forestry and precision agriculture.

1.2 Contribution

This thesis describes a mapping pipeline to extract semantic information from a tree by estimating its physical dimensions and building a parametric model of its topology by using state-of-the-art sensors and real-time capable computation tools. The six main contributions of this thesis are:

- A simulation environment containing a leaf-less tree and a lidar mounted drone to develop and test the proposed tree mapping pipeline in a simplified scenario

- A hardware sensor rig equipped with state-of-the-art sensors to collect pose estimates and 3D point cloud data for semantic tree mapping in the real world

- A deep learning network to identify woody portions of a tree like trunk and branches while ignoring surrounding objects

- An algorithm to split the identified woody portions of the tree into individual trunk and branch segments

- An optimization framework to parameterize each trunk and branch segment into uniquely identifiable cylinders that collectively represent a portion of the semantic tree map

- A multi-sensor factor graph that incorporates two forms of odometry and incrementally assembles the semantic tree map and optimizes its parameters

1.3 Document Overview

This thesis is arranged into seven chapters. This chapter, Chapter 1, provides the thesis statement and contributions of this work. Chapter 2 overviews recent efforts and related work in the area of mapping forests and trees. Chapter 3 describes the simulation environment and hardware sensor rig that were put together. Chapter 4 presents the semantic tree mapping pipeline developed in this thesis. Chapter 5 explains the results obtained on simulation and real-world data. Chapter 6 is the conclusion and Chapter 7 enlists limitations and future work for this thesis.

Chapter 2

RELATED WORK

A brief description of recent work related to obtaining physical dimensions of trees and the generation of their 3D map is provided in this section. Drawbacks for each of the works mentioned is detailed, and advancements that were made to address those drawbacks is then explained. The concluding paragraph of this chapter describes the significance and novelty of this thesis.

Studies that focus on surveying large forest areas at the scale of countries or continents commonly employ satellite-based or airborne remote sensing techniques [Franklin (2001)],[Wulder et al. (2008)]. On the other hand, use cases requiring very fine tree measurements (at the scale of flowers or leaves) usually involve manual measurements made by tree or plant surveyors. A scenario in between the two use cases such as mapping a countable number (hundreds to thousands) of trees often use terrestrial LiDAR scanners (TLS) [Dassot et al. (2011)],[Trochta et al. (2017)]. More recently, drones mounted with light detection and ranging (LiDAR, lidar) sensors are being considered to gather data from many trees in a single measurement effort, as it significantly reduces the manpower and time required to collect and process spatial data of trees.

Lidars use time-of-flight of a laser pulse to estimate distance precisely. A single laser with reflective micro-mirrors or a rotating array of lasers is usually employed to sweep through an area and assemble a 3D point cloud. Terrestrial lidar scanners are high density 3D laser scanners typically used to map forests and geological features. Two good examples of algorithms that use TLS tree data are TreeQSM (Raumonen et al. (2013)) and rTLS (Q et al. (2021)). Both these methods take a sequence of TLS

scans as the input, merge them into a single point cloud offline and then extract various physical tree properties and measurements from the integrated point cloud. Properties like diameter at breast height (DBH), thickness of branches, branching points, branch orientation, variation of branch properties along tree height, canopy volume, etc. are available through TreeQSM and rTLS. However, the main drawback of using TLS methods is the data collection step. It requires the movement and positioning of heavy TLS equipment at multiple locations and the placement of artificial scan-alignment targets in the area being mapped. Additionally, algorithms that operate on TLS data lack real-time computation and decision-making capabilities because individual TLS scans recorded during data collection need to be combined together by an offline data processing algorithm.

To address the issue of immobility of TLS sensors, the robotics community has been working on mapping trees using Unmanned Aerial Vehicles (UAVs). UAVs mounted with image and laser based sensors have the capability to collect a number of metrics quickly while moving around different locations and altitudes in a forest. The system by Fritz et al. (2013) used a UAV flying over a leaf-less tree plantation to obtain Structure from Motion (SfM) tree point clouds to then calculate DBH. Other works by Wallace et al. (2012) and Wallace et al. (2016) used a downward facing camera and a lidar on a UAV to estimate canopy shape while flying above trees. All these systems were operated in broad daylight, in clear surroundings above tree canopies, used ground control points and could therefore rely on GPS and photogrammetric Visual Inertial Odometry (VIO) for their localization and mapping algorithm. Also, they were only able to estimate basic tree traits like DBH or an approximate 2D canopy shape. Although these were great initial studies to show the versatility of UAVs and on-board sensors for operation in an environment with many trees, they were unable to navigate complex or challenging spaces underneath and around the

crown of a tree to extract detailed trunk and branch measurements.

Around 2012, an algorithm to use a 3D lidar scanner and an inertial measurement unit (IMU) for real-time robot odometry and mapping was first proposed called Lidar Odometry and Mapping (LOAM) [Zhang and Singh (2017)]. This algorithm allowed for the processing of 3D lidar scans on-board the data collection sensor rig so that maps and additional on-board decisions could be made in real-time. However, it took years of development and only in 2016 was a reliable open source implementation of the LOAM algorithm called Advanced implementation of LOAM (A-LOAM) released. Variants of LOAM by Shan and Englot (2018) and Shan et al. (2020) using scan context work by Kim and Kim (2018) for more time efficient implementations and better loop closure techniques have been released more recently, and are still being actively improved. All of these algorithms were designed keeping in mind urban scenarios and autonomous cars, but did not hold up in unstructured environments like a forest or agricultural plantation due to the lack of a flat planar and sharp edge features.

In 2020, Semantic-LOAM (SLOAM) was developed to address odometry and mapping within unstructured forest-like environments [Chen et al. (2020)]. The main novelty of SLOAM was the use of semantic segmentation and cylinder fitting to detect and model tree trunks as point cloud features, such that they could be used as landmarks while mapping a tree plantation. SLOAM provided GPS-denied robot odometry while also estimating the DBH of tall trees in real-time. This thesis is inspired by SLOAM but extends its ideas to enable single tree mapping and incremental map refinement.

Compared to existing techniques like SLOAM, the pipeline proposed in this thesis aims to map an individual tree without relying on the presence or visibility of adjacent trees. This capability becomes especially useful in cases where trees are very far

apart such as in savannas or the emergent layer of rainforests, or where undergrowth and drooping canopies obstruct the visibility of adjacent trees. Mapping of an individual tree can easily be extended to accommodate the mapping of multiple trees as well, since this thesis represents trees using semantics that are unique in 3D space. Another novelty that this thesis provides is the ability to improve the map of a tree incrementally during the course of a mapping sequence or across mapping sequences. Both of these capabilities have been brought about by the use of a range image based branch separation algorithm and a novel method to represent cylindrical features as landmarks in a factor graph. Keeping with the advancements made by LOAM and SLOAM over TLS based mapping, the mapping pipeline proposed in this thesis is also able to process live data and create maps in real-time. This thesis hence enables the estimation of semantic information about trees that is useful for forestry and precision agriculture applications with minimal manual effort.

Chapter 3

SETUP AND DATA

The goal of this thesis is to obtain semantic information of a single tree to encode its topology. In this document, all mapping and algorithm discussions pertain to mapping an individual tree although the same pipeline can be extended to store a map for multiple trees taken one at a time within a single continuous mapping effort. Semantic mapping of a tree is done by estimating the location, orientation and dimension of its trunk and branches using two types of sensors. One sensor collects 3D spatial data from a particular location in space and another sensor estimates the pose (position and orientation) at which that spatial data was recorded. Examples of sensors that possess such properties are a 3D lidar and a visual inertial odometry capable camera respectively. The tree mapping pipeline proposed in this thesis uses inputs from two such sensors to build a semantic tree map.

To develop different stages of the tree mapping pipeline, a simulation environment meeting spatial and pose estimate sensory requirements was created. To test the developed tree mapping pipeline in real-world conditions, a hardware sensor rig also having spatial and pose estimate sensors was assembled. Section 3.1 describes the simulation setup and Section 3.2 describes the data that was recorded in it, while Section 3.3 describes the hardware setup and Section 3.4 describes the real-world data that was recorded using it.

3.1 Simulation Setup

The simulation environment represented a simplified scenario of mapping trees in the real world. This environment was used to develop all the stages of the tree

mapping pipeline and each stage was first tested in simulation before being used on noisier and more complex real-world data. Gazebo was the simulation environment of choice because it supported Robot Operating System (ROS) and the Software in the Loop (SITL) PX4 Flight Controller. The simulation environment consisted of a leafless tree and a drone mounted with a lidar. The need for a drone was to provide a movable sensor suite that could continuously collect data for real-time mapping as opposed to manually placing sensors like a TLS and its associated scan alignment targets at multiple discrete locations.

The trunk of the tree was created as a tapering stack of cylinders with three branches at different heights, orientations and thicknesses. Figure 3.1 shows the simulation tree along with the radius measurements for each cylinder. The pre-defined Iris 3DR drone from the SITL PX4 flight controller package was used in this simulation. Global pose estimates of the drone were taken from PX4's MAVROS odometry topic, and a simulated version of the Velodyne VLP-16 Puck was attached to the drone to collect 3D spatial data of the tree.

3.2 Simulation Data

Data of the tree in simulation was recorded using the Iris drone. The Iris drone was programmed to fly around the tree in quadrangle patterns at increasing heights until the lidar scans lost full visibility of the tree. During this flight, all data from the Velodyne lidar and PX4 were recorded into a ROS Bag. Figure 3.2 shows data that was recorded by the lidar at two differnt locations of the drone's flight path.

3.3 Real-World Setup

A hardware sensor rig was developed to collect real-world data. This data was then used to test the developed mapping pipeline in a more realistic environment than

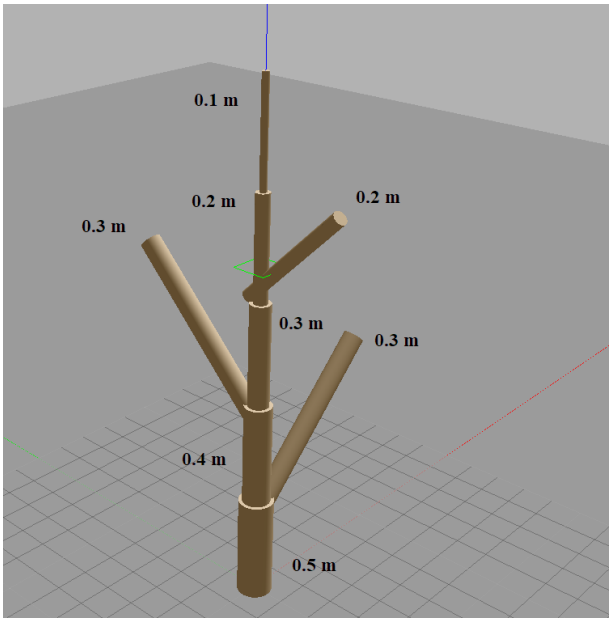

Figure 3.1: Leafless Tree With Radius Dimensions in Simulation.

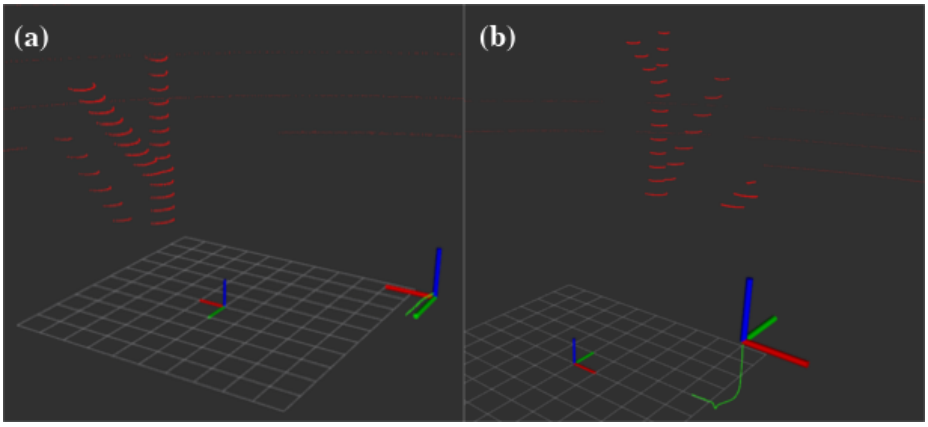

Figure 3.2: Lidar Scans of the Simulation Tree From Two Different Locations.

simulation. Components on the sensor rig were chosen such that the same components could be attached to a drone to perform large scale data recording and mapping. Thus, aspects of size, weight and power consumption were especially important when picking out sensors. This also allowed the hardware sensor rig to be light enough for use as a hand-held mapping device. The main components present on the rig were:

- Velodyne VLP-16 Puck 3D LiDAR
- Intel Realsense T-265 Tracking Stereo Camera
- Intel NUC i7
- PX4 Cube Orange
- Here2 RTK GPS
- Bullet M2 HP antenna
- Lilliput Field Monitor

It was possible to obtain 3D spatial data of trees from various sensors. The Zed2 stereo camera, Intel D435 depth camera, Livox lidar, Intel L515 solid state lidar and the Velodyne VLP-16 Puck were available options. Stereo and depth cameras were ruled out since lidars had a higher measurement precision. They were also robust against changing light conditions and sun glare which was a significant factor to consider under trees with gaps in their canopy. Lidars also had the property of measuring the intensity of their returned rays, which varied depending on the material from which they were reflected. This was an especially useful property for the neural network to contrast between leaves and non-leaves (woody trunk or branch) points. Among the lidar sensors, the Velodyne VLP-16 Puck was the only one which organized its scan into a structure of rings while having centimeter level accuracy beyond 10 meters. Thus, the Velodyne VLP-16 Puck was mounted on the sensor rig.

Pose and odometry estimates for real-world data were obtained by mounting an Intel Realsense T-265 Tracking Camera on the sensor rig. The main processing board to which all the sensor data was fed via ROS was the Intel NUC i7. It was chosen due to its computation ability despite its light weight. 16GB RAM and 256GB disk space

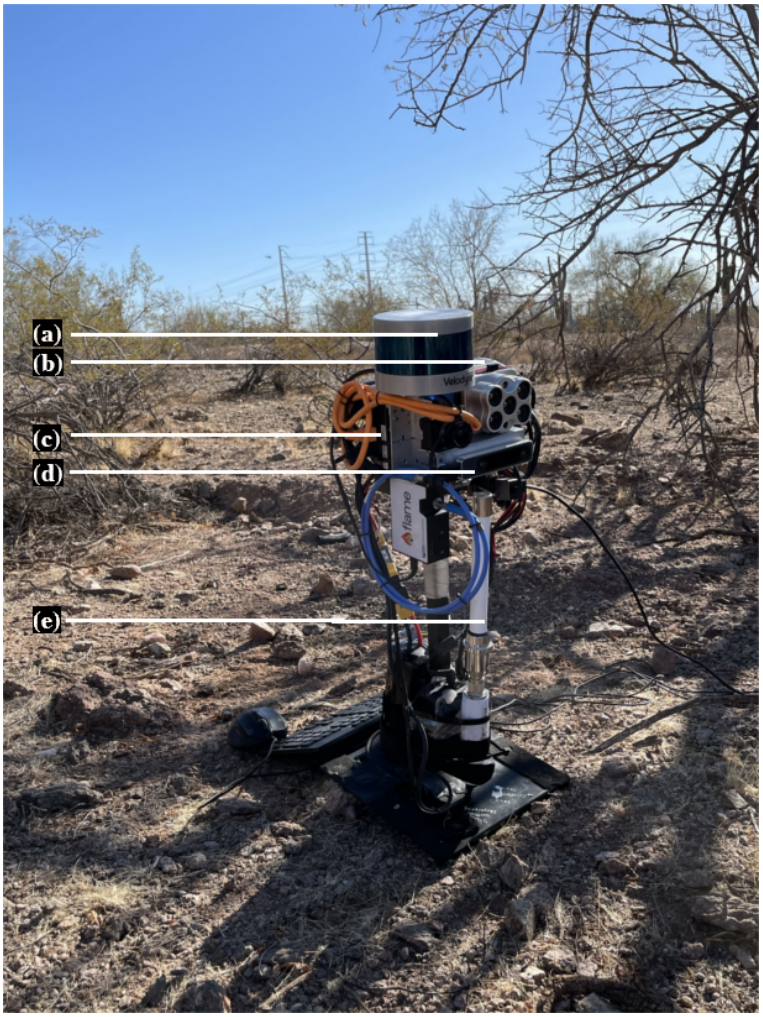

Figure 3.3: The Hardware Sensor Rig With UAV Compatible Sensors.
(a) Velodyne VLP-16 Puck 3D lidar (b) Here2 rover-side GPS antenna (c) Intel NUC i7 (d) Intel Realsense T-265 tracking stereo camera (e) Bullet M2 HP antenna. The PX4 Cube Orange and Lilliput field monitor are attached to the Intel NUC at the back of the sensor rig.

was added to the Intel NUC. For this thesis, the Intel NUC was used only to collect sensor data in the form of ROS Bags. All the core algorithmic processing described in Chapter 4 were performed by replaying recorded ROS Bags on an Alienware 15 R3 laptop with an Intel i7 processor and 16GB RAM like those on the Intel NUC of the sensor rig.

To replicate the setup of a UAV, the sensor rig was also equipped with a PX4

Cube Orange flight controller. A Here2 RTK rover-side GPS antenna was attached to the sensor rig while keeping its local base-station antenna out in the open, unblocked by tree cover to allow geo-referencing of data that was collected. A Bullet M2 omni-directional antenna was mounted on the sensor rig to perform administrative tasks like initiation and termination of data recording, accessing files and rebooting the system via remote login. A Lilliput field monitor was also connected to the Intel NUC to provide visual access to the processes running on the Ubuntu 18.04 operating system before, during and after data collection.

The Velodyne VLP-16 Puck lidar and interface box together weighed 830 g, the Intel NUC weighed 490 g, the PX4 Cube Orange weighed 60 g, the Intel Realsense T-265 Tracking camera weighed 55 g, the Here2 rover-side GPS antenna weighed 50 g and the Blue Ocean Lithium-ion Polymer (LiPo) battery weighed 1150 g. Thus, 2.6 kg was the total weight of the sensor and power supply payload that a drone would need to carry, easily allowing flights of 10–15 minute durations with modern day motors, propellers and electronic speed controllers (ESCs). In this thesis, real-world data was collected by mounting all the sensors on a handheld rig and not by flying a drone, even though weight and power requirements allowed for their use in a drone.

3.4 Real-World Data

Data for a *Carnegiea gigantea* (common name: saguaro) and *Parkinsonia* hybrid (cultivated individual) (common name: palo verde) located near Biodesign C at Arizona State University was collected by switching on the sensor rig and walking around them in independent data collection efforts. Table 3.1 summarizes their common name and corresponding GPS coordinate. In the remainder of this thesis the two plants have been referred to using their common name. Data from the saguaro served as a simpler, leafless and minimal branching test case, while the palo verde tree had

more complexity in branching and many more obstructing leaves. Figure 3.4 provides images of the two trees taken from a mobile phone and lidar scans from sensor data recorded on the handheld rig.

Circumference tape measurements for the saguaro and palo verde were also taken at various locations and the corresponding radius was noted as ground truth measurements as shown in Figures 3.5. For the left, right and central branches of the saguaro however, "diameter" measurements were taken from a point cloud using CloudCompare because the branches were above a height of 4 m and unreachable to measure manually using a tape measure. The radius values calculated from these measurements in CloudCompare were underestimates of the true radii since they were "diameter" measurements from a single-view and single scan point cloud, resulting in only partially visible branches and making the measurements obtained chords of a circle and not the true diameter.

Table 3.1: Real-World Trees Near Biodesign Building C, Arizona State University

Plant	**GPS Coordinate**
Carnegiea gigantea [Common name: saguaro]	33.4193, -111.9278
Parkinsonia hybrid (cultivated individual) [Common name: palo verde]	33.4195, -111.9273

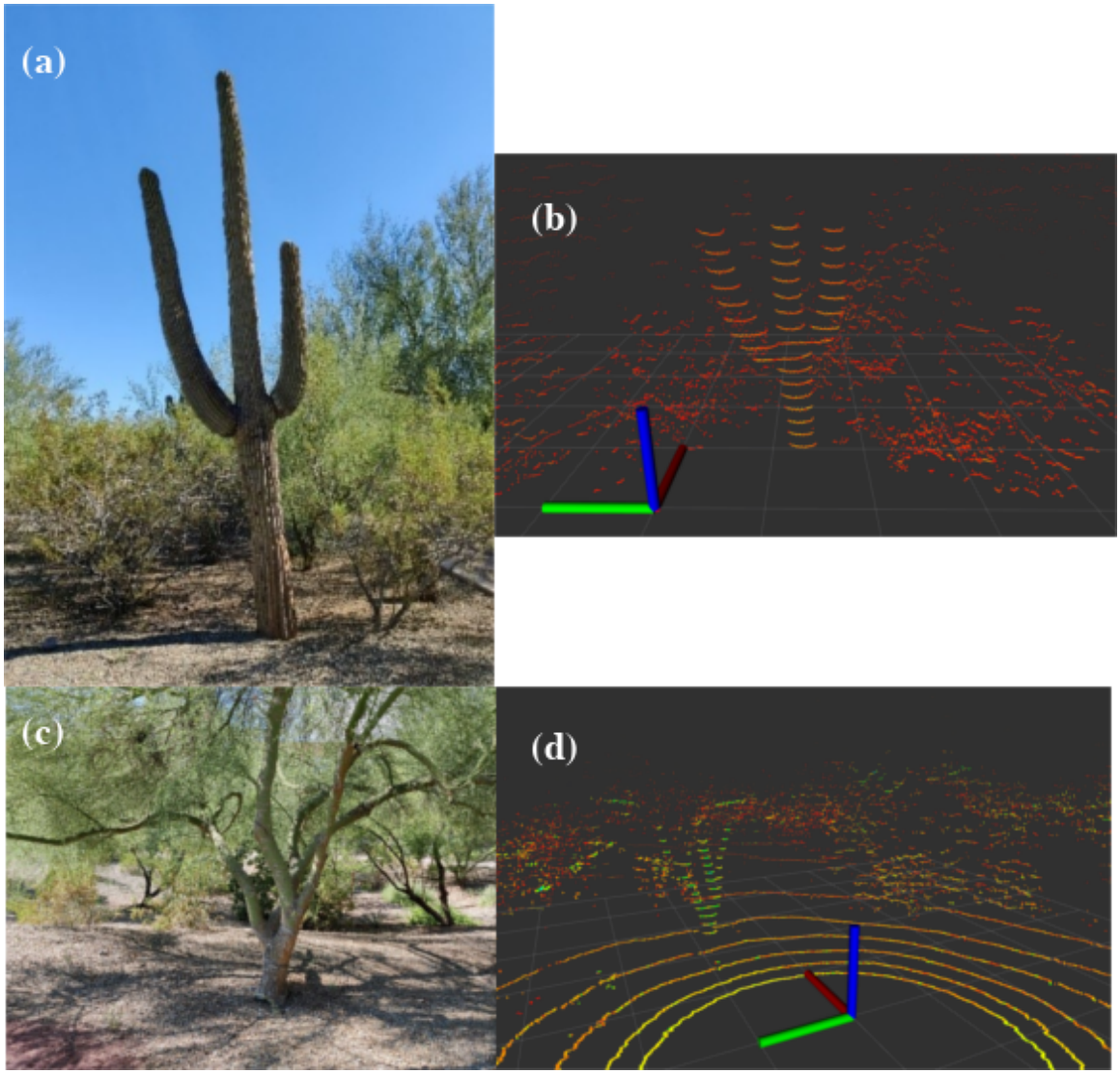

Figure 3.4: Real-World Trees and Their Lidar Scan.
(a, c) Mobile phone image of the saguaro and palo verde respectively. (b, d) Lidar scan from the handheld sensor rig of the saguaro and palo verde respectively.

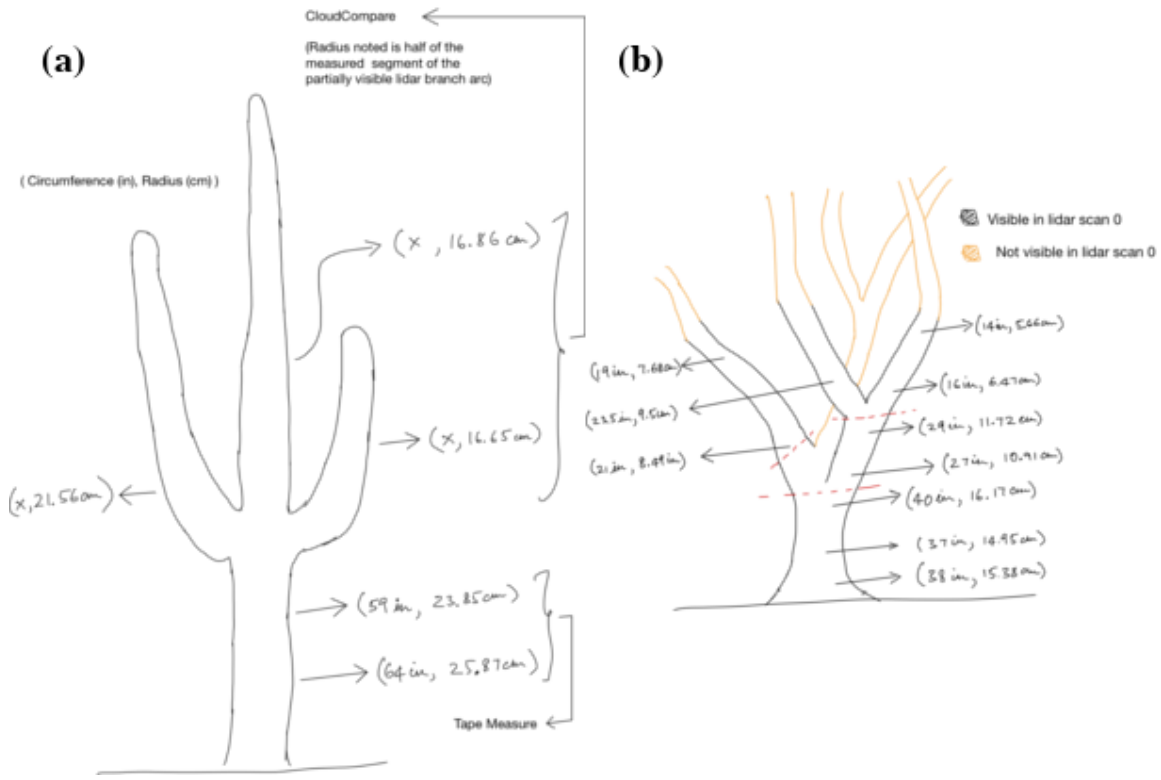

Figure 3.5: Ground Truth Tape Measurement Locations for the Real-World Trees. (a) Saguaro tape measurements (b) Palo verde tape measurements.

Chapter 4

MAPPING PIPELINE

This section describes the four main stages of the tree mapping pipeline developed in this thesis. Stages 1, 2 and 3 of the pipeline operate on 3D point cloud data or its 2D projection called a range image, while stage 4 uses the output of stage 3, 3D point cloud data and 3D pose estimates.

The first stage consists of detecting points in the lidar scan that correspond to woody and cylindrical portions of a tree like its trunk and branches using a deep learning network. The network operates on a 2D range images of corresponding 3D lidar scans. The second stage uses detected tree points from the first stage and identifies individual trunk and branch segments. Like the first stage, the second stage also functions on 2D range images. The third stage parameterizes each of the identified trunk and branch segments from the second stage into a unique 3D cylinder. The collection of the cylinders obtained in this stage constitute a portion of the final semantic tree map that is developed. The fourth stage implements a multi-sensor factor graph. The factor graph is used to integrate all the partial semantic tree maps into a completed map. It also fuses two sources of odometry and incrementally improves the semantic tree map over time. Sections 4.1 - 4.4 provide details for each of the four stages.

4.1 Stage 1: Tree Detection

A deep learning network was used to detect points in the lidar point cloud of a tree that corresponded to woody and cylindrical portions of the tree. The deep learning network needed to be fast in order to keep the overall algorithm operational

in real-time. Thus, large networks like the Pointnet [Qi et al. (2016)] and Pointnet++ [Qi et al. (2017)] which operate directly on 3D lidar scans were not suitable. Instead, a network like the RangeNet++ [Milioto et al. (2019)] which operates on 2D projections of lidar scans called range images was preferred.

RangeNet++ is a semantic segmentation network that contains encoder and decoder blocks and uses a five channel ⟨X, Y, Z, intensity, range⟩ range image as its input. A similar but faster semantic segmentation network called the ERFNet [Romera et al. (2018)] was also an attractive option to consider as mentioned in the SLOAM paper [Chen et al. (2020)]. However, the ERFNet was designed to operate on only RGB image data. Thus, a combination of the two networks using the overall architecture of ERFNet but substituting the input layer with that of RangeNet++ was used.

The output of this modified network was made to match the dimensions of the range image that was fed in at the input. So, by choosing a range image of high enough resolution, a classification for every point in the point cloud as being a woody cylindrical tree point or otherwise could be made. Also, working on range image projections of a lidar scan rather than RBG images ensured that the network was robust against varying light conditions when operating in the sub-canopy region of the tree.

4.2 Stage 2: Branch Separation

This section describes the algorithm used to separate the portions of the tree detected from the output of tree detection of Stage 1 into individual trunk and branch instances. It operates on the range image representation of a lidar scan. Figure 4.1 illustrates a range image of a tree on which this branch separation algorithm is being performed.

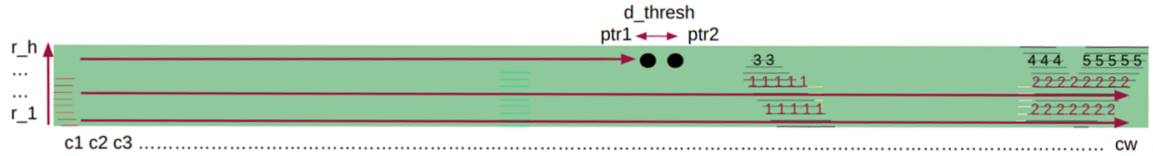

Figure 4.1: Branch Separation Algorithm on a Range Image.
$r_1, ...r_h$ and $c_1, c_2...c_w$ represent the rows and columns of the $h \times w$ range image. The black dots represent the two pointers ptr_1 and ptr_2 while the euclidean distance between their corresponding 3D points in the point cloud are shown as d_{thresh}. The red arrows indicate the directions in which the two pointers move during the course of the algorithm. It begins at the left-most column of the lowest row and proceeds to the right of each row after which it moves to the left-most column of the row above it. Maroon indices on the range image indicate locations of the range image that have been assigned an instance number from the queue while black indices indicate numbers that will be assigned from the queue on the second pass of the pointers along that row. All the points in the range image and 3D point cloud corresponding to the same index get clubbed as a single trunk or branch instance. The green background represents NaN values in the range image that have no corresponding point in the 3D point cloud.

Trees are usually structured such that the order of branching increases as we move up from the ground causing a larger number of branches and branching points to exist at higher portions of the tree. This algorithm makes use of such branching characteristics of a tree and starts at the first column of the lowest row of the range image and moves up one row at a time. A queue is maintained with branch indices that can be assigned to trunk and branch instances in a given row. The queue is empty at the beginning of the algorithm. The algorithm then counts the number of trunk and branch instances present in the first (lower most) row. Assuming row r had n trunk and branch instances, the queue is populated with a sequence of numbers from 0 to $n-1$. In a second pass along the same row, each trunk or branch instance is assigned the value present at the front of the queue and elements are popped out of the queue each time labelling of an instance is completed or a new instance needs to be labelled in that row.

Having completed the first (lowest) row, the algorithm moves onto the row above

it where it counts the number of trunk and branch instances in that row. If the same number of instances exist as the previous row, the original fully populated queue used for indexing in the first row is used for the second row as well. If there is an increase or decrease in the number of instances for a given row, then the queue is populated with a new set of indices starting from index n and going up to the number of trunk and branch instances identified in that row. Once the appropriate indices are pushed into the queue, the trunk and branch instances in that row are labelled with those indices. This process repeats till the top most row of the range image is labelled.

To identify the start and end locations of a new trunk or branch instance in a given row, the algorithm makes use of two pointers, ptr_1 and ptr_2, and a threshold distance d_{thresh}. If the 3D distance between the two pointers is greater than d_{thresh}, a new branch instance is assumed. It should be noted that not all points in the range image have a corresponding point in the 3D point cloud. ptr_1 always points to a column preceding the one pointed to by ptr_2. Algorithms 1 and 2 provide the logic described here in the form of pseudocode. If no 3D point from the point cloud gets transformed to a particular location of the range image, that location will hold an NaN value.

Algorithm 1: Index (Count) Separated Branch Segments.

Input: Pointcloud of detected trunk and branch points, distance threshold

Output: Vector of pcl::PointIndices for each identified branch instance

```
Function Main :
    Convert pointcloud to rangeimage of h × w dims
    queue ← {∅}      // queue holds available branch indices for a given row
    for r ϵ {h-1, h-2...0} do
        countInstances(r)
        if number of instances is zero then
            continue          // no corresponding 3D lidar points in this row
        queue ← indices based on number of instances in row r and row r-1
        pt1 ← NaN
        for c ϵ {0, 1, 2,...w} do
            pt2 ← rangeimage[r][c]
            if isNewInstance(pt1, pt2, distanceThreshold) then
                queue.pop()
            outputVector[queue.front()] ← queue.front()
            pt1 ← pt2
```

Algorithm 2: Identify Individual and Separable Branch Segments.

Input: Pointcloud of detected trunk and branch points, distance threshold

Output: Vector of pcl::PointIndices for each identified branch instance

```
1  Function isNewInstance(pt1, pt2, distanceThreshold):
2      d ← distance between pt1 and pt2
3      if pt1 is NaN && pt2 is NaN then
4          return false
5      else if pt1 is not NaN && pt2 is NaN then
6          return false
7      else if pt1 is NaN && pt2 is not NaN then
8          return true
9      else if pt1 is not NaN && pt2 is not NaN then
10         if d ≥ distanceThreshold then
11             return true
12         return false
13 Function countInstances(r):
14     ctr ← 0
15     pt1 ← NaN
16     for c ϵ {0, 1, 2,...w} do
17         pt2 ← rangeimage[r][c]
18         if isNewInstance(pt1, pt2, distanceThreshold) then
19             ctr ← ctr + 1
20         pt1 ← pt2
21     return ctr
```

4.3 Stage 3: Branch Parameterization

In order to use trunk and branch segments as landmarks, they need to be uniquely identifiable at any location or orientation in space. This is achieved by representing each segment as a 3D cylinder using two vector quantities and a scalar number. One vector represents the axis of the cylinder. The axis vector only encodes the orientation of the cylinder but does not represent its position in space. Thus, the second vector, a normal vector from the origin to the axis vector is utilized. To denote the thickness of the cylinder a scalar quantity of the cylinder's curvature is used. Inverse of the curvature of a cylinder provides the radius of the cross section of the cylinder. All three entities taken together form an $\mathbb{R}^7$ vector $[a^T, n^T, \kappa]$ for a uniquely identifiable cylinder in 3D space, with $a = [a_1, a_2, a_3]^T$ being the axis vector, $n = [n_1, n_2, n_3]^T$ being the normal vector and κ being the curvature of its surface. Figure 4.2 illustrates these vectors and scalar for a sample cylinder.

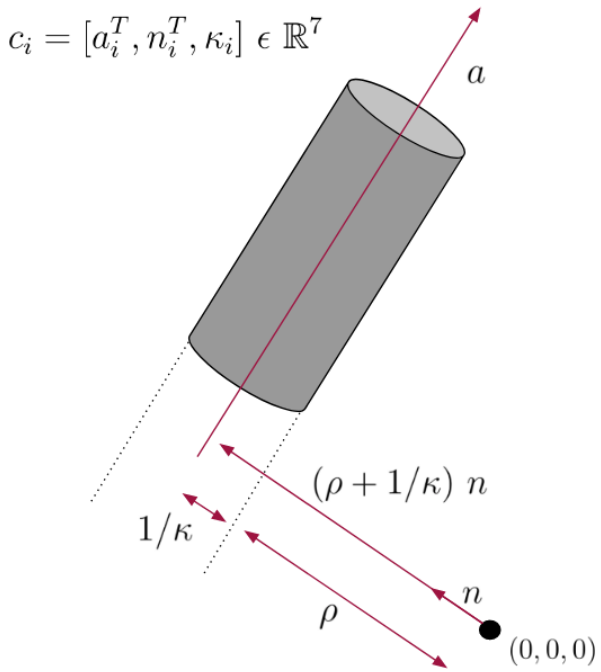

Figure 4.2: Vectors and Scalar Used to Represent a Cylinder.

Now, given a set of 3D lidar points belonging to a trunk or branch segment, the task is to parameterize those points as an $\mathbb{R}^7$ vector representing a cylinder that best fits those points onto its surface. Formulation of a non-linear optimization function to fit a cylinder to 3D points was first formalized by Marshall et al. (2001) and later also used in SLOAM [Chen et al. (2020)]. The same algorithm has been used in this thesis. As no open source implementation of it was available, the algorithm was implemented from scratch for this thesis.

Assume that the branch separation algorithm from Section 4.2 identified a total number of b trunk of branch segments. Let $P_i, i \in \{1, 2, ...b\}$ represent the point cloud of size n for segment i. For each point $p_{(i,j)}, j \in \{1, 2...n\}$ in the point cloud P_i, an error value is calculated using:

$$\begin{aligned} d_{(i,j)} =& \left| (p - \left(\rho + \frac{1}{\kappa}\right)n) \times a \right| - \frac{1}{\kappa} \\ =& \sqrt{|p - \left(\rho + \frac{1}{\kappa}\right)n|^2 - \langle p - \left(\rho + \frac{1}{\kappa}\right)n, a^2 \rangle} - \frac{1}{\kappa} \end{aligned} \tag{4.1}$$

where p is the point $p_{(i,j)}$ in point cloud P_i, ρ is the distance from the origin to the surface of the cylinder, κ is the curvature of the cylinder, a is the normalized axis vector and n is the normalized normal vector.

Figure 4.3 shows the geometric error from Equation (4.1) as a diagram. It is obtained by first finding the difference between point $p_{(i,j)}$ and the vector $(\rho + \frac{1}{\kappa})n$ which extends the normal vector n to the center of the cylinder. Then a cross product of the difference vector with the axis vector a is calculated. The resulting vector is one that exists in the radial plane of the cylinder and is proportional in length to the distance of point $p_{(i,j)}$ from the center of the cylinder. By subtracting the radius $1/\kappa$ from the magnitude of the cross product vector, a measure of point $p_{(i,j)}$'s distance from the surface of the cylinder is obtained. Minimizing such an error when summed across all the points in the point cloud P_i would ensure that the cylinder being fit

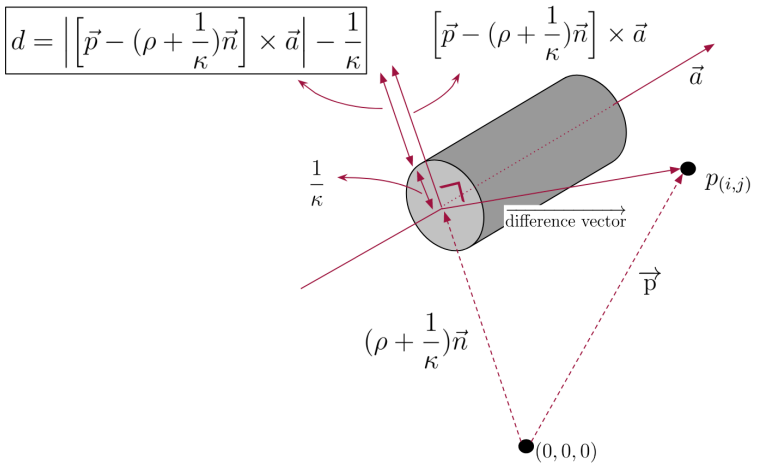

Figure 4.3: Visualization of the Geometric Error Metric That the Cylinder Parameterization Algorithm Optimizes.

contains all the points of instance i as close to the cylinder's surface as possible.

The error metric in Equation 4.1 can behave poorly for optimization problems since small κ values take the error function to infinity. Since Equation (4.1) is continuous and differentiable, based on Marshall et al. (2001), it can be rewritten as:

$$\tilde{d}_{(i,j)} = \frac{\kappa}{2}(|p|^2 - 2\rho\langle p, n\rangle - \langle p, a\rangle^2 + \rho^2) - \langle p, n\rangle \tag{4.2}$$

Thus, the cumulative error over all the points in segment i becomes:

$$d_i = \sum_{j=1}^{n} \tilde{d}_{(i,j)} \tag{4.3}$$

Equation 4.3 is the final error function that is optimized using the Levenberg–Marquardt [Moré (1978)] non-linear optimizer provided by the Ceres [Agarwal et al. (2012)] C++ library. This optimization is performed independently for each of the b trunk and branch instances. A collection of cylinders obtained in this stage from a given lidar scan represent a portion of the final semantic tree map that is constructed in the next stage (Section 4.4).

Note that the axis vector can be oriented in either direction along the cylinder but to maintain uniformity, this work represents the axis of a cylinder in the positive

Z direction when possible. If the cylinder axis lies in the XY plane, the convention used is to consider the axis vector to be oriented toward the positive Y direction. And, if the cylinder is collinear with the X axis, the cylinder axis is considered to be oriented towards the positive X direction. Additionally, the cylinder described above represents one that is infinitely long. There are no parameters used in this formula to encode the length or end points of the cylinder. These parameters of length and end points of the cylinder are extracted manually from the points in the point cloud to which this cylinder is being fit. They are not a part of the cylinder fitting non-linear optimization described here.

4.4 Stage 4: Map Integration

This section describes how partial semantic maps obtained in the previous stage are assembled into a full semantic tree map.

4.4.1 Multi-Sensor Factor Graph

To obtain the full semantic tree map the Simultaneous Localization and Mapping (SLAM) problem needs to be solved. One method by which this can be done is by using the Graph SLAM algorithm [Thrun and Montemerlo (2006)]. A software implementation of Graph SLAM in C++ is the library called GTSAM [Dellaert (2012)], [Dellaert and Kaess (2006)], [Kaess et al. (2008)], [Kaess et al. (2012)]. GTSAM formalizes the SLAM problem using a factor graphs.

A factor graph is a bipartite graph, meaning that it is a connected graph which has two disjointed sets of nodes. In the case of GTSAM, one set of nodes hold the initial estimates or optimized values of random variables, and the other set of nodes called the factors represent error metrics that need to be minimized. Factors are attached to all the random variable nodes which directly contribute to the error metric that

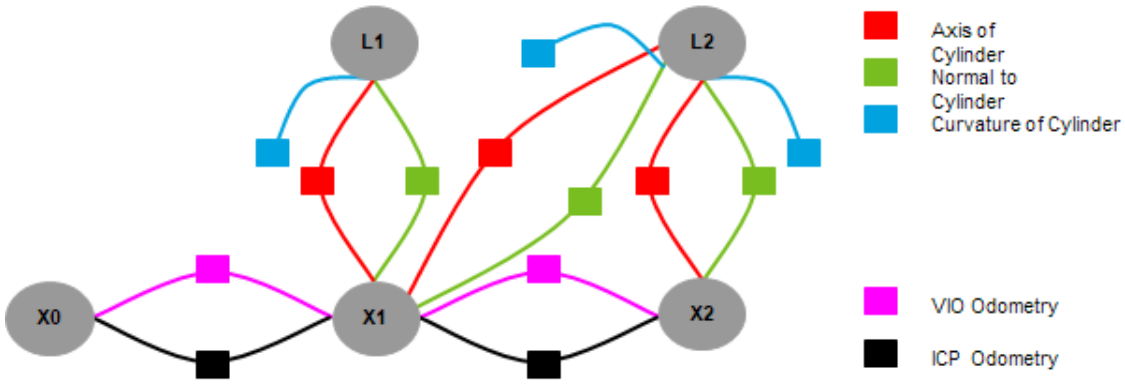

Figure 4.4: GTSAM Graph for Tree Mapping.
Circular nodes represent random variable nodes and square nodes represent factors. The nodes starting with the letter X represent the pose of the robot while nodes starting with the letter L represent landmarks that the robot encountered. The robot starts at X0 which is considered as the origin of the generated map. It then moves to X1 and subsequently to X2 during the first and second time steps respectively. With each movement, two odometry estimates are obtained. One from the Iterative Closest Point (ICP) algorithm applied to successive point clouds and another from either MAVROS in simulation or Visual Inertial Odometry (VIO) from the stereo camera of the hardware sensor rig. These motion estimates are added as odometry factors (pink and black respectively) between successive pose locations. As explained in Section 4.4.4, each $\mathbb{R}^7$ cylinder landmark is separable into three nodes and factors. Each landmark node is implemented internally as three separate nodes: the axis node, normal node and curvature node, while their corresponding factors have been displayed explicitly as the axis factor (red), the normal factor (green) and the curvature factor (blue). From X1 the robot can view landmark L1 and L2, and from X2 the robot only sees landmark L2. Thus, three landmark factors are drawn from L1 and L2 to X1, and L2 to X2. Note that the curvature of a landmark is independent of the pose of the robot and is implemented as a unary factor with no connection to a pose node.

they represent. Optimization of the graph to minimize the error metrics encoded by factors is equivalent to obtaining the marginal probability for the random variable nodes. The optimization step itself is formulated as non-linear least squares problem and solved using the Levenberg–Marquardt [Moré (1978)] algorithm.

Factor graphs are built with nodes and factors incrementally over the course of the entire SLAM process. Figure 4.4 shows how a factor graph as proposed in this thesis for semantic tree mapping can be created. As described in Figure 4.4, different nodes and factors depend on inputs from different sensors, however among the sensors used

in this thesis, each sensor had a different operating frequency. MAVROS odometry was obtained at 200Hz in simulation and VIO estimates from the Realsense camera on the hardware sensor rig were output at 50Hz. Lidar scans from the Velodyne was streamed at 10Hz in simulation and on hardware. In this thesis, the greatest common divisor (GCD) of the frequency of all the sensors was used to add nodes and factors to the graph. This meant that nodes and factors were only added when both, a new odometry value and a new lidar scan were generated. Sections 4.4.2 - 4.4.10 describe the exact implementation details of each type of node and factor used in this thesis to create a semantic map of a tree.

4.4.2 Pose Node

3D Poses added to a GTSAM graph are represented using the `gtsam::Pose3` C++ class having two main data members: `gtsam::Point3` and `gtsam::Rot3`. `Point3` represents 3D translation while `Rot3` represents 3D orientation as a 3x3 rotation matrix.

Whenever a node is added to a graph, an initial estimate for its value needs to be provided. ICP only provides the relative transformation between two poses. So, to get a global initial pose estimate from ICP, an external script had to be written to keep track of the effective transformation upto the latest time step. On the other hand, VIO output from the Realsense stereo camera would directly provide the initial global pose estimate. Out of convenience, initial estimates for pose nodes were added from VIO alone, although a noise variance weighted-average between ICP and VIO would be a preferable form of node initialization.

4.4.3 Pose Odometry Factor

An odometry factor between successive pose nodes were also added to the graph to create a pose error metric and constrain the graph optimization problem. The error metric was the difference between the current time step's pose, and the previous time step's pose transformed by an amount specified by the odometry factor.

The in-built `gtsam::BetweenFactor` class was used to implement the factor, its error metric and corresponding error jacobian. Since there were two modes of odometry measurement in this system, every time a pose node was added to the graph, two factors (or edges) corresponding to VIO and ICP measurements were added between the current and previous pose nodes.

4.4.4 Landmark Node and Factor

As explained in Section 4.3, cylindrical trunk and branch features were denoted using an $\mathbb{R}^7$ vector $c_i = [a_i^T, n_i^T, \kappa_i]$ where $a_i = [a_1, a_2, a_3]_i^T$ was the axis vector, $n_i = [n_1, n_2, n_3]_i^T$ was the normal vector and κ_i was the curvature of the i^{th} cylinder in a given lidar scan. These cylinder features from Stage 3 (Section 4.3) of the mapping pipeline were used as landmarks while creating a map of the semantic tree. Every time a lidar scan was received and a set of cylinder landmark measurements $C_b = \{c_1, c_2, ...c_b\}$ were parameterized, each cylinder $c_i \, \epsilon \, C_b$ was added as a landmark node to the GTSAM graph.

For an incoming cylinder landmark measurement from Stage 3 (Section 4.3) of the mapping pipeline, if no correspondence was found in the graph through data association, a new landmark node was added. New landmark nodes were initialized with the value of the incoming measurement. However, if a correspondence was found within the graph, the value of the corresponding landmark node was updated with the

average of its own value and the value of the incoming cylinder measurement. Ideally, a running average (using an external non-GTSAM data structure to keep track of the total number correspondences made for a landmark through data association) would need to be maintained for all landmark nodes. However, since the permitted data-association error was kept low in this thesis, taking a simple average worked well.

To constrain graph optimization with respect to landmark nodes, error metrics in the form of landmark factors were also added to the graph. The error metric represented the difference between the expected landmark measurement value and the actual landmark measurement obtained at a given robot pose. This meant that the error metric was a function of a particular pose node and one or more landmark nodes in the graph, which can be represented mathematically as:

$$e = f(L, X) \tag{4.4}$$

Landmark nodes in the graph were represented in the global reference frame to ensure that they were stored as unique entities throughout the mapping sequence, whereas incoming landmark measurements were obtained in the local reference frame. Thus, a coordinate transformation as represented in the following equation was necessary to calculate the error metric for landmark factors:

$$\begin{aligned} e &= T_g^l \; L_g - L_l \\ e &= \mathcal{R}_{7\times7} \left(L_g - t_{7\times1} \right) - L_l \\ e &= \begin{bmatrix} \mathcal{R}_{3\times3} & 0_{3\times3} & 0 \\ 0_{3\times3} & \mathcal{R}_{3\times3} & 0 \\ 0_{1\times3} & 0_{1\times3} & 1 \end{bmatrix} \left(L_g - \begin{bmatrix} 0_{1\times3} & 0_{1\times3} \\ 1_{1\times3} & 0_{1\times3} \\ 0_{1\times3} & 0_{1\times3} \end{bmatrix} X_{6\times1} \right) - L_l \end{aligned} \tag{4.5}$$

where L_g is the $\mathbb{R}^7$ cylinder landmark in global reference frame, L_l is the incoming $\mathbb{R}^7$ cylinder measurement in local reference frame, X is the 3D robot pose represented as $[x, y, z, roll, pitch, yaw]_{6\times1}$, T_g^l represents the global to local coordinate transformation

of the entire $\mathbb{R}^7$ cylinder vector, $\mathcal{R}_{7\times7}$ and $t_{7\times1}$ represent the block-wise rotation matrix and translation vector for the axis, normal and curvature portions of an $\mathcal{R}^7$ cylinder vector.

From the above equation it was seen that the first three components corresponding to the error in the axis vector were independent of the next three components corresponding to the normal vector, which in turn were independent of the last component corresponding to the curvature of the cylinder. This meant that for a single cylinder landmark, the axis vector, normal vector and curvature scalar could be implemented as three separate factors. However, a common index was maintained for each of the three types of factors in order to identify that they collectively represented the same cylinder landmark.

Implementing $\mathbb{R}^7$ landmark factors as three separate R^3, R^3 and R^1 nodes and factors allowed for the use of simpler and more common GTSAM data types like `gtsam::Pose3`, `gtsam::Point3` and `double` floating precision. This in turn simplified the calculation of the error Jacobian as well. Sub-sections 4.4.5 to 4.4.10 provide more details about this separated implementation of cylinder landmarks.

4.4.5 Landmark Axis Node

Axis nodes are represented using `gtsam::Point3` data types. The use of `gtsam::Pose3` is not required here since the node only needs to denote a collection of three numbers that represent a vector in space. There is no need to represent both translation and rotation components as is the case in Pose Nodes.

4.4.6 Landmark Axis Factor

The error metric for axis factors correspond to the first three components of equation (4.5). The axis factor does depend on the pose node to which it is associated,

however it only depends on that pose node's orientation. This is because irrespective of where in space a given cylinder is located or translated to, its orientation, represented by the axis vector, will remain the same. Thus, this factor is derived from the `gtsam::BinaryFactor` class and connects the Landmark Axis Node and a Pose Node. The mathematical formula for it is provided below:

$$e_a = T_g^l \; L_{g_a} - L_{l_a} \tag{4.6}$$

$$\begin{aligned} e_a &= \mathcal{R}_{3\times3} \left(L_{g_a} - 0_{3\times1} \right) - L_{l_a} \\ e_a &= \mathcal{R}_{3\times3} \; L_{g_a} - L_{l_a} \end{aligned} \tag{4.7}$$

where $\mathcal{R}_{3\times3}$ is the rotation matrix from global to local frame, L_{g_a} is the axis vector in the global frame in the GTSAM graph, L_{L_a} is the incoming axis measurement in the local frame and e_a is the error vector for the axis. The jacobian for the error vector was calculated using the `gtsam::OptionalJacobian` parameters of `gtsam::Pose3::transformTo()` and using a zero-translation version of the associated pose node variable.

4.4.7 Landmark Normal Node

Normal vector nodes are also represented using `gtsam::Point3` data types. The use of `gtsam::Pose3` is not required here either, since the node only needs to denote a collection of three numbers that represent a vector in space.

4.4.8 Landmark Normal Factor

The error metric for normal-vector factors correspond to the second three components of equation (4.5). The normal-vector factor depends on both, the translation and orientation of the pose to which it is associated. This is because the normal vector is what anchors the location of a cylinder in 3D space. When the position or orientation

of the lidar changes, a corresponding transformation needs to be performed to shift and rotate the cylinder frame of reference to the correct location and orientaiton in space. Thus, it is derived from the `gtsam::BinaryFactor` class and connects the Landmark Normal Node and a Pose Node. The mathematical formula for it is provided below:

$$e_n = T_g^l \; L_{g_n} - L_{ln} \tag{4.8}$$

$$e_n = \mathcal{R}_{3\times3} \; \left(L_{g_n} - x_{3\times1}\right) - L_{ln} \tag{4.9}$$

where $\mathcal{R}_{3\times3}$ is the rotation matrix from global to local frame, L_{g_n} is the normal vector in the global frame in the GTSAM graph, L_{L_n} is the incoming normal vector measurement in the local frame, $x_{3\times1}$ is the robot pose and e_n is the error vector for the normal. The jacobian for the error vector was calculated using the `gtsam::OptionalJacobian` parameters of `gtsam::Pose3::transformTo()` from the associated pose node.

4.4.9 Landmark Curvature Node

The curvature node is the simplest of the landmark nodes. It is represented with a scalar `double` floating precision number.

4.4.10 Landmark Curvature Factor

The curvature factor is also very simple to implement. It is not dependent on the pose, thus, it is just a `gtsam::UnaryFactor` connected to the Landmark Curvature Node alone. The error metric for it can be written mathematically as given below:

$$e_\kappa = T_g^l \; L_{g_\kappa} - L_{l\kappa} \tag{4.10}$$

$$e_\kappa = L_{g_\kappa} - L_{l\kappa} \tag{4.11}$$

where L_{g_k} is the curvature landmark factor in the GTSAM graph, L_{L_k} is the incoming curvature measurement in the local frame and e_k is the error in curvature.

Note that the curvature does not change when a coordinate transformation to or from the global and local frame is performed. So, in this case the curvature in the "local frame" L_{L_k} simply represents the measurement received at the current time step from the cylinder parameterization algorithm. The curvature in the "global frame" L_{g_k} simply represents the curvature node in the GTSAM graph to which the curvature measurement was data associated. In this case, the jacobian of the error with respect to landmarks is simply unity.

Chapter 5

RESULTS

First, results of a preliminary test that was performed to map a tree using an existing state-of-the-art mapping algorithm called LeGO-LOAM [Shan and Englot (2018)] has been described in Section 5.1. Then, results from the fours stages of the tree mapping pipeline on simulation and real-world data have been provided in Sections 5.2 and 5.3 respectively.

5.1 Inadequacy of Current Techniques

After proposing the semantic tree mapping problem statement for this thesis, the first step was to check if existing lidar mapping methods like LOAM could already map trees. A time optimized version of LOAM called LeGO-LOAM was used for these tests by feeding it Velodyne point cloud and IMU odometry data from the ROS Bag recorded in simulation.

LeGO-LOAM was unable to construct any meaningful map. The tree map generated was highly skewed, had many drifted points and did not keep track of the drone's pose as shown in Figure 5.1 (a). Another test was run by placing cuboidal pillars around the tree such that at any given point in time, at least three of the four cuboidal pillars would be visible to the UAV. The purpose of placing the pillars was to provide LeGO-LOAM with a set of sharp edge features and flat surfaces to use for mapping. The pillars were designed to be 3 meters shorter than the tree. In this test, LeGO-LOAM was able to map the tree only until the pillars were present in the lidar scan. Once the drone flew above the height of the pillars and lost visibility of them, tree mapping became distorted. Figures 5.1 (b) and 5.1 (c) show the simulation

environment with the cuboidal pillars and the inadequacy of LeGO-LOAM to map the tree above the height of the pillars.

These tests confirmed that LeGO-LOAM could only map an environment that had sharp edge features and flat planes. Such features are common in urban settings but rare in forests and agricultural lands. Thus, it was confirmed that there was a need for different mapping algorithms for trees, and further work on this thesis was continued.

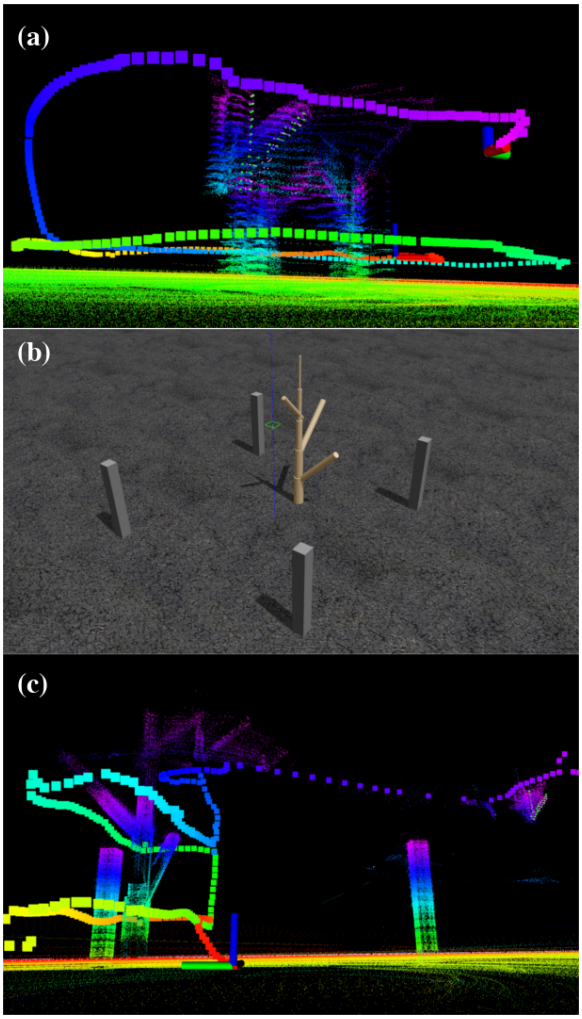

Figure 5.1: Inadequacy of LeGO-LOAM.
(a) Drifted UAV poses and a skewed map of the simulation tree produced by LeGO-LOAM when mapping without cuboidal pillars. (b) Simulation environment with the addition of four cuboidal pillars around the tree. (c) Successful pose estimation and mapping by LeGO-LOAM only up-to the height of the pillars after which drifted UAV poses and a skewed tree map were generated.

5.2 Simulation Results

Results from the four stages of the tree mapping pipeline applied on continuous real-time simulation data have been shown in Figure 5.2 and more details about each stage have been provided in the following paragraphs. Although no specific execution time calculations were performed for each stage of the mapping pipeline, its real-time capability was validated by noting that there was minimal or no packet buffer or loss when the mapping pipeline was operated at 10 Hz (the GCD of the operating frequency of all the sensors as mentioned in 4.4.1).

Stage 1 - Tree Detection: The deep learning network described in Section 4.1 was trained on 396 range images of lidar scans obtained by sampling the simulation ROS Bag at 1Hz. The input range images and output segmented mask images were given a height and width of 32×360. *Intensity* and *Range* channels were not used at the input layer since the entire tree in simulation was leafless and made up of the same material, thus yielding the same intensity value. The training process involved two steps. The first step was to train the encoder portion of the network after which the encoder weights were frozen and the decoder was trained. For the encoder training step, the dimensions of the output mask images was 4×180 (ie. one-eighth the height of the decoder's output layer and a proportional adjustment in width). A soft-max activation function followed by a 2D cross-entropy loss function was used to compute the error between the network's output and the expected mask images. The network yielded an accuracy of 96% when trained for 7 epochs with a batch size of 3 images. Figure 5.2 (a) shows the segmentation result for the 190^{th} lidar scan.

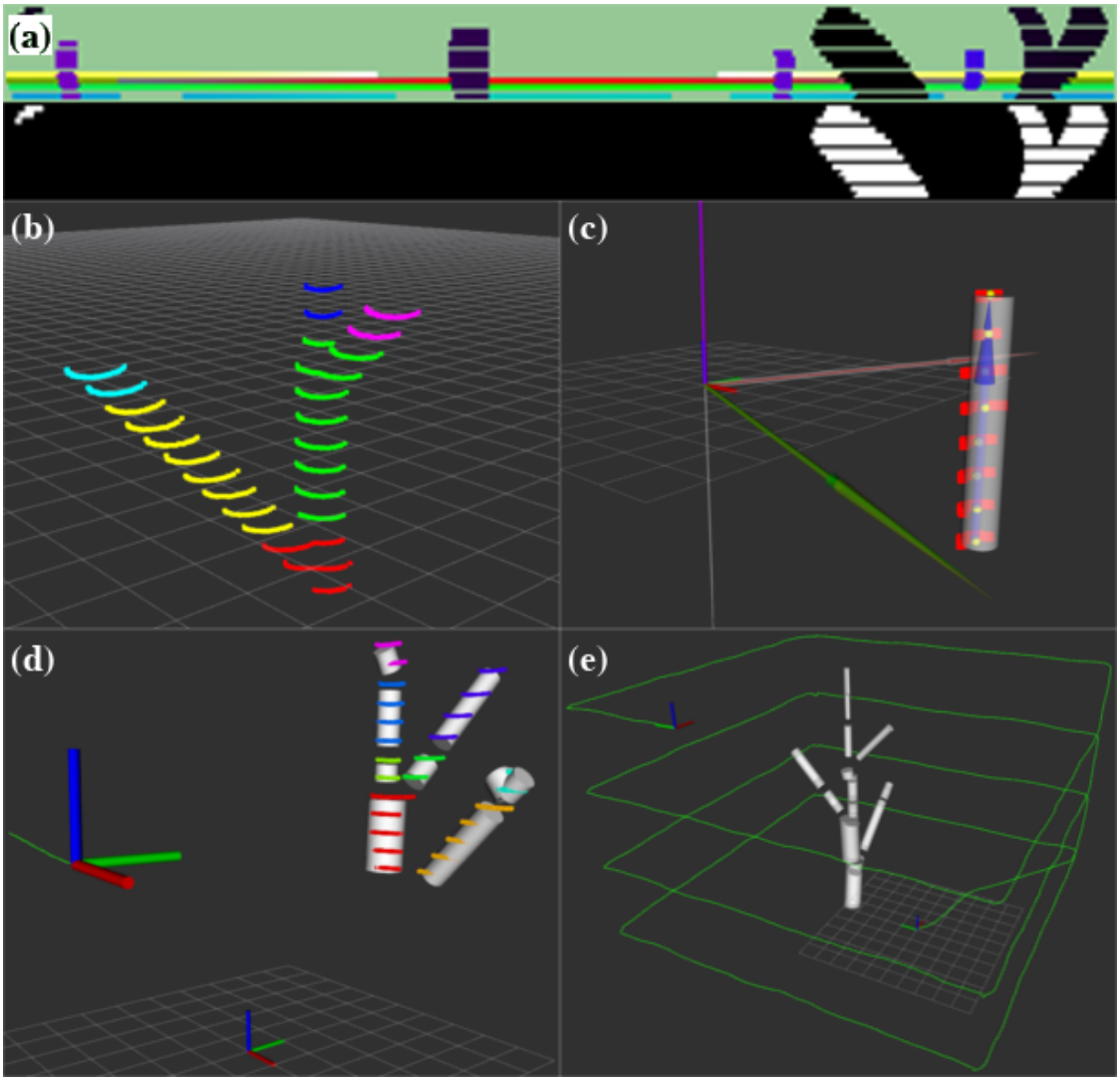

Figure 5.2: Results of Mapping Pipeline on Simulation Data.
(a) Stage 1: Semantic segmentation of a lidar range image to detect tree points in the lidar cloud. White pixels in the binary output image represent the detected tree. (b) Stage 2: Separation and indexing of different branch segments for a particular lidar scan of the tree. Different colors represent different branch segments. (c) Stage 3: Cylinder parameterization performed on lidar points of a single branch segment. Bright red patches represent points from the lidar point cloud, the white cylinder represents the optimized cylinder parameterized for the point cloud, the yellow spheres are an initial estimate of the center of each lidar ring for the given branch segment, the blue and magenta vectors represent the initial and final (optimized) estimates of the axis of the cylinder respectively and the light yellow (overlapped by green vector) and green vectors represent the initial and final estimates of the normal vector to the cylinder respectively. The thin white vectors are a visualizations of the orthogonal basis vectors created to enable unconstrained parameterization of the non-linear cylinder optimization problem from Equation 4.3. (d) Stage 3: A collection of cylinders which constitute the partial un-optimized semantic tree map from a single lidar scan of the tree. (e) Stage 4: The full optimized semantic tree map and UAV path output by the factor graph.

Stage 2 - Branch Separation: The points of the point cloud that were segmented out in Stage 1 were fed to this stage for trunk and branch separation. For this stage to perform reliably over the entire simulation ROS Bag, the pairwise distance threshold $d_{thresh} = 0.28$ m was used. Figure 5.2 (b) shows a color coded lidar scan representing different branch segments that were sectioned out by the branch separation algorithm.

Stage 3 - Branch Parameterization: Lidar scans from 10 different points in the simulation ROS Bag were selected and from those scans, 24 trunk and branch segments were manually separated using Stage 2 of the mapping pipeline. These segments had different positions, orientations and thicknesses. It must be noted that tree point clouds in simulation had much lesser irregularity than the real world since the tree surface was a smooth cylinder in Gazebo. It was ensured however, that the noise modeled into the rays of the lidar sensor in simulation was the same as the noise of ±3cm reported on the Velodyne VLP-16 Puck's datasheet for typical real world working conditions. The Branch Parameterization algorithm when applied to the 24 segments resulted in cylinders that had a radius error of less than 2%. Figure 5.2 (c) shows an example of how a cylinder was fit to a particular branch segment. Figure 5.2 (d) shows an accumulation of cylinders fit to all the branches visible in a particular lidar scan. This set of cylinders represented a portion of the entire semantic tree map being built.

Stage 4 - Map Integration: This stage fused odometry from two different sources and also combined the parameterized trunk and branch cylinders from Stage 3 to generate a full semantic tree map. iSAM2 [Kaess et al. (2012)] was chosen as the underlying type of graph due its ability to optimize incrementally. The graph was updated with odometry nodes and factors from MAVROS and ICP applied on detected trunk and branch points of successive lidar scans. Landmark nodes and factors corresponding to the axis, normal and curvature of the parameterized branches were

also added to the graph. Figure 5.2 (e) shows the UAV's pose and the full semantic tree map after graph optimization (ie. marginalization of all the graph nodes).

To obtain a better semantic tree map, additional tuning for various GTSAM parameters is required. Parameters like the noise variance for the pose and landmark factors, weighted landmark data association equality thresholds for the three types of landmarks, iSAM2 graph relinearization threshold and iSAM2 graph relinearization-skip-steps affect the resulting semantic tree map. Tuning of these parameters is dictated primarily be the stringency with which data association is implemented on the trunk and branch landmarks. The stringency of data association itself is dictated by the noise in robot pose, branch thickness and branch density for a given tree. Additionally, changing the UAV flight path from quadrangles at discrete altitudes to a circular or conical path around the tree and having a more gradual continuous altitude increment would aid in filling out some of the gaps seen in the map.

The results in this section demonstrate that the tree mapping pipeline is able to extract semantic information from a tree to encode its topology from continuously streamed real-time data in the simulation environment.

5.3 Real-World Results

Tables 5.1, 5.2 and Figure 5.3 show results that were obtained on real-world data from branch parameterization. The radius calculated for the base of the saguaro lay within the minimum and maximum range of the ground truth measurement. The radii calculated for the left, right and center branches of the saguaro were a few centimeters higher than their half-chord ground truth measurements, as per the explanation in Section 3.4. In the case of the palo verde, radius values calculated using branch parameterization had a mean error of $+/-0.4$ cm and a maximum error of $+0.84$ cm.

These real-world results show that semantic information being obtained from lidar

scans lie close to the "...current forestry standard of having an error within a quarter inch (0.65 cm) in DBH" as described by Stevn Chen, CEO and co-founder of a drone-based autonomous forest inventory company called TreeSwift. Visual confirmation that position and orientation estimates were correct for cylinders fit on trunk and branch segments of tree point clouds was also obtained.

The results provided in this section were obtained by manually sectioning out trunk and branch segments from raw real-world point cloud data. In order for the full tree mapping pipeline to work in real-time on real-world data and yield more accurate results (in a similar manner to its functioning on simulation data), the following three stages need to be addressed in future work: Stage 1 Tree Detection requires additional training of the neural network using labelled real-world lidar scans and the use of *Intensity* and *Range* channels at the input layer; Stage 2 Branch Separation requires the implementation of an adaptive d_{thresh} parameter to reliably section out trunk and branch segments for varying distances of the hardware sensor rig from the tree being mapped; Stage 4 Multi-Sensor Factor Graph needs to be fully tuned as described in Section 5.2.

Table 5.1: Estimated Radius (cm) of Different Segments of Saguaro.

Segment Name	**Tape Measure (cm)**	**Branch Parameterization (cm)**
Base	23.8 - 25.8 (radius)	25.99
Left	21.5 (half-chord)	23.73
Center	16.8 (half-chord)	19.25
Right	16.6 (half-chord)	23.91

Table 5.2: Estimated Radius (cm) of Different Segments of Palo Verde.

Segment Name	Tape Measure (cm)	Branch Parameterization (cm)
Base	14.95 - 16.17	14.51
Left	7.68 - 8.49	8.79
Right	10.91 - 11.72	12.56
RightLeft	9.5	9.07
RightRight	5.66 - 6.47	5.24

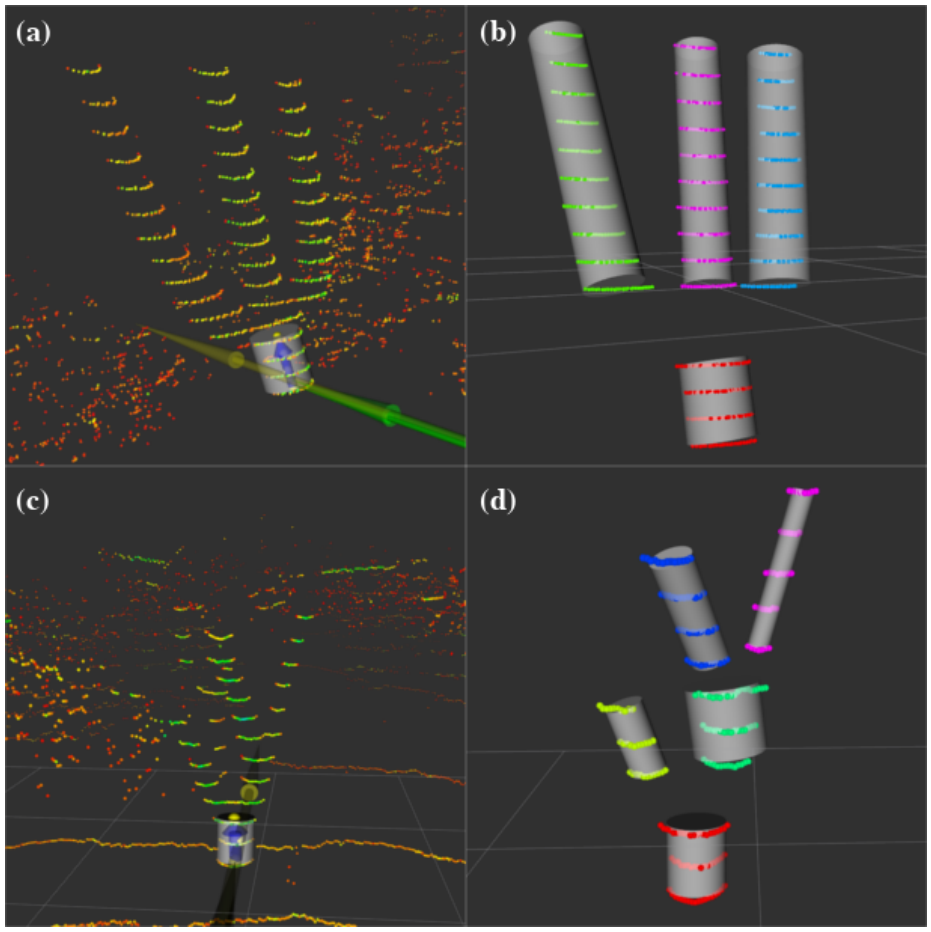

Figure 5.3: Results of Mapping Pipeline on Real-World Data.
(a, c) Cylinders parameterized for the trunk of the saguaro and palo verde overlaid on the full lidar point cloud respectively. (b, d) Collection of cylinders parameterized for different trunk and branch segments of the saguaro and palo verde as provided in Tables 5.1 and 5.2 respectively.

Chapter 6

CONCLUSION

This thesis presented a semantic tree mapping pipeline that created a topological representation of a tree. A simulation environment representing a simplified tree mapping scenario was developed. A hardware sensor rig to collect pose estimates and lidar tree data in the real world was also assembled and used. A deep learning network to identify prominent woody portions of a tree was trained and an algorithm to separate detected portions of a tree point cloud into trunk and branch segments was also proposed. Each segment was then parameterized into a cylinder such that a collection of cylinders would represent a portion of the semantic tree map being generated. A multi-sensor factor graph was then formulated to assemble partial semantic maps into the full map while incrementally refining the map and also accounting for two forms of odometry. It was shown that a sequence of these stages could estimate position, orientation and thickness of trunk and branch segments, making it applicable for both forestry and precision agriculture applications. The simulation environment, hardware sensor rig and tree mapping software pipeline together resulted in a real-time capable end-to-end data recording and mapping framework for trees.

Chapter 7

LIMITATIONS AND FUTURE WORK

This chapter describes limitations pertaining to this thesis and future work to extend its capabilities.

7.1 Limitations

Four main limitations were identified. The first is that accurate odometry estimates can not be obtained when moving in parallel to the axis of a cylinder. This is because every part of a uniform cylinder produces the same landmark feature measurement irrespective of where along its length lidar data is collected. Figure 7.1 (a) illustrates how two different poses X1 and X2 see the same features from the cylinder despite being at two different heights.

The next limitation is related to the R^7 parameterization of cylinders having infinite length. The mathematical formulation of a cylinder as a vector does not take into account how long the cylinder is. Therefore, if the axis of two branches are collinear, then they can throw off the system's odometry and localization estimate. Figure 7.1 (b) illustrates how a branch mix-up can lead to odometry offsets. The tree is shown to have two unique, independent and unconnected branches (in yellow) having the same radius and aligned collinearly on the left and right side of the tree. When data is collected from the left side of the tree the yellow branch on the left is measured. However, when data is collected from the right side of the tree and the yellow branch on the right side is measured, the same landmark measurements as the branch from the left side would be obtained leading to an offset in pose estimates and in-turn an offset in the map being generated.

A point to note about both limitations mentioned above is that, trees are usually structured with an upward and outward branching structure. So, by the very nature of tree shapes, even if a couple of branches fall under the limiting conditions mentioned above, most of the other branches in high likeliness would still be uniquely representable. This would allow pose estimates and landmark updates being performed by the graph slam algorithm to stay on-course but result in a small drop in confidence of its estimates.

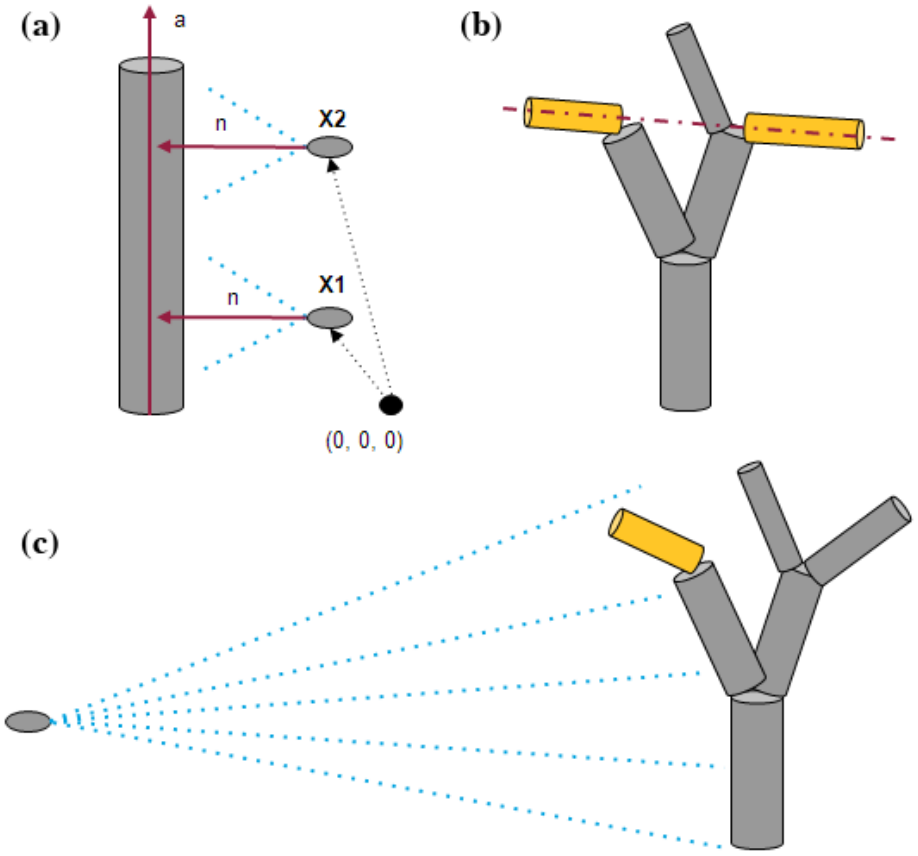

Figure 7.1: Limitations of Tree Mapping Pipeline.
(a) Identical features of a cylinder seen from two different poses X1 and X2. (b) Two different branch segments having the same cylinder parameterization due to collinearity. (c) Dispersal of lidar points leading to lack of lidar points from smaller branches when mapping from too far a distance.

A third limitation arises from the divergent nature of lidar rays. The further away

the rays of a lidar travel, the more they get separated from each other. This is because they are emitted at different diverging angles and continue to travel in a straight line. If mapping is performed on trees from a far of distance, the rays will diverge so much so, that they may miss entire branches all together as shown in Figure 7.1 (c). However, if the lidar is too close to a tree, it would only be able to view a single branch which would reduce the number of landmark associations made per scan and/or lead to the first limitation of moving along the axis of a single visible cylinder. Both of these conditions could result in diminished odometry and mapping outputs. Also, the ideal distance with which mapping should be performed can vary based on the species of the tree being mapped, the thickness and shape of its branches, the resolution of lidar being used and precision with which mapping needs to be performed.

The final limitation of this thesis is that it does not account for bulged, flattened and tapering tree trunk or branches. Instead, it assumes that the entire unit of a branch or trunk is of uniform thickness. This means that when the same branch is viewed from different angles or different heights, is can appear to be slightly different each time (especially in its curvature measurement). Two techniques can be considered to mitigate this issue. The first, more preferable technique, would be to enhance the manner in which branches are modeled. Using new shapes like sections of a cone or other tapered cylindrical shapes would allow for more accurate feature detection and more robust feature association. The other method would be to weight the curvature component a little more loosely during data association. This would allow for measurements of the same branch to still be matched with each other and not discarded simply because of slightly different curvature values when measured at or from different locations.

7.2 Future Work

This thesis has many avenues into which it can be expanded. The branch instance detection algorithm depends on a distance threshold between a pair of points to identify different branches from a range image. This distance threshold was kept fixed in this work, since mapping was performed at approximately a uniform distance for each tree, leading to a uniformly dispersed lidar scan. Adaptive ways of inferring this distance threshold need to be implemented, so that if the distance from which a tree is being mapped changes in the middle of the mapping process, branch instance identification would still function well.

Another point of improvement would be to have additional modes of odometry estimates. Here only VIO and ICP were used for odometry. An extra, outer layer of optimization can be performed as SLOAM proposed, and odometry from that could be added into the GTSAM graph as well.

This thesis focused on tree features that were woody and cylindrical like the trunk and branches. Characterization of the crown of a tree would be of great benefit to the ecology and robotics communities, as it could be used to simulate shade patterns; estimate leaf, twig and floral litter; predict photosynthesis and transpiration properties of a tree; and also provide a new set of features for localization and autonomous navigation at the upper regions of a forest. Alpha shapes and sets of surface normals could be one way to represent the crown of individual trees and the canopy of forests.

Finally, like most autonomous software, the pipeline proposed here needs to be tested against many more trees of the same and different species to identify additional pitfalls, develop fixes and make the system more reliable in the long run.

REFERENCES

Agarwal, S., K. Mierle and Others, "Ceres solver", http://ceres-solver.org (2012).

Chen, S. W., G. V. Nardari, E. S. Lee, C. Qu, X. Liu, R. A. F. Romero and V. Kumar, "Sloam: Semantic lidar odometry and mapping for forest inventory", IEEE Robotics and Automation Letters **5**, 2, 612–619 (2020).

Dassot, M., T. Constant and M. Fournier, "The use of terrestrial lidar technology in forest science: application fields, benefits and challenges", Annals of Forest Science **68**, 5, 959–974, URL https://doi.org/10.1007/s13595-011-0102-2 (2011).

Dellaert, F., "Factor graphs and gtsam: A hands-on introduction", Tech. rep., Georgia Institute of Technology (2012).

Dellaert, F. and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing", The International Journal of Robotics Research **25**, 12, 1181–1203, URL https://doi.org/10.1177/0278364906072768 (2006).

Franklin, S., Remote Sensing for Sustainable Forest Management (CRC Press, 2001), 1 edn., URL https://doi.org/10.1201/9781420032857.

Fritz, A., T. Kattenborn and B. Koch, "Uav-based photogrammetric point clouds-tree stem mapping in open stands in comparison to terrestrial laser scanner point clouds", vol. XL-1/W2 (2013).

Kaess, M., H. Johannsson, R. Roberts, V. Ila, J. J. Leonard and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree", The International Journal of Robotics Research **31**, 2, 216–235, URL https://doi.org/10.1177/0278364911430419 (2012).

Kaess, M., A. Ranganathan and F. Dellaert, "isam: Incremental smoothing and mapping", IEEE Transactions on Robotics **24**, 6, 1365–1378 (2008).

Kim, G. and A. Kim, "Scan context: Egocentric spatial descriptor for place recognition within 3D point cloud map", in "Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems", (Madrid, 2018).

Marshall, D., G. Lukacs and R. Martin, "Robust segmentation of primitives from range data in the presence of geometric degeneracy", IEEE Transactions on Pattern Analysis and Machine Intelligence **23**, 3, 304–314 (2001).

Milioto, A., I. Vizzo, J. Behley and C. Stachniss, "RangeNet++: Fast and Accurate LiDAR Semantic Segmentation", in "IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)", (2019).

Moré, J. J., "The levenberg-marquardt algorithm: Implementation and theory", in "Numerical Analysis", edited by G. A. Watson, pp. 105–116 (Springer Berlin Heidelberg, Berlin, Heidelberg, 1978).

Q, J. A. G., R. Hernandez and A. Sanchez-Azofeifa, rTLS: Tools to Process Point Clouds Derived from Terrestrial Laser Scanning, URL https://CRAN.R-project.org/package=rTLS, r pacakge version 0.2.3.1 (2021).

Qi, C. R., H. Su, K. Mo and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation", CoRR **abs/1612.00593**, URL http://arxiv.org/abs/1612.00593 (2016).

Qi, C. R., L. Yi, H. Su and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space", CoRR **abs/1706.02413**, URL http://arxiv.org/abs/1706.02413 (2017).

Raumonen, P., M. Kaasalainen, M. Åkerblom, S. Kaasalainen, H. Kaartinen, M. Vastaranta, M. Holopainen, M. Disney and P. Lewis, "Fast automatic precision tree models from terrestrial laser scanner data", Remote Sensing **5**, 2, 491–520, URL https://www.mdpi.com/2072-4292/5/2/491 (2013).

Romera, E., J. M. Álvarez, L. M. Bergasa and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation", IEEE Transactions on Intelligent Transportation Systems **19**, 1, 263–272 (2018).

Shan, T. and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain", in "IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 4758–4765 (IEEE, 2018).

Shan, T., B. Englot, D. Meyers, W. Wang, C. Ratti and R. Daniela, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping", in "IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", pp. 5135–5142 (IEEE, 2020).

Thrun, S. and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures", The International Journal of Robotics Research **25**, 5-6, 403–429, URL https://doi.org/10.1177/0278364906065387 (2006).

Trochta, J., M. Krůček, T. Vrška and K. Král, "3d forest: An application for descriptions of three-dimensional forest structures using terrestrial lidar", PLOS ONE **12**, 5, 1–17, URL https://doi.org/10.1371/journal.pone.0176871 (2017).

Wallace, L., A. Lucieer, Z. Malenovský, D. Turner and P. Vopěnka, "Assessment of forest structure using two uav techniques: A comparison of airborne laser scanning and structure from motion (sfm) point clouds", Forests **7**, 3, URL https://www.mdpi.com/1999-4907/7/3/62 (2016).

Wallace, L., A. Lucieer, C. Watson and D. Turner, "Development of a uav-lidar system with application to forest inventory", Remote Sensing **4**, 6, 1519–1543, URL https://www.mdpi.com/2072-4292/4/6/1519 (2012).

Wulder, M. A., C. W. Bater, N. C. Coops, T. Hilker and J. C. White, "The role of lidar in sustainable forest management", The Forestry Chronicle **84**, 6, 807–826, URL https://doi.org/10.5558/tfc84807-6 (2008).

Zhang, J. and S. Singh, "Low-drift and real-time lidar odometry and mapping", Autonomous Robots **41**, 2, 401–416, URL https://doi.org/10.1007/s10514-016-9548-2 (2017).

BIOGRAPHICAL SKETCH

Rakshith Vishwanatha is a Master of Science student at Arizona State University in the Robotics and Autonomous Systems department. Here he has explored various sub-domains of robotics like dynamics and control; localization and mapping; and computer vision while taking up relevant courses and gaining hands-on experience at the Robotics and Intelligent SystEms (RISE) Laboratory and Distributed Robotic Exploration and Mapping Systems (DREAMS) Laboratory. Before attending graduate school he received a bachelor's degree in Electronics and Communication and worked professionally as a Computer Network Engineer for two years. Piecing together science concepts through engineering applications has always been a core interest of Rakshith's, however, he also has a genuine appreciation for animal and nature ecosystems. Connecting with the elements through multi-day hikes, rock climbing, biking and bird photography are some ways he enjoys spending his free time. All in all, working on a thesis consisting of a mix of technical concepts in mathematics, software development and engineering with aspects of ecology and nature has been extremely satisfying for him, and he looks forward to building upon all the experience gained in the process.