Mobile Phone Depth Sensors for Forest Carbon Measurement

by

Amelia Holcomb

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The monitoring, reporting, and verification (MRV) of forest plots, especially their tree-trunk diameters, is critical to achieving both forest protection and reforestation goals. Today's MRV processes are mostly manual, error-prone, and costly to carry out. In this thesis, we design and implement an app running on a smartphone equipped with a time-of-flight sensor that allows efficient, low-cost, and accurate measurement of tree trunk diameters. The core focus is on designing an algorithm to identify, segment, and compute the diameter of a target tree trunk in a depth image of a forest scene, even in the face of natural leaf and branch occlusion. The algorithm runs in real-time on the phone, allowing user interaction to improve the quality of the results. We evaluate the app in realistic settings and find that in a corpus of 55 sample tree images, it estimates trunk diameter with mean error of 7.8%. We also explore a newly released alternative to the time-of-flight sensor, Google's ARCore Depth API, which uses a depth-from-motion algorithm based on a monocular phone camera and accelerometer sensors. We conclude that this API is currently inadequate for the proposed application and offer suggestions for its improvement.

Acknowledgements

First and foremost, I would like to thank my supervisor, Srinivasan Keshav, for his support, feedback, thoughtful questions, and encouragement. I am so lucky to have him as a supervisor – the care and dedication he shows towards his students simply goes above and beyond. I grew so much as an academic, computer scientist, and environmentalist as a result of working with him, and I can't wait to continue on with him to pursue a PhD.

I would also like to thank my co-supervisor, Tim Brecht, for supporting me throughout my time at the University of Waterloo and for his extensive comments on the draft of this thesis. His attentive eye and probing questions pushed me to improve my technical writing and prompted me to think more carefully about every aspect of this work.

The idea for this thesis was conceived through a course project with Connor Tannahill, with whom I took my first foray into the ill-documented world of Android programming for ToF sensors. Here's to all the samosas we ate while hacking away on Jupyter notebooks and the Camera2 API. I would also like to give a huge thank you to the undergraduate research assistants who worked on this project. Bill Tong managed to excise everything to do with the Camera2 API and replace it with AREngine, and he provided invaluable assistance on the first version of depth image segmentation. Bill also collected the summertime evaluation dataset. Megan Penny collected the data on CamToPlan and Google Measure.

Finally, I would like to thank Mom, Dad, Sophie, Ida, and Evan for their love and support through everything. And to my Quaranteam and the Crossword Club: you kept me sane.

Dedication

For Bartholomew.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview and Motivation

Carbon sequestration in trees can play a key role in decarbonizing the atmosphere and averting catastrophic climate change. In its 2018 report, the Intergovernmental Panel on Climate Change (IPCC) highlights that all pathways to limit global warming to 1.5°C rely on active removal of atmospheric carbon dioxide, with most published plans incorporating forest carbon sequestration as a key component of this removal [36]. However, strategies to achieve reforestation and anti-deforestation goals, whether through policy- or market-based incentives, face a technological challenge: they require monitoring, reporting, and verification (MRV) of forest plots *in situ* to measure the actual degree of sequestration achieved [24][17][29].

Current forest MRV is based on a standardized, manual inventory process that involves mapping out sample plots with ribbon or rope, then using tape measures or calipers to find the diameter of each tree trunk in the plot [31]. This labor-intensive process has four negative consequences. First, it limits *sample size*; thus only a tiny fraction of forested land has been sampled. Second, it limits the *number of data points* that can be collected per tree. Third, manual measurements are challenging to carry out in *dense, diverse forests* such as the tropics, where significant undergrowth may surround trunks. Finally, it places an onerous administrative burden on small-scale reforestation efforts.

Terrestrial Laser Scanning (TLS) has been proposed and used to address some of these problems. In this approach, surveying LiDAR instruments, sometimes purpose-built for forestry applications, create precise point clouds of forest scenes at a range of 50-100 meters. However, these instruments are expensive (as costly as $200K USD), complex to

operate, and bulky. They also produce a point cloud data format that requires complex, often vendor-specific software, lengthy offline computation times, and sometimes manual intervention during post-processing. In contrast, we explore the use of novel infrared time-of-flight (depth) sensors, along with a depth-based image segmentation algorithm, to measure tree trunks in near real-time in natural forest settings. Specifically, we obtain depth images using a commodity smartphone, segment the images, that is, separate the trunk from the foreground and background, and then automatically estimate the trunk diameter. We compare this to the results obtained manually through the traditional approach and find that our approach is around four times faster, while incurring a small error of $< 8\%$.

Although we are not the first to use smartphones for forest plot inventories, prior attempts assume that tree trunks are well-spaced, brightly lit, relatively small, and have minimal occlusion [12][13]. In practice – particularly in the natural forest environments that carbon sequestration incentive schemes should hope to foster – these conditions are unlikely to hold. In contrast, we are able to accurately measure tree trunk diameter (scientifically referred to as Diameter at Breast Height, DBH) under complex and realistic field conditions.

Our ultimate goal is to enable small-scale landowners to benefit from policy incentives for healthy, naturally-managed, biodiverse forests, despite low-lying vegetation, climbing vines, and fallen branches. MRV systems for carbon credits should not be limited to well-resourced groups or those with large plots of land to reforest. The MRV process should scale down as well as up, incentivizing reforestation on any small plot of underused land. Existing technological solutions are either prohibitively expensive or require highly favorable forest conditions. Our approach provides measurement tools that facilitate good climate policy.

1.2 System Requirements

In the interest of the goals discussed above, we have the following specific requirements for our system and its associated algorithm:

- *Accessible*: The image-acquisition system should be low-cost, and the computational cost of the algorithm should be low as well. The system should be useable on multiple platforms, and those platforms should be accessible to a relatively non-specialized user.

- *Accurate:* The algorithm should be accurate.
- *Real-time*: The algorithm should complete with a low-enough delay that the user can correct errors in collected data immediately and seamlessly during fieldwork. Ideally, a real-time system should also provide guidance to users to improve data collection.
- *Single image*: The algorithm should not require multiple images of a tree from different vantage points, since these may significantly slow data collection and be difficult to obtain in dense or tropical forests.
- *Handles reasonable occlusion:* The algorithm should not assume that the trunk image is unoccluded.

1.3 Contributions

Our work meets the above requirements, improving the MRV process for small-scale forest carbon sequestration projects by using depth sensor-enabled mobile phones to make forest inventories easier to perform, more efficient, and more accurate. Specifically, we make the following contributions:

- We design an algorithm that exploits a low-cost smartphone time-of-flight sensor to estimate DBH from a single image, even with naturally-occurring occlusion, and in poor lighting conditions.
- We implement our design and build it into an app on a Huawei P30 Pro Smartphone, demonstrating that the algorithm has low enough computational cost to run on this accessible commodity platform in near real-time.
- We evaluate our app in realistic forest settings and find that in a corpus of 55 sample tree images, it estimates DBH with an RMSE of 4.1 cm (7.8%). These results are unfortunately affected by a single outlier; we discuss the appropriate modifications to our algorithm to avoid its cause in the future and note that without this sample, the RMSE drops to 2.5 cm (7.5%).
- We incorporate real-time user feedback through our app UI, increasing trunk detection to 100% in our tested forest without requiring high user expertise.

- Recognizing that even time-of-flight sensors are somewhat specialized hardware, we port our algorithm to Google's ARCore Depth API, which can acquire depth on commonly-used mobile phones equipped only with a monocular camera. We offer specific suggestions for the improvement of ARCore depth-from-motion to allow it to be usable in a forest MRV context.

Chapter 2

Background

In this chapter, we present additional context for our work, especially for readers unfamiliar with current developments in forest science.

2.1 Traditional Forest Inventory

Forest surveying is a well-established field, practiced in both managed timber forests and partially-managed or unmanaged natural growth forests. The United Nations' Intergovernmental Panel on Climate Change (IPCC) collected peer-reviewed practices in their Land Use and Forestry report [31], offering a global standard for forest carbon measurement. Similar detail is provided in the UK's Woodland Carbon Code (WCC) [45]. The WCC lays out a more explicit bureaucratic process as well, including reporting spreadsheets for forest carbon projects that allow them to earn credits and payment from government programs. Both specifications follow the same general process: the land is first divided into relatively homogenous strata, and the strata divided into equally sized plots along a uniform grid. Plot size varies based on the density of trees in the stratum; the WCC recommends targeting around twenty trees per plot. Due to time and resource constraints, not every plot is measured in the field, but instead a subset of plots are sampled and their characteristics extrapolated to the other plots in the stratum.

In the field, the plots are typically marked out by hand, using a GPS device, tape measure, and rope or ribbon; a laborious and time-consuming process. Within each plot, foresters measure the diameter at breast height (DBH) of each tree, and may also record species and total tree height information. DBH measurements are performed either by

measuring tree circumference and dividing by π, or by using specialized calipers that fit around the trunk [37]. Height can be calculated with trigonometry using only a meter stick, but surveyors often use technical equipment such as a hypsometer. Top-of-the-line all-in-one-devices cost around $2000, though there are cheaper models for around $300 USD.[1] The appropriate species identification is done manually by field experts.

Following the forest inventory, estimated carbon sequestration is derived using *allometric equations*, which relate measured tree features to tree biomass. These equations are often specific to a species or other classification of tree (e.g. hardwood, etc.), to account for differing wood density and growth characteristics. Some allometric equations include tree height as an independent variable, but many use only tree classification and DBH, since height is not always easy or possible to determine, especially in dense forests where individual tree tops are not visible. These statistical models are developed based on tree measurement followed by *destructive harvesting*: cutting down the tree to precisely calculate its biomass. Further equations relate biomass to an estimate of a tree's carbon storage, which is aggregated into an estimate of total forest *carbon stock*. Finally, re-measuring the same trees or plots year after year results in an estimate of *carbon sequestration rate*. For an example of the practices used to develop these models across a given classification of tree, see Singh et al. [40].

Though this process is well-established, it has several key limitations. First, it is both time- and cost- intensive. Carrying equipment into the forest, marking out plots, and measuring each tree individually all take time. Second, a relatively high level of expertise is required to correctly place sample plot boundaries, measure trees quickly and accurately, and identify tree species. The time and expertise both translate to costs that small-scale reforestation efforts may not be able to meet. Finally, the entire process yields only a few data points per tree (such as species, DBH, height) – and not rich datasets, such as images, that could provide additional information after the surveyors have left the forest or serve as an auditable trace of the data collection process.

The high barrier to entry also impacts the overall accuracy of the process. The greater the time, cost, and expertise required, the less total land can be easily surveyed, meaning that surveyors have to rely more heavily on statistical sampling methods to extrapolate carbon stock estimates [31]. It also leads to under-representation of forests that are dense, diverse, and hard-to-reach, such as tropical forests.

[1] Sample devices can be found at forestrytools.com.au

2.2 Terrestrial Laser Scanning

One relatively widespread technology for automatic ground-based forest inventories is Terrestrial Laser Scanning (TLS). Forest ecologists use medium-range surveying LiDAR as well as some purpose-built LiDAR instruments to collect point cloud representations of forest plots [50]. Their results have been impressive, speeding up the forest inventory process and leading to direct improvements in allometric equations for biomass estimation [27]. In particular, TLS can estimate biomass without destructive harvesting, by using the full point cloud to estimate tree and branch volume for the entire tree. As a result, TLS has especially improved allometric models for larger, older trees and trees in remote or protected areas, such as tropical rainforests [10] [42].

However, TLS instruments have direct trade-offs between signal noise and range and sensor cost, and instruments with the high-end specifications used in existing forest surveys are quite expensive. One detailed study, published in 2018 but with field work done in 2015, used LiDAR costing 100K-200K USD per instrument [9]. The authors mention that their instruments are at the upper end of TLS scanner cost, with a range of 700 m and a pointing accuracy of millimeters at that range. Several authors mention that TLS scanners *can* cost closer to 10K USD, but their research actually uses an instrument costing over 50K USD [9] [50]. Wilkes et al. note that instrument weight can also be an issue, given that many sensors are over 10 kg and "can be heavy and awkward to carry through dense vegetation" [50]. Finally, the processing of forest TLS point clouds requires a combination of complex, often vendor-specific software for aggregation and segmentation, the full pipeline of which is not standardized nor professionally packaged. For these reasons, TLS seems suitable primarily for well-funded research groups, and to our knowledge has not been widely adopted either in government MRV programs or small-scale reforestation efforts.

2.3 Satellite and Aerial Approaches

Outside of terrestrial approaches, foresters and forest ecologists also work with Aerial Laser Scanning (ALS): LiDAR sensors mounted on planes, helicopters, and Unmanned Aerial Vehicles (UAVs). An aerial forest view can also be obtained from satellite data, which is increasingly available from private and public sources [33]. Moreover, aerial and terrestrial approaches do not need to be mutually exclusive; researchers are discussing using TLS data to improve satellite and aerial estimates of above-ground biomass [41].

Aerial approaches enjoy the advantage of being extremely fast, and are able to cover a wide area of ground, even in locations that might otherwise be difficult to access. However, they naturally cannot obtain data below the forest canopy, and so tend to underestimate forest biomass. Understory trees, or trees whose tops do not reach the outer canopy, are a particular source of error [8]. In addition, aerial approaches are less suited to smaller-scale projects: ALS can be quite expensive (even more so than TLS) because of equipment and flight costs, though cheaper versions are becoming available through the use of drones. Satellite data, on the other hand, is still not available at high enough resolution and frequency to measure carbon storage in individual forest projects. We therefore limit our evaluation to comparisons with terrestrial methods only.

2.4 Depth Measurement on Mobile Phones

Mobile phone software for creating depth images is rapidly becoming more widespread. Apple released depth capabilities in its ARKit in June 2017[2], Huawei offers depth through AREngine[3], and Google published its long-awaited Depth API as part of ARCore in June 2020[4]. As the package names suggest, the target use case of depth data on mobile phones is Augmented Reality (AR), which allows the user to interact virtually with their real-world environment. The Apple and Google software packages provide depth data on a subset of compatible phones even without a specialized depth sensor, using Structure from Motion (SfM) [38] or stereo vision techniques from multiple rear-facing cameras. However, higher resolution, acquisition speed, and responsiveness can be achieved with a dedicated sensor, and thus far Huawei, Apple, and Samsung have all released flagship devices equipped with an infrared time-of-flight (ToF) sensor.

Time-of-flight sensors are low-cost, short-range depth sensors that offer millimeter accuracy for depth measurements at a range of up to 4 m [28]. The sensor emits infrared waves and measures the round-trip time for the waves to bounce off of objects in the environment and return to the sensor. Straightforward velocity calculations convert the time measurement to distance. In Android APIs, the images produced by the depth sensor are formatted in a standard DEPTH16 format, with each pixel containing a 13-bit depth measurement (in mm) and a 3-bit confidence value [15]. However, the Android documentation does not specify what the confidence value should represent, and we found that it was used inconsistently or not at all by different vendor implementations.

[2] https://developer.apple.com/augmented-reality/arkit/
[3] https://developer.huawei.com/consumer/en/hms/huawei-arengine/
[4] https://developers.google.com/ar/develop/java/depth/overview

Time-of-flight sensors are known to be less effective under certain conditions, such as in bright sunlight and when measuring pure black objects. However, we have not encountered this issue in a shady, naturally forested environment. In fact, since time-of-flight sensors emit their own infrared light, they work very well in low light conditions.

2.5 Image Segmentation

Image segmentation, or grouping the pixels that belong to different objects in an image, is an established area of computer vision and a key requirement for automatic forest inventory tools. In this section, we offer a general (non-comprehensive) background on segmentation techniques. We discuss forest-specific approaches to point cloud and image segmentation in Section 3.1.

One basic image segmentation problem is to assign a label to each pixel according to whether it is in the foreground or background of an image. Some well-established approaches to this problem include *k-means segmentation* [34], and *region growing* [1]. K-means segmentation seeks to assign pixels to clusters such that the sum of squared differences between pixels and the mean of their assigned cluster is minimized. In region growing techniques, the algorithm begins with sets of seed pixels from the foreground and/or background of the image. These seeds are grown by iteratively adding connected pixels that are sufficiently similar to their neighbors already in the seeded region.

A more challenging problem is *semantic segmentation*, which attempts to identify the objects contained in an image and label each pixel with its corresponding object. Neural networks have been applied to this task, most notably ResNet [19]. The above algorithms were designed with RGB images, though some could be adapted to depth (or RGB+depth) images as well. In addition, some dedicated research has been done on RGB+D image segmentation through convolutional neural networks and other approaches, though this area is relatively newer[18][39].

In our work, we seek a combination of basic and semantic segmentation. We do not need to label every object in the image, but we do need to find the exact boundary between one chosen foreground object (the tree trunk) and the rest of the image. Small errors in the boundary, while not well-represented in the typical Mean Intersection Over Union (MIOU) error metric for semantic segmentation neural networks [19], may translate to large errors in DBH estimation. To our knowledge, there is no existing neural network for finding trunk DBH from RGB or RGB+D images of trees, nor is there an existing dataset of phone-based RGB+D images large enough to train one.

Chapter 3

Related Work

We now present prior work on using mobile technology to assist with forest inventories. We first discuss the image segmentation problem, then three competing automated approaches to estimate trunk DBH. Finally, we discuss the possibility of using generic mobile tools to tackle the problem.

3.1 Image Segmentation for Forest Measurement

As discussed in Section 2.5, image segmentation is necessary to distinguish trees from the surrounding understory, vegetation, and other trees in the background (see Figures 5.1, 5.2, 6.1, and 6.4 for examples from our dataset). This is a complex task due to occlusions, variegated light conditions, and overall similarity of green and brown shades in the image.

In TLS, some researchers have used specialized dual-wavelength LiDAR to separate leaves from branches [26]. Another approach is to carry out inventories in winter, in the absence of leaves [42]. Of course, relying on leaf-off conditions is not always possible and limits inventories to deciduous forests. A third approach, taken by Bauwens et al., is to clear nearby vegetation by hand during the survey and then use additional manual data processing after the fact [4].

Unlike these more heavy-handed approaches, software-based point cloud segmentation has been studied by Piermattei et al., who make use of the Forest Analysis Inventory Tool, FAIT, developed at TU Wein [32]. The tool itself, as well as the Structure from Motion pre-processing they use to derive tree point clouds, require substantial computational time and power, and are not suitable for use during field operations. For forest TLS, one

software package widely used for segmentation is `treeseg`, developed by Burt et al [6][1]. Like FAIT, this software is run offline after data collection. It separates trunks by looking for neighborhoods of the point cloud that share similar surface normal vectors and are therefore likely to belong to the same surface. Since these tools require extensive offline processing, it is not possible to use them in real-time, which prevents them from being integrated into a system that offers user interaction in the field. More specifically, they cannot prompt users to retake a scan in case of errors in the software processing or final result.

3.2 Structure from Motion (SfM)

Other researchers have also sought cheaper, more accessible automated forest inventory methods than TLS. Some researchers have used Structure from Motion (SfM) algorithms on mobile platforms to estimate the DBH of all trees in a plot [21][32]. Here, users are asked to follow a specific path when moving around the plot perimeter and throughout the plot, in order to ensure coverage of all trees with enough overlap between photos to correlate images. Piermattei et al. keep a 1 m mean baseline between adjacent camera images, and ultimately capture and process about 300-800 images for each plot containing 45-110 trees. Piermattei et al. also note that "additional photos were acquired [...] to avoid occlusions by stems, branches, twigs, and leaves" in denser areas of the plots. The authors captured their images with a Nikon digital SLR camera and performed offline processing with PhotoScan Pro to derive dense point clouds [32].

This approach is less expensive than TLS and may require less expertise. However, it requires users to walk a pre-specified path covering the entire plot, which may not be possible in the presence of undergrowth. It also requires numerous photographs both to perform SfM and to handle occlusion. Finally, SfM requires extensive post-processing of images and cannot provide real-time operator feedback, for example to suggest that another image be acquired due to too much occlusion. Indeed, Piermattei et al. report a trunk detection rate as low as 65% in a plot classified as "difficult" based on the density of trunks and degree of occlusion.

In contrast, our work based on a ToF sensor typically requires only a single photograph per tree to accurately estimate DBH. We are able to perform all of the processing in near real-time on the phone itself, and supply useful messages to allow users to adjust and

[1] https://github.com/apburt/treeseg

correct recorded data while still on-site to ensure trees do not go undetected. Thus, our work simultaneously improves data accuracy and reduces acquisition time.

3.3 Mobile Phone Apps

3.3.1 Generic Mobile Phone Measurement Apps

Before examining purpose-built mobile tools for DBH estimation such as those discussed below, we also identified and tested some generic mobile phone apps that might be accomplish the same task, such as CamToPlan [43] and Google Measure [16]. These are freely available apps that measure the length of flat surfaces. Typically, the user is asked to hold the phone so that the object is displayed in the camera preview, and then tap or draw the edge or line that they wish measured. We experimented with these apps to measure tree DBH in the field, but found that they were unable to obtain consistent or accurate measurements in almost all cases. We discuss our experiments with these apps further in Section 6.3, where we conclude that because of large inconsistencies (up to 42 cm) between measurements of the same tree, they are not currently robust enough to meet the accuracy requirements of a trusted forest MRV system.

3.3.2 Project Tango

Other silvicultural researchers have used mobile phones with depth sensors and AR software to estimate DBH. For example, Fan et al. used Project Tango [13], a purpose-built AR phone from Google[2], to estimate DBH, height, and the position of each tree trunk in their target setting. They achieve an impressively low Root Mean Square Error (RMSE) of 1.26 cm and mean error (or *bias*) of 0.33 cm for DBH measurements. However, silvicultural research focuses on highly-managed forests, such as those used for timber and commercial purposes or found around urban areas, which tend to have well-spaced trees with only one or two age classes, and no weeds or ground cover (especially near the tree base). The authors did not respond to a request for their code, so we were not able to test their system ourselves, but close analysis of their paper reveals ways that this environment dramatically simplifies the computational problem of identifying and measuring trees.

To get some sense of the different environments used, consider Figure 3.1. On the top row, we show two images from the evaluation environment used by Fan et al. The left image

[2]This line of AR phones have since been discontinued.

Figure 3.1: Comparison of forest conditions between Fan et al. and our work. Top: Figures 7 and 8 from the work by Fan et al. [13]. The top right image shows the RGB part of an RGB+depth image used in their evaluation dataset. Bottom left: A picture of two cedar trees, which was taken in the Van Cortlandt Forest used in our evaluation. Bottom right: RGB part of an RGB+depth image of one of the cedars, which we used in our evaluation dataset. This image was taken with the phone held just inside the outer canopy.

gives additional context for the forest (it is from a section of the paper unrelated to DBH, and so has other markings over it); the right image is the RGB part of an RGB+depth image used to determine tree diameter. On the bottom left, we show two cedar trees that exemplify the difficult occlusions that occur in our dataset.[3] Though the trunk is not visible at a distance, by holding the phone just underneath the tree canopy, we capture an RGB+depth image from which we determine diameter; the RGB part is shown on the

[3]Additional sample images from our dataset can also be found in Figures 5.1 and 5.2.

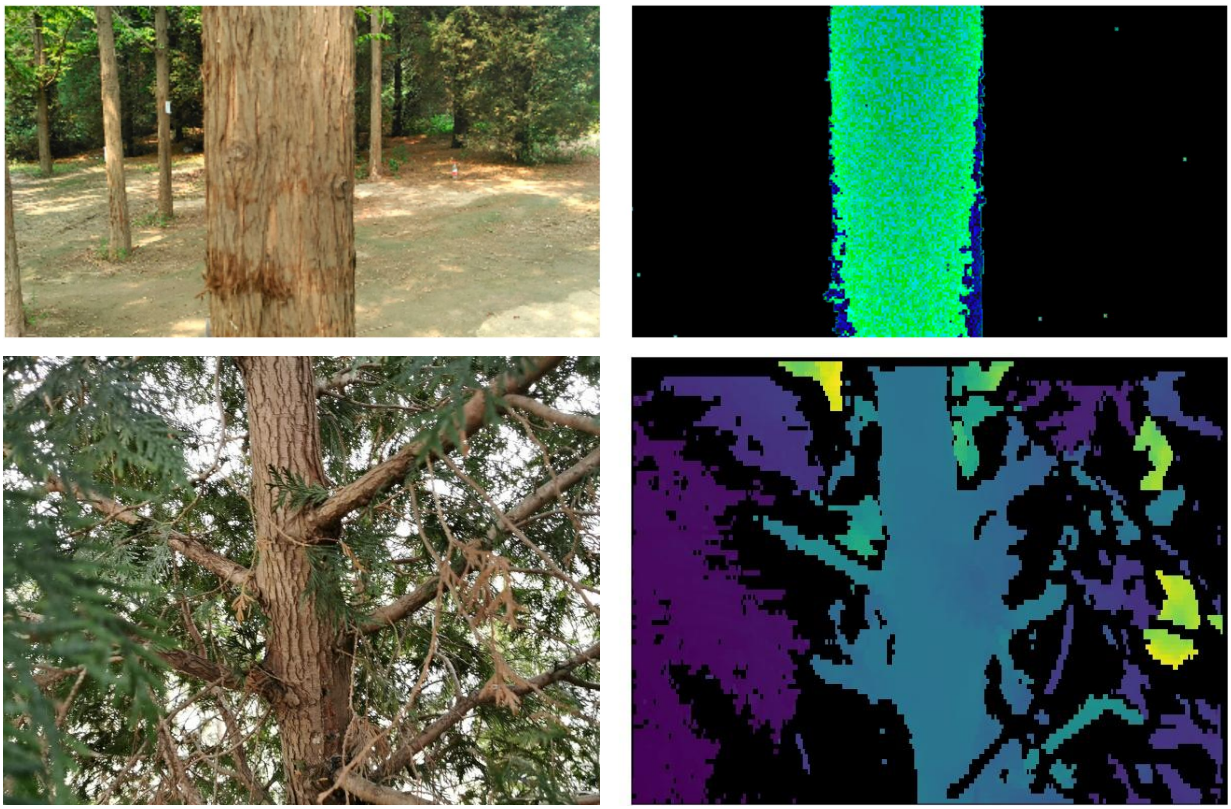


Figure 3.2: Comparison of RGB and depth images between Fan et al. and our work. Top: Figure 8 from the work by Fan et al. [13], showing a sample RGB and depth image from their dataset. Bottom: RGB and depth image for one of the cedar trees used in our dataset.

bottom right of the figure.[4]

In Figure 3.2, we show the RGB and depth images from the example image in Fan et al. and from one of the two cedar trees. The reader may notice the difference in depth images particularly, with the image from Fan et al. showing only the tree trunk, while our sample has many depths that correspond to leaves and branches. Since the depth sensor has a limited range of only a few meters, depth images taken in a forest with well-spaced trees and no low-hanging branches or undergrowth will naturally capture only the target tree trunk, as occurs in the sample image from Fan et al. The depth image therefore

[4]Incidentally, densely canopied cedars are also a good example of where our algorithm performs well compared to manual forest inventories; we discuss this further in section 6.2.

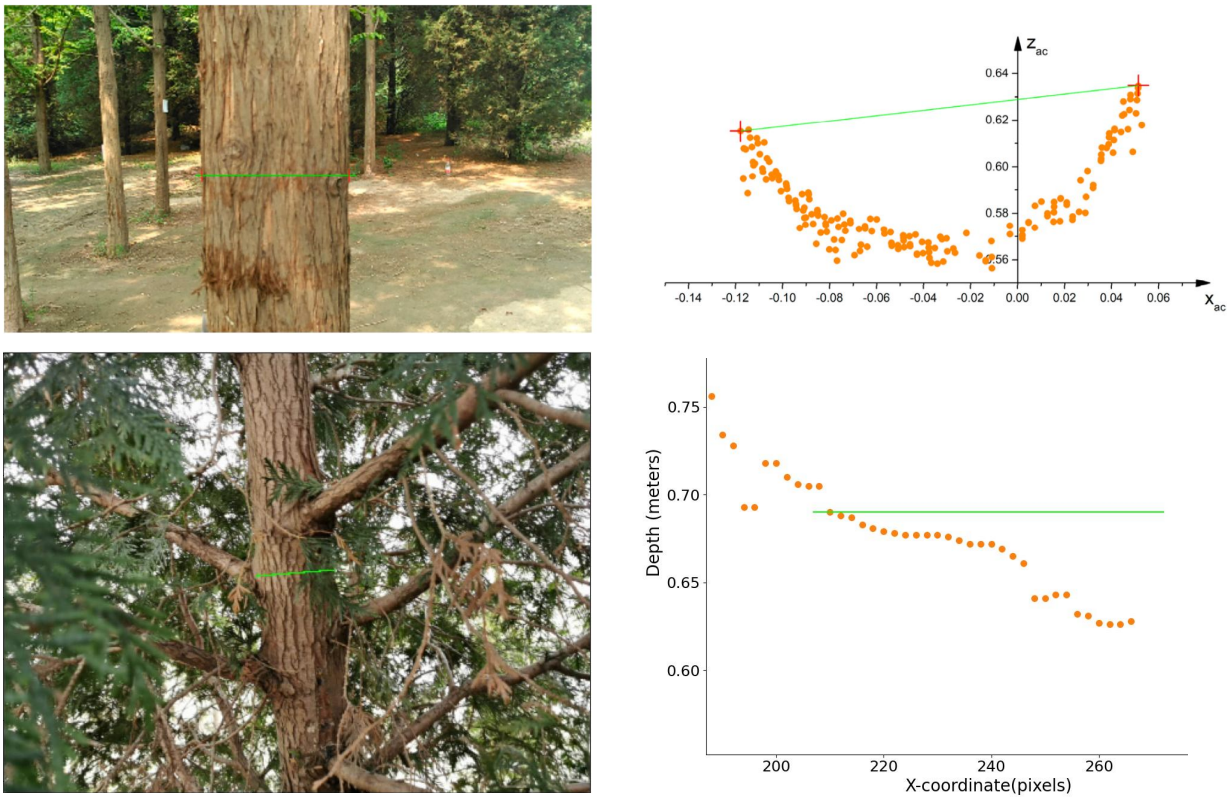


Figure 3.3: Comparison of depth cross-sections between Fan et al. and our work. Top: Figure 12 from the work by Fan et al. [13], showing a sample RGB image with the depth image cross-section from which they estimate tree diameter. The green line shows the estimated diameter line where the cross-section was taken. Bottom: Sample RGB image and depth cross-section from our dataset. The green line shows where the cross-section was taken, along a line that our algorithm has identified as a possible place to measure the diameter of the tree (though we do not use a single cross-section to estimate diameter).

offers a straightforward segmentation algorithm: the tree is the set of pixels with non-zero depth values. Indeed, Fan et al. do not mention in their paper how they handle occlusion or identify the tree trunk to measure, whereas our algorithm required robust image segmentation as its first step.

Fan et al. also seem to have faced a simpler computational problem when estimating trunk diameter. In Figure 3.3, we show a figure from Fan et al. on the top row, in which the authors demonstrate their method for estimating diameter by fitting a circle to a breast-height cross-section of trunk in their captured depth image (the slice is highlighted in green in the RGB image on the left). On the bottom row, we attempt to recreate this figure with the cedar sample, highlighting in green a cross-section in the middle of the captured trunk. Notice that the depth points from the trunk shown by Fan et al. form a rough circle. In the cedar, by contrast, this cross-section of the tree does not look circular, because of the branch on the left side sloping away from the camera and the branch on the right side sloping towards it. Obtaining an accurate diameter estimate by fitting a circle to the points shown in this cross-section would not necessarily be straightforward, even if we attempted to remove the points that we believed corresponded to the branches and leaves.

Notably, a system designed for a silvicultural context is unsuitable in environments that are of most significance from a climate perspective, i.e., naturally-managed forests, which offer ecosystem services such as improved biodiversity and increased habitat for native species. Our system prioritizes climate and sustainability by addressing MRV of natural and naturally-managed forests *in situ*.

3.3.3 Katam

Outside of the academic research community, there is a commercially available mobile phone app, Katam, which proposes to measure forest trees automatically [22]. Katam is mostly a black box in terms of code, methods, and documentation, but we experimented with it in a forest setting to test its capabilities. It appears to use SfM running on the phone, asking the user to take a continuous video as they walk through 20 meters of forest, holding the phone horizontally. The user can optionally place a reference sign (ordered from `www.katam.se`) halfway along the path to increase precision, presumably by helping the system recalibrate its position. Once the video is complete, Katam processes the video fully on the phone, identifying tree trunks and computing their diameters along with other parameters, such as total basal area.

There are three key problems with this approach. First, like the work from Fan et al., Katam does not support forest areas with branch or vegetation occlusions. Their website states that their method is designed for "production-oriented" forest environments (well-thinned timber stands), adding that "if the lower part of the tree is obstructed by undervegetation or dense branches, the system may not be able to identify the tree correctly." Their user guide displays the examples shown in Figure 3.4 for reference. In our

Figure 3.4: Screenshot from the Katam app user guide, showing acceptable forest environments

own tests on an area with dense underbrush, Katam was unable to find any trunks. (See §5.5.3 for further details.) Second, Katam does not continuously process the recording; instead it processes each video after it is taken, which it states can take up to 30 minutes. A new video cannot be taken until the first has been processed, leading to potentially significant delays and downtime while in the field. Third, Katam offers no mechanism for users to correct the mistakes made by its algorithm. Since the video is taken continuously and processed afterwards, a single mis-recorded trunk requires re-taking and re-processing the entire video. In another of our tests, which took place in a largely unoccluded forest setting in leaf-off conditions, Katam identified a trunk where there was none; we could find no way to correct this mistake without redoing the entire section of forest.

That said, the app has an excellent UI and can directly export spreadsheet data files. In general, we do not envision Katam's work as entirely independent of ours, and imagine that our algorithm, which can handle occluded forest conditions (as well as offering real-time feedback and allowing users to correct mistakes) might be integrated into a commercially available app.

Table 3.1: Summary of related work

Capability	TLS	SfM	Tango	Katam	Measurement Apps	ToF
Real-time	✗	✗	✓	✗	✓	✓
Uses single image	✗	✗	✓	✗	✓	✓
Handles some occlusions	✓	✓	✗	✗	✗	✓
Accessible	✗	✓	✗	✓	✓	✓
Accurate	✓	✓	✓	✗	✗	✓

3.4 Summary

Referring back to the system requirements that we laid out in Section 1.2, we summarize prior approaches for DBH measurement in Table 3.1. Here, TLS refers to Terrestrial Laser Scanning, SfM to Structure from Motion as tested by Piermattei et al. [32], Tango to the Google Tango-based system proposed by Fan et al. [13], Katam to the app discussed in §3.3.3, Measurement Apps to Google Measure and CamToPlan, and ToF to our work. In short, we are not aware of prior work that meets these requirements.

Chapter 4

Design and Implementation

This chapter describes the design and implementation of our proposed system, and is laid out as follows. We offer a brief overview in Section 4.1, followed by details of the mobile phone hardware and software we chose to work with in Section 4.2. In Section 4.3, we discuss the core design of our unassisted algorithm. Readers may wish to refer to Figure 4.1 for an overview. In Section 4.4, we discuss the adaptations made to incorporate user feedback into our work. In Section 4.5, we discuss limitations of our design, and finally in Section 4.6 we describe porting our work to an alternative platform.

4.1 System Overview and Design Challenges

We propose a system running on a mobile phone equipped with a depth sensor that can efficiently measure tree DBH, meeting the design goals discussed in Chapter 1. We use the depth sensor both to estimate real-world diameter with high accuracy and to segment the trunk from the surrounding forest. Our algorithm uses estimation techniques that are robust to measurement error and accounts for some occlusions, maintaining high accuracy in a variety of forest conditions. The input to our algorithm is a single depth image, and we use processing techniques with low computational complexity so that it runs in near real-time. The app that runs our algorithm gives feedback to users and allows them to correct mistakes on the spot. The depth sensor that we use is relatively cheap and built into a commodity smartphone, and we consider adapting our designed app to yet more generic hardware in §4.6. The algorithm uses only standard statistical libraries and image analysis algorithms provided by the OpenCV library [5], which has been ported to numerous platforms.

We overcome several challenges when identifying and segmenting a tree trunk from an RGB or depth image. Due to occlusion from leaves and branches, the trunk will not always appear as a single connected object in the depth image. Branches frequently divide sections of the trunk, and the depth sensor widens these gaps, unable to detect waves returning from behind a small area surrounding the occluding object. The depth sensor is, however, able to handle scenes that are difficult to segment using RGB images alone, such as mottled trunk coloring, dim lighting, or climbing vines. In addition, typical edge detection techniques for RGB images are difficult in the frequent case where a second trunk is partially aligned in the background, with the boundary between the two almost indistinguishable in the RGB image.

4.2 Mobile Phone Hardware and Software

We chose to work with the Android OS since that allowed us a wider choice of mobile phones, compared to iOS. We used a Huawei P30 Pro phone, which, at the time of purchase, retailed for around 1100 USD (but has since fallen to a little under 600 USD). We chose this platform because it offered direct access to raw ToF sensor data through both the Android Camera2 API [14] and the Huawei AREngine API [20]. Samsung, the other main manufacturer producing Android phones with a ToF sensor, only allowed ToF use through existing Samsung apps at the time our research began, though it now offers similar flexibility. The P30 has three rear-facing cameras (40, 20, and 8 MP), one rear-facing TOF sensor, and a front-facing camera (32 MP). It is also equipped with 128 GB of SSD storage, 6 GB of RAM, an 8-core CPU and a separate GPU.

Huawei does not publicly disclose the full specifications of its ToF sensor, though investigative teardowns of the phone reveal that the sensor uses Lumentum's flood illuminator to emit Infrared light, and Sony's integrated circuit image sensor [2][52]. The sensor has a resolution of 180×240 pixels. In line with other ToF sensors [28], we found it to have a maximum range of around 3-4 meters. In our system, we direct users to take images at around 1-2 meters from the tree. ToF sensors generally report sub-centimeter accuracy in many conditions [28]. Since the sensor works by emitting infrared light, it can be affected by strong natural light. Also, as with other sensors that emit waves and analyze their return patterns as they bounce off of objects in the surrounding environment, ToF sensors perform less well on surfaces nearly parallel to the direction of emitted waves. This is because such surfaces will tend to deflect waves away from the ToF sensor. They may return by indirect paths when they hit other objects in the environment, or not at all.

Initially, we accessed the raw ToF sensor data directly through the

`android.hardware.camera2` API. This is usable for depth data alone, but does not (as of this writing) allow the user to issue two simultaneous (or near-simultaneous) capture requests to the depth and RGB cameras. Android does offer a multicamera API, with which multiple physical cameras that face the same direction can be combined into a single logical camera at the software layer, with capture requests dispatched to each. However, this API does not support combining a depth sensor with an RGB camera. An Android development guide states, "We can expect that, in most cases, new devices launching with Android Pie will expose all physical cameras (the exception being more exotic sensor types such as infrared) along with an easier to use logical camera." [48] Indeed, when we attempted to use this API to combine the ToF sensor and RGB camera, the ToF sensor results no longer corresponded to correct depths. We suspect that this is a software problem, given that we were able to obtain the two images together through other APIs.

Instead, we used Huawei's AR Engine SDK. The Huawei SDK uses an ARFrame Java object with functions `acquireCameraImage()` and `acquireDepthImage()` to allow the developer to work with the RGB and ToF cameras simultaneously. The Huawei depth data retrieved through this API is pre-calibrated with the RGB image, so the two can be rescaled to the same height and width and then overlaid directly. The returned image from `acquireDepthImage()` is a buffer of `unsigned short`s in DEPTH16 format. The buffer is directly accessible in C++, so an app written in that language would not need to modify the buffer, but the Huawei Java API does not account for the fact that in Java the endianness is reversed. In our app, the two bytes representing each depth pixel must therefore be reversed before they can be parsed as DEPTH16.

In the time since we began this work, Google also released its own Depth API as part of its ARCore SDK.[1] This API has an `acquireDepthImage()` method as well, but it does not offer direct access to the raw ToF sensor results. Instead, ARCore opaquely combines depth-from-motion from the monocular RGB camera with the depth sensor input (if available). We discuss integration with this API separately in Section 4.6.

The DEPTH16 format reserves 3 bits of each pixel for a confidence value indicating the correctness of the measurement [15]. However, the documentation does not describe how this confidence value should be computed nor what, specifically, it represents, and we were unable to use the values in either the Huawei AREngine or Google ARCore APIs. For Huawei, in a typical unoccluded outdoor image at a 2 m range, almost all observed depths have 0% confidence; a scattering of those on nearby objects may have 15% confidence. Note that since the confidence values are represented by 3 bits, there are only 8 confidence values available, split evenly among the range from 0-100%. Thus, 15% is the smallest

[1] https://developers.google.com/ar

possible non-zero confidence value for a DEPTH16 pixel. In depth images returned by ARCore, all pixels have a 100% confidence value.

4.3 Image Processing Algorithm

We now turn to the core image processing algorithm that we developed for identifying trunks and measuring their diameters, which is automatically invoked when the user taps the app's "Capture Image" button. (We discuss the app design and user assistance in Section 4.4.) For the purposes of this algorithm, we assume that trunks consist of a single, roughly cylindrical stem; we discuss this and other assumptions in more detail in Section 4.5. Figure 4.1 shows a high-level overview of the algorithm. In Figure 4.3, we demonstrate each step of the algorithm on a sample tree image, shown in Figure 4.2, with considerable occlusion.

4.3.1 Step 1: Identify Approximate Trunk Depth

The first step is to identify the tree trunk. In this regard, the depth image is easier to work with than an RGB image, because it does not have a continuous background of pixels. The sensor can only detect points within a range of approximately 4 m, so in an outdoor setting many of the pixel depths are set to zero: there are no objects in close range along that direction. An RGB image, by contrast, would have a continuous forest background. Thus, the depth sensor itself gives us a crude segmentation, as we can see from the depth image shown in Figure 4.3 (right image).

In this version of our algorithm, we are attempting to identify the trunk automatically, with the natural assumption that the photographer will attempt to center the tree trunk in the image. We slice the image vertically into thirds, and consider depths in the center third of the image as potentially belonging to the trunk. We expect the trunk be a large set of points with broadly consistent depth (unlike vegetation or branches, which will either be of relatively inconsistent depth or small size). Thus, we bucket the depth values in the center third of the image into 3 cm ranges and take the mode bucket as the approximate trunk depth, δ_m. A sample histogram is shown in Figure 4.3b. We then filter the image for pixels whose depth value is within $\pm 10\%$ of this approximate trunk depth. This forms a rough segmentation, or labeling of the image pixels, I_s.

[2]It was somewhat coincidental that the algorithm captured the intended trunk from the two in this image. However, if it had not, the user can easily correct the mistake, as discussed in §4.4 and §5.4.

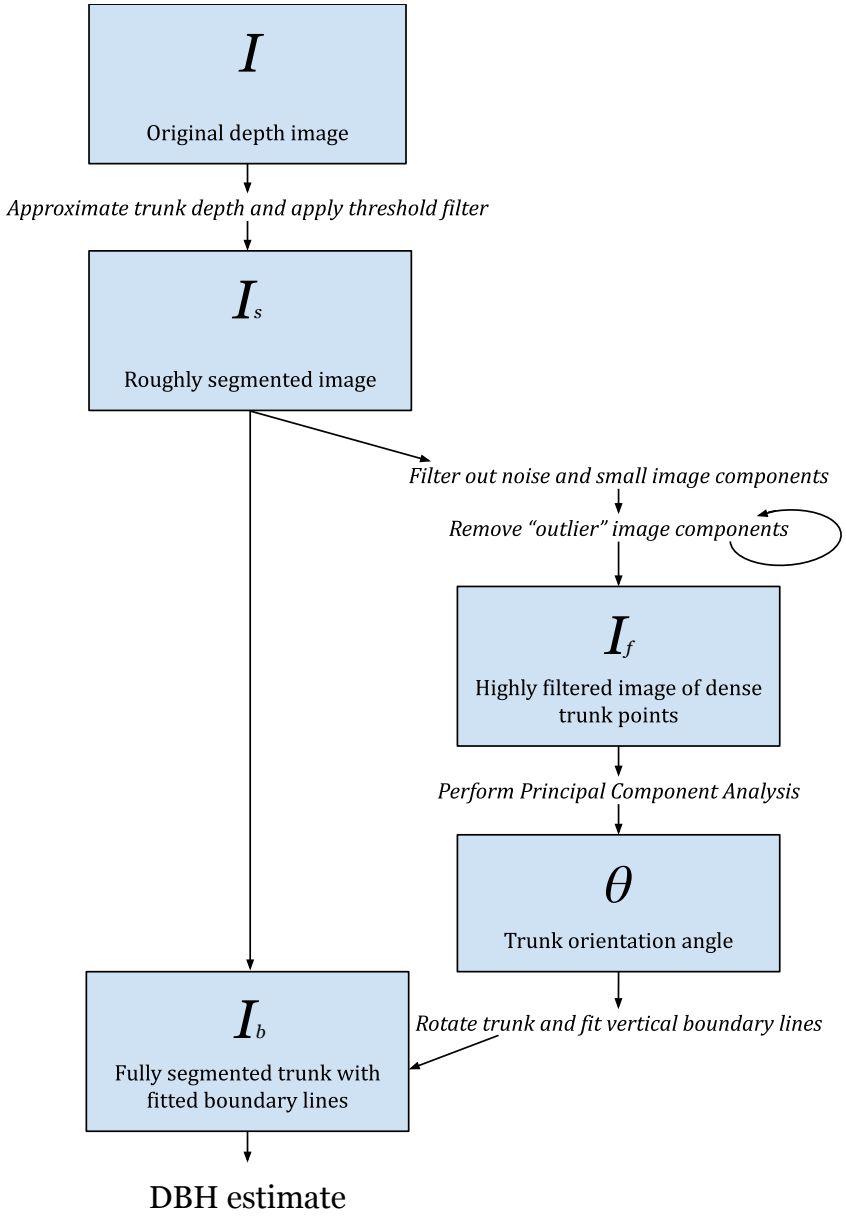


Figure 4.1: Overview of the process for isolating and measuring a tree trunk from a depth image. The RGB image is used to display results to the user, and is not required for the core processing algorithm. We discuss improving the image segmentation based on the RGB image in §6.5.

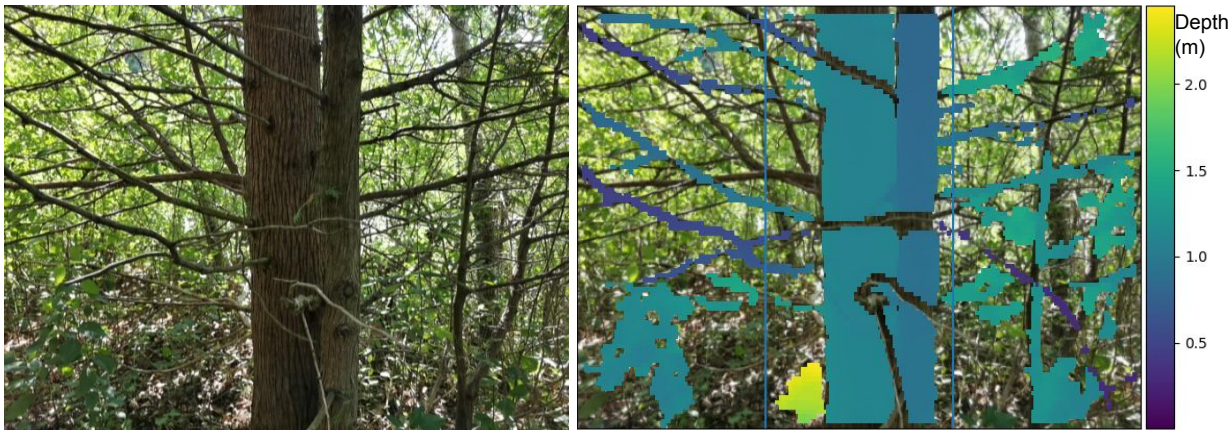


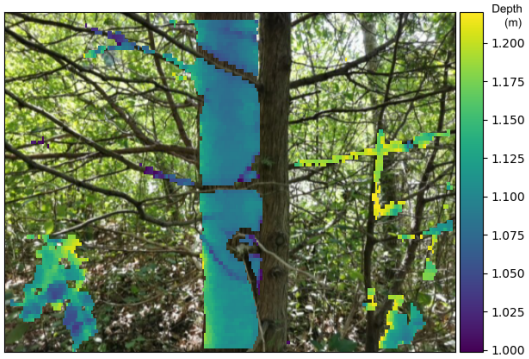
Figure 4.2: Sample image on which we demonstrate the steps of our algorithm. Left: RGB image. Right: RGB image with depth values, in meters, overlaid. The user intended to measure the leftmost of the two trunks captured in this image.[2]

4.3.2 Step 2: Segment Trunk

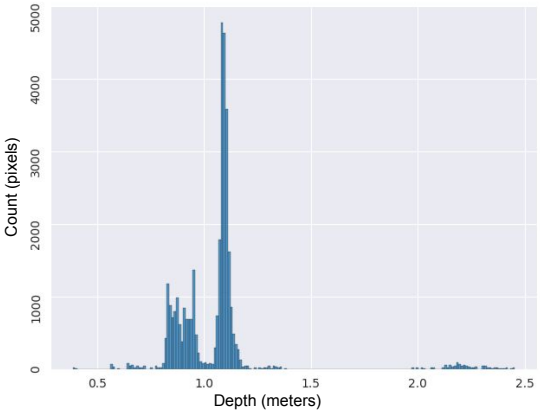
I_s may include some leaves and branches that happen to match the trunk depth, while omitting portions of the trunk that are obscured by closer leaves and branches, as seen in Figure 4.3a. Conceptually, we now want to find a tight boundary for the trunk that will omit these outliers while still containing ("filling in") the missing portion. The distance between the left and right boundary will then correspond to the diameter of the trunk. We also need to find the orientation of the trunk, because the diameter must be measured perpendicular to this orientation.

Initially, we attempted to use the Quickhull algorithm [3] to find a convex hull for the pixels remaining in the center third of the image. This simple approach easily fills in the missing portions of the trunk obscured by occlusions, but is overly sensitive to small protrusions around the edge of the trunk, for example the beginning of lower branches. Instead, we begin by finding the orientation of the trunk, as discussed next, then we rotate the image so that the trunk is vertically aligned and search inward along vertical scan lines to find the appropriate trunk boundaries.

Notably, we use two different versions of the same image for each of these two stages. For reasons described below, the process for identifying the trunk orientation requires a highly filtered image, I_f, which may be missing some of the true trunk pixels. To fit trunk boundaries, we use I_s, the roughly segmented image that may include some non-trunk pixels.

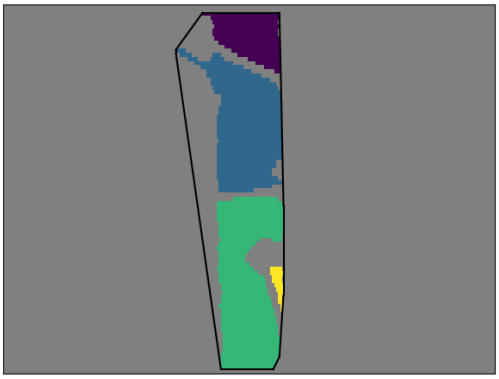


(a) I_s, roughly segmented image



(b) Histogram of depth values

(c) Connected components of I_s[3]



(d) I_f, filtered image with fitted convex hull

(e) Binary version of I_f with eigenvectors

(f) Final trunk boundaries based on I_s

Figure 4.3: Algorithm steps performed on a sample image.

Step 2a: Orient image

Overall, the pixels in I_s are typically dominated by trunk pixels, which form a rough oblong cluster. We follow the approach described by Rehman et al. [35] for automatically aligning such images, finding the principal axis of the trunk using Principal Component Analysis (PCA). PCA is sensitive to outliers, but is able to identify the principal axis relatively well even with a small subset of the trunk pixels missing. As a result, we prefer to over-filter the image, ensuring that few non-trunk pixels remain, even at the possible cost of losing some true trunk pixels. We call this highly filtered input to PCA I_f.

To obtain I_f, we remove two main sets of outlier pixels: small clusters of pixels corresponding to individual leaves or small elements in the environment that happen to share the trunk depth, and substantial objects (e.g. shrubs, large branches, additional trunks) that share the trunk depth. We begin by parsing I_s into its connected components [51], an example of which can be seen in Figure 4.3c. To filter out small clusters, we remove all connected components below a threshold number of pixels, $\alpha = 300$ pixels. This thresholding is analogous to the pre-PCA filtering performed by Rehman et al. [35].

We also remove substantially sized objects, as they will affect the axis found by PCA. Our goal is to keep a set of pixels that represents the majority of the trunk and maintains its general shape and orientation. Depending on the level of occlusion, the trunk may be split across multiple connected components, as is the case in Figure 4.3c. Since the underlying trunk is a large, oblong shape, the set of connected components that represent the trunk should, intuitively, form a dense cluster in the image. Here, we define the *density* of a set of connected components as the ratio of the area of the components to the area of the convex hull around those components. Large components that represent leaves and branches tend to extend beyond the convex hull around the trunk components, because they are far from the main trunk or point in a different direction. Including them in the trunk subset will therefore result in a low density measurement.

The algorithm searches for this dense subset of components by removing components one by one in a given order (discussed below) until the image contains only a set of components above a threshold density. We found through empirical trials that a threshold value of $\beta = 0.60$ works well.

[3] In Subfigures 4.3c and 4.3d, the colors are only to distinguish connected components and do not represent depths.

The detailed algorithm for finding the trunk components is shown below:

Input:
components: Connected components of the image and their areas;
image: $N \times M$ depth image, with background pixels set to zero;
Output:
$N \times M$ filtered depth image, with outlier pixels set to zero;

```
1  components ← SortByHeuristic(components);
2  totalArea ← Sum(components.area);
3  while length(components) > 1 do
4  |  hull ← ConvexHull(image);
5  |  density ← totalArea / hull.volume;
6  |  if density < β then
7  |  |  removed ← components.pop();
8  |  |  image ← SetComponentPixelsToZero(image, removed);
9  |  |  totalArea ← totalArea - removed.area ;
10 |  else
11 |  |  return image
12 |  end
13 end
14 return image
```

The first step of the algorithm, in line 1, is to sort the components. The order in which the components are removed will have an effect on the final result, and we tested two different heuristics for performing this sorting step. Conceptually, both heuristics try to order components by how much they stretch the convex hull away from the correct trunk components.

- Sort the components by the percent of their pixels that lie within a target range where we expect the true trunk to lie. Without user assistance, a reasonable choice is the center third of the image.
- Sort the components by how far their horizontal center of mass is from a target location in the image. In the version of the algorithm with no user assistance, we set the target location to be the horizontal center of mass of the largest non-background

component in I_s, which is assumed to be part of the trunk (see §4.4 for alternatives). Specifically, the horizontal center of mass, m_i, of a set of pixels, i, is the mean x-coordinate of those pixels. We then sort the components by $m_i - m_t$, where t is the target (largest) component.

The first heuristic frequently removed too many components, leaving only a small component that happened to be contained entirely within the target range. We found the second heuristic to work quite well in practice. We discuss images on which the algorithm performs poorly in Section 6.1.

In theory, with a large number of components to remove, this algorithm could be computationally expensive. However, because we begin by filtering out all small connected components, there are usually fewer than ten components remaining in the image. We also minimize the complexity by sorting the components first, and then removing them in a fixed order. We considered a more complex approach, computing a convex hull for each possible set of $n - 1$ components and removing the component whose removal resulted in the highest density convex hull. However, because of the additional time complexity, it makes sense to use this approach only if no other, faster, heuristics are successful.

After completion of this algorithm, we have a highly filtered version of the image, I_f. A sample I_f can be seen in Figure 4.3d. To perform PCA, we use a list of the pixel coordinates where I_f is non-zero. PCA computes the eigenvectors of the covariance matrix of these data points, which indicate the direction of maximum data spread. Based on these eigenvectors, we can identify the principal axis of the trunk and orient the image so that the trunk is vertical. We say that the eigenvector, e, of the principal axis sits at some angle to the horizontal, θ, generally defined by $\theta = arctan(\frac{e_1}{e_0})$, where e_0 and e_1 are the components of e. We align the image by rotating it $90 - \theta$ degrees.

There are two subtleties in translating the eigenvector output of PCA to θ. First, we are using the eigenvector to define an axis, but adding 180 degrees also identifies the same axis. Therefore, we set $\theta' \equiv \theta \mod 180$. (The tree always points "up," though it may lean left or right.) Second, PCA yields two eigenvectors: one pointing along the principal axis of the trunk, and the other perpendicular to it. Notably, the weight of the eigenvalues is not guaranteed to identify the correct eigenvector for this application of PCA [23][35]. We must therefore assume that the tree is relatively upright, and that the correct eigenvector is within 45 degrees of vertical. This is analogous to the approach taken by Kour et al. [23]. A sample of the eigenvectors found can be seen in Figure 4.3e. The sample image we are using is mostly vertical, so in Figure 4.4 we also show an example of the eigenvectors we find on a more tilted trunk. We can see that the eigenvectors identify the principal axis of the trunk relatively well.

Figure 4.4: Eigenvectors identified by PCA on a tilted trunk. Left: RGB image. Right: Binary version of I_s, with eigenvectors overlaid as black arrows.

Step 2b: Fit trunk boundaries

With the trunk oriented vertically, we return to I_s, the minimally filtered image (now rotated), to search for the trunk boundaries. Recall that I_s contains only depth values within $\pm 10\%$ of the approximate trunk depth, and is zero elsewhere. We do not use I_f, since the filtering required to give PCA a clean image may also have removed some of the true trunk pixels. We use a two-pass algorithm to iterate through vertical scan lines of a binary version of I_s.[4] In the first pass of the algorithm, we move inward from the left and right edges of the rotated image, stopping at the first vertical scan line at which the ratio of nonzero pixels to zero pixels exceeds a high threshold, $T_{high} = 0.6$. The high threshold ensures that we continue iterating, even over somewhat large non-trunk objects, to a scan line inside the trunk.

If the algorithm stops here, it tends to underestimate the trunk diameter. Intuitively, this is because the first pass continued until it reached a scan line that was almost certainly *inside* the trunk, but may not be a good representation of the boundary of the trunk, where we would expect to see fewer nonzero pixels due to irregularities in the bark and ToF sensor measurement error near the trunk edge. Quantitatively, stopping after the first pass with a threshold T_{high} resulted in a strong negative bias in the overall results. We therefore add a second pass, which iterates back outwards to the first scan line at which the ratio of nonzero pixels to zero pixels falls below a low threshold, $T_{low} = 0.5$. We select the scan

[4]The binary version is 0 where $I_s = 0$, and 1 otherwise.

line immediately before this one as the trunk boundary. The final segmentation consists of all the pixels in I_s that lie within this boundary, as shown in Figure 4.3f.

4.3.3 Step 3: Estimate Diameter

Finally, we translate the trunk boundaries and depth pixel values into a diameter estimate. Throughout this section, we will use the subscript p to denote a quantity in pixels and m to denote a quantity in meters, in order to better keep track of units in equations containing only variables.

Simple Diameter Approximation

The depth sensor has an intrinsic calibration constant, γ_{1_p}, which is the width, in pixels, of a 1 meter object at a distance of 1 meter. This calibration constant can be obtained through the AREngine API. We define γ_{k_p} as the width, in pixels, of a 1 meter object at a depth of k meters. Based on the principles of camera optics, we have the relationship

$$\gamma_{k_p} = \frac{\gamma_{1_p}}{k_m} \cdot 1.0_m \tag{4.1}$$

We can use this equation to obtain a simple approximation of the diameter as follows. The left and right boundary lines can be defined as the lines $x = l_p$ and $x = r_p$, respectively. The distance between these two lines then gives a rough estimate of the trunk diameter in pixels, $d_p = r_p - l_p$. Using the estimated trunk depth δ_m from §4.3.1, which was defined there as the modal depth in the center third of the image, we can approximate the diameter of the trunk, d_m, in meters, as

$$d_m = \frac{d_p \cdot \delta_m}{\gamma_{1_p}} \tag{4.2}$$

We show the intuition behind this approximation on the left side of Figure 4.5.

The simple approximation given in the above formula tends to consistently underestimate the trunk diameter, especially for large trees. One reason for this is shown on the right side of Figure 4.5. The boundaries represented by r_p and l_p are not at the same depth as the approximate trunk depth δ_m, as was assumed in the simple approximation. The trunk is curved towards the sensor, so the mode depth δ_m found in §4.3.1 is somewhere in the blue region depicted in the figure, closer than the points along the boundaries of

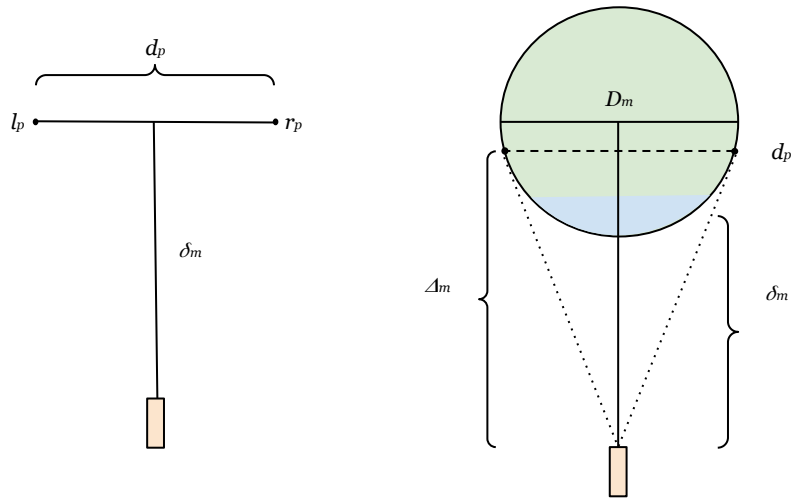


Figure 4.5: Diagram showing different methods of approximating tree diameter. Left: Bird's-eye-view diagram representing a simple approximation of the ToF sensor (colored orange) pointed at a tree. The left and right boundary lines determined in §4.3.2 are pointing straight up out of the page at the points labeled l_p and r_p. We can approximate the tree diameter in pixels, d_p, as the number of pixels between them. In this simple approximation, we assume that the diameter line is at a distance δ_m from the sensor, where δ_m is the approximate trunk depth calculated in §4.3.1. Right: Modified bird's-eye-view diagram of the ToF sensor pointing at a trunk. Since the tree is three-dimensional and roughly cylindrical, the left and right boundary lines found in §4.3.2 lie on a cylinder (shown here as a circle) around the trunk. d_p, the distance between the left and right boundaries, is not the diameter of this cylinder, D_m, but rather the length of a chord slightly closer to the ToF sensor than the true diameter. This is because of the field of view of the ToF sensor (represented by dotted lines). δ_m, the approximate depth of the trunk found using the mode depth in the center third of the image (§4.3.1), does not correspond to Δ_m, the depth of the chord d_p, but instead to a smaller depth somewhere in the blue region at the front of the cylinder.

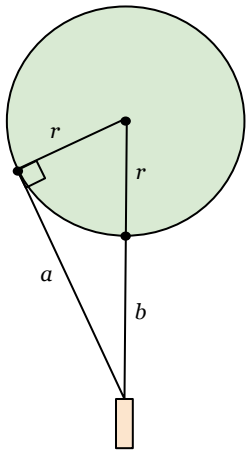


Figure 4.6: Possible geometric method to compute diameter of a circle given the distance from an external point to two tangent points.

the trunk where we are estimating d_p. This effect is most significant in larger trees. Also, the ToF sensor will not actually capture pixels at the widest part of the trunk (along D_m). Rather, the depth pixels on the boundary of the trunk occur on the line tangent to the trunk perimeter that intersects the time-of-flight sensor, shown in dashed lines in the figure. This effect will be more noticeable the larger the tree and the closer it is to the sensor.

Geometric Approaches

It may be intuitive to try to approach this problem mathematically, by imagining the measured depth points to approximate a circle and using trigonometry and other geometric methods to find the diameter of the circle defined by these points.

For example, we might approach this problem by considering the diagram in Figure 4.6. Without any reference to γ or any assumptions about where the boundary points lie on the circle, we have the following equation:

$$a^2 + r^2 = (b + r)^2 \tag{4.3}$$

Solving for $2r$, which is the diameter, in meters, of the circle (D_m), we have:

$$2r = \frac{a^2 - b^2}{b} \tag{4.4}$$

However, the problem with this and other similar approaches is that individual depth points returned by the sensor do not tend to lie along a circle. Moreover, because the ToF sensor does not perform as well on surfaces nearly parallel to it, depth measurements along the boundary of the tree – key to a good approximation of the curvature of the circle – tend to be less robust. Huawei does not quantify the accuracy or sensitivity of the ToF sensor on different surfaces, but in Figure 4.7, we plot the middle cross-section of various trees from our dataset. We can see that using a subset of points along these cross-sections to approximate a circle could result in vastly different diameter estimates depending on which points are chosen.

Moreover, depending on the occlusions in front of the trunk, how accurately offshoot branches have been filtered out, and the presence of burls or other trunk irregularities, it may also be difficult to accurately fit a circle to the cross-section as a whole. This is particularly apparent from the example in the bottom row of Figure 4.7 and from Figure 3.3.

Improved Diameter Approximation

The boundary lines found in Section 4.3.2 actually provide a relatively robust understanding of the width of the tree, because we estimate them based on the entire visible length of the tree, and because they do not rely heavily on the precise depth values at the edge of the tree – only that the values are non-zero. Looking again at Figure 4.5, we also notice that small changes in the location of the tangent points l and r have a small effect on d_p, as long as the tangent points are roughly near the true diameter line of the trunk. (The width of a circle changes most slowly near its diameter.) We make the simplification, therefore, that d_m, the distance in meters corresponding to d_p, is roughly equal to D_m, the true diameter.

By contrast, changes in δ_m will have a linear effect on our estimated DBH (as the simple approximation in Equation 4.2 shows). Therefore, we cannot make the approximation that δ_m, the mode trunk depth, equals Δ_m, the depth at which the boundary lines are measured. Instead, we estimate Δ_m as a correction to our measured depth δ_m by adding some fraction, $\frac{1}{c}$, of the (unknown) true diameter of the tree.

$$\Delta_m = \delta_m + \frac{D_m}{c} \tag{4.5}$$

where $0 < \frac{1}{c} < \frac{1}{2}$. ($\frac{D_m}{c}$ can be at most the radius of the tree, $\frac{D_m}{2}$: in this case δ_m was measured at the closest point on the tree to the sensor, and the ToF sensor was infinitely far from the trunk so that d_p was an image of the true diameter of the tree.)

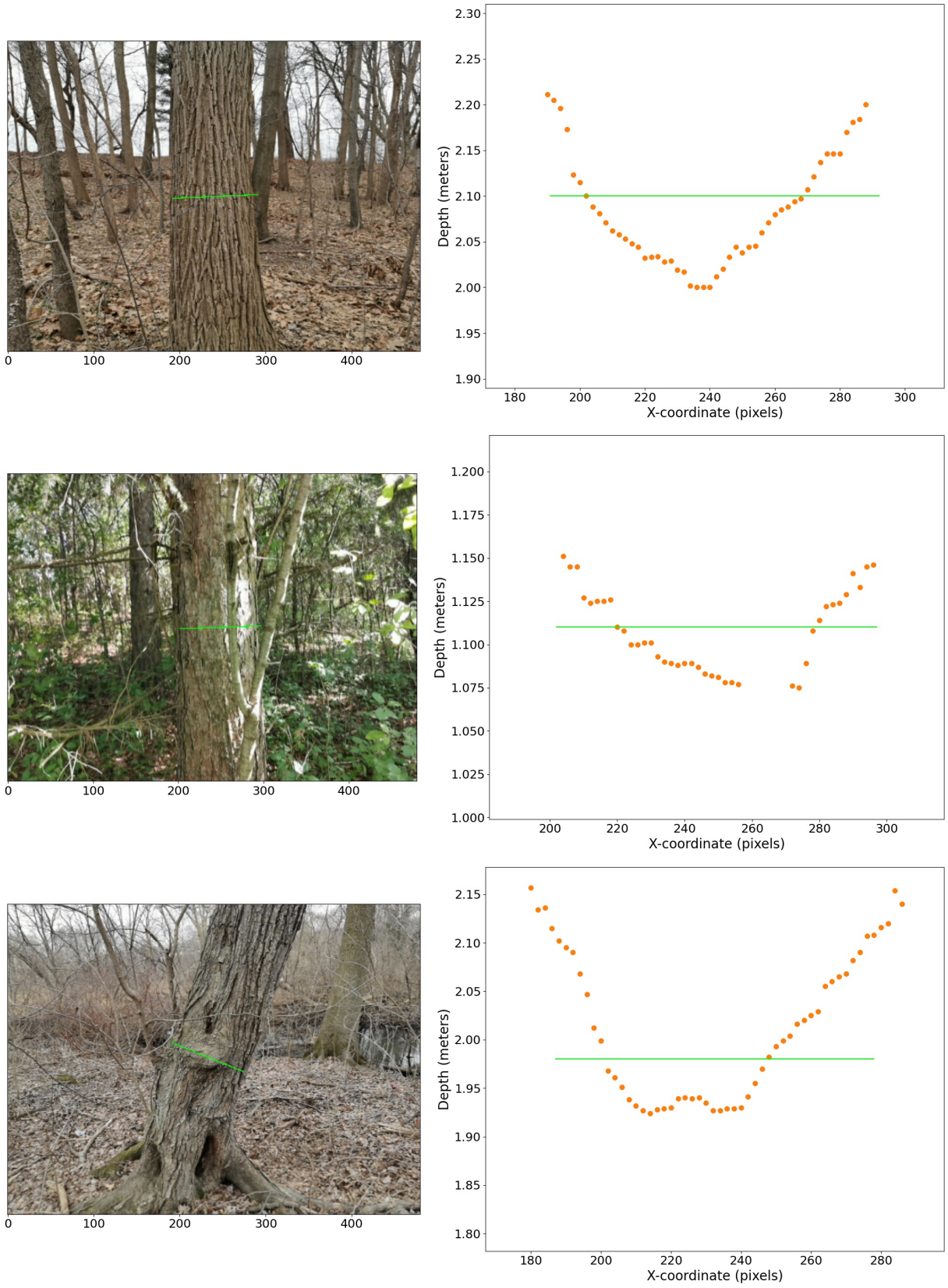


Figure 4.7: Left, RGB images with green line indicating diameter cross-section. Right, cross-section of filtered depth points with green line indicating the width and depth of our estimated diameter. An additional example is shown in Figure 3.3.

Then we rewrite Equation 4.2 in terms of Δ_m, with the approximation discussed above that $d_m = D_m$:

$$D_m = \frac{d_p \cdot \Delta_m}{\gamma_{1_p}} \tag{4.6}$$

Solving these two equations for D_m, the diameter of the tree, we get

$$D_m = \frac{d_p \cdot \delta_m}{\gamma_{1_p} - \frac{d_p}{c}} \tag{4.7}$$

Based on analysis of sample tree cross-sections, we found that in more regularly shaped trunks our estimated depth δ_m tends to lie around halfway between the front of the tree and the approximate depth of the boundary lines. This leads us to set $\frac{1}{c} = \frac{1}{4}$, halfway through the possible range of c.

It is worth discussing the major concern with this equation, namely: what happens when $\gamma_{1_p} - \frac{d_p}{c} \to 0$. If $\gamma_{1_p} = \frac{d_p}{c}$, consider what it means for the physical system. γ_1 is the number of pixels of a 1.0 meter object at one meter away. When $\gamma_{1_p} = \frac{d_p}{c}$, this means some fraction $\frac{1}{c}$ of the observed trunk diameter appears indistinguishable from a 1.0 m wide object viewed by a ToF sensor 1.0 m away. In other words, the full trunk diameter would look like a c meter wide object at a distance of 1.0 m. Even taking c to be as small as possible ($c = 2$), this gives us a trunk diameter of 2.0 m viewed from a distance of 1.0 m – which is to say, the picture we have is indistinguishable from that taken by a ToF sensor placed directly on the trunk.

Alternatively, we could take a much simpler approach, and perform a standard bias correction on our naive diameter approximation from Equation 4.2, multiplying each diameter estimate by a fixed percent. In Section 5.3, we compare the effect of standard bias correction with the approximation proposed in Equation 4.7 on our evaluation results.

4.4 User Assistance

4.4.1 Motivation

The above algorithm has a relatively small computational footprint and can run on the mobile phone directly. This provides a major benefit in comparison to TLS, prior work with SfM, and Katam: the results can be shown to the user in near real-time, allowing

the user to immediately correct any problems without significantly slowing fieldwork. For example, without user feedback, we found that some trunks were not detected, especially small ones ($\leq 8cm$) at a distance $\geq 2m$ away. This is similar to results for SfM reported by Piermattei et al. [32], who find that the average diameter of undetected trunks is 12 cm. Allowing the user to correct problems with undetected trunks does not require any changes to the designed algorithm; we need only modify the app UI to present our results to the user. We discuss this in §4.4.2.

We did not find it necessary to implement additional user interaction in this version of our app beyond presenting our results to the user and allowing the user to re-capture the tree trunk. In the future, though, we might also consider incorporating user input into the algorithm directly, before image capture takes place, for example by allowing the user to tap the intended trunk on the screen. This could replace the first processing step, described in Section 4.3.1, of identifying the approximate trunk depth. The user tap could also be used to select a target trunk component in pre-PCA filtering, described in Section 4.3.2.

4.4.2 App Design

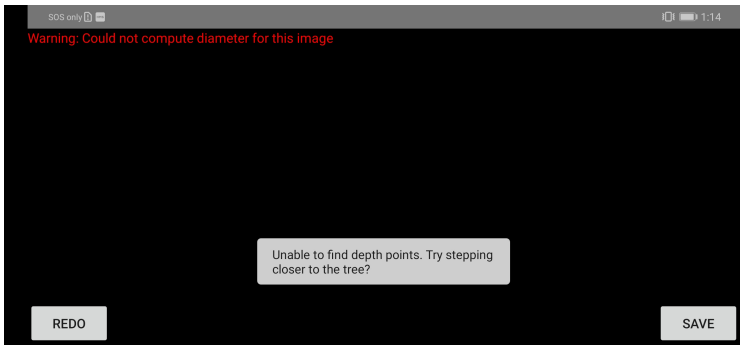
In order to present results to the user and guide them to capture usable images, we design the app as follows.

The main screen shows a continuous preview from the RGB camera, as shown in Figure 4.8a. We considered showing the user the depth map instead, but it was sometimes difficult to understand what the camera was pointing at in the real world based only on the depth map. The preview includes a capture button and two guiding lines splitting the image into thirds, to help the user center the tree trunk in the image. Once the user taps "Capture," the app pins the next RGB and depth image frames in memory and passes them to the processing algorithm. In addition to computing an approximate trunk depth and diameter, the algorithm also returns a visual representation of the results to display to the user. Specifically, it takes the RGB image and overlays the left and right boundary lines, rotated by $90-\theta$ (see Section 4.3.2 for an explanation of this angle). The app displays this result to the user and asks the user to select "Redo" or "Save" depending on whether the algorithm found the trunk boundaries appropriately, as shown in Figure 4.8b.

In addition, errors in the algorithm are diagnosed and displayed for the user. For example, if no depth points are found in the center third of the image, the tree is likely out of range of the ToF sensor. We therefore display a pop-up (the Android term is "Toast") message to the user, asking them to step closer, as shown in Figure 4.8c. Currently, other

(a) Example of main camera preview

(b) Example of algorithm result display



(c) Example of error for phone too far from tree

(d) Example of warning for improper capture

Figure 4.8: Screen captures from the app.

errors are simply shown using a display message asking the user to retake the image, as seen in Figure 4.8d, but one could easily provide error messages with more detailed suggestions for different cases.

4.5 Discussion

In this section, we discuss the assumptions made in the design of our algorithm, some of which may be limitations on the environments in which our system can be used.

- *The tree is within* ± 45 *degrees of vertical*: PCA yields two perpendicular eigenvectors, whose order is not guaranteed to be related to the true principal axis of the trunk. We therefore assume that the correct eigenvector is within 45 degrees of vertical. We could make our design more flexible by testing both eigenvectors and choosing the one whose rotation results in a "plausible" tree image based on existing checks in our algorithm (e.g., the centerline between the two boundary lines contains a high portion of non-zero depth values in I_s), but overall this assumption seems reasonable.

- *The tree does not lean steeply towards or away from the camera*: If the tree is leaning towards or away from the camera, rather than on a plane perpendicular to it, we will not be able to successfully find the orientation of the tree. This will affect the angle of the diameter line and may cause errors fitting a vertical boundary to the trunk. Moreover, we will filter out too many of the true trunk points in the 10% filter. Though it is not straightforward to detect that an image has this problem, we can instruct the user to take pictures in which this is not the case.[5]

- *The trunk is roughly cylindrical*: We assume that the trunk is roughly cylindrical both when fitting boundaries to it and when estimating the DBH, though we can handle some amount of irregularity, such as the large burls found on some of our evaluation samples. The IPCC standard manual measurement techniques also make this assumption. However, we believe that the ideal system should not rely heavily on this assumption, and we discuss extending our system to handle such trees in Section 6.5.

[5]Interestingly, we realized this limitation only after both evaluation datasets had been collected, and none of the images had this problem. It may be somewhat unnatural to stand under or over a steeply leaning tree in order to take a picture of it, though user studies would be required to confirm this.

- *The tree has one trunk*: We only measure one trunk per image. It would be primarily a UI change to allow multiple trunks for a single tree sample, giving the user an option to "add a trunk to this sample" after saving their image of the first trunk.
- *The tree is small enough to fit within camera frame at 2-3 meters away*: At 2 meters away, the camera frame can capture a trunk of roughly 2.7 meter diameter. This nearly three times the maximum tree diameter that we were able to test on. If larger trunks are required, some of the same approaches used to address non-cylindrical trunks could also be used in this context.

4.6 Alternative Design: ARCore SfM

Partway through the development of our work, Google released a Depth API with its ARCore package, giving depth capabilities to Android phones without a ToF sensor. Their Depth API requires only a monocular RGB camera, and uses Structure-from-Motion along with information from the phone's accelerometer and gyroscope to estimate depth as the user moves their phone in relationship to their surroundings [11]. The ARCore Depth API is therefore preferable to Huawei's AREngine for our use case, because it does not require specialized hardware, and is compatible with a much broader range of phones.[6]

In theory, the Depth API is also able to incorporate data from a ToF sensor if one is present on the mobile phone. It therefore offers the possibility of improving our existing system, because SfM complements the weaknesses of a ToF sensor in several key areas. For example, SfM works well in bright sunlight, which is a tricky environment for the ToF sensor [30]. We therefore adapted our app to integrate with ARCore 1.22.0 and compared the two systems. The Depth API is largely standardized between AREngine and ARCore, making it relatively easy to port our code from one software package to the other. We discuss our findings in Section 5.6.

[6] A full list can be found at https://developers.google.com/ar/discover/supported-devices

Chapter 5

Evaluation

We evaluated our work in two distinct forest environments using the Huawei P30 Pro phone. The evaluation section is laid out as follows. In Section 5.1 we discuss the evaluation environments and the procedure we used to carry out our evaluations. In Section 5.2 we present our overall results for each dataset separately and in combination. In Sections 5.3 and 5.4, we consider the impacts of two aspects of our design, namely depth correction and user assistance. In Section 5.5, we examine which variables in our dataset had an effect on the measured error. Finally, in Section 5.6, we discuss the results of our integration with ARCore.

5.1 Evaluation Environment and Procedure

We performed our first evaluation during the summer, in the Laurel Creek and Avon forests of Waterloo, Canada. Both are Carolinian forests [49] with a mixture of broadleaf and conifer species. The former is part of a larger nature preserve, and both are naturally managed forests with multiple northern tree species and age classes. Midsummer leaf-on conditions resulted in significant natural trunk occlusion from leaves and branches across the samples. This forest was relatively young, with measured tree trunks ranging in diameter from 8 - 33 cm. We show sample images from this dataset in Figure 5.1. At this time, the app had no interactive capabilities. Users could see an RGB camera preview on the screen and click "capture," but they were unable to view the captured images or see the results of the algorithm until they had left the site. The algorithm itself was run on a computer, and was not integrated with the phone app. The summertime sample included 28 images of 14 trees, with an image of each tree taken at 1 m and 2 m away.

Figure 5.1: Sample RGB images from the summer dataset, all taken from roughly 2 m away. The images in the left column are categorized as "low" occlusion because they have few to no branches and leaves in front of the trunk. The images in the right column were categorized as "medium" or "high" occlusion. In addition to occlusion, the branches, shrubs, and leaves also make it difficult to walk all the way around the tree in order to obtain enough images to perform SfM.

Our second evaluation was performed in late winter in the Van Cortlandt Park Preserve in New York City. The preserve includes old growth forest and some wetlands, with a diverse array of black oak, sweetgum, red maple, and other, mostly deciduous, native northeastern species [46]. It is under active ecological restoration to remove non-native species such as oriental bittersweet, a vine that strangles native trees, and *Rosa multiflora*, a shrub that grows in large thickets and smothers native plant growth. [44]. Currently, even in the winter, these climbing vines and dense underbrush make for significant trunk occlusion in many areas. The forest is, in places, much older than the Canadian sample, with measured trunk diameters ranging from 6 - 105 cm. We show sample images from this dataset in Figure 5.2. At the time this evaluation was performed, the algorithm was fully integrated with the phone app, running on the phone in real-time and allowing the user to view captured images and results, as well as redo poorly captured trunks. The app also prompted the user to move closer to the tree if necessary. The wintertime sample included 53 images of 29 trunks, with more images per trunk when the first images were rejected by the user. We discuss the specific procedure for saving and re-doing these images below.

In both samples, we established ground truth my measuring the circumference of each tree with a tape measure and to computing the reference DBH to the nearest centimeter, measuring each tree three times and taking the average. For the summertime sample with the non-interactive app, we simply captured two images of each tree, one at 1 m away, and the other at 2 m away. For the wintertime sample with the interactive app, we used the following procedure for capturing images:

1. Stand roughly 1.5 meters from the tree, at a comfortable distance according to the conditions of nearby undergrowth and the need to capture the entire trunk in the frame. The resulting sample included trunks at distances between 1 and 2 meters.

2. Capture one image of the tree, immediately saving the results (even if the algorithm does not appear to have successfully measured the trunk, or warning messages are shown). Note that finding no depth points at all is considered an error in this version of the app, and no image can be saved.

3. Capture additional images of the tree, selecting "redo" instead of "save" until the displayed results showing the computed trunk boundaries appear to line up with the sides of the trunk. If warning messages are shown, follow the on-screen instructions (e.g., step forward, move slightly to avoid occlusion, etc.).

4. If the boundaries do not appear to line up after four capture attempts, simply save the final image.

Figure 5.2: Sample RGB images from the winter dataset, all of which correspond to the last (the "successful") image of a trunk based on user interaction. The images in the left column are categorized as "low" occlusion because they have few to no leaves or branches in front of the target trunk. The images in the right column were categorized as "medium" or "high" occlusion. In dense areas, we have added a red dot to highlight the target trunk. These do not appear in the original image.

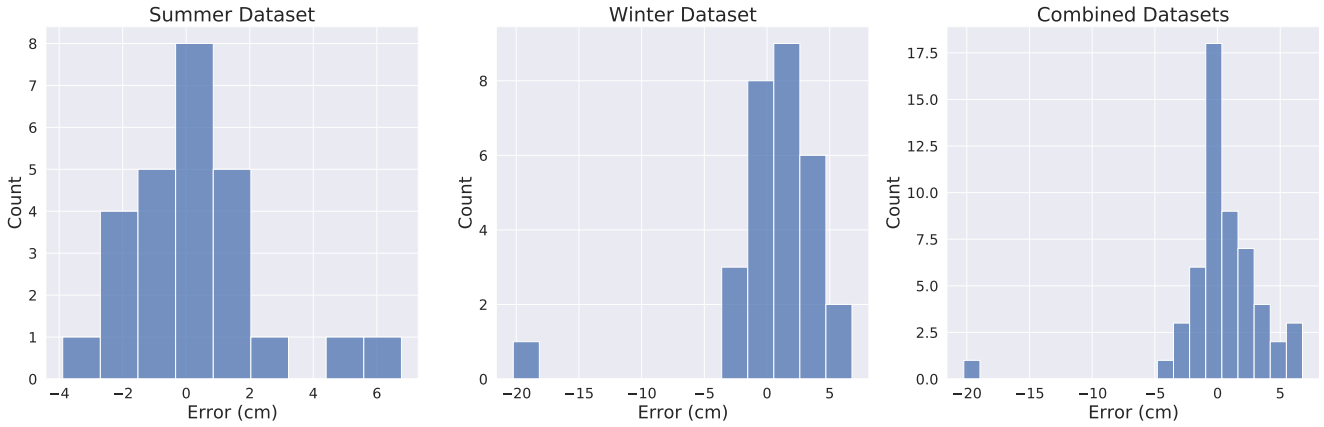


Figure 5.3: Summary of error distribution

The wintertime procedure allows us to compare the results on the same set of trees with and without user assistance, as we do in Section 5.4.

5.2 Summary of Results

The summer sample included 28 images of 14 trees. In one of the samples, the camera was unable to obtain any depth points. This would have been an error in the later version of the app. In another sample, some depth points were captured, but they did not correspond to the tree trunk. This would have been a warning in the later version of the app. As in the evaluation of Structure from Motion by Piermattei et al. [32], we omit these samples from our error evaluation below, and report our detection rate in the sample as 93%.

The winter sample included a total of 53 saved images for 29 trees. For the purposes of this section, we consider only the *last* image saved for each tree – this is the "best attempt" image, which the user believed based on app feedback had successfully captured the trunk. (For an evaluation considering the first attempt for each tree, see Section 5.4.) Our results are therefore reported on 29 images for 29 trees. In no case was the tree undetected; our detection rate among the "best attempt" images was 100%.

In this section, we use the depth correction described in Section 4.3.3. For a comparison to the results without any depth estimate correction as well as a comparison to a standard bias correction, see Section 5.3.

Table 5.1: Summary of error distribution

Dataset	No. Samples	RMSE (cm)	Mean absolute error (cm)	Bias (cm)	Mean % error
Summer	26	2.2	1.5	0.3	8.2
Winter	29	4.7	2.8	0.7	7.6
Combined	55	3.7	2.2	0.5	7.9
Winter*	28	2.8	2.2	1.4	7.2
Combined*	54	2.5	1.9	0.9	7.7

* With the outlier discussed in Section 6.1.3 removed.

In Figure 5.3, we show the distribution of errors in the summer sample, the winter sample, and both samples combined. In our dataset overall, the RMSE was 3.7 cm, with a bias value (mean error) of 0.5 cm. The mean percent error was 7.9%. The RMSE was significantly affected by an outlier in the winter dataset: a 1.04 meter diameter tree with -24 cm of error (23%). We discuss the outlier sample further in Section 6.1.3 and analyze the cause of the error. If we omitted this sample in the combined dataset, the overall RMSE would drop to 2.5 cm, with a bias of 0.9 cm. We show the precise numerical results in Table 5.1.

5.3 Impact of Diameter Estimation Method

In Section 4.3.3, we discussed three possible approaches for estimating the final diameter of the tree, given a segmented image. We first proposed a simple approximation in Equation 4.2:

$$d_m = \frac{d_p \cdot \delta_m}{\gamma_{1_p}} \tag{5.1}$$

We argued that this basic approach would tend to underestimate diameter, with an effect more noticeable in larger trees. We proposed a better approximation in Equation 4.7:

$$D_m = \frac{d_p \cdot \delta_m}{\gamma_{1_p} - \frac{d_p}{c}} \tag{5.2}$$

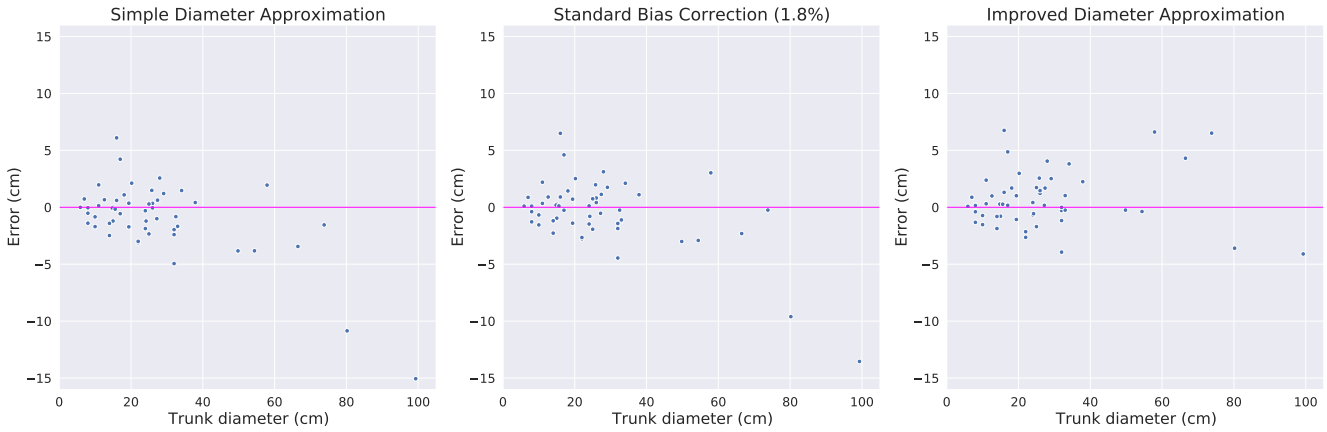


Figure 5.4: Error distribution for different diameter estimation methods[1]

Finally, we noted that the derivation of this second approximation was somewhat complex, and that it might suffice to simply perform a standard bias correction on our original estimate:

$$d_m = \frac{d_p \cdot \delta_m}{\gamma_{1_p}} \cdot (1 + b) \tag{5.3}$$

In Figure 5.4, we show the combined summer and winter results based on each of these three approaches. We omit the outlier sample in these graphs, because the underestimation in that case was primarily caused by the ±10% filter (see §6.1.3 for details) and its presence obscures the overall patterns in the graph. For the standard bias correction, we increase all basic estimates by 1.8%, the mean (signed) percent error of the dataset overall without this outlier.

We observe that the simple diameter approximation, as expected, tends to underestimate trunk diameter across the board, with the effect worsening as trunk diameter increases. A standard bias correction centers the error of smaller trees around zero, but the error still trends downward as the trees get larger. (The choice of bias correction has some effect on the data, but not the overall trend. There are many more small tree samples than large tree samples, so we may consider 1.8% to be an underestimate of the appropriate bias correction. However, increasing the correction to 10% does not change the curve: we

[1]We expect to see the centimeter error widen overall as diameter increases, simply because the tree is getting larger. However, a good diameter estimation method should still roughly center the error around zero.

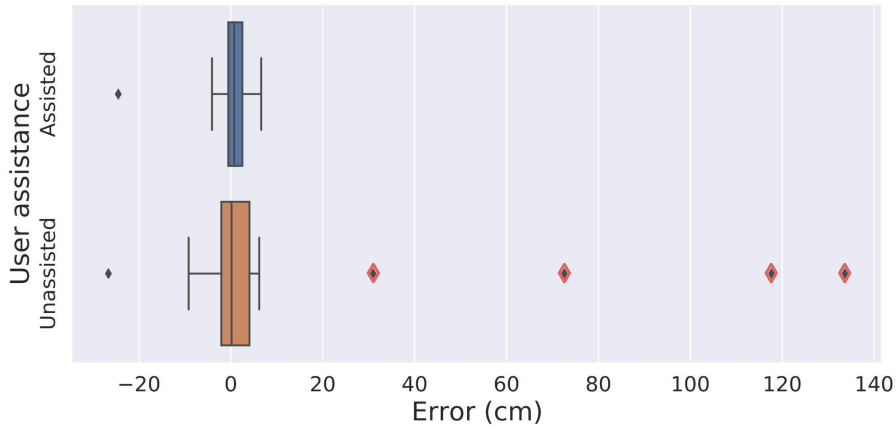


Figure 5.5: Error distribution with and without user assistance in the winter dataset. The box lines show the 25^{th}, 50^{th}, and 75^{th} percentiles of the distribution. The plot whiskers show 1.5 times the interquartile range, and outliers are data points that lie outside of this range. Outliers highlighted in red are those in which the algorithm failed to find the trunk.[2]

then tend to overestimate the diameter of the smaller trunks, while still slightly underestimating the diameter of the largest trunks.) The second diameter approximation, on the other hand, manages to appropriately "straighten out" the error, centering it around zero on both large and small trees.

5.4 Impact of User Assistance

We have two ways of considering the importance of user assistance in our designed system. First, we compare the results in the summer and winter datasets, since the summer dataset was taken using a version of the app that did not allow the user to even view captured images. In the summer dataset, the tree detection rate was only 93%, while in the winter

[2]We consider the algorithm to have failed to find the trunk when at least one of the identified boundary lines does not correspond to the trunk in any way. This happens either when it corresponds to the boundary of another tree in the image, or because the threshold for stopping the iteration of vertical scan lines (see §4.3.2) was never reached. In the latter case, the boundary line corresponds to the left or right edge of the image. Such mistakes result in an enormous estimate for tree diameter; hence the errors of up to 140 cm. If user assistance had not proven capable of eliminating such errors entirely, we might have focused on reducing these errors (e.g., by adjusting the algorithm to return 0 diameter when certain sanity checks fail, so that the error is at least bounded by the size of the tree). We instead rely on user assistance, and do not make these adjustments.

dataset the system achieved 100% detection. While the RMSE was lower in the summer than the winter (2.2 cm vs 4.7 cm), the trees measured were also much smaller (with maximum diameter of 33 cm vs 105 cm), and the mean *percent* error was much more similar (8.2% in summer vs 7.6% in winter). Moreover, the winter dataset mean was heavily affected by a single large tree that exposed a weakness in our algorithm (see Section 6.1.3). Dropping that sample, the summer and winter datasets had more similar RMSEs, and the summertime percent error remained only slightly higher than winter (2.2 cm, 8.2% in summer vs 2.8 cm, 7.2% in winter).

However, the conditions and forests between winter and summer samples were also dissimilar, so there are other confounding variables in the comparison. In Figure 5.5 we compare the results for each winter tree measurement between the first captured image and the last. In 8 of 29 trees (28%), the first image was considered a satisfactory measurement – in these cases, the first and last captured image are the same. We can see that the 25^{th} and 75^{th} percentile error bars are slightly wider in the unassisted sample, but the most important impact of user assistance is in improving trunk detection. In the unassisted set, 4 images had well over 20 cm of error because the tree trunk was not found at all or an incorrect object was identified as the trunk. In the assisted set, the tree trunk was found in all images; the image with around -24 cm of error was the outlier sample in which the trunk was correctly identified but its diameter estimated incorrectly. User assistance also gives us the opportunity to improve even this sample in future work, analyzing what went wrong to provide better guiding messages to the user.

5.5 Variables Affecting Error

In this section, we discuss the impact of several variables on measurement error. All graphs are shown for the combined winter and summer datasets using diameter correction (see Section 5.3). In the case of the winter dataset, we use the last image saved for each tree (i.e. fully user assisted: see Section 5.1).

5.5.1 Trunk Diameter

We examined whether the trunk diameter was correlated with the measurement error. First, we considered the correlation between trunk diameter and and centimeter error. We expect to see some correlation here, with centimeter error increasing simply because the tree itself is larger, and indeed a Pearson correlation coefficient test found moderate correlation

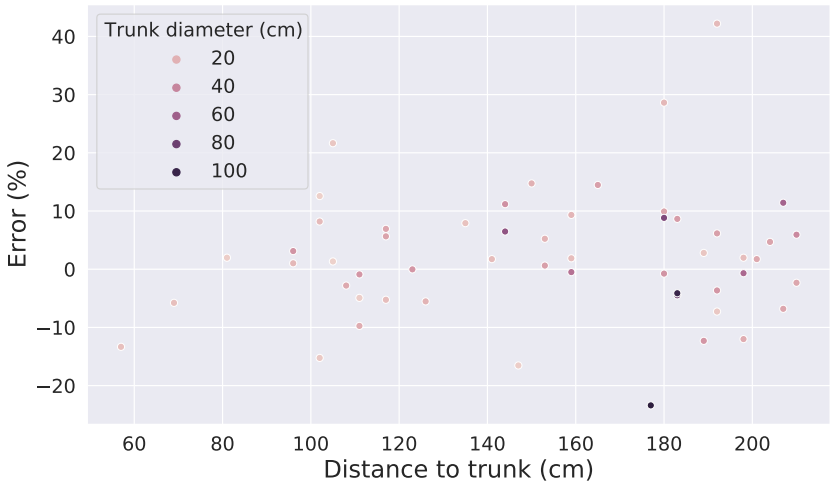


Figure 5.6: Error distribution vs distance to tree trunk, colored by approximate trunk diameter.

($r = -0.38$). (For the original diameter estimate with no correction, the Pearson test found strong correlation, $r = -0.73$.) Next, we considered the correlation between trunk diameter and (signed) percent error. Here, a Pearson test found a weak linear correlation between percent error and trunk diameter ($r = -0.17$).

5.5.2 Distance to Trunk

We also considered whether the distance at which the image was taken has an effect on measurement error. We note that trunk diameter may be a confounding variable here, since the user may need to stand further from large trunks in order to fit the girth of the tree into the frame. However, a Pearson test shows weak linear correlation for both centimeter and (signed) percent error ($r = 0.02$ and $r = 0.12$, respectively). We show the centimeter error distribution in Figure 5.6, which is colored by trunk diameter.

5.5.3 Degree of Occlusion

Finally, we categorized our images by the degree of occlusion (low, medium, or high). We had 16 images categorized as "low," with little to no occlusion; 25 images categorized as "medium," with a moderate amount of occlusion; and 13 images categorized as "high", with

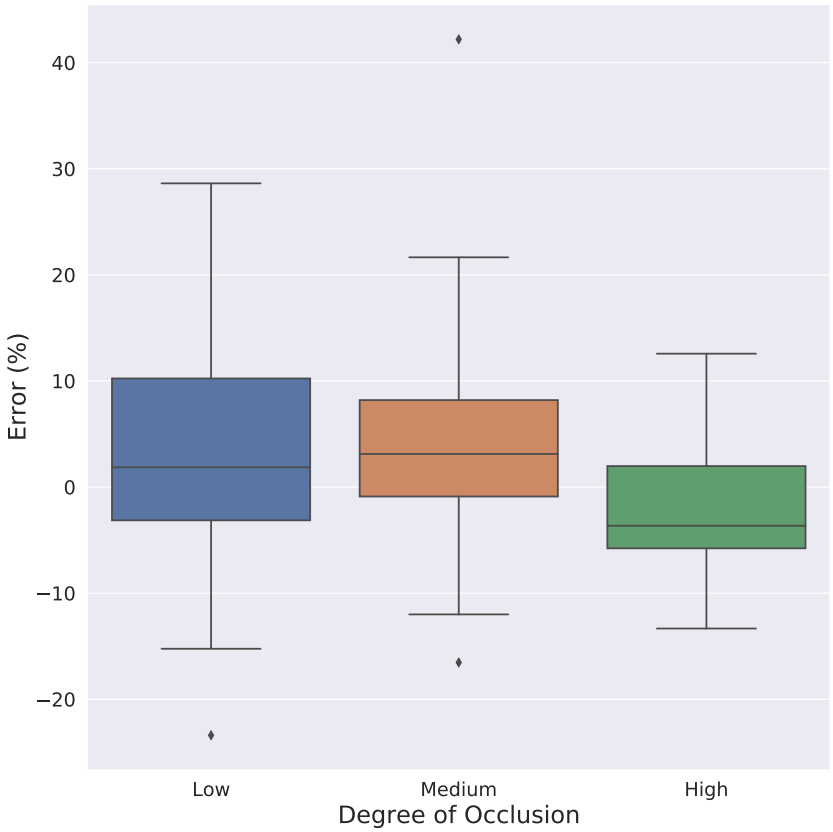


Figure 5.7: Error distribution categorized by level of image occlusion. The box lines show the 25^{th}, 50^{th}, and 75^{th} percentiles of the distribution. The plot whiskers show 1.5 times the interquartile range, and outliers are data points that lie outside of this range.

severe occlusion. A one-way ANOVA test on this small sample did not find a statistically significant difference in the mean error of the three groups ($F = 1.51, p = 0.23$). However, given the subjectivity of classifying the images and the relatively small number of samples in each group, additional evaluation is needed. We show side-by-side percent error boxplots for each category in Figure 5.7.

5.6 ARCore Depth API Evaluation

As discussed in Section 4.6, we also adapted our app to integrate with Google's ARCore Depth API, to attempt to make our system available on an even broader range of phones. In this section, we describe our ARCore tests and findings.

5.6.1 Testing Setup

The ARCore Depth API offers only two configuration options: AUTOMATIC, which opaquely combines SfM and the ToF sensor, if one is present; and DISABLED, which turns off all depth data. As a result, we were unable to compare ARCore SfM (alone) to the ToF sensor (alone) on the same Huawei device. Instead, our three test systems were:

1. Huawei P30 Pro using AREngine (depth sensor alone)
2. Huawei P30 Pro using ARCore (ToF sensor and SfM automatically and opaquely combined)
3. Pixel 4a using ARCore (SfM alone, as the Pixel 4a has no ToF sensor)

For a basic test, we took images of completely unoccluded trees at distances of 1.5 - 2 m.

5.6.2 Findings

ARCore begins with a blurry uniform depth image, gradually improving its per-pixel depth estimates over time as the user moves the phone. We found that this tends to leave shadowy artifacts extending well beyond the boundary of the tree that do not seem to correspond to real depth values. In Figure 5.8, these artifacts can be seen as blue shadows to the left and right of the trunk in the ARCore results (top two images) in the second column. The right-side artifacts may correspond to the smaller trunk in the background, but it is not clear what the artifacts on the left side of the trunk represent. Additionally, though ARCore continually refines its depth estimates, it does not provide a signal to the developer as to when the estimates could be considered "good enough" for a given precision or confidence.

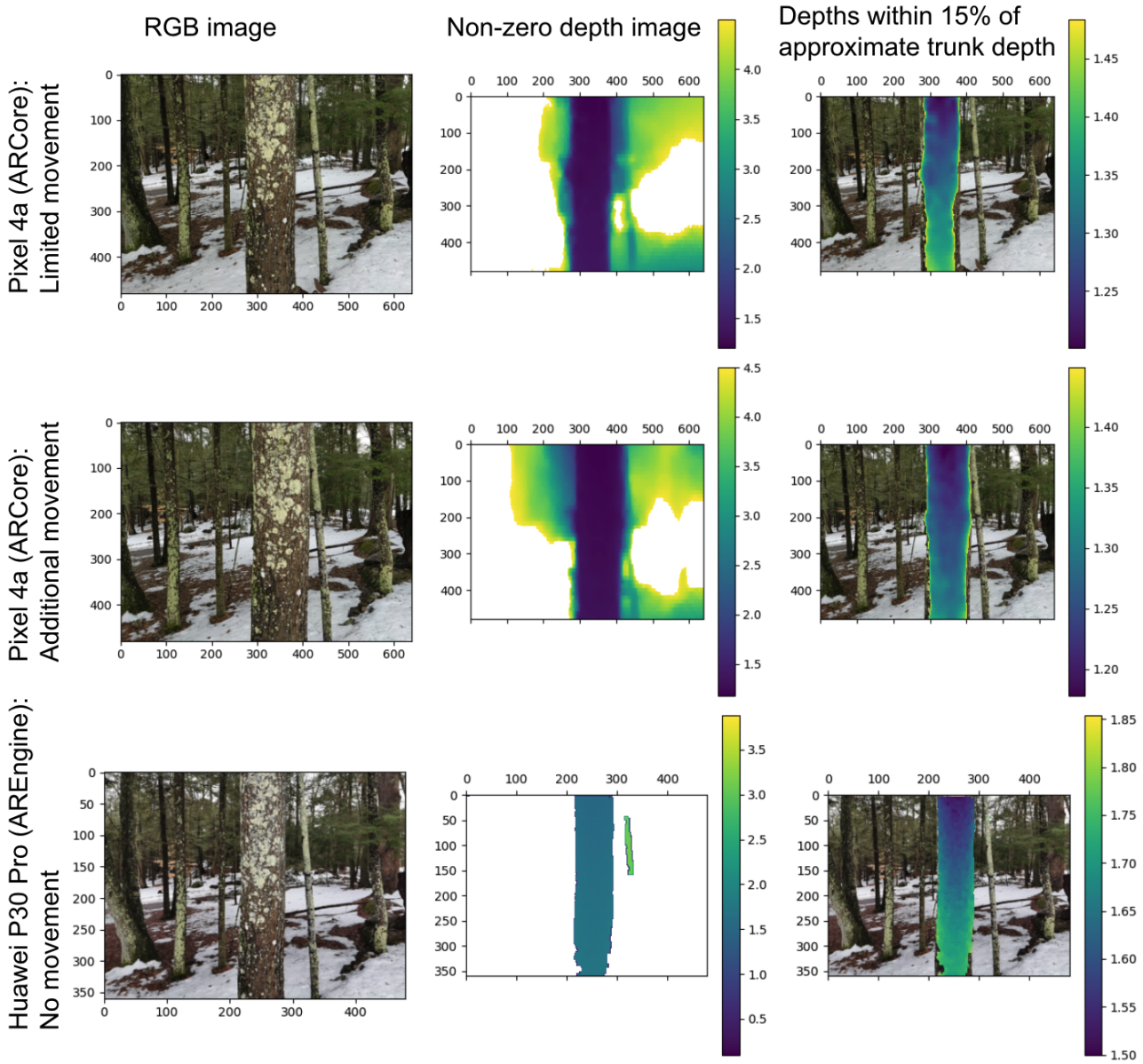


Figure 5.8: Sample images from ARCore and AREngine. "Limited movement" indicates moving the phone left, right, up, and down slowly one time. "Additional movement" indicates moving the phone around in all directions, including walking a few steps, until capturing an image of the trunk with our app appears to show roughly accurate tree diameter lines.

The DEPTH16 image format provides 3 bits per pixel to express a confidence value, but as of the time of writing, ARCore always sets these bits to 0, indicating 100% confidence.[3][4]

Moreover, we found that even when a ToF sensor was available, the data did not appear to be used to its fullest extent. The most obvious indication of this was that ARCore consistently provided lower resolution than the ToF sensor, even when the ToF sensor was available. The returned image was 120×160 on the P30 Pro and 90×160 on the Pixel 4a, in comparison to 180×240 available resolution for the ToF sensor. In addition, in brief tests indoors, we found that ARCore does not seem to use the ToF sensor to overcome known limitations of SfM (e.g., providing a detailed depth map on white walls, in low lighting, for moving objects, or before the phone moves). We did not find the P30 Pro using ARCore to be a noticeable improvement over the Pixel 4a.

Overall, we determined that we cannot yet use ARCore for our proposed application. The most pressing problem is the continual refinement of depth estimates. The developer has no signal as to the quality of the depth estimates and as a result cannot incorporate this information into a processing algorithm or guide the user to move the phone to improve the quality of the image. For our use case, the continual refinement manifests as the object appearing to "grow" over time as more edge depths are added. This can be seen in Figure 5.8 by examining the ARCore results (top two images) in the third column. With additional movement, the depth values slowly "fill in" towards the edges of the tree, compared to the image captured with limited movement. This is not suitable for measuring trees, since we cannot distinguish between a small tree and a low-quality image. A confidence value would be necessary for integration.

We have reported our findings to the ARCore team and have filed bugs where appropriate for minor Depth API issues.

[3]This behavior is documented at https://developers.google.com/ar/reference/java/com/google/ar/core/Frame#acquireDepthImage()

[4]Huawei's confidence values are not usable for this purpose either (see Section 4.2), but we were able to achieve good results without them.

Chapter 6

Discussion

In this chapter, we discuss our proposed system and algorithm in further detail. In Section 6.1, we discuss the strengths and weaknesses of our algorithm, devoting particular attention to analyzing the cause of the major outlier in the winter dataset. In Section 6.2, we discuss data collection time and ease of use, and in Section 6.3, we compare our work to the prior, closely related work of Piermattei et al. [32] and Fan et al. [13]. Finally, in Section 6.5, we offer conclusions and suggestions for future work.

6.1 Strengths and Weaknesses

6.1.1 Examples of correct performance

One area of the Van Cortlandt Forest, where we collected the winter dataset, suffered severe overgrowth from *rosa multiflora*. *Rosa multiflora* is considered an invasive species in eastern North America, and in this area it covered the ground densely with thick thorny stems. Even in wintertime leaf-off conditions, this made it difficult to move around in the area. We show images of the conditions in Figure 6.1. We attempted to measure this section of woods with Katam (see Section 3.3.3), and Katam was unable to identify a single trunk. Our system was not only able to capture all four of the trees in this area, but it also allowed us to measure the trees from further away than manual measurements did, reducing the challenges of reaching the most densely overgrown areas and speeding up measurement.

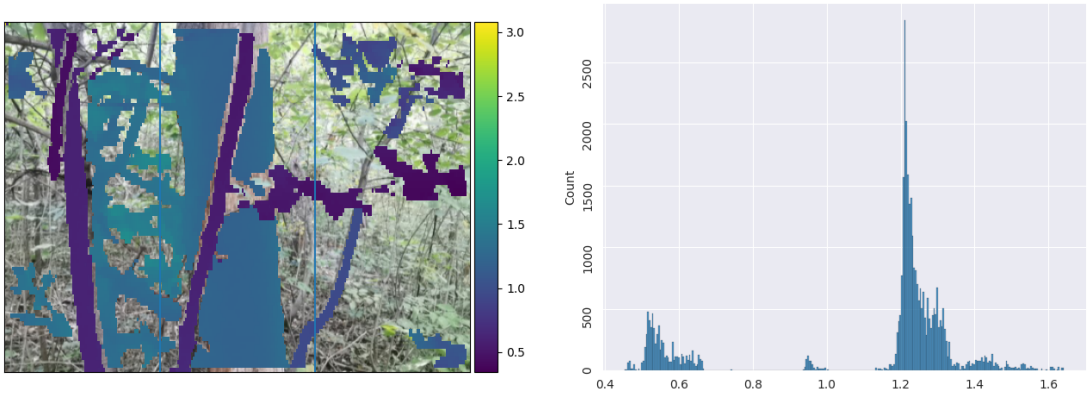
Figure 6.1: Two sample RGB images of trunks in an area with dense undergrowth.

6.1.2 Examples of incorrect performance

Our algorithm does not always successfully filter out occlusion. In some cases, it performs poorly when there are large occluding objects in the foreground, such as large leaning branches or smaller trunks. We show an example of processing such an image in Figure 6.2. A smaller trunk splits the target trunk into multiple disconnected components in the depth image, as seen in subfigures 6.2c and 6.2d. Due to large gaps are between these components, the algorithm removes one of the components of the tree from I_f during its more aggressive filtering stage, as seen in subfigure 6.2e. While this does not affect which points are included in the final search for trunk boundaries, it causes the algorithm to incorrectly rotate the trunk based on only a subset of its depth points. It is worth noting that this example comes from the summer dataset. We did not see this problem in the winter dataset's user-assisted samples, because it was clear during data collection that the app had incorrectly measured the trunk, and the user could move slightly to the left or right and redo the image. In the summer dataset, a second image of the same tree in Figure 6.2, taken from further away, did not have the same problem.

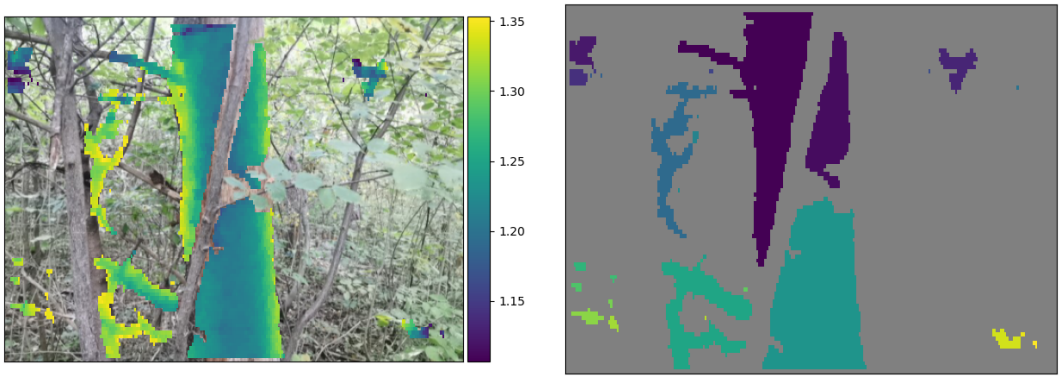
6.1.3 Notable Outlier

There was one significant outlier in our dataset: a 1.04 m diameter tree in Van Cortlandt forest. We show the processing of this image in Figure 6.3. The trunk was the largest measured in our dataset (though the second-largest came close at 99 cm), and it had



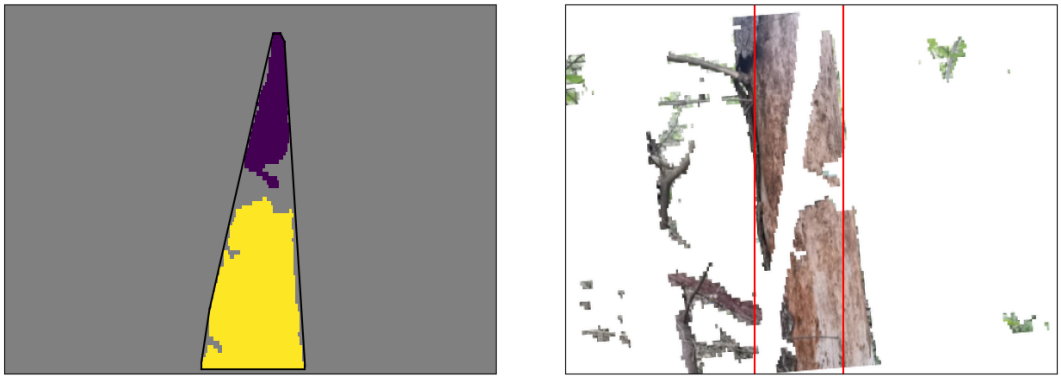
(a) Original image with depth overlaid

(b) Histogram of depth values



(c) I_s, roughly segmented image

(d) Connected components of I_s



(e) I_f, filtered image with fitted convex hull

(f) Final trunk boundaries based on I_s

Figure 6.2: Example of weak performance on a tree "split" by an occluding stem.

No. trees	Manual (s)	App (s)	Speedup
3	246	58	4.2x
2	119	35	3.4x
2	100	37	2.7x
2	179	39	4.6x
3	255	73	3.5x
2	129	38	3.4x
3	189	61	3.1x

Table 6.1: App measurement speedup

several irregularities unusual in smaller trunks. It had four large burls on the right side of the trunk, and curved sharply away from the camera on the left, as can be seen in subfigure 6.3b. The first step of our proposed algorithm, filtering for depths within ±10% of the mode trunk depth, was not designed based on examples of large, irregular trunks, and proved too crude for the purpose. We can see in subfigure 6.3d that one of the burls and a long section on the left side of the trunk were outside of this strict range. This threw off both the rotation of the image and the estimation of trunk boundaries, resulting in a severe under-estimation of the trunk width (by 24 cm, or 23%).

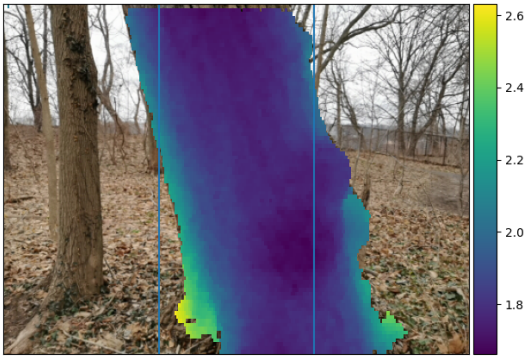
In order to handle large, irregular trunks like this one, a more flexible first step of the algorithm is required. For example, we might consider incorporating the RGB image into the initial (rough) segmentation, which we did not otherwise find necessary. Alternatively, we could search specifically for edges in the depth image, rather than depths within a particular range. Exploring such alternatives is a highly appropriate direction for future work.

6.2 Data Collection Time and Ease of Use

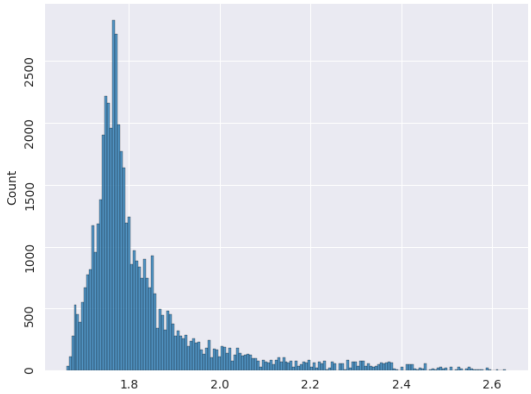
One key strength of our designed system is the speedup for measuring trees in the field. In a separate trip for wintertime data collection in Van Cortlandt Forest,[1] we timed the manual and app-based measurement of trees in sets of two to three nearby trees at once, and found that the app reduced measurement time by up to a factor of 4.6, as shown in Table 6.1. Indeed, we may achieve an even more significant reduction with a production

[1]This data set was unfortunately unusable for DBH evaluation due to a bug in the app that did not affect timing data.

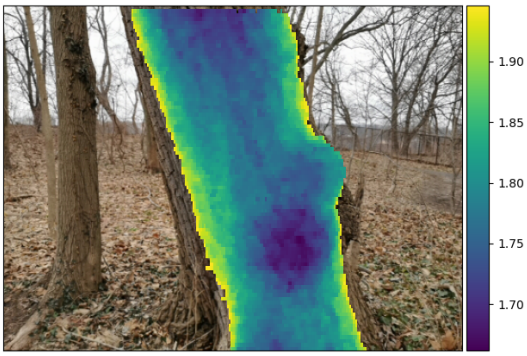
(a) Original RGB image



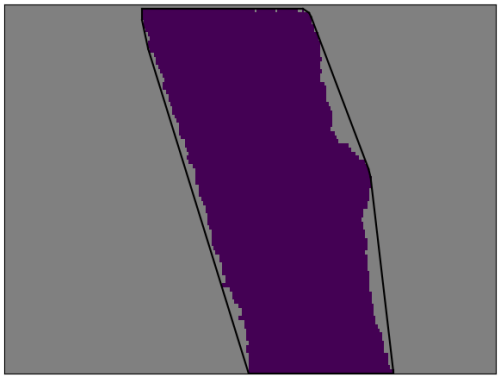
(b) Original image with depth overlaid



(c) Histogram of depth values



(d) I_s, roughly segmented image



(e) I_f, filtered image with fitted convex hull

(f) Final trunk boundaries based on I_s

Figure 6.3: Processing steps for the significant outlier in the winter dataset.

Figure 6.4: Left, a small stand of cedars. Right, image taken by phone held just under the tree canopy.

system measuring many trees consecutively. Some of the time saved was in avoiding walking through underbrush from one tree to the next, since the user could traverse less distance and stand in more convenient locations when using the app. When timing the measurement of more trees consecutively, we may see a more significant effect.

There was one particular sample that highlighted the ease and efficiency of our system: measuring a small stand of northern white cedars (*Thuja occidentalis*) in the Van Cortlandt Forest. The cedars are depicted in Figure 6.4. The trunks are impossible to see or reach directly from the exterior of their canopies, meaning that in order to measure their circumference manually, we had to find an appropriate gap in the branches and crawl underneath to reach around the trunk with a tape measure. By contrast, with the designed app, we simply held the phone just inside the outer canopy and captured the trunk within. The measurement of two cedars took around 2.5 minutes manually (by a relatively small individual, once an appropriate crawlspace near the ground had been identified). It took less than 30 seconds with the app (three images of the first cedar were required to get a successful measurement, since the trunk was so heavily occluded) and required no search

Plot difficulty	Detection rate (%)	Mean DBH of undetected trunks (cm)
Easy	98	9.7
Difficult	65	17.7
Medium	83	12.6
Difficult	91	7.5

Table 6.2: Trunk detection rate reported by Piermattei et al. [32]

for a suitable place to approach the trunk.[2]

6.3 Comparison to Prior Work

We now discuss our work and results in comparison with that reported in the literature for the most closely related work: SfM from Piermattei et al. [32] and the Google Tango system by Fan et al. [13]. We were unfortunately unable to redo the evaluation of their systems on our own forest plots, due to the inability to obtain their source code or measurement tools despite requesting it. While we are unable to make a head-to-head comparison, we find that overall our results are comparable to theirs, with an RMSE of a few centimeters and small bias. Across four different plots, Piermattei et al. report RMSEs between 1.2 - 5.1 cm, and biases between -2.1 - 0.7 cm. Fan et al. report an RMSE of 1.26 cm and a bias of 0.33 cm.

There are two key differences between the data sets used by Piermattei et al. and Fan et al. and our own. First, they both evaluate their systems in an environment with minimal occlusions, with Fan in particular showing photographs taken immediately in front of an entirely unoccluded tree. Piermattei et al. include two plots that they classify as "difficult" based on the degree of occlusion and trunk density. In one of these "difficult" plots, their trunk detection rate falls to 65%, with a 17.7 cm average DBH for undetected trunks. The detection rate and average DBH of undetected trunks for each plot evaluated in their paper is shown in Table 6.2. Second, Piermattei et al. and Fan et al. only consider much smaller trees than those in our dataset. The maximum trunk diameter evaluated by Piermattei et al. was 63.9 cm, while the maximum trunk diameter from Fan et al. was

[2]We timed the measurement of these two trees during the collection of the winter dataset discussed in §5.1. They are not included in the timing-only data collected in Table 6.1.

only 34.5 cm. Even for trees in this small diameter range, Fan et al. already report the effect of increased negative bias for larger trees [13].

Finally, while Fan et al. do not report information about the processing or data collection time for their system, Piermattei et al. report that in their "difficult" plot, image acquisition requires more time than manual DBH measurement due to the need to acquire many additional images, and post-processing requires between 22 hours and 85 hours depending on the computer used [32].

We also compare our work to two solutions from non-academic sources: generic measurement apps and Katam [22]. In both of the generic apps we tested, CamToPlan [43] and Google Measure [16], multiple measurements of the *same* 25 cm diameter tree differed by as much as 16 cm (CamToPlan) and 42 cm (Google Measure), barring them from consideration for a more comprehensive evaluation. We conclude that these apps are not sufficiently robust to use in a forest context at this time. Katam, as discussed above, may offer a strong option in a production-oriented forest. However, it does not advertise itself for use in forest environments with occlusion. Indeed, in our tests, it was unable to find any trees when dense understory was present, and in clearer areas it sometimes added trees where none existed, with no way to correct the problem.

6.4 Ground Truth and Error Measurement

We report our error measurements according to the conventions in the literature [13][32][50], documenting the RMSE, bias, and percent error of our DBH measurements as compared to those found by a manual forest inventory. However, it is worth discussing the limitations of such a comparison and placing these per-tree DBH errors into the broader context of forest carbon measurement. Manual measurements of tree DBH are not "ground truth" in the traditional sense of the word. Trees do not have a single true diameter, even at a specific height, with standard manual methods for approximating DBH assuming that the tree trunk is roughly cylindrical.

Manual measurements are still a meaningful baseline for new work, however, because they form the basis of established allometric equations that relate DBH to estimates of biomass and stored carbon. These allometric equations are statistical relations between easily observable features such as DBH (as measured by manual methods) and aboveground biomass (as measured by destructive harvesting: cutting down and weighing the tree). To serve as a useful input to allometric equations, then, our work seeks to approximate the measurements on which those equations were calibrated.

Several studies have attempted to quantify the error in allometric equations, for example by quantifying the relative contributions of *prediction error* (error associated with uncertainty in the estimated parameters of an allometric equation, as well as the variance in biomass between trees with the same measured attributes) and *measurement error* (error associated with uncertainty in the measurements of tree attributes) [7][47]. Measurement error for manual measurements is related to the variance of these measurements; in a future evaluation of our system we therefore recommend measuring and reporting on the variance of our measurements across multiple samples of the same tree. If this variance falls within the accepted variance of manual measurements, it would provide additional evidence of the reliability of our DBH estimates as an input to allometric equations.

6.5 Future Work

There are several promising avenues for future work. First, there are opportunities to improve our existing app and algorithm in order to make them more robust to different forest conditions. As discussed in Section 6.1.3, the initial filtering step for depths with ±10% of the estimated trunk depth does not perform well on large and substantially irregular trunks. Future work may seek to improve the robustness of this step by incorporating information from the RGB image or from edge-finding algorithms (on either the RGB or depth image).

Entirely new approaches to measuring the trunk from an RGB+D image, such as using neural networks for more advanced image segmentation, should also be explored. In this work, we designed a relatively simple segmentation technique that worked well overall. However, based on their success in similar tasks, we believe that convolutional neural networks would also be well-suited to identify tree trunks in RGB+D images. Of course, this approach would require a large, labeled image dataset that would take substantial work to create, but it could yield better results.

In addition, there are opportunities to make forest inventory data sets richer than those that can be collected by hand. In a manual forest inventory, it is only feasible for the surveyor to record a few data points per tree (DBH, height, species, position). Though our app does yield RGB and depth images, the ToF sensor paired with an RGB camera is also capable of recording a complete RGB+depth point cloud as the user moves around the tree, which could be far more informative. Existing AR technology running on mobile phones is able to create such continuous 3D maps as the user moves, and there are many opportunities to use such a rich dataset. For example, several of the limitations discussed in Section 4.5 could be overcome if the user obtained a 3D point cloud representation that

could flexibly deal with multiple trunks (as with yew trees) or non-cylindrical buttressed trunks (as with banyans). There is some trade-off here, though, since a dense 3D point cloud would require more than one image per tree, taken from different locations around the trunk.

Beyond obtaining better measurements of the tree trunk, the final integration step to obtain a per-tree carbon estimate is to deduce the species or general type of tree, in order to select the appropriate allometric equation. The simplest approximation could be obtained by using the phone GPS to estimate the general location of the forest and selecting an allometric equation suited to trees of that region. For more precision, the RGB photographs could be used to detect tree species. Reference [25] has already investigated identifying tree species from individual photos of leaves or bark using neural networks, and that work could be integrated with our own to provide a complete carbon estimate.

Moreover, with precise position information the terrestrial perspective from the phone could be correlated with other data sources, such as satellite or aerial data, to provide a more complete picture of the forest as a whole: its range, canopy, and understory. Doing so would require precise geographic coordinates as the user moves throughout the forest, which is not straightforward with a dense canopy overhead limiting GPS signal strength.

6.6 Conclusions

In this work, we examine the use of smartphones equipped with depth sensors to estimate the trunk diameter of trees in forest plots. In the natural forest context, trunk images can be occluded, lighting conditions can be challenging, and it may not be easy to walk all the way around each tree. Unlike manual processes or prior work based on SfM or TLS, our approach is fast, low-cost, and allows real-time user interaction. Unlike previous research into using mobile phones, our work considers dense undergrowth, large trees, and natural occlusion as a primary use case. We incorporate our algorithm into an interactive mobile phone app and evaluate our system in realistic, naturally-managed forest settings. We find that in a corpus of 55 sample tree images, it estimates trunk diameter with mean error of 7.8%. This is comparable to the results achieved by prior approaches but, unlike prior work, our solution is capable of getting results in a dense, leafy understory. Finally, recognizing that even a depth sensor is somewhat specialized hardware, we port our app to Google's ARCore Depth API, which makes use of only a monocular RGB camera. We find that this system is not yet robust enough for a forest MRV context and offer specific suggestions for its improvement.

Overall, we believe that our proposed system is a promising direction for research in the use of sophisticated smartphone technologies for the important problem of monitoring, reporting, and verification of tree diameters in forest plots. Moreover, we believe that future research into mobile AR systems such as ARCore should consider outdoor and even forested environments, as well as object measurement, as a primary use case.

References

[1] R. Adams and L. Bischof, "Seeded region growing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641–647, Jun. 1994, ISSN: 1939-3539. DOI: 10.1109/34.295913.

[2] T. Ayari and N. Radufe, "Sony's 3D Time of Flight Sensing Solution in the Huawei P30 pro," SystemPlus Consulting, Nantes, France, Tech. Rep. SP20518, Feb. 2020. [Online]. Available: https://www.systemplus.fr/wp-content/uploads/2020/02/SP20518-Huawei-P30-pro-ToF-Sensor-sample.pdf (visited on 05/07/2021).

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483, 1996.

[4] S. Bauwens, A. Fayolle, S. Gourlet-Fleury, L. M. Ndjele, C. Mengal, and P. Lejeune, "Terrestrial photogrammetry: A non-destructive method for modelling irregularly shaped tropical tree trunks," *Methods in Ecology and Evolution*, vol. 8, no. 4, pp. 460–471, 2017, ISSN: 2041-210X. DOI: 10.1111/2041-210X.12670. [Online]. Available: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.12670 (visited on 09/17/2020).

[5] G. Bradski and A. Kaehler, "OpenCV," *Dr. Dobb's journal of software tools*, vol. 3, 2000.

[6] A. Burt, M. Disney, and K. Calders, "Extracting individual trees from LiDAR point clouds using treeseg," *Methods in Ecology and Evolution*, vol. 10, no. 3, pp. 438–445, 2019, ISSN: 2041-210X. DOI: 10.1111/2041-210X.13121. [Online]. Available: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13121 (visited on 06/02/2020).

[7] Q. Chen, G. Vaglio Laurin, and R. Valentini, "Uncertainty of remotely sensed aboveground biomass over an African tropical forest: Propagating errors from trees to plots to pixels," *Remote Sensing of Environment*, vol. 160, Feb. 2015. DOI: 10.1016/j.rse.2015.01.009.

[8] D. A. Coomes, M. Dalponte, T. Jucker, G. P. Asner, L. F. Banin, D. F. R. P. Burslem, S. L. Lewis, R. Nilus, O. L. Phillips, M.-H. Phua, and L. Qie, "Area-based vs tree-centric approaches to mapping forest carbon in Southeast Asian forests from airborne laser scanning data," *Remote Sensing of Environment*, vol. 194, pp. 77–88, Jun. 2017, ISSN: 0034-4257. DOI: 10.1016/j.rse.2017.03.017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0034425717301098 (visited on 09/21/2020).

[9] M. Disney, M. Boni Vicari, A. Burt, K. Calders, S. Lewis, P. Raumonen, and P. Wilkes, "Weighing trees with lasers: Advances, challenges and opportunities," *Interface Focus*, vol. 8, p. 20 170 048, Apr. 2018. DOI: 10.1098/rsfs.2017.0048.

[10] M. Disney, "Terrestrial LiDAR: A three-dimensional revolution in how we look at trees," *New Phytologist*, vol. 222, no. 4, pp. 1736–1741, 2019, ISSN: 1469-8137. DOI: 10.1111/nph.15517. [Online]. Available: https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/nph.15517 (visited on 05/20/2020).

[11] R. Du, E. Turner, M. Dzitsiuk, L. Prasso, I. Duarte, J. Dourgarian, J. Afonso, J. Pascoal, J. Gladstone, N. Cruces, S. Izadi, A. Kowdle, K. Tsotsos, and D. Kim, "DepthLab: Real-time 3D Interaction with Depth Maps for Mobile Augmented Reality," in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, Virtual Event USA: ACM, Oct. 2020, pp. 829–843, ISBN: 978-1-4503-7514-6. DOI: 10.1145/3379337.3415881. [Online]. Available: https://dl.acm.org/doi/10.1145/3379337.3415881 (visited on 02/12/2021).

[12] G. Fan, F. Chen, Y. Li, B. Liu, and X. Fan, "Development and Testing of a New Ground Measurement Tool to Assist in Forest GIS Surveys," *Forests*, vol. 10, no. 8, p. 643, Aug. 2019. DOI: 10.3390/f10080643. [Online]. Available: https://www.mdpi.com/1999-4907/10/8/643 (visited on 02/08/2020).

[13] Y. Fan, Z. Feng, A. Mannan, T. U. Khan, C. Shen, and S. Saeed, "Estimating Tree Position, Diameter at Breast Height, and Tree Height in Real-Time Using a Mobile Phone with RGB-D SLAM," *Remote Sensing*, vol. 10, no. 11, p. 1845, Nov. 2018. DOI: 10.3390/rs10111845. [Online]. Available: https://www.mdpi.com/2072-4292/10/11/1845 (visited on 02/08/2020).

[14] Google, LLC, *Android.hardware.camera2*. [Online]. Available: https://developer.android.com/reference/android/hardware/camera2/package-summary (visited on 10/29/2020).

[15] ——, *ImageFormat*. [Online]. Available: https://developer.android.com/reference/android/graphics/ImageFormat (visited on 01/12/2021).

[16] ——, *Measure*. [Online]. Available: https://play.google.com/store/apps/details?id=com.google.tango.measure&hl=en_CA&gl=US (visited on 01/13/2021).

[17] J. Grimault, V. Bellassen, and I. Shishlov, "Key elements and challenges in monitoring, certifying and financing forestry carbon projects," Institute for Climate Economics (I4CE), Paris, Tech. Rep. 58, Nov. 2018. [Online]. Available: https://www.i4ce.org/wp-core/wp-content/uploads/2018/11/1106-i4ce2934-PC58-VA.pdf (visited on 01/27/2020).

[18] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning Rich Features from RGB-D Images for Object Detection and Segmentation," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2014, pp. 345–360, ISBN: 978-3-319-10584-0. DOI: 10.1007/978-3-319-10584-0_23.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html (visited on 03/31/2021).

[20] Huawei, *HMS Core: AR Engine*. [Online]. Available: https://developer.huawei.com/consumer/en/hms/huawei-arengine/ (visited on 10/29/2020).

[21] J. Iglhaut, C. Cabo, S. Puliti, L. Piermattei, J. O'Connor, and J. Rosette, "Structure from Motion Photogrammetry in Forestry: A Review," *Current Forestry Reports*, vol. 5, no. 3, pp. 155–168, Sep. 2019, ISSN: 2198-6436. DOI: 10.1007/s40725-019-00094-3. [Online]. Available: https://doi.org/10.1007/s40725-019-00094-3 (visited on 06/22/2020).

[22] Katam Technologies AB, *KATAM™ Forest*, Nov. 2020. [Online]. Available: https://www.katam.se/solutions/forest/ (visited on 03/31/2021).

[23] J. Kour, M. Hanmandlu, and A. Q. Ansari, "Fast Fingerprint Image Alignment," in *Advances in Computer Science, Engineering & Applications*, D. C. Wyld, J. Zizka, and D. Nagamalai, Eds., ser. Advances in Intelligent and Soft Computing, Berlin, Heidelberg: Springer, 2012, pp. 93–99, ISBN: 978-3-642-30157-5. DOI: 10.1007/978-3-642-30157-5_10.

[24] "Land use: Policies for a Net Zero UK," Committee on Climate Change, Tech. Rep., Jan. 2020. [Online]. Available: https://www.theccc.org.uk/publication/land-use-policies-for-a-net-zero-uk/ (visited on 01/28/2020).

[25] M. Lasseck, "Image-based plant species identification with deep convolutional neural networks.," in *CLEF (Working Notes)*, 2017.

[26] Z. Li, A. Strahler, C. Schaaf, D. Jupp, M. Schaefer, and P. Olofsson, "Seasonal change of leaf and woody area profiles in a midlatitude deciduous forest canopy from classified dual-wavelength terrestrial LiDAR point clouds," *Agricultural and Forest Meteorology*, vol. 262, pp. 279–297, Nov. 2018, ISSN: 0168-1923. DOI: 10.1016/j.agrformet.2018.07.014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0168192318302351 (visited on 06/03/2020).

[27] X. Liang, V. Kankare, J. Hyyppä, Y. Wang, A. Kukko, H. Haggrén, X. Yu, H. Kaartinen, A. Jaakkola, F. Guan, M. Holopainen, and M. Vastaranta, "Terrestrial laser scanning in forest inventories," *ISPRS Journal of Photogrammetry and Remote Sensing*, State-of-the-art in photogrammetry, remote sensing and spatial information science, vol. 115, pp. 63–77, May 2016, ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2016.01.006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0924271616000204 (visited on 02/13/2020).

[28] M. Lindner, I. Schiller, A. Kolb, and R. Koch, "Time-of-Flight sensor calibration for accurate range sensing," *Computer Vision and Image Understanding*, vol. 114, pp. 1318–1328, Dec. 2010. DOI: 10.1016/j.cviu.2009.11.002.

[29] R. S. Mbatu, "REDD+ research: Reviewing the literature, limitations and ways forward," *Forest Policy and Economics*, vol. 73, pp. 140–152, Dec. 2016, ISSN: 1389-9341. DOI: 10.1016/j.forpol.2016.09.010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389934116302921 (visited on 01/27/2020).

[30] R. Nair, K. Ruhl, F. Lenzen, S. Meister, H. Schäfer, C. S. Garbe, M. Eisemann, M. Magnor, and D. Kondermann, "A Survey on Time-of-Flight Stereo Fusion," in *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications: Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New*

Modalities, ser. Lecture Notes in Computer Science, M. Grzegorzek, C. Theobalt, R. Koch, and A. Kolb, Eds., Berlin, Heidelberg: Springer, 2013, pp. 105–127, ISBN: 978-3-642-44964-2. DOI: 10.1007/978-3-642-44964-2_6. [Online]. Available: https://doi.org/10.1007/978-3-642-44964-2_6 (visited on 04/09/2021).

[31] "Good Practice Guidance for Land Use, Land-Use Change and Forestry," Intergovernmental Panel on Climate Change, Tech. Rep., 2003. [Online]. Available: https://www.ipcc-nggip.iges.or.jp/public/gpglulucf/gpglulucf_files/Chp4/Chp4_3_Projects.pdf (visited on 01/28/2020).

[32] L. Piermattei, W. Karel, D. Wang, M. Wieser, M. Mokroš, P. Surový, M. Koreň, J. Tomaštík, N. Pfeifer, and M. Hollaus, "Terrestrial Structure from Motion Photogrammetry for Deriving Forest Inventory Data," *Remote Sensing*, vol. 11, no. 8, p. 950, Jan. 2019. DOI: 10.3390/rs11080950. [Online]. Available: https://www.mdpi.com/2072-4292/11/8/950 (visited on 06/03/2020).

[33] Planet Team, *Planet Application Program Interface: In Space for Life on Earth*, San Francisco, CA, 2017. [Online]. Available: https://api.planet.com.

[34] S. Ray and R. H. Turi, "Determination of number of clusters in K-means clustering and application in colour segmentation," in *The 4th International Conference on Advances in Pattern Recognition and Digital Techniques*, 1999, pp. 137–143.

[35] H. Z. U. Rehman and S. Lee, "Automatic Image Alignment Using Principal Component Analysis," *IEEE Access*, vol. 6, pp. 72 063–72 072, 2018, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2018.2882070.

[36] J. Rogelj, D. Shindell, K. Jiang, S. Fifita, P. Forster, V. Ginzburg, C. Handa, S. Kobayashi, E. Kriegler, L. Mundaca, R. Séférian, M. V. Vilariño, K. Calvin, J. Emmerling, S. Fuss, N. Gillett, C. He, E. Hertwich, L. Höglund-Isaksson, D. Huppmann, G. Luderer, D. L. McCollum, M. Meinshausen, R. Millar, A. Popp, P. Purohit, K. Riahi, A. Ribes, H. Saunders, C. Schädel, P. Smith, E. Trutnevyte, Y. Xiu, W. Zhou, K. Zickfeld, G. Flato, J. Fuglestvedt, R. Mrabet, and R. Schaeffer, "Mitigation Pathways Compatible with 1.5°C in the Context of Sustainable Development," in *Global Warming of 1.5°C. An IPCC Special Report on the impacts of global warming of 1.5°C above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty*, 2018.

[37] B. Schlegel, J. Gayoso, and J. Guerra, "Manual de Procedimentos para Inventarios de Carbono en Ecosistemas Forestales," in *Proyecto FONDEF*, Report No. D98I1076, Jan. 2001.

[38] J. L. Schonberger and J.-M. Frahm, "Structure-from-Motion Revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 4104–4113, ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.445. [Online]. Available: http://ieeexplore.ieee.org/document/7780814/ (visited on 03/10/2021).

[39] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor Segmentation and Support Inference from RGBD Images," in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 746–760, ISBN: 978-3-642-33715-4. DOI: 10.1007/978-3-642-33715-4_54.

[40] V. Singh, A. Tewari, S. P. Kushwaha, and V. K. Dadhwal, "Formulating allometric equations for estimating biomass and carbon stock in small diameter trees," *Forest Ecology and Management*, vol. 261, no. 11, pp. 1945–1949, Jun. 2011, ISSN: 03781127. DOI: 10.1016/j.foreco.2011.02.019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S037811271100106X (visited on 03/10/2021).

[41] A. E. L. Stovall and H. H. Shugart, "Improved Biomass Calibration and Validation With Terrestrial LiDAR: Implications for Future LiDAR and SAR Missions," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 10, pp. 3527–3537, Oct. 2018, ISSN: 1939-1404, 2151-1535. DOI: 10.1109/JSTARS.2018.2803110. [Online]. Available: https://ieeexplore.ieee.org/document/8304601/ (visited on 05/29/2020).

[42] A. E. L. Stovall, A. G. Vorster, R. S. Anderson, P. H. Evangelista, and H. H. Shugart, "Non-destructive aboveground biomass estimation of coniferous trees using terrestrial LiDAR," *Remote Sensing of Environment*, vol. 200, pp. 31–42, Oct. 2017, ISSN: 0034-4257. DOI: 10.1016/j.rse.2017.08.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0034425717303668 (visited on 05/20/2020).

[43] Tasmanic Editions, *CamToPlan - AR measurement / tape measure*. [Online]. Available: https://play.google.com/store/apps/details?id=com.tasmanic.camtoplanfree&hl=en_CA&gl=US (visited on 01/13/2021).

[44] C. Taylor, "Most Wanted Invasive Plant Species in Our Natural Areas," *Van Cortlandt Park Alliance*, Nov. 2018, Section: Natural Selections. [Online]. Available: https://vancortlandt.org/2018/11/08/most-wanted-invasive-plant-species-in-our-natural-areas/ (visited on 03/31/2021).

[45] UK Forestry Commission, *UK Woodland Carbon Code*. [Online]. Available: https://www.woodlandcarboncode.org.uk/standard-and-guidance/2-project-governance/2-5-monitoring (visited on 01/27/2020).

[46] *Van Cortlandt Park Preserve*. [Online]. Available: https://www.nycgovparks.org/greening/nature-preserves/site?FWID=48 (visited on 03/31/2021).

[47] A. G. Vorster, P. H. Evangelista, A. E. L. Stovall, and S. Ex, "Variability and uncertainty in forest biomass estimates from the tree to landscape scale: The role of allometric equations," *Carbon Balance and Management*, vol. 15, no. 1, p. 8, May 2020, ISSN: 1750-0680. DOI: 10.1186/s13021-020-00143-6. [Online]. Available: https://doi.org/10.1186/s13021-020-00143-6 (visited on 05/21/2021).

[48] O. Wahltinez, "Getting the Most from the New Multi-Camera API," *Android Developers, Medium*, Nov. 2018. [Online]. Available: https://medium.com/androiddevelopers/getting-the-most-from-the-new-multi-camera-api-5155fb3d77d9 (visited on 10/29/2020).

[49] G. E. Waldron, *Trees of the Carolinian forest*. Boston Mills Press, 2003.

[50] P. Wilkes, A. Lau, M. Disney, K. Calders, A. Burt, J. Gonzalez de Tanago, H. Bartholomeus, B. Brede, and M. Herold, "Data acquisition considerations for Terrestrial Laser Scanning of forest plots," *Remote Sensing of Environment*, vol. 196, pp. 140–153, Jul. 2017, ISSN: 0034-4257. DOI: 10.1016/j.rse.2017.04.030. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S003442571730189X (visited on 06/01/2020).

[51] K. Wu, E. Otoo, and A. Shoshani, "Optimizing connected component labeling algorithms," Lawrence Berkeley National Laboratory, Jan. 2005. [Online]. Available: https://escholarship.org/uc/item/7jg5d1zn (visited on 03/28/2021).

[52] J. Yoshida, *P30 Pro Teardown Proves Huawei's Flash Catch-up*, Apr. 2019. [Online]. Available: https://www.eetimes.com/p30-pro-teardown-proves-huaweis-flash-catch-up/ (visited on 05/07/2021).