Robotic Vision for 3D Modeling and Sizing in Agriculture

Mohamad Qadri

CMU-RI-TR-21-43

July 29, 2021

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
George Kantor, Chair
Michael Kaess
Oliver Kroemer
Ming-Fang (Allie) Chang

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Copyright © 2021 Mohamad Qadri. All rights reserved.
Abstract

Obtaining accurate perceptual information is a critical component in agricultural robotics since there is a heavy need for interaction with the environment to perform tasks such as pruning, harvesting, and phenotyping. In this thesis, we tackle the problem of perception and 3D modeling in agriculture through the use of stereo cameras in the context of three applications: apple fruitlets sizing, online 3D reconstruction through in-hand camera manipulation, and sorghum field 3D modeling.

First, a deep-learning-based vision system aimed at measuring apple fruitlets in the field is presented achieving an accuracy of less than 1mm compared to ground truth measurements.

The problem of sorghum field 3D modeling is then tackled using Simultaneous Localization and Mapping (SLAM) techniques. An object-level feature association algorithm is proposed that enables the creation of 3D reconstructions robustly by taking advantage of the structure of robotic navigation in agricultural fields. An object-level SLAM system is presented that utilizes recent advances in deep learning-based object detection and segmentation algorithms to detect and segment semantic objects in the environment which are used as landmarks for SLAM. The SLAM system does not use inertial sensory measurements and only relies on visual odometry from a stereo camera capturing images at the frame rate of 5Hz. The object-based feature association algorithm enabled mapping 78% of a sorghum range on average in contrast with traditional visual features which have an average mapped distance of 38%. The system is also compared against ORB-SLAM2, a state-of-the-art visual SLAM algorithm, and shows significant performance improvement in the average mapped distance metric.

Finally, we tackle the problem of 3D reconstruction and mapping through the use of an in-hand camera attached to a mobile robot. A planning strategy to perform robotic arm scanning, a 3D reconstruction system, and preliminary apple fruitlet mapping strategies are proposed. The system was deployed in the field and used to autonomously scan tree canopies, collect datasets, and build 3D reconstructions in apple orchards.
Acknowledgments

I would like to thank my advisor George Kantor for his support and advice over the course of my studies and for giving me the opportunity to explore and work on a variety of interesting and highly rewarding problems. I want to also thank Abhisesh Silwal, Francisco Yandun, and all my lab mates for making this journey enjoyable and helping me grow as a robotic engineer and researcher. Also, a special thank you to my parents and my two siblings for their tremendous support and encouragement over the past couple of years.

I want to also especially thank our research sponsors and partners. First, the NSF / USDA NIFA Cyber-Physical Systems Program under Contract NIFA 2020-01469-1022394 and our partners at the University of Massachusetts Amherst. And second, the DOE ARPAE TERRA Program under contract DE-AR0001134 and our partners at Clemson University.
## Contents

1 Introduction 1

2 Related Work 5
   2.1 2D Object Detection and Segmentation 5
   2.2 3D Modeling in Agriculture 5
   2.3 Robotic Vision with an In-Hand Camera in Agricultural settings 6
   2.4 Deep Networks for Depth Generation 6
   2.5 Simultaneous Localization and Mapping 7

3 Apple Fruitlet Sizing using Deep Learning Algorithms 9
   3.1 Computer Vision Pipeline 10
      3.1.1 Faster-RCNN for Object Detection 10
      3.1.2 Image Segmentation 12
      3.1.3 Disparity Map Generation using MadNet 14
      3.1.4 Ellipse Fitting and Apple Fruitlet Sizing 16
   3.2 Data Collection 18
   3.3 Results 20
   3.4 Discussion 23

4 3D Sorghum Mapping 25
   4.1 Proposed Method 25
      4.1.1 System Overview 25
      4.1.2 Feature Extraction 26
      4.1.3 Data Association Algorithm 27
      4.1.4 3D Pose Estimation 30
      4.1.5 Factor Graph Optimization 31
      4.1.6 Pointcloud Postprocessing 34
      4.1.7 Summary of System 34
   4.2 Results 35
      4.2.1 Maximum Distance Mapped 37
   4.3 Discussion 39
5 Data Collection Using Autonomous Manipulation
  5.1 Mechanical Design .............................................. 41
  5.2 System Architecture ........................................... 42
  5.3 Arm Trajectory Planning ....................................... 43
    5.3.1 Pointcloud Centroid Calculation .......................... 43
    5.3.2 Adaptive Scanning Plane Selection ....................... 44
    5.3.3 Waypoint Generation ...................................... 44
    5.3.4 Arm Scanning Trajectory Planning ....................... 47
  5.4 Fully Autonomous Data Collection ............................ 48
    5.4.1 Navigation ................................................. 48
    5.4.2 Finite State Machine ...................................... 48
  5.5 Results ................................................................ 49
    5.5.1 Autonomous Scanning ....................................... 49
    5.5.2 Fully Autonomous Data Collection ....................... 52
    5.5.3 Discussion .................................................... 54

6 3D Reconstruction and Mapping Using In-Hand Camera Images 55
  6.1 3D Reconstruction .................................................. 55
    6.1.1 Object Detection ............................................ 55
    6.1.2 Disparity Map Generation using SGBM .................... 56
    6.1.3 ROS Message Synchronization .............................. 56
    6.1.4 Pointcloud Transformation as an Initialization for ICP 57
    6.1.5 3D Pointcloud Merging .................................... 58
  6.2 Apple Fruitlet Mapping in Tree Canopies ....................... 61
    6.2.1 Algorithm 1: Fruitlet Mapping by 3D Coordinates Averaging 62
  6.3 Discussion ........................................................ 65
    6.3.1 Algorithm 2: Fruitlet Mapping using Factor Graphs .......... 65

7 Conclusion and Future Work ........................................ 67
  7.1 Conclusion ....................................................... 67
  7.2 Future work ...................................................... 67

Bibliography ......................................................... 69
List of Figures

1.1 Aerial views of a sorghum breeding panel at Clemson University’s Pee Dee Research and Educational Center in Florence, SC. The field is organized into plots, where each plot is approximately $2m \times 5m$ and contains a different variety of sorghum ........................................ 2

1.2 Robotic system used to automate the data collection process in apple orchards 3

3.1 An apple fruitlet cluster tagged with an april tag. An ID is assigned to each fruitlet in the cluster .................................................. 9

3.2 Computer Vision Pipeline used to Generate the Apple Fruitlet Size Measurements ............................................................. 10

3.3 An example of fruitlet detection using faster-rcnn ............................................. 11

3.4 Generative adversarial networks. Figure borrowed from: [31] .................. 12

3.5 Example of a training sample used to train the pix2pix network .............. 13

3.6 Segmentation result for all fruitlets in a cluster ........................................ 13

3.7 Self-supervised Disparity Generation using Left-Right Image Consistency .. 14

3.8 Example 1 of left and right stereo images of a fruitlet cluster............... 15

3.9 Example 2 of disparity maps generated using SGBM (right) and MADNet (left) 16

3.10 Camera used to image the fruitlet .......................................................... 19

3.11 Caliper used to obtain ground truth size values ........................................ 19

3.12 Example images of a fruitlet cluster captured in the field (apple orchard) .. 19

3.13 Distribution of ground truth fruitlet sizes ............................................. 20

3.14 Distribution of fruitlet sizes measured by the computer vision pipeline .. 21

3.15 Average computer vision measurement error (in mm) compared against ground truth caliper measurements. ................................. 21

3.16 Comparing the distributions of apple fruitlet sizes from ground truth measurements against the computer vision pipeline for the collection day 05/27/2020 22

3.17 Comparing the distributions of apple fruitlet size from ground truth measurements against the computer vision pipeline for the collection day 06/01/2020 23

4.1 System overview of SLAM system which uses sorghum seeds as semantic landmarks .......................................................... 26

4.2 Example detected sorghum seeds which serve as landmarks for our SLAM system .......................................................... 26
4.3 Our detection and segmentation pipeline. The center of each ellipse is considered the projection of a 3D landmark.

4.4 An example of the same sorghum seed imaged in 2 consecutive images where its position in the right image (time = t+1) relative to the left image (time = t) is predominantly defined by a horizontal translation.

4.5 Proposed feature association pipeline.

4.6 (a) The first row shows the bounding box detections. Each bounding box bounds one sorghum seed. The second row shows the output of the proposed data association pipeline for five consecutive images. (b) We note that the displacement of a projected 3D landmark between 2 consecutive frames is predominantly defined by a horizontal translation.

4.7 Example of a factor graph

4.8 Huber loss with $\sigma = 1$ in orange. The squared error loss in green.

4.9 Factor graph representation

4.10 An example of a reconstructed 3D scene. The first 1 meter is reconstructed from the sequence of images in Fig. 4.6.

4.11 Layout of a sorghum row in a sorghum field

4.12 Examples of matches obtained using our data association algorithm versus matches obtained using the OpenCV implementations of SIFT, SURF, AKAZE, and ORB (before applying the motion constraints). Traditional feature detectors return a high number of incorrect matches.

5.1 The Warthog robot, UR5 arm, and the slider operating in the field

5.2 The 7th DOF (left) and the UR5 robot arm (right)

5.3 Planning 3D reconstruction pipeline

5.4 Illustration of the desired raster scan behavior

5.5 (RVIZ simulation) Calculated waypoints at a specific 3D location and orientation facing the tree canopy with $\text{recur\_depth\_v} = 1$ and $\text{recur\_depth\_v} = 0$

5.6 The RRT-Connect Algorithm. Figure borrowed from [24]

5.7 Finite State Machine for Fully Autonomous Data Collections. The state machine transitions between navigation, calibration and autonomous scanning.

5.8 Aerial view of the apple orchard in Amherst, Massachusetts

5.9 x position of the camera in world frame

5.10 y position of the camera in world frame

5.11 z position of the camera in world frame

5.12 The blue triangles indicate the location of the apple trees where the mobile robot needs to stop and run the scanning pipeline. The blue and black lines indicate the desired robot path.

5.13 Autonomous navigation result. The triangle in light green indicate the trees that the robot scanned successfully from both sides. The line in light green indicate the path that the robot was able to follow successfully. Red triangles and line indicate the section which was manually aborted.

5.14 Apple fruitlet trees in the field
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Example of bounding box detections on a testing image</td>
<td>56</td>
</tr>
<tr>
<td>6.2</td>
<td>Example of a disparity map generated using SGBM OpenCV</td>
<td>56</td>
</tr>
<tr>
<td>6.3</td>
<td>Method used to perform ROS message synchronization</td>
<td>57</td>
</tr>
<tr>
<td>6.4</td>
<td>Point-to-plane ICP Method. Figure borrowed from [29]</td>
<td>58</td>
</tr>
<tr>
<td>6.5</td>
<td>Pointcloud Merging Process</td>
<td>59</td>
</tr>
<tr>
<td>6.6</td>
<td>An Example of a generated 3D reconstruction of a tree canopy using 197 images (offline)</td>
<td>60</td>
</tr>
<tr>
<td>6.7</td>
<td>Algorithms for 3D Apple Fruitlet Mapping</td>
<td>61</td>
</tr>
<tr>
<td>6.8</td>
<td>Example of semantic matches over 4 consecutive frames with the camera moving “down” between 2 vertical waypoints (see Fig 5.5). Algorithm 1 was used to estimate the 3D location of fruitlets.</td>
<td>62</td>
</tr>
<tr>
<td>6.9</td>
<td>Example of how the value $N_i$ is found for an example cluster. The figure consists of 3 consecutive images. The same cluster (circled in red) is visible in all three. In the first image 4/5 fruitlets were detected. In the second image, 4/5 fruitlets were detected, and finally 5/5 fruitlets were detected in the last image. Therefore $N_i = (\min(4,4,5) + \max(4,4,5))/2 = 4.5$</td>
<td>64</td>
</tr>
<tr>
<td>6.10</td>
<td>Three consecutive images of the same section of a tree canopy. 3D landmarks are projected onto the images. Each projection is represented by a colored dot. It can be seen that the 2 fruitlets boxed in red could still be tracked although they are occluded by a leaf in the last frame.</td>
<td>65</td>
</tr>
<tr>
<td>6.11</td>
<td>Factor graph for apple fruitlet mapping</td>
<td>66</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Precision and recall of the Faster-RCNN network on the training set</td>
<td>11</td>
</tr>
<tr>
<td>3.2</td>
<td>Precision and recall of the Faster-RCNN network on the testing set</td>
<td>11</td>
</tr>
<tr>
<td>3.3</td>
<td>Number of fruitlets with ground truth measurements for each collection day</td>
<td>19</td>
</tr>
<tr>
<td>3.4</td>
<td>Mean and Median (in mm) of the ground truth fruitlet size measurements</td>
<td>20</td>
</tr>
<tr>
<td>3.5</td>
<td>Mean and Median (in mm) of the computer vision measurements</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Maximum distance mapped (in meters). All methods compared under the same camera motion assumptions. Ground truth distances are extracted from GPS readings</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>Maximum distance mapped - ORB-SLAM2 vs OURS</td>
<td>38</td>
</tr>
<tr>
<td>6.1</td>
<td>Precision and recall of the Faster-RCNN network on the training set</td>
<td>55</td>
</tr>
<tr>
<td>6.2</td>
<td>Precision and recall of the Faster-RCNN network on the testing set</td>
<td>55</td>
</tr>
<tr>
<td>6.3</td>
<td>Accuracy values using algorithm 1 vs directly counting the fruitlets using the number of bounding boxes</td>
<td>64</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The global food supply is currently under pressure due to several factors, including population growth, climate change, migration from rural to urban areas, and the increasing population age in advanced economies [9]. These factors contribute to the current shortage of trained workers who can perform tasks such as pruning, harvesting, and crop inspection and phenotyping in agricultural fields. These tasks are often repetitive, physically demanding, and conducted in difficult environments which motivate the research and development of automation solutions. These tasks do however necessitate accurate perceptual information and some level of scene understanding prior to performing automated interactions with the environment. In this thesis, we tackle the problem of perception and modeling in agriculture through the use of stereo cameras in the context of three applications: apple fruitlets sizing, sorghum field modeling, and the online 3D reconstruction of tree canopies using a robotic in-hand camera. Stereo-based visual systems are often the preferred choice when imaging agricultural fields because of their wide availability and reasonable cost. Stereo cameras can also resolve finer details compared to LIDAR, which only provides a sparse representation of the environment. In all three applications, we used active lighting-based stereo camera systems, as proposed in [49], which produced images invariant to lighting conditions and that were shown to enable training of detection and segmentation networks on less training data.

Fruit sizing and growth rate measurements are important information used by growers to make more informed management decisions. Take the application of predicting chemical thinner response on apple fruitlet growth as an example. This is typically performed via prediction models with necessary data, such as fruitlet diameter length, being manually collected over multiple collection days. [18] presented a strategy for data collection and a model to predict the abscission or persistence of apple fruitlets: the growers need to tag 105 spurs on seven individual trees with each fruitlet in the spur being identified and numbered. The size of these fruitlets needs to be manually measured and recorded over multiple measurement days to serve as input to the prediction model. Robotics and computer vision could be of use to automate all or part of this data collection process motivating our first application of interest (Chapter 3): automating the process of sizing apple fruitlets using computer vision techniques. This vision system is, however, not restricted to this
1. Introduction

Figure 1.1: Aerial views of a sorghum breeding panel at Clemson University’s Pee Dee Research and Educational Center in Florence, SC. The field is organized into plots, where each plot is approximately $2m \times 5m$ and contains a different variety of sorghum.

particular application, and major components will be re-used in later chapters for different applications.

Our second application of interest is mapping in sorghum fields using Simultaneous Localization and Mapping (SLAM) techniques (Chapter 4). SLAM enables autonomous robot navigation in unknown environments by providing a joint estimate of the robot poses and the 3D location of landmarks. SLAM has important applications in agriculture since the 3D reconstructions obtained from mapping agricultural fields can provide valuable information for downstream tasks such as plant phenotyping, crop counting, and yield prediction. One such application is high-throughput sorghum seed phenotyping. Sorghum is an essential grain for food and energy production, and significant efforts are being put into utilizing phenotyping to improve crop quality and understand sorghum genetic variations [2]. However, existing visual-based SLAM and 3D reconstruction algorithms such as ORB-SLAM2 and Structure from Motion (SFM) pipelines that fundamentally rely on the accuracy of traditional visual feature matching algorithms such as SIFT and ORB often fail. These failures are due to the lack of texture in the images, variations in luminosity levels, the dynamics of the environment (for example, leaves or crops moving due to the wind), and the presence of repeated patterns.

On the other hand, Robotic navigation in agricultural fields exhibit a specific structure: farms are composed of rows arranged in a straight line (Fig. 1.1b), and robots traverse the field one row at a time, ideally moving in a straight line while taking images of plants from the sides [33]. In this thesis, we propose a method that takes advantage of this structure and presents a SLAM system and an object-level data association algorithm that use the
imaged sorghum seeds as semantic landmarks for SLAM. The layout of farms for other types of fruits and seeds is similar to that of a sorghum field. Hence, our SLAM system can be easily adapted to construct robust 3D models of other plantations and obtain accurate phenotyping data.

In Chapter 5, we visit the problem of automating the image collection of apple fruitlets in orchards. We propose an algorithm for scanning apple tree canopies using a robotic in-hand camera attached to a mobile robot shown in Fig. 1.2. The system was successfully used to autonomously collect images of 48 apple trees for each of 3 collection days. Integration of the arm scanning routine with the autonomous navigation system described in [48] and in-field testing results of the fully autonomous system are also described.

In Chapter 6, we propose a system to perform online 3D reconstruction of tree canopies using the robotic in-hand camera images, and two initial stage algorithms to perform apple fruitlet mapping in tree canopies. Previous work of robotic system deployed in the field such as [48], include a camera system which is typically fixed and attached to the base of the robot. 3D models constructed from the fixed camera images are then used to perform tasks such as pruning and harvesting. However, tree canopies have generally some depth, and obstacles, such as branches, might not be seen from the viewpoint of the fixed camera. Using an in-hand camera to perform full 3D reconstruction of tree canopies or using the in-hand camera to perform pointcloud completion as proposed in [48] as future work could alleviate such problems.
1. Introduction
Chapter 2

Related Work

2.1 2D Object Detection and Segmentation

Several impressive 2D object detection (YOLOv3 [42], SSD [27], Faster-RCNN [43]) and semantic segmentation networks (PSPNet [56], Mask-RCNN [20]) based on deep neural networks, have recently been proposed. [23] presented DaSNet-V2, a deep network based one stage detector which performs the detection and instance segmentation of apple fruits as well as the semantic segmentation of branches in apple orchards. [14] proposed a robust and accurate multi-class apple detection method based on Faster-RCNN which detects apples in different conditions such as non-occluded, leaf-occluded, and branch-occluded with the goal of using this information to improve the picking and planning strategy for autonomous robot harvesting robots. StalkNet [1] proposed an image processing pipeline used to measure stalk width. It used a Faster-RCNN network for image detection and a Fully Convolutional Network (FCN) to segment stalks. Ellipses are then fitted to the segmented stalks and used to measure their width. [39] improved the segmentation accuracy of StalkNet by using an image-to-image translation conditional adversarial network “pix2pix” [22] as the segmentation network.

2.2 3D Modeling in Agriculture

Building accurate 3D models are critical in many robotic automated tasks in agriculture. These reconstructions can, for example, serve as sensory input to downstream tasks such as pruning and harvesting. 3D reconstructions can also be used to automate the extraction of phenotyping data which can be expensive and time-consuming to collect manually. [36] proposed a SLAM system for outdoor agricultural environments which uses geometric primitive shapes fitted into 3D points as semantic 3D landmarks. The system was tested on two applications: grape counting in vineyards and dormant season grape plant pruning. [8] tackled the problem of building a 3D model by merging two sides of a row in an apple orchard. They use global features and semantic information to obtain an initial alignment of the two sides of a row which is later further refined. The author proposed methods to use this model to measure traits such as canopy volume, trunk diameter, and tree height. [15]
2. Related Work

presented a methodology for apple fruits detection and 3D mapping. A Mask R-CNN is used to detect and segment fruits in images and structure from motion (SFM) was used to generate a 3D model. An SVM was trained to identify and remove detections that are false positives. [57] tackled the problem of delineating apple trees in a trellis structured orchard and perform apple fruit counts for each tree. Two 3D models for the same set of trees are generated. The first in the winter season (when the trees are leafless) called “winter pointcloud” and is used to detect the trunks and identify the branches belonging to each tree. Another 3D model is constructed during harvest season called “harvest pointcloud” and is used to localize the apples in 3D. The assignment of each apple fruit to the tree it belongs to is done by mapping the location of the fruits in the harvest pointcloud onto the winter pointcloud.

2.3 Robotic Vision with an In-Hand Camera in Agricultural settings

We present a few relevant works in robotic vision using an in-hand camera in agricultural settings.
In [50], a robotic platform with two arms, each fitted with two multi-camera sensorpods, is used to collect a dataset of sorghum plants with multiple horizontal and vertical views for each plant. In-field 3D reconstructions of sorghum plants are generated and segmentation techniques are used to extract plant subunits called “phytomers”. Lehnert et al. [26] proposed the 3D Move to See (3DMTS) approach to navigate an end effector around obstacles to obtain a better view of the fruit in preparation for robotic harvesting. The approach guides the arm toward the path that maximizes the area of fruits visible (formulated as an objective function) in an image by comparing images from nine cameras arranged in a 3D array on the end effector. In [54], the authors proposed an enhancement to 3DMTS which removed the need for the 3D camera array by using a single Convolutional Neural Network (CNN) to estimate the gradient of the objective function which is then used to update the arm position. The authors in [32] presented a Visual Servo Control law for the task of citrus harvesting where two cameras (a fixed camera and an eye-on-hand camera) are used as the perception system. The fixed camera is used to generate a global map of fruit locations. A controller is then used to move the arm such that one of the fruits enters the field of vision of the eye-on-hand camera. Finally, a translation controller is used to move the end-effector and complete the harvesting task.

2.4 Deep Networks for Depth Generation

Depth estimation is a core topic in computer vision and robotics and classical algorithms such, as SGM [21], have been developed to address the problem. More recently, deep learning-based approaches for depth estimation have been proposed. Early work, such as [25] and [13], considered the problem in a supervised learning setting where neural networks are trained to estimate dense depth from RGB images. However, obtaining ground truth labels for depth estimation can be expensive and time-consuming. Hence, researchers have
more recently proposed self-supervised approaches that minimize the need of capturing ground truth depth. For example, [16] proposed an unsupervised deep network to regress depth from single images and a novel loss function that enforces “consistency” between the left and right stereo images in the training set leading to improved results and comparable performance to fully-supervised baselines. However, the limitation of these systems is that their training and testing data generally come from the same dataset and hence lack generalization to new environments. Several approaches have been proposed to remedy this problem by performing online adaptation. For example, [55] proposed a method to perform an online adaptation of deep networks weights for the task of depth prediction on a target dataset different from the source/training dataset. It introduced a sequential learning method Online Meta-Learning with Adaptation (OMLA) designed for a sequential learning setting with both fast convergence during training and fast adaptation to different data sources. A Meta-pre-training method, which uses OMLA, is proposed. Its objective is to obtain good disparities on the source/training dataset as well as converging to network weights that can serve as good initialization when adapting the network to predict disparities on a target dataset \( T \). To reduce the domain shift problem between the training and target distributions, a feature alignment method is utilized, derived from batch normalization, to align the source and target feature distribution to a common reference distribution.

2.5 Simultaneous Localization and Mapping

We present related works in SLAM in the context of our target application of mapping in sorghum fields. Current state-of-the-art indirect feature-based Visual-SLAM algorithms, such as ORB-SLAM2 [34], depend on the accuracy of the extracted features, such as ORB, and can fail in environments where feature matching is not robust. Direct SLAM methods such as LSD-SLAM [11] operate directly on pixel-level brightness information and achieve tracking and pose estimation by minimizing the photometric error between the new frame and the current keyframe. However, direct methods are more susceptible to changes in lighting conditions, common in images taken in an agricultural field. Several works exist on object-level SLAM: SLAM++ [46] is an RGB-D based system that detects known objects in the environment, which are then used as high-level landmarks. However, SLAM++ requires the creation of a database of high-quality 3D models used as prior. The authors in [4] proposed a method for the online discovery of objects in indoor environments without the use of prior models. Each object is represented by the centroid of its associated pointcloud. However, only non-planar regions are considered as potential landmarks which limits the method’s applicability in our settings since sorghum panicles appear to be mostly planar from the viewpoint of our camera. In addition, our objective is to use individual seeds as landmarks and not the entire sorghum panicle. With the progress made in 2D object detection and segmentation, numerous works combine deep learning and SLAM: Fusion++ [30] utilizes Mask-RCNN [20] to initialize Truncated Signed Distance Function (TSDF) reconstructions, which are, in turn, used as object-level representations of landmarks. Fusion++ mostly
2. Related Work

targets indoor environments. QuadricSLAM [37] and CubeSLAM [53] are monocular-based
SLAM systems. They propose two different 3D landmarks representations, “quadrics”
and “cuboids,” respectively, which are estimated from 2D bounding boxes without prior
models. They are based on a factor-graph based SLAM formulation that jointly estimates
the 3D pose of these objects along with the camera poses. The authors in [36] explain
that the performance of QuadricSLAM and similar approaches will be poor in cluttered
environments such as a sorghum field where there is a significant overlap between bounding
boxes. Similarly, ROSHAN [38] introduced an ellipsoid landmark-based monocular SLAM
system which constrains the ellipsoid representation using information from detected
bounding boxes, a prior on the shape of the object, and a “texture” plane estimated from
triangulated feature points on the surface of the object. In our application, the overlap
of the detected seeds’ bounding boxes and the lack of texture in the images limit the
applicability of such methods. Kimera [45] uses Visual-Inertial Odometry to estimate the
state of the robot and builds a semantic mesh model of the environment. We note that our
system focuses on Stereo-Visual Odometry.

Most state-of-the-art SLAM algorithms are framed as factor graph optimization problems
where nodes represent robot poses or 3D landmarks in the environment, and edges (factors)
are probabilistic constraints on these variables derived from sensor measurements. We
use GTSAM [6] as the underlying optimization framework, a library optimized for solving
sparse factor graph representations.
Chapter 3

Apple Fruitlet Sizing using Deep Learning Algorithms

Precision thinning models for apple fruitlets typically require manually tagging several apple fruitlet clusters and performing manual size measurement errors across several days, a time-consuming and labor-intensive operation. In this section, we tackle the problem of apple fruitlet sizing using computer vision techniques. An example of an apple fruitlet cluster consisting of five fruitlets is shown in Fig. 3.1. Our objective is to accurately measure the size of each fruitlet in a cluster using computer vision techniques. The detection and segmentation pipelines described in this section will be a building block for later applications and are adapted from the prior work in [1] and [39].

Figure 3.1: An apple fruitlet cluster tagged with an april tag. An ID is assigned to each fruitlet in the cluster
3.1 Computer Vision Pipeline

3.1.1 Faster-RCNN for Object Detection

Faster-RCNN [43] is a Convolutional Neural Network (CNN)-based architecture to perform object detection composed of a CNN to extract features from the input images, a region proposal network (RPN) used to generate object proposals, and a classification network to predict object classes all trained end-to-end. A Faster-rcnn network was trained with 450 training field images of fruitlet clusters (∼4500 bounding boxes) to detect apple fruitlets using an Nvidia 1080Ti GPU.
3. Apple Fruitlet Sizing using Deep Learning Algorithms

Figure 3.3: An example of fruitlet detection using faster-rcnn

We report the precision and recall values of the network on the training and test sets in tables 3.1 and 3.2.

Table 3.1: Precision and recall of the Faster-RCNN network on the training set

<table>
<thead>
<tr>
<th>IOU Threshold</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.917</td>
<td>0.858</td>
</tr>
<tr>
<td>0.6</td>
<td>0.915</td>
<td>0.856</td>
</tr>
<tr>
<td>0.9</td>
<td>0.83</td>
<td>0.777</td>
</tr>
</tbody>
</table>

Table 3.2: Precision and recall of the Faster-RCNN network on the testing set

<table>
<thead>
<tr>
<th>IOU Threshold</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.896</td>
<td>0.812</td>
</tr>
<tr>
<td>0.6</td>
<td>0.890</td>
<td>0.807</td>
</tr>
<tr>
<td>0.9</td>
<td>0.801</td>
<td>0.743</td>
</tr>
</tbody>
</table>
3.1.2 Image Segmentation

**Generative Adverserial Networks (GANs)**

A Generative Adverserial network [17] consists of two networks: a generator \( G \) and a discriminator \( D \) where \( G \) and \( D \) are both differentiable functions. The generator aims at producing data that matches closely the training distribution while the discriminator aims at distinguishing real images (training images) from the images produced by the generator. Concretely, \( D \) and \( G \) play the following minimax game with the objective defined as follows:

\[
\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} \left[ \log(D(x)) \right] + \mathbb{E}_{z \sim p(z)} \left[ \log(1 - D(G(z))) \right]
\] (3.1)

For the case of image generation, \( D(x, \theta_d) \) is network parametrized by \( \theta_d \) that takes in as input an image \( x \) and outputs a scalar indicating the probability of whether \( x \) is a real image (sampled the training data) or fake (generated by the generator). Therefore, \( D \) maximizes the log-likelihood for the binary classification problem:

- Training images: real (1)
- Generated images: fake (0)

The Generator \( G(z, \theta_g) \) is a network parametrized by \( \theta_g \) that takes an a noise vector \( z \) sampled from a noise prior \( p(z) \) and tries to learn a mapping from \( p(z) \) to the real world distribution \( p_{\text{data}} \). \( G \) tries to minimize the quantity \( \log(1 - D(G(z))) \) or in other world, tries to minimize the probability that the discriminator \( D \) classifies it’s output \( G(z) \) as fake.

![Figure 3.4](image-url)

*Figure 3.4: Generative adversarial networks. Figure borrowed from: [31]*

**Conditional GANs / pix2pix**

The generator, in the vanilla formulation of GANs, try to learn the underlying training images distribution with no additional constraints. Hence, generators are free to learn any mapping from \( p(z) \) (the noise distribution) to \( p_{\text{data}} \). Conditional GANs (CGANs) conditions the output of the generator on an input image. In CGANs, the training data is in the form \((x, y)\) where \( x \) is the original image and \( y \) is the labeled image that \( x \) is conditioned on. The
training objective is

\[
\text{Loss} = \min_G \max_D \mathcal{L}_c(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{3.2}
\]

where:

\[
\mathcal{L}_c(G, D) = \mathbb{E}_{x,y \sim p_{\text{data}}} [\log D(x,y)] + \mathbb{E}_{x \sim p_{\text{data}}, z \sim p(z)} [\log(1 - D(x,G(x,z)))] \tag{3.3}
\]

\[
\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y \sim p_{\text{data}}, z \sim p(z)} ||y - G(x,y)||_1 \tag{3.4}
\]

The discriminator task remains the same: to distinguish real image pairs \((x,y)\) from fake pairs consisting of the input image and the output of the generator \((x,G(x,z))\). In addition to fooling the discriminator, the generator output now also needs to produce images that are near the label \(y\) in an L1 sense which is encouraged by introducing the loss term \(L_{L1}\). Lastly, we note that instead of sampling the noise \(z\) from a specific distribution \(p(z)\) such as a Gaussian distribution, noise is introduced in the form of a dropout applied to layers of the generator during training and testing.

### Conditional GANs to segment fruitlets

We train the pix2pix network using \(\approx 300\) training images each consisting of the original image and the ground truth segmentation mask (Fig. 3.5). When generating the size measurements (during testing), each detected apple fruitlet is cropped out (using the bounding box image coordinates), resized to a fixed size of \(256 \times 256\) px, and passed to the pix2pix generator. The result of this operation is a segmented image crop of each fruitlet. The image crop is then resized to the original cropped area to be later used to calculate the fruitlet width. Fig. 3.6 shows examples of segmentation masks overlayed on top of the image of an apple fruitlet cluster.

![Figure 3.5: Example of a training sample used to train the pix2pix network](image1)

![Figure 3.6: Segmentation result for all fruitlets in a cluster](image2)
3. Apple Fruitlet Sizing using Deep Learning Algorithms

3.1.3 Disparity Map Generation using MadNet

In order to obtain depth estimates from stereo, we first attempted to use traditional methods for disparity generation such as SGBM. However, due to errors when performing stereo calibration before data collection in the field, we opted for a deep learning-based method for disparity generation which we found to be less sensitive to calibration error (although stereo rectification is also assumed for the following deep learning method).

Self-supervised Disparity Generation using Left-Right Image Consistency

Formulating depth estimation as a supervised learning problem using deep networks is impractical since obtaining ground truth depth estimates for training in a variety of scenes is difficult. Instead, unsupervised disparity generation using left-right image consistency [16] poses depth estimation as an image reconstruction problem: given a rectified stereo image pair, the aim is to learn the correspondence field $d^r$ that when applied to the left image $I_l$, enable us to obtain a reconstruction of the right image $\tilde{I}_r$ and similarly $d^l$ that allows for the reconstruction of the left image $\tilde{I}_l$ from the right image $I_r$. Note from Fig. 3.7 that the left image is used to generate both disparities and that the right image is only needed during training. The training loss consists of three main terms (further detail can be found in [16]):

- **Appearance matching loss** which encourages $I_l$ and $\tilde{I}_l$, and, $I_r$ and $\tilde{I}_r$ to be similar.
- **Disparity Smoothness Loss** which enforces disparities to be smooth.
- **Left-Right Disparity Consistency Loss** that encourages the left and right disparities to be consistent which leads to more accurate results as opposed to predicting disparity from one view only.

![Diagram](image)

Figure 3.7: *Self-supervised Disparity Generation using Left-Right Image Consistency*
Deep Stereo Generation using MADNet

[52] proposes a lightweight architecture MADNet and a modular adaptation algorithm (MAD) that trains sub-portions of the network independently which are designed for real-time adaptive deep stereo network via self-supervision. The aim is to finetune a pre-trained network online, leveraging a self-supervised loss function similar to the left-right consistency method, to generate accurate disparity estimates from images sampled from a target distribution that is different from the source/training distribution.

Finetuning for disparity generation on apple fruitlet dataset

We leverage the network finetuning process with MADNet to generate disparity estimates from the apple fruitlet stereo images. During data collection, a separate video for each fruitlet cluster was recorded on each day. We treat each video as a separate dataset and use the frames to first finetune MADnet (pre-trained on the Kitti dataset) and second, generate disparity maps for each frame in the video sequence (as shown in Fig. 3.2). Since we are performing the fruitlet sizing operation offline with no online computational constraints, we perform full back-propagation on all network layers (instead of using MAD) which generated better results. The network is finetuned for 20 epochs with a learning rate of 0.0001. Note that all images are resized to 676px × 540px and that the max_disparity parameter is equal to 256px.

Results

Figure 3.9 shows the disparity maps generated using SGBM and MADNet from the stereo images in Fig. 3.8. For the specific task of fruitlet sizing, good depth information particularly at the location of the fruitlets is required. We found that MADNet performed better for this task and this particular dataset since it generated smoother disparity maps and returned acceptable disparity values near the center of the fruitlets. SGBM generally generated noisier maps due to its sensitivity to calibration errors in addition to requiring frequent re-tuning of the its parameters for different clusters. A more careful stereo calibration process was used in following in-field data collections (chapter 6) resulting in better results using SGBM.

Figure 3.8: Example 1 of left and right stereo images of a fruitlet cluster
3. Apple Fruitlet Sizing using Deep Learning Algorithms

3.1.4 Ellipse Fitting and Apple Fruitlet Sizing

Pinhole Camera Model

We assume throughout this thesis a simple pinhole camera model to describe the projection of 3D points onto the image plane. The general pinhole camera matrix can be written as:

\[ P = KT_{\text{cam}} \]

where \( K = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \) is an intrinsic matrix with \( f_x \) and \( f_y \) representing the focal length of the camera in the x and y axis respectively, \( s \) is the skew parameter, and \((c_x, c_y)\) are the coordinate of the principal point.

\( T_{\text{cam}} \in SE(3) \) is a rigid transformation expressing the pose of the camera in a world coordinate frame. Hence, given a 3D point expressed in world frame in homogeneous coordinates \( W = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T \), its projection to image frame \( p \) is given by

\[ p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = KT_{\text{cam}}W \]

Apple Fruitlet Sizing

Once a segmentation mask is generated for all fruitlets in the cluster and a disparity map is computed for the left image, we perform the following operations to obtain the size of each fruitlet in metric units (cm).

- Use OpenCV binary thresholding on each segmented fruitlet to obtain a binary mask. i.e the segmented area will hold a value of 1 while other areas will hold the value 0. Experimentally, the threshold value is set to 200.
- Use OpenCV.findContours to obtain the contours around the segmented areas and select the largest contour \( C \).
• Use OpenCV.fitEllipse on the contour \( C \) to obtain the best ellipse fitted to the segmented apple fruitlet. The function returns:
  
  - \( x \) : the x coordinate of the center
  - \( y \) : the y coordinate of the center
  - \( Ma \) : the major semi-axis
  - \( ma \) : the minor semi-axis
  - \( \theta \) : the rotation angle

Note that OpenCV.fitEllipse uses a constrained least-squares formulation [12] to find the parameters of the best fitting ellipse. The general conic can be represented using the following second-order polynomial:

\[
F(\sigma, u) = \sigma \cdot u = \sigma_1 x^2 + \sigma_2 xy + \sigma_3 y^2 + \sigma_4 x + \sigma_5 y + \sigma_6
\]

where: \( \sigma = [\sigma_1 \ \sigma_2 \ \sigma_3 \ \sigma_4 \ \sigma_5 \ \sigma_6]^T \) and \( u = [x^2 \ \ xy \ \ y^2 \ \ x \ \ y \ \ 1]^T \). The parameters \( \sigma \) are found by minimizing the sum of square algebraic distances between the curve (in our case ellipse) and the \( N \) data points \( x_i \) (in our case, the contour points):

\[
D(\sigma) = \sum_{i=1}^{N} F(x_i)^2
\]

• Set the disparity \( d \) to equal the maximum disparity value inside a square with a side length of \( a^2 \) centered at \((x, y)\)

• The size of a fruitlet is then equal to:

\[
\text{Size(fruitlet)} = \frac{\text{baseline} \times ma}{d} \quad (3.6)
\]

Eq. 3.6 can be obtained after performing the following few simplifications:

• Let the focal length equal to \( f \). Find the coordinates of the co-vertices \((x_1, y_1)\) and \((x_2, y_2)\) of the fitted ellipse:

\[
\begin{align*}
x_1 &= x + \frac{ma}{2} (\cos(\theta + \frac{\pi}{2})) \\
y_1 &= y + \frac{ma}{2} (\sin(\theta + \frac{\pi}{2})) \\
x_2 &= x - \frac{ma}{2} (\cos(\theta + \frac{\pi}{2})) \\
y_2 &= y - \frac{ma}{2} (\sin(\theta + \frac{\pi}{2}))
\end{align*}
\]

• Assume that the two pixels at \((x_1, y_1)\) and \((x_2, y_2)\) have the same disparity \( d \). Hence
after unprojecting \((x_1, y_1)\) and \((x_2, y_2)\) to 3D we get that:

\[
Z = Z_1 = Z_2 = \frac{f \times \text{baseline}}{d},
\]

\[
X_1 = \frac{x_1 \times Z}{f},
\]

\[
Y_1 = \frac{y_1 \times Z}{f},
\]

\[
X_2 = \frac{x_2 \times Z}{f},
\]

\[
Y_2 = \frac{y_2 \times Z}{f}.
\]

- The 3D size of the fruitlet is equal to:

\[
\text{Size(fruitlet)} = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2}
\]

\[
\text{Size(fruitlet)} = \sqrt{\left(\frac{x_1 \times Z}{f} - \frac{x_2 \times Z}{f}\right)^2 + \left(\frac{y_1 \times Z}{f} - \frac{y_2 \times Z}{f}\right)^2}
\]

\[
\text{Size(fruitlet)} = \sqrt{\left(\frac{\text{baseline}}{d}\right)^2[(x_1 - x_2)^2 + (y_1 - y_2)^2]}
\]

\[
\text{Size(fruitlet)} = \frac{\text{baseline}}{d} \sqrt{[ma \times \cos(\theta + \frac{\pi}{2})]^2 + [ma \times \sin(\theta + \frac{\pi}{2})]^2}
\]

\[
\text{Size(fruitlet)} = \frac{\text{baseline} \times ma}{d}
\]

### 3.2 Data Collection

The data is collected in an apple fruitlet orchard. Before the start of the data collection, 89 apple fruitlet clusters in total were selected for measurement and tagged with an apriltag as shown in Fig. 3.1. During data collection, a human operator manually moves around a stereo camera (Fig. 3.10) to capture several images of apple fruitlet clusters as well as collecting ground truth data measurements using a caliper (Fig. 3.11). Example of a series of images of an apple fruitlet cluster is shown in Fig. 3.12. Our objective is to use these images to obtain accurate fruitlet size measurements using computer vision techniques when compared to the ground truth data. In addition to taking images and caliper measurements, each fruitlet in a cluster is assigned an ID as shown in Fig. 3.1. This ID is used to identify and track each fruitlet across multiple days. The data was collected using a custom camera with an image resolution of 3376px \(\times\) 2704px. The clusters were imaged over a period of
10 days with 2-3 days separating two consecutive measurement days to allow for fruitlet growth. The number of fruitlets imaged on each day is reported in table 3.3.

We use the computer vision pipeline to generate size measurements for all fruitlets over the 5 measurement days. The april tags were used to identify each cluster and the IDs assigned to the fruitlets in the cluster (Fig. 3.1) were used to associate and track the same fruitlet across days.
3. Apple Fruitlet Sizing using Deep Learning Algorithms

3.3 Results

Figures 3.13 and 3.14 show the distribution of apple fruitlet sizes measured via caliper (ground truth) and using the computer vision pipeline. The ”x” symbol indicates to the mean value and the horizontal line the median. It can be seen that the two measurement methods generate similar trends over the 5 measurements days. The mean and median values are reported in tables 3.4 and 3.5.

Table 3.4: Mean and Median (in mm) of the ground truth fruitlet size measurements

<table>
<thead>
<tr>
<th></th>
<th>May 27</th>
<th>May 29</th>
<th>June 1</th>
<th>June 3</th>
<th>June 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>5.280</td>
<td>6.199</td>
<td>7.885</td>
<td>11.362</td>
<td>14.647</td>
</tr>
<tr>
<td>Median</td>
<td>4.9</td>
<td>5</td>
<td>5.9</td>
<td>12.65</td>
<td>16.7</td>
</tr>
</tbody>
</table>

Table 3.5: Mean and Median (in mm) of the computer vision measurements

<table>
<thead>
<tr>
<th></th>
<th>May 27</th>
<th>May 29</th>
<th>June 1</th>
<th>June 3</th>
<th>June 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>5.528</td>
<td>6.609</td>
<td>7.981</td>
<td>11.262</td>
<td>14.208</td>
</tr>
<tr>
<td>Median</td>
<td>5.21</td>
<td>5.505</td>
<td>6.34</td>
<td>12.2</td>
<td>15.9</td>
</tr>
</tbody>
</table>

Figure 3.13: Distribution of ground truth fruitlet sizes.
3. Apple Fruitlet Sizing using Deep Learning Algorithms

Figure 3.14: Distribution of fruitlet sizes measured by the computer vision pipeline

Figure 3.15 shows the average error in sizes generated by the computer vision pipeline compared against the ground truth caliper measurements for each collection day. The average error across the five collection days was less than 1mm with higher error observed on earlier days due to fruitlets being relatively smaller in size.

Figure 3.15: Average computer vision measurement error (in mm) compared against ground truth caliper measurements.
Figures 3.16 and 3.17 show the distributions of apple fruitlet sizes for two collections days (5/27 and 06/01) from measurements collected using a caliper (in orange) and computer vision (in blue). We notice that the two distribution follow similar trends: Gaussian on earlier days and then bimodal for later collection days as fruitlets either abscise or continue to grow.

![Ground truth measurements](image1)

![Pipeline measurements](image2)

Figure 3.16: *Comparing the distributions of apple fruitlet sizes from ground truth measurements against the computer vision pipeline for the collection day 05/27/2020*
3.4 Discussion

We conclude that the computer vision measurements are generally close to the ground truth sizes in terms of both individual fruitlet sizes and the general shape of the size distributions. However, there are still differences that we attribute to the following:

- Measurements using the caliper return the true diameter of each fruitlet while in the proposed computer vision pipeline, the minor axis of a fitted ellipse is measured. Therefore, the accuracy of our method depends on how much of the fruitlet surface is visible from the camera view.
- Partially occluded fruitlets leading to incorrect segmentations and hence, incorrect size measurements.
- Error in the depth estimates
- Accuracy of the segmentation network
A major bottleneck in the proposed pipeline is tracking a fruitlet over multiple days to compute its size and measure its growth rate. To compare computer vision measurements against ground truth in the experiments described in this chapter, each fruitlet in the image was manually matched to its ground-truth value using the assigned fruitlet IDs (Fig. 3.1) which is a time-consuming and laborious process. Hence, automatic association of fruitlets in both time and space is important in the future to remove the need of manually tagging clusters, assigning IDs to fruitlets, and performing manual matching of results.

Finally, automating the fruitlet measuring process will allow for collecting more data measurements that can enable the creation of new precision thinning models based on size distributions instead of tracking and measuring individual fruitlets which is the current practice [18]. Growers usually have a targeted number of fruitlets to should remain on each tree to allow for optimal growth. Distributions such as 3.16 and 3.17 (but created from a larger population of fruitlets) can give growers a better understanding of how many fruitlets will naturally abscise without the application of chemicals. Hence, more informed decisions can be made on the amount of thinning chemicals that need to be applied to reach the targeted number of fruitlets. This has an immediate environmental impact as it can result in fewer applied chemicals. Pushing toward this goal, we propose in chapter 6, a data collection process that utilizes a robotic arm and an in-hand camera to collect a large number of apple fruitlet clusters.
Chapter 4

3D Sorghum Mapping

Visual SLAM systems are an essential component in agricultural robotics that enable autonomous navigation and the construction of accurate 3D maps of agricultural fields. However, lack of texture, varying illumination conditions, and lack of structure in the environment pose a challenge for Visual-SLAM systems that rely on traditional feature extraction and matching algorithms such as ORB or SIFT. In this chapter, we propose 1) an object-level feature association algorithm that enables the creation of 3D reconstructions robustly by taking advantage of the structure in robotic navigation in agricultural fields, and 2) An object-level SLAM system that utilizes recent advances in deep learning-based object detection and segmentation algorithms to detect and segment semantic objects in the environment used as landmarks for SLAM. We test our SLAM system on a stereo image dataset of a sorghum field. We show that our object-based feature association algorithm enables us to map 78% of a sorghum range on average. In contrast, with traditional visual features, we achieve an average mapped distance of 38%. We also compare our system against ORB-SLAM2, a state-of-the-art visual SLAM algorithm.

4.1 Proposed Method

4.1.1 System Overview

We describe our system’s main blocks shown in Fig. 4.1: the Detection-and-Segmentation block takes an image as an input and outputs a set of extracted semantic keypoints. The Object-Level-Feature-Association block performs temporal or stereo matching between two sets of semantic keypoints. Iterative Closest Point (ICP) is used to determine the relative transformation between two frames and hence, an estimate of the current camera pose. Initial estimates of the 3D landmarks and camera poses are used as the initial guess for a factor graph-based non-linear optimization, which returns estimates of the camera poses and landmarks’ 3D location up to time $t$. 

4. 3D Sorghum Mapping

4.1.2 Feature Extraction

Our feature extraction pipeline uses the detection and segmentation presented in Chapter 3. A Faster-RCNN network with a VGG16 backbone is used as the detection network and returns a bounding box for each seed seen in the image. The network was trained using the...
alternating optimization method with the learning rates: $[80000 \ 40000 \ 80000 \ 40000]$. The anchor window size parameters were set to: base size = 8, ratios = 0.5, 1, 2 and scales $= 2^i$ for $i \in [0, 1, 2, 3, 5, 6]$. Each image was split into overlapping tiles (an overlap of 20px in width and height) which were then used for training. During inference, each incoming image is first split into tiles and inference is performed on each tile independently. Finally all detected bounding boxes are remapped to original image size and Non-Max Suppression (NMS) is used to remove overlapping detections (specifically in the overlap areas between tiles). Fig. 4.2 shows an example of sorghum seed detections.

Each bounding box, generated by Faster-RCNN, is cropped out and passed to the pix2pix network, which generates a new image with a segmentation mask for the detected seed. An ellipse is then fitted to the segmented area, and the ellipse’s center is used as one semantic keypoint. The Faster-RCNN and pix2pix networks were trained on an Nvidia RTX 2080 GPU.

![Detection and Segmentation Pipeline](image)

Figure 4.3: *Our detection and segmentation pipeline. The center of the each ellipse is considered the projection of a 3D landmark.*

### 4.1.3 Data Association Algorithm

The data association algorithm considers the detected seeds in an image as nodes in a graph and makes the following observations: 1) the position of each node relative to its surrounding nodes should stay approximately constant across images. 2) the difference in 2D coordinates of the same node appearing in two consecutive images (or in a stereo image pair) is predominantly defined by a horizontal translation (Fig. 4.6).

We run the detection and segmentation pipeline at time $t$ on both the left and right stereo images. We obtain two sets of points: $C^L_t$, which is the set of 2D coordinates corresponding to the centers of the sorghum seeds that appear in the left image, and similarly, $C^R_t$ which we obtain from the right image. In stereo matching, the objective is to associate each point in $C^L_t$ to its more likely image in $C^R_t$. In temporal matching, the aim is to associate each point in $C^L_{t-1}$ to its more likely image in $C^L_t$. Our data association algorithm is agnostic to what type of matching we are performing since the objective is the same in both cases: assign each 2D point in a set $U$ to its more likely pair in another set $V$. 


4. 3D Sorghum Mapping

Figure 4.4: An example of the same sorghum seed imaged in 2 consecutive images where its position in the right image (time = t+1) relative to the left image (time = t) is predominantly defined by a horizontal translation.

Figure 4.5: Proposed feature association pipeline.

Defining the Linear Sum Assignment Problem

The object-level data association algorithm between stereo pairs or successive temporal frames is framed as a linear sum assignment problem (LSAP) optimization problem [3]. We define a bipartite graph $G = (U, V, E)$. Each vertex $s_{ab} \in U$, with coordinate $(a, b)$ in 2D camera frame, corresponds to the projection of a 3D landmark onto image A. Similarly, each vertex $s_{xy}$ with coordinates $(x, y) \in V$, corresponds to the projection of a landmark onto image B. An edge $c_{ij} \in E$ between nodes $s_{ab}$ and $s_{xy}$ defines the cost of associating $s_{ab}$ to $s_{xy}$. By introducing an assignment matrix $\varphi$, LSAP can be framed as the following optimization problem:

$$\min_{\varphi \in S_n} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \varphi_{ij} \text{ subject to: } \sum_{j=1}^{n} \varphi_{ij} = 1, \ i \in U \quad (4.1)$$

$$\sum_{i=1}^{n} \varphi_{ij} = 1, \ j \in V \quad (4.2)$$

$$\varphi_{ij} \in \{0, 1\} \quad (4.3)$$
4. 3D Sorghum Mapping

Where $S_n$ is the set of all possible assignment of nodes in $U$ to nodes in $V$.

**Cost function**

Since sorghum panicles are rigid bodies in space, the distance from a particular seed to its neighboring seeds should stay approximately constant as the robot moves in the environment. In terms of our graphical model, we add the constraint that the sum of the Euclidean distances of a node $s_{ab} \in U$ to its surrounding nodes in $U$ should be approximately equal to the sum of the Euclidean distances of the $s_{xy} \in V$ and its surrounding nodes in $V$. We define a heuristic cost function that captures this relative geometric structure between the landmarks:

For each node $s_{ab} \in U$, define four sets of neighbouring nodes: $L_{ab}$ (left), $R_{ab}$ (right), $T_{ab}$ (top), and $B_{ab}$ (bottom) satisfying the following conditions:

$$L_{ab} = \{ \forall s' = (c, d) \in U \mid 0 < a - c < \Delta \text{ and } |d - b| < \epsilon \} \tag{4.4}$$

$$R_{ab} = \{ \forall s' = (c, d) \in U \mid 0 < c - a < \Delta \text{ and } |d - b| < \epsilon \} \tag{4.5}$$

$$T_{ab} = \{ \forall s' = (c, d) \in U \mid 0 < b - d < \Delta \text{ and } |c - a| < \epsilon \} \tag{4.6}$$

$$B_{ab} = \{ \forall s' = (c, d) \in U \mid 0 < d - b < \Delta \text{ and } |c - a| < \epsilon \} \tag{4.7}$$

We define $L_{xy}$, $R_{xy}$, $T_{xy}$ and $B_{xy}$ similarly for $s_{xy} \in V$. The cost of associating node $s_{ab}$ to node $s_{xy}$ is defined as:

$$C(s_{ab}, s_{xy}) = r \cdot C'(L_{ab}, L_{xy}) + r \cdot C'(R_{ab}, R_{xy}) + r \cdot C'(B_{ab}, B_{xy}) + r \cdot C'(T_{ab}, T_{xy}) + |b - y| \tag{4.8}$$

where:

$$C'(X, Y) = \sum_{s' \in X} \sqrt{(s'(x) - a)^2 + (s'(y) - b)^2} \over \sum_{s' \in Y} \sqrt{(s'(x) - x)^2 + (s'(y) - y)^2}$$

$r$ is a constant, and $|b - y|$ is a term added to penalize matching landmarks with high vertical difference $|b - y|$. $s'(x)$ and $s'(y)$ are respectively the $x$ and $y$ coordinate of one of the surrounding nodes. The Hungarian algorithm [3] solves the LSAP in $O(n^3)$. Efficient CUDA-based implementations running on Nvidia GPUs [28] provide significant speedups compared to CPU-based implementations.

**Cost as a matching confidence measure**

The Hungarian algorithm returns an assignment matrix $\varphi$, which is a bijection from $U$ to $V$. Row $i$ in $\varphi$ contains all zeros except at a column $j$ where $\varphi_{ij} = 1$ meaning that node $i$ has been matched with node $j$. This assignment corresponds to the cost $c_{ij}$. Filtering out assignments where $c_{ij} > \epsilon$ allows us to keep only high confidence matches (matches with low cost). Figure 4.6 shows examples of semantic matches over a sequence of 5 images.
4. 3D Sorghum Mapping

Figure 4.6: (a) The First row shows the bounding box detections. Each bounding box bounds one sorghum seed. The second row shows the output of the proposed data association pipeline for five consecutive images. (b) We note that the displacement of a projected 3D landmark between 2 consecutive frame is predominantly defined by a horizontal translation.

4.1.4 3D Pose Estimation

Rigid Body Motion

A rigid body \( B \) is composed of a set of points \( \{ p_i \} \subset \mathbb{R}^3 \) such that the distance between any two points \( p_1 \subset B \) and \( p_2 \subset B \) remains constant during motion or if forces are exerted on the body. Rigid body motion are described by a translation \( t^b_a \in \mathbb{R}^3 \) and a rotation \( R^b_a \in SO(3) \) of the coordinate frame \( B \) attached to the body relative to a world/inertial coordinate frame \( A \). We define the Special Euclidian Group \( SE(3) \) as the product space of \( \mathbb{R}^3 \) with \( SO(3) \): \( SE(3) = \mathbb{R}^3 \times SO(3) \).

Using homogeneous coordinate systems, we can represent a transformation \( P \in SE(3) \) using a \( 4 \times 4 \) matrix as

\[
P = \begin{bmatrix}
R & t \\
0 & 1
\end{bmatrix}
\]

\( SE(3) \) is a group under matrix multiplication. Hence, we can compose multiple rigid body transformation to form a new transformation. Given \( P^b_a \in SE(3) \) the transform from frame \( B \) to frame \( A \) and \( P^c_b \in SE(3) \) being the transform from frame \( C \) to frame \( B \), then the transform from frame \( C \) to frame \( A \) is given by \( P^c_a = P^b_a P^c_b \). A point in \( \mathbb{R}^3 \) expressed relative to frame \( C \) (denote the coordinates of \( p \) in frame \( C \) as \( p_c \)) can be expressed in frame \( A \) (denote the coordinates of \( p \) in frame \( A \) as \( p_a \)) by left multiplying with the transform \( P^b_a \):

\[
p_a = P^c_a p_c
\]

Point-to-Point Iterative Closest Point

Iterative Closest Point (ICP) is an algorithm used to perform pointcloud registration: Find a transformation \( T \) that aligns two pointclouds \( S \) (source) and \( D \) (target). First,
a correspondence set $\mathcal{K} = (s, d)$ is computed with $p \in \mathcal{S}$ and $d \in \mathcal{D}$ and second, the transformation $T$ is found by minimizing an objective function $\mathcal{O}(T)$. In the Point-to-Point ICP variant $\mathcal{O}(T)$ is:

$$T_{\text{opt}} = \arg\min_T \mathcal{O}(T) = \arg\min_T \sum_i ||s_i - T \cdot d_i||^2, \quad T_{\text{opt}} \in SE(3)$$

### 3D pose estimation in sorghum field

Given stereo matches $S_t = \{x_t, y_t, u_t\}$ at time $t$ and stereo matches $S_{t-1} = \{x_{t-1}, y_{t-1}, u_{t-1}\}$ at time $t - 1$ where $u$ refers to the $x$ coordinate of the matched keypoint in the right stereo image, two pointclouds $PC_{t-1} = \{p_i\}$ and $PC_t = \{q_j\}$, where $p_i, q_j \in \mathbb{R}^3$, can be obtained by unprojecting $S_t$ and $S_{t-1}$ respectively to 3D.

$$p_i^x = \frac{(x_t - c_x) \cdot p_i^z}{f}; \quad p_i^y = \frac{(y_t - c_y) \cdot p_i^z}{f}; \quad p_i^z = \frac{b \cdot f}{d} \quad (4.9)$$

and similarly for $q_j$. $b$ refers to the stereo baseline, $d = x - u_R$ is the disparity in pixels and, $f, c_x$, and $c_y$ refer to the intrinsic camera parameters. The relative pose transformation $\hat{P} \in SE(3)$ between the previous and current camera pose is obtained by minimizing the translation between the two pointclouds and assuming zero rotation (since the robot moves in a straight line while capturing stereo images from the sides, the transformation should be defined mostly by a translation). The translation vector is then projected onto the direction of motion (known as a prior). That is:

$$\hat{P} = \begin{bmatrix} I_{3 \times 3} & \Pi(t) \\ 0 & 1 \end{bmatrix} \text{ with } \Pi \text{ being the orthogonal projection}$$

operator, and $t$ found by minimizing the error term:

$$E = \sum_{i=1}^N ||\hat{R}p_i + t - q_i||^2 \quad \text{with} \quad \hat{R} = I_{3 \times 3} \quad (4.10)$$

The estimated camera pose at time $t$ is then $P_t = P_{t-1}\hat{P}$. This variant of point-to-point ICP has been implemented in Libpointmatcher [40].

### 4.1.5 Factor Graph Optimization

#### Factor Graphs

Factor graphs are powerful graphical models widely used to perform inference in a number of robotics application such as SLAM and structure from motion. Factor graphs are bipartite graphs $F = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ where $\phi_i \in \mathcal{U}$ represent factors, $x_j \in \mathcal{V}$ are variables and $e_{ij} \in \mathcal{E}$ are edges linking factors with variables. Factor graphs represent the factorization of a function $F(X) = \prod_i \phi_i(X_i)$ with edges indicating the subset of variables that each factor depend on and thus, informing about the independence relationship between variables. For example, the factor graph in Fig. 4.7 represents the factorization of the function:

$$\phi(X) = \phi(x_1, x_2, x_3, x_4) = \phi_1(x_1)\phi_2(x_1, x_2)\phi_3(x_2, x_3)\phi_4(x_3)\phi_5(x_2, x_4)$$
Factor graphs can be used to solve MAP estimation problem over the variables $(x_1, x_2, x_3, x_4)$:

$$X_{MAP} = \arg\max_X \phi(X) = \arg\max_X \prod_i \phi_i$$

### Robust Huber Estimator

Least squares is a technique used to approximate the solution of overdetermined systems and is used in a number of applications namely data fitting (parameter estimation). Given $N$ observations $(x_i, y_i)$ and a model function $f(x, \theta)$ parametrized by vector $\theta$, the objective is to adjust the value of $\theta$ to best fits the data. The least-square problem can be defined as minimizing the sum of a function of residuals (the ground truth observation minus the value predicted by the model given parameters $\theta$):

$$S = \min_{\theta} \sum_i H(r_i)$$

where $r_i = y_i - f(x_i, \theta)$. In ordinary least-squares, $H$ is simply the squared norm:

$$S_1 = \min_{\theta} \sum_i ||r_i||_2^2$$

Ordinary least squares is however not robust in the presence of outliers since all residuals are weighted equally in the cost function $S_1$. Therefore one outlier with a high residual can heavily influence the result.

A solution is to use robust M-estimators such as the Huber loss which weights the residuals differently depending on how how large the residual is. The equation of the Huber estimator is:

$$H(r_i) = \begin{cases} 
\frac{r_i^2}{2} & |r_i| \leq \sigma \\
\sigma |r_i| - \frac{\sigma^2}{2} & \text{otherwise}
\end{cases}$$

It can be seen in Fig. 4.8 that the Huber loss grows quadratically from small residuals and linearly for large values of $r_i$ while in squared error loss is quadratic over all values of $r_i$. 
4. 3D Sorghum Mapping

Figure 4.8: Huber loss with $\sigma = 1$ in orange. The squared error loss in green

Figure 4.9: Factor graph representation

Factor Graph for Sorghum Mapping

We use our data association algorithm to find temporal matches $M_t$ between the previous and current frame: for every landmark seen in the left stereo image at time $t - 1$, we find its corresponding 2D projection in frame $t$.

We add a landmark to the factor graph if it is observed by at least two different camera poses. To obtain a good initial guess of the 3D location of landmark $i$ observed at time $t$, we average out all its previous 3D location estimates as follows:

- If landmark $i$ has been observed from exactly one previous camera pose $P_j$ where $j \in [1 \ldots t - 1]$, we set its 3D coordinate estimate in world reference frame to

\[
l_i^W = \frac{PC_j^W[i] + PC_t^W[i]}{2} \in \mathbb{R}^3
\]

(4.11)

where $PC_j^W[i]$ and $PC_t^W[i]$ are obtained by transforming the landmark’s 3D location
estimates $PC_j[i]$ and $PC_t[i]$ from their respective camera frames to the world reference frame.

- If landmark $i$ has been seen from $N > 1$ previous poses in $[P_1 \ldots P_{t-1}]$, we extract out its 3D optimized coordinate ($l_W^{\text{optimized}}$), from the previous optimization step and set the new 3D coordinate estimate of landmark $i$ to:

$$l_W^t[i] = N \cdot l_W^{\text{optimized}} + PC_W^t[i] \in \mathbb{R}^3$$  \hspace{1cm} (4.12)

The estimates of the 3D landmark location and camera pose are used as the initial guess for the non-linear optimization. We construct the non-linear factor graph in GTSAM at time $t$ as shown in Fig. 4.9. Each factor in the factor graph is a stereo projection factor, a constraint between pose $P_i$ and a landmark $l_W^j$. It is composed of a \textit{StereoPoint2} $(x,u_R,y)$ where $(x,y)$ is the projection of landmark $l_W^j$ onto the image plane at pose $P_i$, and $u_R$ is the $x$ coordinate of the landmark in the corresponding right stereo image taken at pose $P_i$. We use the Huber robust noise model which allows for modeling outliers. The estimated pose $\tilde{P}$ between consecutive frames also specifies a motion constraint between the two camera poses $P_i$ and $P_{i+1}$. The motion noise model is tuned according to our prior on camera motion (i.e. we specify a small uncertainty in the rotation and high uncertainty in the direction of motion $\Pi(\tilde{t})$). The factor graph is optimized using a Dogleg batch optimizer.

### 4.1.6 Pointcloud Postprocessing

Due to the potential false positives and false negatives produced by Faster-RCCN as well as the filtering of matches described in section III.C.3, the final optimized pointcloud can contain outliers as well as missing seeds. After the retrieval of the optimized camera poses and landmarks, we project the centers of all 2D segmented ellipses $\{s_i\}$ in each left stereo image to 3D world coordinates using equation 4.9 to get the set of 3D points $\{X_i\}$. (If a center $s_i$ was filtered during the stereo matching process, we assign it the same depth as its closest stereo-matched 2D neighbor). We add $X_i$ to the final pointcloud if no point $X'$ exists such that $|X' - X_i| < T$. Finally, for each point $X_i$ in the pointcloud, we calculate the variance of the distance between $X_i$ and its $N$ closest neighbors and reject $X_i$ if the variance is greater than a threshold. The variance threshold and $N$ are both determined experimentally.

### 4.1.7 Summary of System

In this section, we give a summary of the proposed system (Fig. 4.1): For every new stereo frame (time $t$), the Detection-and-Segmentation block detects and segments out all seeds in the right and left stereo images and fits an ellipse to each segmented area. The centers of all ellipses are extracted to obtain two sets of semantic keypoints $C_L^t$ from the left image and $C_R^t$ from the right image, which are then matched using the Object-Level-Feature-Association block to obtain a set of stereo matches. The stereo matches are unprojected to 3D to get a pointcloud $PC_t$. Each point in the pointcloud corresponds to an unprojected center of one sorghum seed. We obtain the initial estimate of the pose $P_t$ at time $t$ by solving for
the relative transformation \( \tilde{P} \) between \( PC_t \) and \( PC_{t-1} \) using Iterative Closest Point (ICP) and constraining the rotation matrix to being identity. We also perform temporal feature association between the segmented seeds from the left image at time \( t (C^L_t) \), and the left image at the previous timestep \( t - 1 (C^L_{t-1}) \). The 3D location estimate of already seen landmarks is refined, and new landmarks are added to the factor graph. We then run a batch optimizer to obtain the estimated 3D location of all seeds and the camera trajectory up to time \( t \). An example of a generated 3D reconstruction is shown in Fig. 4.10.

4.2 Results

![Diagram of a sorghum row in a sorghum field]

Figure 4.11: Layout of a sorghum row in a sorghum field

The system was tested on a stereo image dataset of a sorghum field collected on August 21, 2018, in Florence, SC. It was captured using a custom stereo camera with a 0.11m baseline at a rate of 5Hz with an image resolution of 4096px \( \times \) 3000px. The camera was mounted on a mobile field robot [33]. Sorghum fields are composed of rows, and each row
is composed of several ranges. A range is ≈ 4m long and may contain a different variety of sorghum. Empty spaces with no plants (≈ 1.5m long) separate two consecutive ranges (see Fig. 4.11). First, we show in Fig. 4.12 examples of matches obtained using our data
association algorithm versus matches obtained using the OpenCV implementations of the SIFT, SURF, AKAZE, and ORB algorithms with a Brute Force Descriptor Matcher. Fig. 4.12 shows that our data association algorithm produces accurate semantic matches, while the remaining four algorithms return a high number of incorrect correspondences. We note that the results shown in Fig. 4.12 are before applying the additional motion constraints on the traditional features matching methods. However, the motion constraints are applied for all methods when performing the quantitative evaluation in subsection 4.2.1.

4.2.1 Maximum Distance Mapped

We use the Maximum Distance Mapped as a comparison and indicator metric for the stability of SLAM systems and performance of data association algorithms. The presence of many erroneous matches or few detected features affect the performance and stability of SLAM algorithms. For example, in ORB-SLAM2, a small number of detected/matched features could cause the system to enter the “lost” state. Particularly in agricultural settings, traditional feature detectors have low accuracy due to the presence of repeated patterns and lack of texture. In our setting, the problem is further exacerbated by the lack of inertial sensory measurements and by the low frame rate of the stereo camera. Hence, we consider the maximum distance that a SLAM system was able to map before such failures occur as our comparison metric of interest. We report the results obtained from testing on image sequences of 8 sorghum ranges. First, we compare our feature-level data association with existing traditional feature extraction algorithms. We replaced the Object-Level-Feature-Association block (Fig. 4.5) with the existing OpenCV keypoint detection implementations of SIFT, SURF, ORB, and AKAZE. We used a Brute force matcher for feature matching and filtered out the matches using D.Lowe’s ratio test with a ratio set to 0.8. We then also apply our prior constraints on the motion of the camera (i.e. use the same assumptions about the camera motion made in our proposed data association algorithm). We disregarded all matched keypoint pairs \((x_1, y_1), (x_2, y_2)\) where \(|y_2 - y_1| > \epsilon\) given our prior knowledge that the displacement of a keypoint between 2 consecutive frames should mostly be defined by a horizontal translation (Fig. 4.6). We noticed that resizing the images and decreasing the image resolution worsens the OpenCV keypoint detection and matching algorithms’ performance. Thus, we conduct all tests on full resolution images.

Table 4.1 shows the maximum distance mapped in meters that was achieved with each method. Using our proposed feature matching algorithm, we can map 3 out of 8 ranges completely and map 65% of the remaining 5 ranges on average (78% on average across the 8 sorghum ranges). SIFT performed the best out of the remaining four algorithms with which we are able to map around 38% of the 8 sorghum ranges on average. The higher performance of our proposed method is primarily due to relying on the geometric relation between neighbouring seeds instead of on visual features.
Table 4.1: Maximum distance mapped (in meters). All methods compared under the same camera motion assumptions. Ground truth distances are extracted from GPS readings.

<table>
<thead>
<tr>
<th>Range</th>
<th>Feature detector + Matcher</th>
<th>ORB + BF</th>
<th>AKAZE + BF</th>
<th>OURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (3.56 m)</td>
<td>SIFT + BF</td>
<td>1.86 m</td>
<td>1.55 m</td>
<td>0.2 m</td>
</tr>
<tr>
<td>2 (5.00 m)</td>
<td>SURF + BF</td>
<td>0.25 m</td>
<td>0.25 m</td>
<td>Failed</td>
</tr>
<tr>
<td>3 (4.42 m)</td>
<td>ORB + BF</td>
<td>0.5 m</td>
<td>0.38 m</td>
<td>Failed</td>
</tr>
<tr>
<td>4 (4.1 m)</td>
<td>AKAZE + BF</td>
<td>1.47 m</td>
<td>0.6 m</td>
<td>Failed</td>
</tr>
<tr>
<td>5 (4.78 m)</td>
<td>OURS</td>
<td>0.57 m</td>
<td>0.74 m</td>
<td>Failed</td>
</tr>
<tr>
<td>6 (3.94 m)</td>
<td>Range 6</td>
<td>1.4 m</td>
<td>0.19 m</td>
<td>0.11 m</td>
</tr>
<tr>
<td>7 (5.03 m)</td>
<td>Range 7</td>
<td>3.15 m</td>
<td>0.26 m</td>
<td>Failed</td>
</tr>
<tr>
<td>8 (4.43 m)</td>
<td>Range 8</td>
<td>4.04 m</td>
<td>0.94 m</td>
<td>Failed</td>
</tr>
</tbody>
</table>

Object-Level SLAM systems that rely on prior object models or use specific representations for the landmarks (such ellipsoid) have limited applicability in our environment [36]. Therefore, we compare the performance of our SLAM algorithm against ORB-SLAM2, a state of the art feature-based SLAM system using DBoW2 for feature matching.

Table 4.2: Maximum distance mapped - ORB-SLAM2 vs OURS

<table>
<thead>
<tr>
<th>Range</th>
<th>ORB-SLAM2</th>
<th>OURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (3.56 m)</td>
<td>0.35 m</td>
<td>3.56 m</td>
</tr>
<tr>
<td>2 (5.00 m)</td>
<td>0.25 m</td>
<td>5 m</td>
</tr>
<tr>
<td>3 (4.42 m)</td>
<td>0.18 m</td>
<td>2.85 m</td>
</tr>
<tr>
<td>4 (4.10 m)</td>
<td>0.25 m</td>
<td>2.31 m</td>
</tr>
<tr>
<td>5 (4.78 m)</td>
<td>0.31 m</td>
<td>2.31 m</td>
</tr>
<tr>
<td>6 (3.94 m)</td>
<td>0.12 m</td>
<td>3.2 m</td>
</tr>
<tr>
<td>7 (5.03 m)</td>
<td>0.33 m</td>
<td>3.72 m</td>
</tr>
<tr>
<td>8 (4.43 m)</td>
<td>0.26 m</td>
<td>4.43 m</td>
</tr>
</tbody>
</table>

We report in Table 4.2 the distance mapped by ORB-SLAM2 before it enters the “lost” state after which, the system tries to re-localize at each new frame. We see that ORB-SLAM2 cannot map more than 0.35 m since very few keypoints could be detected and matched robustly across frames. This is consistent with the performance of ORB features when used as the frontend of our proposed SLAM system and is caused by the same reasons explained above (repeated patterns, lack of texture, and low camera frame rate).
4.3 Discussion

In this section, we presented an algorithm that reasoned about the geometric information in the scene to perform feature matching leading to improved results compared to existing SLAM systems that rely on traditional visual matching algorithms. However, the time complexity of the Hungarian algorithm makes our method better suited for offline-SLAM or for environments with few features to match. Hence, interesting directions for future work include evaluating the online performance in agricultural settings of learning-based methods such as [47], that incorporate both the positional and visual information of features. Future work also includes evaluating our method’s rotational drift, reasoning about occlusions to improve the matching accuracy as well as building 3D models for different types of crops.
4. 3D Sorghum Mapping
Chapter 5

Data Collection Using Autonomous Manipulation

In this chapter, a manipulation and planning strategy for an in-hand robotic arm is presented for autonomous data collection in apple orchards. Integration into a fully autonomous navigation pipeline is also described.

5.1 Mechanical Design

The mechanical system is depicted in Fig. 5.1. A 6DOF UR5 arm with a reaching range of \( \approx 1 \text{m} \) is used. A stereo camera equipped with a flash system is attached to the arm tool-end and used to capture images of tree canopies. The robot arm 6DOF is extended by adding a prismatic joint (1.35m long) to the base to obtain a 7 DOF manipulator. This kinematically redundant system offers several benefits:

- Increases the volume of the robot-arm workspace to cover the entire tree canopy without the need to move the ground robot.

- Enables the motion planner to find solutions for goal positions that would otherwise not be possible due to singularities or collisions.

- Increases the convergence time of motion planning algorithms as explained in [48].

More details about the 7th DOF are given in [48].
5. Data Collection Using Autonomous Manipulation

Figure 5.1: *The Warthog robot, UR5 arm, and the slider operating in the field*

![Image of the Warthog robot, UR5 arm, and the slider operating in the field]

5.2 System Architecture

Figure 5.3 depicts the system architecture. The arm trajectory planning routine selects a plane and plans a raster-scan-like path (shown in Fig. 5.4) for the arm to traverse. The stereo camera attached to the arm (baseline of 0.06m) captures images of the tree canopy at a rate of 5Hz while the arm is scanning the canopy. The 3D Reconstruction routine (chapter 6) then creates a 3D reconstruction of the scene using the in-hand camera images.

Figure 5.2: *The 7th DOF (left) and the UR5 robot arm (right)*

![Image of the 7th DOF and the UR5 robot arm]
5.3 Arm Trajectory Planning

5.3.1 Pointcloud Centroid Calculation

At the beginning of the pipeline (time t=0), the in-hand stereo camera starts at a home position similar to what is shown in Fig. 5.1 but with the stereo camera pointing forward, i.e. pointing to the tree canopy. Since the distance of the trees from the mobile robot may vary depending on the stopping position of the mobile robot and the structure of the tree, we need to determine a scanning plane that will more likely be free of obstacles (e.g. branches) over the entire trajectory. Hence, at t = 0, we fit a plane using RANSAC (Random sample consensus) to the pointcloud generated from the stereo images and compute the 3D coordinates of the centroid of the fitted plane:

\[
C = \begin{pmatrix}
X_c \\
Y_c \\
Z_c
\end{pmatrix}
\]
5.3.2 Adaptive Scanning Plane Selection

We select a plane parallel to the $y$ and $z$ axis at a depth $d$ where

$$\max(L, X_c - T) < d < \min(U, X_c - T)$$  \hspace{1cm} (5.1)$$

Tests have been conducted to select empirical values for $L$ and $U$ such that planned path has minimum unnecessary arm maneuvers (note that RRT-CONNECT is used to plan part of the trajectory). $T$ is a predefined distance separating the scanning plane from the pointcloud’s centroid $C$. We then compute 4 extreme waypoints on the plane at depth $d$:

- $U_L$: Upper leftmost waypoint
- $U_R$: Upper rightmost waypoint
- $L_L$: Lower leftmost waypoint
- $L_R$: Lower rightmost waypoint

5.3.3 Waypoint Generation

Once a scanning plane is determined, a set of ordered waypoints is computed (Fig. 5.5) for the robotic arm follow. Each waypoint is defined by the target 6DOF pose of the camera at that location enforcing that the camera faces the tree canopy. The height and number of intermediate waypoints depend on the distance of the arm from the canopy, the height of the tree and the number of desired horizontal waypoints. Hence, the waypoints computation is parameterized by two variables: $\text{recur\_depth}_v$ and $\text{recur\_depth}_h$. Given the 3D upper and lower points of the scan $U_L, U_R, L_L$, and $L_R$, the waypoints are computed by performing recursive midpoint computations (i.e return all midpoints between 2 given points up to a recursion depth). The main waypoint generation function is described in Algorithm 2.

**Algorithm 1 Recursive Midpoint Computation**

1: procedure \textsc{Recursive\_Midpoint} (\text{recur\_depth}, \text{index\_left}, \text{index\_right}, \text{pose\_vector})
2:   if \text{recur\_depth} == 0 then
3:     return
4:   end if
5:   \text{index\_mid} = \frac{\text{index\_left} + \text{index\_right}}{2}
6:   \text{new\_waypoint\_pose} = \frac{\text{pose\_vector}[\text{index\_left}] + \text{pose\_vector}[\text{index\_right}]}{2}
7:   \text{pose\_vector}[\text{index\_mid}] = \text{new\_waypoint\_pose}
8:   \text{RECURSIVE\_MIDPOINT}(\text{recur\_depth} - 1, \text{index\_left}, \text{index\_mid}, \text{pose\_vector})
9:   \text{RECURSIVE\_MIDPOINT}(\text{recur\_depth} - 1, \text{index\_mid}, \text{index\_right}, \text{pose\_vector})
10: end procedure
Algorithm 2  Waypoint Generation Procedure

1: procedure generate waypoints (recur_depth_v, recur_depth_h, U_L, U_R, L_L, L_R)
2:   num_entries_v = 1 + 2^{recur_depth_v}
3:   num_entries_h = 1 + 2^{recur_depth_h}
4:  \( \triangleright \) waypoints_vertical_left and waypoints_vertical_right are arrays of size num_entries_v will hold the values of all leftmost and rightmost waypoints respectively as shown in Fig. 5.5
5:  \( \triangleright \) Add the precomputed upper leftmost waypoint \( U_L \) to the front and lower leftmost waypoint \( L_L \) to end of waypoints_vertical_left
6:  \( \triangleright \) Add the precomputed upper rightmost waypoint \( U_R \) to the front and lower rightmost waypoint \( L_R \) to end of waypoints_vertical_right
7:  waypoints_vertical_left = new ARRAY[num_entries_v]
8:  waypoints_vertical_right = new ARRAY[num_entries_v]
9:  \( \triangleright \) Combine waypoints_vertical_left and waypoints_vertical_right into a single path while making sure the ordering of waypoints results in a raster-scan pattern
10: flip = 1
11: pose_final = newArray[num_entries_v \times num_entries_h]
12: for i=1:size(waypoints_vertical_left) do
13:   \( \triangleright \) tmp will hold the horizontal calculated waypoints for each 2 extreme vertical points
14:   tmp = new ARRAY[num_entries_h];
15:   tmp[0] = waypoints_vertical_left[i];
16:   tmp[num_entries_h-1] = waypoints_vertical_left[i];
17:   \( \triangleright \) Arm moves Left to right and then right to left etc.
18:   \( \triangleright \) Make sure to add waypoint at the correct index in to the final waypoint array
19: if flip == -1 then
20:    tmp = reverse(tmp)
21: end if
22: s = i*num_entries_h
23: for w=1:size(tmp) do
24:   poses_final[s+w] = tmp[w]
25: end for
26: flip = -flip
27: end for
28: return poses_final
29: end procedure
5. Data Collection Using Autonomous Manipulation

Figure 5.4: Illustration of the desired raster scan behavior

Figure 5.5: (RVIZ simulation) Calculated waypoints at a specific 3D location and orientation facing the tree canopy with $\text{recur\_depth}_v = 1$ and $\text{recur\_depth}_v = 0$
5. Data Collection Using Autonomous Manipulation

5.3.4 Arm Scanning Trajectory Planning

In this section, we set \( \text{recur\_depth\_v} = N \) and \( \text{recur\_depth\_h} = 0 \) (i.e. we have \( 2^N + 1 \) vertical scanning levels and 2 horizontal waypoints at each extreme). Example for \( \text{recur\_depth\_v} = N = 1 \) and \( \text{recur\_depth\_h} = 0 \) is shown in Fig. 5.5. We use the RRT-Connect planner and the ROS-Moveit library to plan for the scanning path. RRT-Connect [24] grows two rapidly-exploring random trees (RRTs) from both the source and the goal that advance toward each other. A solution is found when the two trees meet. The algorithm is shown in Fig. 5.6. As explained in [48], MoveIt internally plans for all 7 DOFs however, a custom hardware-level driver was developed that splits the trajectories (6 DOF of the arm and the 7th DOF which is the slider) in real-time to control all joints synchronously.

\[
\text{CONNECT}(T, q)
\]

1. repeat
2. \( S \leftarrow \text{EXTEND}(T, q); \)
3. until not \( (S = \text{Advanced}) \)
4. Return \( S \);

\[
\text{RRT\_CONNECT\_PLANNER}(q_{\text{init}}, q_{\text{goal}})
\]

1. F_a.init\( (q_{\text{init}}); F_b.init(q_{\text{goal}}); \)
2. for \( k = 1 \) to \( K \)
3. \( q_{\text{rand}} \leftarrow \text{RANDOM\_CONFIG}(); \)
4. if not \( (\text{EXTEND}(F_a, q_{\text{rand}}) = \text{Trapped}) \) then
5. if \( (\text{CONNECT}(F_b, q_{\text{new}}) = \text{Reached}) \) then
6. Return \( \text{PATH}(F_a, F_b); \)
7. SWAP\( (F_a, F_b); \)
8. Return \( \text{Failure} \);

Figure 5.6: The RRT-Connect Algorithm. Figure borrowed from [24]

Planning Strategy

We found that planning a path from the home position to the first waypoint and then from one waypoint to the next using RRT-Connect leads to unpredictable arm behavior due to the randomness of the algorithm (i.e. different paths can be planned for the same start and end waypoints in different runs of the algorithm) potentially leaving the arm in configurations where it might be difficult to further maneuver from. Therefore, planning was performed in steps a described in Algorithm 3.
5. Data Collection Using Autonomous Manipulation

Algorithm 3 Planning Algorithm

1: procedure PLAN_FOR_SCAN( LIST_WAYPOINTS)
2:     i = 0
3:     while i ≤ size(list_waypoints) do
4:         \( y_1 \) = Orthogonally project the center of list_waypoints[i]’s frame onto the slider (the slider is considered a line in 3D)
5:         Move slider from current position to point \( y_1 \) keeping the robot arm joint angles fixed. The stereo camera ends up with pose \( P \)
6:         Use the RRT-Connect planner to plan a trajectory from \( P \) to list_waypoints[i].
7:         Execute the plan
8:         \( y_2 \) = Orthogonally project the center of list_waypoints[i+1]’s frame onto the slider
9:         Move slider from current position to point \( y_2 \) keeping the robot arm joint angles fixed
10:     i = i+2
11: end while
12: end procedure

We note that moving the arm between the left and right waypoints on the same scanning height is performed by keeping the UR5 arm joints fixed and only moving the 7th DOF slider. RRT-Connect was used restrictively to plan from a waypoint to the waypoint immediately “under” it. i.e plan between adjacent configurations in configuration space. This, coupled with the post-processing and simplification of the solution performed by MoveIt, resulted in planned paths that are predictable (across runs of the algorithm) and that follow a raster scan behavior as desired.

5.4 Fully Autonomous Data Collection

The manipulation pipeline was integrated with an existing autonomous navigation pipeline based on the work in [48].

5.4.1 Navigation

An IMU (Inertial Measurement Unit) and an RTK-GPS receiver mounted on the robot were used as the main localization sensors. The RTK-GPS is highly accurate with an error of \( \sim \pm 0.5 \) cm in the horizontal direction and \( \sim \pm 1.5 \) cm in the vertical direction. A 2D extended Kalman Filter (EKF) fused the IMU and RTK positions to localize the robot. We manually collect RTK-GPS waypoints in addition to identifying waypoints, in front of selected trees, where the mobile robot has to stop and run the scanning pipeline (which we call scanning waypoints).

5.4.2 Finite State Machine

The high-level execution of the Manipulation, Navigation, and Slider Calibration module were coordinated via a finite state machine (FSM) which is shown in Fig. 5.7. The robot navigates between the rows of the apple orchards, stops in front of each selected tree,
5. Data Collection Using Autonomous Manipulation

Figure 5.7: Finite State Machine for Fully Autonomous Data Collections. The state machine transitions between navigation, calibration and autonomous scanning.

calibrates the slider, and finally scans the tree. Failure detection is performed in each module, leaving the FSM in an error state which serves as an indicator for human operators to intervene. Future work includes increasing the robustness and integration between the various components of the system.

5.5 Results

5.5.1 Autonomous Scanning

In-field Deployment

The autonomous scanning pipeline was used to collect an apple fruitlet dataset during a field trip to an apple orchard located in Amherst, Massachusetts from May 16th - May 26th 2021 (Aerial view shown in Fig. 5.8). The data collection consisted of manually guiding and stopping the ground mobile robot in front of an apple tree and then, manually starting the autonomous scanning routine. Two apple varieties and 24 trees per variety were selected for imaging. Approximately 300 stereo images were collected per tree. In addition, the robot arm joint angles, ROS tf transforms, and stereo camera parameters were recorded. The autonomous scanning pipeline successfully scanned all 48 trees for each of 3 collections days (72 scans in total) with no manual interventions. A single scan covers two trees and we define a successful scan as one that completes the trajectory and saves the data without manual intervention. Each scan took approximately 2 minutes and 30 seconds to complete.
Figure 5.8: *Aerial view of the apple orchard in Amherst, Massachusetts*

**Analysis of a Scan**

Figures 5.9, 5.10, and 5.11 show the x, y and z position of the camera expressed in world frame for five typical runs of the scanning routine.

In Fig. 5.9, we see that the camera maintains the same \( d \) position, i.e. the scanning depth determined by equation 5.1, over the entire duration. For each run, a scanning plane, specific to this particular scanning run, is determined and maintained over the entire trajectory.

![The x translational component of the in-hand camera wrt to world frame](image)

**Figure 5.9: x position of the camera in world frame**
Figure 5.10 shows the $y$ position of the camera as the scanning run proceeds with the parameters $\text{recur\_depth_v}$ and $\text{recur\_depth_h}$ set to 2 and 0 respectively. In other words, we have $2^2 + 1 = 5$ vertical waypoints and 5 transitions from the extreme points of the scanning plane (left to right and right to left). The $y$ position of the camera intersects the origin 5 times indicating 5 successful transitions.

The $y$ translational component of the in-hand camera wrt to world frame

Finally, Fig. 5.11 shows the $z$ position of the camera over the entire trajectory of a scan. 5 different scanning heights (ranging from $\sim 1.8m$ to $\sim 1.25m$) corresponding to 5 transitions between vertical waypoints can be seen.
5.5.2 Fully Autonomous Data Collection

A dataset of 20 apple trees of a single variety was collected using the fully autonomous system on May 25th, 2021. The dataset consists of the stereo images only. 24 scanning waypoints were selected. From each scanning waypoint, two trees could be scanned with a single scan run. Figure 5.12 shows the complete desired path for the robot. The blue triangles indicate the approximate location of the trees in front of which the robot needs to stop to run the scanning pipeline once from each side. The robot starts at the Start location, moves forward to point A and performs a turn to enter the next row at point B. The robot then continues forward to point C, performs a turn and re-enters the same row at point C. Finally, the robot continues to point B, performs a turn and continues to the End point.

Navigation

Figure 5.13 shows the result of the autonomous navigation task. the triangle in light green indicate the trees that the robot scanned successfully from both sides. The line in light green indicate the path that the robot was able to follow successfully. The mobile robot successfully followed the navigation waypoints which include turning from one row to the next. Two manual interventions were performed to put the mobile robot on track because it deviated heavily from the navigation waypoints and risked crashing. The robot also successfully stopped at ten scanning waypoints with an orientation parallel to the tree canopy which is the desired orientation. The orientation of the mobile robot directly affects the performance of the scanning routine: how much of the tree canopy’s surface is covered.

![The z translational component of the in-hand camera wrt to world frame](image.png)

Figure 5.11: z position of the camera in world frame
and whether obstacles exist along the scanning path. Due to software errors, the navigation was aborted at point D. The red line and triangles show the section of the desired path that the robot was unable to complete. Concretely, the robot successfully scanned $\approx 42\%$ of the trees designated for scanning.

Figure 5.12: The blue triangles indicate the location of the apple trees where the mobile robot needs to stop and run the scanning pipeline. The blue and black lines indicate the desired robot path.

Figure 5.13: Autonomous navigation result. the triangle in light green indicate the trees that the robot scanned successfully from both sides. The line in light green indicate the path that the robot was able to follow successfully. Red triangles and line indicate the section which was manually aborted.
5.5.3 Discussion

The apple trees encountered in the field can be considered mostly “flat” with limited depth in their canopies (example shown in Fig. 5.14). Hence, we found that 1) the scanning plane selection procedure and 2) stopping the mobile robot at distance of $\sim 1.5m$ to $\sim 2m$ away from the tree canopy, enough to prevent collisions between the arm and tree branches or fruitlets. However, future work can include integrating MoveIt with a 3D occupancy mapping libraries like OctoMap to reason about the obstacles in the scene and make this method applicable in agricultural fields where the trees do not have a flat structure. This can also enable performing more complex manipulation tasks in tree canopies.

Figure 5.14: Apple fruitlet trees in the field

Interesting direction for future work also include using computer vision methods, similar to the one described in chapter 3, to measure the size of fruitlets from the images captured using autonomous scanning. From analyzing and comparing these size measurements against 1) the ground truth caliper measurements and 2) the data collected using the hand held camera, it is possible to further adjust the scanning and planning procedure to capture better images and generate more accurate automated measurements in future data collections. Finally, further work includes working on the software integration between the navigation and scanning pipelines to enable a fully autonomous data collection system.
Chapter 6

3D Reconstruction and Mapping Using In-Hand Camera Images

This chapter describes a 3D reconstruction system using the robotic in-hand camera images and preliminary algorithms to perform apple fruitlet mapping in canopies.

6.1 3D Reconstruction

6.1.1 Object Detection

A Faster-RCNN network [43] is used to perform object detection of the apple fruitlets. 179 images were labeled for training (each of size 1440px × 1080px) with ~ 6200 bounding boxes. Each image was split into overlapping tiles for both training (for a total of ~ 1550 tiles used for training) and inference similar to section 4.1.2. The network was trained using the alternating optimization method with the learning rates: $[80000 \ 40000 \ 80000 \ 40000]$. Example of fruitlet detections on a testing image is shown in Fig. 6.1. The precision and recall values on the testing set are reported in tables 6.1 and 6.2.

Table 6.1: Precision and recall of the Faster-RCNN network on the training set

<table>
<thead>
<tr>
<th>IOU Threshold</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.800</td>
<td>0.898</td>
</tr>
<tr>
<td>0.6</td>
<td>0.789</td>
<td>0.885</td>
</tr>
</tbody>
</table>

Table 6.2: Precision and recall of the Faster-RCNN network on the testing set

<table>
<thead>
<tr>
<th>IOU Threshold</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.829</td>
<td>0.645</td>
</tr>
<tr>
<td>0.6</td>
<td>0.799</td>
<td>0.622</td>
</tr>
</tbody>
</table>
6. 3D Reconstruction and Mapping Using In-Hand Camera Images

6.1.2 Disparity Map Generation using SGBM

The Semi-global matching (SGBM) algorithm in OpenCV [21] was used to generate the disparity maps using the left and right stereo pairs. The stereo camera was re-calibrated in the field before each data collection day to ensure the generation of accurate disparities (an example is shown in Fig. 6.2).

(a) Original image of apple fruitlet clusters  
(b) Corresponding disparity map

Figure 6.2: Example of a disparity map generated using SGBM OpenCV

6.1.3 ROS Message Synchronization

The Faster-RCNN network inference time and the disparity map generation, which runs on CPU, may have different run times depending on the PC specs used to run the pipeline. Hence, synchronization of the output messages of the two nodes is necessary to prevent
incorrect pairing of messages which resulting in corrupted 3D scene reconstructions. Fig. 6.4 shows the method used to perform ROS message synchronization. The timestamp $T$ of the left stereo ROS message is extracted. Both the disparity generation and Faster-RCNN ROS nodes will set the timestamps of their output messages to be the same timestamp $T$. Once the Faster-RCNN detections are generated, a separate ROS node will be triggered to calculate the transform from the inhand link of the camera to the world frame with the outgoing message having timestamp $T$ as well. The ROS Approximate Time Policy message synchronizer, which uses the timestamps incoming messages to perform message pairing, is used to synchronize all incoming messages. The 3D reconstruction and mapping node then processes the synchronized messages.

![Figure 6.3: Method used to perform ROS message synchronization](image)

6.1.4 Pointcloud Transformation as an Initialization for ICP

Forward Kinematics

The forward kinematics of a robotic arm aims at calculating the configuration of the end-effector relative the base of the robot, given the relative transformations between each two adjacent links. Let $T_{l_{i-1}}^{l_i}$ be the transformation between links $l_i$ and $l_{i-1}$, then the transform from tool frame $t$ to robot base $B$ is given by:

$$T_B^t = T_B^{l_1} T_{l_1}^{l_2} \cdots T_{l_n}^t$$

Calculating the Transform from Camera frame to World frame

The disparity map is used to project the points to 3D using equations 4.2. However, this pointcloud is expressed in camera frame. We use the TF ROS package to extract a transform from the inhand frame to the world frame $P_{\text{inhand}}^{\text{world}}$ where the inhand frame is a user defined
6.3D Reconstruction and Mapping Using In-Hand Camera Images

frame in the robot URDF whose origin coincides with the camera frame but rotated by a 90 degree relative to the z axis (i.e. \( P_{\text{camera}}^{\text{inhand}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \)) and is a fixed transform throughout the scanning routine. Hence, an initial estimate of the transform from the camera frame to world frame is equal to \( P_{\text{camera}}^{\text{world}} = P_{\text{inhand}}^{\text{world}} P_{\text{camera}}^{\text{inhand}} \) and is applied to the pointclouds prior to ICP registration.

6.1.5 3D Pointcloud Merging

Point-to-plane Iterative Closest Point (ICP)

The point-to-plane ICP variant is used to perform pointcloud merging. Let \( S \) be the source pointcloud and \( D \) be the destination pointcloud. For a point \( s_i \in S \) and its corresponding point \( d_i \in D \), the objective function is to minimize the sum of squared distances between \( \{s_i\} \) and the tangent planes at \( \{d_i\} \). Let \( n \) is the unit normal at \( d_i \), then the point-to-plane ICP objective \( O(T) \) is:

\[
T_{opt} = \arg\min_T O(T) = \arg\min_T \sum_i ((T \cdot s_i - d_i) \cdot n_i)^2, \quad T_{opt} \in SE(3)
\]

![Figure 6.4: Point-to-plane ICP Method. Figure borrowed from [29]](image)

3D Pointcloud Merging

Once the pointcloud calculated from stereo images at time \( t \) (\( PC^t \)) is transformed to the fixed world frame, we pass it to PCL’s inbuilt statistical outlier filter (SOR) and voxel grid sampler to reduce its size and remove outliers. We then perform point-to-plane ICP to register \( PC^t \) with the merged 3D representation of the scene \( M^{t-1} \) which was computed by merging all pointclouds from times 0...\( t-1 \). ICP returns a transform \( T \). Finally, a “distance filter” is applied to \( M^t \) which removes all points whose depth is greater than a pre-defined threshold (in the future, this threshold can be a function of the centroid depth found in section 5.3.1). We also apply the SOR filter (every \( N \) frame only to prevent
degrading the completeness of the pointcloud) and the voxel grid sampler to $M^t$ before processing the next incoming pointcloud at time $t + 1$. Figure 6.6 shows an example of a generated pointcloud of an apple tree and Fig. 6.5 illustrates the pipeline.

![Diagram of pointcloud merging process](image)

Figure 6.5: Pointcloud Merging Process
We note that the processing rate of the 3D reconstruction node is heavily dependent on the pre-defined ICP and filter parameters. For example, with the default parameters shown below and running the pipeline on a PC with an NVIDIA 1050TI GPU and an Intel Core i7-8750H CPU @ 2.20GHz CPU, the pipeline processing rate was \(\sim 0.5\) frame per second. When used offline, the pointcloud merging method was successfully used to merge all 300 images collected during a scan.

- ICP
  - Max Correspondence Distance: 0.001
  - Max number of iterations: 55
- Voxel grid
  - Leaf size: 0.005
- SOR
  - Number of points for mean distance estimation: 10
  - Standard deviation multiplier threshold: 0.95
6.2 Apple Fruitlet Mapping in Tree Canopies

A preliminary algorithm to map apple fruitlets in tree canopies was implemented and initial results were generated. The main idea is to leverage the prior on the camera pose provided by forward kinematics and maintain a representation of the 3D world frame coordinates of all apple fruitlets that have been seen at times $0 \ldots t - 1$. Fruitlet matching is then performed in 3D world frame as follows:

- Let $P_{\text{camera} \rightarrow \text{world}}(t)$ be the transform described in section 6.1.4 computed at time $t$ and $T$ be the transform obtained from running ICP on $P_{\text{CAM}}^t$ and $M_{t-1}$.
- Suppose a fruitlet $A$ is observed from camera pose $j$ a time $j \in 0 \ldots t - 1$ then:
  - The center of the 2D bounding box detection of fruitlet $A$, $c^j_A$, is projected to 3D using the disparity map calculated at time $j$. We obtain the 3D coordinates $C^j_A$ in camera frame.
  - $C^j_A$ is transformed to world frame by applying the transform $T P_{\text{camera} \rightarrow \text{world}}(j)$ to get the 3D point $F^j_A$ (i.e $F^j_A = T P_{\text{camera} \rightarrow \text{world}}(j)C^j_A$).
- Suppose that the same fruitlet $A$ is observed from a different camera pose $t$ at time $t$. 

Figure 6.7: Algorithms for 3D Apple Fruitlet Mapping
6.3D Reconstruction and Mapping Using In-Hand Camera Images

If \( t > j \) then we perform the same operations:

- The center of the 2D bounding box of fruitlet \( A, c^t_A \), is projected to 3D using the disparity map calculated at time \( t \). We obtain the 3D coordinates \( C^t_A \) in camera frame.

- \( C^t_A \) is transformed to world frame by applying the transform \( TP^t_{\text{camera}}(t) \) to get the 3D point \( F^t_A \) (i.e. \( F^t_A = TP^t_{\text{camera}}(t)C^t_A \)).

- Since \( F^t_A \) and \( F^j_A \) are two different 3D location estimates in world frame of the same fruitlet \( A \) then the euclidean distance \(||F^t_A - F^j_A||_2\) ought to be “small”. And hence, the 2D measurements \( c^t_A \) and \( c^j_A \) can be associated.

In the next subsection, we present an algorithm that utilizes the above observation to perform 3D mapping in tree canopies.

6.2.1 Algorithm 1: Fruitlet Mapping by 3D Coordinates Averaging

Once we have the set of 3D locations of apple fruitlets detected at time \( t \) expressed in world frame \( \{F^t_i\} \), the Hungarian algorithm is used to associate the new fruitlets with those seen up to time \( t - 1 \), \( \{F^{t-1}_j\} \), in a similar way to what is described section 4.1.3. For each new fruitlet \( F^t_i = \begin{bmatrix} F^t_ix \\ F^t_iy \\ F^t_iz \end{bmatrix} \), the cost of associating it with a fruitlet \( F^{t-1}_j = \begin{bmatrix} F^{t-1}_jx \\ F^{t-1}_jy \\ F^{t-1}_jz \end{bmatrix} \) is \( c_{ij} = ||F^t_i - F^{t-1}_j||_2 \). The Hungarian matcher will return the optimal assignment from \( \{F^t_i\} \) to \( \{F^{t-1}_j\} \). An assignment with \( c_{ij} < \epsilon \) is considered a match and an assignment with \( c_{ij} > \epsilon \) is considered a potential new landmark. Given a match between \( F^t_i \) and \( F^{t-1}_j \), the 3D location estimate of \( F^{t-1}_j \) is then updated as follows:

\[
F^t_j = \frac{(N - 1) \cdot F^{t-1}_j + F^t_i}{N}
\]

where \( N - 1 \) is the number of times landmark \( j \) has been observed up to time \( t - 1 \).

Figure 6.8: Example of semantic matches over 4 consecutive frames with the camera moving “down” between 2 vertical waypoints (see Fig 5.5). Algorithm 1 was used to estimate the 3D location of fruitlets.

This concludes algorithm 1. The final map consists of all 3D landmarks \( F^t_j \), expressed in
world frame, that have been observed by at least \( X \) number of frames. \( X \) is an adjustable parameter. In the case where many false positives are returned by the detection algorithm, \( X \) is set to a high value so that only landmarks that are detected with higher confidence (i.e. consistently detected by the Faster RCNN network) are included in the final map. Figure 6.8 show examples of semantic matches over 4 consecutive frames with the camera moving “down” between 2 vertical scanning waypoints (see Fig 5.5).

Results

We conducted initial tests to quantify the 3D map’ accuracy by comparing the apple fruitlet count using algorithm 1 against the ground truth number of fruitlet in a tree canopy. In the following experiment, we set \( X = 3 \) frames and \( \epsilon = 0.02m \). Let \( \Pi(X) \) be the perspective projection of a 3D Point \( X \) expressed in camera frame onto the image plane. Given a camera \( n \) at time \( t \), the 3D fruitlets locations in the map are re-expressed in camera \( n \)’s frame of reference by applying the transform \( (TP_{\text{camera}})^{-1}(t) \). The points are then projected onto the image plane: \( \Pi(TP_{\text{camera}})^{-1}(t)X) \). Each 2D projection should theoretically correspond to the center of a 2D bounding box.

To evaluate our approach, we collected the following data:

- 18 fruitlet clusters were selected.
- For each cluster \( i \), we recorded the ground truth number of fruitlets \( G_i \) by manually counting the fruitlets from the images.
- We used algorithm 1 and the aforementioned procedure of projecting fruitlets onto the image plane. Then, for each cluster \( i \), we counted the number of fruitlets \( M_i \) (each 2D projection is counted as one fruitlet).
- For each cluster \( i \), We also determined the number of fruitlet \( N_i \) by simply counting the number of 2D bounding boxes for the cluster \( i \) in a single image (only counting true positives). Since the number of detected fruitlets in a cluster (number of bounding boxes) can vary from one frame to another, we set \( N_i \) to be the average of the minimum and maximum number of detected fruitlets per cluster across all frames where the cluster is visible. An example to clarify the procedure of calculating the value \( N_i \) is described in Fig. 6.9.
- Finally we calculated:

\[
G = \sum_{i=1}^{18} G_i = 84 \\
M = \sum_{i=1}^{18} M_i = 73 \\
N = \sum_{i=1}^{18} N_i = 58
\]
Figure 6.9: Example of how the value \( N_i \) is found for an example cluster. The figure consists of 3 consecutive images. The same cluster (circled in red) is visible in all three. In the first image \( 4/5 \) fruitlets were detected. In the second image, \( 4/5 \) fruitlets were detected, and finally \( 5/5 \) fruitlets were detected in the last image. Therefore \( N_i = (\min(4, 4, 5) + \max(4, 4, 5))/2 = 4.5 \)

We report the count accuracy values for the two approaches in Table 6.3.

Table 6.3: Accuracy values using algorithm 1 vs directly counting the fruitlets using the number of bounding boxes

<table>
<thead>
<tr>
<th>Algorithm 1 (( \frac{M}{N} ))</th>
<th>bounding boxes count (( \frac{N}{G} ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>87%</td>
<td>70%</td>
</tr>
</tbody>
</table>

Algorithm 1 shows a 17% improvement over the baseline in terms of total number of fruitlets counted. One of the primary advantages of fusing multiple images, in addition to handling the false negatives of the 2D detection network, is the ability to track fruitlets that may be occluded in one frame but visible from other frames as shown in Fig. 6.10.
6.3 Discussion

We note that algorithm 1 still falls short of ground truth. In addition to the inaccuracies of the 2D object detector, we observed that a number of 2D projections were not exactly overlapping with the centers of the fruitlet bounding boxes. In some instances, the 3D estimate error is high enough that the 2D projections do not land on fruitlets. These 3D location estimate errors are due to depth estimation errors and to camera pose transform errors caused by the vibration of the in-hand camera while performing the scanning routine (especially when the linear slider moves). Hence a potentially better approach is to use these 3D locations estimates as initialization to a factor graph based batch optimization over the poses of the camera and the 3D landmark locations. We conclude this chapter by giving a brief description and initial results for the proposed method using factor graphs.

6.3.1 Algorithm 2: Fruitlet Mapping using Factor Graphs

We use a similar approach to algorithm 1 for feature matching. In addition to the averaged 3D location of landmarks and the number of times each landmark was observed $N$, the 2D measurement of each landmark (the 2D projection of the landmark into the image plane) is also recorded for all frames. In the current implementation, the 2D measurements are the centers of the bounding boxes. The factor graph in Fig. 6.11 (implemented in GTSAM) is updated after processing each incoming frame. All factors between poses $P_i$ and landmarks $l^W_j$ are stereo projection factors. In addition, only landmarks that have been observed by at least $X$ frames are added to the graph. We tune the noise models in the factor graph to
have low uncertainty in camera poses and higher uncertainty in 2D measurement and 3D landmarks’ locations. Finally, the Dogleg batch optimizer is used to return the optimized poses and landmark locations (incremental smoothing algorithms can be considered in the future). We note that algorithm 1 can be considered a limiting case of algorithm 2 where we assume infinite certainty about the pose of the camera and only update individual states (position of the landmarks) independently of each other via averaging. Initial experiments with algorithm 2 consisted of visually inspecting the 3D models obtained from using the optimized factor graph poses to perform pointcloud merging. We noticed that the 3D registration was poor and produced worse results when compared to 3D models obtained using ICP-based registration. This result can be attributed to the high variance in the bounding boxes coordinates returned by the Faster-RCNN network leading to significant errors in the 2D projection measurements used as input to the factor graph. Hence, the camera poses resulting from batch optimization by minimizing the reprojection error loss will be inaccurate and as a result, the resulting 3D reconstruction will be poor. Obtaining more accurate 2D measurements can be done by training the Faster-RCNN on more labeled data, using the centers of the segmented fruitlets instead of the bounding box centers as 2D measurements (as done for sorghum mapping) and further experimenting with tuning the noise models of the factor graph.
Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we tackled the problem of perception and modeling in agriculture through the use of stereo cameras in the context of three applications:

- Chapter 3 presented a deep-learning based vision system that can measure apple fruitlets with less than 1 mm of error compared to ground truth measurements.

- Chapter 4 presented a SLAM system that takes advantage of the structure of agricultural fields and a data association framework that utilizes the segmented sorghum seeds as landmarks in the environment. We showed that our object-level data association algorithm enables our SLAM system to perform significantly better than traditional feature detection and matching algorithms in a setting with low texture, repetitive patterns, and images captured at a slow frame rate. Our system also performs better than existing state-of-the-art Visual-SLAM algorithms such as ORB-SLAM2 in terms of maximum mapped distance in challenging agricultural environments.

- Chapter 5 presented a manipulation and planning strategy for autonomous data collection in apple orchards using an in-hand robotic arm. Integration into a fully autonomous navigation pipeline was also described.

- Chapter 6 proposed a 3D reconstruction system, that uses the in-hand camera images. Initial algorithms to perform apple fruitlet mapping in canopies were also presented and analyzed.

7.2 Future work

Directions for future work include improving our apple fruitlet sizing system by further experimenting with detection and segmentation algorithms and reasoning about occluded fruitlets.

Future work pertaining to mapping sorghum fields include evaluating the rotational drift of the SLAM algorithm and using the system to reconstruct larger-scale 3D models for different types of crops. In addition, due to the high computational complexity of
the Hungarian Algorithm, the proposed method is better suited for offline-SLAM or in settings with few objects to match. Thus, another direction for future work includes investigating learning-based feature matching algorithms and their online performance in agricultural environments. Future work also includes reasoning about occlusions to improve data association accuracy and the quality of the SLAM solution.

With respect to mapping apple fruitlets in orchards, the presented algorithms in section 6.2 are preliminary and more thorough testing need to be conducted in the future to ensure their practical validity. As described in the results section of 6.2.2, the factor graph method requires more accurate 2D measurements which can be obtained by training a better detector network and using the centers of the segmented area of fruitlets as the 2D projections.

Finally, an interesting direction is tracking and measuring fruits over multiple days which will fully automate tasks such as predicting thinner response on apples as described in [18]. A potential solution is to leverage autonomous navigation and GPS-RTK to navigate in the field by following the same navigation waypoints on each collection day. Since GPS-RTK is highly accurate, the change in the reference world frame, which we defined as located at the base of the robot, will be relatively small. A 3D model of the same tree can be built and registered to the world frame on each day and geometric information such as the distance for a fruitlet to its neighboring fruitlets could be used to identify and match fruitlets across days.
Bibliography


[39] Tanvir Parhar, Harjat Bin Baweja, Merritt Jenkins, and George Kantor. A deep learning-


[48] Abhisesh Silwal, Francisco Yandun, Anjana Nellithimaru, Terry Bates, and George Kantor. A path toward fully autonomous robotic vine pruning. *To be submitted to Field Robotics*. 1, 5.1, 5.3.4, 5.4


[52] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano.


[55] Zhenyu Zhang, Stéphane Lathuilière, Andrea Pilzer, Nicu Sebe, Elisa Ricci, and Jian Yang. Online adaptation through meta-learning for stereo depth estimation, 2019. 2.4
