Automated phenotyping of seeds using traditional and convolutional neural network-based methods

by

Kai Zhao

M.S., Kansas State University, 2021

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2024

Abstract

The last decade has witnessed a rapid expansion of computer vision and scientific computing, particularly within industrial robotics, automotive technologies, and healthcare. Concurrently, solid material modeling and computer visualization in agriculture have emerged as useful tools to solve complex problems. In the domain of modern seed breeding programs, the efficient and accurate analysis of seeds is paramount for successful outcomes. Phenotypes, encompassing morphological parameters of agricultural entities like seeds, roots, and leaves, are crucial to predicting their behavior under varying environmental conditions. Seed volume is particularly significant among these phenotypes, but even this fundamental metric, seed volume, proves challenging to calculate efficiently using conventional equipment. While volume estimation is a thoroughly studied problem for regular objects, the same principles do not seamlessly extend to irregularly shaped objects like seeds. The irregular shapes of seeds further complicate the accurate estimation of seed volume while ensuring its preservation.

This dissertation delves mainly into applying computer vision and scientific computing within the agricultural domain, explicitly focusing on low-throughput phenotyping (LTP). The primary emphasis lies in simulating, visualizing, and accurately calculating the volume of individual seeds. In this context, the dissertation proposes a comprehensive system comprising hardware and software components to address this challenge.

This dissertation presents several noteworthy contributions. The primary contribution involves proposing an innovative modeling framework designed to capture seed images effectively. Compared to previous frameworks, this new framework supports various backgrounds and enhances the ability to efficiently capture accurate images by rotating the seed at a constant rate and parsing the video to extract images at fixed angles.

The second contribution is extending traditional digital image processing (DIP) methods. A comprehensive analysis and comparison of diverse standard contour detection approaches is conducted using various background color contexts. The traditional space carving algorithm is extended by using a novel, multi-threaded approach to reduce the time required to carve seeds significantly.

Furthermore, this research extends the application of convolutional neural network (CNN)-based methodologies to detect edges in DIP methods. Following a thorough analysis of two prominent CNN methods, a novel, straightforward CNN model is proposed to facilitate contour detection. A detailed analysis of both DIP and CNN-based approaches measures the accuracy and efficiency of each approach and extension.

The dissertation contributes to the field by developing a user-friendly interface tailored for efficient seed analysis by agronomists.

Another direction for computer vision algorithms is at the field level in estimating fractional vegetation cover using Hough lines and linear iterative clustering.

Finally, model checking is used to verify the correctness of computer vision algorithms and other distributed algorithms, such as those to ensure mutual inclusion and mutual exclusion.

Automated phenotyping of seeds using traditional and convolutional neural network-based methods

by

Kai Zhao

M.S., Kansas State University, 2021

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Computer Science
Carl R. Ice College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2024

Approved by:

Major Professor
Mitchell L. Neilsen

Copyright

Abstract

The last decade has witnessed a rapid expansion of computer vision and scientific computing, particularly within industrial robotics, automotive technologies, and healthcare. Concurrently, solid material modeling and computer visualization in agriculture have emerged as useful tools to solve complex problems. In the domain of modern seed breeding programs, the efficient and accurate analysis of seeds is paramount for successful outcomes. Phenotypes, encompassing morphological parameters of agricultural entities like seeds, roots, and leaves, are crucial to predicting their behavior under varying environmental conditions. Seed volume is particularly significant among these phenotypes, but even this fundamental metric, seed volume, proves challenging to calculate efficiently using conventional equipment. While volume estimation is a thoroughly studied problem for regular objects, the same principles do not seamlessly extend to irregularly shaped objects like seeds. The irregular shapes of seeds further complicate the accurate estimation of seed volume while ensuring its preservation.

This dissertation delves mainly into applying computer vision and scientific computing within the agricultural domain, explicitly focusing on low-throughput phenotyping (LTP). The primary emphasis lies in simulating, visualizing, and accurately calculating the volume of individual seeds. In this context, the dissertation proposes a comprehensive system comprising hardware and software components to address this challenge.

This dissertation presents several noteworthy contributions. The primary contribution involves proposing an innovative modeling framework designed to capture seed images effectively. Compared to previous frameworks, this new framework supports various backgrounds and enhances the ability to efficiently capture accurate images by rotating the seed at a constant rate and parsing the video to extract images at fixed angles.

The second contribution is extending traditional digital image processing (DIP) methods. A comprehensive analysis and comparison of diverse standard contour detection approaches is conducted using various background color contexts. The traditional space carving algorithm is extended by using a novel, multi-threaded approach to reduce the time required to carve seeds significantly.

Furthermore, this research extends the application of convolutional neural network (CNN)-based methodologies to detect edges in DIP methods. Following a thorough analysis of two prominent CNN methods, we trained a new model with our seed dataset to facilitate contour detection. A detailed analysis of both traditional and CNN-based approaches measures the accuracy and efficiency of each approach and extension.

The dissertation contributes to the field by developing a user-friendly interface tailored for efficient seed analysis by agronomists.

Another direction for computer vision algorithms is at the field level in estimating fractional vegetation cover using Hough lines and linear iterative clustering.

Finally, model checking is used to verify the correctness of computer vision algorithms and other distributed algorithms, such as those to ensure mutual inclusion and mutual exclusion.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express gratitude of the highest order to my advisor Dr. Mitchell Neilsen for his unwavering support on every front throughout my doctoral program. His passion for research and learning is the impetus that has propelled my dissertation. Without his patient encouragement, understanding, and professional guidance, this dissertation would never have been realized.

I would also like to thank Dr. Torben Amtoft, Dr. Doina Caragea, and Dr. Paul Armstrong for serving on my doctoral committee and providing me with valuable input and motivation through my doctoral journey.

My sincere appreciation to the fellow students, faculty, and staff in the Department of Computer Science at Kansas State University. I would like to express my gratitude to my research collaborators Mr. Venkata Margapuri, Mr. Chendi Cao, Mr. Anthony Atkinson, Mr. Querriel Arvy Mendoza, Ms. Shaziya Mohammed, Mr. Friday James, Dr. Daniel Brabec, and Ms. Sophia Grothe.

I would like to express my gratitude to my parents, my wife, and my daughter for their unconditional love, support, patience, and trust during my doctoral endeavor.

Chapter 1 - Introduction

Applications related to seed and plant breeding necessitate the capability to accurately estimate morphometric characteristics, often referred to as phenotypes, of various agricultural entities and products such as seeds, leaves, and fruits. This estimation is crucial for establishing a correlation between morphometry and the behavior or characteristics of the plants. Plant phenotyping involves the comprehensive assessment of complex plant traits, encompassing aspects such as growth, development, tolerance, resistance, architecture, physiology, ecology, yield, and fundamental quantitative parameters vital for evaluating complex traits [1]. It entails assessing individual quantitative parameters and understanding how they contribute to the overall plant phenotype. Several reliable techniques have been developed to estimate morphological traits critical for plant phenotyping. These encompass estimations related to plant biomass [2][3], root morphology [4][5], leaf morphology [6][7], and fruit traits [8][9]. These measurements and assessments are pivotal in enhancing our understanding of plant characteristics and behavior. In addition to the traits mentioned above, seed volume is a crucial morphological trait that dramatically aids phenotyping efforts. Accurately estimating seed volume contributes significantly to our understanding of plant genetics, breeding, and related research. Estimating seed volume poses a significant challenge due to the irregular shapes of seeds. Seeds, the fundamental agricultural entities from which complex root and shoot systems develop, play a critical role in seed-functional ecology and seed-trait correlates. Seed volume is a key parameter overwhelmingly represented by mean seed mass [10][11] and is highly relevant in studies related to seed ecology and functional traits. In integrating the ecological and functional correlates of seed size distributions, especially in soil seed bank studies, sorting seeds by size is common. However, conventional methods involving sieving present limitations as they sort seeds based on linear dimensions rather than volume. Moreover, these methods are laborious,

time-consuming, and can potentially damage the seeds [12]. Alternative techniques for seed volume estimation include water displacement, volume slicing, and silhouette-based volume sculpting. Water displacement involves measuring the volume of an object by the amount of water it displaces when submerged. However, this technique can alter the structural integrity of seeds as they absorb water and increase in size. Volume slicing divides the object in the image into cross-sections and calculates the volume of each section, aggregating them to determine the total volume. Silhouette-based volume estimation, on the other hand, entails constructing a 3D model of the object using multi-view imagery. While no volume estimation technique can be deemed 100% accurate, image-based approaches like volume slicing and silhouette-based estimation preserve the seeds' structural integrity and chemical composition. Additionally, image-based analysis offers reproducible results across different experiments, reducing subjective bias. Estimating volume, a three-dimensional quantity, from two-dimensional entities (images) necessitates a meticulously curated setup for achieving 3D reconstruction of seeds. The presented article further explores the silhouette-based volume estimation technique and introduces a low-cost, end-to-end silhouette-based volume sculpting framework. This framework reconstructs the 3D model of a seed using multi-view imagery, capturing the seed from various perspectives. It's important to note that the proposed system operates in a low-throughput manner, processing one seed at a time.

1.1 Related Work

Koc [13] conducted a study that applied ellipsoid slicing and image processing techniques to estimate the volume of a watermelon. The ellipsoid approximation method relied on parameters such as the watermelon's length and major and minor diameters. In the context of image processing, the grayscale contour representing the boundary of the watermelon was

extracted. This boundary image was then employed to approximate the volume using the disk method [14]. The results obtained from both techniques were compared against the volume estimated through water displacement. The comparison revealed a higher degree of similarity between water displacement and image processing techniques as compared to the similarity between water displacement and ellipsoid slicing.

Roussel, Geiger, Fischbach, Jahnke, and Scharr [15] presented a methodology for 3D reconstruction of plant seed surfaces, including those with diameters as small as 200 μm. They employed the shape-from-silhouette approach in their study. The automated system had a relatively low throughput as it processed one seed at a time. However, a noteworthy accomplishment of this work was its effective management of variations in camera pose.

Cao and Neilsen [16] expanded on Roussel's research by introducing an affordable 3D measurement system for individual seed volumes. Their methodology involved using 3D-printed components to create a framework for mounting the camera. Seeds were placed on a rotating platform against a black background. Cameras captured sequential images of the rotating seed, and volume estimation was performed using silhouette-based sculpting. To validate the system, comparisons were made with a ceramic ball of known geometry. The system's estimated volume showed a difference of less than 3% compared to the actual volume. Zhao and Margapuri [17] further enhanced the hardware and software and conducted experiments with soybean and wheat seeds, comparing slicing methods. This study reinforced the effectiveness of the 3D reconstruction method and achieved automation.

Pound, French, Murchie, and Pridmore [18] presented a method for the automated generation of three-dimensional models of plant shoots using multiple color images, all captured with a single cost-effective camera. The reconstruction algorithm relied on an initial point cloud estimate as a foundation for extending plant surfaces in three dimensions. The resulting reconstructed plants were represented as a sequence of small planar sections, collectively portraying the intricate architecture of leaf surfaces. The delineation of each leaf patch was refined by applying the level-set method optimizing the model by incorporating image data, curvature constraints, and neighboring surface positions. This proposed methodology underwent testing on diverse datasets, including wheat, rice, and a virtual dataset, enabling the assessment of reconstruction accuracy.

Yang and Cho [19] conducted a study focused on 3D crop reconstruction and automated phenotyping index analysis using machine learning techniques. In their experimental setup, they designed and deployed a system for reconstructing 3D images of red pepper plants and performing automated analysis. The system used a Kinect v2 with a depth sensor and a high-resolution RGB camera to capture highly accurate reconstructed 3D images. These reconstructed 3D images were compared with conventionally reconstructed images, and data extracted from them were analyzed alongside directly measured features such as leaf number, width, and plant height. The results indicated that the proposed method exhibited an error of approximately 5 mm or less during the reconstruction and analysis of 3D images, affirming its suitability for phenotypic analysis.

Potmesil [20] presented a methodology for creating octree models that represent 3D solid objects using their silhouettes obtained from a series of images. The silhouettes of the objects

were projected onto an image plane, and 3D conic volumes were created based on the center of these projections. The 3D object model was then generated by intersecting these conic volumes derived from the sequence of images. To efficiently process the 3D volume data, hierarchical octree structures were employed. Each object's volume was labeled using a connectivity-labeling algorithm, and surface-normal vectors were integrated into the volume elements of their surfaces.

Sankaran, Wang, and Vandemark [21] introduced a rapid phenotyping method using image analysis to estimate the size of chickpeas. The experiment involved samples collected from 72 plots located in two different areas. The study included the use of images that depicted chickpeas both in clusters and individually. The Watershed algorithm was applied to segment the clustered chickpeas, and a custom macro developed in ImageJ facilitated seed counting and size estimation. The size of chickpeas was estimated using the US Penny as a reference object. The results revealed a strong correlation between the seed size estimated through the image processing technique and the actual ground-truth data.

Golbach, Kootstra, Damjanovic, Otten, and Zedde [22] introduced a high-throughput system designed for the 3D reconstruction of plants. The 3D reconstruction process was based on a silhouette-based technique. Multiple cameras were strategically used to capture plant images from various perspectives, aiming to improve reconstruction quality. However, an increase in the number of cameras also heightened the computational complexity of the system. As a result, a balance was sought, and ten cameras were chosen to strike an optimal trade-off. The results showed a significant correlation between the algorithm-generated outcomes and ground-truth measurements obtained through manual assessment.

1.2 Contribution and Dissertation Overview

To determine the volume of irregularly shaped seeds essential precisely and efficiently for contemporary breeding systems, we employed a 3D reconstruction methodology rooted in computer vision. Our novel modeling framework significantly enhances the efficacy of image acquisition while mitigating errors stemming from manual interventions. Concurrently, we conducted a comprehensive comparison of diverse image processing techniques, encompassing traditional HSV-based analysis and color thresholding, alongside machine learning approaches utilizing CNN. Through meticulous analysis and synthesis, we delineated the merits and demerits of each method. Furthermore, we juxtaposed and scrutinized 3D reconstruction methodology against alternative computer vision technologies, notably the slicing method, offering invaluable benchmarks for future research endeavors.

This dissertation presents several noteworthy contributions:

In Chapter 2, we proposed an innovative modeling framework designed to capture seed images effectively [17]. Compared to previous frameworks, this new framework supports various backgrounds and enhances the ability to efficiently capture accurate images by rotating the seed at a constant rate and parsing the video to extract images at fixed angles.

In Chapter 3, we extended traditional digital image processing (DIP) methods. A comprehensive analysis and comparison of diverse standard contour detection approaches is conducted using various background color contexts. Furthermore, we also extended the application of convolutional neural network (CNN)-based methodologies to detect edges in DIP

methods. Following a thorough analysis of two prominent CNN methods, we trained a new model with our seed dataset to facilitate contour detection. A detailed analysis of both traditional and CNN-based approaches measures the accuracy and efficiency of each approach and extension.

In Chapter 4, we extended the core algorithm of the original 3D reconstruction technology, space carving algorithm, using a novel, multi-threaded approach to reduce the time required to carve seeds significantly [23]. And through comparative analysis with other computer vision technologies (slicing methods), we demonstrated the reliability and accuracy of 3D reconstruction technology.

In Chapter 5, we designed a user-friendly interface specifically crafted to facilitate seed analysis by agronomists. Concurrently, we generated a more vivid and tangible 3D model utilizing the point cloud method.

In Chapter 6, We proposed a technique to precisely estimate the amount of fractional vegetation cover (FVC) in an area from images by drawing inferences from and extending upon the Daubenmire method, a semi-quantitative ocular based FVC estimation technique [24]. The proposed technique estimates the area occupied by FVC in metric units in addition to the percentage, while current techniques only estimate the percentage of FVC within images. The precise estimation of the FVC helps plant scientists make discoveries of plant varieties that make good companions and offer well-informed suggestions to the agricultural community.

In Chapter 7, we established a checking model to verify the correctness of computer vision algorithms and other distributed algorithms, such as those to ensure mutual inclusion and mutual exclusion [25].

Chapter 2 - Novel Modeling Framework

2.1 Hardware Framework

Achieving 3D reconstruction of seeds necessitates capturing multiple images from various angles at consistent angular intervals. To ensure precise image capture, a low-cost hardware setup is recommended. The distinct components of this hardware setup are detailed below and depicted in Figure 2.1 and Figure 2.2.

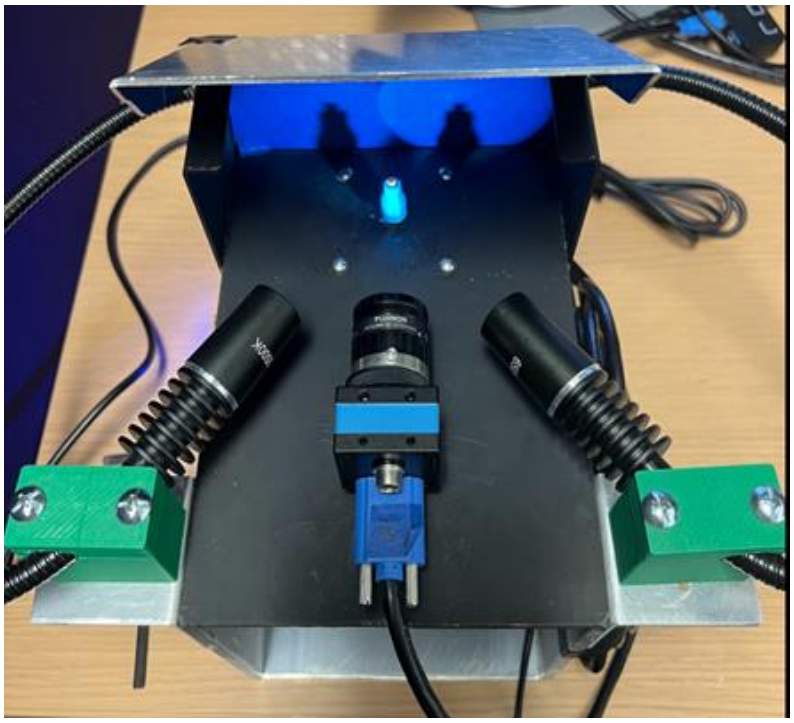

Figure 2.1. Top View of Hardware Setup for Single Seed Reconstruction

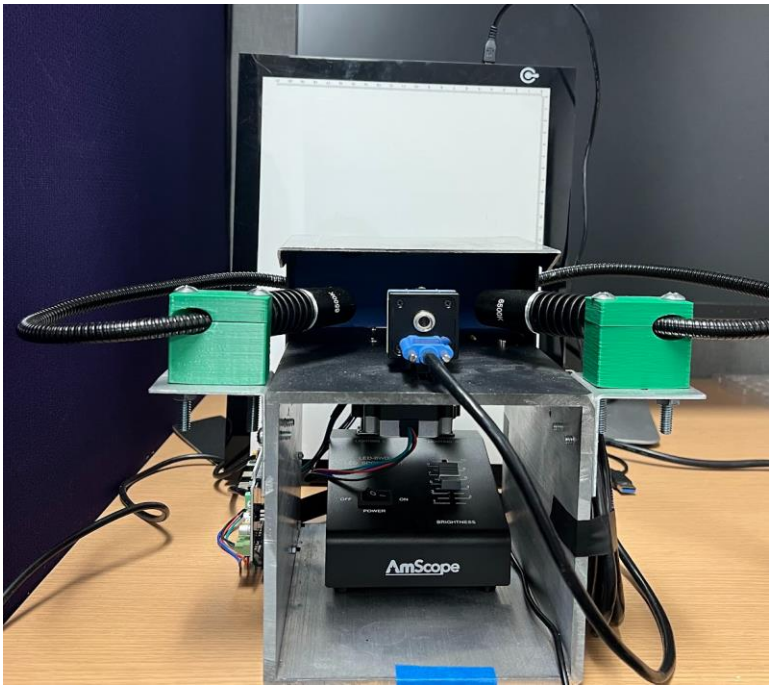

Figure 2.2. Front view of Hardware Setup for Single Seed Reconstruction

a) Seed Station (blue holder): The seed station accommodates the seed while being rotated by a NEMA 23 stepper motor. The size of the seed station is precisely calibrated to accommodate the seed comfortably. Power for the stepper motor is supplied through a 12V power source, and its operation is meticulously controlled by an Arduino microcontroller utilizing a Synthetic gShield stepper driver.

b) Stepper Motor: The control of the stepper motor is facilitated through an Arduino UNO R3 board coupled with a gShield v5 board. G-Code strings, generated from a Python program, are employed to direct and manage the stepper motor. In this experiment, a default setting of a 10-degree rotation is utilized through micro-stepping. In simpler terms, we require a set of 36 two-dimensional pictures to encompass the seed comprehensively. The empirical [16] evidence indicates that this set of 36 pictures is sufficiently effective for our purpose.

In previous iterations, manual control of the stepper motors was facilitated through the Universal G-Code Sender, a desktop application that establishes a connection with the stepper motors. However, in the current iteration, the operation of the stepper motors is automated. Python is now used to send G-code corresponding to a specific rotation angle to the stepper motor. NEMA 23 stepper motors are designed to have 200 steps per revolution, resulting in a rotation of 1.8° per step. It is important to emphasize that stepper motors allow rotation only in multiples of 1.8°, meaning you can achieve 9° or 10.8° but not exactly 10°. Figure 2.3 visually illustrates the disparity in motor setup between the two iterations.

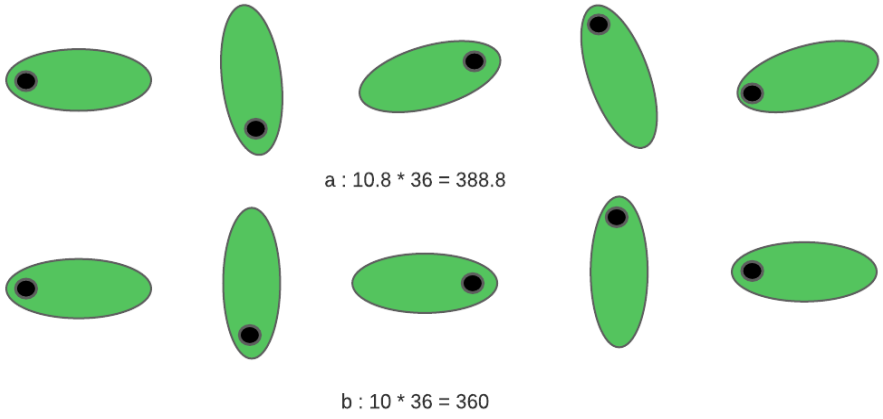

Figure 2.3. Rotation comparison with different iterations

c) Image Capture: Images of the seed are acquired using a color camera, specifically the DFK 37BUX287. The camera is positioned horizontally to the seed. Leveraging the API provided by ImagingSource IC software, we extracted the fundamental parameter-setting function of the camera and integrated it into our software. Following the motor's speed configuration, we captured the necessary picture frame by recording the video.

In the previous iteration, the camera was affixed to the mount at a 10° orientation, causing an incomplete image with the bottom surface of the seed not being captured.

Subsequently, the camera orientation was recalibrated to ensure it was perpendicular to the seed kernel, thus enabling a comprehensive capture of the bottom surface. Figure 2.4 visually illustrates the disparity in camera setup between the two iterations.

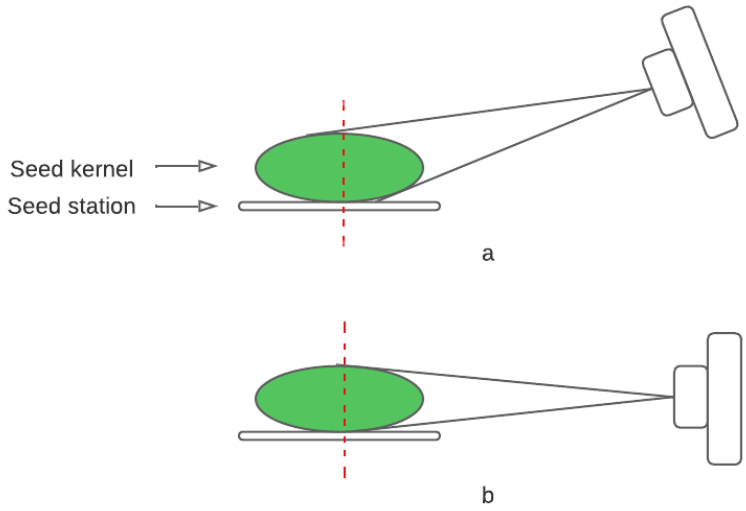

Figure 2.4. Camera Orientation in (a) Previous Iteration (b) Current Iteration

d) Light Source: Adjacent to the seed station, two power-charged LED light sources are positioned to illuminate the seed kernel, ensuring it appears bright during image capture.

e) The aluminum frame: The aluminum frame (Created by Dan Brabec, USDA ARS Center for Grain and Animal Health Research in Manhattan, KS) serves as the structural foundation, connecting all hardware components to maintain stability and precision during image capture. A spray-painted black surface and lid are utilized to minimize the influence of ambient light. A camera track is strategically positioned at the center of the surface, offering versatility in accommodating various camera types and enabling different distance settings. Brackets secure

the LED lights on either side of the surface, ensuring consistent color portrayal in the pictures by setting the optimal light angle. Below the surface, there is a designated space to house the motor, LED controller, Arduino UNO R3 board, and gShield v5 board.

2.2 Software Framework

The software implementation for controlling the hardware framework and conducting 3D reconstruction is achieved using both Python and C. C is primarily utilized for volume calculation, whereas Python takes charge of various tasks such as image capture, conversion of original images to binary mask images, and 3D seed reconstruction. The application employs different software packages, each specified with its respective version in Table 2.1.

Software	Package	Version	Note
Python (3.9)	Numpy	1.21.6	Read files and process images.
	Open3d	0.16.0	Produce a 3D model with a smooth surface.
	OpenCV	4.5.1.48	Detect contour and process images
	Pyinstaller	4.10	Install project to.exe file
	Torch	2.0.1	Train the CNN model.
	Xlrd	2.0.1	Read a .xls file
	Xlwt	1.3.0	Store the result in a .xls file
	Pillow	9.3.0	Read and save images.
C	Carve.c		Calculate the volume.
Cam_Driver		5.1.0.1719	Install the camera.

Table 2.1. The software used in the project

Achieving high-quality and precise images involves a three-step process:

Step 1: Configure the camera and motor properties.

Stabilize the camera position on the aluminum frame, position the seeds accordingly, and commence by configuring the fundamental camera properties upon securing the lid. Default the frame rate value to 30 frames/second, and set the brightness, gain, and exposure to automatic. Subsequently, fine-tune the color properties, precisely adjusting the white balance to enhance the visibility of the target seeds in the image. Aim for a clear depiction with minimal or no reflection, considering that varying colors or LED backgrounds necessitate distinct auto-reference values for optimal results. Numerous experiments and comparisons were conducted to achieve a series of clear and precisely rotating images. It was established through careful analysis that a motor turning speed of 0.17768 was optimal, and the camera was configured to intercept one frame every four frames. Figure 2.5 shows the user interface of the camera property.

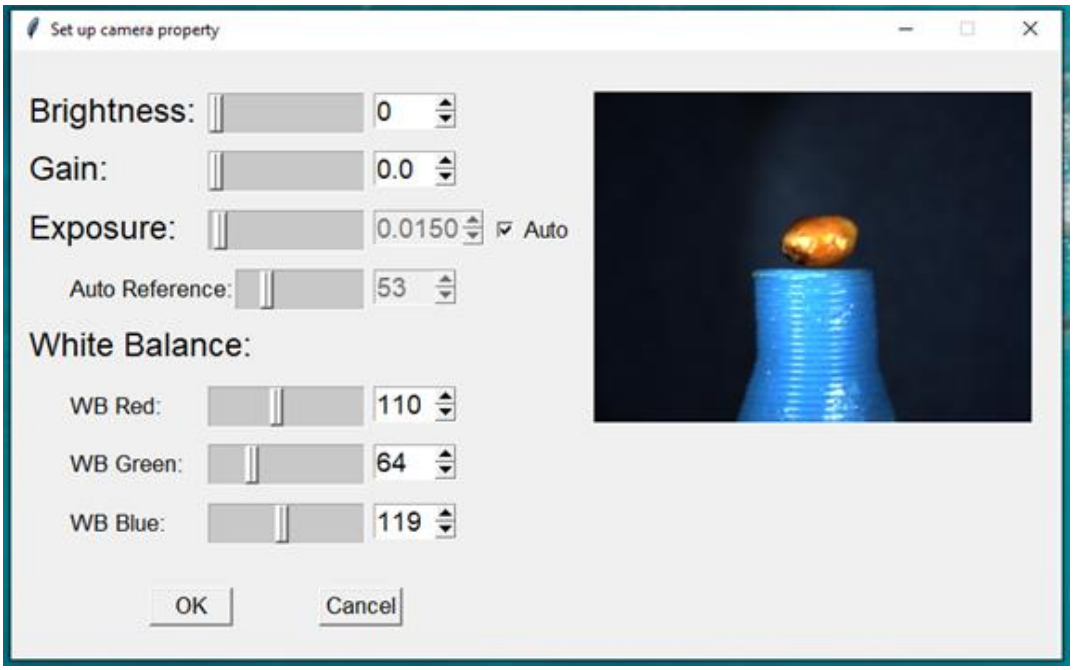

Figure 2.5. UI of camera property setup

Step 2: Capture 38 frames from the video.

The motor's speed experiences instability during the initial and final moments of operation. Capturing a picture during these instances would result in an inaccurate rotation angle of the seed (not precisely 10 degrees). To mitigate this, we initiate the motor to rotate one and a half turns, equivalent to 540 degrees. From this rotation, we intercept the middle 380 degrees, enabling us to capture 38 pictures, as illustrated in Figure 2.6. The theoretical fastest time is 54 / (30 / 4) = 7.2 seconds.

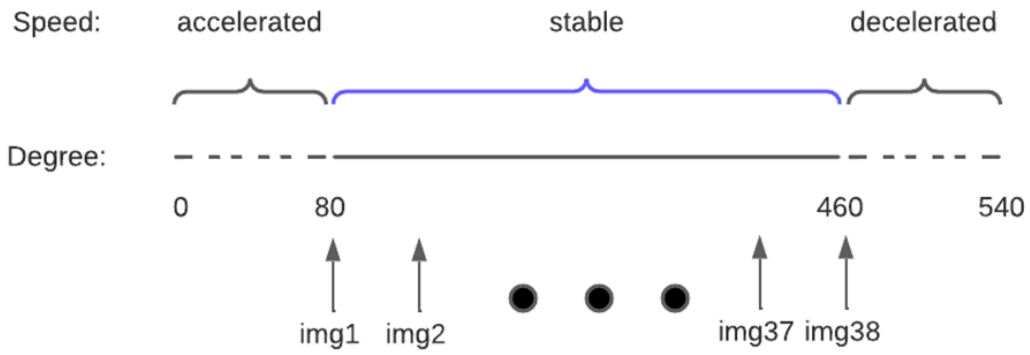

Figure 2.6. The process of capturing 38 images

Step 3: Validate the captured frames to ensure accuracy and reliability.

Once we have acquired a set of 38 photos, it is crucial to verify the accuracy of the rotation. This verification involves a comparison between the first and 37th pictures. Employing identical parameters and methods, contour detection is performed on both images, enabling the extraction of coordinates for the upper left corner of the seed rectangle area and the length and width of the rectangle area. These values are then compared between the two sets. To prevent coincidences, we repeat the comparison with the second picture and the 38th picture. This process continues until both sets of comparisons yield the same results. Subsequently, the first 36 pictures are saved and prepared for the subsequent steps, as illustrated in Figure 2.7.

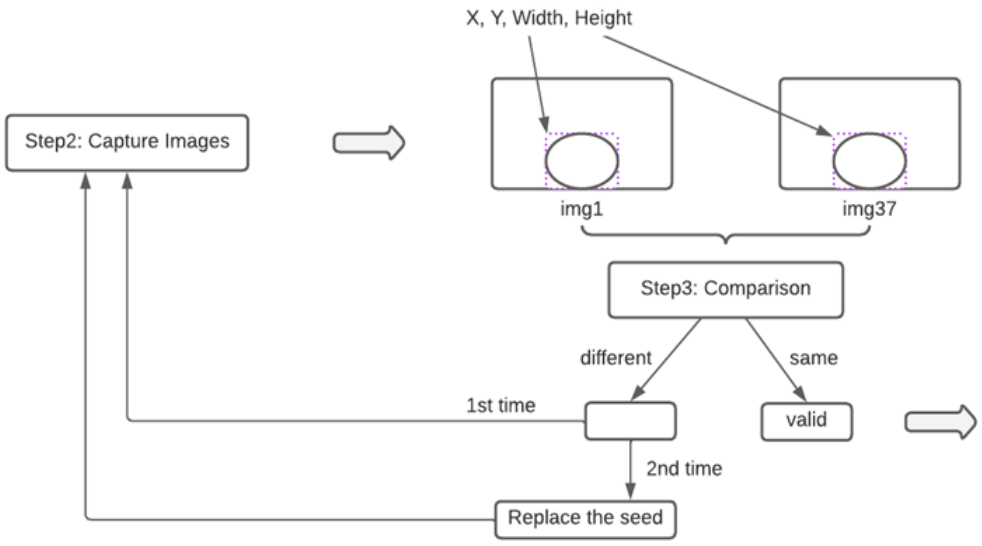

Figure 2.7. Validating images

In summary, our framework achieves automated phenotypic analysis, enhancing the precision of seed volume calculations and extracting additional phenotypic parameters for each seed. A comparative representation of these advancements against the previous version is presented in Figure 2.8:

		Framework	Previous
Input	Stable	Yes	No
	Angle	No	Yes
	Ways	36 / time (Automatically), exactly rotated	1 / time * 36 (Manually), over rotated
	Time	< 8	≈54
Output	Volume	Yes	Yes
	Other properties	Lenth, Width, Height	N/A

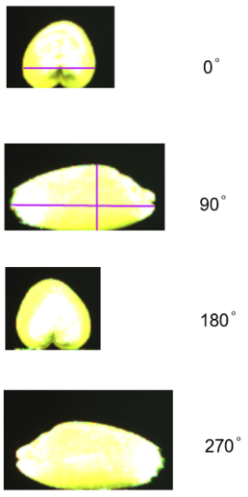

Figure 2.8. Comparison between current version and previous version

Chapter 3 - Digital Image Processing (DIP)

Digital image processing (DIP) involves leveraging a digital computer to manipulate digital images using specific algorithms [26][27]. Situated within the realm of digital signal processing, digital image processing offers numerous advantages compared to analog image processing. It allows for a broader spectrum of algorithms to be applied to input data and helps mitigate issues like noise accumulation and distortion during processing. Given that images are defined in two or more dimensions, digital image processing can be conceptualized as operating within multidimensional systems.

3.1 Pre-process the Digital Image

Before commencing formal digital image processing, a preprocessing step involving seed kernel detection is essential. The images obtained in the previous step tend to be excessively large and often contain a significant background area. To address this, we cropped the acquired images, endeavoring to position the seed centrally within the image as much as possible.

In the previous version of the application, the user was responsible for identifying the pixel coordinates for the most probable location of the seed kernel. This process involved defining three specific rectangular regions: the search rectangle, template rectangle, and crop rectangle.

Search Rectangle: This bounding rectangular region encompasses the template rectangle. It represents an area within all the images where the seed is undoubtedly located. For instance, in a dataset of 36 images, regardless of the varying seed kernel positions, the search rectangle consistently contains the seed in any image when searched.

Template Rectangle: This larger rectangular region includes the crop rectangle. The offset between the crop and template rectangles remains fixed, allowing for accurate cropping of the seed kernel. The crop rectangle adjusts its position with any changes to the template rectangle's position.

Crop Rectangle: This rectangle is positioned within the template rectangle, closely encircling the seed kernel. Its position is linked to the template rectangle, ensuring a consistent offset between the two facilitating precise cropping of the seed kernel.

These rectangular regions are visualized in Figure 3.1 for clarity.

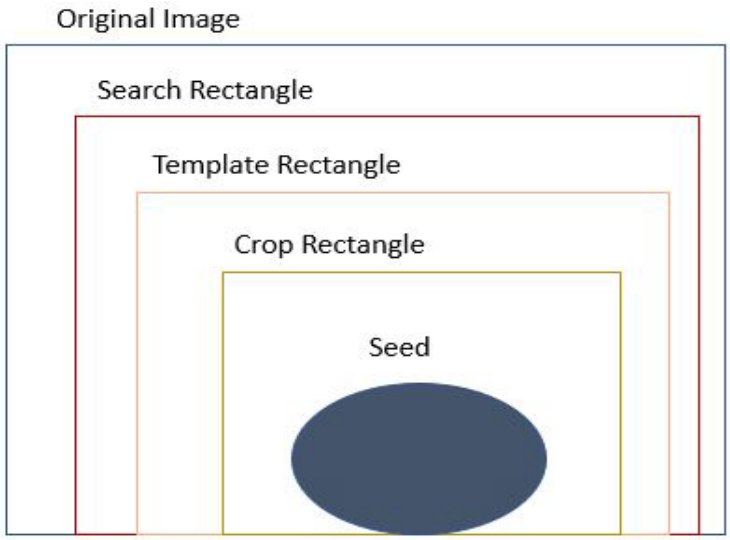

Figure 3.1. Rectangular Regions to Crop Seed Kernel within Image

The three rectangular regions are experiment-specific; whenever the seed's position on the seed station changes, these regions must be redefined. However, determining the criteria for the rectangular areas is laborious, prone to errors, and unrealistic in terms of user expectations. In

the current iteration, we have made significant enhancements by automating the seed kernel's location determination through image processing. The steps to achieve this are outlined below:

1. In each acquired image, identify the contour of the seed kernel and crop the rectangular region surrounding the contour, as demonstrated in Figure 3.2.

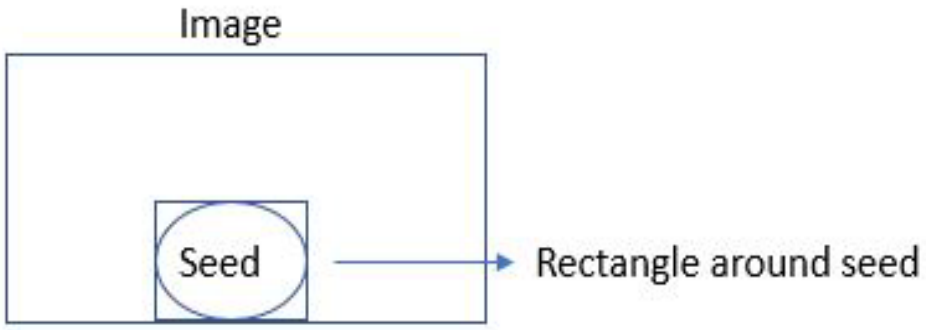

Figure 3.2. Detected Rectangular around the Seed Kernel Contour

Note: It's essential to consider that the contour size of the seed kernel varies slightly within each image due to the seed kernel's rotation on the seed station. As the seed kernel circularly rotates on the station, assuming the circle's diameter is 'd' and the distance between the camera and the center of the seed station is 'x', the seed kernel is positioned at distances 'x - d' and 'x + d' at two distinct points during rotation. The distance 'x - d' represents the point closest to the camera where the seed kernel appears larger, and 'x + d' represents the farthest point where the seed kernel seems smaller. To standardize all images for a consistent analysis, it is crucial to normalize the cropped image sizes, ensuring uniformity.

2. Adjust the size of all images in the dataset to match the mean size of the images containing the largest and smallest seed kernel contours.

3. Detect the contour of the seed kernel in each of the resized images (similar to step 1).

The requirement to position the seed in the middle is directly linked to the subsequent cutting algorithm. We conducted an experiment using a standard metal sphere with a diameter of 3.95mm to compare results when the sphere was placed in the middle position versus when it was not. The theoretical volume of this standard metal sphere is calculated to be 32.27 cubic millimeters:

$V=4\pi r^3 / 3 = 4 \cdot \pi \cdot 1.983 \approx 32.26933$

This comparison is essential to assess the accuracy and effectiveness of our subsequent cutting algorithm, shown in Figure 3.3.

Figure 3.3. The sphere is not in the middle (above); the sphere is in the middle (bottom)

3.2 Digital Image Processing

The primary objective of DIP in this dissertation is to transform the original image into a binary mask image. This conversion serves to streamline the implementation of our subsequent sculpting algorithm. The challenge lies in accurately detecting the silhouette or contour of the seed, which constitutes a focal and pivotal aspect discussed in this dissertation.

Image edge detection has attracted widespread attention from researchers since its inception. Figure 3.4 outlines the evolution of edge detection algorithms [28]. Contour detection is a type of edge detection and a basic task in image processing and computer vision. It involves the identification of object boundaries within images, a key process with a wide range of applications, including object recognition, shape analysis, and image segmentation. Much research has delved into image edge detection, especially in face recognition and autonomous

driving. However, in the agricultural domain, especially for seed phenotyping, it has not been extensively explored. This dissertation aims to fill this gap by providing a brief overview of image contour detection research focusing on seeds in the agricultural domain. Contour detection is significantly influenced by image quality, with color saturation due to ambient light and the background color playing a crucial role. This dissertation tests various traditional detection methods under different background colors and compares them with CNN detection methods with a consistent background color.

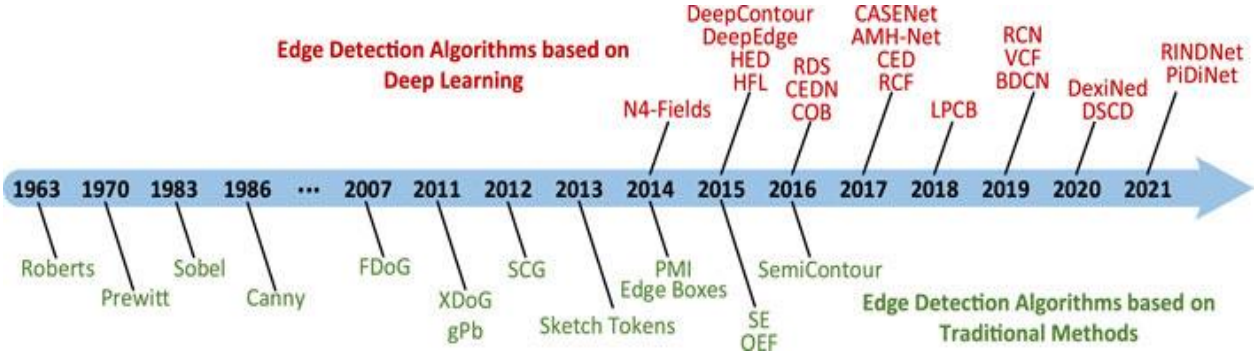

Figure 3.4. Development of edge detection algorithms [28]

3.2.1 Extension of the Traditional DIP

It is recognized that seeds vary significantly in terms of shapes and colors, even within the same species. This inherent diversity poses a challenge in image analysis. Achieving a clear seed outline necessitates separating the seed from its background environment. This is why previous iterations utilized appropriate LED lighting and a black background. A well-adjusted LED with suitable brightness effectively highlights the seed outline while preventing excessive darkness at the bottom where the seed connects to the holder. Striking the right balance in brightness is crucial, as excessive brightness can enlarge the seed's contour and affect contour detection due to surface reflections. On the other hand, very low brightness may make the

contour small. The black background contrasts with the seeds, minimizing the impact of weak black reflections on the detection process.

HSV [29] (Hue, Saturation, Value) was employed for contour detection in previous iterations. Still, it demanded precise control of LED lights due to the varying reflective effects based on different seed types and colors. The experiments primarily focused on wheat, corn, and soybean, which share similar yellow-green hues and surface smoothness. Hence, using fixed HSV values was feasible. However, the previous iteration's approach is no longer applicable in the current experiments with various milo types, ranging in size from 3mm to 6mm and exhibiting colors from white to black (with some white seeds having black spots). Figure 3.5 serves as an illustrative example. In the initial image, black spots are visible on the surface of Milo seeds. During contour detection, HSV is applied within the original black background. However, this approach causes challenges, as the black part of the seed surface blends with the black background, leading to imperfections in the obtained outline. Even considering alternative edge detection methods like Canny [30], the distinct characteristics of milo seeds necessitate a tailored approach.

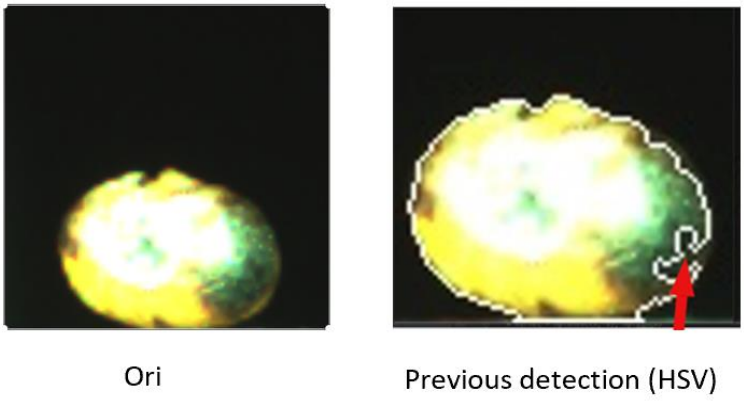

Figure 3.5. The original image with HSV contour result

A strategic enhancement was introduced to address the imperfections in the seed contour depicted. Before utilizing HSV for contour detection, the Canny algorithm was applied to the image. The Canny algorithm effectively distinguishes the black spot part from the black background, allowing for more precise contour detection. The result combines this enhanced image with the original one, enabling a more accurate HSV-based contour detection.

However, it's essential to acknowledge that the Canny algorithm has certain drawbacks, notably contour discontinuity, which can lead to suboptimal outcomes. For instance, as indicated by the red arrow in Figure 3.6, the contour of the target seed may be partially ideal due to these inherent limitations.

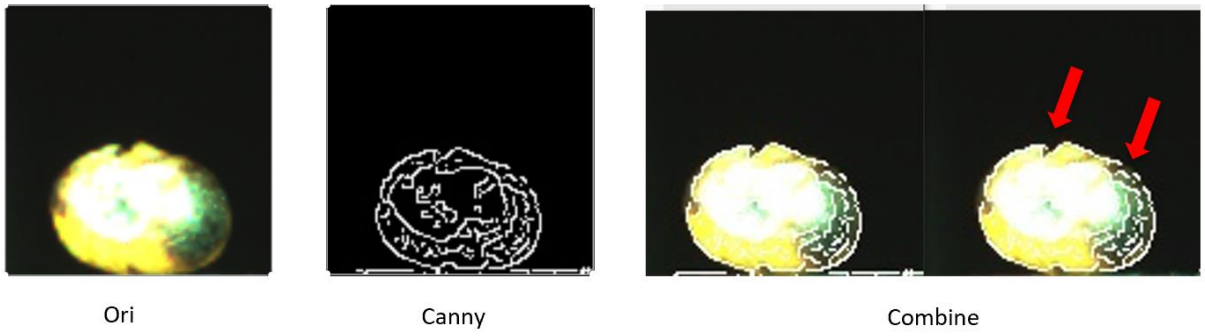

Figure 3.6. The original image with Canny contour result

The main issue arises from the black spots on the seeds closely resembling the color of the background plate, prompting the adoption of a blue background. Consequently, we transitioned from an HSV-based detection image to a color threshold approach. Unlike HSV's fixed light intensity value, the color threshold parameter defines a color range. This adjustment allows for a more extensive processing range, accommodating seeds of various colors, even when their reflection affects the image.

In contrast to the HSV-based method, where contour detection is based on seed color, the color threshold method considers the background color for contour detection. This proves advantageous as it maintains a consistent background color (blue in our case) regardless of the seed color. The color threshold method offers a more versatile and standardized approach, especially when dealing with seeds of diverse colors, as shown in Figure 3.7. Furthermore, we can extract the color values of the seeds themselves, thereby offering additional research data for seed researchers.

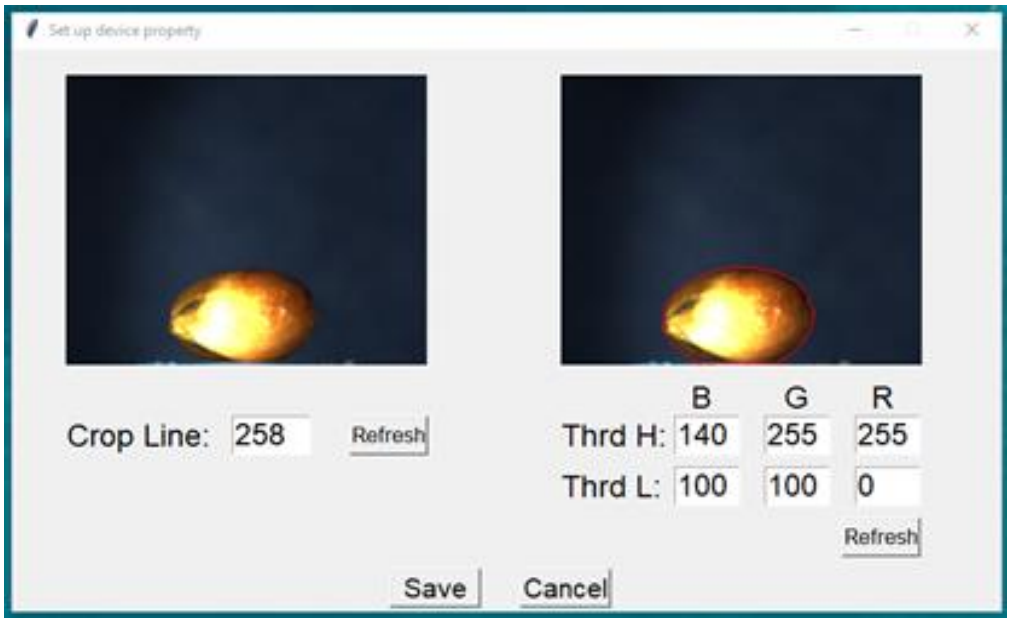

Figure 3.7. Detect contour using automated binary color threshold

Figure 3.8 presents a comparison, showcasing contour detection and the conversion into binary mask images of the same seed under varying color backgrounds. The mask image on the blue background yields more favorable results, although certain edge parts may still experience distortion due to the interplay of seed and background colors. Given our primary objective of

obtaining seed outlines rather than focusing on color, we propose a novel LED background board.

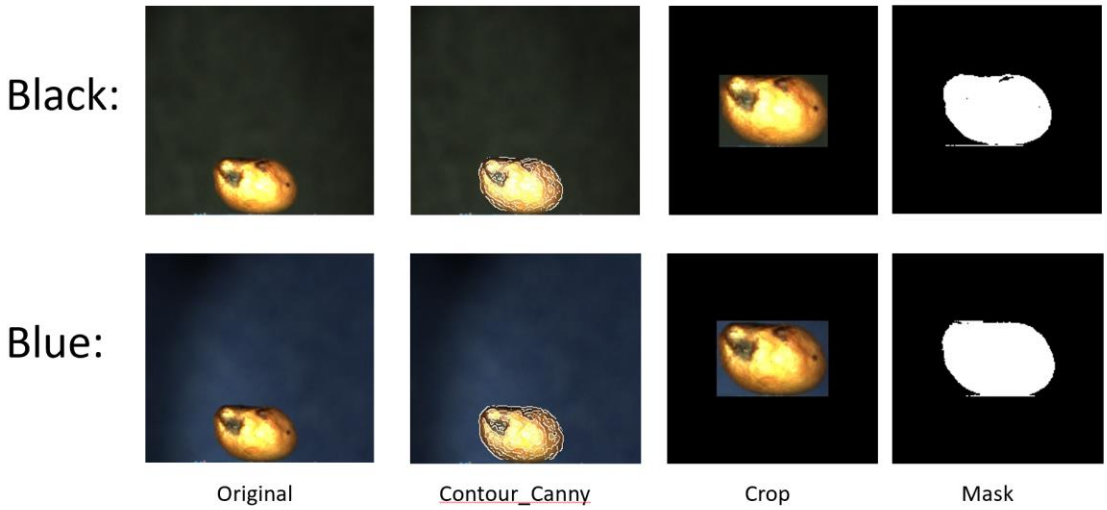

Figure 3.8. Comparison of black and blue backgrounds

This innovative approach involves deactivating the previous two LED lights and installing the LED background board. The light is directed from behind the seeds towards the front. Consequently, regardless of the seed color, the black region in the image corresponds to the seed area, while the rest represents the background color. This innovative setup entirely eradicates the impact of the seed's inherent color on contour detection, as shown in Figure 3.9. Compared to contour detection on a blue background, contour detection with the LED background proves more precise. The intensity of the LED background can be adjusted. The ME456 LED Tracer is a low-cost background with a color temperature of 10,000 to 12,000 K and an active area of 9 x 12 inches. As illustrated in Figure 3.10, contour detection on the blue background exhibits inaccuracies, particularly evident in the detection indicated by the red arrow.

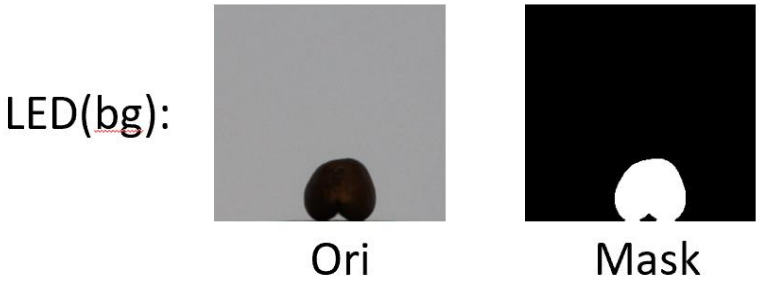

Figure 3.9. Original image under LED background and mask image

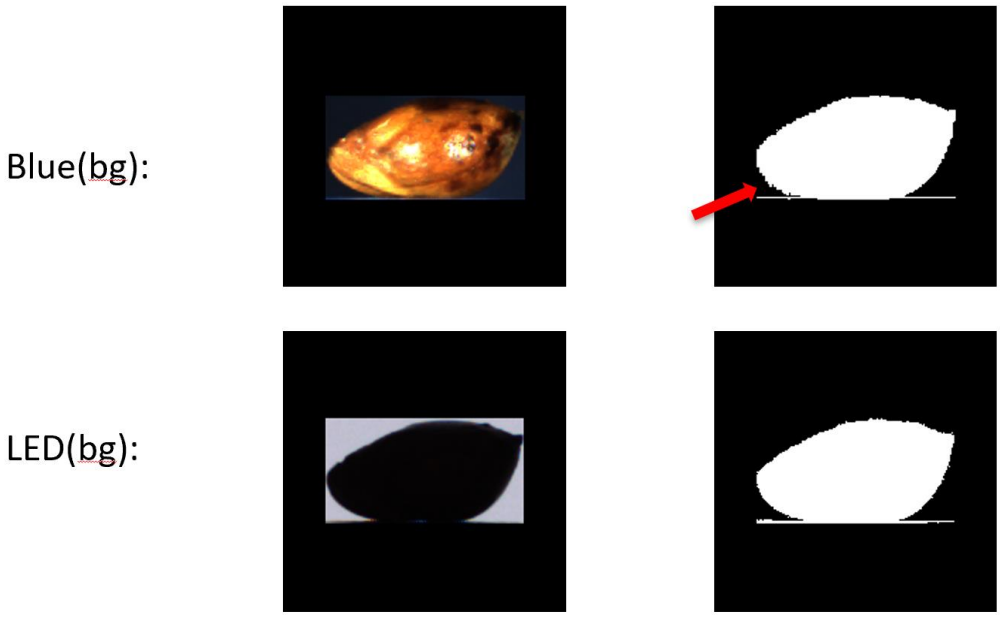

Figure 3.10. Comparison between blue background and LED background

To validate the testing accuracy under the LED background, we conducted multiple tests on seven reference objects with varying sizes and shapes, as illustrated in Figure 3.11. Each of these reference objects underwent testing ten times, and the results are averaged.

Figure 3.11. Steel reference objects

The outcome is presented in Table 3.1, which distinctly reveals that the difference between the test results obtained through the 3D method and the actual volume is less than 3%. Note that the test results of cylinder and cubes are smaller due to their rounded edges, so the actual error is less than 3% after recalculating the real size.

	Theoretical (mm)				Micrometer (mm)				3D (mm)				Comparison of Vol (3D VS Micro)			
Type	L/D	W	H	Vol	L/D	W	H	Vol	L/D	W	H	Vol	Diff	Error/Ave	Error/Min	Error/Max
Ball_sm	4			33.510	3.97			32.762	3.99	3.95	3.98	31.947	-0.815	-2.5%	-2.2%	-2.8%
Ball_md	4.75			56.115	4.75			56.115	4.77	4.69	4.73	54.521	-1.594	-2.8%	-2.6%	-3.3%
Ball_lg	5			65.450	4.99			65.058	5	4.95	4.97	63.085	-1.973	-3.0%	-2.9%	-3.3%
Cone	3		4.50	10.603	3.2		4.52	12.117				11.948	-0.169	-1.4%	-1.3%	-1.6%
Cylinder	6		4	113.09	5.96		4.04	112.71	5.98	5.94	4.21	108.83	-3.877	-3.4%	-2.9%	-4.0%
Cub_sm	3		3	27.000	2.99		2.97	26.552				25.58	-0.972	-3.7%	-2.7%	-4.7%
Cub_lg	5		5	125.00	4.97		5.24	129.43				125.35	-4.081	-3.2%	-2.7%	-4.1%

Table 3.1. References test with LED background

3.2.2 Extension of Convolutional Neural Network (CNN) DIP

In recent years, the integration and advancement of artificial intelligence and machine learning algorithms have led to a proliferation of image edge detection algorithms in digital image processing. However, due to factors like localization accuracy, edge detection precision, noise sensitivity, and variations in detection algorithm accuracy, image edge detection methods are continuously evolving to fulfill diverse requirements. Notably, several edge detection methods utilizing deep learning have emerged. This dissertation primarily focuses on two

methods: Holistically-nested Edge Detection (HED) [31] and Dense Extreme Inception Network for Edge Detection (DexiNed) [32], both of which are multi-scale feature fusion-based approaches.

3.2.2.1 Holistically nested Edge Detection (HED)

HED was created in 2015, using the state-of-the-art VGG-16[33] as the backbone network and transfer learning to initialize the network weights. The term "holistically" signifies that the algorithm endeavors to train an image-to-image network, while "nested" underscores the process of iteratively refining the edge prediction map for enhanced accuracy during the output generation. This approach aims to overcome the limitations of Canny edge detectors by leveraging end-to-end deep neural networks. The network receives an RGB image as input and produces an edge map as the output. Notably, the edge maps generated by HED demonstrate superior preservation of object boundaries within the image. Shown as Figure 3.12.

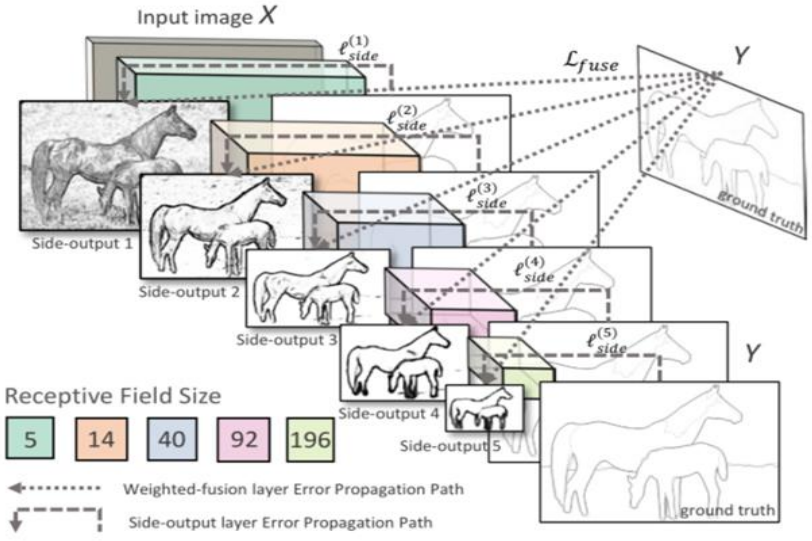

Figure 3.12. The prominent backbone architecture of HED [31]

HED has implemented several enhancements building upon VGG-16:

1)VGG Structure Modification:

a. Introduction of a side branch before each max-pooling layer.

b. Eliminating the pooling layer after the fifth block and removing all fully connected layers.

2) Deep Supervision: Computation of label images and backpropagation is performed with each side output image.

3) Feature Fusion: Stack the output prediction maps and apply convolution to enable the network to learn appropriate weights. Weighted fusion of these feature maps is carried out to generate a more refined edge prediction map.

3.2.2.2 Dense Extreme Inception Network for Edge Detection (DexiNed)

"DexiNed" represents a pioneering deep learning model designed to produce thin edge maps resembling the human eye's perception, as shown in Figure 3.13. This model is versatile and well-suited for various edge detection tasks, requiring no extensive training or fine-tuning. DexiNed is an advancement built upon the foundations of both HED and Xception[34]. Xception is another improvement to Inception v3 proposed by Google after Inception. It mainly uses depth-wise separable convolution to replace the original convolution operation in Inception v3.

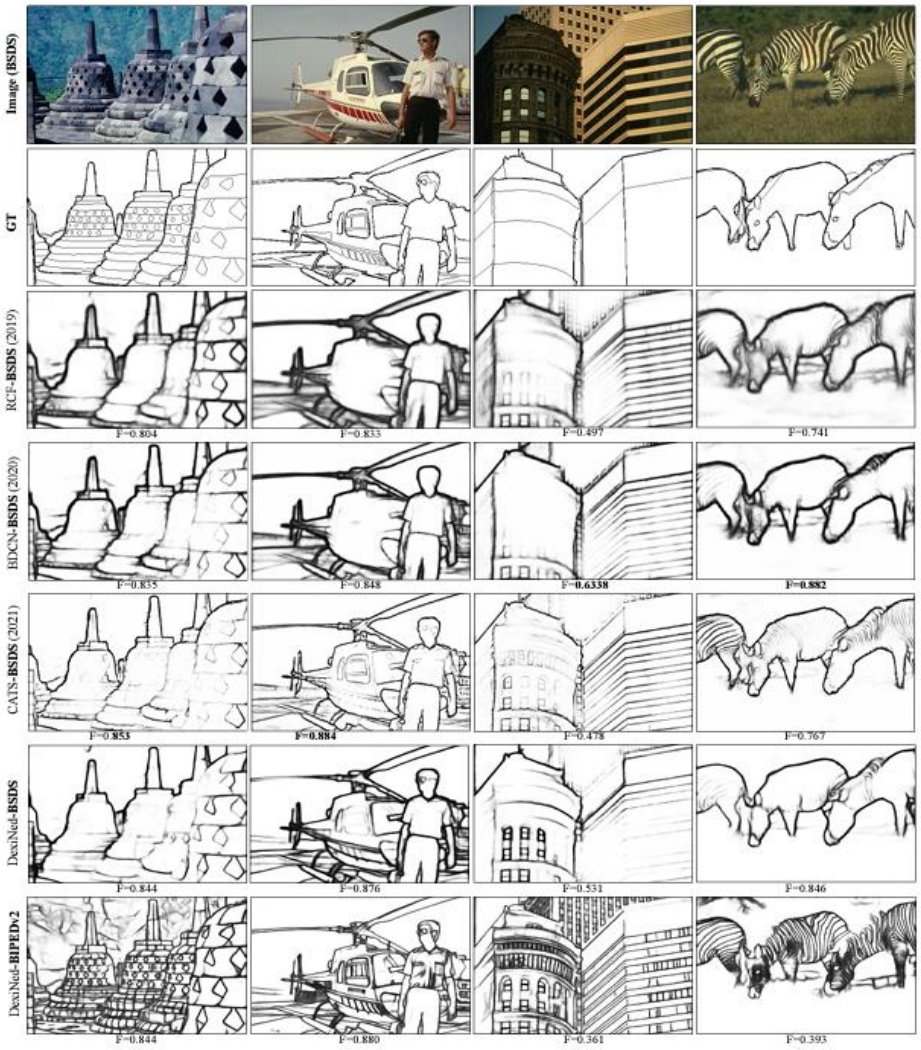

Figure 3.13. The results of several algorithms on the BSDS dataset. The suffix in model names (i.e. BSDS and BIPEDv2) indicated the training of that model [32]

"DexiNed" is structured as two sub-networks: Dense Extreme Initialization Network (Dexi), which accepts an RGB image as input, and Up sampling Block (UB) that utilizes the feature map output from each Dexi block as input, as shown in Figure 3.14.

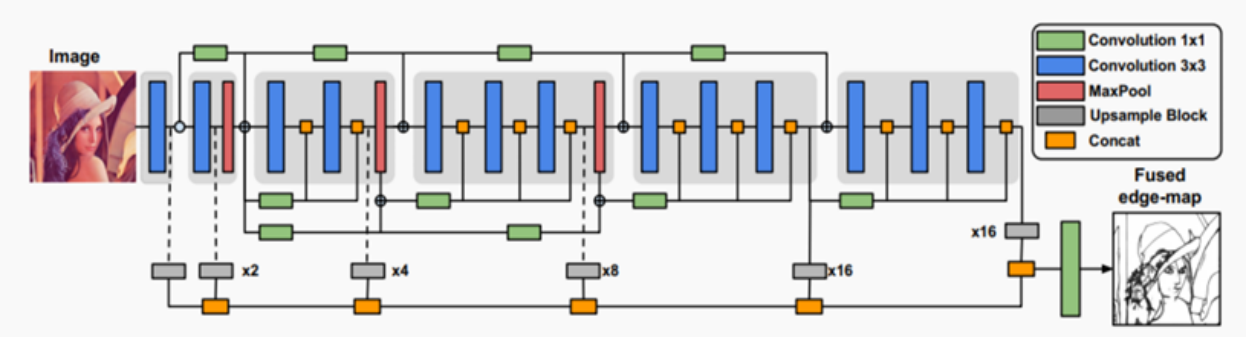

Figure 3.14. DexiNed architecture [32]

1）DexiNed Architecture

DexiNed comprises an encoder featuring 6 output blocks inspired by Xception networks. The network initially generates feature maps at each main block. Subsequently, the up-sampling block, as defined in Section 2.2, produces intermediate edge maps. These edge maps, created by the up-sampling block, are concatenated and passed to the learning filter at the network's conclusion, generating a fused edge map. Notably, all up-sampling modules operate without parameter sharing.

In the architectural layout, the blue block consists of a dual convolutional layer stack with a 3×3 kernel size, followed by batch normalization and ReLU activation function (excluding the last convolution in the final sub-block). Max pooling is set with 3×3 kernels and a stride of 2. Given the multi-scale learning approach, akin to HED, an up-sampling process follows (as indicated by grey horizontal blocks).

Given the numerous convolutional operations, essential edge features are lost in each depth block, necessitating multiple connections. Starting from the fourth convolutional layer, edge feature loss becomes more pronounced. To address this:

i) After the max pooling operation and before summing with the main connection, the edge connection averages the output of each sub-block (as depicted in the green rectangle at the bottom).

ii) Edge connections from max pooling and block 2 feed subblocks in blocks 3, 4, and 5. However, subblocks in 6 are exclusively fed from block 5 output.

2) Up sampling Block

"DexiNed" is designed to enhance the visualization of predicted edge maps by producing thin edges. A crucial element in refining these edges is the up-sampling block (UB). Each output from the Dexi block is directed to UB.

UB comprises sub-blocks stacked conditionally, and each sub-block consists of two layers: a convolutional layer and a deconvolution layer. There are two sub-block types:

Sub-block 1 (sub-block 1): Fed from the Dexi module or sub-block 2 input and is utilized when the ratio between the feature map and the ground truth map is 2.

Sub-block 2 (sub-block 2): Activated when the difference exceeds 2, and it iterates until the feature map scale is 2 concerning GT.

This approach ensures precise scaling and refinement of feature maps, contributing to the effectiveness of DexiNed's edge detection capabilities.

3.2.2.3 Validation of CNN

To validate the applicability of these models in seed contour detection, this dissertation implemented both HED and DexiNed models as per the original author's guidelines. Two images were chosen from the original database for verification purposes. The results of this verification are presented in Figure 3.15.

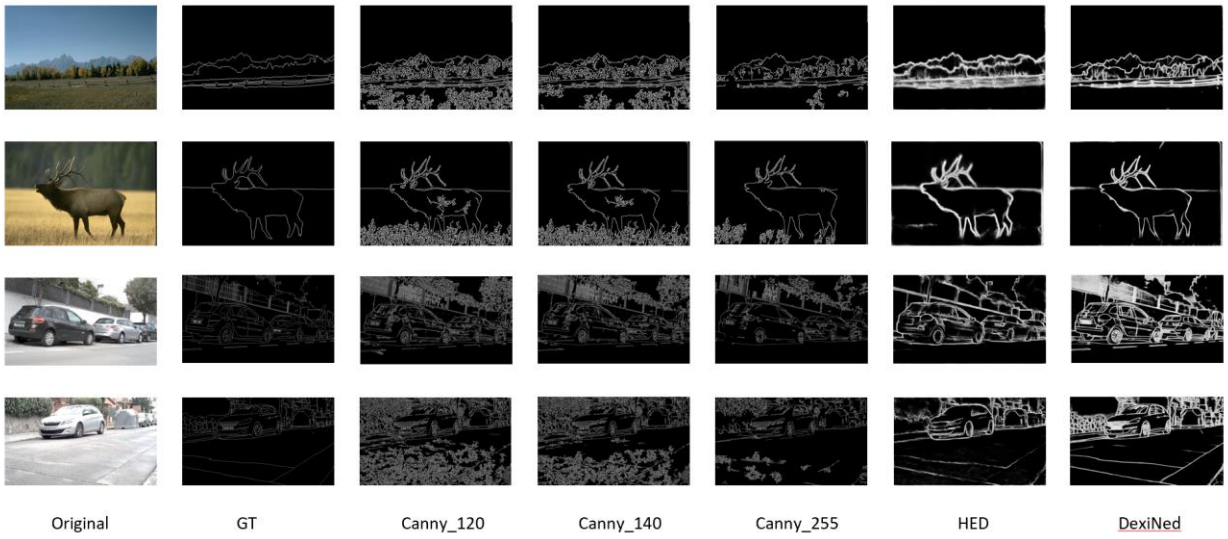

Figure 3.15. Original images and results of different methods

To evaluate the potential impact of lighting conditions on the model, we captured images in both bright and dark environments for comparative analysis. Additionally, to assess the influence of background color on the model, we compared images taken with a blue background to those taken with an LED background. The results, Figure 3.16 and Figure 3.17, clearly demonstrate that both lighting and background color have an almost negligible impact on the performance of the models.

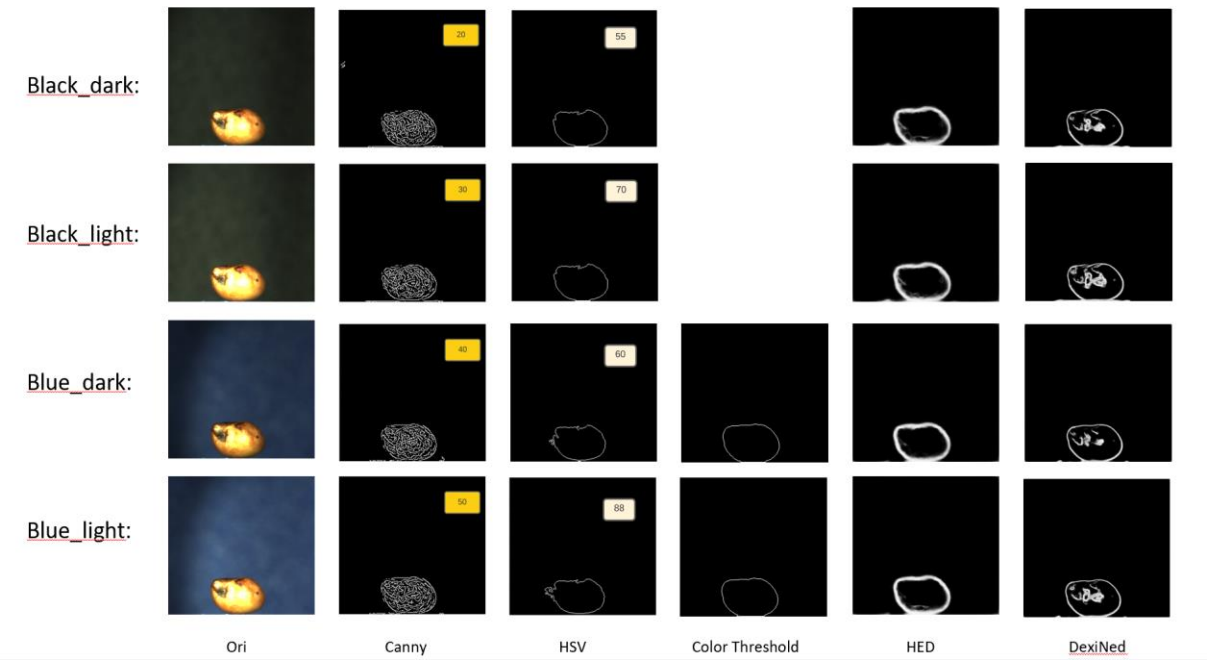

Figure 3.16. The results under different lighting and background

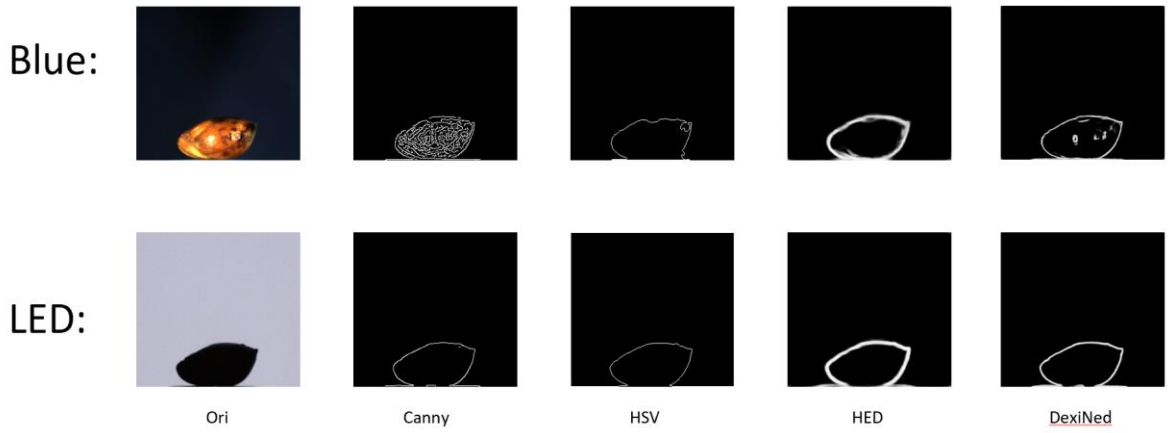

Figure 3.17. The results with blue and LED backgrounds

Both methods are primarily designed for edge detection, and the quality of their results can vary based on the training dataset. Overall, their performance is quite satisfactory. It's worth noting that DexiNed, which utilizes a custom dataset, demonstrates even more impressive results, as shown in Table 3.2.

Dataset	Methods	ODS	OIS	AP	Size	Test Time
MDBD	HED	0.851	0.864	0.890	55M	0.46
	DexiNed	0.859	0.864	0.917	138M	0.25
BSDS500	HED	0.790	0.808	0.811	55M	0.46
	DexiNed	0.729	0.745	0.689	138M	0.25
BIPED	HED	0.829	0.847	0.869	55M	0.46
	DexiNed	0.859	0.967	0.905	138M	0.25

Table 3.2. Comparison of different methods

However, both edge detection methods still need to be improved when applied to our seed contour detection. HED, while proficient in contour detection with minimal noise within the contours, tends to produce excessively thick and blurry contours, thereby affecting accuracy. On the other hand, DexiNed produces skinny and clear contour lines, but some contours are discontinuous, and there tends to be abundant noise within the contours. These limitations are likely attributed to their training datasets, which primarily focus on edge detection rather than contour detection. Moreover, a common drawback of both methods is their larger model size. The images involved in our seed contour detection are relatively simple; hence, we need to construct a smaller model incorporating the essential functions of the previous models and train the new model using our seed contour data.

3.3 DexiNed Model with new Dataset

To optimize the utilization of DexiNed, we took the initiative to create our own database. Subsequently, we employed this database to train a custom model tailored to our specific needs. This approach ensures that the model is fine-tuned to our requirements and enhances its

performance in addressing our unique challenges or objectives. By leveraging our proprietary database for training, we aim to maximize the effectiveness and relevance of the DexiNed model in our specific context.

3.3.1 Create Dataset for Training

Agricultural seed researchers classify milo into four sizes: small, small-medium, large, and extra-large. Additionally, they divided Milo into two color categories: white and black (dark green with black spots, to be exact). To fully capture the variation in size and color, we crossed the sizes and categories and selected two seeds from each, for a total of sixteen seeds. For each seed, 36 photos were taken, with each surround flipped upside down, for 1152 photos. Contour detection on blue background plate using color threshold method. Subsequently, manual parameter adjustments were performed to fine-tune the detection process based on the different characteristics of each seed. This meticulous approach can obtain corresponding ground truth images. We obtained carefully curated and suitable 648 images for further analysis and research.

Considering that 648 images might be insufficient for CNN training, we employed image enhancement techniques for augmentation. Each image and its ground truth image underwent various transformations, including random horizontal inversion, rotation, horizontal and vertical shifting, brightness adjustments (strengthening or weakening), and scaling, as shown in Figure 3.18. This process generated a set of 21 corresponding images for each original image. Consequently, we amassed a total of 13,608 image sets. For training purposes, 80% of these sets, selected randomly, were designated as the training dataset.

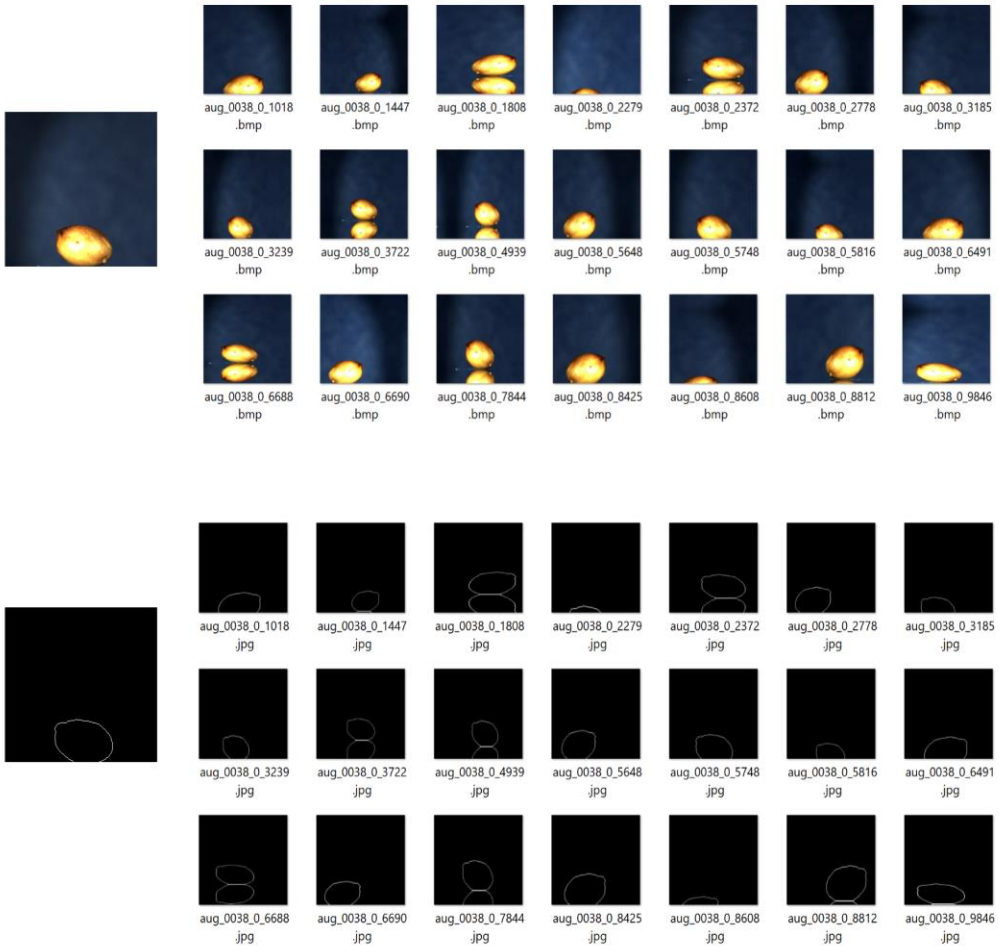

Figure 3.18. Single sample and augmented images with ground truth

3.3.2 Result

After 30 training epochs (as shown in Figure 3.19), the new model has successfully met our expected objective. The model's output exhibits a desirable outcome, where only the contour lines are retained. Moreover, these contour lines are skinny and exhibit no disconnections, closely aligning with the characteristics of the ground truth images. As shown in Figure 3.20.

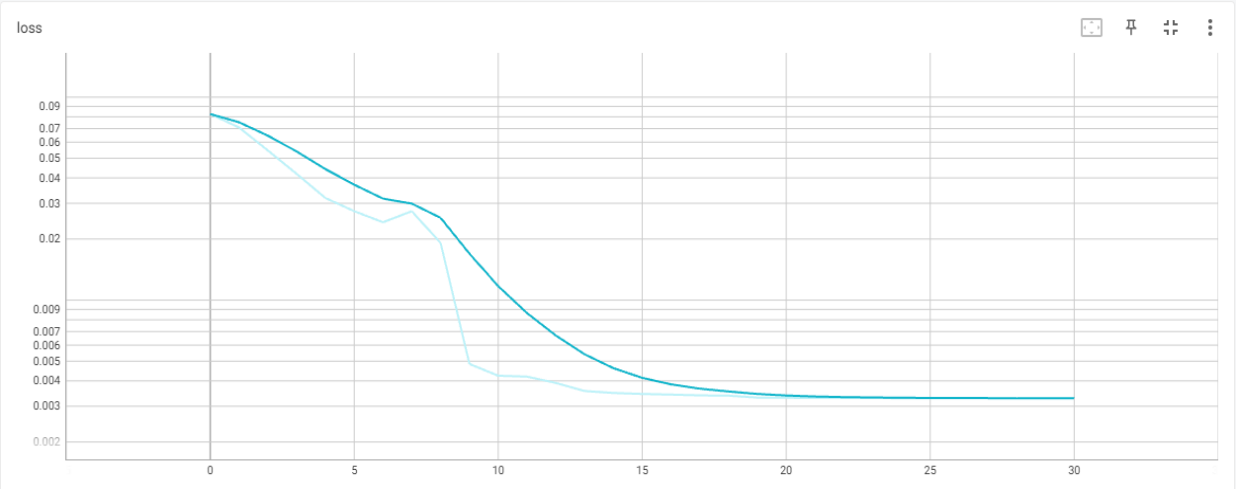

Figure 3.19. The curve line of loss for each epoch

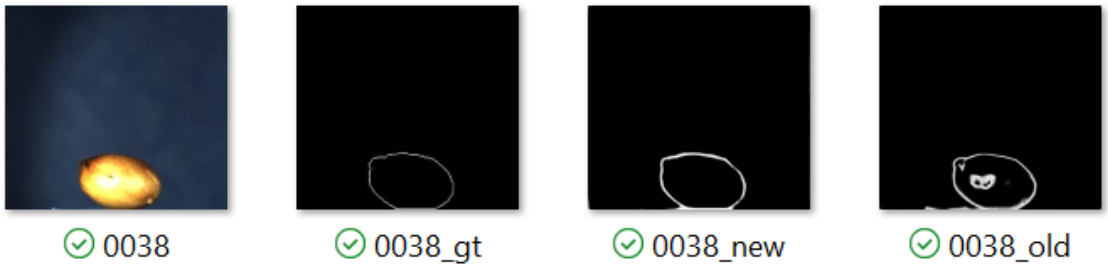

Figure 3.20. From left to right: original image, ground true image, test result using DexiNed with new training dataset, test result using DexiNed with old training dataset.

3.4 Summary of DIP

This dissertation discusses popular and user-friendly traditional contour detection methods, namely the HSV-based and color threshold-based methods mentioned earlier. The effectiveness of these methods varies depending on the background color. The former relies on the color of the seed itself, precisely gauging the degree of light reflection on the seed surface. On the other hand, the latter is contingent on the background color. Table 3.3 below provides a summary of their effectiveness.

Methods	The main cons	Running Time(s)	Accuracy Level
Sobel Filter	Contours detected do not include the black part of the seed.	3.686	1
Canny	Contours are not closed.	0.003	1
HSV_Black	Contours detected do not include the black part of the seed.	0.003	1
HSV_Blue	User must set light LED value.	0.003	1
HSV_LED	Seed color is not precisely obtained.	0.003	3
Color Threshold	Users must specify the color range.	0.013	2
CNN	The size of the model is very large.	0.46/0.25	3

Table 3.3. The summary of each contour detection approach

We divide the accuracy of contour detection into three levels:

Level 1: Low accuracy with adjusting some parameters manually.

Level 2: High accuracy with adjusting some parameters manually; or median accuracy without adjusting any parameters manually.

Level 3: High accuracy without adjusting any parameters manually.

The level definition of accuracy is: If the edge of any edge image is broken or significantly deviated, it is a low level. When comparing one image group (36 images per seed), all edge images must be close to the original image, with an error of 1 pixel allowed, which is high level. When comparing one image group, the probability that the edge image fits the original image does not reach 90%, which is the median level.

Chapter 4 - Shape-from-Silhouette Method for Seed Kernel Reconstruction

4.1 Base Introduction

The silhouette of an object in an image refers to the contour that separates the object from the background. The shape from silhouette method requires multiple images captured from different angles to perform 3-D reconstruction of a seed kernel. For each of the images captured, the silhouettes are segmented using a method known as background subtraction. The silhouettes are then back-projected onto a common 3-D space with projection centers equal to the camera locations. The back-projection onto a common space requires the intrinsic camera matrix K and distance between the origin of the working volume and camera center. Figure 4.1 offered a sample of cameras setting. The origin of the working volume is considered to be the intersection point (IP) of the seed bottom horizontal line and the line drawn through the center of the image. A total of N images equidistant from each other spaced at rotation angles ai where i ε {1…, N} are acquired. The rotation is around the vertical axis through the IP and parallel to the Y-axis of the camera. A grayscale threshold on each of the acquired images is applied and segmented into a binary mask Mi where i ε {1…, N}. For each image, the camera projection matrix Pi is calculated from the rotation angle ai by $PPii = KK(RRii \mid TTii)$ where Ri is the rotation matrix and Ti is the translation matrix corresponding to the given angle ai. Then, an equidistantly spaced cubic voxel grid around the world origin is defined around the world origin. The size of each voxel is set to 1 mm3. Each voxel center with homogenous world coordinates $\vec{XX}$ is projected to a point, $\vec{xx}$i in each mask Mi by $\vec{xx}$i = Pi$\vec{XX}$. If a voxel belongs to the foreground object, its value V ($\vec{XX}$) is set to 1. If the voxel does not belong to the foreground object, its

value V ($\overrightarrow{XX}$) is set to 0. The mask Mi in volume carving is sensitive to misalignment of the object volume and needs to be carefully adjusted.

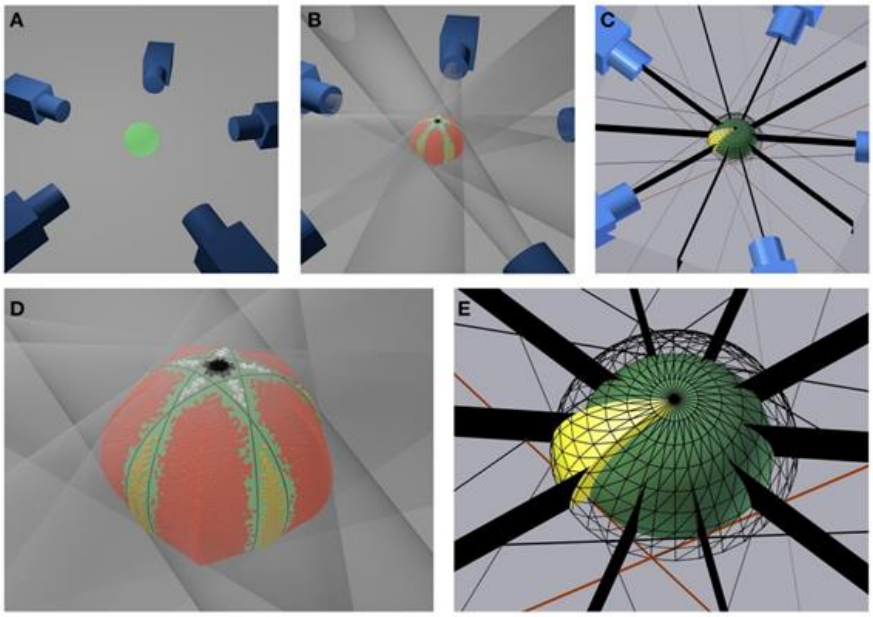

Figure 4.1. Five cameras setting [35]

4.2 Materials and Methods

The concepts that help to understand the estimation of volume using the silhouette-based approach are explained as follows. Please note that the underlying mathematical derivations are out of the scope of the article.

Voxel: A voxel is the unit of information that defines a point in 3-D space. A pixel in 2-D space is analogous to a voxel in 3-D space. Simply put, they are virtual cubic blocks that are representative of 3-D space.

World Coordinate System: It is the basic 3-D cartesian coordinate system with an origin assigned by means of arbitrary assumption. In other words, it may be defined as any point in the 3-D space.

Camera Coordinate System: It is a 3-D coordinate system that measures relative to the camera's origin and orientation. It is possible to apply the operations of rotation and translation to convert a point in the world coordinate system to the camera coordinate system. There exists a 4 x 4 transformation matrix known as Camera Extrinsic Matrix which applies rotation and translation to convert a point from world coordinate system to camera coordinate system. The camera extrinsic matrix changes with the physical location of the camera.

Image Coordinate System: It is the coordinate system that projects the 3-D point from camera coordinate system to the 2-D plane. Only the height and width of the 3-D point are captured but there is no sense of depth in the image coordinate system. Hence, it is a lossy transformation that cannot be reversed.

Pixel Coordinate System: In order to discretize the image coordinate system, it is divided into pixels. The pixel coordinates of an image are discrete values within a range computed by dividing the coordinates in the image coordinate system by pixel width and pixel height.

Intrinsic Parameters: The parameters that define the relationship between the image coordinates and camera coordinates are called Intrinsic parameters. Intrinsic parameters are specific to the camera in use and a meticulous calibration of the intrinsic parameters is required to avoid image distortion. The intrinsic parameters typically considered are:

1. Focal Length: Focal Length is defined as the distance between the image plane and origin of the camera coordinate system.

2. Principal Point: The Principal Point is the point where the Optical axis i.e. the Z-axis of the camera coordinate system, intersects the image.

Figure 4.2 presents a graphical representation of the intrinsic parameters of focal length and principal point.

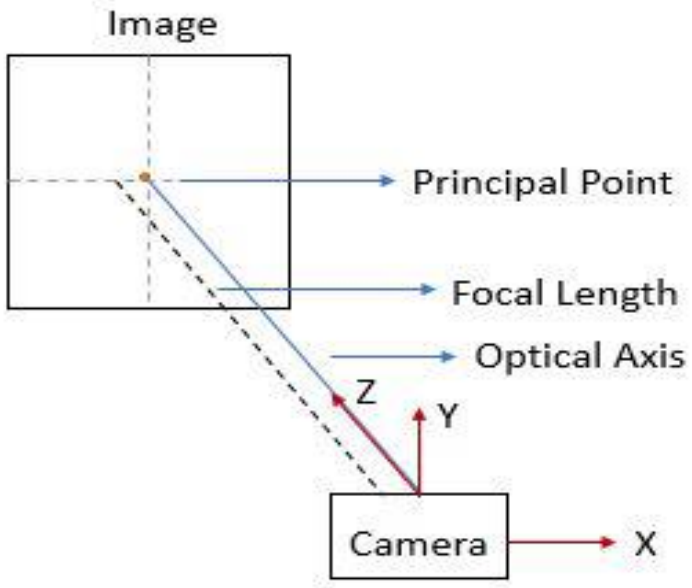

Figure 4.2. Intrinsic parameters

4.3 Multithreaded Space Carving

Once we have the image and transform it into a 2D binary mask image using digital image processing (DIP), we apply our space carving algorithm. This algorithm is instrumental in deriving the target's 3D rendering and volume. Figure 4.3 illustrates the rendering of a single

mask image following the space carving process, while Figure 4.4 showcases the rendering of a collection of mask images after space carving, along with the final 3D image.

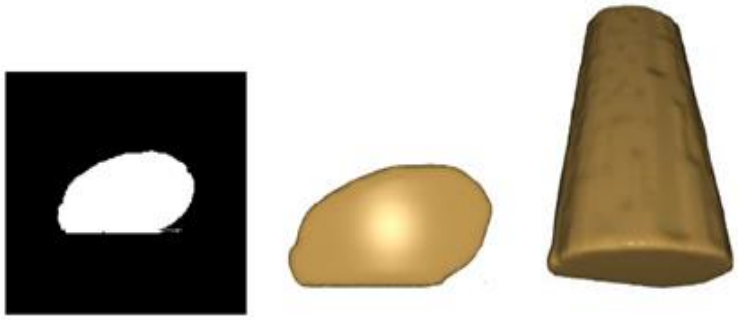

Figure 4.3. Mask/voxels after carving, front/high-angle view

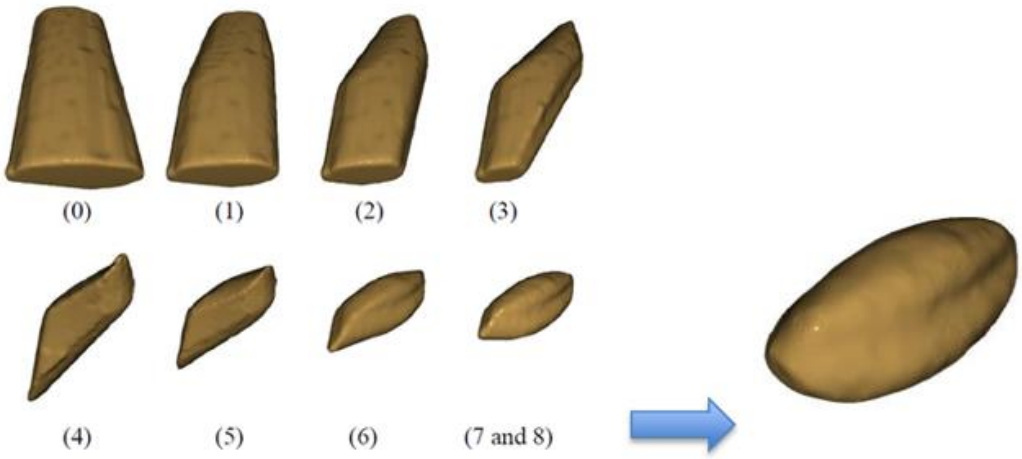

Figure 4.4. Space carving with masks and final voxels

The algorithm used for space carving is simple and can be easily parallelized. For each mask, carve away the voxels that are not projected onto a mask (the black portion of the image shown in Figure 4.3). Thus, different masks can be processed in parallel by different threads, each responsible for processing a subset of the masks. Although it is logical to allow all of the threads to run in parallel, it turns out to be more efficient to have one (or possibly 2) threads run first and preprocess the voxels by carving away a single (or orthogonal) mask.

Suppose that the voxels are stored in a byte array V[depth][height][width], and the projection matrices are stored in P[mask][The basic algorithm is based on the classic Space Carving Algorithm (Kutulakos, et al., 2000), but instead of projecting each voxel onto all masks, we consider the voxels that can be carved away by projecting them onto a single mask. The voxel data structure is shared by all threads. Since they are only setting an initial voxel value from 1 to 0 if the voxel is carved away, there is no race condition.

The Base Space Carving Algorithm is shown below:

1. Initialize the voxels V to be a volume containing the true object. For the experiment, we just used a simple 3-D 100 x 100 x 100 voxel grid.

2. For each mask:

    For each voxel v ϵ V:

        a. Project v onto the mask to 2-D point v

        b. If v is in the background of the mask (i.e., it is non-photo-consistent) (black) and v is 1, set v = 0

In this way, each mask can be processed by a different thread. However, once a voxel has been carved, it's not necessary to see if it should be carved away again. Thus, we can add a test to check the target voxel to see if it has already been carved, before going to the work of projecting it onto the mask to see if it should be carved based on the current mask. This leads to a second algorithm that we call the Check Target Space Carving Algorithm:

1. Initialize the voxels V to be a volume containing the true object.

2. For each mask:

    For each voxel v ϵ V, if v == 1:

a. Project v onto the mask to 2-D point v

b. If v is in the background of the mask (i.e., it is non-photo-consistent) (black) and v is 1, set v = 0

As we shall see, this performs better for a small number of threads, but it performs worse for many threads because of the extra test. Finally, to really speed things up, we found that greater benefits can be realized by allowing one (or a few) threads to pre-carve the object, before unleashing most of the worker threads to finish the job. The intuition is that, due to caching, if we had 36 threads all carving based on their own mask, then they would need to check all the voxels. In our case, that is 1,000,000 voxels. But, as we saw with a typical seed in Figure 4.4, about 90% of the voxels are removed by carving out with the first mask. Thus, we settled on the Preprocess and Check Target Space Carving Algorithm:

1. Initialize the voxels V to be a volume containing the true object.
2. For the first mask, preprocess the voxels:

   For each voxel v ϵ V, if v == 1:

   a. Project v onto the mask to 2-D point v

   b. If v is in the background of the mask (i.e., it is non-photo-consistent) (black) and v is 1, set v = 0
3. For each remaining mask using updated voxels V:

   For each voxel v ϵ V, if v == 1:

   a. Project v onto the mask to 2-D point v

   b. If v is in the background of the mask (i.e., it is non-photo-consistent) (black) and v is 1, set v = 0

Of course, the improvement in speed depends on the fact that the object (in our case a seed) is a relatively small percentage of the total number of voxels in 3-D space; in our example, around 10%, but significant improvement in performance should be obtained with even a larger number of voxels remaining after the preprocessing step, at least up to 20%.

4.3.1 Performance Analysis

Experiments using multiple seeds were performed on a Dell G16 7620 Windows 11 Pro with an Intel® Core ™ i7-12700H CPU @ 2.3 GHz, 14 Core(s), and 20 Logical Processors, and 16 GB RAM, and a more modest Dell Latitude 7480 Windows 10 Pro with an Intel® Core ™ i7-6600U CPU @ 2.6 GHz, 2 Core(s), and 4 Logical Processor(s), and 8 GB RAM.

For the Base Space Carving Algorithm, running on a Dell G16, the average read time is 2.75 milliseconds. This could be reduced to 1.75 ms if we assumed the voxels would be set to all 1s initially. With one thread, space carving took an average of 0.231 seconds as shown in Figure 4.5. This average was computed by running the experiment 500 times. The time ranged between 0.229 sec. to 0.243 sec. Nearly linear speedup is achieved going from 1 to 6 threads. With 6 threads, the speedup is 5.27 with an average carve time of 0.044 sec. After that, note that 7 threads exhibit worse performance because there are 36 tasks which are evenly divided among 6 threads, but not evenly divided among 7 threads. With 6 threads, each thread is assigned 6 tasks, but with 7 threads, most are assigned 5 tasks, but one thread is assigned 6 tasks. The next peak in performance is seen with 12 threads (each thread is assigned 3 tasks) and the speedup is 7.29. Next, with 18 threads (each thread is assigned 2 tasks), the speedup is 7.89 with a carve time of 0.029 sec. Finally, a slightly better speedup of 7.98 is obtained with 36 threads.

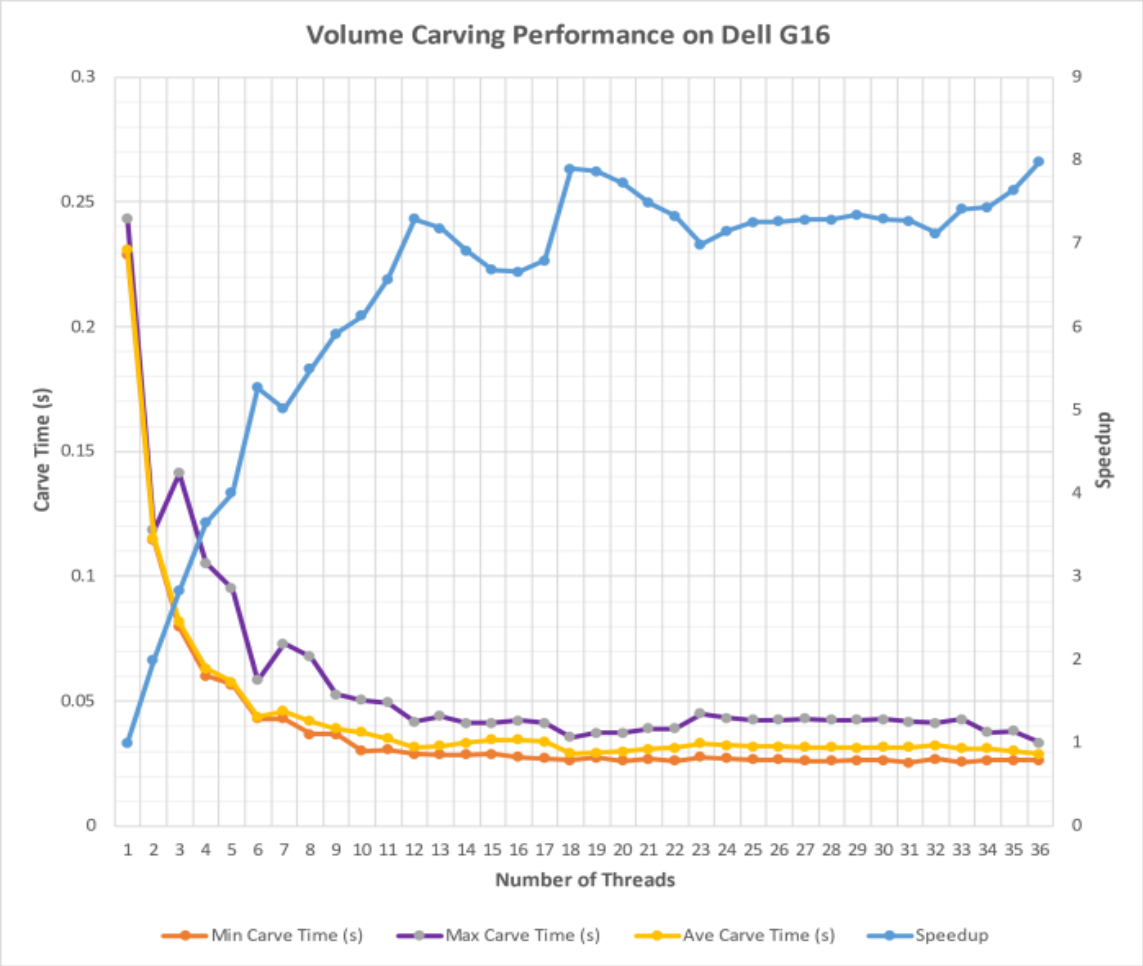

Figure 4.5. Performance analysis

A very interesting result was obtained while comparing the execution time of all three carving algorithms on a Dell G16. We completed 500 trials for each algorithm using a range of worker threads, from 1 to 36, and compute the average carve time required for all threads to complete as shown in Figure 4.6. Remaining tasks are assigned to threads using a simple modular assignment. With 36 tasks numbered 0 to 35 (each associated with one of the masks to carve), and with n threads, numbered 0 to n-1, in the first two algorithms, thread i is assigned tasks { i + j*n | j=0, .., k, where $(36-i)/n - 1 \leq k < (36-i)/n$ }. For example, if n = 2, thread 0 is assigned all even tasks and thread 1 is assigned all odd tasks. If n = 36, then each thread is

assigned one task; that is, thread i is assigned task i, for i = 0, ..., 35. For n = 12, each thread is assigned three tasks:

Thread 0: task 0, 12, 24

Thread 1: task 1, 13, 25

..

Thread 11: task 11, 23, 35

In this way, the threads can be scheduled to run on 12 cores, each with an equal amount of work for n = 18, each thread is assigned two tasks. After carving, we completed a sanity check to ensure that the same number of voxels remained. For our test case, 21,234 voxels of the available 1,000,000 remained after carving. And it only took about 0.0001 s = 0.1 msec. to compute. The time to write out the 1MB voxel cube took about 0.0006 s = 0.6 msec. For the Base Algorithm, the total time to read, volume carve, validate, and write the result was only about 0.033 seconds on average with 18 threads (of that 0.029 second for carving), and ranged up to 0.235 seconds for one thread (with 0.231 seconds for carving). Thus, in all cases, the dominant time was the time to complete space carving, accounting for over 87% of the time required.

To speed that up, we added the check to verify that a voxel hasn't already been carved. The second algorithm, Check Target, obtains an optimal speed of 0.0259 with just 2 threads. Note however that once the number of threads exceeds more than half the number of tasks (36), then the performance degrades to be worse than the Base Algorithm. This is because the additional test isn't really helping any more.

Finally, to obtain the best result, we execute one thread sequentially to preprocess or pre-carve the voxels before the other threads even start running. The purple line in Figure 4.6 shows that this approach is best and achieves a carving time of 0.0111 seconds consistently over a wide range of threads.

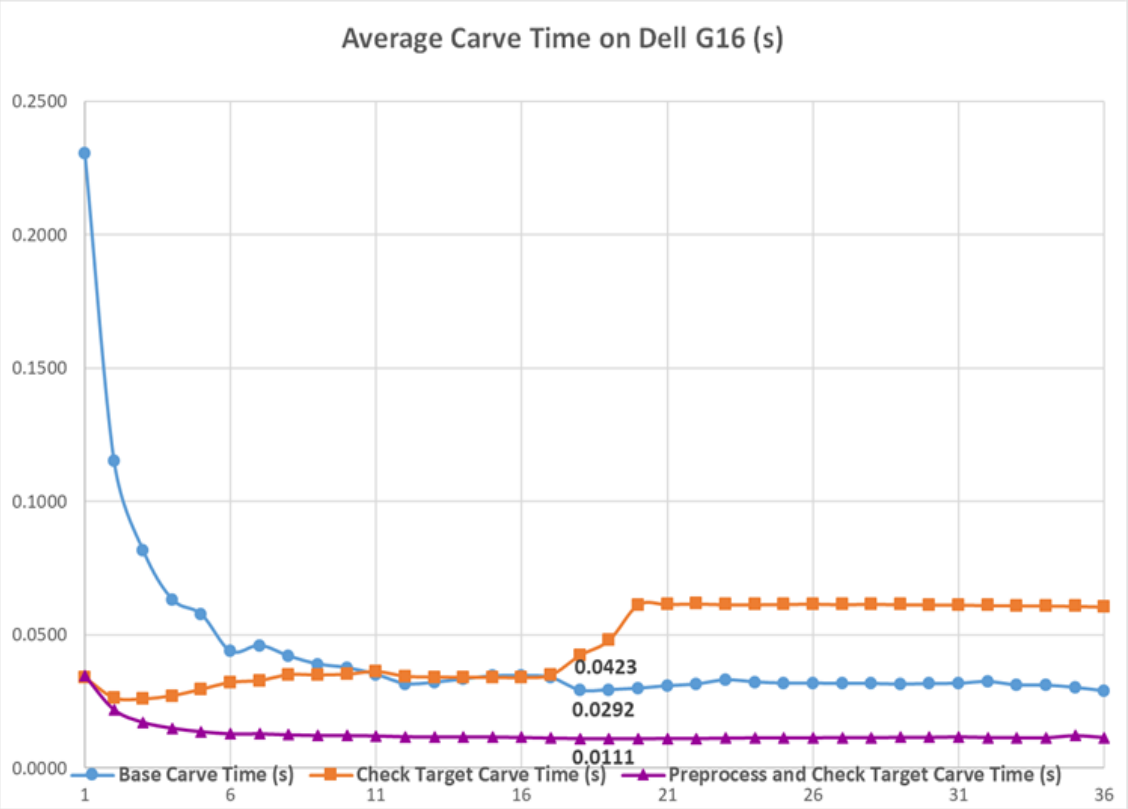

Figure 4.6. Performance analysis of all three algorithms on Dell G16

With more modest hardware, similar results are obtained, shown in Figure 4.7. The same test is performed on a Dell Latitude 7480 with 1 to 4 threads. Since the hardware only has a dual core, adding more threads doesn't improve the results. For the Base Algorithm, the best performance is obtained with 4 threads, with an average carving time of 0.2794 seconds. By checking the target, the second algorithm achieves an optimal carving time with three threads at

0.0487 seconds. Finally, preprocessing helps a little bit and achieves an optimal carving time of 0.0415 seconds with four threads. Note that the improvement is not as much because fewer cores are available. However, it is important to note that the modifications to the Base Algorithm did result in improved carving times.

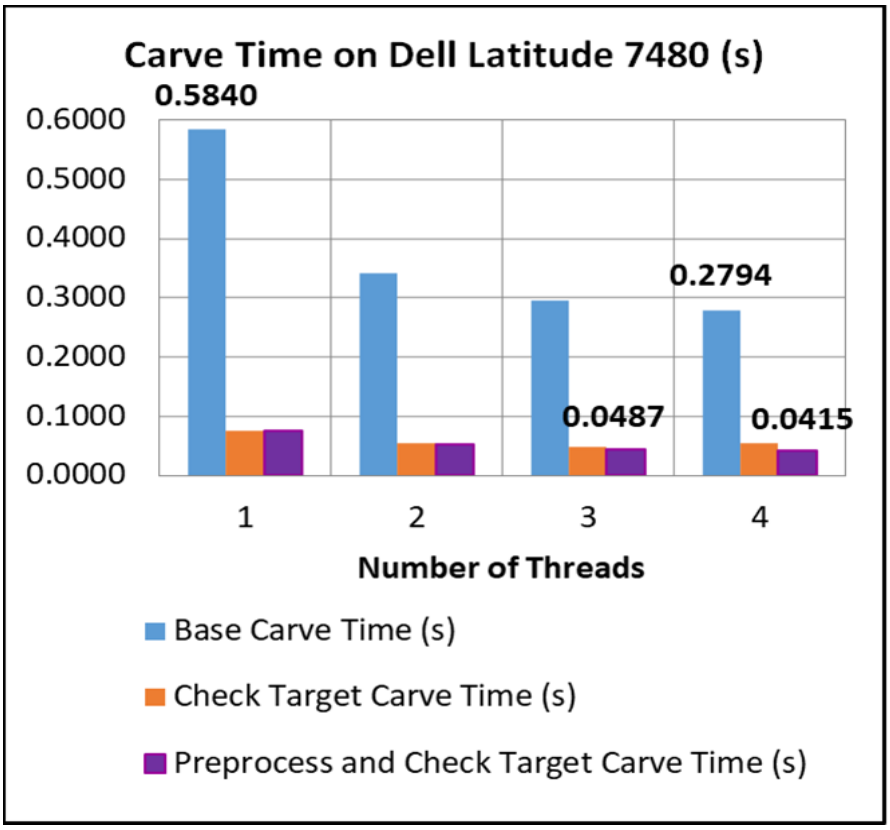

Figure 4.7. Performance on Dell Latitude Dual Core

On relatively modest hardware, the framework is capable of computing seed volume in just 0.5 seconds. However, our newly developed multi-threaded version can yield results in a mere 0.03 seconds on a more advanced machine equipped with multiple cores. This multi-threaded version demonstrates nearly linear speedup with up to 6 threads. In summary, this new framework with a multi-threaded version offers a practical and cost-effective solution for accurately determining the seed's volume, even on less powerful hardware.

4.4 Compare with Slicing Methods

The volume of each of the seed kernels is estimated based on the proposed 3-D reconstruction technique. However, there is no known standard technique to estimate the volume of a seed kernel. Manual measurement using a caliper is error prone since the measurements are not reproducible across different individuals and subject to bias. To verify and validate that the volumetric estimation made by the 3-D reconstruction technique is within reason, the seed kernel is approximated to a mathematical shape whose volume may be estimated. While the likelihood that the shape of any seed kernel fully conforms to a mathematical shape is minimal, the approximation acts to verify that the results output by the 3-D reconstruction technique are reasonable. In general, seed kernels have varying shapes which makes developing a universal model that works for all seed types of complexes. As part of the experiments for the current work, the seeds of wheat and soy are considered. The closest mathematical shape that the seeds of soy and wheat may assume is that of the ellipsoid since it captures the properties of both spherical (soybean) and oblong (wheat) seed types. The volume of a seed kernel upon being assumed to be an ellipsoid is the aggregate of the individual cross-sectional volumes. Each cross-section of the ellipsoid is an elliptic cylinder, as shown in Figure 4.8. The volume of the ith cross-section is given by $vi = \pi.Xi.Yi.Zi$ where vi is the volume of the ith cross-section of an ellipsoid (elliptic cylinder), Yi is the radius of the major axis and Zi is the radius of the minor axis of the ith cross-section of the ellipsoid and Xi is the height of the ith cross-section of the ellipsoid.

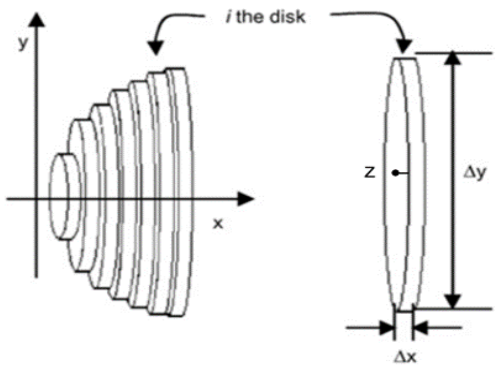

Figure 4.8. The principle of Slicing

The goal of image acquisition in slicing methods is to capture the seed kernel without ignoring even the slightest detail. The estimation of volume using the ellipsoid slicing technique requires the top view and side view of the seed kernel. The length and width of the seed kernel are obtained from the top view whereas the side view is required to obtain the thickness of the seed kernel. The acquisition of seed kernel images is performed using two different techniques, Figure 4.9, that capture the seed kernel from the top view and side view.

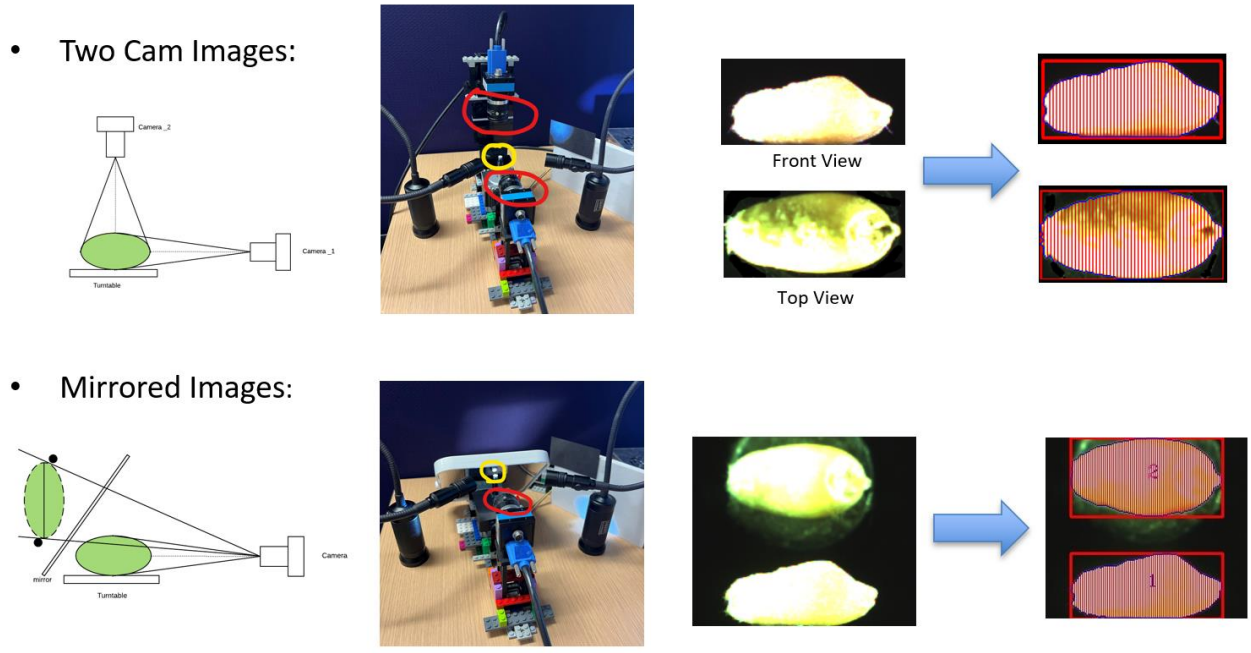

Figure 4.9. Setting of two cameras image (above); Setting of mirrored images (below).

Experiments using the proposed 3-D reconstruction framework and ellipsoid slicing techniques are conducted on a random sample of five soy seed kernels and five wheat seed kernels. The experiment using each of the techniques is carried out for a total of 20 times to also detect any variability in the estimations. The images used for the ellipsoid slicing technique are captured using the mirror-based approach and two-camera based approach. To compare the volumetric estimations between the two techniques, Relative Standard Deviation (RSD) is used as the metric.

RSD, otherwise known as Coefficient of Variation, is the measure of standard deviation, relative to the mean, of the dataset. It is a unitless measure of dispersion obtained by dividing the standard deviation by the mean value of a sample. Its use is preferred in cases where a comparison on the dispersion of two datasets of different units is made. Computationally, it is defined as RSD = s/M where s is the standard deviation and M is the mean value of the dataset. A comparison of the RSD for each of the five wheat seeds using the mirror-based approach and

3-D reconstruction approach is shown in Table 4.1. Similarly, a comparison for five soy seed kernels is shown in Table 4.2.

Wheat						
ID	Mirror-based Approach			3-D Reconstruction Approach		
	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation
1	0.318	27.602	0.011	0.13	27.912	0.004
2	0.244	29.401	0.008	0.102	30.670	0.003
3	0.443	31.642	0.014	0.142	31.645	0.004
4	0.356	27.409	0.012	0.127	28.560	0.004
5	0.296	25.897	0.011	0.108	26.742	0.004

Table 4.1. Mirror-based approach VS 3D approach for wheat

Soy						
ID	Mirror-based Approach			3-D Reconstruction Approach		
	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation
1	0.638	147.918	0.004	0.3845	146.2841	0.002
2	0.605	106.057	0.005	0.3448	104.1521	0.003
3	0.973	132.855	0.007	0.375	130.1393	0.002
4	1.347	151.763	0.008	0.5594	150.6711	0.003
5	0.468	135.773	0.003	0.3283	133.4657	0.002

Table 4.2. Mirror-based approach VS 3D approach for soybean

A comparison of the RSD for each of the five wheat seeds using the two camera-based approach and 3-D reconstruction approach is shown in Table 4.3. Similarly, a comparison for five soy seed kernels is shown in Table 4.4.

Wheat						
ID	Two Camera-based Approach			3-D Reconstruction Approach		
	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation
1	0.473	28.727	0.016	0.195	28.47	0.006
2	0.408	30.332	0.013	0.229	32.615	0.007
3	0.356	32.368	0.011	0.207	33.437	0.006
4	0.344	27.845	0.012	0.109	29.042	0.003
5	0.254	26.582	0.009	0.196	27.093	0.007

Table 4.3. Two Camera-based approach VS 3D approach for wheat

Soy						
ID	Two Camera-based Approach			3-D Reconstruction Approach		
	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation	Standard Deviation	Avg. Volume (mm^3)	Relative Standard Deviation
1	1.406	147.209	0.009	0.369	147.517	0.002
2	0.768	106.897	0.007	0.372	105.779	0.003
3	0.705	132.406	0.005	0.313	132.135	0.002
4	0.813	152.475	0.005	0.392	151.990	0.002
5	0.636	135.205	0.004	0.423	133.147	0.003

Table 4.4. Two Camera-based approach VS 3D approach for soybean

Please note that the five seeds of wheat and soy used in the experiment are the same. However, there is a minor variation in the results of the 3-D reconstruction approach between the experiments due to variation in light intensity and position of the seed kernel on the seed station. The position and light intensity are maintained constant in each experiment. The obtained results demonstrate superior RSD with the 3-D reconstruction approach in comparison to the two-camera and mirror-based approaches. The RSD of all seed kernels, soybean, or wheat, is lower for the proposed 3-D reconstruction technique in every experiment. The lower RSD indicates

that lower variance and higher level of reliability for the 3-D reconstruction technique in comparison to the others.

Chapter 5 - User Interface and 3D Model.

5.1 User Interface

The hardware and primary operational principles of the project have been successfully implemented. To cater to the needs of the Agricultural Seed Breeding Research Institute, we recognized the importance of a user-friendly interface. Consequently, we have developed a desktop application with a simplified one-click operation, as illustrated in Figure 5.1.

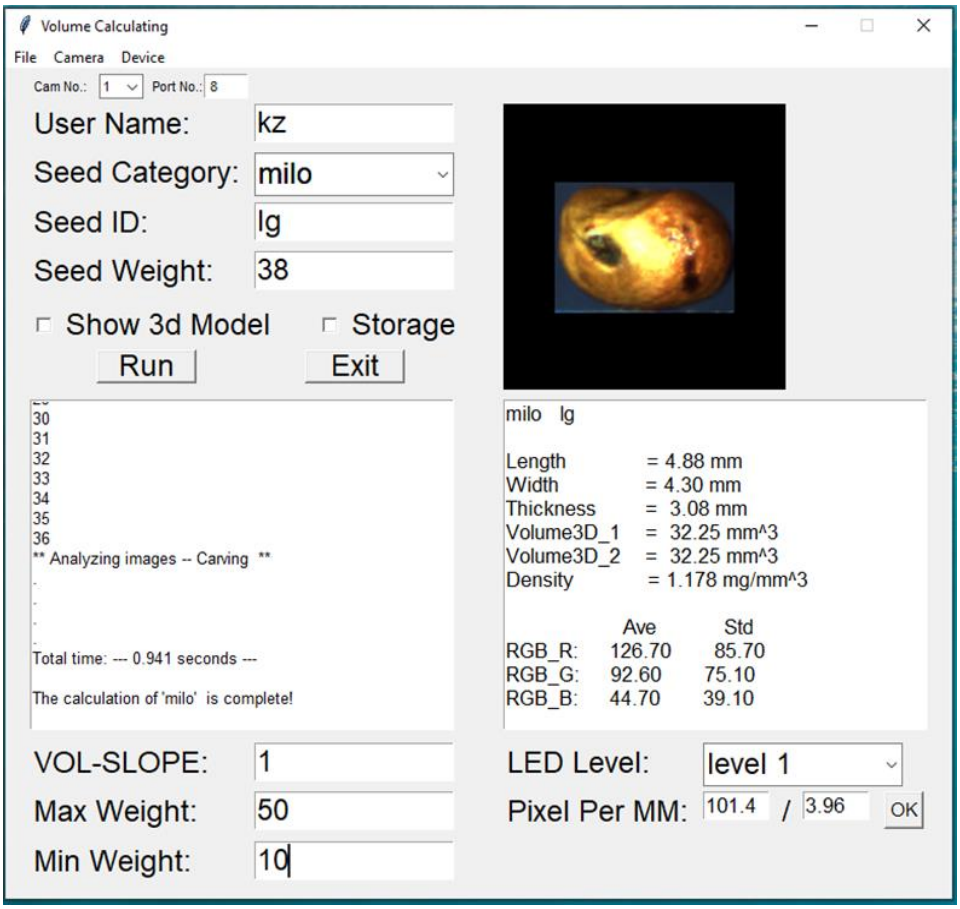

Figure 5.1. The Main User Interface

Below, we provide an overview of the elements within the user interface。

1. “Cam No.”

Select the number of cameras that come with the laptop itself or have been connected externally (excluding the cameras on our device).

2. “Port No.”

When the motor with the controller board is connected to the computer via USB, the ports of different computers are different. And the port number is easy to get from the computer.

3. “Username:”

The user who uses this application for the experiment.

4. “Seed Category:”

It lists some seed categories, such as wheat, milo, corn, and soybean. You could select one based on your experiment.

5. “Seed ID:”

The id of the current seed.

6. “Seed Weight:”

The weight of the current seed (unit is “mg”).

7. “Show 3D Model:”

It will display the 3D model of the current seed, which will take a longer time to generate the result.

8. “Storage:”

Store the captured images and the single result from the current experiment in a separate folder, which will be used for review in the future. The name of the separate folder is the current date and time, such as “2023-02-02^10-06-46”.

9. “Run:”

The main button to calculate the seed.

10. “VOL-SLOPE:”

The ratio of the actual volume and the experiment result. It is based on standard reference objects (cuboid, cylindrical, sphere).

11. “Max Weight” and “Min Weight”:

The weight range of this type of seed.

12. “LED Level:”

In fact, it corresponds to the Motor turning speed.

Level 1 corresponds to the Led light level 1; others correspond to the Led light lever 2 or more.

13. “Pixel Per MM（PPM）:”

The ratio of the pixel of the reference object (standard ball) in the image and the real size of it in the real world. When you need to change the value of them, please remember to press the “OK” button. As the distance between the camera and the target changes, the value of PPM will also change. We use three reference objects of different sizes to determine the PPM of the current camera position, as shown in Table 5.1.

(36 images)	Pixel(ave)	mm	pixel/mm	para_mm	para_pixel
lg	127.6111	4.99	25.57337		
md	121.4444	4.75	25.56725		
sm	101.4444	3.96	25.61728		
ave			25.58597	3.96	101.3204
				3.96	101.3

Table 5.1. The PPM of different reference objects

5.2 3D Voxel Model and Point Cloud

Following our 3D reconstruction method, we acquired a 3D model comprising numerous minuscule voxels. While we successfully determine the volume of the seed, the model's surface appears quite uneven. To address this, we employ a point cloud to refine and smoothen the model's surface. A point cloud [36] is a set of data points in space. The points may represent a 3D shape or object. Each point position has its set of Cartesian coordinates (X, Y, Z), which is the center of a voxel.

The process is as follows, and the result is shown in Figure 5.2:

a. Create the triangle mesh with points(vertex) and triangles using Poisson.
b. Calculate the normal of each triangle.
c. Smooth the surface of the model to make it look more natural.
d. Storage the mesh with vertex, triangle, and normal in a .stl file that could print the model in a 3D printer.

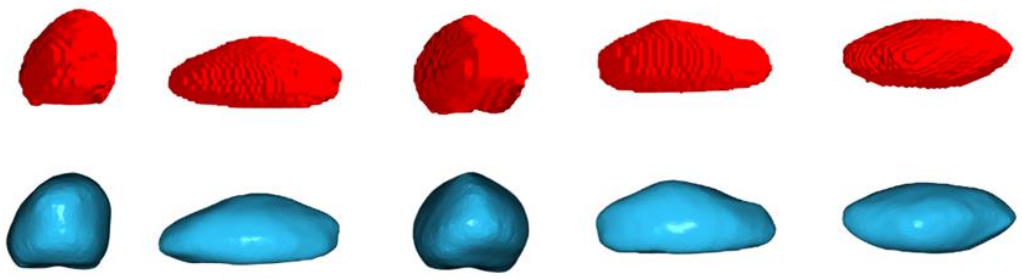

Figure 5.2. The original 3D model (above) and the point cloud 3D model (below)

Chapter 6 - Fractional Vegetal Cover Estimation using Hough Lines and Iterative Clustering.

6.1 Introduction

Companion planting is the idea of growing multiple species of plants so that they reap mutual benefits, such as improved crop yield, soil quality and pest control. One of the foremost instances of companion planting in North America is that of 'three sisters' pioneered by the Native Americans. The plant varieties of beans, corn and squash constitute the 'three sisters' where their cultivation in proximity led to the benefits of shelter and growth support for plants in addition to improved soil quality and decreased soil erosion [37]. Other combinations of plant varieties that complement each other's growth include artichoke and cucumber, beetroot and onion, and tomatoes and carrot whereas the combinations of parsnip and carrot, potatoes and pumpkin, and peas and garlic are instances of poor companions that deter one another's growth and development [38]. Hence, the discovery of companion plant species that can co-exist is of significant value to the agricultural sector. A key metric that helps estimate the growth of plant species is Fractional Vegetation Cover (FVC). FVC is defined as the percentage of the ground surface covered by vegetation elements from the overhead perspective [39]. Different techniques such as Point-and-Line cover Estimation, Plot-based cover estimation, Ocular-based estimation and semi-quantitative ocular based estimation are available to estimate FVC at a certain location. However, the techniques currently available are labor-intensive, stochastic and subjective in nature. In addition, the techniques are prone to overestimating the vegetation cover due to field survey design [40]. The need for a technique that precisely analyzes the amount of FVC persists. The article proposes a technique to precisely estimate the amount of FVC in an area from images by drawing inferences from and extending upon the Daubenmire method, a semi-quantitative

ocular based FVC estimation technique. While current techniques estimate the percentage of FVC within images, the proposed technique provides an estimate on the area occupied by FVC in metric units in addition to percentage. The precise estimation of the FVC helps the plant scientists make discoveries of plant varieties that make good companions and offer well-informed suggestions to the agrarian community.

6.2 Related work

The development of vegetation cover rating scales started in the early 1900s. The most widely accepted scale at the time was the rating scale proposed by Braun-Blanchet [41]. The scale involved estimating and classifying the vegetation cover within a certain area into one of five cover classes from one to five where one indicated cover less than five percent, two indicated covers between five and twenty-five percent, three meant covers between 25% and 50%, four meant covers between 50% and 75%, and five meant cover between 75% and 100%.

RF Daubenmire [42] proposed the canopy coverage method in 1959. It was a semi-quantitative ocular based estimation technique and regarded as one of the most accurate methods for vegetation cover analyses over the years. It involved meticulously placing a 20- x 50- cm quadrat along a tape on permanently located transects and classifying the amount of vegetation cover into one of six cover classes. The cover classes are similar to Braun-Blanchet's classes with the difference being that cover class five was divided into two classes where cover class five indicated vegetation cover between 75% and 95% and class six indicated vegetation cover between 95% and 100%.

Booth, Cox and Berryman [43] proposed Sample Point, a free vegetation cover estimation tool from digital images using manual point sampling. The application shows users multiple single-pixel sample points (defaults to 100) on the image and allows them to classify the pixel as belonging to one of nine categories. The application uses the classifications made by the users to identify the percentage of pixels belonging to each of the nine categories. The results are output to an excel spreadsheet and have been comparable to the results from the most accurate field experiments for vegetation cover estimation.

Systat Software Inc. [44] provides specialized scientific software products for research in fields such as environmental sciences, life sciences and engineering. SigmaScan Pro 5.0, an image processing software tool developed by Systant Software Inc. may be used to estimate the percentage of pixels that belong to vegetation from digital imagery. While the tool was not developed for vegetation cover estimation, it is able to be adapted for the use case.

Patrignani and Ochsner [45, 46] developed a software tool named Canopeo to estimate the fractional green canopy cover from digital images. The tool was developed using Matlab and red-to-green (R/G), blue-to-green (B/G) and excess green index (2G-R-B). Desktop and mobile versions of the tool for android and iOS are available as free downloads. The tool provides a percentage estimate of the amount of green cover within an image and provides a grayscale image highlighting the green cover. However, the tool doesn't provide the vegetation cover estimate in metric units making it hard to reproduce results among experiments across individuals due to difference of height of image capture.

Louhaichi. M., Hassan, S., Clifton, K., & Johnson, D. E. [47, 48] proposed VegMeasure, a software tool that processes digital imagery collected in a specialized manner known as Digital Vegetative Charting Technique (DVCT). VegMeasure provides classification of imagery and measures change over time. However, DVCT requires a digital camera with built-in GPS that can be mounted to a stand, so images are captured from a fixed height with lens pointed orthogonal to the surface. VegMeasure requires that the camera, its height, and orientation be kept constant throughout the image capture process for its estimation and analysis.

Laliberte et. al [49] demonstrated that object-based image analysis upon the conversion of RGB scale images to IHS (intensity-hue-saturation) scale images is a viable approach for estimating total cover of vegetation, bare soil, and fractional components of green and senescent vegetation. The image analysis tool used for the experiments was eCognition where the image was segmented into homogenous areas based on three parameters: scale, color, and shape.

6.3 Materials and Methods

The breeders at the Land Institute in Salina, KS aim to estimate FVC across multiple plant varieties with the goal to identify companion plant species and make quality recommendations that may be made to the farming community. The design of the experiment is inspired by the Daubenmire canopy coverage method and visually depicted by Figure 6.1. Briefly, the Daubenmire technique involves using a 20 cm x 50 cm quadrat around vegetation cover along a tape on permanently located transects. The vegetation within the quadrat is measured by trained plant scientists based on ocular estimation. Since estimating the precise amount of plant cover from ocular observation is farfetched, plant scientists classify (estimate) the plant cover into one of six cover classes labelled one through six, as defined by Daubenmire.

Class One indicates crop cover less than five percent, class two indicates crop cover between five and twenty-five percent, class three indicates crop cover between 25% and 50%, class four indicates crop cover between 50% and 75%, class five indicates crop cover between 75% and 95%, and class six indicates crop cover between 95% and 100%. As part of the experiment, a PVC frame of known dimensionality is dropped across multiple rows of crops as shown in Figure 6.2(a) and the amount of vegetation cover within each of the segments of the PVC frame is estimated independently at regular intervals over a stipulated time frame. Since the experiment progresses over time, accurate readings at each time step are essential to estimate the growth of vegetation. An algorithm that leverages image processing and analysis is proposed to eliminate the need for ocular estimation of FVC and measure FVC from digital images with precision. The development of the algorithm is performed on images from two datasets where one of the datasets comprises 33 images that capture alfalfa at a resolution of 3024 x 4032 pixels and the other comprises 177 images that capture a one-segment PVC frame laid around plants of Kura at a resolution of 5184 x 3456 pixels, as shown in Figure 6.2. Broadly, the algorithm is a three-step process where the first and second steps are the detection and extraction of the PVC frame and vegetation cover from the image respectively. The final step is the estimation of the amount of vegetation cover within each of the segments of the PVC frame. The algorithm is developed in Python using the OpenCV image processing library.

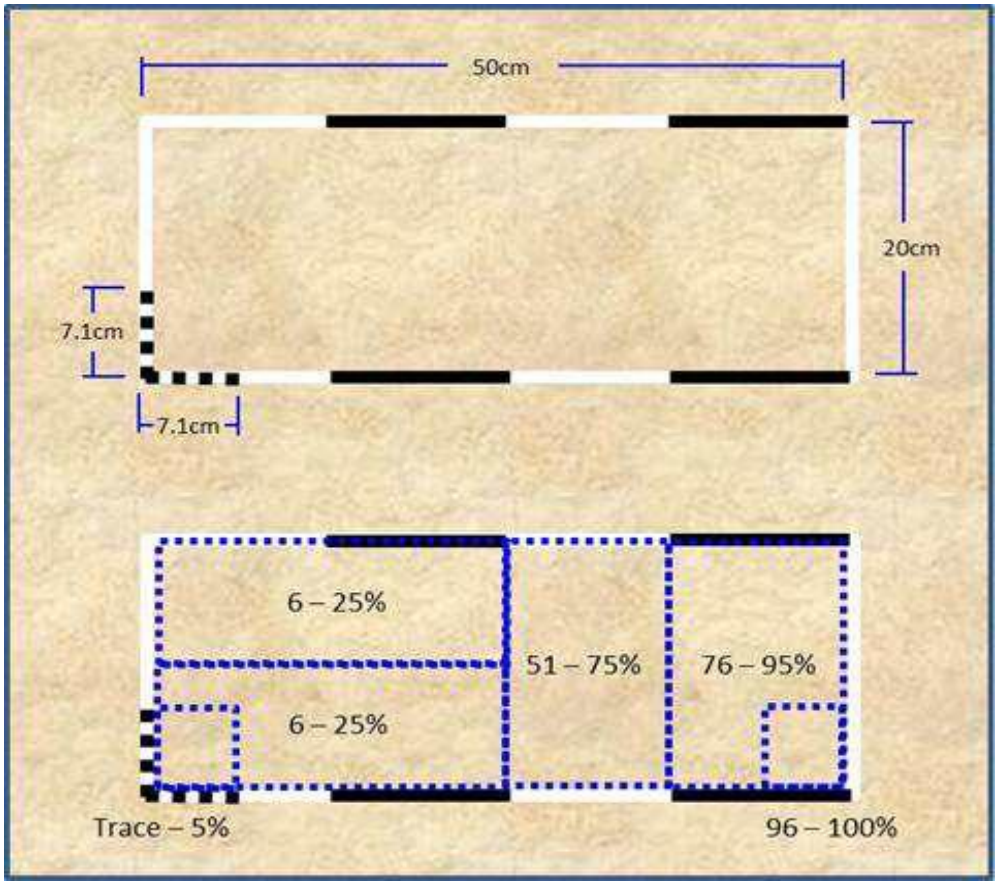

Figure 0.1. Daubenmire Quadrat with Ground Cover Classes [50]

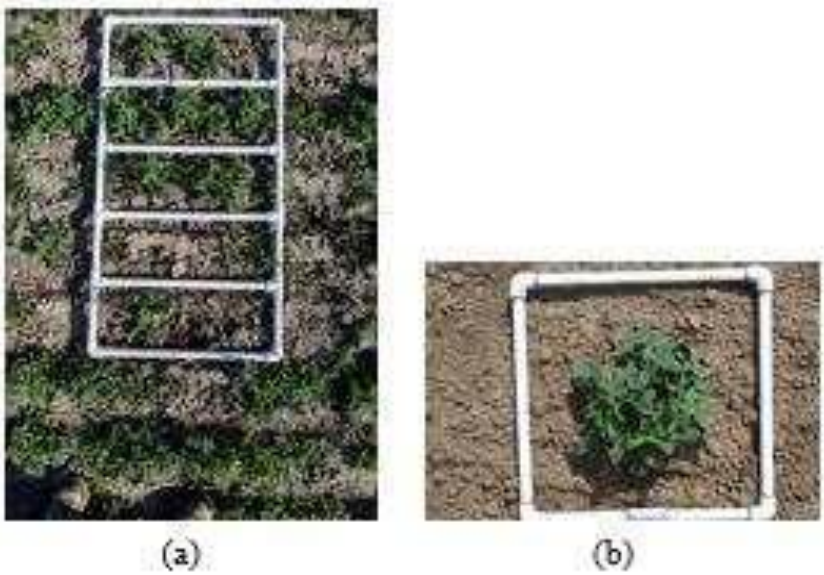

Figure 0.2. (a) Vegetation Cover within Five-Segment PVC Frame (b) Kura Plant within One-Segment PVC Frame

6.3.1 Noise Removal and PVC Frame Extraction from Image

In addition to vegetation cover within the PVC frame, the image consists of area outside the PVC frame. The area outside the PVC frame is irrelevant in terms of vegetation cover estimation within the grid. Hence, the first step of the process is the meticulous determination of the region of interest i.e. the PVC frame and pixels within the PVC frame. The image processing concepts that are relevant to the process are HSV color space, Perspective Transform and 'bitwise and' operation.

HSV Color Space: HSV refers to Hue, Saturation and Value which make up the co-ordinates for the color space. It is a cylindrical color space where the radius represents Saturation, the vertical axis represents value, and the angle represents Hue. Intuitively, Hue is the dominant color visible to an observer, Saturation is the amount of white light mixed with a hue and Value is the chromic notion of intensity. As Value decreases, the color gets closer to black whereas the intensity of the color increases as Value increases.

Perspective Transform: Perspective Transform corrects the perspective of an image to bird's eye or top view. In other words, it helps view the image as if it were captured with the camera held orthogonally to the surface. The application of the transform immensely helps reduce skew in images captured using freehand. The downside to images being skewed is that objects in skewed images appear to be of a different size and orientation in comparison to their true dimension. In the context of the current algorithm, the application of the perspective transform to the images helps reduce skew, thereby making the process of identifying the PVC frame easier.

Bitwise AND Operation: The ‘bitwise and’ operation is used to extract a specific region from an image using a mask. The mask is a grayscale image provided to indicate the region of the image required to be extracted. Typically, the mask's pixels are either white (255) or black (0). The original image is also converted to grayscale and all the regions of the image that are not a black (0) pixel are set to white (255). The ‘bitwise and’ operation operates on a pixel-by-pixel basis where it identifies the common regions between the two and sets them to white (255). The algorithmic steps to remove noise and extract the PVC frame are described as follows and the results produced after each step are shown in Figure 6.3.

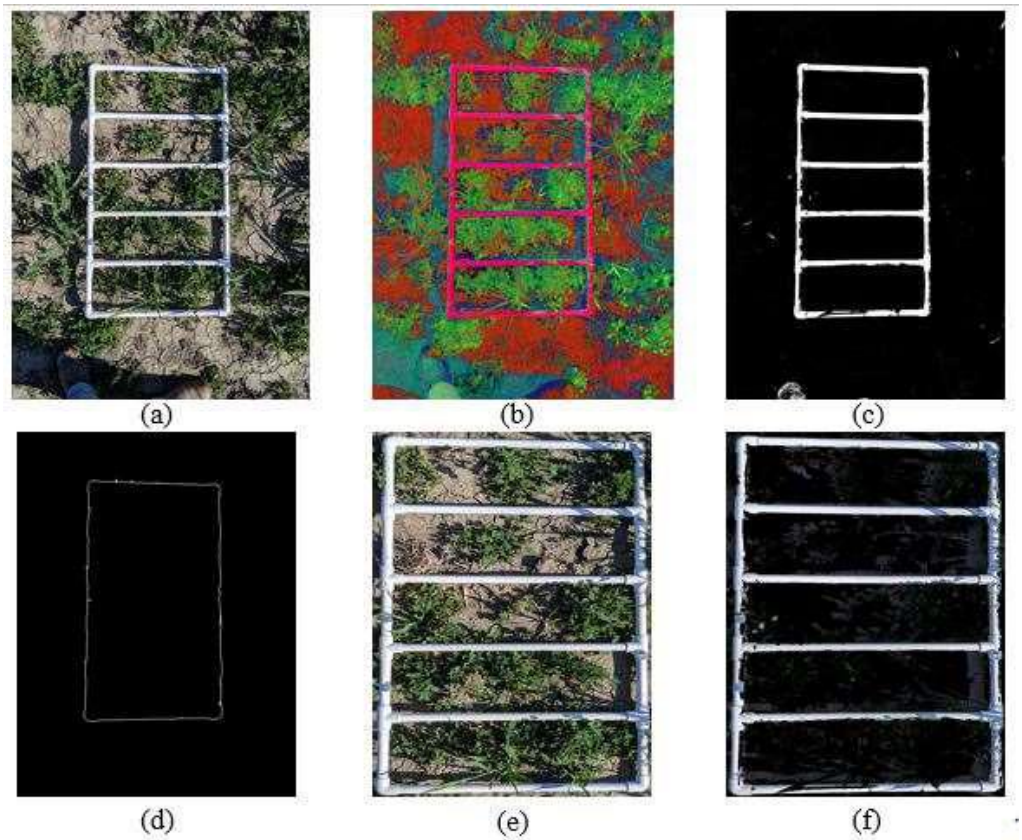

Figure 0.3. (a) Original Image (b) Original Image in HSV Color Scheme (c) Grayscale Image of PVC Frame (d) Contour of the PVC Frame (e) Original Image without Noise (f) Extracted PVC Frame from Original Image

1. Convert the original image in RGB color space to HSV color space and identify the pixels that belong to the PVC frame using the lower and upper ranges of (70, 0, 110) and (180, 255, 255) for (hue, saturation, value) respectively.

2. Convert the detected pixels of the PVC frame to grayscale and apply a median blur on the image to smoothen the image, if necessary.

3. Perform canny edge detection on the image and detect contours on the edges obtained.

4. Identify the largest contour of the contours detected. The idea is that the largest contour runs along the circumference of the PVC frame since the contours of each of the individual segments are significantly smaller in comparison to the outer contour which encapsulates the PVC frame.

5. Plot the minimum area rotated rectangle around the contour of the PVC frame.

6. Perform a ‘bitwise and’ operation on the input image using the minimum area rotated rectangle as the mask. Doing so precisely identifies the location of the PVC frame on the image. The region of the image within the PVC frame is the only portion of the image to be processed for vegetation cover estimation.

7. Apply perspective transform on the image to view the image from the top view. It corrects skew and makes the task of detecting segments on the PVC grid easier.

8. Convert the output of step 7 to HSV format and apply the lower and upper ranges mentioned in step 1 to precisely extract the PVC frame.

6.3.2 Vegetation Cover Detection using Simple Linear Iterative Clustering (SLIC)

On the region of interest identified in the previous step, SLIC is applied to segment the image into multiple superpixels. To elaborate, the process of partitioning an image into multiple segments by assigning a label to every pixel in the image is called image segmentation. The idea is that segments with the same label are perceptually similar and share common characteristics. Image segmentation helps to identify similar regions in images and extract semantic information from them. While segmenting images by pixel is an intuitive approach, segmenting by super-pixel is more optimal. A super-pixel is defined as a group of pixels that perceptually belong together by sharing common characteristics such as pixel color, intensity and other low-level properties. Modern cameras result in images of high resolution. While the level of detail within the images is excellent, it overburdens the image processing algorithms used to process the images resulting in longer runtimes. The key benefit to using super-pixels over pixels is that they shrink the pixel space required to be processed by the algorithm and in turn, lead to shorter runtimes without compromising accuracy. In addition, a single pixel by itself does not provide any semantic information whereas a superpixel provides semantic information that helps make key discoveries within images.

Simple Linear Iterative Clustering (SLIC): SLIC is an image processing technique that clusters pixels to generate super-pixels based on their color similarity and proximity in the image plane. SLIC uses the five-dimensional [LABXY] space for the clustering where [LAB] is the

pixel color vector in CIELAB color space and [XY] is the pixel position on the XY plane [51]. The CIELAB color space is defined by the International Commission on Illumination (CIE) to improve upon the fact that the conventional RGB color space only allows for the representation of 40% of the colors that the human eye can perceive. The L channel represents lightness which ranges from black to white, the A channel represents the value on the axis that ranges from green to red and the B channel represents the value on the axis that ranges from blue to yellow. A key characteristic of the CIELAB color space is that its dimensions are nonlinearly scaled in the sense that spatial distance between two colors corresponds to perceptual distance and is uniform across the color space. For instance, consider two pairs of different colors that appear to be equally similar such as blue and light blue, red and light red. When Euclidean distance is computed between the two pairs of colors in the CIELAB color space, it is observed the Euclidean distance between both pairs of colors is the same.

In order to use the Euclidean distance measure in the five-dimensional [LABXY] space, the spatial distance between two points is required to be normalized since it is subject to image size. Consider an image containing N pixels and let K be the number of super-pixels in the segmented image. The size of each super-pixel is N/K, and a super-pixel center is present at every grid interval $S = \sqrt{N/K}$. The center of each of the super-pixels $Ck = [Lk, Ak, Bk, Xk, Yk]$ where $k = [1, K]$ is chosen at regular grid intervals S. The spatial extent of each of the super-pixels is approximately $S2$ i.e. the area of a super pixel and the pixels that are associated with any super-pixel center lie within a $2S \times 2S$ area around the super-pixel center on the XY plane. Euclidean distances in CIELAB color space are perceptually meaningful for small distances but not large distances. The distance measure Ds used in SLIC is the aggregate of the LAB distance

and XY plane distance normalized by the grid interval S. The mathematical notations are as follows:

LAB distance, $Dlab = \sqrt{(lk - li)^2 + (ak - ai)^2 + (bk - bi)^2}$

XY plane distance, $Dxy = \sqrt{(xk - xi)^2 + (yk - yi)^2}$

SLIC distance measure, $Ds = Dlab + (m/S) * Dxy$ where m is the factor that controls compactness of the super-pixel. The greater the value of m, the more compact the cluster.

The SLIC algorithm is similar to any other clustering algorithm in the sense it begins by sampling K regularly spaced cluster centers and moving them to seed location corresponding to the lowest gradient position in a 3 x 3 neighborhood. Image gradients are computed as $G(x, y) = ||I(x + 1, y) - I(x - 1, y)||^2 + ||I(x, y + 1) - I(x, y - 1)||^2$ where I (x, y) is the LAB vector corresponding to the pixel at position (x, y) and ||. || is the L2 norm. Each pixel in the image is associated with the nearest cluster center whose search area overlaps the pixel. After all the pixels are associated with the nearest cluster center, a new center is computed as the average LABXY vector of all the pixels belonging to the cluster. The process is iteratively repeated until convergence.

The regions that indicate plant cover are identified by inferring from the semantics of the super-pixels put out by SLIC. The steps in the process are described as follows.

1. Segment the image into a pre-defined number of super pixels using the SLIC function from the OpenCV image processing library. It is observed that segmenting the image into 300 super-pixels identifies the plant cover best although the algorithm may execute faster at the

expense of accuracy if the image is segmented into fewer super-pixels. The image segmented into super-pixels is shown in Figure 6.4.(a).

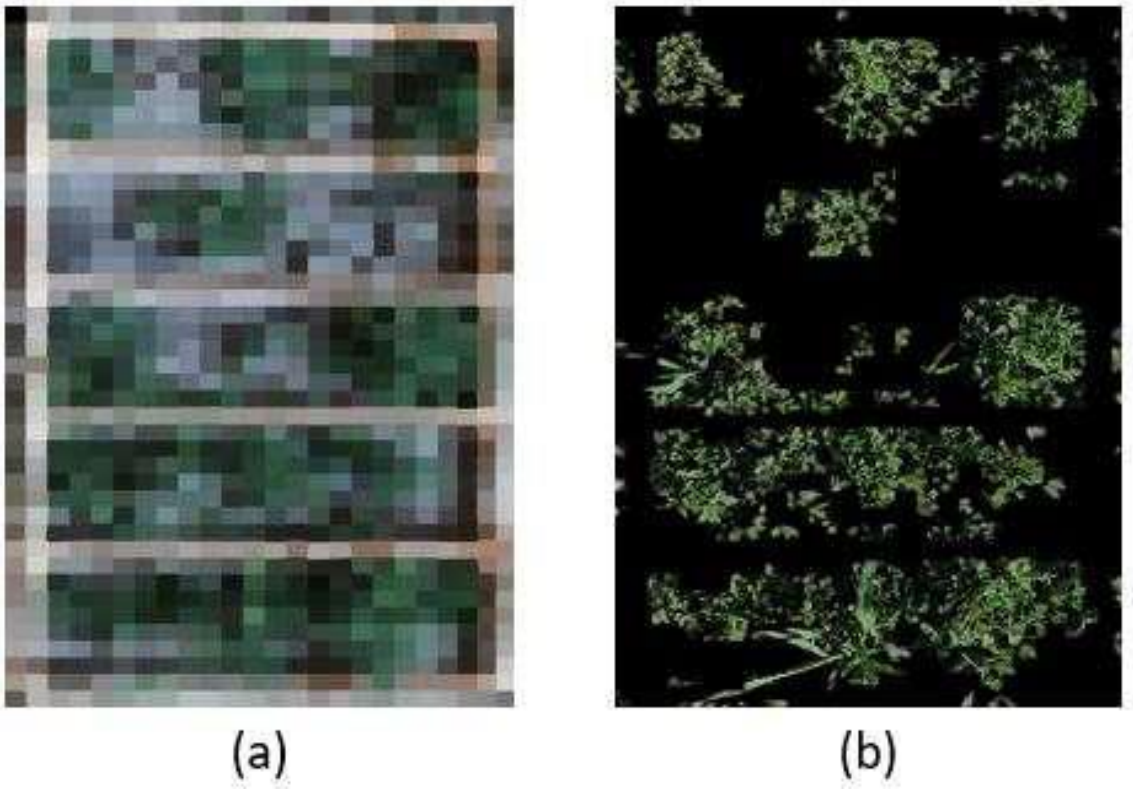

Figure 0.4. (a) Super-pixel Segmentation of Original Image (b) Detected Vegetation Cover in RGB Color Space

2. Find the average color value within each of the superpixels and assign it as the label that represents the super-pixel. It ensures that a given super-pixel has only one color.

3. Find unique labels within the image and apply a filter that identifies the plant cover. An observation that is made based on experimentation is that the super-pixels that consist of plants are green. Hence, all the green labels are identified, and a grayscale image is created

where the locations of green labels are plotted in white (255). The image represents the vegetation cover within the image.

4. (Optional) Perform a ‘bitwise and’ operation on the image using the grayscale image from step 3 as the mask. The resulting image shows the vegetation cover in the image within each of the segments of the PVC frame in the 3channel RGB color space. The step, whose output is as shown in Figure 6.3 (b), is merely to observe the vegetation detected within the image for user validation but is not required for analysis.

6.3.3 Vegetation Cover to PVC Frame Segment Association

Once the PVC frame and vegetation cover are identified, the next step in the process is to associate the vegetation cover with each segment of the PVC frame. The Hough Lines Transform is used to detect each of the segments of the PVC frame. The Hough Lines Transform is typically used to identify straight lines in images [5, 53]. Typically, straight lines are represented using the slope-intercept method as $y = mx + c$ where m is the slope and c is the y-intercept. However, Hough Transform relies on representing straight lines using the pair of polar co-ordinates, (ρ, θ). The first parameter, ρ, is the shortest distance from the origin to the line. The second parameter, θ, is the angle between the x-axis and distance line. The equation of the straight line in polar co-ordinate system is given by, $\rho = x * \cos(\theta) + y * \sin(\theta)$ where (x, y) is a point on the line. The Hough Transform takes the edges from a binary image as input. The Hough Transform maps each of the pixels to multiple points in Hough (or parameter) space. An edge pixel is mapped to a sinusoid in 2D parameter space, (ρ, θ) representing all possible lines that could pass through the point. It is referred to as the voting stage. The sinusoids of collinear points in the Hough space cross each other indicating collinearity. There are two variants of Hough Transform, Standard

and Probabilistic [54]. The primary difference between the two variants is the computational complexity. Consider an image that has M pixels as edge points and a Hough space divided in N_ρ x N_θ accumulators. In case of the Standard Hough Transform, each of the M edge pixels is used for computation which means that the computational complexity is $O(M * N_\theta)$ for the voting stage and $O(N_\rho * N_\theta)$ for the search stage. However, in the Probabilistic Hough Transform, only a subset m of M edge pixels is used for computation which means that the computational complexity is reduced to $O(M * N_\theta)$ for the voting stage resulting in a faster execution of the algorithm.

The algorithmic steps to associate vegetation cover with PVC frame is described as follows.

1. Consider the image of the PVC frame obtained as the output in section B.

2. Detect straight lines on the image using the Probabilistic Hough Lines transform. The detected line segments indicate the segments of the PVC frame. A vote measure of 180 is used to start the search for straight lines. In case no lines are identified at the vote range, it is lowered by 10 until lines are detected.

3. Filter the line segments to identify the ones that run along the segments of the PVC frame. Any given segment in the PVC frame should have four lines with slope ~0.0 i.e. horizontal or nearly horizontal lines and two lines with infinite slope i.e. vertical or nearly vertical lines associated with it. In order to detect the lines, parallel lines with slopes less than 15 and slopes higher than 100 are retrieved. Lines with slopes under 15 are oriented at less than 10 degrees with the X-axis whereas lines with

slopes above 100 degrees to infinity are oriented at over 45 degrees with the X-axis. It is possible that more lines than expected are detected along the PVC frame segment. In order to contain only the expected six lines, any overlapping line segments or ones that are closer in position and orientation than a certain threshold are merged together into a single line segment. All line segments that are separated by less than 10 pixels in distance and 0.2 degree in orientation are grouped together and only a single line segment from amongst the group is considered for analysis.

4. Extend the line segments along the PVC frame segments by a factor of five (or higher) to ensure that the horizontal line segments intersect with the vertical line segments in case they do not.

5. Determine the segment of the PVC frame to which each of the line segments belong based on the position of the line segments in the image. In the case of PVC frames that contain multiple segments, the association of line segments to the PVC frame segments helps estimate the amount of vegetation cover within the PVC frame segments.

6. Consider the line segments associated with each segment of the PVC frame and determine the innermost lines i.e. two inner vertical and two inner horizontal lines. Find the four points of intersection of the innermost lines. The idea is to identify the inner rectangles (polygons) within the PVC frame.

7. Create a grayscale image and plot a polygon using the points of intersection as the four vertices of the polygon. Fill the polygon in white (255), as shown in Figure 6.5 (b). In a PVC frame with n segments, n polygons are obtained where each of the polygons indicates a segment within the PVC frame.

8. Perform a 'bitwise and' operation on the output of section III(b), as shown in Figure 6.5 (b), using each of the grayscale images as the mask. The pixels indicating vegetation cover are plotted in white (255), as shown in Figure 6.5 (c).

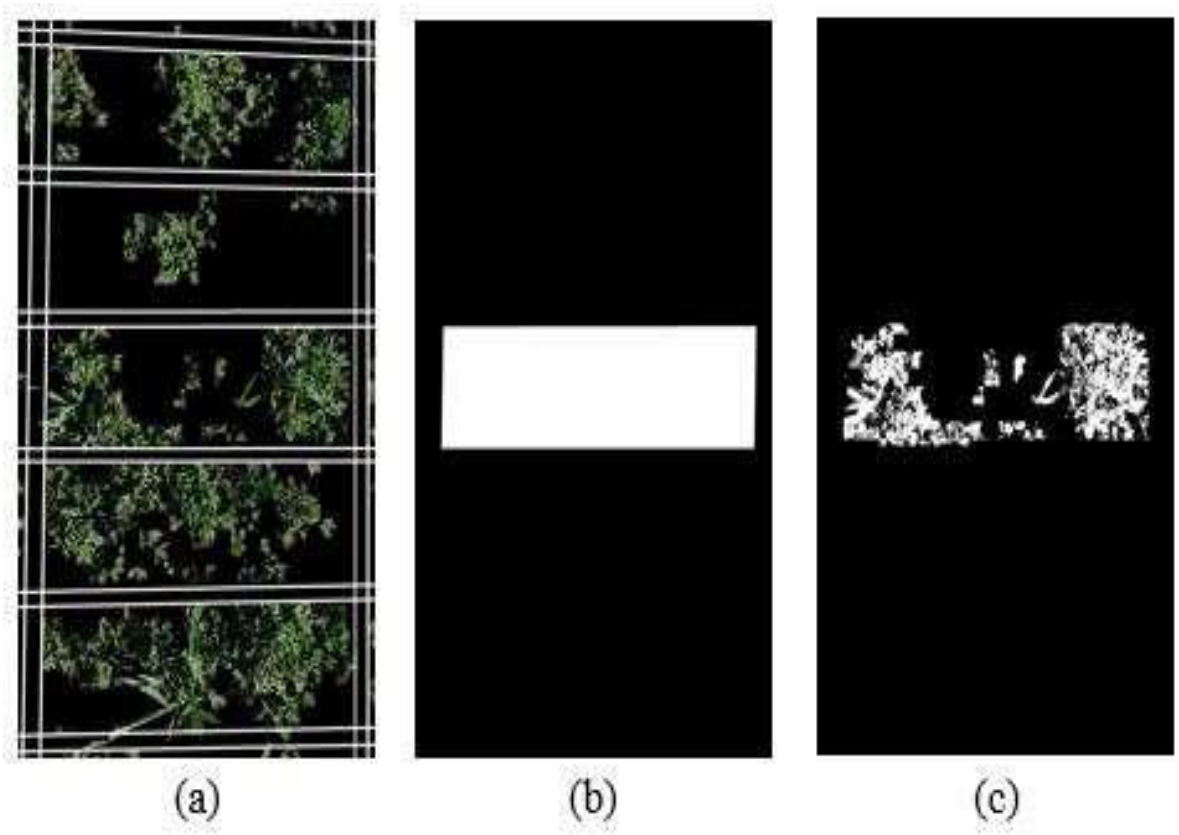

Figure 0.5. (a) Detected Hough Lines for the PVC Frame with Vegetation Cover (b) Polygon Mask for Segment 3 (c) Vegetation Cover within Segment 3

6.3.4 Amount of Vegetation Cover Estimation

The percentage of vegetation within each segment and the amount of vegetation cover in metric units are estimated by the algorithm. The ratio of the number of pixels that indicate vegetation cover, and the number of pixels present in each segment of the PVC frame gives the percentage of vegetation cover within each PVC frame segment. In other words, the ratio of white pixels in images obtained from step 7 and step 8 gives the percentage of vegetation cover. The estimation of vegetation cover in metric units requires the knowledge of dimensions of the PVC frame. The length and width of the inner rectangles of the PVC frame shown in Figure 6.2 (a) are 19.75" and 6.75" respectively. The number of pixels present within the rectangle are obtained from polygon masks plotted in step 7 of section III(C). Please note that each polygon might have a slightly different pixel count since the image may be partly skewed. Since the number of vegetation cover pixels are known from step 8 of section III(C), the area of vegetation cover is given by the equation, $Vegetation\ Cover\ (sq.\ in) = (area\ of\ rectangle\ (sq.\ in)\ x\ number\ of\ vegetation\ cover\ pixels)/\ (number\ of\ polygon\ pixels)$.

6.4 Result and discussion

In the absence of ground truth measurements for the vegetation cover within each of the PVC frames, the proposed algorithm is evaluated on two applications SamplePoint and Canopeo. SamplePoint is a desktop application that facilitates foliar cover measurements from nadir imagery by superimposing a systematic or random array of up to 225 crosshairs targeting single image pixels and provides a platform for simple, manual classification of the pixels. SamplePoint is a product that has been developed in collaboration with the USDA and serves as the benchmark for vegetation cover estimation. Canopeo is a tool available as desktop and mobile applications to estimate fractional vegetation canopy cover. Both SamplePoint and

Canopeo provide vegetation cover estimates as a percentage of the image size but not in metric units. It is perhaps due to the lack of a reference object in the image. The proposed algorithm is compared to the measurements provided by SamplePoint and Canopeo across 100 Kura plant images, similar to Figure 6.3 (b). The Cosine Similarity [55] measure is used to compare similarity among the three measures. Cosine Similarity is the cosine of the angle between two vectors that are typically non-zero and within an inner product space. Mathematically, it is defined as the division between the dot product of vectors and product of the magnitude of each vector and is expressed as, similarity = $A \bullet B / \|A\| \|B\|$ where A and B are the two vectors compared for similarity. A curve line of the estimates of vegetation cover across the 100 images using each of the three measures is shown in Figure 6.6 and Figure 6.7. The cosine similarity measure between the proposed algorithm and SamplePoint is 0.995 whereas that between the proposed algorithm and Canopeo is 0.99. It is worth noting that the cosine similarity between SamplePoint and Canopeo is 0.99. The results indicate a high degree of cadence among the techniques and demonstrate the quality of results produced by the proposed algorithm. As observed from the median and mean values, the results produced by the proposed algorithm lie between that of the results produced by SamplePoint and Canopeo. The median of the proposed algorithm is 12.04 whereas that of SamplePoint and Canopeo are 13 and 9.14 respectively. The mean of the proposed algorithm is 12.40 whereas that of SamplePoint and Canopeo are 14.21 and 10.01 respectively. The key benefit to using the proposed technique is that the algorithm provides the measurement of the amount of vegetation cover in metric units along with percentage. It significantly improves reproducibility of the results across multiple individuals conducting the experiments since it eliminates the dependence of the results on the height from which the image is captured.

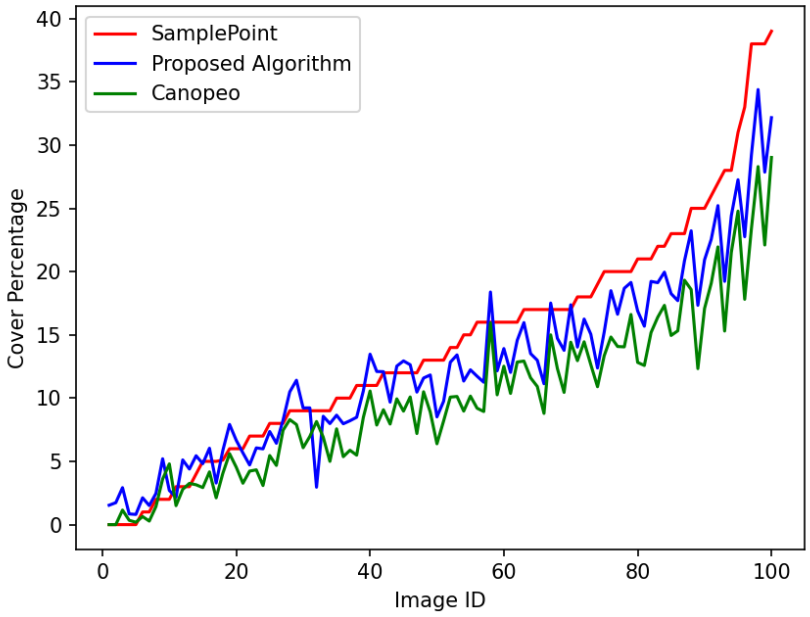

Figure 0.6. Result sorted by SamplePoint

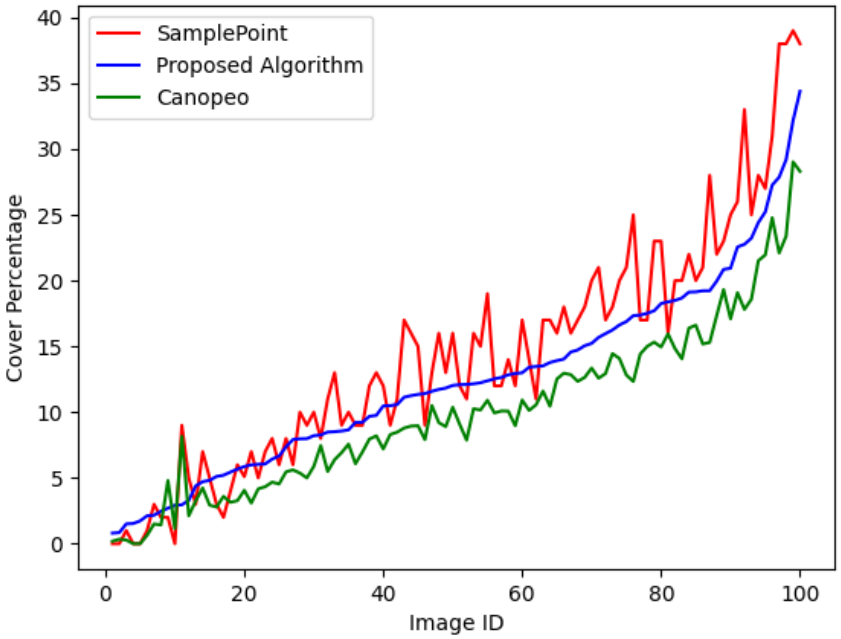

Figure 0.7. Result sorted by proposed algorithm.

Chapter 6 - Model Checking of Synchronization and Race Conditions

7.1 Mutual Inclusion and Mutual Exclusion Algorithms

In modern systems, many problems require multitasking – multiple tasks running at the same time. Multiple tasks (or processes) in a distributed system may access a resource simultaneously or execute a given function at the same time.

7.1.1 Instruction

The instructions used to update a shared resource are commonly referred to as a critical section (CS). If execution within each critical section is not controlled, more than one process may try to update a shared resource at the same time which may cause inconsistencies in the shared resource. Algorithms that guarantee at most one process is allowed in its CS at any given time are called mutual exclusion algorithms. Mutual exclusion is an example of concurrency control. Dijkstra first raised this question in 1965 [56]. In some cases, up to *k* processes can be executing in their critical sections simultaneously, this is called *k-exclusion*. In the simplest case, 1-exclusion is just mutual exclusion. Distributed mutual inclusion is complementary to distributed mutual exclusion. Mutual exclusion restricts processes entering their critical sections, while mutual inclusion restricts processes from leaving their CS. For 1-inclusion, also called mutual inclusion, at least one task must be in its CS. For instance, to provide higher system availability, we may require more than one server task to be available to process requests. Due to this complementary relationship, it is easier to speculate on algorithms to solve the inclusion problem based on similar principles after the problem of mutual exclusion is solved. Combining *k-exclusion* and *m-inclusion*, mutual exclusion and mutual inclusion may be generalized and

expressed as a *(k, m) – exclusion, inclusion* algorithm. For example, (1,0) – exclusion, inclusion is ordinary mutual exclusion. In a distributed environment, message passing is typically used to achieve mutual exclusion [57]. Distributed mutual exclusion algorithms can be divided into centralized algorithms and distributed algorithms. Distributed algorithms can be divided into token-based algorithms and permission-based algorithms [58]. This paper focuses on token-based distributed algorithms. Possession of a token allows a process to enter or leave its critical section. Different tokens are used for inclusion and exclusion.

UPPAAL is a toolbox for verification of real-time systems jointly developed by Uppsala University and Aalborg University. It has been applied successfully in case studies ranging from communication protocols to multimedia applications [60]. The tool is designed to verify systems that can be modeled as timed automata networks extended with integer variables, structured data types, and channel synchronization. The timed automata are finite state machines with real-valued clocks. It uses a dense time model in which the time variable is evaluated as a real number, and all clocks are synchronized. In UPPAAL, a system is simulated as a parallel network of several such timed automata. UPPAAL includes a simulator for random and guided simulation of the timed automata. A model checker is provided to verify safety and liveness properties. Properties are expressed in a subset of Computation Tree Logic (CTL). For an invariant property p, the expression A[] p means that on all paths (A), the property p is always ([]) satisfied. For a liveness property p, or to test for reachability, the expression E<> p means that on some path (E) the property p is eventually (<>) satisfied.

Several mutual exclusion algorithms and mutual inclusion algorithms form the basis for this work. The algorithms in the area of mutual exclusion are divided into two categories, token-

based and permission-based. The works [56, 58, 61, 64] are permission-based where a process is restricted from entering its critical section until permission is obtained from a quorum of processes. For token-based algorithms, such as [64, 66], a process is restricted from entering its critical section until a token within the scope of the system is obtained, giving the process the right to enter its critical section. For *k-exclusion*, a set of *k* tokens can be passed in a logical ring to enforce k-exclusion [63]. Thiare extended the algorithm to achieve self-stabilization [64]. The focus of this paper will be on token-based ring algorithms for *k-exclusion* and *m-inclusion*.

7.1.2 Token-based Ring Algorithms

Token-based ring algorithms are used on networks logically arranged as rings. The token in the network is the entity that controls concurrency in the network to ensure that the system is devoid of any race conditions, a condition where multiple tasks (or processes) try to access and modify the same shared resource. UPPAAL models are designed for both k-exclusion and m-inclusion.

Assumptions: Both algorithms rely on a common set of assumptions:

1. No loss of messages between different processes (also called nodes or tasks).
2. No delay in transmission of tokens between different processes (nodes).
3. Messages arrive in order.

7.1.2.1 Token Ring k-Exclusion Algorithm

The algorithm and correctness properties of the Token Ring *k-Exclusion* Algorithm [59, 61, 62, 64, 65] are listed below:

Algorithm:

1. All processes (nodes) form a logical ring structure. Tokens are passed between nodes in a clockwise (or counterclockwise) direction to a neighbor.

2. A node that receives a token has the right to enter its CS.

3. A token is transmitted to the next node after a node leaves its CS.

4. If a node does not need to enter its CS, any received token is passed directly to the next node without delay.

5. For *k-exclusion, k* exclusion tokens circulate in the ring.

Properties:

1. Safety: At any instant, at most k nodes can be in their critical sections.

2. Liveness: A node that wants to enter its critical section should eventually be allowed to enter.

3. Fairness: Each node gets a fair chance to execute in its CS. Fairness is only biased by the logical ordering imposed by the ring. The property means the CS execution requests are executed in the order of their arrival (a logical clock determines time) in the system.

7.1.2.2 Token Ring m-Inclusion Algorithm

The algorithm and correctness properties of the Token Ring *m-Inclusion* algorithm [59] are as follows:

Algorithm:

1. All processes (nodes) form a logical ring structure. The tokens pass between the nodes in a clockwise (or counterclockwise) direction to a neighbor.

2. A node that receives a token has the right to leave its CS.

3. A token is transmitted to the next node after a node enters its CS.

4. If a node does not need to leave its CS, any received token is passed directly to the next node without delay.

5. For *m-inclusion*, *m* inclusion tokens circulate in the ring.

Properties:

1. Safety: At any instant, at least m tasks must be executing in their critical sections.

2. Liveness: A task that wants to leave its CS should eventually be allowed to leave.

3. Fairness: Each process gets a fair chance to execute outside its CS. Fairness is only biased by the logical ordering imposed by the ring. This property means that requests to leave CS are executed in the order of their arrival (a logical clock determines time) in the system.

7.1.3 Model Checking and Analysis

7.1.3.1 *k-Exclusion* Model Checking

The distributed model is a ring network modeled in UPPAAL with five nodes where at most two nodes ($k = 2$) may simultaneously enter their critical sections. The variable ‘nb’ records neighbors of each node, and three channels to communicate with a passive observer used to verify properties. The variable ‘count’ records the number of nodes in the CS. The structure ‘S’ records the information at each node, including ‘requesting’, ‘numTokens’ and ‘inCS’. Initially, nodes 0 and 2 hold tokens. Each of the nodes starts in the Non-CS state. The model description in UPPAAL is as shown in Figure 7.1.

```
const int n = 5; // Number of tasks (nodes)
const int k = 2; // Number of tokens for k-exclusion

chan request[n];
urgent chan granted[n];
urgent chan token;

int[0,n] nb[n][2] = {{4,1},{0,2},{1,3},{2,4},{3,0}}; // Neighbors of node i, nb[i][0] = prev, nb[i][1] = next
int count = 0; // Number of nodes in CS

// Data struct to hold the state of each node.
struct{
    int[0,1] requesting;     // Requesting to enter CS
    int[0,k] numTokens;      // Number of tokens held
    int[0,1] inCS;           // In critical section (must hold at least one token)
} S[n] = {{0,1,0},{0,0,0},{0,1,0},{0,0,0},{0,0,0}};
```

Figure 6.1. Model Description for *k-Exclusion*

The model is defined using two templates: Node and Observer. The Node template provides the state machine that models the *k-exclusion* algorithm, as shown in Figure 7.2. In the Non-CS state, if a token reaches a node that does not request for one, the token is passed to the node's neighbor. However, if a token reaches a requesting node, the node may enter the CS. Whenever a node enters its CS, it sets variables S[id].inCS to 1 and S[id].requesting to 0. In the event a node in its CS receives another token, the token is passed to the neighboring node. The model implemented as part of the work restricts the amount of time that a node may spend in the CS to 10-time units. At the end of the 10-time units, the node leaves the CS, passes the token to the neighboring node and sets variable, S[id].inCS to 0. The count of the number of nodes in the CS is incremented when a node enters its CS, and it is decreased when a node leaves its CS.

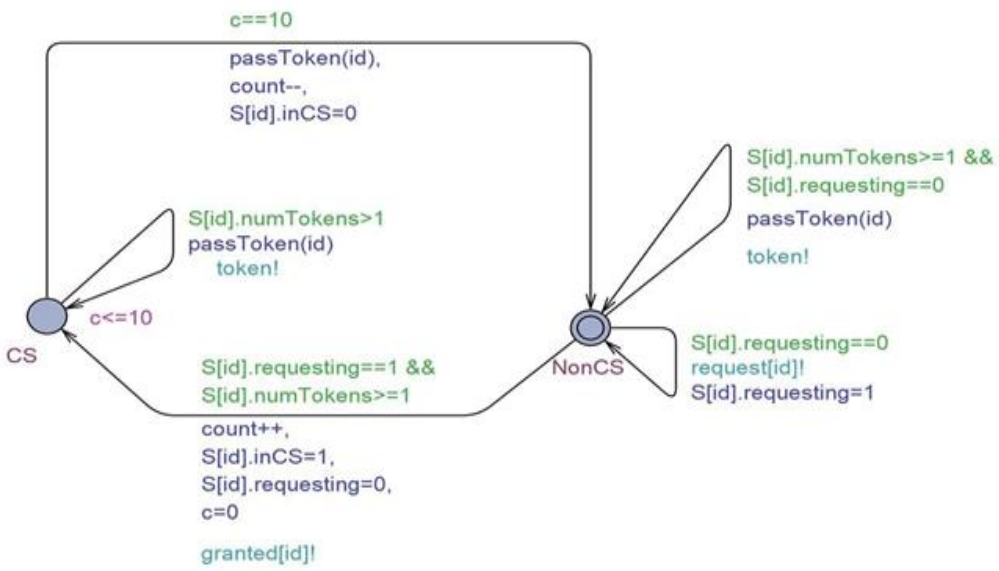

Figure 6.2. Model of Node for *k-Exclusion*

The Observer template models a process used to verify a liveness property of the model. It is used to verify that a node requesting access to a token receives the token in at most 40-time units. As shown in Figure 7.3, the observer may enter the 'BAD' state if a requesting node does not receive the token within 40-time units. Each node has its own observer, and each observer uses its own local clock x to time the delay between a node requesting and entering its CS.

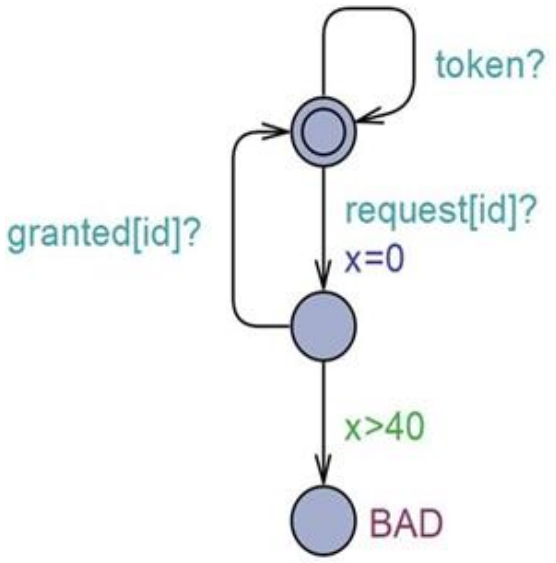

Figure 6.3. Model of Observer for *k-Exclusion*

Verification of Properties: The ETL for *k-Exclusion* is shown in Figure 7.4 and the justification for the verification of each of the properties is as described below.

1. A[] (not deadlock): No deadlocks in the system.
2. E<> (N2.CS && S[2].numTokens== 1): On some path, eventually node 2 may enter the CS
3. A[] (count <= *k*): At most *k* nodes may enter their CS at same time.
4. E<> (N2.CS && N1.CS): Two nodes can enter their CS, *k* = 2.
5. A[] (not Obs1.BAD): A node requesting to enter waits for at most 40-time units before entering the CS.

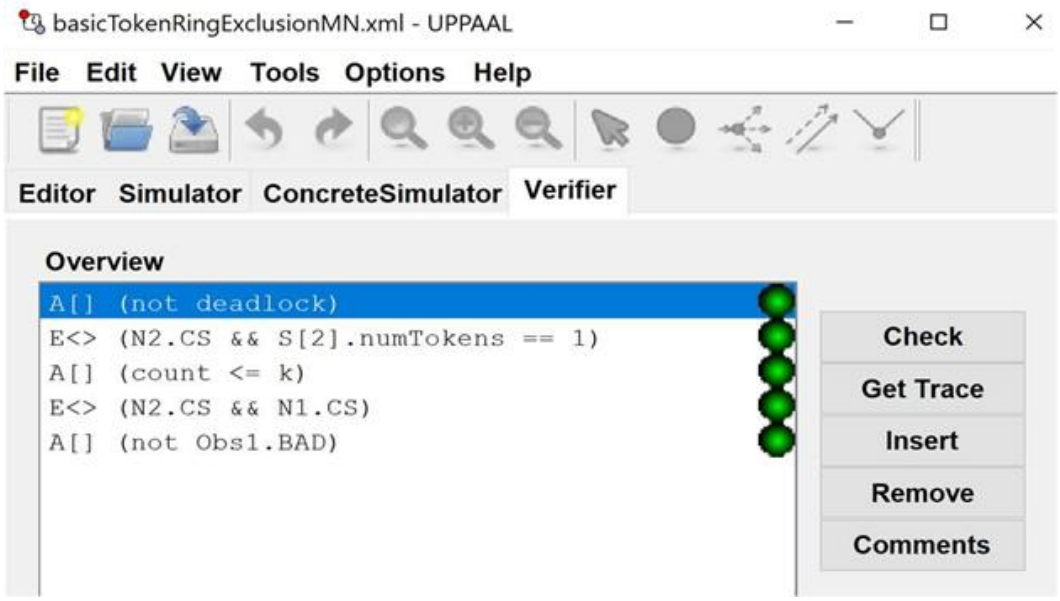

Figure 6.4. *k-Exclusion* Properties Verified in UPPAAL

7.1.3.2 *m-Inclusion* Model Checking

The idea of *m-inclusion* is that at least *m* tasks must be present in its critical section at any given point in time. It is fair to state that inclusion is complementary to exclusion in terms of behavior. As a result, the distributed UPPAAL model for inclusion closely follows that of the model for exclusion and is a ring network with five nodes where at least one node ($m = 1$) is required to execute in its critical section at any given point in time. In other words, at most four nodes may leave their critical sections simultaneously. Figure 7.5 shows the declarations for the UPPAAL model that achieves m-inclusion. The variable 'count' records the number of nodes out of the CS and variable 'outCS' tracks the status of the node. Initially, nodes 0, 1, 2 and 3 are initialized with tokens and each of the nodes starts in the CS state. Figure 7.6 shows the state machine that achieves m-inclusion. The state machine is similar to that of exclusions with the only difference being that the states 'CS' and 'Non-CS' are swapped. The observer to ensure that the model does not enter a bad state is the same as the observer used for *k-exclusion*. This

demonstrates that inclusion and exclusion are complementary to each other and lays the basis to achieving inclusion and exclusion in one model.

```
// Place global declarations here.
const int n = 5; // Number of tasks (nodes)
const int j = 4; // Number of tokens for m-inclusion = n-m (where m = 1)

chan request[n];
urgent chan granted[n];
urgent chan token;

// Neighbors of node i, nb[i][0] = prev, nb[i][1] = next
int[0,n] nb[n][2] = {{4,1},{0,2},{1,3},{2,4},{3,0}};
int count = 0; // Number of nodes in NonCS

// Data struct to hold the state of each node.
struct{
    int[0,1] requesting;     // Requesting to enter CS
    int[0,j] numTokens;      // Number of tokens held
    int[0,1] outCS;           // Leave critical section (must hold at least one token)
} S[n] = {{0,1,0},{0,1,0},{0,1,0},{0,1,0},{0,0,0}};
```

Figure 6.5. Declarations for *m-Inclusion* Distributed UPPAAL Model

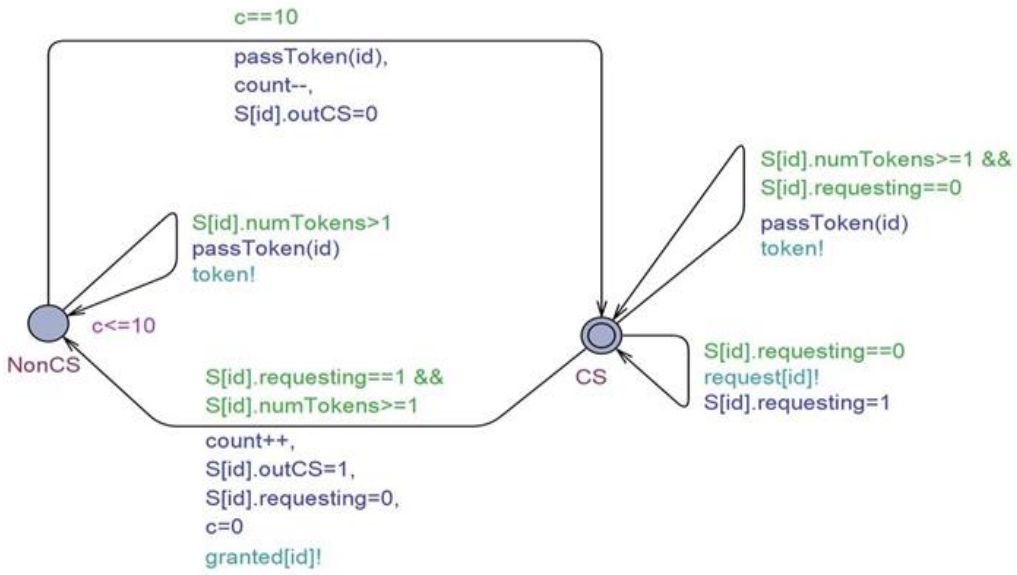

Figure 6.6. Model of Node for *m-Inclusion*

Verification of Properties: The ETL for *m-Inclusion* is shown in Figure 7.7 and the justification for the verification of each of the properties is as described below.

1. E<> (N2.CS && N1.CS): Eventually on some path, two (or j) nodes are allowed to enter their CS at the same time.

2. A[] (count <= j): On any given path, at most j nodes may enter the CS at the same time.

3. A[] (not obs1.BAD): On any given path, a node requesting to enter CS waits for at most forty time units before entering the CS.

4. A[] (not deadlock): On any given path, there are no deadlocks in the system.

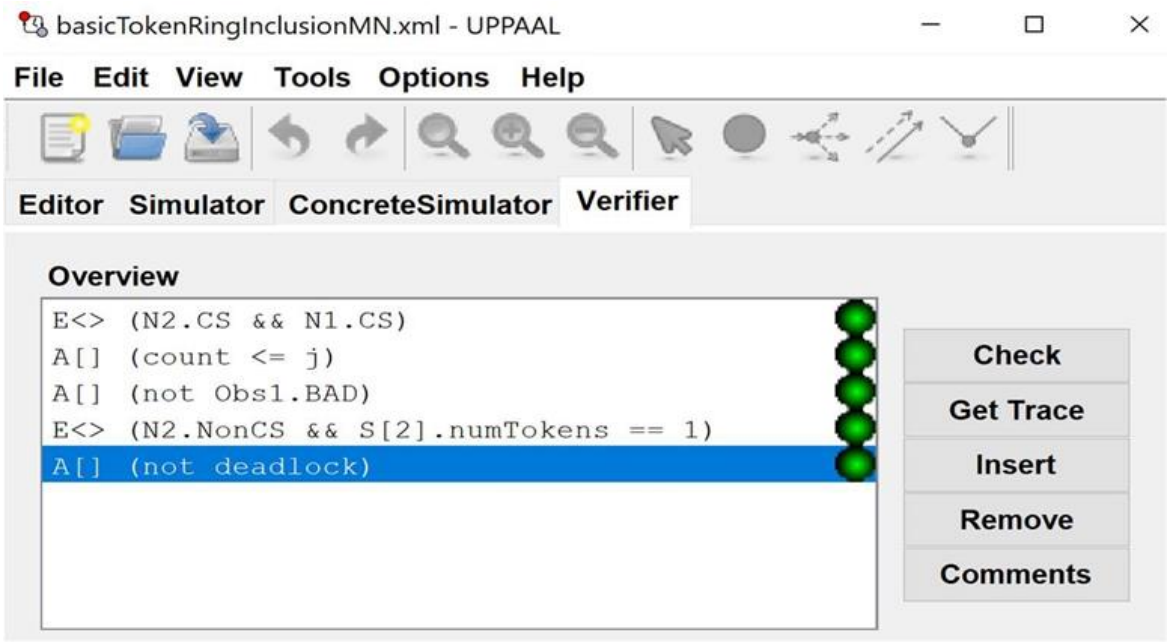

Figure 6.7. ETL for Verification in UPPAAL

7.1.3.3 (k, m)-Exclusion, Inclusion Model Checking

In k-exclusion, the token meant for exclusion is used to enter the CS. The idea of *m-inclusion* is similar to that of *k-exclusion* except that an inclusion token is acquired to leave the

CS. Figure 7.8 shows the declarations for *(3, 2)-Exclusion, Inclusion* where at least two nodes and at most three nodes may be in their CS. The variable n indicates the number of nodes, *k*, the number of tokens for *k-exclusion* and *m*, the number of tokens for *m-inclusion* equal to (n – m). Note that since we want two nodes to always be in their critical sections, we limit the number of inclusion tokens to n-m.

In the model as shown in Figure 7.9, two types of tokens are sent with calls to passToken() and passOutToken(). PassToken() passes tokens to allow a node to enter the CS and passOutToken() passes tokens to allow a node to leave the CS. The idea is that a node that wishes to enter the CS sets the variable, 'requesting', to 1. Upon receiving the 'enter' token as it passes through the network, the node enters the CS. Similarly, a node that wishes to exit the CS sets the variable, 'requestingOut', to 1 and exits the CS upon receiving the 'exit' token as it passes through the network. The 'requesting' and 'requestingOut' variables are set to 0 upon an entry and exit of the CS respectively.

```
// Place global declarations here.
const int n = 5; // Number of tasks (nodes)
const int k = 3; // Number of tokens for k-exclusion
const int j = 3; // Number of tokens for m-inclusion j = n - m

// Neighbors of node i, nb[i][0] = prev, nb[i][1] = next
int[0,n] nb[n][2] = {{4,1},{0,2},{1,3},{2,4},{3,0}};
int countIn = 2;  // Number of nodes in CS = m = 2
int countOut = 3; // Number of nodes out of CS = n - m = 3

// Data struct to hold the state of each node.
struct{
    int[0,1] requesting;     // Requesting to enter CS
    int[0,k] numTokens;      // Number of tokens held
    int[0,1] inCS;           // In critical section (must hold at least one in token)
    int[0,1] requestingOut;  // Requesting to leave CS
    int[0,j] numOutTokens;   // Number of out tokens held
    int[0,1] outCS;          // Out critical section (must hold at least one out token)
} S[n] = {{0,1,1,0,0,0},
          {0,1,0,0,1,1},
          {0,1,1,0,0,0},
          {0,0,0,0,1,1},
          {0,0,0,0,1,1}};
```

Figure 6.8. Model Description for *(3, 2)-Exclusion, Inclusion* with Five Nodes

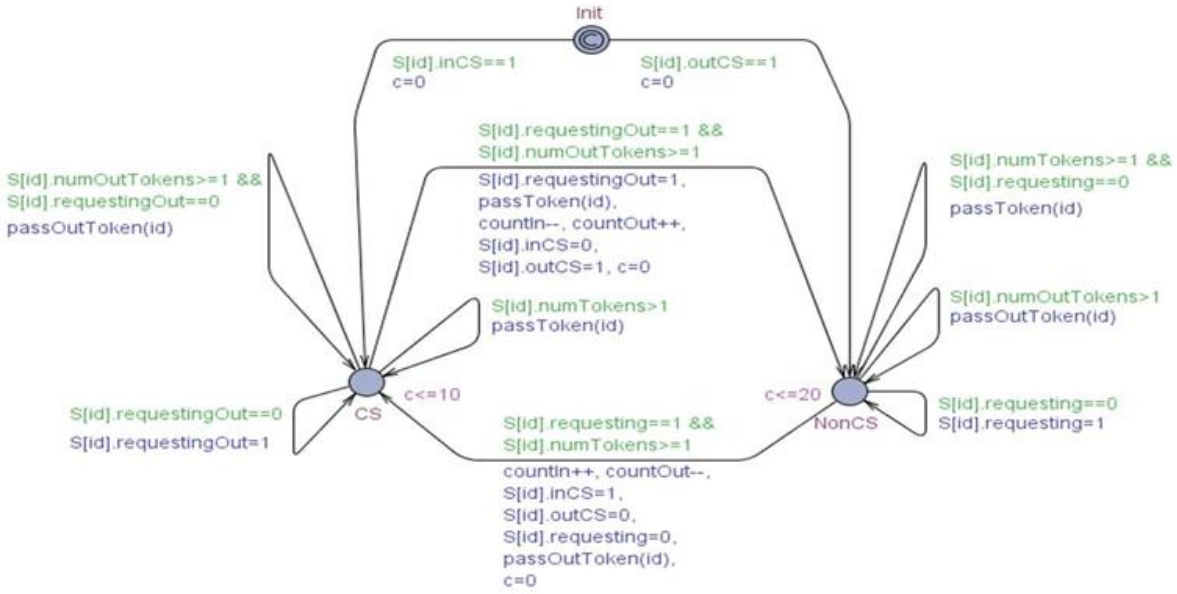

Figure 6.9. UPPAAL Model for *(k, m)-Exclusion, Inclusion*

Verification of Properties: The ETL for *(k, m)-Exclusion, Inclusion* is shown in Figure 7.10 and the justification for the verification of each of the properties is as described below.

1. E<> (N1.CS && N2.CS && N3.CS): Eventually on some path, the nodes N1, N2 and N3 are allowed to enter the CS at the same time.

2. A[] (countIn <= *k*) && (countIn >= *m*): Always on all paths, a minimum of *m* nodes and a maximum of *k* nodes are present in the CS.

3. E<> (countIn > *k*): Eventually on some path, more than *k* nodes are present in the CS. The property is not satisfied as verified by the verifier.

4. A[] (countIn + countOut == 5): Always on all paths, a node is either in CS or outside of CS but never in an intermediary state. Hence, the sum of countIn and countOut equals the number of nodes in the system.

5. A[] (not deadlock): Always on all paths, there are no deadlocks in the system.

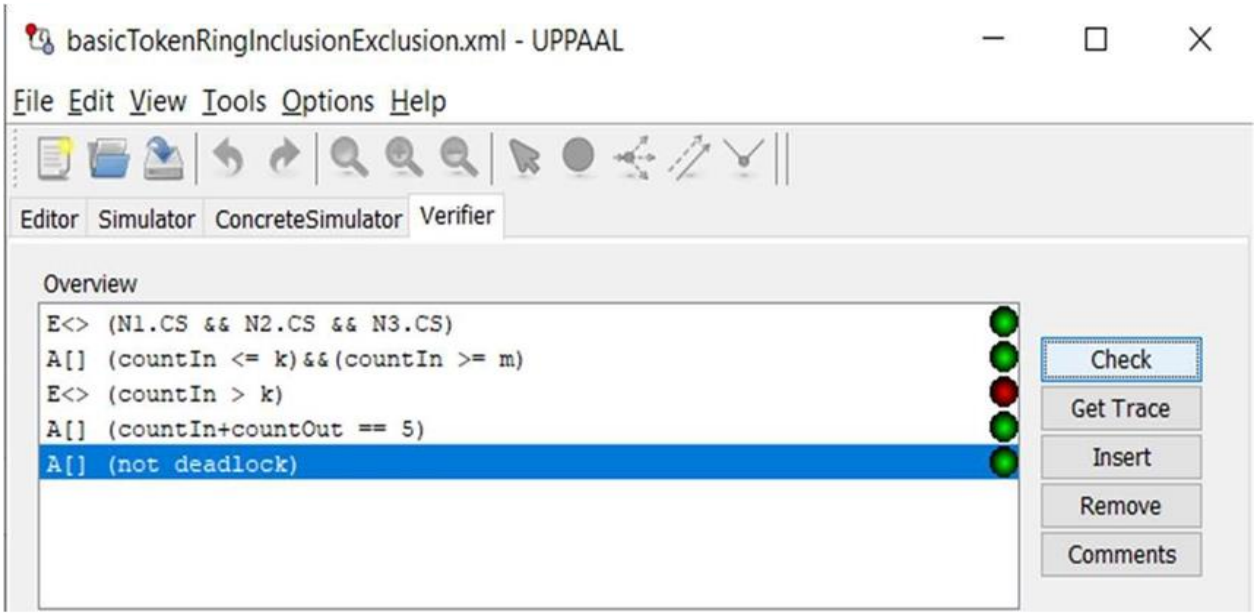

Figure 6.10. ETL for Verification in UPPAAL

7.2 Computer vision algorithms (Carving Algorithm)

To simulate the carving algorithm, we initialized a 3*3*3 matrix as the Voxel V and converted two 2D images into 3D projection views. The following declarations are illustrated in Figure 7.11.

mask []: record the voxels. Initial the default value of each voxel is 1.

view []: record the projections (2 images). Assume the target is in the center of the Voxel, shown in Figure 7.12.

pos []: record the status of voxel. Value 0 presents the voxel is in the background; 1 presents the voxel is in the target; 2 presents the voxel has been checked.

cost: record the running time. Assume project function needs 3, and check function needs 1.

volume: record the volume of target. It is 1 in our sample model.

```
// Place global declarations here.
int [0,1] mask[3][3][3] = {{{ 1,1,1 },{ 1,1,1 },{ 1,1,1 }},
                           {{ 1,1,1 },{ 1,1,1 },{ 1,1,1 }},
                           {{ 1,1,1 },{ 1,1,1 },{ 1,1,1 }}};
int [0,2] view[2][3][3][3] = {{{{ 0,0,0 },{ 0,0,0 },{ 0,0,0 }},
                               {{ 0,0,0 },{ 1,1,1 },{ 0,0,0 }},
                               {{ 0,0,0 },{ 0,0,0 },{ 0,0,0 }}},
                              {{{ 0,0,0 },{ 0,0,0 },{ 0,0,0 }},
                               {{ 1,0,0 },{ 0,1,0 },{ 0,0,1 }},
                               {{ 0,0,0 },{ 0,0,0 },{ 0,0,0 }}}};
typedef int[0,2] pos;
int cost = 0;       // the running time
int volume = 27;          // target volume, our sample is 1.
```

Figure 6.11. Declarations in Carving Algorithm Model

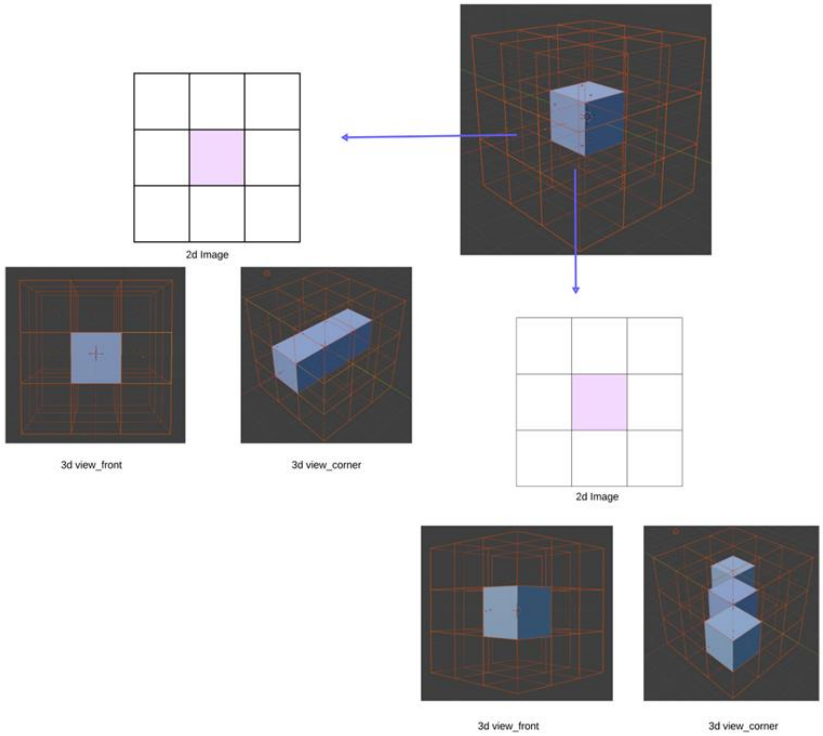

Figure 6.12. 3D projection of 1st view (Left); 3D projection of 2nd view (below)

7.2.1 Base Space Carving Algorithm

In the base space carving algorithm, every voxel must undergo projection and check value. Completing the process needs traversing the entire voxel space, comprising 27 voxels. For each image (projection), the cost amounts to 27 * (3 + 1) = 108. The volume decreases by one when the voxel value is 1 and the projection value is 0. This is depicted in Figure 7.13.

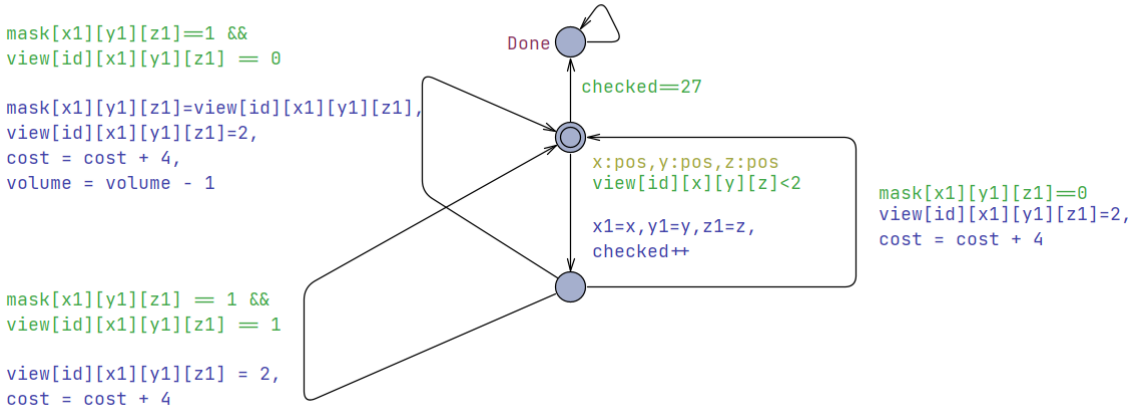

Figure 6.13. Base Space Carving Algorithm Model

We need to verify our model after simulating it, and the verifications and results shown in the following:

E<> (T1. Done and T2. Done) Satisfied.

E<> (volume == 1) Satisfied.

E<> (volume == 0) Unsatisfied.

E<> (T1. Done and T2. Done and cost <140) Unsatisfied.

E<> (T1. Done and T2. Done and volume! = 1) Unsatisfied.

7.2.2 Check Target Space Carving Algorithm

In the Check Target Space Carving Algorithm, we first execute the check function followed by the projection function. Specifically, the highest cost is calculated as 27 * (3 + 1) + (27 * 1 + 3 * 4) = 147, which is lower than that of the base carving algorithm. Shown in Figure 7.14.

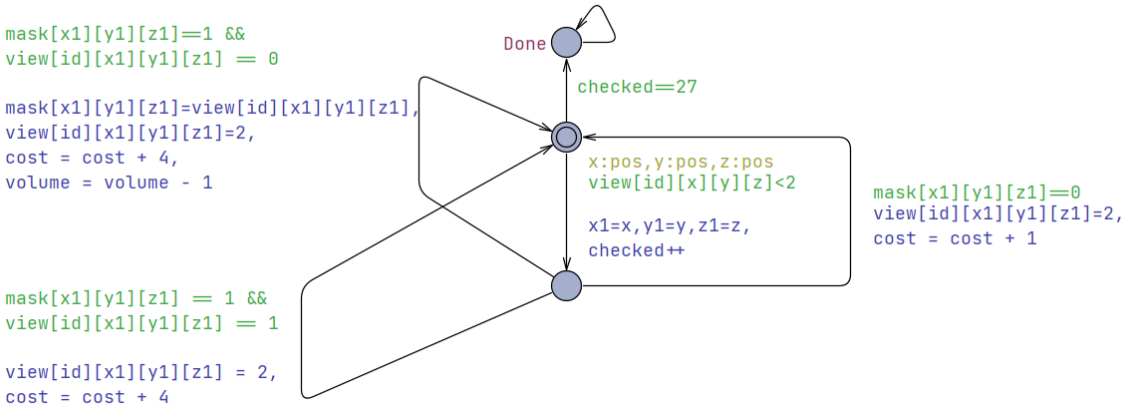

Figure 6.144. Check Target Space Carving Algorithm Model

The verifications are similar the base carving algorithm, but adding a new one:

E<> (T1. Done and T2. Done and cost <= 147) Satisfied.

7.2.3 Preprocess and Check Target Space Carving Algorithm

In the Preprocess and Check Target Space Carving Algorithm, we introduced a new variable, Flag, to indicate the second thread to begin processing. Consequently, the first thread handles the first image, while the second thread manages the second image. This results in a cost of 27 * (3 + 1) + 3 * 4 = 120, which is lower than the costs associated with the previous two algorithms. Shown in Figure 7.15.

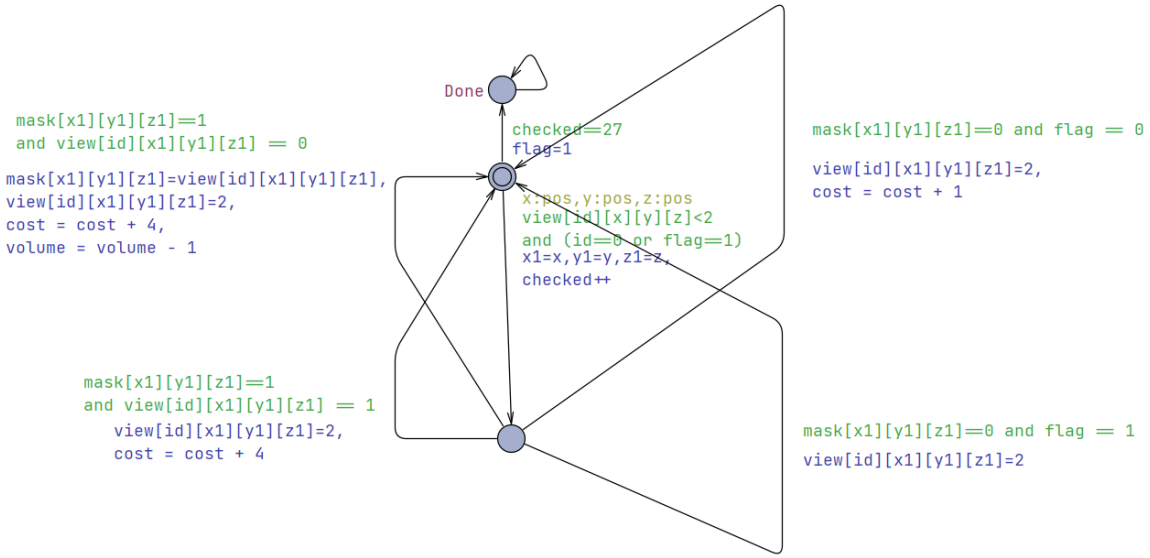

Figure 6.15. Preprocess and Check Target Space Carving Algorithm Model

The verifications are similar the base carving algorithm, but adding a new one:

E<> (T1. Done and T2. Done and cost == 120) Satisfied.

Chapter 7 - Conclusion and Future Work

To determine the volume of irregularly shaped seeds essential precisely and efficiently for contemporary breeding systems, we employed a 3D reconstruction methodology rooted in computer vision. Our novel modeling framework significantly enhances the efficacy of image acquisition while mitigating errors stemming from manual interventions. Concurrently, we conducted a comprehensive comparison of diverse image processing techniques, encompassing traditional HSV-based analysis and color thresholding, alongside machine learning approaches utilizing CNN. Through meticulous analysis and synthesis, we delineated the merits and demerits of each method. Furthermore, we juxtaposed and scrutinized 3D reconstruction methodology against alternative computer vision technologies, notably the slicing method, offering invaluable benchmarks for future research endeavors.

There is little room for improvement in the traditional contour detection method itself. Increasing the resolution of the camera only improves the accuracy a small amount. Traditional contour detection is sufficient for most seeds and has high accuracy and fast detection speed, especially with the LED background method. However, if seed researchers also need to know the color of the seeds, then methods that use Convolutional Neural Networks (CNNs) provide an advantage.

While the CNN models, HED and DexiNed, demonstrate impressive capabilities, they still require refinement when applied to a specific seed contour detection task. HED excels in detecting contours with minimal noise inside, but tends to generate overly thick and blurry contours, impacting accuracy. In contrast, DexiNed produces thin and clear contour lines but may result in some discontinuous contours and exhibit higher noise levels within them. These

limitations are likely due to the models' training datasets, which are primarily focused on edge detection rather than contour detection. Additionally, both models suffer from large model sizes. Our seed contour images are relatively simple. To address this, we intend to develop a smaller model that retains the essential characteristics of the HED and DexiNed models but tailored for a seed contour dataset.

In addition, the current algorithmic evaluation model, being inadequately scaled, fails to aptly demonstrate the efficiency of our refined algorithm, thereby compromising its persuasiveness. Scaling up to a larger model, such as a 7*7*7 configuration, would undoubtedly yield superior results and bolster the credibility of our findings.

References

[1] Whan, Alex P et al. “GrainScan: a Low Cost, Fast Method for Grain Size and Colour Measurements.” Plant methods 10.1 (2014): 23–23.

[2] Menzel MI, Tittmann S, Bühler J, Preis S, Wolters N, Jahnke S, Walter A, Chlubek A, Leon A, Hermes N, Offenhäuser A, Gilmer F, Blümler P, Schurr U, Krause HJ. Non-invasive determination of plant biomass with microwave resonators. Plant Cell Environ. 2009 Apr;32(4):368-79. doi: 10.1111/j.1365-3040.2009.01931.x. Epub 2009 Jan 2. PMID: 19143992.

[3] Golzarian, M. R., Frick, R. A., Rajendran, K., Berger, B., Roy, S., Tester, M., & Lun, D. S. (2011). Accurate inference of shoot biomass from high-throughput images of cereal plants. *Plant methods*, *7*(1), 1-11.

[4] Kumar, P., Huang, C., Cai, J., and Miklavcic, S. J. (2014). Root phenotyping by root tip detection and classification through statistical learning. *Plant Soil* 380, 193–209. doi: 10.1007/s11104-014-2071-3

[5] Walter, A., Silk, W. K., & Schurr, U. (2009). Environmental effects on spatial and temporal patterns of leaf and root growth. *Annual review of plant biology*, *60*, 279-304.

[6] Jansen, M., Gilmer, F., Biskup, B., Nagel, K. A., Rascher, U., Fischbach, A., et al. (2009). Simultaneous phenotyping of leaf growth and chlorophyll fluorescence via growscreen fluoro allows detection of stress tolerance in *Arabidopsis thaliana* and other rosette plants. *Funct. Plant Biol.* 36, 902–914. doi: 10.1071/FP09095

[7] Arvidsson, S., Pérez-Rodríguez, P., and Mueller-Roeber, B. (2011). A growth phenotyping pipeline for *Arabidopsis thaliana* integrating image analysis and rosette area modeling

[8] Costa, C., Antonucci, F., Pallottino, F., Aguzzi, J., Sun, D. W., Menesatti, P., et al. (2011). Shape analysis of agricultural products: a review of recent research advances and potential application to computer vision. *Food Bioproc. Technol.* 4, 673–692. doi: 10.1007/s11947-011-0556-0

[9] Monforte, A. J., Diaz, A., Caño-Delgado, A., and van der Knaap, E. (2014). The genetic basis of fruit morphology

[10] Fenner, M. (Ed.). (2000). *Seeds: the ecology of regeneration in plant communities*. Cabi.

[11] Saatkamp, A., Cochrane, A., Commander, L., Guja, L. K., Jimenez-Alfaro, B., Larson, J., ... & Walck, J. L. (2019). A research agenda for seed-trait functional ecology. *New Phytologist, 221*(4), 1764-1775.

[12] Sankaran, S., Wang, M., & Vandemark, G. J. (2016). Image-based rapid phenotyping of chickpeas seed size. *Engineering in agriculture, environment and food, 9*(1), 50-55.

[13] Koc, Ali Bulent. “Determination of Watermelon Volume Using Ellipsoid Approximation and Image Processing.” Postharvest biology and technology 45.3 (2007): 366–371.

[14] Riddle, D.F., 1979. Calculus and Analytic Geometry. Wadsworth Publishing Company, Inc. Belmont, CA, USA. pp. 506 – 507.

[15] Roussel, Johanna et al. “3D Surface Reconstruction of Plant Seeds by Volume Carving: Performance and Accuracies.” Frontiers in plant science 7 (2016): 745–745.

[16] C. Cao and M. Neilsen, "Efficient Seed Volume Measurement Framework," 2020 International Conference on Computational Science and Computational Intelligence (CSCI), 2020, pp. 1328-1334

[17] Kai Zhao, Venkat Markapuri, Mitchell Neilsen (2022). Automated Phenotyping of Single Seeds using a Novel Volume Sculpting Framework. In Proceedings of the 2022 ASABE Annual International Meeting. Paper No. 2200403, pp. 1-17. https://doi.org/10.13031/aim.202200403

[18] Pound, Michael P et al. “Automated Recovery of Three-Dimensional Models of Plant Shoots from Multiple Color Images.” Plant physiology (Bethesda) 166.4 (2014): 1688–1698.

[19] Yang, Myongkyoon, and Seong-In Cho. “High-Resolution 3D Crop Reconstruction and Automatic Analysis of Phenotyping Index Using Machine Learning.” Agriculture (Basel) 11.10 (2021): 1010–.

[20] Potmesil, Michael. “Generating Octree Models of 3D Objects from Their Silhouettes in a Sequence of Images.” Computer vision, graphics, and image processing 40.1 (1987): 1–29. Web.

[21] Sankaran, S., Wang, M., & Vandemark, G. J. (2016). Image-based rapid phenotyping of chickpeas seed size. *Engineering in agriculture, environment, and food*, *9*(1), 50-55.

[22] Golbach, F., Kootstra, G., Damjanovic, S., Otten, G., & van de Zedde, R. (2016). Validation of plant part measurements using a 3D reconstruction method suitable for high-throughput seedling phenotyping. *Machine Vision and Applications*, *27*(5), 663-680.

[23] Mitchell L. Neilsen, Kai Zhao (2023). Multi-threaded space carving for 3-D seed reconstruction. In Proceedings of the 29th International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 2101-2107, July 23-27, 2023.

[24] Venkat Margapuri, Trevor Rife, Chaney Courtney, Brandon Schlautman, Kai Zhao, Mitchell L. Neilsen (2022). Fractional Vegetation Cover Estimation using Hough Lines and Linear Iterative Clustering. In Proceedings of the IEEE 5th International Conference on Electronics Technology (ICET), pp. 1-7.
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10052996

[25] Kai Zhao, Venkat Margapuri and Mitchell Neilsen (2021). Model Checking Mutual Inclusion and Mutual Exclusion Algorithms. In Proceedings of the 30th International Conference on Software Engineering and Data Engineering. Volume 77, pp 60-69.

[26] Solomon, C.J.; Breckon, T.P. (2010). Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab. Wiley-Blackwell. doi:10.1002/9780470689776. ISBN 978-0-470-84473-1.

[27] Wilhelm Burger; Mark J. Burge (2007). Digital Image Processing: An Algorithmic Approach Using Java. Springer. ISBN 978-1-84628-379-6.

[28] Sun R, Lei T, Chen Q, Wang Z, Du X, Zhao W, Nandi AK. Survey of image edge detection. Frontiers in Signal Processing. 2022 Mar 9; 2:826967.

[29] https://en.wikipedia.org/wiki/HSL_and_HSVen

[30] Canny, J., A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[31] Xie, S., & Tu, Z. (2015). Holistically-Nested Edge Detection. *ArXiv*. /abs/1504.06375

[32] Soria, X., Sappa, A., Humanante, P., & Akbarinia, A. (2021). Dense Extreme Inception Network for Edge Detection. *ArXiv*. https://doi.org/10.1016/j.patcog.2023.109461

[33] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv*. /abs/1409.1556

[34] Chollet, F. (2016). Xception: Deep Learning with Depthwise Separable Convolutions. *ArXiv*. /abs/1610.02357

[35] Roussel, J., Geiger, F., Fischbach, A., Jahnke, S., & Scharr, H. (2016). 3D surface reconstruction of plant seeds by volume carving: Performance and accuracies. Frontiers in Plant Science, 7(June2016), 1–13. https://doi.org/10.3389/fpls.2016.00745

[36] Wikipedia. Point cloud. Retrieved from https://en.wikipedia.org/wiki/Point_cloud on Dec. 2, 2023.

[37] Colombe, S., Nepal, M. P., Browning, L. B., Miller, M. L., & White, P. T. (2018). Three Sister Crops: Understanding American Indian Agricultural Practices of Corn, Beans and Squash.

[38] Little, B. (2000). Companion planting in Australia. New Holland.

[39] Abbott, L. (2013). Vegetation Cover Introduction. Retrieved from: https://www.youtube.com/watch?v=TXV5DxEaSR4

[40] Jonasson, S. (1988). Evaluation of the point intercept method for the estimation of plant biomass. Oikos, 101-106.

[41] Bonham, C. D., Mergen, D. E., & Montoya, S. (2004). Plant cover estimation: a contiguous Daubenmire frame. Rangelands, 26(1), 1722.

[42] Bonham, C. D., Mergen, D. E., & Montoya, S. (2004). Plant cover estimation: a contiguous Daubenmire frame. Rangelands, 26(1), 1722.

[43] Booth, D. T., Cox, S. E., & Berryman, R. D. (2006). Point sampling digital imagery with ‘SamplePoint’. Environmental Monitoring and Assessment, 123(1), 97-108.

[44] Systat Software, Inc. Retrieved from: https://www.environmentalexpert.com/software/sigmascan-version-pro-50-automatically-imagesanalyze-software-560982

[45] Chhetri, M., & Fontanier, C. (2021). Use of Canopeo for Estimating Green Coverage of Bermudagrass during Postdormancy Regrowth. HortTechnology, 31(6), 817-819.

[46] Patrignani, A., & Ochsner, T. E. (2015). Canopeo: A powerful new tool for measuring fractional green canopy cover. Agronomy Journal, 107(6), 2312-2320.

[47] Louhaichi, M. (2019). VegMeasure® Software: User Report.

[48] Louhaichi, M., Hassan, S., Clifton, K., & Johnson, D. E. (2018). A reliable and non-destructive method for estimating forage shrub cover and biomass in arid environments using digital vegetation charting technique. Agroforestry Systems, 92(5), 1341-1352.

[49] Laliberte, A. S., Rango, A., Herrick, J. E., Fredrickson, E. L., & Burkett, L. (2007). An object-based image analysis approach for determining fractional cover of senescent and green vegetation with digital plot photography. Journal of Arid Environments, 69(1), 1-14.

[50] Bonham, C. D., Mergen, D. E., & Montoya, S. (2004). Plant cover estimation: a contiguous Daubenmire frame. Rangelands, 26(1), 1722.

[51] Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Süsstrunk, S. (2010). Slic superpixels (No. REP_WORK).

[52] Aggarwal, N., & Karl, W. C. (2006). Line detection in images through regularized Hough transform. IEEE transactions on image processing, 15(3), 582-591.

[53] Duda, R. O., & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. Communications of the ACM, 15(1), 11-15.

[54] Kiryati, N., Eldar, Y., & Bruckstein, A. M. (1991). A probabilistic Hough transform. Pattern recognition, 24(4), 303-316.

[55] Xia, P., Zhang, L., & Li, F. (2015). Learning similarity with cosine similarity ensemble. Information Sciences, 307, 39-52.

[56] Dijkstra, E. W. (1983). Solution of a problem in concurrent programming control. Communications of the ACM, 26(1), 21-22.

[57] Kshemkalyani, A. D., & Singhal, M. (2011). Distributed computing: principles, algorithms, and systems. Cambridge University Press

[58] Raynal, M. (1991). A simple taxonomy for distributed mutual exclusion algorithms. ACM SIGOPS Operating Systems Review, 25(2), 47-50.

[59] Kakugawa, H. (2015). Mutual inclusion in asynchronous message-passing distributed systems. Journal of Parallel and Distributed Computing, 77, 95-104.

[60] Cicirelli, F., Nigro, L., & Pupo, F. (2011). Modelling and Verification of Concurrent Programs Using UPPAAL. ECMS, 2011, 525-533.

[61] Neilsen, M. L. (2014). Real-Time Token-Based Mutual Exclusion Algorithms. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA) (p. 1). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp).

[62] Afek, Y., Kutten, S., & Yung, M. (1997). The local detection paradigm and its applications to self-stabilization. Theoretical Computer Science, 186(1-2), 199-229.

[63] Dijkstra, E. W. (1982). Self-stabilization in spite of distributed control. In Selected writings on computing: a personal perspective (pp. 41-46). Springer, New York, NY.

[64] Thiare, O. (2009). Several-Tokens Distributed Mutual Exclusion Algorithm in a Logical ring networks. In International Conference on Machine Learning and Computing (pp. 567-572).

[65] Wu, W., Cao, J., & Raynal, M. (2007, December). A dual-token-based fault tolerant mutual exclusion algorithm for manets. In International Conference on Mobile Ad-Hoc and Sensor Networks (pp. 572-583). Springer, Berlin, Heidelberg.

[66] Raymond, K. (1989). A tree-based algorithm for distributed mutual exclusion. ACM Transactions on Computer Systems (TOCS), 7(1).