Development of a Machine Vision-Based Yield Monitoring System for Vegetable Crops

by

Amanda Boatswain Jacques

Bachelor of Engineering, McGill University, 2016

Thesis submitted in partial fulfillment of the requirements for the

Degree of Master of Science

Department of Bioresource Engineering

Macdonald Campus of McGill University,

Montreal, Quebec, Canada

July 17th, 2019

Abstract

Crop yield estimation and mapping are important tools that can help growers efficiently use their available resources and have access to detailed representations of their farm. Technical advancements in computer and machine vision have improved the detection, quality assessment and yield estimation processes for crops including apples, citrus, mangoes, maize, figs and many other fruits. However, similar methods capable of exporting a detailed yield map for vegetable crops have yet to be fully developed. A machine vision-based yield monitor was designed to perform identification, size categorization and continuous counting of shallot onions in-situ during the harvesting process. The system is composed of a video logger and a global navigation satellite system (GNSS), coupled with computer software developed in Python. Computer vision analysis is performed within the tractor itself while an RGB camera positioned directly above the harvesting conveyor collects real time video data of the crops under natural sunlight conditions. Vegetables are segmented using Watershed segmentation, detected on the conveyor and then classified by size. Results showed that the system was able to correctly detect 62.6% of onions in a subsample of the dataset and resulted in a linear regression with a coefficient of determination (R^2) of 0.49 between true and estimated counts. The software was also evaluated on its ability to classify the onions into 3 size categories (small, medium and large). A total of 55.9% of 271 analyzed onions were correctly categorized, with the highest performance achieved in the large class (73.3%), followed by the small class (58.7%) and medium class (44.4%). Based on the obtained results, occasional occlusion of vegetables and inconsistent lighting conditions were the main factors that inhibited performance. Finally, these geotagged images were used to map the size distribution of the shallot onions on a small section of the onion field. Although further enhancements are envisioned for the prototype system to improve overall detection

and size classification, its modular and novel design allows it to be used to map a selection of crops including carrots, shallot onions, Chinese radish and lettuce crops. The system has the potential to benefit many producers of small vegetable crops by providing them with useful harvest information in real time that can significantly improve current harvesting logistics.

Keywords: Precision agriculture; yield estimation; machine vision; watershed segmentation; shape detection; shallot onions; size estimation.

Résumé

L'estimation et la cartographie du rendement des cultures sont des outils importants qui peuvent aider les producteurs agricoles. Ils améliorent la gestion de leurs ressources disponibles et donnent accès à des représentations détaillées de leur ferme. Le progrès technique en vision artificielle a amélioré le processus de détection, d'évaluation de la qualité et d'estimation du rendement des cultures, notamment pour les pommes, les agrumes, les mangues, le maïs, les figues et plusieurs autres fruits. Malgré cet avancement technologique, aucune méthode permettant d'exporter une carte détaillée du rendement des cultures de légumes n'a encore été pleinement développée.

Dans ce projet, un capteur de rendement basé sur la vision artificielle a été conçu pour effectuer l'identification, la catégorisation par taille et le recensement en continu des échalotes françaises in situ pendant le processus de récolte. Le système est composé d'un enregistreur vidéo et d'un système de positionnement par satellites, le tout jumelé d'un logiciel informatique développé en Python. D'une part, l'analyse de la vision par ordinateur est effectuée dans le tracteur, tandis qu'une caméra RGB positionnée directement au-dessus du convoyeur récolte les données vidéo des cultures. Les images sont recueillies en temps réel, grâce à la lumière naturelle du soleil. L'analyse informatique permet d'identifier les légumes sur le convoyeur et de les classer par grandeur.

Une méthode de segmentation par ligne de partage des eaux a été utilisée pour isoler les oignons dans les images. Dans un sous-échantillon de l'ensemble de données, le système était capable de détecter correctement 62,6% des oignons. Les résultats ont aussi montré que le système avait abouti à une régression linéaire avec un coefficient de détermination (R^2) de 0,49 entre les quantités réelles et estimées de légumes. Le logiciel a également été évalué sur sa capacité à classer les oignons en trois catégories de taille (petit, moyen et gros). Au total, 55,9% des 271 échalotes françaises analysées ont été

correctement classées. La meilleure prédiction a été obtenue pour la classe des gros oignons avec 73,3% des oignons correctement classifiés, suivi des petits (58,7%) et finalement des moyens (44,4%). D'après les résultats obtenus, l'obstruction occasionnelle de légumes et les conditions d'éclairage irréguliers ont été les principaux facteurs limitant la performance.

Les images géomarquées ont été utilisées pour cartographier la distribution des oignons, en fonction de leur taille, sur une petite partie du champ de légumes. Bien que l'amélioration des dispositifs de détection et de classification par taille soient envisagées pour ce prototype, sa conception modulaire et innovante lui permet déjà d'être utilisé pour cartographier les cultures suivantes: carottes, échalotes, radis chinois et laitue. Le système pourrait bénéficier à de nombreux producteurs de petites cultures de légumes en leur fournissant des informations utiles sur les récoltes, en temps réel. Ces informations sont susceptibles d'améliorer considérablement la logistique des récoltes.

Mots-clés: Agriculture de précision; estimation du rendement; vision artificielle; segmentation par ligne des eaux; détection de forme; échalotes françaises; estimation de la taille.

Dedication

This dissertation is dedicated to my mother, Sandra Boatswain. My mother has worked hard to raise her children primarily on her own, while facing the challenges of her turbulent and busy life. I am forever grateful for the sacrifices and unconditional love she has provided to each one of my siblings and me. She has been my mentor, my confidante, my best friend and forever a source of inspiration and admiration.

Acknowledgements

I would first like to extend my most profound gratitude to Delfland, Inc. and its team for their financial and technical support throughout the entire duration of this project. Their assistance with the assembly and mounting of the yield mapping system truly was invaluable. I would like to give special thanks to Guillaume Cloutier for his insight, encouragement and patience, and allowing me to test the system on the company's fields.

I would like to thank Mitacs Canada for their generous financial support throughout this project in partnership with Delfland, Inc.

I would like to thank my colleagues in the Precision Agriculture and Sensor Systems (PASS) research team for their great help, support and inspiration. Special thanks are due to both Connor Miller and Maxime Leclerc for their help designing and machining various components of the system, and to Roberto Mario Buelvas for assistance with sensor setup and troubleshooting. I would also like to thank Md Saifuzzaman and Dr. Jaesung Park for their help cleaning and processing the geographic data reported in this paper. Thank you as well to Marie-Christine Marmette for her encouragement and optimism when I encountered occasional hurdles during the redaction of this dissertation.

Thank you to Trevor Stanhope and Bharath Sudarsan, past members of the PASS research team, who took the time to answer my questions concerning machine vision and valuable advice.

Thank you to Dr. James J. Clark, for providing his advice and expertise in computer vision when needed.

Thank you to my family: my mother, my father, my sisters, my brothers and my aunts, who encouraged me and cheered me on every step of the way of this project. To my boyfriend, for his undying support and wisdom.

Finally, I would like to express great gratitude and recognition to Dr. Viacheslav Adamchuk for his supervision and guidance. His faith in me never faltered even when I believed I was about to hit rock bottom. I am forever grateful for all the wonderful opportunities he provided by allowing me to complete this Master of Science (publishing papers, travelling to conferences, meeting exceptional scientists and presenting my research at multiple occasions, being president of the robotics club under his supervision) and pursue knowledge in one of the most exciting and novel fields of computer science and robotics. I would like to thank him for his patience and undying enthusiasm, and for the wonderful and hardworking environment that he has maintained in our research team.

Contributions of the Authors

The research in this thesis has been submitted for publication in two conference proceedings (2017 ASABE Annual International Meeting and 2018 ICPA). The author of this thesis was responsible for the development of an algorithm for detecting shallot onions using machine vision, and the integration of this algorithm in a full-fledged system to be used in the field. The author also designed and carried out the experimental and analytical work to meet the research objectives of this thesis and was responsible for the preparation of the manuscript summarizing this research. Dr. Viacheslav Adamchuk, a professor in the Department of Bioresource Engineering of McGill University, acted as the thesis supervisor. He presented the research idea and continuously offered scientific advice and technical guidance during the study. He is also responsible for editing and reviewing the manuscript before final publication. Guillaume Cloutier was a representative of the partner organization (Delfland, Inc.) funding this research and gave permission for testing the system on the field. Dr. James J. Clark, who is a McGill professor in the department of Electrical and Computer Engineering, provided scientific advice to the author regarding the development of the computer vision algorithm. He also was a member of the author's graduate committee and helped with progress assessment. Maxime Leclerc and Connor Miller provided technical assistance with machining components of the system and fabrication of the machine vision system mounting bracket.

Publications related to this thesis:

1. **Boatswain Jacques, A. A.**, Adamchuk, V. I., Cloutier, G., Clark, J. J., & Miller, C. (2018). Development of a Machine Vision Yield Monitor for Shallot Onion Harvesters. In: *Proceedings of the 14th International Conference on Precision Agriculture*, Montreal, Quebec, 24-27 June 2018. International Society of Precision Agriculture (published online at https://ispag.org/proceedings/?action=download&item=5401, 13 pages).

2. **Boatswain Jacques, A. A**., Adamchuk, V. I., Cloutier, G., Clark, J. J., & Leclerc, M. (2017*).* A Machine Vision Yield Monitor for Vegetable Crops. In: *Proceedings of the 2017 American Society of Agricultural and Biological Engineers Annual International Meeting*, Spokane, Washington, 17-19 July 2017. *American Society of Agricultural and Biological Engineers* (published online at https://elibrary.asabe.org/azdez.asp?JID=5&AID=50188&CID=spo2017&T=1&redirType=techpapers.asp&confid=spo2017/, 10 pages).

Format of Thesis

This dissertation is partially a reformatting of two conference papers that have been prepared for publication. Following the general introduction in **Chapter 1** and the literature review in **Chapter 2**, **Chapter 3** describes the development of a machine vision algorithm for shallot onion detection and the implementation of this algorithm in a system that performs yield mapping. **Chapter 4** presents the results of a feasibility study as well as the results obtained from field testing. Following this, a discussion section critiques the findings of this research and offers future improvements. General conclusions (**0**), references and appendices of supplemental materials complete this thesis.

Table of Contents

Abstract ii
Résumé iv
Dedication vi
Acknowledgements vii
Contributions of the Authors ix
Format of Thesis xi
Table of Contents xii
List of Tables xiv
List of Figures xv
List of Acronyms xvii

Chapter 1. Introduction 1
1.1. Yield Monitoring 1
1.2. Computer Vision 3
1.3. Research Objective 4

Chapter 2. Literature Review 5
2.1. Digital Imagery 5
2.1.1. Computer Vision and Machine Vision 5
2.1.2. Camera Systems 6
2.2. Applications of Computer Vision in Agriculture 9
2.3. Machine Vision for Yield Determination 11
2.4. Challenges of Machine Vision Applications 14
2.5. Summary of Literature Review 15

Chapter 3. Materials and Methods 17
3.1. Feasibility Study 17
3.1.1. Image Acquisition 17
3.1.2. Software 18
Software Structure 18
Initial Algorithm 20
3.1.3. Distortion Correction 23
3.1.4. Conveyor Speed Calculation 25
3.1.5. Processing Unit 25
3.2. Prototype System Design 26
3.2.1. System Components 26
3.2.2. Mounting Bracket Design 27
3.2.3. Electrical System Design 27
3.2.4. Segmentation 30
3.2.5. Definition of Vegetable Size Categories 32
3.2.6. Size Calibration 33
3.2.7. Statistical Analysis 34

Chapter 4. Results and Discussion 35
4.1. General System Performance 35
4.1.1. Integrability 35
4.1.2. System Cost 36
4.1.3. System Assembly 36
4.1.4. Reliability 37
4.2. Results of Feasibility Study 37
4.2.1. Segmentation Results 37
4.2.2. Onion Detection Performance 39
4.2.1. Conclusions of Feasibility Study 42
4.3. Prototype Performance 43
4.3.1. Size Estimation 43
4.3.2. Segmentation Results 48
4.3.3. Onion Detection Results 49
4.3.4. Yield Map 55
4.4. Future Improvements 57

Chapter 5. Conclusions 59

References 61

Appendix A: Python Code 65
A-1 Initial Version of Python Code (Feasibility Study) 65
A-2 Final Version of Python Code (Field Trial) 67
Yield Monitor Class 67
Image Preprocessing 71
Config File Loading 76
GPS Sentence Parsing 76
Main File 77
Image Preprocessing (Updated post field trials) 78
Statistical Analysis 83

Appendix B: Hardware Specifications 84

Appendix C: Additional Figures 86

Appendix D: Definition of Performance Metrics 88

Appendix E: Additional Data 88
E-1 Size Prediction Results 90
E-2 Median-Filtered Onion Count Predictions 96

List of Tables

Table 3.1. Intrinsic camera parameters of the Nikon KeyMission 170. 24
Table 3.2. Computer specifications 28
Table 3.3. Shallot onion classes defined for the computer vision algorithm. 33
Table 4.1. Price Breakdown of the machine vision yield monitor by component. 36
Table 4.2. Kolmogorov-Smirnov test results for the small class 47
Table 4.3. Kolmogorov-Smirnov test results for the medium class 47
Table 4.4. Kolmogorov-Smirnov test results for the large class 47
Table 4.5. Summary of shallot onion detection results 50
Table 4.6. Summary of Type I and Type II error distribution for the Initial Method. 54
Table 4.7. Summary of Type I and Type II error distribution for the Watershed Method. 54
Table 4.8. Summary of detection performance metrics 54
Table B-1. Prototype Camera specifications.....84
Table B-2. Solid state drive specifications 84
Table B-3. GPS Sensor specifications.....85
Table E-1.1. Size prediction results for the small onion class.....90
Table E-1.2. Size prediction results for the medium onion class.....93
Table E-1.3. Size prediction results for the large onion class.....95
Table E-2.1. Size prediction results and positioning for all onions.....96

List of Figures

Figure 1.1. Shallot onion harvesting machine with trailer (a) and onion field (b).....4

Figure 2.1. Images of a CCD sensor (a) (Ahmed2IQ, 2009) and a CMOS sensor (b) (Nyman, 2012). 7

Figure 2.2. An example of feature intensification using a date fruit. Original image (a), thresholded image (b) and edge image (c). Reprinted from Al-Ohali (2011). 11

Figure 2.3. Apple yield map of an orchard block created using a computer vision algorithm. Reprinted from Bargoti & Underwood (2017). See electronic version for colours. 13

Figure 2.4. Result of splitting a region representing two different apples. Reprinted from Wang et al. (2012).....15

Figure 3.1. Nikon KeyMission 170 action camera (Nikon, 2018). 17

Figure 3.2. Three views considered for the placement of the camera. 18

Figure 3.3. Software Flow Diagram. See electronic version for colors. 19

Figure 3.4. RBG (a) and HSV (b) color models (Datumizer, 2010a; 2010b). See electronic version for colors. 21

Figure 3.5. Illustration of several types of radial lens distortion (OpenCV Documentation, 2018).....24

Figure 3.6. The distorted image (a) from the Nikon KeyMission 170 camera and the same image that has been undistorted (b). The undistorted image was created using the remapping function of OpenCV. 25

Figure 3.7. Diagram of the machine vision based yield mapping system. 26

Figure 3.8. Model of the machine vision camera bracket (left), (a) is a metal piece used to deflect incoming onions from the camera (b), and (c) is the external light source. The left image (d) shows the setup positioned on the farm harvester. 27

Figure 3.9. System Assembly. 29

Figure 3.10. Determination of size classes for shallot onion classification. 32

Figure 3.11. Shallot onion size calibration results. 34

Figure 4.1. Hue intensity distribution of a sample image in the feasibility study. See electronic version for colors. 38

Figure 4.2. Segmentation results. 39

Figure 4.3. Onion detection results. See electronic version for colors. 40

Figure 4.4. Onion detection accuracy of the current machine vision algorithm (top) and accuracy obtained by doubling the output (bottom). 41

Figure 4.5. Shallot onions detected by the computer vision algorithm vs. manual count. 42

Figure 4.6. Color thresholded result of the calibration object (a) and detection of the ball in the original image (rb). The diameter of the ball in pixel length is determined using a CHT. 44

Figure 4.7. Manually determined onions sizes vs. predicted onion sizes from WST algorithm. 45

Figure 4.8. Cumulative fraction plots of the predicted diameter and true diameter distributions. 46

Figure 4.9. Size class distributions for the manual sorting vs. computer vision algorithm. 48

Figure 4.10. Segmentation Results. 49

Figure 4.11. Onion detection results. (a) shows final detection results for the initial algorithm, and (b) shows results from the WST segmentation method. See electronic version for colours. 51

Figure 4.12. Shallot onions detected by the computer vision algorithm vs. manual count. Top shows results for the initial algorithm developed during the feasibility study, and bottom shows the improved WST segmentation algorithm results. 52

Figure 4.13. Shallot onions detected by the computer vision algorithm vs. manual count. Top shows results for the initial algorithm developed during the feasibility study, and bottom shows the improved WST segmentation algorithm results. 53

Figure 4.14. Image showing the sampling points collected on the shallot onion field (a) and the boundary of the entire study area (b) 55

Figure 4.15. Final yield maps for the small (a), medium (b), large (c) onion classes and the total count (d) of the shallot onion field. 56

Figure C-1. Example image of the size calibration setup (**section 3.2.6**). 86

Figure C-2. Modification of the size classes for shallot onion classification. 87

List of Acronyms

ASABE	American Society of Agricultural and Biological Engineers
CCD	Charged-Couple Device
CHT	Circular Hough Transform
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CV	Computer Vision
FN	False Negative
FP	False Positive
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HIS	Hue-Saturation-Intensity
HSV	Hue-Saturation-Value
ICPA	International Conference on Precision Agriculture
IDW	Inverse Distance Weighting
IFPRI	International Food Policy Research Institute
IPS	In-Plane-Switching
IR	Infrared
KS	Kolmogorov-Smirnov
LCD	Liquid Crystal Display
MLP	Multi-Layer-Perceptron
MV	Machine Vision
PA	Precision Agriculture
R^2	Coefficient of Determination
RGB	Red-Green-Blue
RMSE	Root Mean Square Error
SVM	Support Vector Machine
SSCM	Site-Specific Crop Management
SSD	Solid State Drive
TN	True Negative
TP	True Positive
WST	Watershed Transformation
3D	3 Dimensional

Chapter 1. Introduction

1.1. Yield Monitoring

Throughout history, agriculture has remained one of the most important industries as it provides rich produce distributed in mass quantities. Traditionally, agricultural practices require only a few personnel and yet, manage to sustain an unimaginable number of people. Despite the continuous growth and industrialization of agriculture, farming practices have faced many challenges in recent years. With the global population expected to reach 9 billion by 2050, the agricultural field will need to double its productivity to meet this ever-growing demand (United Nations Department of Economic and Social Affairs, 2017). However, with weather becoming more unpredictable due to climate change, farmers need to consider more droughts, floods, temperature fluctuations and weather disasters, rendering this task of expansion increasingly more difficult.

Precision agriculture (PA), or site-specific crop management (SSCM), is a farming management concept that relies on observation and sensing to include inter and intra-field variability in crops or pasture management practices (McBratney, Whelan, & Ancev, 2005). It aims to develop decision support systems that improve farm management by increasing the value of returns while decreasing input costs. This can be done with the use of data mining systems that gather detailed information to develop software capable of facilitating management practices. According to the International Food Policy Research Institute (IFPRI), the adoption of PA practices has been identified as one of the main drivers of yield increase, while promoting the sustainable use of depleting resources, such as water and arable land (Aisenberg, 2017). Technological advancements in sensing, computing power and robotic systems are gradually leading to potential increases in productivity for commercial farmers, who are now turning to new, innovative tools

and methods to enhance their current practices while making them more precise, less wasteful and more effective.

More specifically, crop yield estimation and mapping are important PA tools that can help growers efficiently keep track of their available resources and have access to detailed representations of their farm. Accurate yield estimation allows growers to efficiently manage their harvest logistics, crop storage and sales, and account for losses in a timely manner (Nuske, et al., 2014). Early and accurate predictions are also a key factor for market planning and trade (Bargoti & Underwood, 2017; Cheng, Damerow, Sun, & Blanke, 2017).

However, commercial PA techniques for specialty crops such as fruits and vegetables, including the existence of yield monitoring systems, have not been developed to their full potential. This is mainly due to the large diversity in harvesting methods for specialty crops and those grown for smaller market as compared to row crops. Currently, yield estimation for specialty crops is often done by tedious manual sampling methods which are labor intensive, long and costly (Dorj, Lee, & Yun, 2017; Nuske et al., 2014). Other methods rely heavily on imprecise historical or empirical data which is then extrapolated (Cheng et al., 2017). Moreover, these calculations and measurements performed by humans are often prone to bias and sparsity leading to false predictions (Bargoti & Underwood, 2017). The adoption of automatic PA techniques could substantially benefit specialty crops. This is because in comparison to field crops such as cereals, cotton, hay and grain, specialty crops often require more resources and may be more sensitive to sudden changes in growth conditions. Specialty crops are known to produce high value products, and the development of accurate yield monitoring systems would help farmers keep better track of crop quality and reduce their operating costs by adjusting their thinning practices and the size of the harvest labor force (Patel, Jain, & Joshi, 2012).

1.2. Computer Vision

Among the numerous sensing techniques used in PA, digital imaging is one which has been adopted in various applications such as robotic harvesting, weed control, phenotyping, pruning, seeding, spraying, thinning, sorting and packaging (Kapach, Barnea, Mairon, Edan, & Ben-Shahar, 2012). In each of these applications, computational methods extract useful information from digital images or videos with the help of computer vision (CV), allowing for the automation of the tasks described previously. Moreover, these automatic tasks can be accomplished in a non-destructive manner which is important for high value specialty crops (Automated Imaging Association, 2014). More and more, CV is being employed as a substitute for traditional visual observations, such as counting objects along a conveyor, detecting serial numbers, searching for physical defects and sorting or grading. As a result, there has been a significant reduction in labor requirements and processing time and yet, there is now better consistency and uniformity in measurements (Sun, 2008). Recent advances in robotics and automation allow the gathering of large amounts of visual data from different methods, including simple camera systems (Blok, Barth, & Berg, 2016; Payne, Walsh, Subedi, & Jarvis, 2013; Pothen & Nuske, 2016), unmanned ground vehicles (UGV's) (Wang, Nuske, Bergerman, & Singh, 2012) or unmanned aerial vehicles (UAV's) equipped with color cameras that capture images of even very large farms (Telledis & Levin, 2014). This data can also benefit from high spatial and temporal resolution, which can be condensed and visualized using geospatial mapping software such as ArcGIS[1]. Image processing techniques can be used to analyze this information and extract key properties of the farm such as health, location of crops, spatial distribution and alternatively crop yield.

[1] Mention of a trade name, proprietary product, or company name is for presentation clarity and does not imply endorsement by the authors, or McGill University, nor does it imply exclusion of other products that may also be suitable.

1.3. Research Objective

This study focuses primarily on the use of computer and machine vision (MV) to perform detection, crop-yield estimation and yield mapping for the French grey shallot (*Allium oschaninii*). Shallot onions grow in clusters, where bulbs rest on the surface of the soil (**Figure 1.1b**). They are harvested uniformly using a windrower and a trailer. A conveyor belt (**Figure 1.1a**) collects the onions from the ground using a paddle, and the onions are then deposited in a trailer which stores them during harvest. Spatial variabilities in soil type, soil fertility and other cropping conditions contribute to disparity in onion size, and onion size is an important limiting factor when determining the percentage of harvest destined to external suppliers. Quality assessment and sorting of shallots is traditionally done by human visual inspection, and usually only after harvesting is fully completed. The research objective of this study was to develop a new yield mapping technology for specialty crops capable of performing quantity and quality assessment. Specific objectives were to 1) develop a low-cost prototype system that could be easily integrated in agricultural harvesting practices and 2) to perform yield mapping of crop size distribution to allow better management practices on the farm.

a) **b)**

Figure 1.1. Shallot onion harvesting machine with trailer (a) and onion field (b).

Chapter 2. Literature Review

2.1. Digital Imagery

Digital imaging or digital image acquisition is a process whereby an image in the physical world is captured by a sensor, such as a camera, and then quantized into a discrete data structure capable of being understood by a computer. Image digitization corresponds to sampling color and intensity values into a matrix form with several rows (M) and columns (N). The finer the sampling (i.e. the larger the values of M and N), the more detailed the approximation of the analog image will be. Each individual sampling point is called a pixel, and M and N indicate the dimensions of the overall pixel grid (Sonka, Hlavac, & Boyle, 2015). A grayscale image consists of one single channel where pixel intensities vary from a value of 0 (indicating the complete absence of light) to 255 (full presence of light). If an image is presented in color, it will have multiple superimposed channels. The most common format for color images is the Red-Green-Blue (RGB) format where an image is decomposed into three channels where each one is equal to one of the primary colors of light (Sonka et al., 2015).

2.1.1. Computer Vision and Machine Vision

CV is a process where computational methods are used to recreate the effect of human vision, aiding computers to understand and perceive images. This is done through the creation of mathematical models that relate the digitized input image to phenomena in the physical world. These models are created using geometry, physics, statistics and learning theory (Huang, 1996; Sonka et al., 2015). Machine vision (MV) uses CV technology and methods to create systems which “see” steps along a production line. A MV system must be comprised of standard components: a camera, computer software for the analysis and processing of images, a pattern recognition module and an output component (monitor, robotic arm, etc.). A machine vision

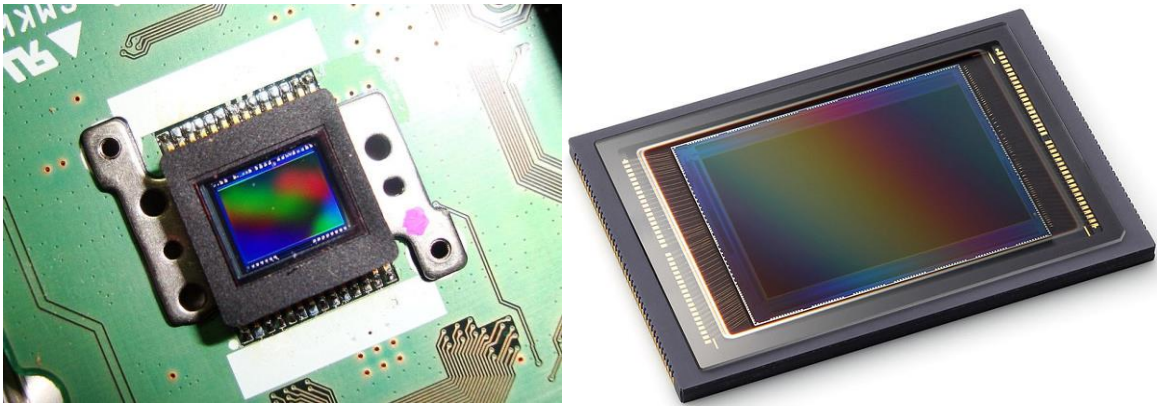
system uses a camera to first view an image or video stream. CV algorithms then process the image and extract valuable data from it, and finally, this data is used to control and activate other components of the system. A MV system cannot function without the use of computer software, differing from CV which can be used alone, and includes many technologies, software and hardware products, integrated systems, actions, procedures and expertise (Graves & Batchelor, 2003; Sonka et al., 2015). It is important to understand the difference between these two disciplines as they will be widely used throughout the redaction of this thesis.

2.1.2. Camera Systems

Some of the earliest CV studies made use of black and white (also referred to as monochrome) cameras to perform fruit detection (Gongal, Amatya, Karkee, Zhang, & Lewis, 2015). Using a combination of geometric features, texture and reflectance, it was possible to identify fruits such as melons or green immature oranges with relatively high accuracy (75-88.0%) (see Appendix D for definition of performance metrics) using these simple systems (Cardenas-Weber, Hetzroni, & Miles, 1991; Dobrusin, Edan, Grinshpun, & Peiper, 1992; Edan, Rogozin, Flash, & Miles, 2000; Plá, Juste, & Ferri, 1993). However, after further analysis, it was noted that with additional sensor data results could be improved further by adding another layer of data beyond intensity, such as a color filter to amplify the contrast between a crop and the background.

Later studies integrated color cameras as the primary sensor. Color cameras with either CCDs (charged-couple devices) or CMOS (complementary metal-oxide semiconductors) (**Figure 2.1**) have been widely used in research geared towards robotics and the automation of agricultural practices and operations (Bac, Van Henten, Hemming, & Edan, 2014; Gongal, Amatya, Karkee, Zhang, & Lewis, 2015; Mollazade, Omida, & Arefi, 2012). CCD devices convert light into an electrical charge which must be transported across the chip without distorting the signal using a special manufacturing process. An analog to digital converter translates every pixel value to a

digital value. These sensors can provide very high-quality images which are less susceptible to noise. In CMOS cameras, light is converted to electrical signals using electronics which are directly integrated on the surface of the sensor: Several transistors located at every pixel move the electrical charge using a direct wire connection, rendering data transfer more rapid. CMOS can also benefit from a higher dynamic range, allowing them to capture both high-lit regions and shadowed areas in the same image. Moreover, these sensors are very inexpensive because unlike CCD sensors, they are manufactured using traditional processes. CCDs are used in applications that focus primarily on capturing high-quality images with a high number of pixels and excellent light sensitivity.



a) **b)**

Figure 2.1. Images of a CCD sensor (a) (Ahmed2IQ, 2009) and a CMOS sensor (b) (Nyman, 2012).

Both sensors are equipped with a Bayer filter, which allows for the capture of three channel RGB images. Therefore, color cameras provide additional layers of information when compared to monochrome cameras. Moreover, in most cases of fruit detection, color is one of the most prominent features of interest when estimating yield in canopies (Gongal et al., 2015). Color segmentation is also possible when using color images, and it can be performed in different color spaces including the RGB and Hue-Saturation-Value (HSV) color spaces (Gongal et al., 2016; Hannan, Burks, & Bulanon, 2009; Linker, Cohen, & Naor, 2012; Wang et al., 2012). However,

color sensing does come with disadvantages, the most predominant being its high sensitivity to changes in lighting and motion. This is especially true in outdoor conditions, where natural light may exhibit high variance in intensity on both the temporal and spatial scale. Despite this, existing methods have been developed that manage to detect fruit, such as oranges and mangoes in natural lighting conditions with accuracies within 74% and 90% (Hannan et al., 2009; Payne et al., 2013; Sengupta & Lee, 2014).

Spectral cameras are another type of sensor which collect and process spectral information, capturing objects of interest by analyzing their reflectance properties at different wavelengths. This type of sensing has proved to be efficient predominantly when the color of the fruit or vegetable of interest is like that of the surrounding background (Bulanan, Burks, & Alchanatis, 2010; Wang & Li, 2015). Similarly, thermal cameras can also be used to differentiate objects that have a similar color range as their surroundings. These sensors make use of the infrared (IR) energy emitted by an object, which is also known as its heat signature. As an object increases in temperature, it emits larger amounts of radiation, thus, simultaneously increasing its heat signature. A thermal camera can create an electronic image by detecting small temperature differences in an object and registering these as a spatial array. This is possible because an object's temperature will rarely be the same as that of the objects around it. A study performed by Stajnko, Lakota, & Hočevar (2004) used thermal imaging to estimate the number and diameter of apple fruits in an orchard. The algorithm was able to correctly determine the apple fruit count with an accuracy between 83% and 88%, and the R^2 values between the measured diameter of the apples and the diameter determined by the algorithm were between 0.68 and 0.88.

Stereo vision systems are another alternative for object detection in 3-dimensional (3D) space. These systems consist of two or more cameras separated by a small distance, mimicking binocular human vision. Images of the same scene viewed from different angles are captured and then matched to estimate the displacement (also known as disparity) of an object. Researchers

have used this method to both identify and locate objects such as apple fruit in a 3D space for applications of robotic harvesting and high-resolution yield estimation and mapping (Mirbod, Yoder, & Nuske, 2016; Wang et al., 2012). However, stereo vision is typically not favored due to the method's high complexity and long computation time (Hannan & Burks, 2004). Nonetheless, recent advances in hardware development have contributed to an increase in stereo-vision based applications.

2.2. Applications of Computer Vision in Agriculture

The use of CV and MV has helped, through the automation of agricultural processes, to increase productivity, reduce production costs, monitor and increase yield quality and reduce the need for manual labour (Kapach et al., 2012; Sun, 2008). In earlier studies, MV was predominantly applied in production lines to automate processes using image processing techniques and was considered relatively easy to integrate in the various production and handling procedures of fruits and vegetables. A typical machine vision system is composed of two primary modules: one for image processing and another for pattern recognition. The image processing module analyzes the composition of the image and proceeds by passing it to a pattern recognizer. This second module classifies the image using one or multiple pre-defined quality categories that correspond with the desired patterns within the image. These recognizable patterns, also known as features, may represent blobs, edges, corners or lines (Al-Ohali, 2011; Stanhope, 2016).

Many MV algorithms have been developed for the classification of vegetables and fruits on conveyor belts or similar apparatuses (Benalia, et al., 2016; Mizushima & Renfu, 2013; Wang & Li, 2015). A first example of this would be the algorithm for automatic segmentation of color images for apple sorting and grading developed by Akira Mizushima and Renfu Lu (2013). Using a support vector machine (SVM) model and a set of training examples, an algorithm was constructed to define the boundary between the pixel spaces corresponding to fruit and those

corresponding to background. The contours of the apples were extracted using binary images that mapped these two distinct spaces. The SVM model, combined with the procedures of Otsu's method (1979), performed the segmentation process with an average segmentation error (percentage of mislabeled pixels over correctly labeled pixels) between 3.31% and 25.5% for several types of red apples. A similar methodology was applied by Wang and Li (2015) when developing a multimodal machine vision system for the quality inspection of onions. Their results showed that 88.9% of healthy and defective onions were identified correctly. Similarly, Al-Ohali (2011) developed a CV-based system for sorting and grading dates on a conveyor belt. After determining the key external features of good and substandard quality dates, the system classified the dates into three grade categories using RBG images, where grade one dates represented fruits of highest quality. Various mathematical algorithms were discussed to perform the MV process known as feature description, where key features such as flabbiness, shape, size and color intensity are all expressed using mathematical equations. This is done to convert physical and visual properties into numerical constants and coefficients that could be used for quality assessment (**Figure 2.2**). Although the system developed by Al-Ohali was capable of sorting approximately 80% of the grade 2 dates correctly, this percentage was lower for grade 1 and grade 3 fruit. Misclassifications arose from the system occasionally miscalculating size, shape and color distribution due to the overwhelming number of input features. It was therefore concluded that using less input features yielded better quality results. In more recent models, features are learned systematically with the use of neural networks and large amounts of data (Kamilaris & Prenafeta-Boldu, 2018). Other later applications include methods for identifying crops in the field. Blok et al. (2016) developed a MV algorithm for identifying broccoli heads on a farm, which could eventually be integrated into a fully autonomous selective harvesting process. A texture and color-based segmentation was used to isolate the heads from the background. Results from the automatic segmentation method were compared with those obtained from two human experts by comparing the spatial overlap of the predicted and true broccoli head regions.

The precision score of the segmentation was 99.5% and overall accuracy of the image segmentation was 92.4%. Other studies include work by Kondo et al. (2009), who developed a machine vision system for autonomous harvesting of tomato fruit clusters using stereo images of tomatoes in a greenhouse. The images were converted to the Hue-Saturation-Intensity (HSI) color space to generate chromacity distribution plots of H-versus-I. These plots were used to cluster fruit region properties and develop a classifier. The research results showed a 73% success rate in locating the stems of clusters.

a) **b)** **c)**

Figure 2.2. An example of feature intensification using a date fruit. Original image (a), thresholded image (b) and edge image (c). Reprinted from Al-Ohali (2011).

2.3. Machine Vision for Yield Determination

Major applications of CV in agriculture have been developed in fruit detection, where the goal is to identify individual fruits, segment them from scenes with branches, foliage, sky, and localize them in space for yield estimation or as an initial step to the development of robotic harvesting systems (Kapach et al., 2012). Many of these applications are methods developed for counting apple fruits using canopy images (Gongal et al., 2015; Linker et al., 2012; Wang et al., 2012; Zhou, Damerow, Sun, & Blanke, 2012). Stajnko et al (2004) developed a method for detecting apple fruit using thermal imaging. Images were collected at five time periods (June to August) to model apple growth over the season. Images were taken late at night to capture a more defined temperature gradient between apple fruits and foliage. Coefficients of determination

(R^2) between manually detected apples and the estimated number of apples ranged from 0.83 to 0.88. It was also noted that more mature apples were easier to detect due to their ability to radiate more heat. Wang et al. (2012) created a similar stereo vision-based system using a two-camera stereo rig. This system was stationed on an autonomous orchard vehicle designed to work at night with artificial lighting. It converted apples to the HSV color space, and then used color segmentation and specular reflection to separate both red and green apples from foliage. The error obtained for crop yield estimation was -3.2% for red apple trees, and 1.2% in green apple trees with additional calibration due to significant fruit occlusion. Gongal et al. (2016) later developed an over-the-row machine vision system using both an RGB and stereo camera which captured dual images from both sides of the plant canopy and localized them in space. The experiment was performed in a controlled environment using a covered system with artificial lighting and a tunnel structure. Using image processing and clustering, apples were identified in the images based on shape and color with an accuracy of 78.9%. More state-of-the-art methods (Bargoti & Underwood, 2017) have adapted machine learning techniques, such as Multi-Layered Perceptrons (MLPs) and Convolutional Neural Networks (CNNs), to perform pixel level fruit-segmentation under natural sunlight in orchards. The binary images were processed using both an image segmentation based on a Watershed Transform (WST) and a Circle Hough Transform (CHT). The watershed algorithm was able to detect apples with an R^2 value of 0.83 and output an apple yield map for an orchard block using an on-board Novatel SPAN Global Positioning Inertial Navigation System (GPS/INS) recording the vehicle position and pose with every image taken (**Figure 2.3**).

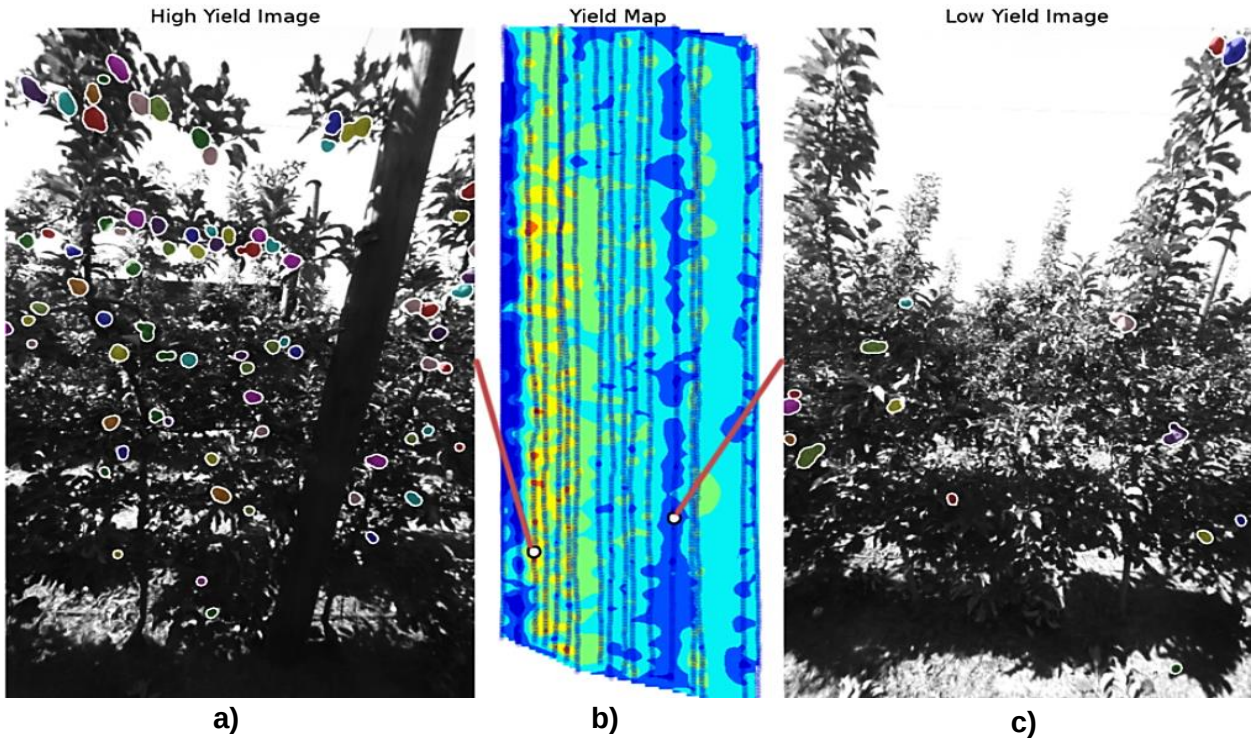


Figure 2.3. Apple yield map of an orchard block created using a computer vision algorithm. Reprinted from Bargoti & Underwood (2017). See electronic version for colours.

Individually geo-referenced images are segmented, and fruit detection is performed to obtain a fruit count per image (b). Examples of a high yield (a) and low yield (c) image are pictured.

Studies have also focused on the identification of citrus fruits in similar conditions (Dorj et al., 2017; Sengupta & Lee, 2014). In these two studies, computer vision algorithms were developed to count citrus fruits on trees using image processing and estimate early overall yield. Sengupta and Lee (2014) used shape and texture analysis to detect immature green citrus fruit in a canopy. Texture classification was performed using a Support Vector Machine (SVM), Canny edge detection and a graph-based connect component algorithm and Hough line detection. The algorithm accurately detected 80.4% of citrus fruit. The study by Dorj et al. (2017) was based primarily on the color features of orange fruits. The algorithm consisted of converting the images to the HSV color space, thresholding, orange color detection, removal of noise using a median filter, watershed segmentation and counting. Overall, this algorithm obtained a high correlation (R^2 = 0.93) between the predicted count of oranges and human observation.

2.4. Challenges of Machine Vision Applications

Although CV systems have proven to have high detection rates and show promising results, the presence of many external factors in farm image data (variations due to illuminations, occlusions, clustering, etc.) has often negatively influenced the results. Therefore, it is crucial that algorithms remain invariant to these factors to provide a reliable outcome (Bargoti & Underwood, 2017). Moreover, farm image data is prone to large intra-class variations primarily due to variable illumination conditions, occlusion by other crops or foliage, clustering of crops, camera view-point, and seasonal maturity levels leading to crops of varying size, shape or color (Bargoti & Underwood, 2017; Hannan et al., 2009; Sengupta & Lee, 2014). Changes in object reflectance can cause object detection to be somewhat unreliable and may lead to incorrect or incomplete segmentation due to a non-uniform distribution of light intensity (Gongal et al., 2016). This problem can be addressed by creating a controlled, uniform lightning environment from which visual data is taken. Examples of controlled lighting environments include an over the row platform with integrated LED lights (Gongal, et al., 2016), a wooden box with a painted black interior (Al-Ohali, 2011) or simply performing the experiment at nightfall (Nuske, et al., 2014; Wang et al., 2012), Other alternative solutions include using additional cues such as a perimeter-based detection method on top of basic color detection (Hannan et al., 2009; Payne et al., 2013) when variable lighting conditions are unavoidable.

Other existing challenges are the multiple detection of the same object within sequential images, or occlusion by other objects or fruits which can lead to miscounting in yield calculation applications. Gongal et al. (2016) used a 2D and 3D imaging approach where apples identified in multiple images were mapped together in a common coordinate system that correctly identified and removed duplicates. The apples in the orchard were represented in a 3-dimensional space where apples registered with the same X, Y and Z coordinates were considered as one fruit. Wang et al. (2012) developed a similar software that calculated the distance between every two

apples, and then merged the apples as one whenever this distance was below a given threshold (**Figure 2.4**). Hannan et al. (2009) used a centroid-based detection method to identify fruit clusters as a single fruit, and a perimeter-based detection method to locate the individual fruits which had a success rate of 93% and a false detection rate of 4%.

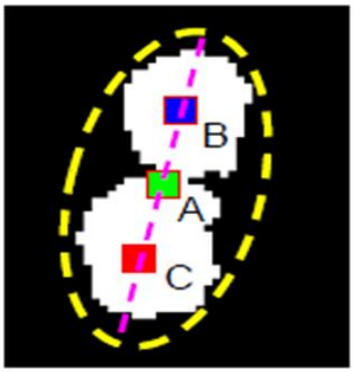


Figure 2.4. Result of splitting a region representing two different apples.
Reprinted from Wang et al. (2012).

2.5. Summary of Literature Review

Extensive work has been done to perform the detection of fruits in orchard environments such as for apples (Gongal, et al., 2016; Linker et al., 2012; Wang et al., 2012; Zhou et al., 2012), oranges (Dorj et al., 2017; Hannan et al., 2009), mangoes (Payne et al., 2013), and berries (Nuske, et al., 2014; Pothen & Nuske, 2016). Sorting processes have also been developed for fruits on conveyor systems (Al-Ohali, 2011; Sofu, Erb, Kayacan, & Cetis, 2016); however, none have attempted to develop a system directly linked to industrial harvesters that can generate a yield map. The initiative to develop better automated crop-estimation systems for vegetables, such as a machine vision-based yield monitor for vegetable crops, is one that has yet to reach its full potential. For this research project, applications of CV and MV are explored to develop a system for the yield mapping and size characterization of shallot onions. Size needed to be determined in terms of a standard 2D metric, and localization in space or even within the image

was not essential. Therefore, after careful analysis, an RGB CMOS high-resolution camera was chosen as the final sensor for this application. Size is an important property which can be described mathematically by a selection of parameters such as volume, weight, length and diameter (Moreda, Ortiz-Cañavate, García-Ramos, & Ruiz-Altisent, 2009). Many CV and MV methods have been proposed to non-destructively measure the size of various specialty crops including but not limited to apple, berries, citrus and dates (Al-Ohali, 2011; Gongal et al., 2016; Mirbod et al., 2016; Sengupta & Lee, 2014). Although studies have been performed for quality inspection of sweet onions (Shahin, Tollner, Gitaitis, Sumner, & Maw, 2002; Wang & Li, 2014; Wang & Li, 2015), similar work facilitating the yield estimation of shallots remains scarce. Therefore, an over-the-row MV system was created to accelerate the yield estimation process by running visual inspection on the go during harvesting and perform real-time characterization of crop quality.

Chapter 3. Materials and Methods

3.1. Feasibility Study

A feasibility study was performed to determine whether a suitable algorithm for shallot onion detection could be developed. This study consisted of mounting an RGB camera on the conveyor to capture video data of the harvesting process. This video data would then be used to create a software that would be integrated in the final yield mapping system. The following sections describe the key components and processes that were involved during the feasibility study.

3.1.1. Image Acquisition

A Nikon KeyMission 170 action camera (Nikon, Minato, Tokyo, Japan) was mounted on each individual harvester using a magnetic mounting base which could easily be attached to any of the existing harvesting machinery (**Figure 3.1**). The KeyMission was chosen due to its ability to provide high resolution images and video, as well as its resistance to changes in outdoor conditions such as wind, dust and even rain (Balletti et al., 2014). These traits were essential given the nature of this project.

Figure 3.1. Nikon KeyMission 170 action camera (Nikon, 2018).

The data collected included videos of the harvesting process that were approximately 6 to 8 min long. This corresponded to the total time it took to harvest a single crop row. During operation on the field, the video resolution was set to 1920 x 1080 pixels and a frame rate of 60 frames/s. Multiple positions for the camera were tested to find a location where an optimal view of the vegetable crops would be attained. For the view to be considered optimal, the image provided by the camera had to be clear and chances of miscounted vegetables needed to be low. **Figure 3.2** shows a representation of the multiple camera positions that were considered for the system's operation and the final position chosen.

a) **b)** **c)**

Figure 3.2. Three views considered for the placement of the camera.

The first (a) is a direct top view of the conveyor belt. The second (b) is a side view with the camera mounted on one of the conveyor's sides, and the third (c) is an interior view of the onions just before they are placed on the conveyor belt. The direct top view was chosen for it provided the clearest image with the least amount of shape distortion due to projection.

3.1.2. Software

Software Structure

One of the main challenges of this study was to select appropriate, efficient and fast methods for detecting, intensifying and classifying the characteristic traits of the onion bulbs. A series of image-preprocessing steps were performed to extract regions of similar texture and identify them as desirable objects within the image. **Figure 3.3** shows a flow diagram of the basic structure of the software and the aspects of the algorithm that were developed during the feasibility study.

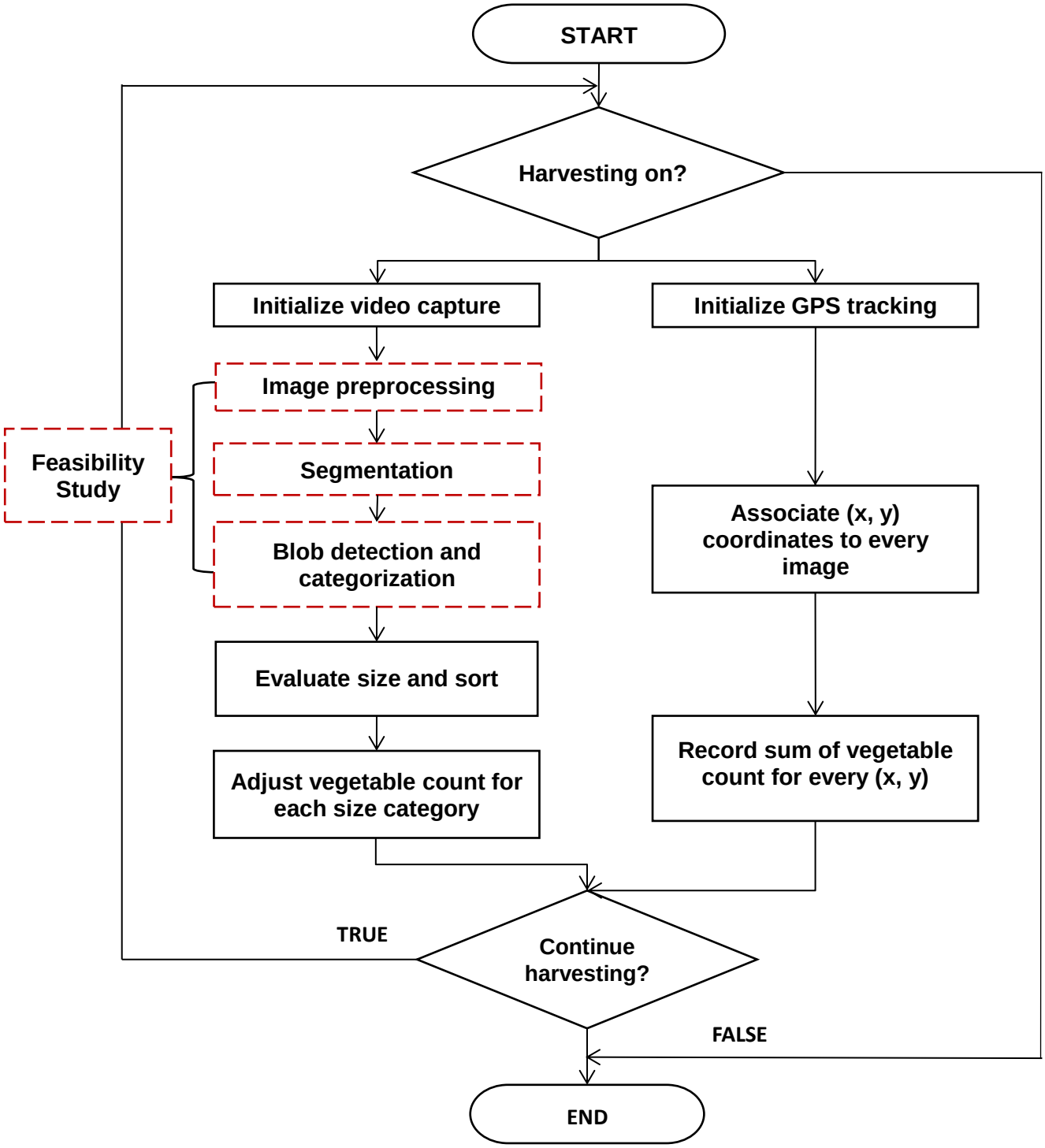


Figure 3.3. Software Flow Diagram. See electronic version for colors.

Initial Algorithm

Segmentation is a process where regions of interest are extracted from an image by separating the foreground objects (shallot onions) from the background (conveyor). Accurate segmentation is crucial since it is the starting point for the succeeding steps such as size classification and counting (Bargoti & Underwood, 2017; Mizushima & Renfu, 2013). Challenges including highly variable illumination and shadows can significantly affect the segmentation process and make it ineffective. Performing data collection in a controlled lighting environment (i.e. nightfall) can help achieve better segmentation results (Pothen & Nuske, 2016; Nuske, et al., 2014; Wang et al., 2012). However, in practice, onion harvesting usually occurs in natural daylight and incorporating cameras on tractors will be easier for growers if large experiments are performed during normal operation times and conditions (Bargoti & Underwood, 2017).

Digital cameras typically capture images in the RGB format, where each channel corresponds to the intensity of the three primary colors of light (red, green and blue). All colors are then created by the additive reproduction process of various amounts of red, green and blue, and brightness values ranging from 0 to 255 for each color. For example, red, green and blue are defined by the vectors (255, 0, 0), (0, 255, 0), and (0, 0, 255), respectively. White can be represented by combining all three components at their highest intensity (255, 255, 255), and black is the absence of all colors in each channel (0, 0, 0). **Figure 3.4a** illustrates a model of the cartesian RGB color space. The RGB model is not the most intuitive for discerning color from a perceptual point of view as it is difficult to extract characteristics such as lightness and intensity (Gongal et al., 2015; Wang et al., 2012). Therefore, images are converted to the HSV color space illustrated in **Figure 3.4b** using the following conversion formulae (Nishad & Chezian, 2013).

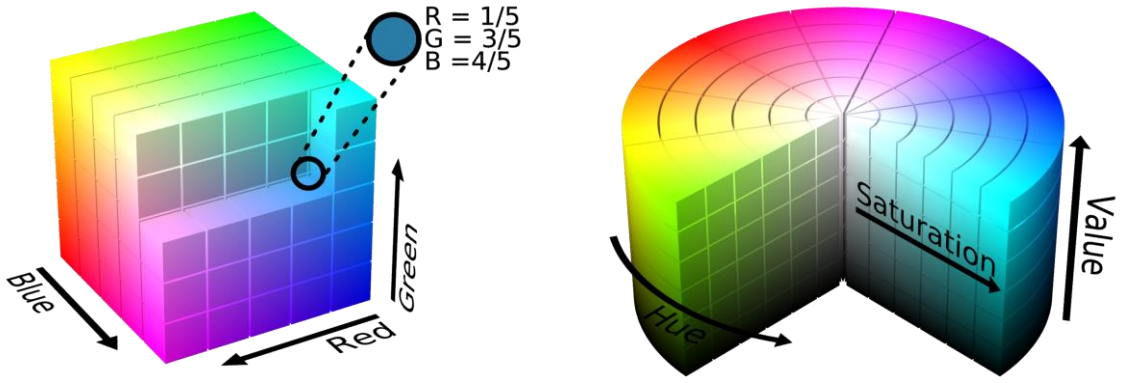


Figure 3.4. RBG (a) and HSV (b) color models (Datumizer, 2010a; 2010b). See electronic version for colors.

- The *hue* (H) of a color is the pure color we are examining. All tones and shades of a given color correspond to the same unique hue. Hues are defined using an angle ranging between 0 and 360 along the horizontal cross-section of the cylinder.
- The *saturation* (S) of a color describes how much white is present within the color. A fully saturated color is strong in pigment. For example, tints of red have saturations ranging between 0 and less than 1, while white has a saturation of 0.
- The *value* (V) of a color describes its lightness, or how much black is present within the color. A value of 0 would be black, where lightness increases gradually as value approaches 1.

To convert an RGB color into the HSV space, we must first determine the maximum (M) and minimum (m) intensities of each pixel, and the difference between them, also known as the *chroma* (Δ).

$$M = \max(R, G, B)$$

$$m = min(R, G, B) \tag{3.1}$$

$$\Delta = M - m$$

H is represented by a piecewise function where the chromatic intensity is determined by a two-color difference component. The function relies on the value of M, which gives the angular position of the color on the cylinder. The hue is then normalized by adding a value of either 0, 2 or 4. The result H' is then measured in degrees and has a value ranging from 0° to 360° (Agoston, 2005).

$$H' = \begin{cases} undefined, \ if\ \Delta = 0 \\ \frac{G-B}{\Delta}\ mod 6, \ if\ M = R \\ \frac{B-R}{\Delta} + 2, \ if\ M = G \\ \frac{R-G}{\Delta} + 4, \ if\ M = B \end{cases} \qquad H = 60° \times H' \tag{3.2}$$

V is derived from the maximum colour component, M.

$$V = M \tag{3.3}$$

Finally, to determine S, we divide Δ by M.

$$S = \begin{cases} 0, & if\ V = 0 \\ \frac{\Delta}{M}, & otherwise \end{cases} \tag{3.4}$$

The color indices of the HSV space have proven to be more robust against multiple external factors, such as varying lighting conditions. Standard indices such as HSV and normalized RGB have also been used successfully for plant segmentation (Golzarian et al., 2012). These color indices were obtained through experimentation with images from the test videos. Based on the preliminary analysis, the best HSV pixels ranges for shallot onions were H[0:46], S[40:180], and V[40:255]. The computer code related to this section can be found in Appendix A-1.

Lastly, two morphological operations were applied using image filters. The first is an erosion operation, which is done to eliminate unwanted boundary pixels or protrusions from the

objects. The second operation is dilation, which enlarges any objects in the image and fills small holes within them. Object pixels in the original image that form a region defined as R are eroded by an operator A, producing a smaller region known as R' in the new image. This operation is often represented as:

$$R' = R \, \theta \, A \tag{3.5}$$

Where the symbol θ is read as “eroded by”. The matrix operator A can have any shape, but typically symmetric erosion operators are most common. A single erosion that is followed by a single dilation by the same A is called an opening, and a dilation followed by an erosion by the same operator is called a closing (Gose et al., 1996). These operations were applied using OpenCV functions and an elliptic structuring kernel (12 x 12 pixels) known as A, adapting the process for objects having more circularity. Following this, the image was then partitioned into various segments corresponding to either a vegetable region or the background. Built-in OpenCV functions were used to detect the blobs within the image and determine properties such as pixel area, the diameter and position of the centroid for each blob.

3.1.3. Distortion Correction

Although the KeyMission provided a very high-resolution image, images displayed a large amount of distortion, a phenomenon where lines which are straight in the real-world deviate from their rectilinear projections in the image (**Figure 3.5**). This effect is due to the wide-angle lens of the action camera which is designed to have a large field of view despite its small focal length. The most predominant form of distortion observed was radially symmetric distortion or barrel distortion. This distortion had to be corrected to extract quantitative measurements that corresponded to real-world dimensions (Balletti et al., 2014). Correction was done using distortion calibration methods in CV libraries such as OpenCV. A camera calibration process was performed with a total of 25 pictures of a checkerboard pattern (13” x 9”) located at multiple positions within

the camera's field of view to estimate the intrinsic parameters of the camera. These parameters included the focal length of the lens *(F_x, F_y)*, the coordinates of the center of projection of the image (*Cx, Cy*), and the radial lens distortion coefficients (K_1, K_2, K_3, K_4, K_5) which are all used to create a distortion model (Balletti et al., 2014).

Figure 3.5. Illustration of several types of radial lens distortion (OpenCV Documentation, 2018).
Figure 3.5. shows an undistorted image (a), and two common types of radial distortion: barrel distortion (b) which occurs typically then $K_1 > 0$, and pincushion distortion (c) when $K_1 < 0$.

Table 3.1 shows the intrinsic camera parameters obtained for the KeyMission camera during the calibration process. The average reprojection error in pixels was used to estimate the accuracy of the calibration process and give an estimation of the precision of the calculated parameters. The reprojection error should ideally by as close to zero as possible (Balletti et al., 2014). For this set of parameters, the average reprojection error for the set of calibration images was estimated at 0.046 pixels. An undistorted image from the data set is shown in **Figure 3.6.**

Table 3.1. Intrinsic camera parameters of the Nikon KeyMission 170.

Focal Length		Principal Point		Distortion Coefficients				
F_x	F_y	C_x	C_y	K_1	K_2	K_3	K_4	K_5
801.2	797.1	955.5	563.4	-3.006E-01	9.924E-02	-2.676E-03	2.766E-04	-1.586E-02

a) **b)**

Figure 3.6. The distorted image (a) from the Nikon KeyMission 170 camera and the same image that has been undistorted (b). The undistorted image was created using the remapping function of OpenCV.

3.1.4. Conveyor Speed Calculation

The speed of the conveyor was calculated by using the distance travelled by one of the conveyor paddles over time. The distance between two adjacent paddles was measured to be 33.0 cm, and the total time calculated for 20 paddles to be observed was equal to approximately 9 s. The paddle and conveyor speed were then found to be 0.711 m/s.

3.1.5. Processing Unit

For the feasibility study, all computer processing was done on a Lenovo Flex 4 laptop computer (Lenovo Ltd, Quarry Bay, Hong Kong). Image processing and development of the detection algorithm were performed using the Python (version 3.5.2) coding language (Python Software Foundation, Wilmington, Delaware, USA) and the OpenCV (version 3.2.0) libraries (Itseez, Inc., San Francisco, California, USA). The initial algorithm was designed to run on a 64-bit PC with an Intel® Core™ i7-7500 CPU processor (Intel, Santa Clara, California, USA), with a 2.70 GHz clock speed and 8GB of RAM.

3.2. Prototype System Design

3.2.1. System Components

Development of the system began in the winter of 2017, and since then, there have been continuous improvements over a two-year period. Two main versions of the software have been developed: the first relying solely on video data to recognize onions in images, while the second has an integrated to enable yield mapping along with image analysis. The original concept of the system with all its components is illustrated in **Figure 3.7.** A camera would continuously record data of the shallot onion crops during harvesting and send each image to a computer software capable of counting the total number of vegetables and finally classify them by size. This system would allow the creation of a layered yield map showing the size distribution of the onions across a field. A webcam was used instead of an action camera to allow for easier flow of images directly to the software.

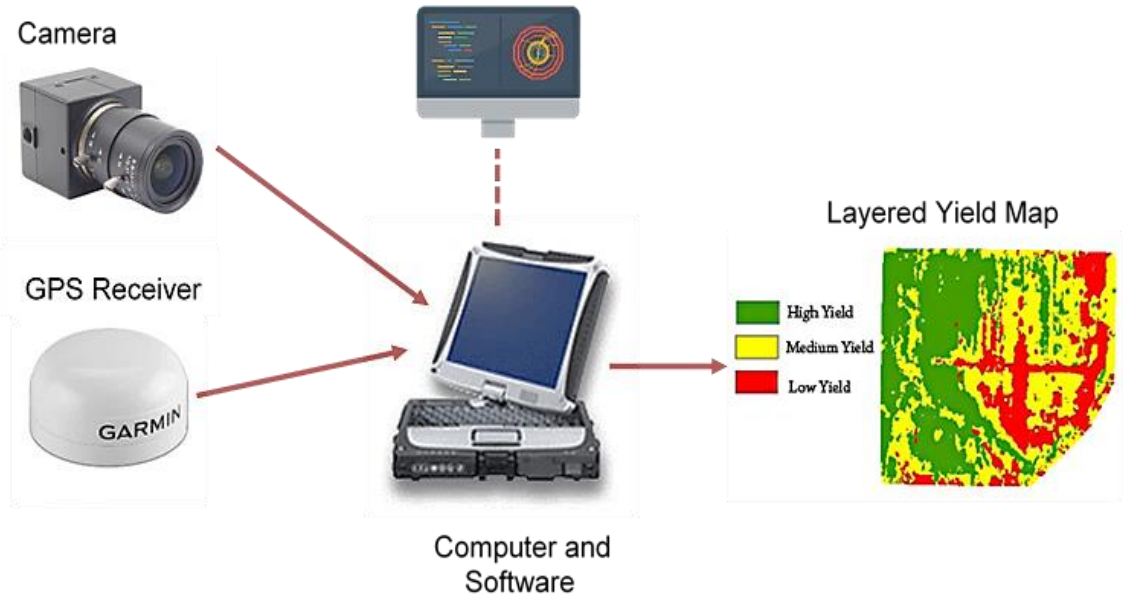


Figure 3.7. Diagram of the machine vision based yield mapping system.

The camera records continuous video of the crops being collected by the harvester. Each frame is coupled with a location tag given by the GPS receiver. The data is analyzed using a computer vision software and exports a yield map showing the size distribution of the crops.

3.2.2. Mounting Bracket Design

A customized bracket (**Figure 3.8**) provided a vertical camera orientation, capturing an image where the camera is facing downwards and directly on the conveyor. The bracket was positioned at the end of the harvester's conveyor to help reduce the amount of onions falling backwards and being detected more than once by the algorithm. An on-board positioning system provided the geographic coordinates of every detected onion in the field. Shock absorbing pads made of Sorbothane (Sorbothane, Kent, Ohio, USA) a synthetic viscoelastic urethane polymer, were placed beneath all the top pieces of the bracket to reduce the vibration effects of the conveyor and to help stabilize the camera.

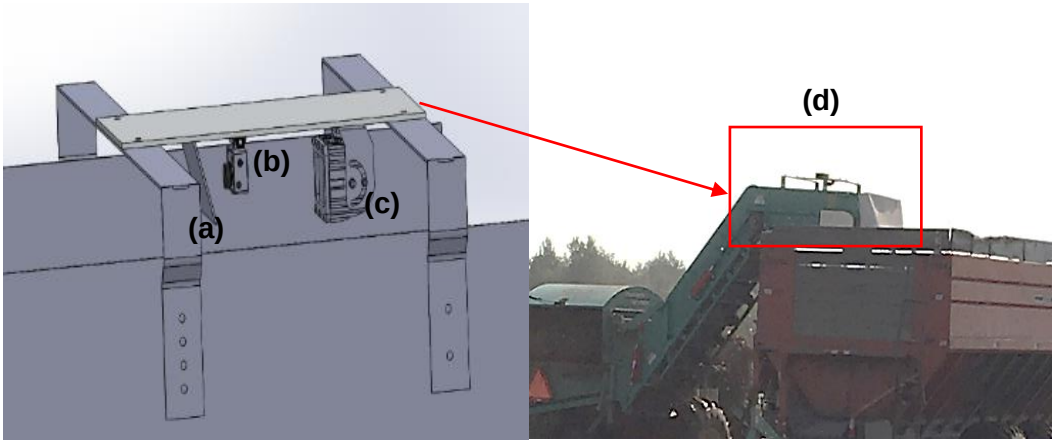


Figure 3.8. Model of the machine vision camera bracket (left), (a) is a metal piece used to deflect incoming onions from the camera (b), and (c) is the external light source. The left image (d) shows the setup positioned on the farm harvester.

3.2.3. Electrical System Design

A control box was designed to house all the electronics and computer hardware used by the system (**Figure 3.9**). The main structure of the box was a complete watertight and crush resistant Seahorse protective case (Seahorse, El Cajon, California, USA). Holes were machined directly on the case to attach connectors for the devices integrated into the system. All key processing components were placed within the box, while an ELP 1080P USB Camera Box (Ailipu

Technology Co., Ltd, Shenzhen, Guangdong, China) and a Garmin 19x HVS NMEA 0183 GPS sensor (Garmin Ltd., Olathe, Kansas, USA) were placed directly above the conveyor belt for image and location acquisition. Power supplied to the devices originated from a 12-V power socket located inside the tractor cab and supplied power to the MINIX NEO-Z83-4-PRO-VESA computer (MINIX, Kowloon Bay, Hong Kong), Eyoyo 10 Inch IPS LCD monitor (Shenzhen Eyoyo Tech. Co., Ltd., Los Angeles, California, USA) and a Logitech K400 Wireless Touch keyboard (Logitech International S. A., Lausanne, Switzerland). A fuse was integrated in power source line to prevent all sensors from damage in case of oversupply. Similarly, a fuse was also attached to the voltage wire of the GPS sensor. Ground and voltage points of all devices were joined using a screw terminal. The webcam was the only device powered separately through the computer using one of the available USB ports. For the prototype, the main attributes required were a high enough RAM and processor speed to allow the algorithm to run in real-time. The processor also needed to be small enough to fit within the available space in the tractor cab while not inhibiting the driver. **Table 3.2** shows some of the main specifications for the MINIX computer.

Table 3.2. Computer specifications

Attribute	Value
Brand Name	MINIX
Series	FBA_MINIX-NEO-Z83-4-PRO-VESA
Item Height	12.7 centimeters
Item Width	40 millimeters
Processor Type	Intel Atom
Processor Speed	1.92 GHz
Number of Processors	4
Memory Size	4 GB
Memory Type	DDR3 SDRAM
Hard Disk Size	32 GB
Hard Disk Interface	Solid State
Wireless Standard	2.4 GHz Radio Frequency
Voltage	3 volts
Hardware Platform	PC
Operating System	Windows 10 Pro (64-bit)
Lithium battery Voltage	3 volts
Lithium battery Weight	0.12 grams

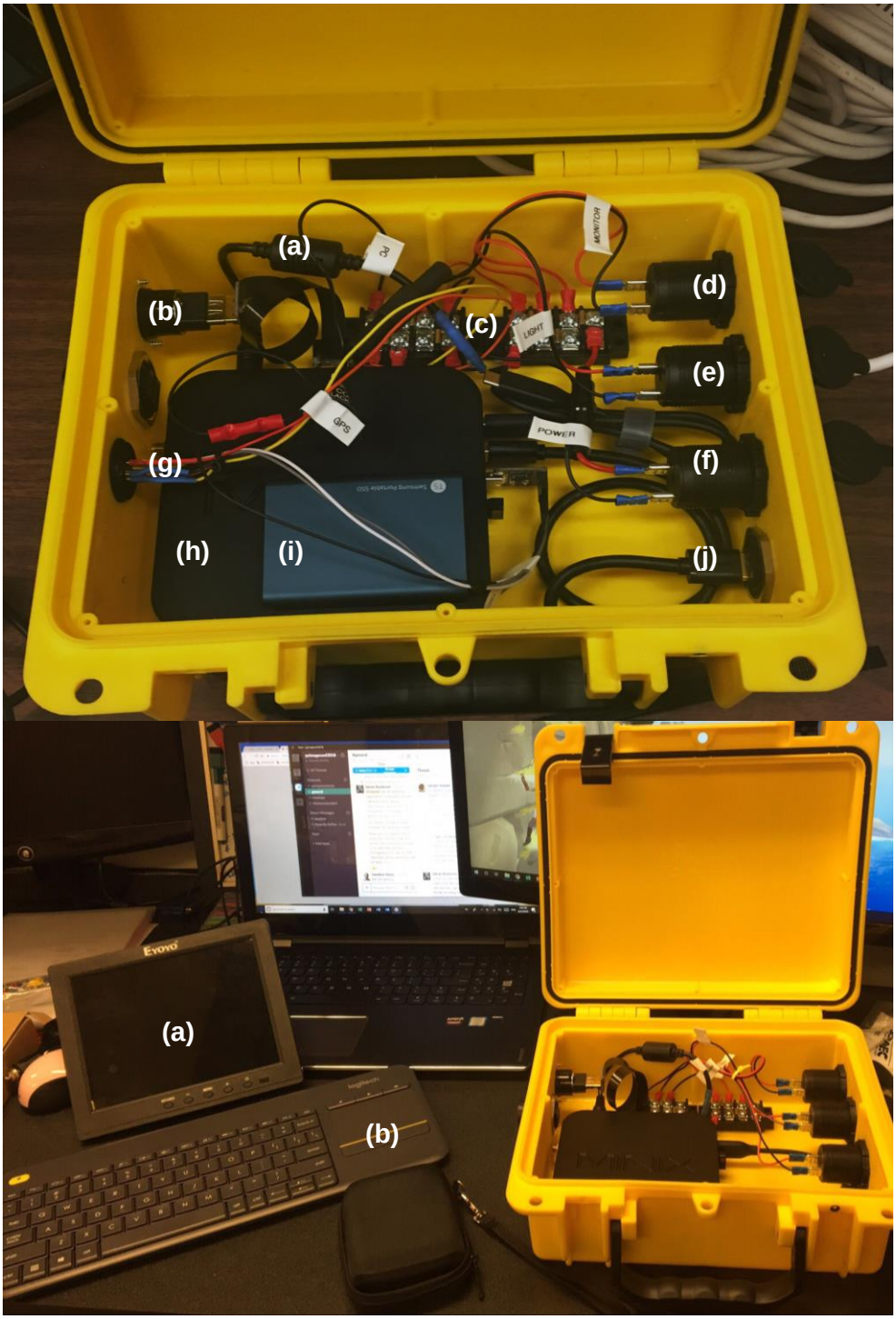


Figure 3.9. System Assembly.

Top: main assembly of the yield monitoring system including the (a) computer power line, (b) HDMI input, (c) terminal block, (g) GPS sensor attachment, (d, e, f) 12-V power sockets, (h) MINIX computer, (i) external solid state drive and (j) USB input for the webcam. Bottom: The computer from the main assembly is controlled using a wireless keyboard (b) and the output is shown on a small monitor (a).

After the feasibility study was completed, a new camera was chosen to facilitate integrability with the software. This camera was a 2.0 Megapixel resolution camera from ELP which could connect to the computer using a USB 2.0 adapter. Video data was recorded at a resolution of 640 x 480 pixels and a frame speed of 120 frames/s, and frames were saved to a Samsung T5 portable external solid-state drive (SSD) (Samsung Electro-Mechanics, Suwon, Gyeonggi-do, South Korea) every two seconds with a corresponding GPS position. Full specifications of the camera, SSD and GPS sensor can be found in **Table B-2**, **Table B-3** and **Table B-4** of Appendix B.

3.2.4. Segmentation

Once the images are converted to HSV, color thresholding was performed using three methods. The first was Otsu's (1979) thresholding selection method which has been largely used in CV applications in agriculture (Gongal, et al., 2016; Mizushima & Renfu, 2013; Mollazade et al., 2012). Otsu's (1979) method automatically determines a threshold using the histogram of a grayscale image. An image histogram can be defined as "a density function $\hat{\rho}(x)$ […] where the range of variable x is divided into a finite number of adjacent intervals that include all the data. These intervals are also called cells or bins" (Gose et al., 1996). In image processing, the variable x can correspond to intensity, color, or a given feature descriptor. Histograms are an essential tool for representing tonal variance within an image and have many applications in image thresholding, segmentation and edge detection. This threshold minimizes the weighted intra-class variance σ_w^2 and is defined as a weighted sum of variance of the two classes:

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + w_1(t)\sigma_1^2(t) \tag{3.6}$$

where ω_0 and ω_1 are the probabilities of the two classes separated by a threshold *t,* and σ_0^2 and σ_1^2 are the variances of these two classes. To obtain proper segmentation, the image must have an intensity histogram that is bi-modal. Otsu's (1979) method was applied to the hue channel.

In the second method, the image was thresholded by applying a band pass filter to the hue channel and a high pass filter on the saturation channel. The threshold values were determined by analyzing the hue channel histogram and selecting the hue region where most of the onions were located. Both methods were then followed by a second segmentation based on texture properties using the magnitude of the red color intensity. This method is often used in apple detection, since apple fruit have a very distinct red color when compared with the green foliage of apple trees (Stajnko et al., 2009; Zhou et al., 2012). Canny edge detection (Canny, 1986) located the contour lines of the onions, and shape properties were extracted from the binary image to identify regions corresponding to onions.

A third method was later adapted after the field trials to improve detection results even further. Segmentation using a marker-based watershed transform (WST) was performed to improve separation of regions that were adjacent and overlapping (Meyer & Beucher, 1990; Vincent & Soille, 1991). In this segmentation approach, an image is interpreted as a topographical surface where the gradient image magnitudes can be represented as elevations. Region edges are equivalent to watershed boundaries and low-gradient region interiors are the catchment basins. The watershed segmentation algorithm attempts to group all pixels belonging to the same catchment basin using the distances of each binary pixel in the mask to the nearest 0-value (background) pixel. Once these steps are completed, a marker-controlled watershed segmentation is performed by labelling the regions that can be considered foreground with high confidence (Sonka et al., 2015). Connected component analysis using 8-connectivity is performed on these regions to label them as individual onions. Finally, properties of these regions, such as size and circularity, were assessed before identifying them as identified onions and placing them in the correct size category.

3.2.5. Definition of Vegetable Size Categories

The algorithm was further expanded by sorting the detected vegetables into various size categories allowing producers to quantify the number of crops of a given size type. Samples of shallot onions ranging within four size categories (small, medium, large and extra-large onions) were analyzed by weighing each onion and taking their dimensions with a caliper. These dimensions included an approximate circular diameter measured through the center of the onion, and a major axis and minor axis for an elliptic approximation. Classes for the machine vision algorithm were determined by plotting each of these defining traits versus onion weight. The highest correlation was found between the minor axis parameter and weight (R^2 = 0.93). **Figure 3.10** shows the minor axis values for onions sorted by the conventional method (legend) and the thresholds established after classifying the data using the model. A summary of these classes is shown in **Table 3.3.**

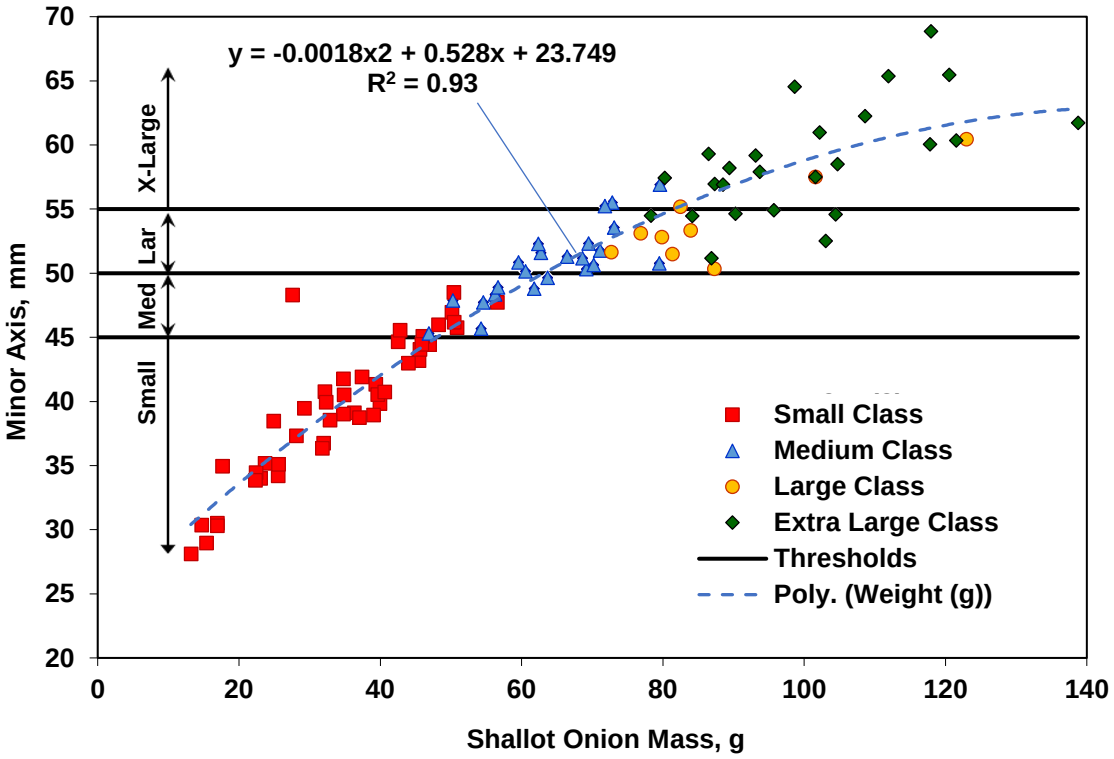


Figure 3.10. Determination of size classes for shallot onion classification.

Table 3.3. Shallot onion classes defined for the computer vision algorithm.

Size Class	Small	Medium	Large	Extra Large
Minor axis range (mm)	25 - 45	45 - 50	50 - 55	> 55

3.2.6. Size Calibration

Once size classes were defined, a method was developed to automatically calculate the size of the onions in the images collected by the yield monitor. Previous research has made use of systems integrating different types of imagery such as thermal (Stajnko et al., 2004) or stereo-vision (Mirbod et al., 2016) to do this. Others have used sophisticated time-of-flight-based 3D cameras (Gongal, et al., 2016) to perform size determination and precision mapping of fruits in 3D space. However, in this study, the requirement was to characterize onion size by generalizing values within 3 to 4 size categories. Therefore, a simple size calibration method was chosen. Real vegetable size can be estimated by calculating the area of the pixels occupied by the vegetable itself within the image and then directly correlating it to their real-world dimensions by using a reference object of known size (Al-Ohali, 2011; Stajnko et al., 2009). The reference object is isolated and measured in each image, and then a suitable pixel to metric ratio (P_m) is determined. P_m is defined by taking the ratio of a pixel distance (P_d) and the true value of this same distance (T_d) in a real-life metric unit of choice.

$$P_m = {}^{P_d}/_{T_d} \tag{3.7}$$

It is important that the dimensions of the reference object remain known and that it is easy to identify and segment from the image. To facilitate size calibration, this object could be placed in the same location in every frame. A total number of 35 onions were analyzed by comparing their true size measured with a caliper with that predicted by the algorithm. A standard 300 mm ruler was used as the reference object for this test (**Figure C-1** of Appendix C). Results showed a very high correlation (R^2 = 0.94) between the predicted results and the true size values, and a standard error (SE) was 2.33 mm (**Figure 3.11**). Therefore, this method was deemed suitable for the prototype during the in-field trials.

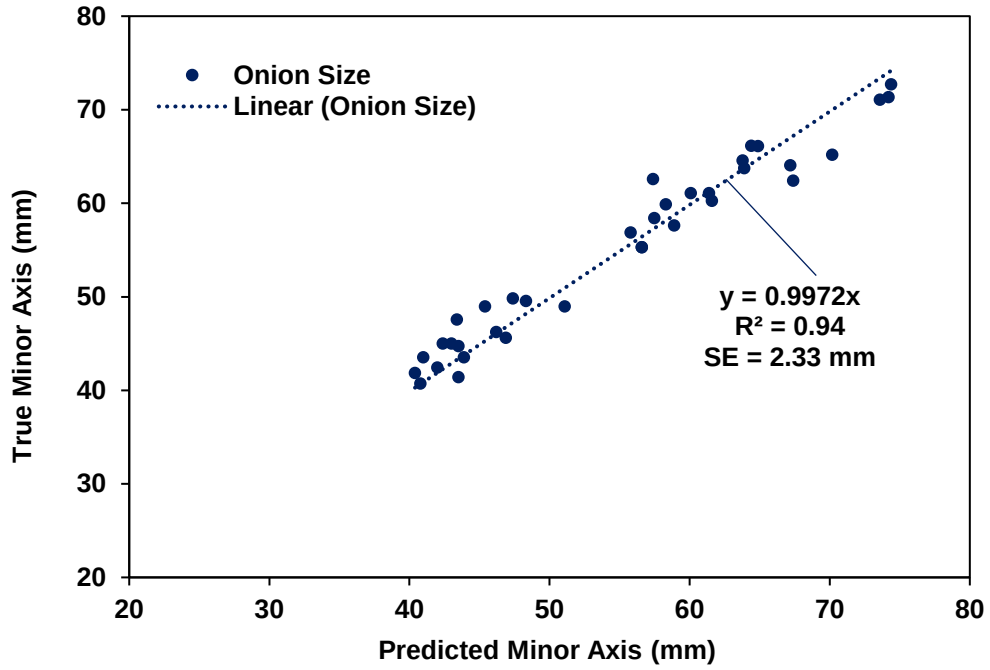


Figure 3.11. Shallot onion size calibration results.

3.2.7. Statistical Analysis

A two sample Kolmogorov-Smirnov (KS) test was applied to determine whether the predicted diameter values for each size category would differ significantly from the manually measured values. The KS test statistic is defined as the maximum distance D between the two curves and is given by:

$$D = \max((F(Y_i) - \frac{i-1}{N}, \frac{i}{N} - F(Y_i)) \quad 1 \leq i \leq N \tag{3.8}$$

F is the theoretical cumulative distribution being tested and Y_i is a given observation within the total number of observations N. To use the KS-test, the distribution must be continuous and fully specified. A two-tailed test was performed with the following conditions:

$$H_0 : The\ two\ samples\ are\ drawn\ from\ the\ same\ distribution.$$
$$H_1 : The\ two\ samples\ are\ not\ drawn\ from\ the\ same\ distribution. \tag{3.9}$$

The KS statistic was computed using the two-sided asymptotic KS distribution.

Chapter 4. Results and Discussion

4.1. General System Performance

Design of the yield monitoring system was completed under a set of constraints defined by both the partner organisation and the research team. The primary constraints were the following: (1) it needed to be easily integrated into the operator's daily practices, (2) inexpensive, (3) relatively easy to assemble and dismount from the harvester, (4) reliable and most importantly, (5) it had to provide an accurate prediction of vegetable size. In the following sections, we assess the performance of the system in terms of these initial constraints. Size prediction performance will be discussed in **section 4.3** together with the performance of the final detection algorithm.

4.1.1. Integrability

The system was designed to be easily operated by anyone within the tractor. To run the yield monitoring program, the operator simply had to log into the computer, open a terminal and run the following line of code:

```
python main.py -c conf.json
```

This command initialized a python console and ran the yield monitoring program using a preset JSON configuration file. In this configuration file, constants related to the settings of the system were stored separately and called by name in the program. Such settings included the baud rate of the GPS sensor, the path where the collected images would be stored on the external drive and the location of the spreadsheet containing the yield predictions for all images in the data set.

4.1.2. System Cost

Table 4.1 lists the components used to build the system and their respective suppliers and prices. The total price of the yield monitoring system was $1,212 CAD (excluding manufacturing and assembly costs), proving to be far less expensive than most farm equipment.

Table 4.1. Price Breakdown of the machine vision yield monitor by component.

Component	Function	Manufacturer	Price ($CAD)
MINIX NEO Z83-4	Computer Vision Analysis	MINIX	240
SAMSUNG T5 External SSD 250GB	Image Storage	Samsung	149
Garmin GPS 19x HVS NMEA 0183	GPS receiver	Garmin	302
ELP Varifocal USB camera	Image Acquisition	ELP	124
Wireless Keyboard	Command Entry	Logitech	50
Display Monitor	Display	Eyoyo	97
USB Camera Mount	Structural	Wasserstein	20
Seahorse 300Protective Case	Structural	Seahorse	34
USB wires	Data Transfer	UGREEN	16
HDMI Cable	Data Transfer	UGREEN	15
Black 20CM FPV Flat HDMI Cable	Data Transfer	Permanent	21
USB Connectors	Electrical Component	Conec	26
8 Pin Connector and Receptacle	Electrical Component	Conxwall	32
HDMI Weathertight Connector	Electrical Component	Switchcraft	18
12-V Power Supply Sockets	Electrical Component	Foxnovo	42
Dual Row Terminal Block x2	Electrical Component	Philmore-Datak	7
Silicone Insulated Wire	Electrical Component	N/A	8
RS-232 to USB adapter	Signal Converter	N/A	10
Total Cost			**1212**

4.1.3. System Assembly

Assembly of the system was feasible in less than one hour. During this time, the USB camera was fixed on the bracket of the harvester using locknuts, bolts and washers. Once this was completed, the next step was to connect the camera to the processor using two USB extension plugs. The wires were then fixed onto the harvester and tractor using zip ties and were run to the driver's cab where all other hardware was located. The 12-V power source was connected to the control box and powered all the devices including the GPS sensor. The USB webcam was the only component that was powered by the computer itself since it required a 5-V DC input. Although assembly of the system was somewhat time consuming, in practice, this task

would be completed only once per season: the device is mounted on the tractor at the beginning of the harvesting season and then provides continuous data collection until the end of harvest. The system is only dismounted once the data collection has been completed.

4.1.4. Reliability

One disadvantage of the system was the occasional disconnection of wires which would unfortunately lead to a complete power loss. Under these circumstances, data collection would come to a halt and the system would have to be rebooted. This may have been due to the type of connectors used (12-V cigarette lighter adapters), which tended to move around in the sockets and eventually disconnect while the tractor was running. To overcome this, special care needed to be taken to ensure that all components were properly fixed during operation. To improve the system, the next iteration would need to include more stable connectors, especially those originating from the main power supply; ideally, these would have a type of security lock. This would provide a constant stable connection resulting in a more reliable system.

4.2. Results of Feasibility Study

4.2.1. Segmentation Results

The aim of the feasibility study was to determine whether onion detection and count could accurately be done using a CV software. To properly isolate individual onions and count them, an appropriate segmentation needed to be attained. The first segmentation method tried involved converting the image to the HSV color space, and then obtaining a histogram of the hue distribution. The second segmentation method was Otsu's method, described previously by equation 3.6.

Figure 4.1 shows a hue channel histogram of a sample image from the dataset. Since the range 0 to 360 cannot be represented using only 8-bit integers in OpenCV, hue values range from

0 to 180. The high bin count at the value 121 corresponds to a shade of blue violet, representing the surface behind the conveyor belt. The two modes in the histogram are located between 0-30 (left region) and 90-120 (right region). Otsu's method determines an average threshold of 60, situated roughly in the middle of the two peaks, where the left region represents the shallot onions and the right region represents the light-green portions of the paddle and conveyor. The manually determined threshold combined the left hue peak and the hue values from 165-180.

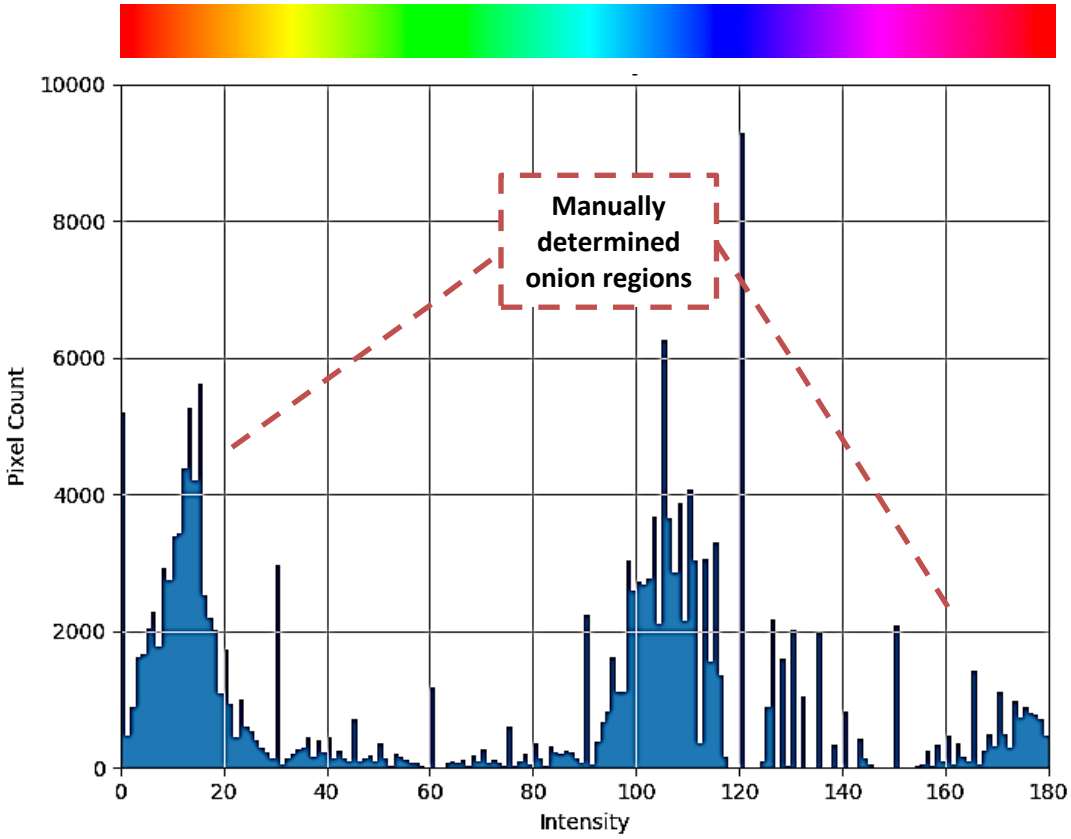
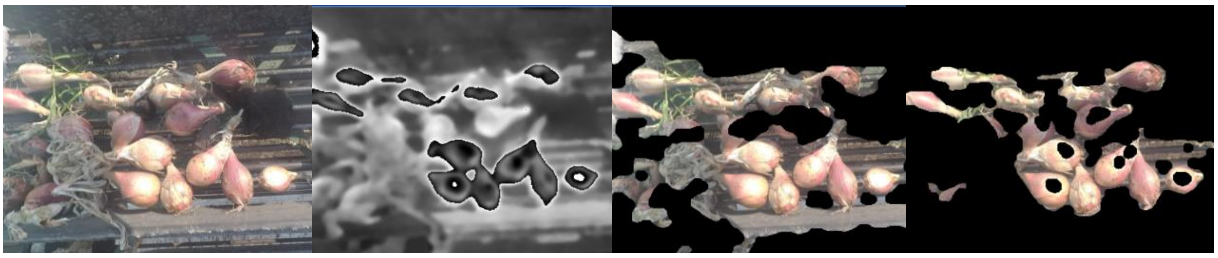


Figure 4.1. Hue intensity distribution of a sample image in the feasibility study. See electronic version for colors.

An original image of the onions on the conveyor and the results obtained from the various segmentation methods is shown in **Figure 4.2**. A red color intensity texture image (**Figure 4.2b**) was used to further extract regions that were high in red chroma after the initial global thresholding. In most cases, Otsu's method (**Figure 4.2c**) led to over segmentation, capturing not

only the onion regions but also much of the conveyor system and stems. This may have been due to the varying number of onion pixels in each image which affected the histogram distributions, and in some cases, made it unimodal when onion counts were unusually low. Combining the HSV color threshold with the red texture image reduces the number of false positives as the algorithm first checks to determine if the object is within the proper color range, and then analyses the image further and searches for a distinct red intensity.



(a) **(b)** **(c)** **(d)**

Figure 4.2. Segmentation results.

Original conveyor image (a) and results of the image processing and segmentation algorithm. (b) is the red color intensity image, (c) is the image segmented using Otsu's method and (d) is the segmentation performed using the manually selected hue and saturation thresholds.

4.2.2. Onion Detection Performance

To assess onion detection performance of the feasibility study algorithm, 34 random screenshots were gathered to compare the algorithm's number of detected shallot onions and the true shallot onion count performed by manual observation. Examples of these screenshots are shown in **Figure 4.3**. According to the results, the general onion detection rate for the feasibility study proved to be relatively low: the mean number of manually counted onions was equal to 16.1, ranging from 4 to 37 onions in observed examples with a standard deviation of 6.16. The mean from the automatically detected onions (i.e. algorithm) was 7.17, with ranges between 1 and 15 correctly observed onions and a standard deviation of 3.10. Performance of the algorithm is summarized in **Figure 4.4** (top). Although the algorithm underestimated the true crop load

(regression slope of 0.45), it is important to note that there was a high correlation between the manual count and algorithm count, with an R^2 value of 0.71.

(a) **(b)** **(c)**

Figure 4.3. Onion detection results. See electronic version for colors.

Onions identified by the algorithm are located on the image using ellipses. Colors represent size ranges. In this case, blue corresponds to small and green to medium sized onions. Size estimation accuracy was not evaluated in this stage of the study.

By taking the automatically determined onion count and dividing it by 0.45, this increased the efficiency of predictions dramatically, raising the detection regression slope to 0.99 (**Figure 4.4**, bottom). **Figure 4.5** shows a stacked column graph of the detected shallot onions by the algorithm and the difference between the automatic and true manual counts. The total number of onions in the frame is given by adding the undetected onions and correctly detected onions together. After careful analysis, it was noted that under low onion count the algorithm showed better results, missing at most 3 onions per frame. The number of falsely detected onions remained between 0 and 7 in all cases. High detection rates occurred when the onions were not clustered together or superimposed which caused them to be segmented as a single object and thereafter making them difficult to isolate. The algorithm also missed onions primarily located in shadowy regions, as well as bulbs that were hidden by onion stems and occluded other vegetables.

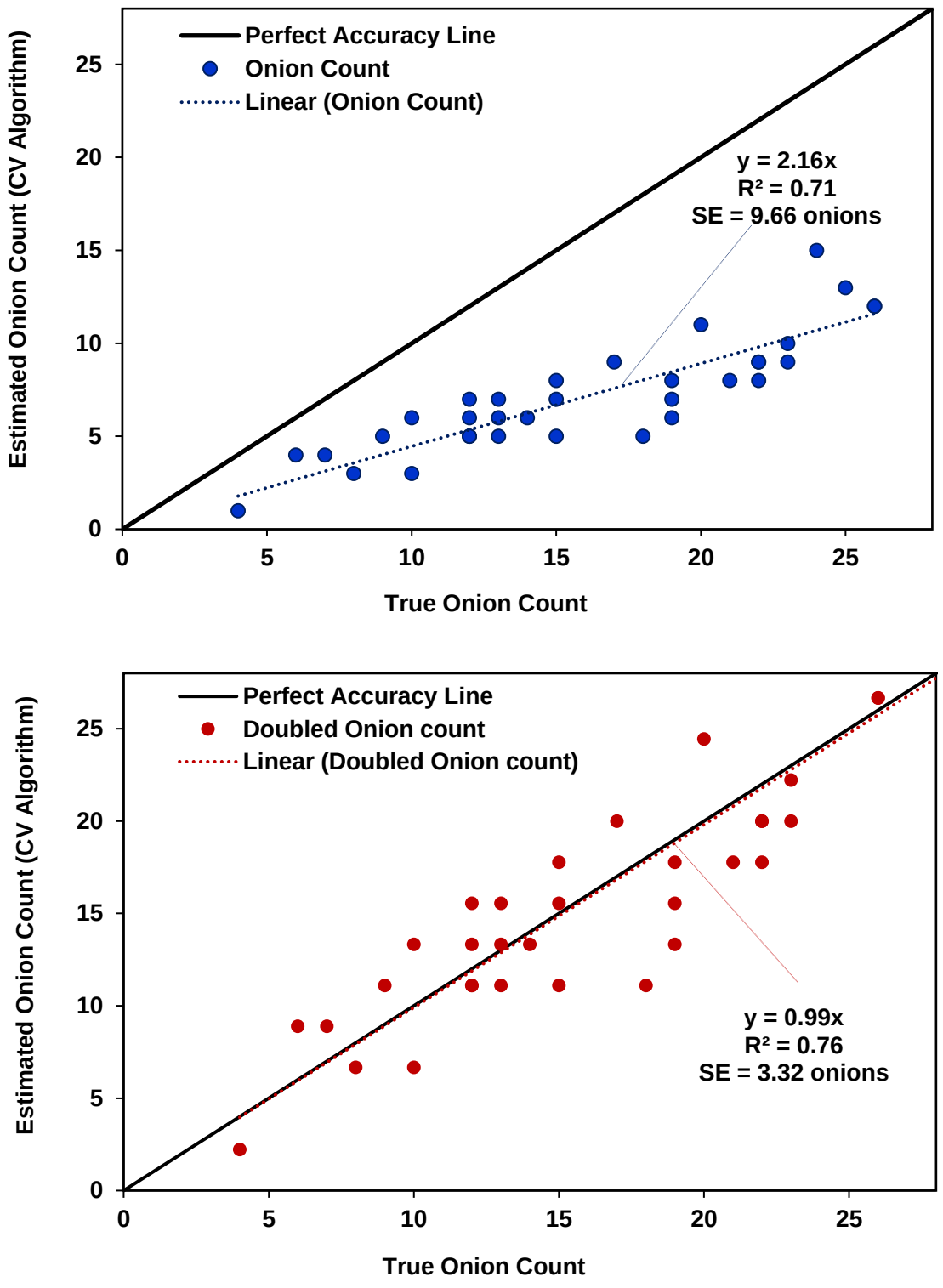


Figure 4.4. Onion detection accuracy of the current machine vision algorithm (top) and accuracy obtained by doubling the output (bottom).

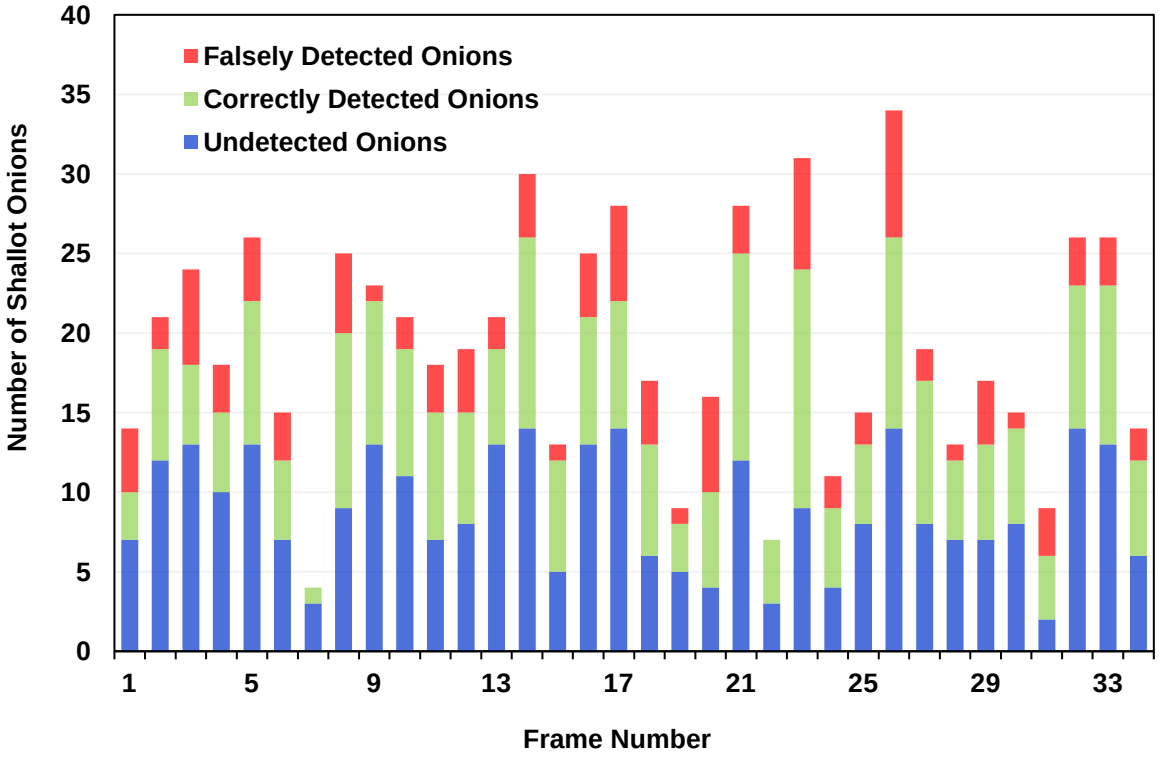


Figure 4.5. Shallot onions detected by the computer vision algorithm vs. manual count.

Other falsely detected onions corresponded to bulbs that were detected twice, which was caused by improper boundary definition by edge detection. However, false detection was relatively low (3 onions) and the average deviation between the algorithm and manual count was 8.88.

4.2.1. Conclusions of Feasibility Study

The goal of this study was to address the processes described in the feasibility study section of **Figure 3.3** and to use image processing techniques to develop an initial algorithm for automatic onion detection. The main challenges encountered were primarily uneven segmentation caused by either inconsistent lighting and occasional occlusion by stems or other shallot onions. This created an incentive to rework the algorithm and improve detection for the

final field testing. A fully functional system was therefore created and added to the machine vision bracket to perform simultaneous image analysis and position tracking. Moreover, size estimation using the method described in **section 3.2.5** was integrated and evaluated.

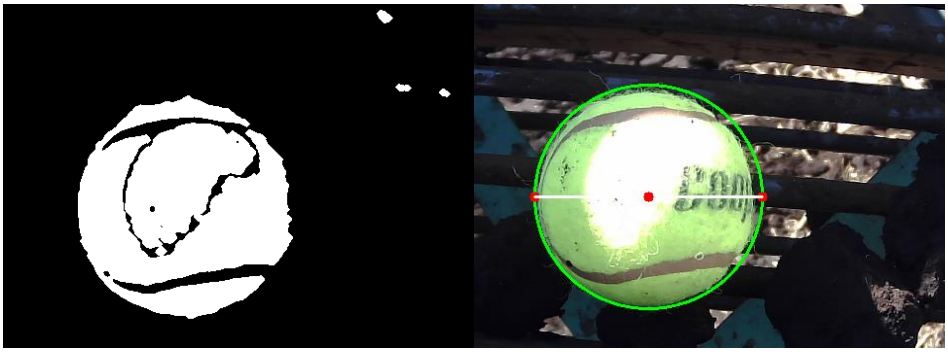
4.3. Prototype Performance

The system was mounted on a commercial shallot onion harvester (Univerco Inc., Napierville, Quebec, Canada) located on a farm in Napierville, Quebec. Images, onion counts, and geographical coordinates were recorded and saved to a file. These results are reported and analyzed in the following sections.

4.3.1. Size Estimation

A tennis ball was selected as the calibration object for the field trial due to its very distinct bright yellow color which could easily be segmented using color thresholding in the field. **Figure 4.6** shows the results from this segmentation (a) and the detection of the ball in a sample image from the conveyor (b). **Sections 3.2.5** and **3.2.6** describe the approach used to extract the size information of the onions from the images following the segmentation step. OpenCV approximates the ball using a CHT and extracts its diameter in pixel length. The value of P_m is determined using the diameter of the ball which was equal to 65.4 mm. For the field trial, the recorded P_m was roughly 3.38 pixels/mm per image. In the initial version of the software, an elliptical approximation was set to characterize the shape of the shallot onions. However, after careful examination, it was noted that this approximation gave poor results when predicting the true size distribution of the onions. Almost all onions detected were labelled as being small which was incorrect. A circular approximation was later selected and the new model of onion diameter vs. mass is depicted in **Figure C-2** of Appendix C. In the univariate quadratic polynomial that is fitted to the new model, the coefficient of the second-degree variable is of a larger magnitude, making the curve of the

parabola sharper than the similar curve in **Figure 3.10**. The number of size categories was then decreased to 3 instead of 4 (small, medium and large). Parts of the medium and large class are merged, and the new thresholds are depicted in **Figure C-2.**



(a) **(b)**

Figure 4.6. Color thresholded result of the calibration object (a) and detection of the ball in the original image (rb). The diameter of the ball in pixel length is determined using a CHT.

Within the dataset, a total of 92 images were randomly chosen to assess the performance of the size prediction method. Within these images, 271 onions were analyzed by measuring the diameter of the automatically detected boundary of the bulb and comparing it with the true boundary of the onion. Results showed that the overall performance of this algorithm was very poor (R^2 =0.011) with 55.9% of onions correctly classified and a Root Mean Square Error (RMSE) of 11.3 mm. **Figure 4.7** shows the results obtained after performing a linear regression for each of the 3 size categories. The lowest detection rate was found within the medium onion category (44.4%). Size predictions of the onions within the small class were reasonably better with 88 correct predictions out of 150 small onions observed (58.6%). The best size classification performance was within the large class (73.3%). The mean difference between the predicted and true values of the medium class (1.68 mm) was lower in absolute value than that of both the small (6.86 mm) and large classes (2.06 mm).

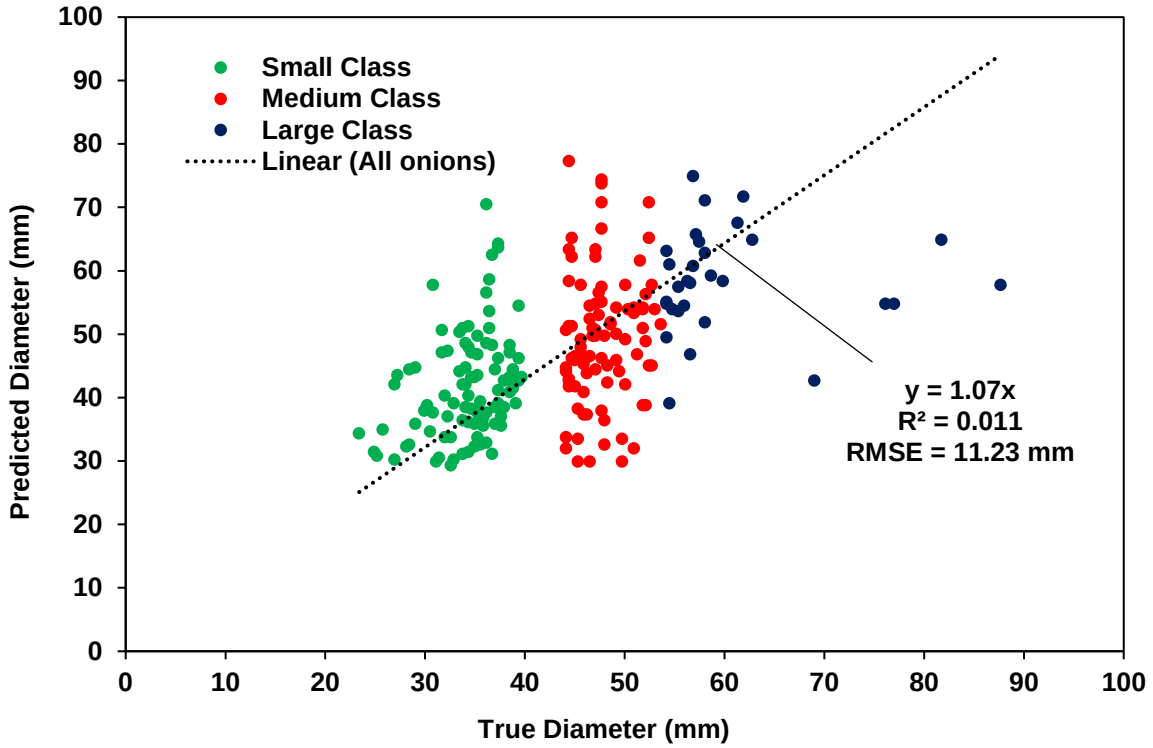


Figure 4.7. Manually determined onions sizes vs. predicted onion sizes from WST algorithm.

Following are results of the two sample KS test which was applied to determine whether the predicted diameter values for each size category differed significantly from the manually measured values. Cumulative fraction plots were created for each pair of observations to visualize their respective distributions (**Figure 4.8**). The KS statistic was computed using the two-sided asymptotic KS distribution. If the KS statistic was small or the given p-value was shown to be high, then the null hypothesis could not be rejected. Results from the statistical analysis for each size class are tabulated in **Table 4.2**, **Table 4.3** and **Table 4.4.** According to these results, the null hypothesis is rejected for both the small and medium classes but remains true for the large class with a confidence interval of $\alpha = 0.05$.

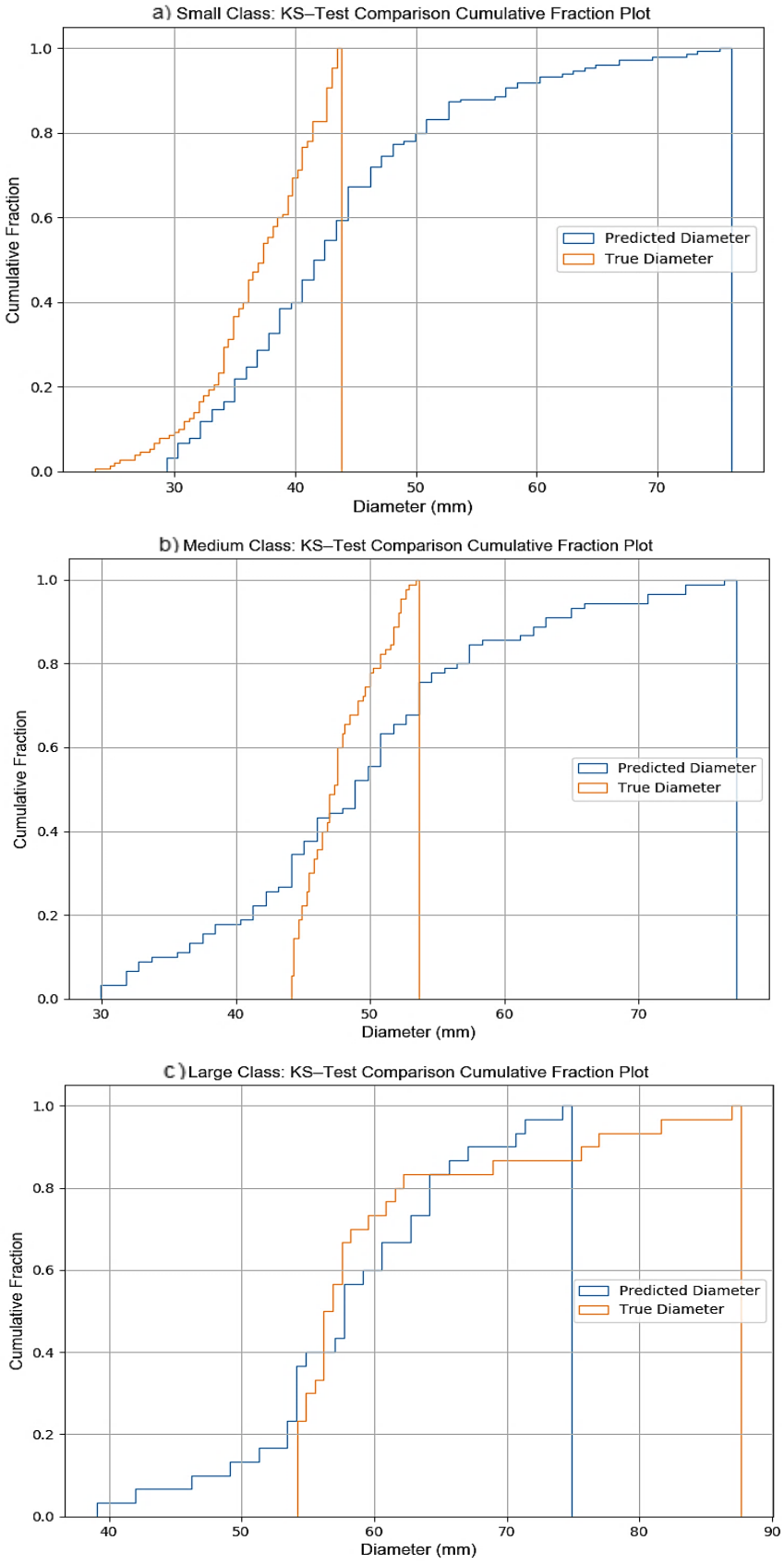


Figure 4.8. Cumulative fraction plots of the predicted diameter and true diameter distributions.

Table 4.2. Kolmogorov-Smirnov test results for the small class

	Diameter (Predicted)	Diameter (True)
Mean (mm) (N = 150)	43.87	37.00
Median (mm)	42.50	37.32
Min (mm)	29.32	23.40
Max (mm)	76.12	43.83
Standard Deviation (mm)	10.41	1.62
Mean Difference (mm)	6.86	
KS Test Statistic	0.4199	
P-Value	2.849E-12	

Table 4.3. Kolmogorov-Smirnov test results for the medium class

	Diameter (Predicted)	Diameter (True)
Mean (mm) (N = 90)	49.46	47.78
Median (mm)	49.76	47.39
Min (mm)	29.91	44.13
Max (mm)	77.30	53.61
Standard Deviation (mm)	10.44	2.75
Mean Difference (mm)	1.68	
KS Test Statistic	0.3222	
P-Value	1.191E-4	

Table 4.4. Kolmogorov-Smirnov test results for the large class

	Diameter (Predicted)	Diameter (True)
Mean (mm) (N = 30)	58.44	60.51
Median (mm)	58.20	57.01
Min (mm)	39.09	54.20
Max (mm)	74.93	87.67
Standard Deviation (mm)	8.12	8.79
Mean Difference (mm)	-2.06	
KS Test Statistic	0.2333	
P-Value	0.3420	

Finally, the onion size distribution obtained from the algorithm was compared to that calculated after manual sorting of 15 shallot onion rows originating from the same shallot onion field. **Figure 4.9** shows the final output percentages for each size class. Predictions remained accurate for the small and medium classes with a 0.41% and 4.19% difference from the observed true percentages for the small class and medium class, respectively. However, there was a strong deviation in results obtained from the extra-small, large and extra-large class (26.97%, 58.46% and 66.14%, respectively).

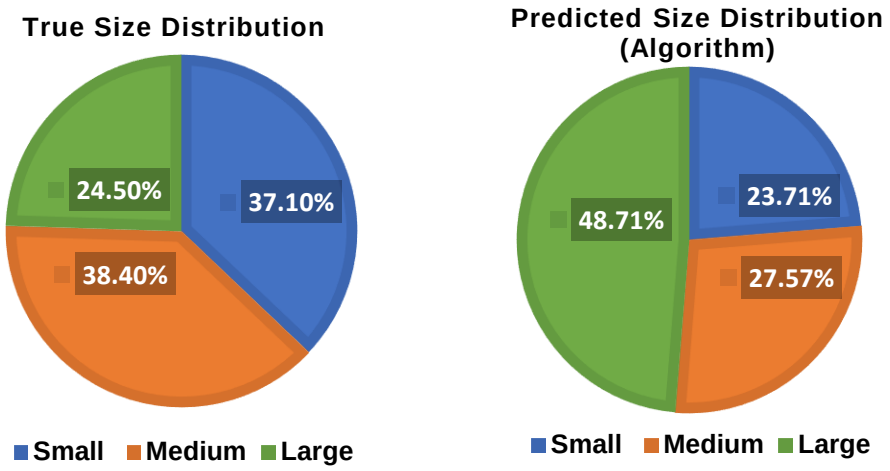


Figure 4.9. Size class distributions for the manual sorting vs. computer vision algorithm.

4.3.2. Segmentation Results

Figure 4.10 illustrates the pipeline used to identify the onions and calculate their sizes. Images were subject to variable lighting conditions, which affected the performance of segmentation in some cases due to the presence of either shadowy or very bright regions. The various preprocessing steps involved blurring the original image (**Figure 4.10a**) with a median filter (9x9) to remove speckle noise, then blurring it once more with a 9x9 Gaussian filter (**Figure 4.10b**). The image was converted to the HSV color space and first segmented using color thresholding. Color ROIs included the onion skin and areas on the onion that exhibited specular reflection (**Figure 4.10c**). Morphological operations of opening (**Figure 4.10d**) and closing (**Figure 4.10e**) were applied to refine the onion regions and remove noise from the objects of interest. Segmentation using a marker-based watershed transform (WST) was performed using the distance transform of each image **Figure 4.10f**), and these regions were individually labelled (**Figure 4.10g**). Small onions were identified using blue contours, medium onions with green contours and large onions with red contours (**Figure 4.10h**).

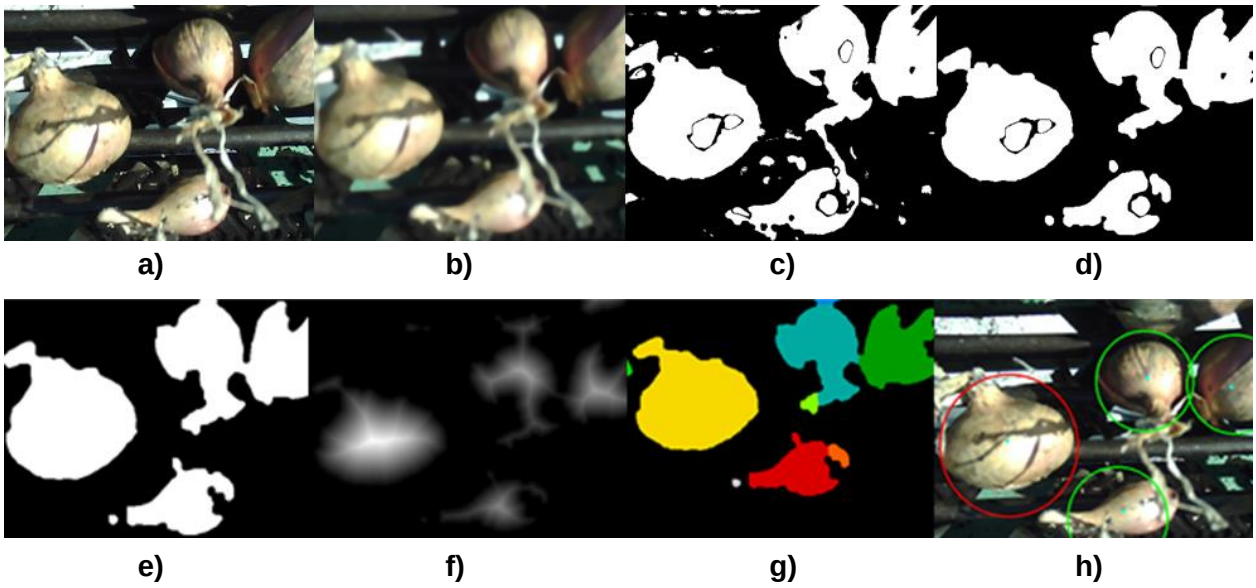


Figure 4.10. Segmentation Results.

The original image (a) is first preprocessed by blurring using a 9x9 median filter and 9x9 Gaussian filter (b). The image is then converted to the HSV color space and thresholded using a predetermined threshold value (c). Morphological operations of opening (d) and closing are applied (e), and the distance transform (f) is computed. Watershed segmentation is performed on the image to isolate individual onion regions (g) and identify and classify them in the original image (h).

4.3.3. Onion Detection Results

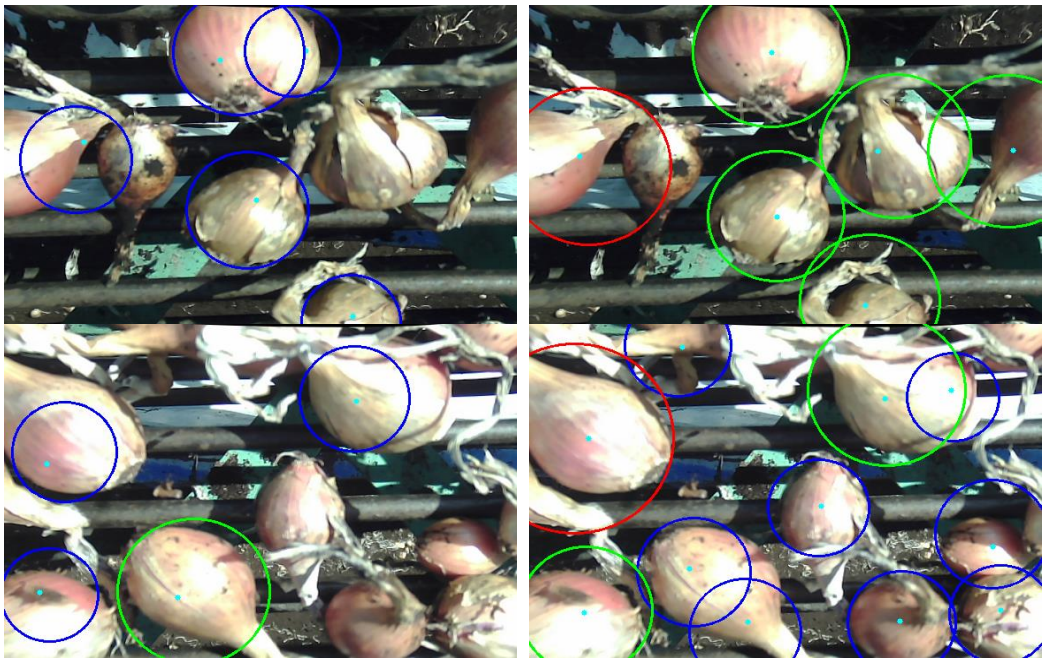
A summary of the final detection results obtained for both methods is reported in Table **4.5**. Initially, the feasibility study algorithm was used to detect and count the shallot onions during harvesting. Out of 1180 images collected during the trial, a subset of 246 images was selected to evaluate the performance of the algorithm. In these 246 images, 1667 onions were manually identified. The initial algorithm managed to detect a total of 713 onions, within these, 597 were true onions (83.7%) and 116 were false detections. However, the total number of correctly detected onions was relatively low compared to the true number of onions in the dataset (35.8%). With the WST method, the total number of onions detected increased to 1467, and a total of 1115 of these detections corresponded to true onions (76.0%). There were 1782 onions manually

identified in the dataset. The total number of large detections also increased drastically with the WST method (from 12 to 390).

Table 4.5. Summary of shallot onion detection results

Method	Total Detected Onions	Correctly Detected Onions	Correctly Detected Onions (%)	Mean (per image)	Standard Deviation	Standard Error (Actual vs. Correctly Detected)
Initial	713	596	35.8	2.90	1.96	2.47
Watershed	1467	1115	62.6	5.64	2.99	1.98

Figure 4.11 shows screenshots of results obtained by the initial algorithm (a) and WST algorithm (b). The WST segmentation method failed mostly when there were large reflective areas present on the onion which would appear almost entirely white and that were not properly captured by color thresholding. Bright spots in some images caused by inconsistent lighting also led to false detections. In some extreme cases, the image was overly saturated making the onion regions appear almost uniform. This would leave large holes within the onion which were not filled even after noise removal with opening/closing, preventing the allocation of a single minimum to that specific region. As with the initial method, the conveyor background would sometimes mistakenly be identified as shallots and some onions would also be detected twice. As in the feasibility study, onions missed by the initial algorithm were, for most cases, onions that were partially visible and on the border of the image, onions occluded by larger bulbs or stems, or bulbs that were present in shadowy regions. False detections corresponded to onions clustered in the trailer that were visible through the conveyor paddles or onions that were improperly segmented causing the same bulb to be identified twice. Detection did increase with the adaptation of the WST method. However, this change also led to an increase in the number of false detections which can also been seen in **Figure 4.12** and **Figure 4.13**.



(a) Initial Algorithm **(b) WST Segmentation**

Figure 4.11. Onion detection results. (a) shows final detection results for the initial algorithm, and (b) shows results from the WST segmentation method. See electronic version for colours.

Linear regression was performed on the total number of onions detected by each method vs. the actual (true manually detected) onion count. Looking at the two accuracy plots for both the original method (**Figure 4.12**, top) and the watershed method (**Figure 4.12**, bottom), the coefficient of determination obtained for the initial method was low (R^2 = 0.33). The slope of the trendline was also low, showing that the original algorithm could only detect about 41% of onions present. As count remained low, the predictions were near the perfect accuracy line but as count gradually increased beyond 4 onions, the algorithm began to miss more vegetables. Possibly, this was due to the improper segmentation of onions that overlap when the image is more cluttered. For the WST method, the estimated count has significantly improved and the correlation between the automated count and true count was much higher (R^2 =0.49). One downfall of this was the increase of false detections from an average of 0.48 onions per image to 1.36 onions per image (**Figure 4.13**).

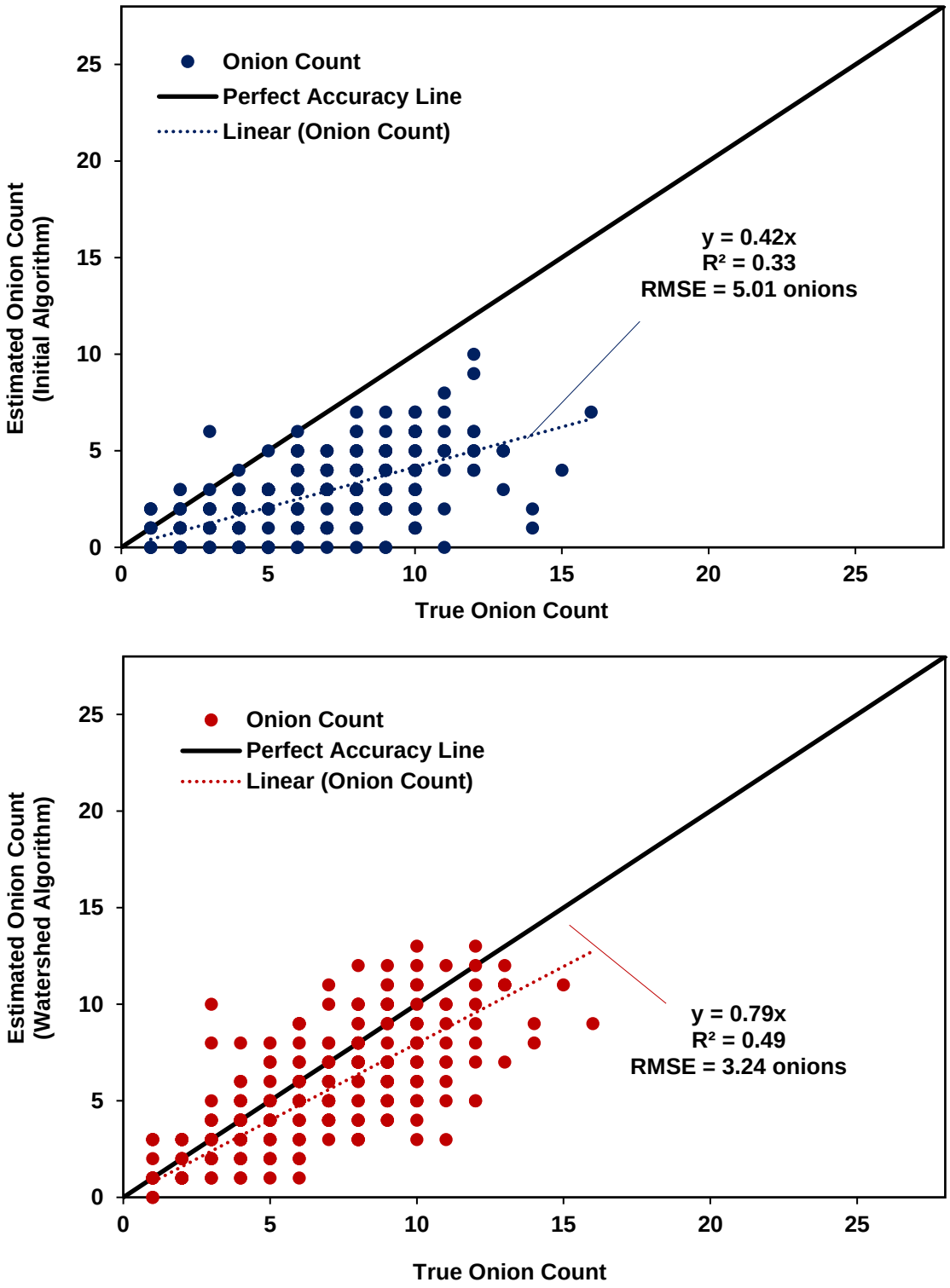


Figure 4.12. Shallot onions detected by the computer vision algorithm vs. manual count. Top shows results for the initial algorithm developed during the feasibility study, and bottom shows the improved WST segmentation algorithm results.

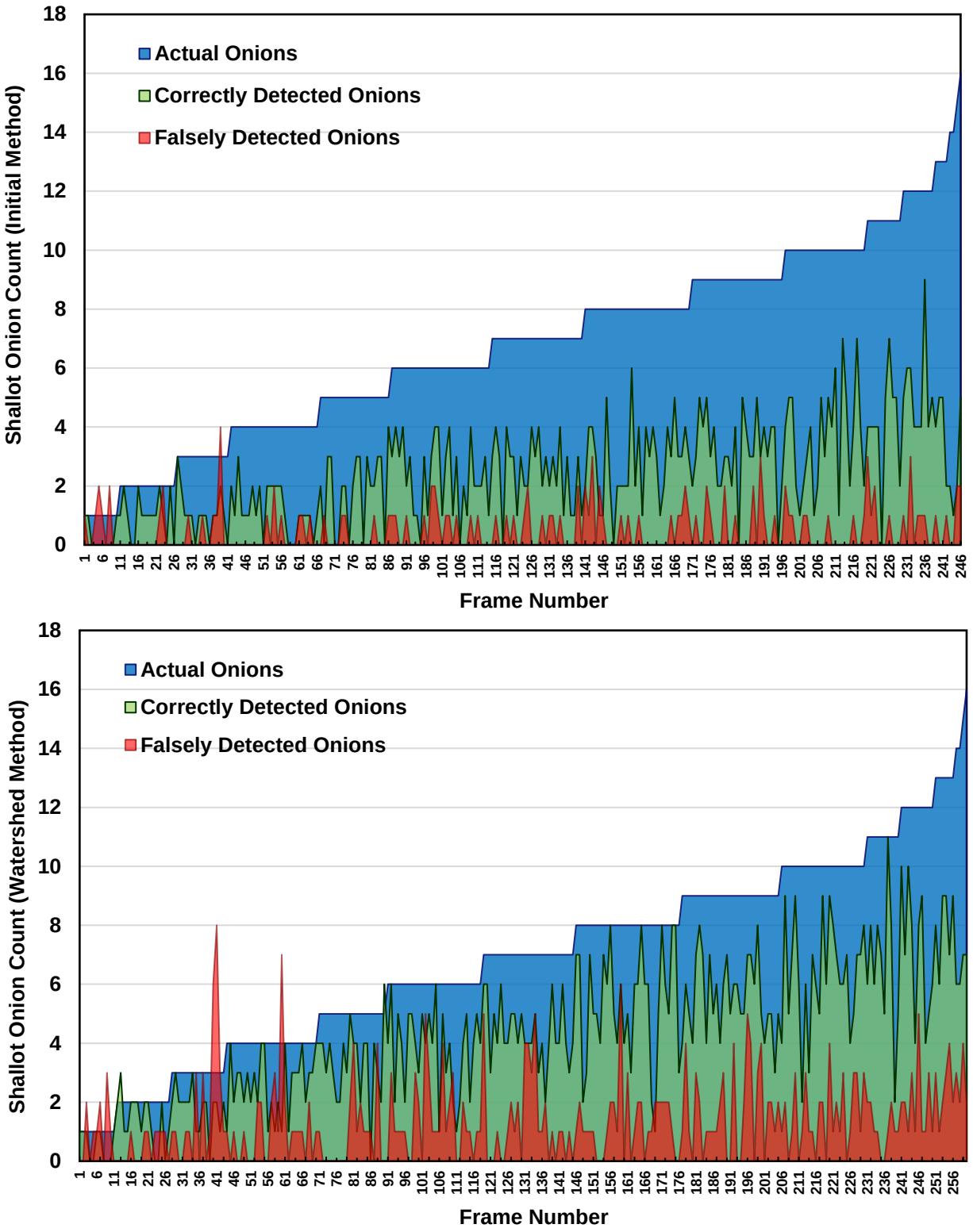


Figure 4.13. Shallot onions detected by the computer vision algorithm vs. manual count. Top shows results for the initial algorithm developed during the feasibility study, and bottom shows the improved WST segmentation algorithm results.

The error was modelled to compare both results and select the method that performed the best. To do this, the true positive (TP), false positive (FP) and false negative (FN) counts were calculated from the visual observations. TPs corresponded to onions that were correctly detected, FPs were other objects (background, stems, rocks) that were falsely classified as onions, and FNs were onions that were missed by the detection algorithms. Results of this analysis are featured in **Table 4.6** and **Table 4.7**.

Table 4.6. Summary of Type I and Type II error distribution for the Initial Method.

	Type I Error (Falsely Detected Onions)	Type II Error (Missed Onions)
Total Sum	117	962
Mean (per frame)	0.475	3.91
Median	0	4
Standard Deviation	0.738	2.50
Max	4	13
Min	0	0

Table 4.7. Summary of Type I and Type II error distribution for the Watershed Method.

	Type I Error (Falsely Detected Onions)	Type II Error (Missed Onions)
Total Sum	352	418
Mean (per frame)	1.35	1.61
Median	1	1
Standard Deviation	1.41	1.81
Max	8	8
Min	0	0

Results show that detection did improve with the adoption of the WST method. The total number of false detections and missed onions in the initial method (1079) decreased to 770 using the WST algorithm, although at the expense of a higher false detection rate. From **Table 4.8**, we can see that adopting the Watershed Method allows for an increase in overall performance, although it does cause a small decrease in precision. Stricter analysis will need to be done to develop an improved algorithm that can help improve the resistance to type I error.

Table 4.8. Summary of detection performance metrics

	Initial Method	Watershed Method
Precision	0.836	0.760
Recall	0.383	0.727
Overall Accuracy	0.356	0.592
Standard Deviation	1.41	1.81

4.3.4. Yield Map

The system was tested on a portion of the onion field equivalent to two full rows of shallot onion crops. Issues related to the power supply caused data collection to halt midway. However, enough data was collected create a map showing the distributions of the 3 size categories. The full study area is pictured in **Figure 4.14**.

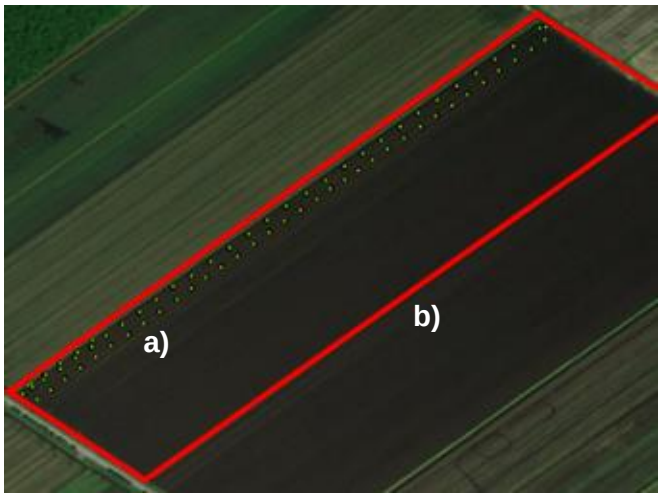


Figure 4.14. Image showing the sampling points collected on the shallot onion field (a) and the boundary of the entire study area (b) .

A total of 871 data points was collected during the field trial. A median filter was applied to the dataset by taking the median of the onion count values at every 10 points for each size category, leaving a total of 88 points for mapping that were separated by an average distance of 23 m. Inverse distance weighting (IDW) was selected as the spatial interpolation method with a weight power of 2 and search radius of 5. Although a small portion of the field was mapped, the three maps did show similar trends (**Figure 4.15**). For example, the north side of the field reported a high yield value for all three size classes. The edges of the field correctly report a low onion count, which reflects the images on the sides of the field that pictured mostly the conveyor and soil. Overall, the spatial variation between adjacent points was very high. Further analysis will need to be done to determine an appropriate sample point density that would give the most accurate representation of the shallot onion field and give more precise yield predictions.

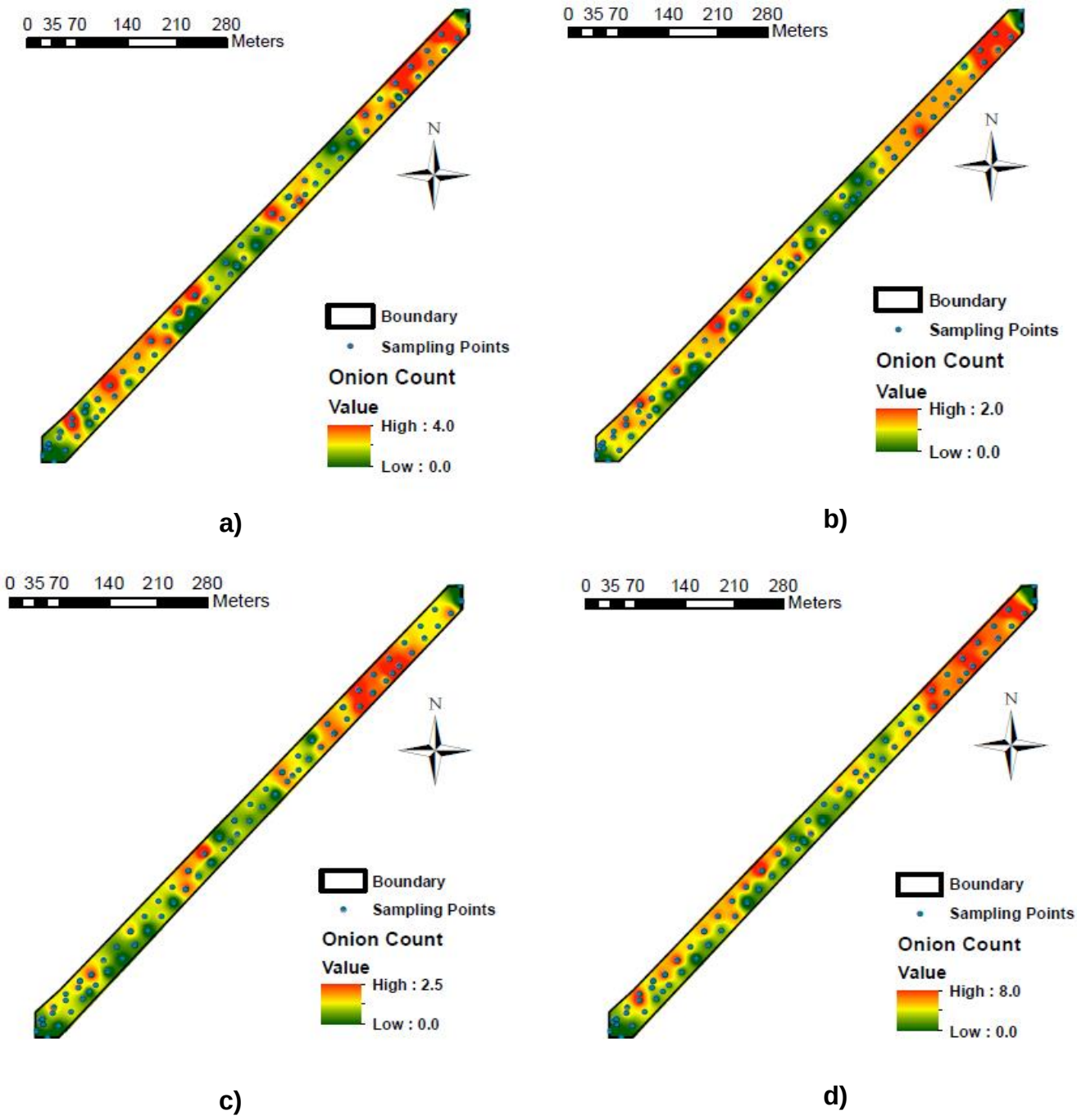


Figure 4.15. Final yield maps for the small (a), medium (b), large (c) onion classes and the total count (d) of the shallot onion field.

4.4. Future Improvements

Although the system was able to gather images and perform yield estimation on the go during the harvesting process, there were some drawbacks during operation. One of these drawbacks was related to the method of supplying power to the system. The yield monitor received power directly from the tractor using a 12-V outlet. This outlet was then linked to the control box using a 12-V cigarette lighter socket. Although the 12-V sockets did manage to supply enough power to the box, the connector would occasionally detach from the cigarette lighter receptacles, therefore, cutting off the power source for the entire system. Special care needed to be taken to ensure that the connector would stay fixed within the socket during operation. A solution to this could be the replacement of the cigarette lighter receptacles with lock tight connectors for power and ground like those used for the GPS sensor, creating a more stable and secure connection that could better withstand movement in the driver's cab.

Another disadvantage of the system was related to the effectiveness of the size calibration process. Once the yield monitoring program was started, calibration images had to be taken by placing a tennis ball in front of the camera and recording a small set of frames that would later be used to establish a pixel metric. However, the thresholds for the color segmentation of the tennis ball were set using trial and error with previous images from a different run. Therefore, the segmentation could sometimes be faulty depending on the existing lighting conditions of the following run. A threshold setting method capable of being adjusted on the field would allow a bit of play when establishing the color thresholds for segmentation. Another option could be to use the distance between paddles to calibrate the size estimation algorithm, although this would require the development of a new method to segment the paddles from the image.

Onions missed by the detection algorithm were, for most cases, onions that were partially visible and on the border of the image, onions occluded by larger bulbs or stems, or bulbs that were present in shadowy regions. False detections corresponded to onions clustered in the trailer

that were visible through the conveyor paddles or onions that were improperly segmented causing the same bulb to be identified twice. Further enhancements of the algorithm must be made to better separate individual onion regions and to increase overall accuracy. A way to enhance this could be to develop a more resilient algorithm using a form of machine learning called semantic segmentation. A convolutional neural network structure like that of Bargoti and Underwood (2017) could potentially learn features that could accommodate for all variabilities in the appearance of onions. These variations include examples of onions in clusters, in shaded areas or occluded by stems.

Enhancements to the algorithm will also need to be made to improve overall detection rates while maintaining a low number of false positives. This may be done by increasing the video frame rate and analyzing sequential frames and monitoring them for onions that appear in multiple frames. This may reinforce the possibility of an object being an onion by incorporating a visual tracking algorithm.

Conclusions

Providing quality and quantity assessment of shallot onion crops during harvesting is crucial for securing higher returns and establishing more efficient management practices. This research focused on the use of computer vision as an alternative for yield estimation practices for specialized vegetable crops.

A fully functional system was developed to record image and position data of shallot onion bulbs during harvesting and create a geo-tagged image database for precision yield mapping. Computer software was developed to detect shallot onions in images and determine their sizes. The system was able to properly detect 62.6% of onions in a subsample of the dataset using a Watershed segmentation method. The software also reliably categorized large sized shallots with an accuracy of 73.3% but was limited when predicting small (58.6%) and medium (44.4%) onion sizes. This was primarily due to improper boundary definition of bulbs that were either on the border of the image or occluded by other bulbs or stems. There was also difficulty detecting onions in shadowy regions. Despite this, it did correctly predict the percentages of the medium and small classes when compared to post-harvest data for 15 rows of the examined field.

Hardware limitations will need to be properly revisited to develop a more reliable system that can withstand harsh conditions of the agricultural environment. These would include more stable power connections, a camera that is resistant to high amounts of dust and variable lighting. Developing a new algorithm based on modern machine learning techniques and artificial intelligence may also strengthen detection results. However, this system would most likely require a more powerful processor which will increase its price.

The incorporation of computer vision into agriculture is growing. Although further development is envisioned for this current system, it will help producers manage their harvesting strategies more efficiently. It served as a low-cost initial prototype which managed to provide

insight regarding the feasibility and economic potential of such systems. More care will be taken to produce a second prototype and increase the system's reliability and deliver a better product that could be used in the long term.

References

Agoston, M. K. (2005). *Computer graphics and geometric modelling v.1: Implementation and Algorithms.* USA: Springer. doi:10.1007/b138805

Ahmed2IQ. (2009). [Photograph of a CCD sensor]. *Wikipedia Commons.* Retrieved March 2019, from https://commons.wikimedia.org/wiki/File:CCD_sensor.JPG

Aisenberg, I. (2017, October). Precision farming enables climate-smart agribusiness. (I. F. Corporation, Ed.) *EMCompass, no. 46*.

Retrieved from https://openknowledge.worldbank.org/handle/10986/30372

Al-Ohali, Y. (2011). Computer vision-based date fruit grading system: Design and implementation. *Computer and Information Science, 23*(1), 29-36.

doi: https://doi.org/10.1016/j.jksuci.2010.03.003

Altman, D. G., & Bland, J. M. (2005). Standard deviations and standard errors. *BMJ, 331*(7521), 903. doi: https://dx.doi.org/10.1136%2Fbmj.331.7521.903

Automated Imaging Association. (2014, October). *Computer vision vs. machine vision*. Retrieved from Vision Online: https://www.visiononline.org/vision-resources-details.cfm/visionresources/Computer-Vision-vs-Machine-Vision/content_id/4585

Bac, C., Van Henten, E., Hemming, J., & Edan, Y. (2014). Harvesting robots for high-value crops: State-of-the-art review and challenges ahead. *Journal of Field Robotics, 31*(6), 888-911. doi: https://doi.org/10.1002/rob.21525

Balletti, C., Guerra, F., Tsioukas, V., & Vernier, P. (2014). Calibration of action cameras for photogrammetric purposes. *Sensors, 14*(9), 17471-17490.

doi: https://dx.doi.org/10.3390%2Fs140917471

Baratloo, A., Hosseini, M., Negida, A., & El Ashal, G. (2015). Part 1: Simple Definition and Calculation of Accuracy, Sensitivity and Specificity. *Emergency (Tehran), 3*(2), 48-49.

Bargoti, S., & Underwood, J. (2017). Image segmentation for fruit detection and yield estimation in apple orchards. *Journal of Field Robotics, 34*(6), 1039-1060. doi: https://doi.org/10.1002/rob.21699

Benalia, S., Cubero, S., Prats-Montalbán, J.-M., Bernardi, B., Zimbalatti , G., & Blasco, J. (2016). Computer vision for automatic quality inspection of dried figs (Ficus carica L.) in real-time. *Computers and Electronics in Agriculture, 120*, 17-25. doi: https://doi.org/10.1016/j.compag.2015.11.002

Blok, P., Barth, R., & Berg, W. (2016). Machine vision for a selective broccoli harvesting robot. *IFAC-PapersOnLine, 49*(16), 66-71. doi: https://doi.org/10.1016/j.ifacol.2016.10.013

Buckland, M., & Gey, F. (1994). The relationship between recall and precision. *Journal of the American Society for Information Science, 45*(1), 12-19. doi:https://doi.org/10.1002/(SICI)1097-4571(199401)45:1%3C12::AID-ASI2%3E3.0.CO;2-L

Bulanan, D., Burks, T., & Alchanatis, V. (2010). A multispectral imaging analysis for enhancing citrus fruit detection. *Environmental Control in Biology, 48*(2), 81-91. doi: https://doi.org/10.2525/ecb.48.81

Canny, J. F. (1986). A Computational Approach to Edge Detection. *Institute of Electrical and Electronics Engineers Transactions on Pattern Analysis and Machine Intelligence, 8*(6), 679-698. doi: https://doi.org/10.1109/TPAMI.1986.4767851

Cardenas-Weber, M., Hetzroni, A., & Miles, G. (1991). Machine vision to locate melons and guide robotic harvesting. *American Society of Agricultural Engineers.* St. Joseph, Michigan: American Society of Agricultural Engineers (Paper No. 91-7006).

Cheng, H., Damerow, L., Sun, Y., & Blanke, M. (2017). Early yield prediction using image analysis of apple fruit and tree canopy features with neural networks. *Journal of Imaging, 3*(1), 6-19. doi: https://doi.org/10.3390/jimaging3010006

Datumizer. (2010a). [Photograph of the RGB Cube Showing a Lowgamma Cutout]. *Wikipedia Commons.* Retrieved March 2018, from https://commons.wikimedia.org/wiki/File:RGB_Cube_Show_lowgamma_cutout_b.png

Datumizer. (2010b). [Photograph of the HSV color model mapped to a cylinder]. *Wikipedia Commons.* Retrieved March 2018 from https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png

Dobrusin, Y., Edan, Y., Grinshpun, J., & Peiper, U. (1992). Real time image processing for robotic melon harvesting. *American Society of Agricultural Engineers.* St. Joseph, Michigan: American Society of Agricultural Engineers (Paper No. 92-3515).

Dorj, U.-O., Lee, M., & Yun, S.-S. (2017). A yield estimation in citrus orchards via fruit detection and counting using image processing. *Computers and Electronics in Agriculture, 14*, 103-112. doi: https://doi.org/10.1016/j.compag.2017.05.019

Edan, Y., Rogozin, D., Flash, T., & Miles, G. (2000). Robotic melon harvesting. *Institute of Electrical and Electronics Engineers Transactions on Robotics and Automation, 16*(6), 831-835. doi: https://doi.org/10.1109/70.897793

Golzarian, M.-R., Lee, M.-K., & Desbiolles, J. (2012). Evaluation of color indices for improved segmentation of plant images. *Transactions of the American Society of Agricultural and Biological Engineers, 55*(1), 261-273. doi: http://dx.doi.org/10.13031/2013.41236

Gongal, A., Amatya, S., Karkee, M., Zhang, Q., & Lewis, K. (2015). Sensors and systems for fruit detection and localization: A review. *Computers and Electronics in Agriculture, 116*, 8-19. doi: https://doi.org/10.1016/j.compag.2015.05.021

Gongal, A., Silwal, A., Amatya, S., Karkee, M., Zhang, Q., & Lewis, K. (2016). Apple crop-load estimation with over-the-row machine vision system. *Computers and Electronics in Agriculture, 120*, 26–35. doi: http://dx.doi.org/10.1016/j.compag.2015.10.022

Gose, E., Johnsonbaugh, R., & Jost, S. (1996). *Pattern recognition and image processing.* New Jersey, USA: Prentice Hall PTR.

Graves, M., & Batchelor, B. (2003). *Machine vision for the inspection of natural products.* London, England: Springer.

Hannan, M. W., Burks, T. F., & Bulanon, D. M. (2009). A machine vision algorithm combining adaptive segmentation and shape analysis for orange fruit detection. *Agricultural Engineering International XI: CIGR Journal, 9*, Manuscript 1281.

Hannan, M., & Burks, T. (2004). Current developments in automated citrus harvesting. *American Society of Agricultural Engineers Annual International Meeting.* St. Joseph, Michigan. doi: http://dx.doi.org/10.13031/2013.16726

Huang, T. (1996). Computer vision: Evolution and promise. *19th CERN School of Computing* (pp. 21–25). Geneva, Switzerland: CERN.

Kamilaris, A., & Prenafeta-Boldu, F. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture, 147*, 70-90. doi: https://doi.org/10.1016/j.compag.2018.02.016

Kapach, K., Barnea, E., Mairon, R., Edan, Y., & Ben-Shahar, O. (2012). Computer vision for fruit harvesting robots – state of the art and challenges ahead. *International Journal of Computational Vision and Robotics, 3*, 4-34. doi: https://doi.org/10.1504/IJCVR.2012.046419

Kondo, N., Yamamoto, K., Shimizu, H., Yata, K., Kurita, M., Shiigi, T., . . . Nishizu, T. (2009). A machine vision system for tomato cluster harvesting robot. *Engineering in Agriculture, Environment and Food, 2*(2), 60-65. doi: https://doi.org/10.1016/S1881-8366(09)80017-7

Linker, R., Cohen, O., & Naor, A. (2012). Determination of the number of green apples in RGB images recorded in orchards. *Computers and Electronics in Agriculture, 81*, 45-57. doi: https://doi.org/10.1016/j.compag.2011.11.007

McBratney, A., Whelan, B., & Ancev, T. (2005). Future directions of precision agriculture. *Precision Agriculture, 6*, 7-23. doi: https://doi.org/10.1007/s11119-005-0681-8

Meyer, F., & Beucher, S. (1990). Morphological segmentation. *Journal of Vision Communication and Image Representation, 1*, 21-46.

Mirbod, O., Yoder, L., & Nuske, S. (2016). Automated measurement of berry size in images. *IFAC-PapersOnLine, 49*(16), 79-84. doi: https://doi.org/10.1016/j.ifacol.2016.10.015

Mizushima, A., & Renfu, L. (2011). Cost benefit analysis of in-field presorting for the apple industry. *Transactions of the American Society of Agricultural and Biological Engineers, 21*(1), 33-40. doi: http://dx.doi.org/10.13031/2013.29638

Mizushima, A., & Renfu, L. (2013). An image segmentation method for apple sorting and grading using support vector machine and Otsu's method. *Computers and Electronics in Agriculture, 94*, 29–37. doi: http://dx.doi.org/10.1016/j.compag.2013.02.009

Mollazade, K., Omida, M., & Arefi, A. (2012). Comparing data mining classifiers for grading raisins based on visual features. *Computers and Electronics in Agriculture, 84*, 124–131. doi: https://doi.org/10.1016/j.compag.2012.03.004

Moreda, G. P., Ortiz-Cañavate, J., García-Ramos, F. J., & Ruiz-Altisent, M. (2009). Nondestructive technologies for fruit and vegetable size determination – a review. *Journal of Food Engineering, 92*(2), 119–136. doi: http://dx.doi.org/10.1016/j.jfoodeng.2008.11.004

Nikon. (2018, April). *KeyMission 170*. Retrieved from Nikon: https://en.nikon.ca/nikon-products/product/action-camera/keymission-170.html

Nishad, P. M., & Chezian, R. (2013). Various colour spaces and colour space conversion algorithms. *Journal of Global Research in Computer Science, 4*(1), 44–48.

Nuske, S., Wilshusen, K., Achar, S., Yoder, L., Narasimhan, S., & Singh, S. (2014). Automated visual yield estimation in vineyards. *Journal of Field Robotics, 31*(5), 837-860. doi: https://doi.org/10.1002/rob.21541

Nyman, B. (2012). *[Photograph of a CMOS image sensor].* Retrieved March 2019, from https://www.flickr.com/photos/bnsd/8186971124

OpenCV. (2018, November 15). *Camera calibration and 3D reconstruction*. Retrieved from OpenCV Documentation: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

Otsu, N. (1979). A threshold selection method from gray-level histograms. *Institute of Electrical and Electronics Engineers Transactions on Systems, Man, and Cybernetics, 9*(1), 62–66. doi: https://doi.org/10.1109/TSMC.1979.4310076

Patel, H. N., Jain, R. K., & Joshi, M. V. (2012). Automatic segmentation and yield measurement of fruit using shape analysis. *International Journal of Computer Applications in Technology, 45*(7), 19-24.

Payne, A. B., Walsh, K. B., Subedi, P. P., & Jarvis, D. (2013). Estimation of mango crop yield using image analysis – Segmentation method. *Computers and Electronics in Agriculture, 91*, 57–64. doi: http://dx.doi.org/10.1016/j.compag.2012.11.009

Plá, F., Juste, F., & Ferri, F. (1993). Feature extraction of spherical objects in image analysis: An application to robotic citrus harvesting. *Computers and Electronics in Agriculture, 8*(1), 57–72. doi: https://doi.org/10.1016/0168-1699(93)90058-9

Pothen, Z., & Nuske, S. (2016). Automated assessment and mapping of grape quality through image-based color analysis. *IFAC-PapersOnLine, 49*(16), 72-78. doi: https://doi.org/10.1016/j.ifacol.2016.10.014

Sengupta, S., & Lee, W. S. (2014). Identification and determination of the number of immature green citrus fruit in a canopy under different ambient light conditions. *Biosystems Engineering, 117*, 51-61. doi: http://dx.doi.org/10.1016/j.biosystemseng.2013.07.007

Shahin, M. A., Tollner, E. W., Gitaitis, R. D., Sumner, D. R., & Maw, B. W. (2002). Classification of sweet onions based on internal defects using image processing and neural network techniques. *Transactions of the American Society of Agricultural Engineers, 45*(5), 1613-1618. doi: http://dx.doi.org/10.13031/2013.11046

Sofu, M. M., Erb, O., Kayacan, M. C., & Cetis, B. (2016). Design of an automatic apple sorting system using machine vision. *Computers and Electronics in Agriculture, 127*(C), 395–405. doi: https://doi.org/10.1016/j.compag.2016.06.030

Sonka, M., Hlavac, V., & Boyle, R. (2015). *Image processing, analysis and machine vision* (4th ed.). Scarborough, Ontario, Canada: Cengage Learning.

Stajnko, D., Lakota, M., & Hočevar, M. (2004). Estimation of number and diameter of apple fruits in an orchard during the growing season by thermal imaging. *Computers and Electronics in Agriculture, 42*(1), 31-42. doi: http://dx.doi.org/10.1016/S0168-1699(03)00086-3

Stajnko, D., Rakun, J., & Blanke, M. (2009). Modelling apple fruit yield using image analysis for fruit color, shape and texture. *European Journal of Horticultural Science, 74*(6), 260-267.

Stanhope, T. (2016). *Applications of low-cost computer vision for agricultural implement feedback and control.* Montréal, Québec: McGill University, Department of Bioresource Engineering.

Sun, D. (2008). *Computer vision technology for food quality evaluation.* New York, New York, USA: Elsevier Inc.

Telledis, I., & Levin, E. (2014). Photogrammetric image acquisition with small unmanned aerial systems. *American Society for Photogrammetry and Remote Sensing 2014 Annual Conference.* Louisville, Kentucky: ASPRS. Retrieved from https://www.asprs.org/a/publications/proceedings/Louisville2014/tellidis.pdf

United Nations Department of Economic and Social Affairs. (2017, May 28). *World population projected to reach 9.8 billion in 2050, and 11.2 billion in 2100*. Retrieved from United Nations Department of Economic and Social Affairs: https://www.un.org/development/desa/en/news/population/world-population-prospects-2017.html

Vincent, L., & Soille, P. (1991). Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *Institute of Electrical and Electronics Engineer Transactions on Pattern Analysis and Machine Intelligence, 13*(6), 583-598.

Wang, Q., Nuske, S., Bergerman, M., & Singh, S. (2012). Automated crop yield estimation for apple orchards. *Proceedings of the International Symposium on Experimental Robotics*, (pp. 745-748). Québec, Canada.

Wang, W., & Li, C. (2014). Size estimation of sweet onions using consumer-grade RGB-depth sensor. *Journal of Food Engineering, 142*, 153–162. doi: http://dx.doi.org/10.1016/j.jfoodeng.2014.06.019

Wang, W., & Li, C. (2015). A multimodal machine vision system for quality inspection of onions. *Journal of Food Engineering, 166*, 291–301. doi: http://dx.doi.org/10.1016/j.jfoodeng.2015.06.027

Zhang, B., Huang, W., Lia, J., Zhao, C., Fan, S., Wu, J., & Liu, C. (2014). Principles, developments and applications of computer vision for external quality inspection of fruits and vegetables: A review. *Food Research International, 62*, 326-343. doi: http://dx.doi.org/10.1016/j.foodres.2014.03.012

Zhou, R., Damerow, L., Sun, Y., & Blanke, M. (2012). Using color features of cv. “Gala” apple fruits in an orchard in image processing to predict yield. *Precision Agriculture, 13*(5), 568-580. doi: https://doi.org/10.1007/s11119-012-9269-2

Appendix A: Python Code

A-1 Initial Version of Python Code (Feasibility Study)

```
1.  # -*- coding: utf-8 -*-
2.  """
3.  Machine Vision Yield Monitor Program
4.  @author: amanda
5.  """
6.
7.  import numpy as np
8.  import cv2
9.  import os
10. import math
11. import matplotlib.pyplot as plt
12.
13. # Creates an elliptical structuring element for the opening/closing operations
14. # Elliptical Kernel
15.
16. """
17. >>> cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
18. array([[0, 0, 1, 0, 0],
19.        [1, 1, 1, 1, 1],
20.        [1, 1, 1, 1, 1],
21.        [1, 1, 1, 1, 1],
22.        [0, 0, 1, 0, 0]], dtype=uint8)
23. """
24.
25. ellipse_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(10,10))
26.
27. # Define range of Red onion color in HSV
28. # onions
29. upper_red = np.array([46,180, 255])
30. lower_red = np.array([0,40,40])
31.
32. def ellipse_perimeter(major_axis, minor_axis):
33.     a = major_axis/2
34.     b = minor_axis/2
35.     h = math.pow((a-b), 2)/math.pow((a+b), 2)
36.     perimeter = math.pi*(a+b)*(1+ 3*h/(10 + math.sqrt(4-3*h)))
37.     return perimeter
38.
39. directory = 'C:/Users/amand/Desktop/Delfland Test 09152017/afternoon_test_09152017'
40.
41. for root, dirs, filenames in os.walk(directory):
42.         for i, file in enumerate(filenames):
43.             imgpath = os.path.join(root,file) # Reconstructs the file path using the
    root_directory and current filename
44.             print(imgpath)
45.
46.             #while(cap.isOpened()):
47.             #ret, frame = cap.read()
48.             # Resize Images
49.             img = cv2.imread(imgpath).copy()
50.             # Resize Images
51.             # Determines the new aspect ratio (r) and set the new dimensions for the im
    age
52.             r = 500/img.shape[1]
```

```
            new_dim = (500, int(img.shape[0]*r))
            img = cv2.resize(img, new_dim, interpolation = cv2.INTER_AREA)
            # Convert the image to HSV colorspace
            blur = cv2.GaussianBlur(img,(7,7),0)
            img2 = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
            # Threshold the HSV image to get only blue colors
            mask = cv2.inRange(img2, lower_red, upper_red)
            #mask = cv2.inRange(img2, lower_red, upper_red)
            opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, ellipse_kernel)
            closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, ellipse_kernel)
            closing = cv2.GaussianBlur(closing,(3,3),0)
            # dilate makes the in range areas larger
            closing = cv2.dilate(closing, None, iterations=3)

            # Bitwise-AND mask and original image
            res = cv2.bitwise_and(img, img, mask = closing)
            dst, contours, hierarchy = cv2.findContours(closing, cv2.RETR_TREE, cv2.CHA
IN_APPROX_SIMPLE)

            for c in contours:
                moments = cv2.moments(c)
                if moments['m00'] != 0.0:
                    cx = int(moments['m10']/moments['m00'])
                    cy = int(moments['m01']/moments['m00'])
                    centroid = (cx,cy)
                    if len(c) > 5:
                        ellipse = cv2.fitEllipse(c)
                        major_axis,minor_axis = ellipse[1]
                        if (major_axis < 110) and (minor_axis<110):

                            print('major_axis: ')
                            print(major_axis)
                            print('minor_axis')
                            print(minor_axis)
                            perimeter = ellipse_perimeter(major_axis, minor_axis)
                            print('perimeter: ')
                            print(perimeter)
                            #if perimeter > 60: # Lobok
                            if (80 < perimeter) and (perimeter < 800): # Shallot onion

                                cv2.ellipse(img,ellipse,(0,0,255),2)
                    else:
                        pass


            cv2.imshow('Final Result', img)
            cv2.waitKey(1000)

cv2.destroyAllWindows()

      # End of program
```

A-2 Final Version of Python Code (Field Trial)

Yield Monitor Class

```
1.  # -*- coding: utf-8 -*-
2.
3.  ## USAGE cd to final_code
4.  # python YieldMonitor.py --conf conf.json
5.
6.  """
7.  Development of a Machine Vision Based Yield Monitor for Shallot Onions
8.  Precision Agriculture and Sensor Systems Group (PASS)
9.  McGill University, Department of Bioresource Engineering
10.
11. yield_monitor.py --- This is a yield monitoring program for the masters thesis of
12. Amanda Boatswain Jacques. This software detects onion shapes, classifies them by
13. size, and then exports them into a .CSV file with GPS data.
14. """
15.
16. # program Properties
17. __author__ = "Amanda Boatswain Jacques"
18. __version__ = 9.0
19.
20. # import the necessary python libraries
21. from datetime import datetime
22. import conf
23. import os
24. import pandas as pd
25. import serial
26. import sys
27. import time
28.
29. # computer vision
30. import cv2
31. import numpy as np
32. import preprocess_image
33. import size_calibration
34.
35. # create yield monitor class
36. class YieldMonitor:
37.     def __init__(self, config):
38.         # set current file path
39.         self.current_dir = sys.argv[0]
40.
41.         # load the configuration file
42.         if conf is None:
43.             raise ValueError
44.
45.         else:
46.             self.conf = conf.Conf(config)
47.             sources = self.conf["camera_sources"]
48.
49.         # initialize the camera
50.         for source in sources:
51.             try:
52.                 self.camera = cv2.VideoCapture(source)
53.
54.                 if self.camera.isOpened():
```

```
                    self.pretty_print("[INFO] CAMERA", "OK: Camera successfully opened")
                    self.pretty_print("[INFO] CAMERA", "Camera initialized!")

            except Exception as e:
                self.pretty_print("[ERROR] CAMERA", "Error: %s" % str(e))
                self.close()

        # create a directory for storing the images and .csv
        self.path = self.conf["external_drive"]
        date = datetime.strftime(datetime.now(), "%Y%m%d_%H%M" +"/")
        self.image_directory = self.path  + date
        self.result_directory = self.image_directory + "result_images/"
        self.pretty_print("[INFO] IMAGES", "Images will be saved in: " + self.image_directory)

        if not os.path.exists(self.image_directory):
            os.makedirs(self.image_directory)
            os.makedirs(self.result_directory)

    ### useful Functions
    def pretty_print(self, task, msg):
        # Pretty Print
        date = datetime.strftime(datetime.now(), '%d/%b/%Y %H:%M:%S')
        print('[%s] %s\t%s' % (date, task, msg))

   ### camera Functions
    def capture_image(self, write=False, ramp_frames = 40):
        """ Captures a single image from the camera and returns it in PNG format
        read is the easiest way to get a full image out of a VideoCapture object."""

        #self.pretty_print("[INFO] CAMERA", "Capturing photo...")

        # let the camera stabilize for 40 frames
        for i in range(ramp_frames):
            try:
                (retval, self.bgr) = self.camera.read()

            except Exception as e:
                self.pretty_print("[ERROR] CAMERA", "Error: %s" % str(e))
                self.close()

        if self.bgr is not None:
            cv2.imshow("Captured Image", self.bgr)
            cv2.waitKey(100)

            # save the image
                    if write == True:
                        date = datetime.strftime(datetime.now(), "%Y%m%d"+"_"+"%H%M%S")

                        #add directory here
                        self.filename = self.image_directory + date + ".png"
                        cv2.imwrite(self.filename, self.bgr)

                    else:
                        pass

                    return self.bgr

           # perform size calibration
           def calibrate_monitor(self):
```

```
113.                 self.pixel_metric = size_calibration.calibrate(self.conf["calibration_di
    rectory"])
114.                 self.pretty_print("[INFO] SIZE CALIBRATION", "Calibration completed.")
115.
116.                 return self.pixel_metric
117.
118.             # perform image processing and detect the onions in an image
119.             def find_onions(self, write=True):
120.                 original, preprocessed = preprocess_image.preprocess(self.bgr, resize=Fa
    lse)
121.                 self.small_count, self.medium_count, self.large_count, self.result = pre
    process_image.find_onion_contours(
122.                         preprocessed, original, self.pixel_metric)
123.
124.                 if write == True:
125.                     date = datetime.strftime(datetime.now(), "%Y%m%d"+"_"+"%H%M%S")
126.                     self.result_filename = self.result_directory + date + ".png"
127.                     cv2.imwrite(self.result_filename, self.result)
128.
129.
130.                 return(self.small_count, self.medium_count, self.large_count)
131.
132.             def init_gps(self):
133.                 """  Initialize the gps sensor and set the baudrate. """
134.                 COMNUMS = self.conf["gps_ports"]
135.                 self.gps = serial.Serial()
136.
137.                 self.pretty_print("[INFO] GPS", "Initializing GPS... ")
138.                 # detect the active gps port and save it
139.                 for port in COMNUMS:
140.                     try:
141.                          self.gps = serial.Serial(port, timeout = 0.2)
142.                          self.gps_port = port
143.                          # explicit close 'cause of delayed GC in java
144.                          #self.gps.close()
145.
146.                     except serial.SerialException:
147.                         pass
148.
149.                 if self.gps.isOpen():
150.                     # set the gps baudrate
151.                     self.gps.baudrate = self.conf["gps_baudrate"]
152.                     self.pretty_print("[INFO] GPS", "GPS at port %s with baud %s! " % (s
    elf.gps_port, self.gps.baudrate))
153.
154.                 else:
155.                     self.pretty_print("[ERROR] GPS", "GPS not found!.")
156.                     self.close()
157.
158.
159.             def get_position(self):
160.                 """ Get the current position (latitude, longitude, speed) of the image.
    """
161.                 # retrieve only the RMC sentences
162.                 code = "RMC"
163.
164.                 while True:
165.                     try:
166.                         line = self.gps.readline()
167.                         line = line.decode("utf-8")
168.                         #print(line)
```

```
169.
170.                         if line.find(code) > 0:
171.                             break
172.
173.                     except UnicodeDecodeError:
174.                         pass
175.
176.                 gps_data = line.split(",")
177.
178.                 # only report GPS sentences if an active valid fix was received
179.                 #if gps_data[2] == "V":
180.                 if gps_data[2] == "A":
181.                     self.latitude = gps_data[3]
182.                     self.latitude_char = gps_data[4]
183.                     self.longitude = gps_data[5]
184.                     self.longitude_char = gps_data[6]
185.                     self.speed = gps_data[7]
186.
187.                     if self.speed is not None:
188.                         self.speed = float(gps_data[7])*1.852
189.                         self.speed = format(self.speed, ".3f")
190.                 else:
191.                     pass
192.
193.                 return (self.latitude, self.longitude, self.speed)
194.
195.             def run(self):
196.                 """ Run the program continuously. Get captures,
197.                 analyze them, and then save the current position. """
198.
199.                 self.pretty_print("[INFO] RUNNING", "Running yield monitoring program. P
    ress CTRL+C to exit.")
200.
201.                 # open the GPS port, give some time for GPS and camera to stabilize
202.
203.                 time.sleep(5)
204.                 self.data = []
205.
206.                 while (True):
207.                     try:
208.                         self.capture_image(write =True)
209.                         small, medium, large = self.find_onions()
210.                         lat, long, speed = self.get_position()
211.                         self.log = [small, medium, large, lat, long, speed]
212.                         columns = ['S', 'M', 'L', 'Latitude', 'Longitude', 'Speed (km/h)
    ']
213.                         cv2.putText(self.result, str(columns), (10, 40), cv2.FONT_HERSHE
    Y_SIMPLEX, 0.8, (0,0,255), 2, cv2.LINE_AA)
214.                         cv2.putText(self.result, str(self.log), (10, 70), cv2.FONT_HERSH
    EY_SIMPLEX, 0.75, (255,255,255), 2, cv2.LINE_AA)
215.                         cv2.imshow("result", self.result)
216.                         cv2.waitKey(100)
217.
218.                         if self.filename is not None:
219.                             self.log = [small, medium, large, lat, long, speed, self.res
    ult_filename]
220.
221.                         self.data.append(self.log)
222.
223.                         print(self.log)
224.
```

```
                    except KeyboardInterrupt:
                        cv2.destroyAllWindows()
                        self.gps.close()
                        self.data = np.array(self.data)
                        if self.filename is not None:
                            self.df = pd.DataFrame(self.data, columns = ['Small Onions', 'Medium Onions', 'Large Onions', 'Latitude', 'Longitude', 'Speed', 'Filename'])
                        else:
                            self.df = pd.DataFrame(self.data, columns = ['Small Onions', 'Medium Onions', 'Large Onions', 'Latitude', 'Longitude', 'Speed'])
                        print(self.df)

                        break

                return self.df

            ### write everything to csv
            def save_log(self):
                time.sleep(2)
                test_filename = input("Please enter the name of the results file (use only numbers, letters and underscores):  " )
                self.pretty_print("[INFO] SAVING", "Saving results from test run as %s " % (test_filename + ".csv"))
                self.df.to_csv(self.conf["log_file_path"] + test_filename + ".csv")

            ### close the program
            def close(self):
                # shut down the program and delete camera source
                self.pretty_print("[INFO] WARN", "Shutdown triggered!")
                time.sleep(5)
                self.gps.close()
                self.camera.release()
                cv2.destroyAllWindows()

                sys.exit()

        # End of program
```

Image Preprocessing

```
# -*- coding: utf-8 -*-
"""
Created on Thu Apr 26 13:09:08 2018
preprocess_image_updated.py

@author: Amanda

Machine Vision Yield Monitor Program
"""
""" Import Libraries """
# import the necessary python libraries
import numpy as np
import cv2
import os
import math
from skimage.feature import peak_local_max
from skimage.morphology import watershed
from scipy import ndimage
```

```
import size_calibration

""" Define Functions and Variables """

pixel_metric = size_calibration.calibrate("./calibration_images_undistorted/")
print("The pixel metric is: ", pixel_metric)


def ellipse_perimeter(major_axis, minor_axis):
    a = major_axis/2
    b = minor_axis/2
    h = math.pow((a-b), 2)/math.pow((a+b), 2)
    perimeter = math.pi*(a+b)*(1+ 3*h/(10 + math.sqrt(4-3*h)))
    return perimeter

def auto_canny(image, sigma=0.60):
    # compute the median of the single channel pixel intensities
    v = np.median(image)
    # apply automatic Canny edge detection using the computed median
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edged = cv2.Canny(image, lower, upper)
    # return the edged image
    return edged

# creates an elliptical structuring element for the opening/closing operations
ellipse_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (12, 12))

# define range of Red onion color in HSV
Lupper_red = np.array([50, 255, 255])
Llower_red = np.array([0,40,0])

Uupper_red = np.array([180,255, 255])
Ulower_red = np.array([160,40,0])

lower_white = np.array([0, 0, 240], dtype = "uint8")
upper_white = np.array([60, 30, 255], dtype ="uint8")

image_directory = "C:/Users/Amanda/Documents/yield_monitor_results_copy/20180924_1411_u
ndistorted/"

# original preprocessing method
def preprocess_original(image):

    #cv2.imshow("Original Image", image)
    # perform Mean Shift Filtering
    shifted = cv2.pyrMeanShiftFiltering(image, 14, 50)

    # convert the image to HSV colorspace and blur
    blur = cv2.medianBlur(shifted, 9)
    blur = cv2.GaussianBlur(blur, (9,9),0)
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    """
    ## perform mean subtraction normalization

    hue_mean = np.ones(h.shape, dtype=np.uint8)*np.mean(h)
    hue_mean = hue_mean.astype(np.uint8)
    subtracted_hue = cv2.subtract(h, hue_mean)

```

```
    sat_mean = np.ones(s.shape, dtype=np.uint8)*np.mean(s)
    sat_mean = sat_mean.astype(np.uint8)
    subtracted_sat = cv2.subtract(s, sat_mean)

    val_mean = np.ones(v.shape, dtype=np.uint8)*np.mean(v)
    val_mean = val_mean.astype(np.uint8)
    subtracted_val = cv2.subtract(v, val_mean)
    subtracted = cv2.merge((subtracted_hue, subtracted_sat, subtracted_val))

    cv2.imshow("Mean Substracted Image", subtracted)
    cv2.imshow("Mean Subtracted Hue Channel", subtracted_hue)
    cv2.imshow("Mean Subtracted Saturation Channel", subtracted_sat)
    cv2.imshow("Mean Subtracted Value Channel", subtracted_val)

    #subt_ret,subt_thresh = cv2.threshold(subtracted_hue,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    """
    # threshold the HSV image to get only red colors
    color_mask_lower = cv2.inRange(hsv, Llower_red, Lupper_red)
    color_mask_upper = cv2.inRange(hsv, Ulower_red, Uupper_red)
            color_mask_white = cv2.inRange(hsv, lower_white, upper_white)
            color_mask = color_mask_lower + color_mask_upper +color_mask_white
            color_opening = cv2.morphologyEx(color_mask, cv2.MORPH_OPEN, ellipse_kernel)

            color_closing = cv2.morphologyEx(color_opening, cv2.MORPH_CLOSE, ellipse_kernel)
            cv2.imshow("Color Segmentation", color_closing)

            ## chromacity calculations
            # split the BGR channels
            b, g, r = cv2.split(blur)
            b = b.astype('float')
            g = g.astype('float')
            r = r.astype('float')

            # add the 3 channels together
            merged = np.add(r, b)
            merged = np.add(merged, g)

            # calculate red intensity
            R_I = 3*r - b -g
            R_I = R_I.astype('uint8')
            cv2.imshow("Red Intensity Image", R_I)

            # calculate chromacity of the red channel
            chro_r = np.divide(r, merged)*255
            chro_r = chro_r.astype('uint8')
            cv2.imshow("Chromacity Image", chro_r)

            # equalize red chromacity and apply a median blur, remove noise
            equ = cv2.equalizeHist(chro_r)
            equ = cv2.medianBlur(equ, 9)
            grayscaled = R_I.copy()
            grayscaled = cv2.equalizeHist(grayscaled)
            grayscaled= cv2.GaussianBlur(grayscaled, (9,9),0)

            chro_edged = cv2.Canny(grayscaled, 180, 255)
            #chro_edged = cv2.dilate(chro_edged, ellipse_kernel, iterations=1)
            cv2.imshow("Chromacity Edged", chro_edged)
            final_mask = cv2.bitwise_and(chro_edged, color_closing)
```

```
138.             cv2.imshow("Final Mask", final_mask)
139.
140.             small, medium, large, result = find_onion_contours(final_mask.copy(),
141.                                                                image.copy(), pixel_metri
    c)
142.             cv2.imshow("Final Result", result)
143.             cv2.waitKey(2000)
144.             return (small, medium, large), result
145.
146.         def preprocess_watershed(image):
147.             # create a copy of the image to draw on
148.             clone = image.copy()
149.             #cv2.imshow("Original", clone)
150.             wts_result = np.zeros(image.shape, dtype=np.uint8)
151.             # perform Mean Shift Filtering
152.             #shifted = cv2.pyrMeanShiftFiltering(image, 14, 50)
153.             #cv2.imshow("Mean Shift Filtering", shifted)
154.             shifted = clone.copy()
155.             # convert the image to HSV colorspace and blur
156.             blur = cv2.medianBlur(shifted, 9)
157.             blur = cv2.GaussianBlur(blur, (9,9),0)
158.             cv2.imshow("Blurred Image", blur)
159.             hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
160.             h, s, v = cv2.split(hsv)
161.
162.             # threshold the HSV image to get only red colors
163.             color_mask_lower = cv2.inRange(hsv, Llower_red, Lupper_red)
164.             color_mask_upper = cv2.inRange(hsv, Ulower_red, Uupper_red)
165.             color_mask_white = cv2.inRange(hsv, lower_white, upper_white)
166.             color_mask = color_mask_lower + color_mask_upper + color_mask_white
167.             cv2.imshow("Color Mask", color_mask)
168.             color_opening = cv2.morphologyEx(color_mask, cv2.MORPH_OPEN, ellipse_kernel)

169.             cv2.imshow("Opening", color_opening)
170.             color_closing = cv2.morphologyEx(color_opening, cv2.MORPH_CLOSE, ellipse_ker
    nel)
171.             cv2.imshow("Closing", color_closing)
172.
173.             # sure background area
174.             sure_bg = color_closing.copy()
175.
176.             # Finding sure foreground area
177.             dist_transform =  ndimage.distance_transform_edt(sure_bg)
178.             cv2.imshow("Distance Transform CV2", dist_transform/np.max(dist_transform[:,
    :]))
179.             thresh = color_closing.copy()
180.
181.             # compute the exact Euclidean distance from every binary
182.             # pixel to the nearest zero pixel, then find peaks in this distance map
183.             localMax = peak_local_max(dist_transform, indices=False, min_distance=20,
184.                                       labels=thresh)
185.             # perform a connected component analysis on the local peaks,
186.             # using 8-connectivity, then appy the Watershed algorithm
187.             markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
188.             labels = watershed(-dist_transform, markers, mask=thresh)
189.             print("[INFO] {} unique segments found".format(len(np.unique(labels)) - 1))

190.             # loop over the unique labels returned by the Watershed
191.             # algorithm
192.
193.             total_small = int()
```

```
194.            total_medium = int()
195.            total_large = int()
196.
197.            if (len(np.unique(labels)) - 1) >= 1:
198.                for label in np.unique(labels):
199.                    # if the label is zero, we are examining the 'background'
200.                    # so simply ignore it
201.                    if label == 0:
202.                        continue
203.                    # otherwise, allocate memory for the label region and draw
204.                    # it on the mask
205.                    mask = np.zeros(color_closing.shape, dtype="uint8")
206.                    mask[labels == label] = 255
207.                    #cv2.imshow("Watershed Mask", mask)
208.                    #cv2.waitKey(300)
209.                    small, medium, large, wts_result = find_onion_contours(mask, clone,
210.                                                                           pixel_metric)
211.
212.                    clone = wts_result.copy()
213.
214.                    total_small = total_small + small
215.                    total_medium = total_medium + medium
216.                    total_large = total_large + large
217.
218.            if not wts_result.any():
219.                wts_result = image.copy()
220.
221.            #cv2.imshow("Final Result", wts_result)
222.            #cv2.waitKey(500)
223.
224.            return (total_small, total_medium, total_large), wts_result
225.
226.        def find_onion_contours(mask, original, pixel_metric):
227.            # initialize lists of all the onion types present in the image
228.            small_onions = []
229.            medium_onions = []
230.            large_onions = []
231.            dst, contours, hierarchy = cv2.findContours(mask, cv2.RETR_LIST,
232.                                                        cv2.CHAIN_APPROX_SIMPLE)
233.            result = original.copy()
234.
235.            for c in contours:
236.
237.                moments = cv2.moments(c)
238.                if moments['m00'] != 0.0:
239.                    cx = int(moments['m10']/moments['m00'])
240.                    cy = int(moments['m01']/moments['m00'])
241.                    centroid = (int(cx), int(cy))
242.                    ((x, y), radius) = cv2.minEnclosingCircle(c)
243.
244.                    if len(c) > 5:
245.                        #print("radius : ", radius)
246.                        #if (radius<60) and (radius>16):
247.                        ellipse = cv2.fitEllipse(c)
248.                        major_axis, minor_axis = ellipse[1]
249.                        width = radius/pixel_metric
250.                        #print("width in mm", width)
251.
252.                        try:
```

```
                            aspect_ratio = minor_axis/major_axis
                            #print("aspect ratio :", aspect_ratio)

                        except ZeroDivisionError:
                            pass

                        # filter the onions by size
                        if 15 <=  width < 22: # small onion
                            color = (255, 0, 0) # Blue
                            small_onions.append(c)
                        if (width > 22) and (width < 27): # Medium Onion
                            medium_onions.append(c)
                            color = (0, 255, 0) # Green
                        if (width > 27) and (width < 40): # Large Onion
                            large_onions.append(c)
                            color = (0, 0, 255) # Red

                        if any((small_onions, medium_onions, large_onions)):
                            if  15 <= width <= 40:
                                # draw the centroid of the circle as well as the onion b
    order
                                cv2.circle(result, centroid, 3, (255, 255, 0), -1)
                                #cv2.ellipse(result, ellipse, color, 2)
                                cv2.circle(result, (int(x), int(y)), int(radius), color,
     2)
                else:
                    pass

            # return the counts for each onion category
            return len(small_onions), len(medium_onions), len(large_onions), result
```

Config File Loading

```
# import the necessary packages
from json_minify import json_minify
import json

class Conf:
    def __init__(self, confPath):
        # load and store the configuration and update the object's dictionary
        conf = json.loads(json_minify(open(confPath).read()))
        self.__dict__.update(conf)

    def __getitem__(self, k):
        # return the value associated with the supplied key
        return self.__dict__.get(k, None)

```

GPS Sentence Parsing

```
# -*- coding: utf-8 -*-
"""
@author: Amanda
"""
```

```
5.
6.  import serial
7.
8.  port ="COM7" # Add to config file
9.  BAUDRATE = 38400 # Add to config file
10.
11. gps = serial.Serial(port)
12. gps.baudrate = BAUDRATE
13.
14. while True:
15.     # flush the gps serial port
16.     #gps.flushInput()
17.     try:
18.         line = gps.readline()
19.         line = line.decode("utf-8")
20.         gps_data = line.split(",")
21.
22.         # retrieve only the RMC sentences
23.         if gps_data[0] == "$GPRMC":
24.             # only report GPS sentences if an active valid fix was received
25.             #if gps_data[2] == "V":
26.             if gps_data[2] == "A":
27.                 latitude = gps_data[3]
28.                 latitude_char = gps_data[4]
29.
30.                 longitude = gps_data[5]
31.                 longitude_char = gps_data[6]
32.                 speed = gps_data[7]
33.                 #speed = format(speed, ".3f")
34.                 #speed = str(speed)
35.
36.                 print('%s %s %s %s %s %s' % (latitude, latitude_char, longitude,
37.                       longitude_char, speed, "km/h" ))
38.
39.                 print('%s %s %s %s ' % (latitude, latitude_char, longitude,
40.                       longitude_char))
41.
42.     except KeyboardInterrupt:
43.         gps.close()
44.         break
```

Main File

```
1.  # -*- coding: utf-8 -*-
2.  """
3.
4.  @author: Amanda
5.  """
6.  import YieldMonitor
7.  import time
8.  import argparse
9.
10. # construct the argument parse and parse the arguments
11.
12. ap = argparse.ArgumentParser()
13. ap.add_argument("-c", "--
    config", required=True, help="path to the configuration file")
14. args = vars(ap.parse_args())
15.
```

```
YM = YieldMonitor.YieldMonitor(args["config"])
time.sleep(2)

YM.pretty_print("[INFO] RUNNING", "Yield Monitor initialized!")
YM.calibrate_monitor()
time.sleep(2)
YM.init_gps()

YM.run()
YM.save_log()


YM.close()
```

Image Preprocessing (Updated post field trials)

```
# -*- coding: utf-8 -*-
"""
preprocess_image_updated.py

@author: Amanda

Machine Vision Yield Monitor Program
"""

""" Import Libraries """
# import the necessary python libraries
import numpy as np
import cv2
import os
import math
from skimage.feature import peak_local_max
from skimage.morphology import watershed
from scipy import ndimage
import size_calibration

""" Define Functions and Variables """

pixel_metric = size_calibration.calibrate("./calibration_images_undistorted/")
print("The pixel metric is: ", pixel_metric)


def ellipse_perimeter(major_axis, minor_axis):
    a = major_axis/2
    b = minor_axis/2
    h = math.pow((a-b), 2)/math.pow((a+b), 2)
    perimeter = math.pi*(a+b)*(1+ 3*h/(10 + math.sqrt(4-3*h)))
    return perimeter

def auto_canny(image, sigma=0.60):
    # compute the median of the single channel pixel intensities
    v = np.median(image)
    # apply automatic Canny edge detection using the computed median
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edged = cv2.Canny(image, lower, upper)
    # return the edged image
    return edged

```

```
# creates an elliptical structuring element for the opening/closing operations
ellipse_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (12, 12))

# define range of Red onion color in HSV
Lupper_red = np.array([50, 255, 255])
Llower_red = np.array([0,40,0])

Uupper_red = np.array([180,255, 255])
Ulower_red = np.array([160,40,0])


lower_white = np.array([0, 0, 240], dtype = "uint8")
upper_white = np.array([60, 30, 255], dtype ="uint8")

image_directory = "C:/Users/Amanda/Documents/yield_monitor_results_copy/20180924_1411_undistorted/"

# original preprocessing method
def preprocess_original(image):

    # perform Mean Shift Filtering
    shifted = cv2.pyrMeanShiftFiltering(image, 14, 50)

    # convert the image to HSV colorspace and blur
    blur = cv2.medianBlur(shifted, 9)
    blur = cv2.GaussianBlur(blur, (9,9),0)
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)

    """
    ## perform mean subtraction normalization

    hue_mean = np.ones(h.shape, dtype=np.uint8)*np.mean(h)
    hue_mean = hue_mean.astype(np.uint8)
    subtracted_hue = cv2.subtract(h, hue_mean)

    sat_mean = np.ones(s.shape, dtype=np.uint8)*np.mean(s)
    sat_mean = sat_mean.astype(np.uint8)
    subtracted_sat = cv2.subtract(s, sat_mean)

    val_mean = np.ones(v.shape, dtype=np.uint8)*np.mean(v)
    val_mean = val_mean.astype(np.uint8)
    subtracted_val = cv2.subtract(v, val_mean)
    subtracted = cv2.merge((subtracted_hue, subtracted_sat, subtracted_val))

    cv2.imshow("Mean Substracted Image", subtracted)
    cv2.imshow("Mean Subtracted Hue Channel", subtracted_hue)
    cv2.imshow("Mean Subtracted Saturation Channel", subtracted_sat)
    cv2.imshow("Mean Subtracted Value Channel", subtracted_val)

    #subt_ret,subt_thresh = cv2.threshold(subtracted_hue,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    """
    # threshold the HSV image to get only red colors
    color_mask_lower = cv2.inRange(hsv, Llower_red, Lupper_red)
    color_mask_upper = cv2.inRange(hsv, Ulower_red, Uupper_red)
    color_mask_white = cv2.inRange(hsv, lower_white, upper_white)
            color_mask = color_mask_lower + color_mask_upper +color_mask_white
            color_opening = cv2.morphologyEx(color_mask, cv2.MORPH_OPEN, ellipse_kernel)
```

```
            color_closing = cv2.morphologyEx(color_opening, cv2.MORPH_CLOSE, ellipse_ker
    nel)

            ## chromacity calculations
            # split the BGR channels
            b, g, r = cv2.split(blur)
            b = b.astype('float')
            g = g.astype('float')
            r = r.astype('float')

            # add the 3 channels together
            merged = np.add(r, b)
            merged = np.add(merged, g)

            # calculate red intensity
            R_I = 3*r - b -g
            R_I = R_I.astype('uint8')

            # calculate chromacity of the red channel
            chro_r = np.divide(r, merged)*255
            chro_r = chro_r.astype('uint8')

            # equalize red chromacity and apply a median blur, remove noise
            equ = cv2.equalizeHist(chro_r)
            equ = cv2.medianBlur(equ, 9)

            grayscaled = R_I.copy()
            grayscaled = cv2.equalizeHist(grayscaled)
            grayscaled= cv2.GaussianBlur(grayscaled, (9,9),0)

            chro_edged = cv2.Canny(grayscaled, 180, 255)
            chro_edged = cv2.dilate(chro_edged, ellipse_kernel, iterations=1)
            final_mask = cv2.bitwise_and(chro_edged, color_closing)
            small, medium, large, result = find_onion_contours(final_mask.copy(),
                                                               image.copy(), pixel_metri
    c)

            return (small, medium, large), result

        def preprocess_watershed(image):
            # create a copy of the image to draw on
            clone = image.copy()
            wts_result = np.zeros(image.shape, dtype=np.uint8)
            # perform Mean Shift Filtering
            shifted = cv2.pyrMeanShiftFiltering(image, 14, 50)

            # convert the image to HSV colorspace and blur
            blur = cv2.medianBlur(shifted, 9)
            blur = cv2.GaussianBlur(blur, (9,9),0)
            hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)
            h, s, v = cv2.split(hsv)

            # threshold the HSV image to get only red colors
            color_mask_lower = cv2.inRange(hsv, Llower_red, Lupper_red)
            color_mask_upper = cv2.inRange(hsv, Ulower_red, Uupper_red)
            color_mask_white = cv2.inRange(hsv, lower_white, upper_white)
            color_mask = color_mask_lower + color_mask_upper + color_mask_white
            color_opening = cv2.morphologyEx(color_mask, cv2.MORPH_OPEN, ellipse_kernel)

            color_closing = cv2.morphologyEx(color_opening, cv2.MORPH_CLOSE, ellipse_ker
    nel)
```

```
159.
160.            # sure background area
161.            sure_bg = color_closing.copy()
162.
163.            # Finding sure foreground area
164.            dist_transform =  ndimage.distance_transform_edt(sure_bg)
165.            #cv2.imshow("Distance Transform CV2", dist_transform/np.max(dist_transform[:
    ,:]))
166.            thresh = color_closing.copy()
167.
168.            # compute the exact Euclidean distance from every binary
169.            # pixel to the nearest zero pixel, then find peaks in this distance map
170.            localMax = peak_local_max(dist_transform, indices=False, min_distance=20,
171.                                      labels=thresh)
172.            # perform a connected component analysis on the local peaks,
173.            # using 8-connectivity, then appy the Watershed algorithm
174.            markers = ndimage.label(localMax, structure=np.ones((3, 3)))[0]
175.            labels = watershed(-dist_transform, markers, mask=thresh)
176.            print("[INFO] {} unique segments found".format(len(np.unique(labels)) - 1))

177.
178.            # loop over the unique labels returned by the Watershed
179.            # algorithm
180.
181.            total_small = int()
182.            total_medium = int()
183.            total_large = int()
184.
185.            if (len(np.unique(labels)) - 1) >= 1:
186.                for label in np.unique(labels):
187.                    # if the label is zero, we are examining the 'background'
188.                    # so simply ignore it
189.                    if label == 0:
190.                        continue
191.                    # otherwise, allocate memory for the label region and draw
192.                    # it on the mask
193.                    mask = np.zeros(color_closing.shape, dtype="uint8")
194.                    mask[labels == label] = 255
195.                    small, medium, large, wts_result = find_onion_contours(mask, clone,

196.                                                                           pixel_metric)

197.
198.                    clone = wts_result.copy()
199.
200.                    total_small = total_small + small
201.                    total_medium = total_medium + medium
202.                    total_large = total_large + large
203.
204.            if not wts_result.any():
205.                wts_result = image.copy()
206.
207.            return (total_small, total_medium, total_large), wts_result
208.
209.        def find_onion_contours(mask, original, pixel_metric):
210.            # initialize lists of all the onion types present in the image
211.            small_onions = []
212.            medium_onions = []
213.            large_onions = []
214.            dst, contours, hierarchy = cv2.findContours(mask, cv2.RETR_LIST,
215.                                                        cv2.CHAIN_APPROX_SIMPLE)
```

```
216.           result = original.copy()
217.
218.           for c in contours:
219.
220.               moments = cv2.moments(c)
221.               if moments['m00'] != 0.0:
222.                   cx = int(moments['m10']/moments['m00'])
223.                   cy = int(moments['m01']/moments['m00'])
224.                   centroid = (int(cx), int(cy))
225.                   ((x, y), radius) = cv2.minEnclosingCircle(c)
226.
227.                   if len(c) > 5:
228.                       #print("radius : ", radius)
229.                       #if (radius<60) and (radius>16):
230.                       ellipse = cv2.fitEllipse(c)
231.                       major_axis, minor_axis = ellipse[1]
232.                       width = radius/pixel_metric
233.                       #print("width in mm", width)
234.
235.                       try:
236.                           aspect_ratio = minor_axis/major_axis
237.                           #print("aspect ratio :", aspect_ratio)
238.
239.                       except ZeroDivisionError:
240.                           pass
241.
242.                       # filter the onions by size
243.                       if 15 <=  width < 25: # small onion
244.                           color = (255, 0, 0) # Blue
245.                           small_onions.append(c)
246.                       if (width > 25) and (width < 45): # Medium Onion
247.                           medium_onions.append(c)
248.                           color = (0, 255, 0) # Green
249.                       if (width > 45) and (width < 50): # Large Onion
250.                           large_onions.append(c)
251.                           color = (0, 0, 255) # Red
252.
253.                       if any((small_onions, medium_onions, large_onions)):
254.                           if  15 <= width <= 65:
255.                               # draw the centroid of the circle as well as the onion b
    order
256.                               cv2.circle(result, centroid, 3, (255, 255, 0), -1)
257.                               #cv2.ellipse(result, ellipse, color, 2)
258.                               cv2.circle(result, (int(x), int(y)), int(radius), color,
     2)
259.               else:
260.                   pass
261.
262.           # return the counts for each onion category
263.           return len(small_onions), len(medium_onions), len(large_onions), result
```

Statistical Analysis

```
# -*- coding: utf-8 -*-
"""
Created on Mon Feb 8 14:51:24 2019

@author: Amanda
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats

file_path = "C:/Users/Amanda/Documents/Thesis/Statistics/ks_data.xls"
print(file_path)

data = pd.read_excel(file_path, sheet_name="large_class", names = ["Predicted Diameter (mm)","True Diameter (mm)"])

# create plot area
fig, ax = plt.subplots(figsize=(8, 4))

n_bins = 50
predicted = np.asarray(data["Predicted Diameter (mm)"])
true = np.asarray(data["True Diameter (mm)"])


n, bins, patches = ax.hist(predicted, n_bins, density=True, histtype='step',
                           cumulative=True, label='Predicted Diameter')

n, bins, patches = ax.hist(true, n_bins, density=True, histtype='step',
                           cumulative=True, label='True Diameter')


# Compute the Kolmogorov-Smirnov statistic on the 2 gathered samples.
rs_statistic, p_value = stats.ks_2samp(predicted, true)

print("RS Statistic :", rs_statistic)
print("P Value:", p_value)


# tidy up the figure
ax.grid(True)
ax.set_title('Large Class: KS-
Test Comparison Cumulative Fraction Plot', fontname="Arial", fontsize = "large")
ax.legend(loc='right', prop={'size': 10})
ax.set_xlabel('Diameter (mm)', fontname="Arial", fontsize = "large")
ax.set_ylabel('Cumulative Fraction', fontname="Arial", fontsize = "large")

plt.show()
```

Appendix B: Hardware Specifications

Table B-2. Prototype Camera specifications

Attribute	Value
Brand Name	ELP
Resolution	2.0 Megapixel 1080P
Sensor	1/2.7" CMOS
Picture Format	MJPEG or YUY2 optional
USB	Protocal USB2.0 HS/FS
Exposure	Auto exposure AEC Support
White Balance	Auto white blance AEB Support
Effective pixels	1920 (H) x 1080 (V) pixels 1280 (H) x 1024 (V) pixels 1280 (H) x 720 (V) pixels 1024 (H) x 768 (V) pixels 800 (H) x 600 (V) pixels 640 (H) x 480 (V) pixels 352(H) x 288 (V) pixels 320 (H) x 240 (V) pixels
Performance	1920 (H) x 1080 (V) pixels MJPEG 30fps YUY2 6fps 1280 (H) x 1024 (V) pixels MJPEG 30fps YUY2 6fps 1280 (H) x 720 (V) pixels MJPEG 60fps YUY2 9fps 1024 (H) x 768 (V) pixels MJPEG 30fps YUY2 9fps 800 (H) x 600 (V) pixels MJPEG 60fps YUY2 21fps 640 (H) x 480 (V) pixels MJPEG 120fps YUY2 30fps
Voltage	DC 5-V/current 150mA
Size	Size 32x32mm/38*38
Work Temperature	DEGREES (-20~70)
Hardware Platform	PC; Mac; Android OS
Other Specifications	Adjustable parameters Brightness/Contrast/Color saturation /Definition/Gamma/WB Night vision optional, Support IR Cut and IR board for night vision

Table B-3. Solid state drive specifications

Attribute	Value
Brand Name	Samsung
Series	T5
Color	blue
Item Height	7.6 centimeter
Item Width	10 millimeters
Hard Disk Size	250 GB
Hard Disk Technology	Portable
Hardware Platform	PC;Mac;Android OS

Table B-4. GPS Sensor specifications

Attribute	Value
Brand Name	Garmin
Version	GPS 19x HVS (NMEA 0183)
Dimensions (DxH)	3 19/32" x 1 15/16" (91.6 mm x 49.5 mm)
Weight	7.1 oz (201 g)
Cable length	30 ft (9.14 m)
Temperature Range	-22° to 176° F (-30° to 80° C)
Compass-safe distance	5.9" (150 mm)
Power source input	8-33 Vdc, unregulated
Input current	40 mA at 12 Vdc

Appendix C: Additional Figures

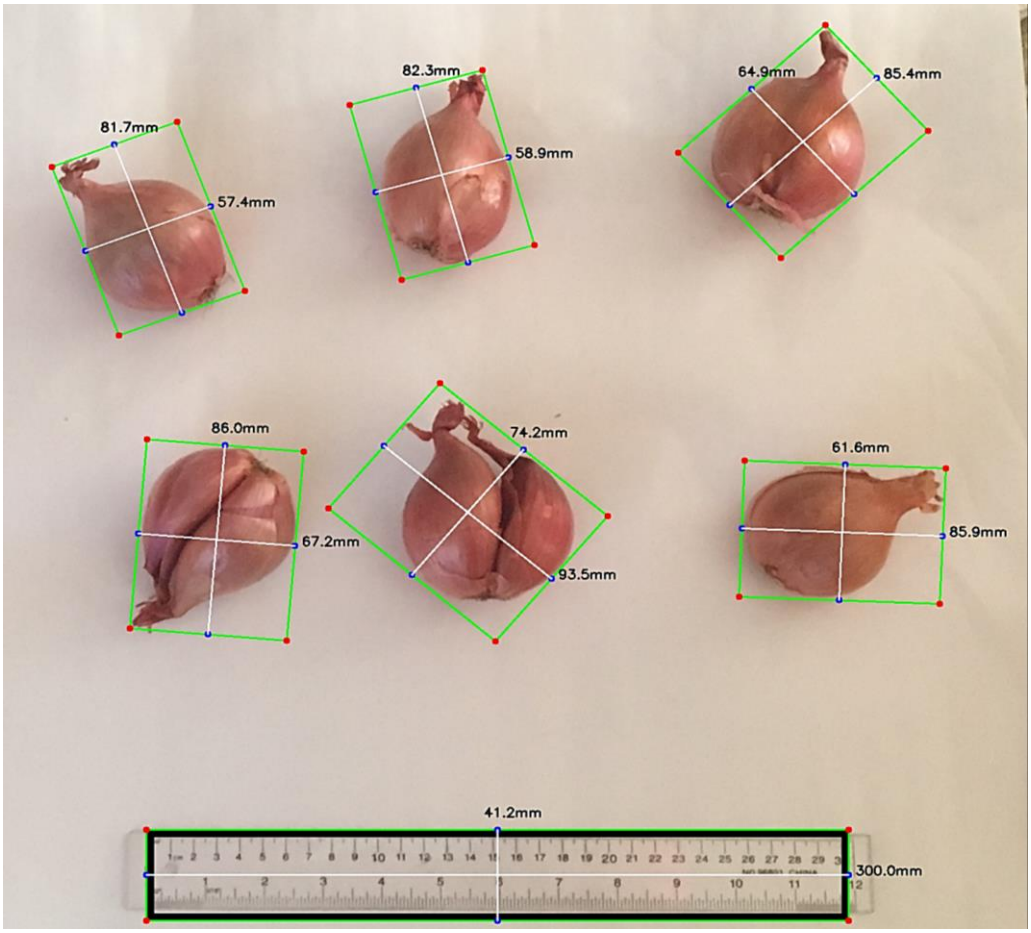


Figure C-1. Example image of the size calibration setup (**section 3.2.6**).

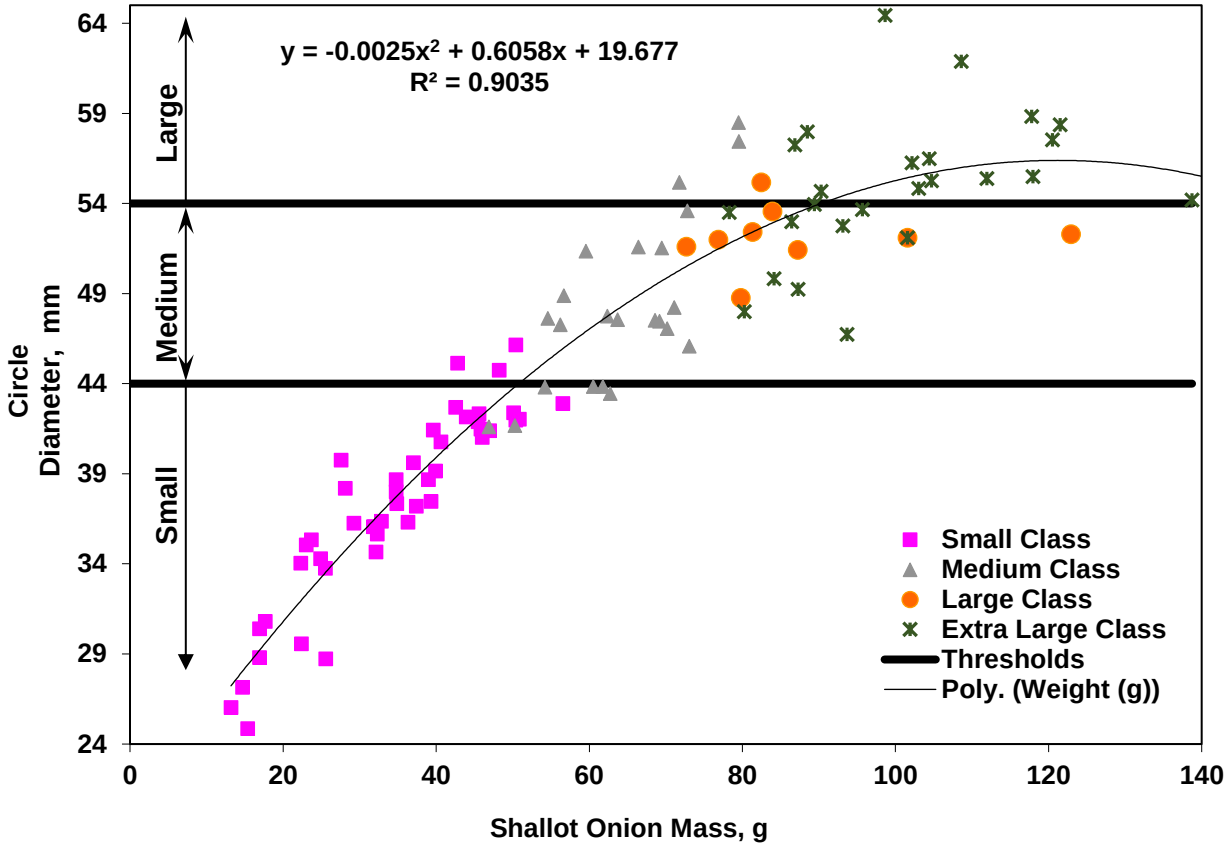


Figure C-2. Modification of the size classes for shallot onion classification.

Appendix D: Definition of Performance Metrics

Accuracy is the degree of similarity between a measurement of a given quantity and the true value of that same quantity. Accuracy is an important metric for evaluating the performance of a classification algorithm as it represents the algorithm's capacity to correctly classify the cases. It is calculated by summing the number of true positive (TP) and true negative (TN) classifications and dividing by the total number of classifications (TP + TN + false positives (FP) + false negatives (FN)), and like all metrics is often multiplied by 100 to yield a percentage. Although it is a powerful indicator of overall performance, accuracy alone is not enough to determine the strength of the algorithm and if it has correctly learned the task at hand (Baratloo et al., 2015).

$$Accuracy = \frac{\Sigma\, TP + \Sigma\, TN}{\Sigma\, TP + \Sigma\, FP + \Sigma\, TN + \Sigma\, F\, N} \quad \text{(D-1)}$$

Precision, also referred to as positive predictive value (PPV), is used to determine the algorithm's capacity to correctly identify positive cases with respect to all the cases the algorithm has classified as positive. It is calculated by dividing the number of true positives by the number of predicted positives which itself is a sum of the true positives and false positives (Baratloo et al., 2015).

$$Precision = \frac{\Sigma\, TP}{\Sigma\, TP + \Sigma\, FP} \quad \text{(D-2)}$$

Precision is an indicator of how reproducible and repeatable a measurement is under unchanged conditions and is used to evaluate the exactness of a model. Accuracy and precision are independent of each other, meaning that a set of values can be either accurate, precise, both at the same time, or neither.

Recall is the fraction of relevant instances (TP) that have been correctly identified over the total amount of relevant instances (TP and FN). Recall and precision are typically used in unison to report the performance of a classification system. Precision indicates the quality of the positive

prediction capability of the model, while recall indicates the completeness or quantity of correct predictions with respect to all positive instances present. High precision would mean that the algorithm returned a greater amount of relevant results than irrelevant ones, while a high recall value would mean that the algorithm returned most of the relevant results (Baratloo et al., 2015; Buckland & Gey, 1994).

$$Recall = \frac{\Sigma\, TP}{\Sigma\, TP + \Sigma\, FN} \tag{D-3}$$

Standard error (SE) is calculated as the standard deviation of the distribution associated with that error, or an estimate of that same standard deviation. For a sample, the standard error is equivalent to the standard deviation s divided by the square root of the sample size n :

$$SE = \frac{s}{\sqrt{n}} \tag{D-4}$$

SE is used to approximate the uncertainty around the estimate of the mean measurement, and it is most useful as a means of calculating a confidence interval (Altman & Bland, 2005).

Root Mean Square Error (RMSE) is the standard deviation of the residuals or prediction errors. RMSE is used to measure the difference between values predicted by a model and the values observed.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{D-5}$$

Where:

$n =$ number of observations,

$\hat{y}_i =$ predicted value for the i[th] observation,

$y_i =$ observed value.

Appendix E: Additional Data

E-1 Size Prediction Results

Table E-1.2. Size prediction results for the small onion class.

Predicted Diameter (Pixels)	True Diameter (Pixels)	Predicted Diameter (mm)	True Diameter (mm)	Difference	Predicted Size Class	True Size Class
116	79	34.36	23.40	10.96	SMALL	SMALL
106	84	31.39	24.88	6.52	SMALL	SMALL
104	85	30.80	25.17	5.63	SMALL	SMALL
118	87	34.95	25.77	9.18	SMALL	SMALL
142	91	42.06	26.95	15.10	SMALL	SMALL
102	91	30.21	26.95	3.26	SMALL	SMALL
147	92	43.54	27.25	16.29	SMALL	SMALL
109	95	32.28	28.14	4.15	SMALL	SMALL
110	96	32.58	28.43	4.15	SMALL	SMALL
150	96	44.43	28.43	15.99	MEDIUM	SMALL
121	98	35.84	29.02	6.81	SMALL	SMALL
151	98	44.72	29.02	15.70	MEDIUM	SMALL
128	101	37.91	29.91	8.00	SMALL	SMALL
131	102	38.80	30.21	8.59	SMALL	SMALL
117	103	34.65	30.51	4.15	SMALL	SMALL
127	104	37.61	30.80	6.81	SMALL	SMALL
195	104	57.75	30.80	26.95	LARGE	SMALL
101	105	29.91	31.10	-1.18	SMALL	SMALL
103	106	30.51	31.39	-0.89	SMALL	SMALL
159	107	47.09	31.69	15.40	MEDIUM	SMALL
171	107	50.64	31.69	18.95	MEDIUM	SMALL
114	108	33.76	31.99	1.78	SMALL	SMALL
136	108	40.28	31.99	8.29	SMALL	SMALL
125	109	37.02	32.28	4.74	SMALL	SMALL
160	109	47.39	32.28	15.10	MEDIUM	SMALL
99	110	29.32	32.58	-3.26	SMALL	SMALL
114	110	33.76	32.58	1.18	SMALL	SMALL
132	111	39.09	32.87	6.22	SMALL	SMALL
102	111	30.21	32.87	-2.67	SMALL	SMALL
170	113	50.35	33.47	16.88	MEDIUM	SMALL
149	113	44.13	33.47	10.66	MEDIUM	SMALL
105	114	31.10	33.76	-2.67	SMALL	SMALL
142	114	42.06	33.76	8.29	SMALL	SMALL
123	114	36.43	33.76	2.67	SMALL	SMALL
172	114	50.94	33.76	17.18	MEDIUM	SMALL
142	115	42.06	34.06	8.00	SMALL	SMALL
130	115	38.50	34.06	4.44	SMALL	SMALL
164	115	48.57	34.06	14.51	MEDIUM	SMALL
151	115	44.72	34.06	10.66	MEDIUM	SMALL
136	116	40.28	34.36	5.92	SMALL	SMALL
122	116	36.13	34.36	1.78	SMALL	SMALL
106	116	31.39	34.36	-2.96	SMALL	SMALL
162	116	47.98	34.36	13.62	MEDIUM	SMALL
173	116	51.24	34.36	16.88	MEDIUM	SMALL
146	117	43.24	34.65	8.59	SMALL	SMALL

129	117	38.21	34.65	3.55	SMALL	SMALL
159	117	47.09	34.65	12.44	MEDIUM	SMALL
121	118	35.84	34.95	0.89	SMALL	SMALL
109	118	32.28	34.95	-2.67	SMALL	SMALL
146	118	43.24	34.95	8.29	SMALL	SMALL
127	119	37.61	35.24	2.37	SMALL	SMALL
147	119	43.54	35.24	8.29	SMALL	SMALL
114	119	33.76	35.24	-1.48	SMALL	SMALL
158	119	46.79	35.24	11.55	MEDIUM	SMALL
168	119	49.76	35.24	14.51	MEDIUM	SMALL
133	120	39.39	35.54	3.85	SMALL	SMALL
121	120	35.84	35.54	0.30	SMALL	SMALL
110	120	32.58	35.54	-2.96	SMALL	SMALL
120	121	35.54	35.84	-0.30	SMALL	SMALL
124	121	36.72	35.84	0.89	SMALL	SMALL
127	122	37.61	36.13	1.48	SMALL	SMALL
111	122	32.87	36.13	-3.26	SMALL	SMALL
164	122	48.57	36.13	12.44	MEDIUM	SMALL
191	122	56.57	36.13	20.44	LARGE	SMALL
238	122	70.49	36.13	34.36	LARGE	SMALL
181	123	53.61	36.43	17.18	MEDIUM	SMALL
172	123	50.94	36.43	14.51	MEDIUM	SMALL
198	123	58.64	36.43	22.21	LARGE	SMALL
105	124	31.10	36.72	-5.63	SMALL	SMALL
163	124	48.28	36.72	11.55	MEDIUM	SMALL
211	124	62.49	36.72	25.77	LARGE	SMALL
130	125	38.50	37.02	1.48	SMALL	SMALL
121	125	35.84	37.02	-1.18	SMALL	SMALL
150	125	44.43	37.02	7.40	MEDIUM	SMALL
132	126	39.09	37.32	1.78	SMALL	SMALL
139	126	41.17	37.32	3.85	SMALL	SMALL
156	126	46.20	37.32	8.89	MEDIUM	SMALL
215	126	63.68	37.32	26.36	LARGE	SMALL
217	126	64.27	37.32	26.95	LARGE	SMALL
120	127	35.54	37.61	-2.07	SMALL	SMALL
125	127	37.02	37.61	-0.59	SMALL	SMALL
130	128	38.50	37.91	0.59	SMALL	SMALL
144	128	42.65	37.91	4.74	SMALL	SMALL
146	130	43.24	38.50	4.74	SMALL	SMALL
138	130	40.87	38.50	2.37	SMALL	SMALL
163	130	48.28	38.50	9.77	MEDIUM	SMALL
159	130	47.09	38.50	8.59	MEDIUM	SMALL
140	131	41.46	38.80	2.67	SMALL	SMALL
150	131	44.43	38.80	5.63	MEDIUM	SMALL
257	131	76.12	38.80	37.32	LARGE	SMALL
132	132	39.09	39.09	0.00	SMALL	SMALL
144	133	42.65	39.39	3.26	SMALL	SMALL
156	133	46.20	39.39	6.81	MEDIUM	SMALL
184	133	54.49	39.39	15.10	LARGE	SMALL
146	134	43.24	39.69	3.55	SMALL	SMALL
132	134	39.09	39.69	-0.59	SMALL	SMALL
104	134	30.80	39.69	-8.89	SMALL	SMALL
151	134	44.72	39.69	5.03	MEDIUM	SMALL
138	135	40.87	39.98	0.89	SMALL	SMALL
143	135	42.35	39.98	2.37	SMALL	SMALL
150	135	44.43	39.98	4.44	MEDIUM	SMALL

179	135	53.01	39.98	13.03	MEDIUM	SMALL
172	135	50.94	39.98	10.96	MEDIUM	SMALL
247	135	73.15	39.98	33.17	LARGE	SMALL
121	136	35.84	40.28	-4.44	SMALL	SMALL
151	136	44.72	40.28	4.44	MEDIUM	SMALL
205	136	60.71	40.28	20.44	LARGE	SMALL
114	137	33.76	40.58	-6.81	SMALL	SMALL
138	137	40.87	40.58	0.30	SMALL	SMALL
131	137	38.80	40.58	-1.78	SMALL	SMALL
173	137	51.24	40.58	10.66	MEDIUM	SMALL
181	138	53.61	40.87	12.74	MEDIUM	SMALL
180	138	53.31	40.87	12.44	MEDIUM	SMALL
162	138	47.98	40.87	7.11	MEDIUM	SMALL
250	138	74.04	40.87	33.17	LARGE	SMALL
143	139	42.35	41.17	1.18	SMALL	SMALL
150	139	44.43	41.17	3.26	MEDIUM	SMALL
131	140	38.80	41.46	-2.67	SMALL	SMALL
194	140	57.46	41.46	15.99	LARGE	SMALL
137	140	40.58	41.46	-0.89	SMALL	SMALL
152	140	45.02	41.46	3.55	MEDIUM	SMALL
200	140	59.23	41.46	17.77	LARGE	SMALL
143	141	42.35	41.76	0.59	SMALL	SMALL
126	141	37.32	41.76	-4.44	SMALL	SMALL
110	144	32.58	42.65	-10.07	SMALL	SMALL
128	144	37.91	42.65	-4.74	SMALL	SMALL
145	144	42.94	42.65	0.30	SMALL	SMALL
206	144	61.01	42.65	18.36	LARGE	SMALL
226	144	66.93	42.65	24.29	LARGE	SMALL
221	144	65.45	42.65	22.80	LARGE	SMALL
147	145	43.54	42.94	0.59	SMALL	SMALL
139	145	41.17	42.94	-1.78	SMALL	SMALL
123	145	36.43	42.94	-6.52	SMALL	SMALL
161	145	47.68	42.94	4.74	MEDIUM	SMALL
170	145	50.35	42.94	7.40	MEDIUM	SMALL
226	145	66.93	42.94	23.99	LARGE	SMALL
100	146	29.62	43.24	-13.62	SMALL	SMALL
148	146	43.83	43.24	0.59	SMALL	SMALL
133	146	39.39	43.24	-3.85	SMALL	SMALL
117	146	34.65	43.24	-8.59	SMALL	SMALL
159	146	47.09	43.24	3.85	MEDIUM	SMALL
151	146	44.72	43.24	1.48	MEDIUM	SMALL
151	146	44.72	43.24	1.48	MEDIUM	SMALL
140	147	41.46	43.54	-2.07	SMALL	SMALL
148	147	43.83	43.54	0.30	SMALL	SMALL
147	147	43.54	43.54	0.00	SMALL	SMALL
180	147	53.31	43.54	9.77	MEDIUM	SMALL
181	147	53.61	43.54	10.07	MEDIUM	SMALL
196	147	58.05	43.54	14.51	LARGE	SMALL
142	148	42.06	43.83	-1.78	SMALL	SMALL

Table E-1.3. Size prediction results for the medium onion class.

Predicted Diameter (Pixels)	True Diameter (Pixels)	Predicted Diameter (mm)	True Diameter (mm)	Difference	Predicted Size Class	True Size Class
114	149	33.76	44.13	-10.37	SMALL	MEDIUM
108	149	31.99	44.13	-12.14	SMALL	MEDIUM
171	149	50.64	44.13	6.52	MEDIUM	MEDIUM
151	149	44.72	44.13	0.59	MEDIUM	MEDIUM
149	149	44.13	44.13	0.00	MEDIUM	MEDIUM
145	150	42.94	44.43	-1.48	SMALL	MEDIUM
144	150	42.65	44.43	-1.78	SMALL	MEDIUM
141	150	41.76	44.43	-2.67	SMALL	MEDIUM
173	150	51.24	44.43	6.81	MEDIUM	MEDIUM
173	150	51.24	44.43	6.81	MEDIUM	MEDIUM
197	150	58.35	44.43	13.92	LARGE	MEDIUM
261	150	77.30	44.43	32.87	LARGE	MEDIUM
214	150	63.38	44.43	18.95	LARGE	MEDIUM
173	151	51.24	44.72	6.52	MEDIUM	MEDIUM
156	151	46.20	44.72	1.48	MEDIUM	MEDIUM
210	151	62.20	44.72	17.47	LARGE	MEDIUM
220	151	65.16	44.72	20.44	LARGE	MEDIUM
157	152	46.50	45.02	1.48	MEDIUM	MEDIUM
141	152	41.76	45.02	-3.26	SMALL	MEDIUM
155	152	45.91	45.02	0.89	MEDIUM	MEDIUM
129	153	38.21	45.31	-7.11	SMALL	MEDIUM
101	153	29.91	45.31	-15.40	SMALL	MEDIUM
113	153	33.47	45.31	-11.85	SMALL	MEDIUM
162	154	47.98	45.61	2.37	MEDIUM	MEDIUM
159	154	47.09	45.61	1.48	MEDIUM	MEDIUM
166	154	49.16	45.61	3.55	MEDIUM	MEDIUM
195	154	57.75	45.61	12.14	LARGE	MEDIUM
126	155	37.32	45.91	-8.59	SMALL	MEDIUM
138	155	40.87	45.91	-5.03	SMALL	MEDIUM
153	155	45.31	45.91	-0.59	MEDIUM	MEDIUM
148	156	43.83	46.20	-2.37	SMALL	MEDIUM
126	156	37.32	46.20	-8.89	SMALL	MEDIUM
101	157	29.91	46.50	-16.59	SMALL	MEDIUM
177	157	52.42	46.50	5.92	MEDIUM	MEDIUM
157	157	46.50	46.50	0.00	MEDIUM	MEDIUM
184	157	54.49	46.50	8.00	LARGE	MEDIUM
168	158	49.76	46.79	2.96	MEDIUM	MEDIUM
172	158	50.94	46.79	4.15	MEDIUM	MEDIUM
168	159	49.76	47.09	2.67	MEDIUM	MEDIUM
150	159	44.43	47.09	-2.67	MEDIUM	MEDIUM
171	159	50.64	47.09	3.55	MEDIUM	MEDIUM
210	159	62.20	47.09	15.10	LARGE	MEDIUM
214	159	63.38	47.09	16.29	LARGE	MEDIUM
185	159	54.79	47.09	7.70	LARGE	MEDIUM

179	160	53.01	47.39	5.63	MEDIUM	MEDIUM
191	160	56.57	47.39	9.18	LARGE	MEDIUM
128	161	37.91	47.68	-9.77	SMALL	MEDIUM
156	161	46.20	47.68	-1.48	MEDIUM	MEDIUM
186	161	55.09	47.68	7.40	LARGE	MEDIUM
251	161	74.34	47.68	26.66	LARGE	MEDIUM
249	161	73.75	47.68	26.06	LARGE	MEDIUM
239	161	70.78	47.68	23.10	LARGE	MEDIUM
225	161	66.64	47.68	18.95	LARGE	MEDIUM
194	161	57.46	47.68	9.77	LARGE	MEDIUM
110	162	32.58	47.98	-15.40	SMALL	MEDIUM
123	162	36.43	47.98	-11.55	SMALL	MEDIUM
168	162	49.76	47.98	1.78	MEDIUM	MEDIUM
143	163	42.35	48.28	-5.92	SMALL	MEDIUM
152	163	45.02	48.28	-3.26	MEDIUM	MEDIUM
174	164	51.53	48.57	2.96	MEDIUM	MEDIUM
175	164	51.83	48.57	3.26	MEDIUM	MEDIUM
169	166	50.05	49.16	0.89	MEDIUM	MEDIUM
155	166	45.91	49.16	-3.26	MEDIUM	MEDIUM
183	166	54.20	49.16	5.03	LARGE	MEDIUM
149	167	44.13	49.46	-5.33	MEDIUM	MEDIUM
101	168	29.91	49.76	-19.84	SMALL	MEDIUM
113	168	33.47	49.76	-16.29	SMALL	MEDIUM
142	169	42.06	50.05	-8.00	SMALL	MEDIUM
166	169	49.16	50.05	-0.89	MEDIUM	MEDIUM
195	169	57.75	50.05	7.70	LARGE	MEDIUM
182	170	53.90	50.35	3.55	MEDIUM	MEDIUM
108	172	31.99	50.94	-18.95	SMALL	MEDIUM
180	172	53.31	50.94	2.37	MEDIUM	MEDIUM
183	172	54.20	50.94	3.26	LARGE	MEDIUM
158	173	46.79	51.24	-4.44	MEDIUM	MEDIUM
208	174	61.60	51.53	10.07	LARGE	MEDIUM
131	175	38.80	51.83	-13.03	SMALL	MEDIUM
172	175	50.94	51.83	-0.89	MEDIUM	MEDIUM
182	175	53.90	51.83	2.07	MEDIUM	MEDIUM
183	175	54.20	51.83	2.37	LARGE	MEDIUM
131	176	38.80	52.13	-13.33	SMALL	MEDIUM
165	176	48.87	52.13	-3.26	MEDIUM	MEDIUM
190	176	56.27	52.13	4.15	LARGE	MEDIUM
152	177	45.02	52.42	-7.40	MEDIUM	MEDIUM
220	177	65.16	52.42	12.74	LARGE	MEDIUM
239	177	70.78	52.42	18.36	LARGE	MEDIUM
152	178	45.02	52.72	-7.70	MEDIUM	MEDIUM
195	178	57.75	52.72	5.03	LARGE	MEDIUM
182	179	53.90	53.01	0.89	MEDIUM	MEDIUM
174	181	51.53	53.61	-2.07	MEDIUM	MEDIUM

Table E-1.3. Size prediction results for the large onion class.

Predicted Diameter (Pixels)	True Diameter (Pixels)	Predicted Diameter (mm)	True Diameter (mm)	Difference	Predicted Size Class	True Size Class
167	183	49.46	54.20	-4.74	MEDIUM	LARGE
185	183	54.79	54.20	0.59	LARGE	LARGE
186	183	55.09	54.20	0.89	LARGE	LARGE
213	183	63.08	54.20	8.89	LARGE	LARGE
132	184	39.09	54.49	-15.40	SMALL	LARGE
206	184	61.01	54.49	6.52	LARGE	LARGE
182	185	53.90	54.79	-0.89	MEDIUM	LARGE
181	187	53.61	55.38	-1.78	MEDIUM	LARGE
194	187	57.46	55.38	2.07	LARGE	LARGE
184	189	54.49	55.98	-1.48	LARGE	LARGE
197	190	58.35	56.27	2.07	LARGE	LARGE
158	191	46.79	56.57	-9.77	MEDIUM	LARGE
196	191	58.05	56.57	1.48	LARGE	LARGE
205	192	60.71	56.86	3.85	LARGE	LARGE
253	192	74.93	56.86	18.07	LARGE	LARGE
222	193	65.75	57.16	8.59	LARGE	LARGE
218	194	64.56	57.46	7.11	LARGE	LARGE
175	196	51.83	58.05	-6.22	MEDIUM	LARGE
212	196	62.79	58.05	4.74	LARGE	LARGE
240	196	71.08	58.05	13.03	LARGE	LARGE
200	198	59.23	58.64	0.59	LARGE	LARGE
197	202	58.35	59.83	-1.48	LARGE	LARGE
228	207	67.53	61.31	6.22	LARGE	LARGE
242	209	71.67	61.90	9.77	LARGE	LARGE
219	212	64.86	62.79	2.07	LARGE	LARGE
144	233	42.65	69.01	-26.36	SMALL	LARGE
185	257	54.79	76.12	-21.32	LARGE	LARGE
185	260	54.79	77.00	-22.21	LARGE	LARGE
219	276	64.86	81.74	-16.88	LARGE	LARGE
195	296	57.75	87.67	-29.91	LARGE	LARGE

E-2 Median-Filtered Onion Count Predictions

Table E-2.1 Size prediction results and positioning for all onions.

Small (Median)	Medium (Median)	Large (Median)	Sum (Median)	Latitude (Median)	Longitude (Median)	Speed (km/h) (Median)
0	0	0	0	45.191998	-73.350543	1.259
0	0	0	0	45.191999	-73.350543	1.287
0	0	0	0	45.191990	-73.350547	0.398
1	0	0	1	45.192084	-73.350422	1.241
2	1	2	4	45.192089	-73.350416	1.287
3	1	1	5	45.192223	-73.350239	2.611
4	2	1	7	45.192382	-73.350009	2.982
1	1	2	3	45.192552	-73.349776	3.047
3	2	2	7	45.192713	-73.349544	3.065
2	0	2	4	45.192720	-73.349533	3.037
4	1	1	6	45.192893	-73.349311	3.028
3	2	1	5	45.193065	-73.349080	3.380
2	1	1	4	45.193258	-73.348824	3.464
3	1	2	6	45.193266	-73.348815	3.417
3	1	1	5	45.193455	-73.348568	3.380
2	2	1	5	45.193645	-73.348314	3.352
4	1	1	6	45.193825	-73.348056	3.463
4	2	2	8	45.194029	-73.347771	3.510
3	1	1	5	45.194038	-73.347759	3.676
2	1	3	6	45.194245	-73.347470	3.639
2	1	0	3	45.194455	-73.347192	3.676
2	1	1	4	45.194672	-73.346926	3.658
2	1	1	4	45.194680	-73.346914	3.565
2	1	1	4	45.194880	-73.346632	3.574
4	1	1	5	45.195080	-73.346341	3.714
1	1	2	4	45.195288	-73.346055	3.676
3	1	1	5	45.195296	-73.346042	3.667
3	0	1	4	45.195496	-73.345758	3.685
2	1	0	3	45.195698	-73.345492	3.667
1	1	2	4	45.195904	-73.345221	3.648
2	1	1	4	45.196111	-73.344945	3.630
2	1	1	4	45.196121	-73.344931	3.648
3	1	2	6	45.196333	-73.344655	3.528
2	1	2	5	45.196531	-73.344376	3.408
3	1	2	6	45.196739	-73.344093	3.602
4	1	1	6	45.196947	-73.343810	3.630
3	2	1	5	45.197156	-73.343521	3.547
4	2	1	7	45.197362	-73.343251	3.852
0	0	0	0	45.197661	-73.342782	4.195
2	0	0	2	45.197461	-73.342775	3.260
3	2	2	7	45.197315	-73.342968	3.149
2	1	1	4	45.197150	-73.343205	3.297

3	2	1	5	45.196994	-73.343454	2.815
2	1	2	5	45.196811	-73.343682	2.991
4	1	2	7	45.196634	-73.343917	3.028
1	1	3	5	45.196544	-73.344036	2.954
3	1	1	5	45.196455	-73.344154	3.111
3	1	1	5	45.196301	-73.344390	2.788
2	1	2	5	45.196295	-73.344397	2.936
3	2	2	6	45.196119	-73.344630	3.093
1	1	1	3	45.195960	-73.344865	3.139
2	1	2	5	45.195782	-73.345100	3.167
2	1	1	4	45.195778	-73.345109	3.102
2	1	2	5	45.195610	-73.345345	2.871
2	1	1	3	45.195454	-73.345561	2.917
2	1	1	4	45.195320	-73.345754	2.704
4	0	1	5	45.195245	-73.345859	2.964
2	1	2	4	45.195168	-73.345965	2.880
3	0	0	3	45.195014	-73.346173	2.778
2	1	1	3	45.194841	-73.346410	3.334
1	0	1	2	45.194668	-73.346650	3.093
3	2	1	5	45.194502	-73.346877	2.890
1	0	1	2	45.194401	-73.346993	2.945
2	1	1	4	45.194306	-73.347109	2.954
2	0	0	2	45.194133	-73.347351	3.065
2	1	1	3	45.193965	-73.347586	2.852
1	1	2	4	45.193799	-73.347813	2.963
0	1	1	2	45.193790	-73.347824	2.945
1	0	0	1	45.193618	-73.348037	2.954
3	1	1	5	45.193451	-73.348259	3.010
2	1	0	3	45.193268	-73.348509	2.843
2	0	1	3	45.193095	-73.348741	2.973
3	0	1	3	45.193085	-73.348752	3.001
2	0	0	2	45.192912	-73.348976	3.047
3	0	0	3	45.192747	-73.349234	2.982
2	0	1	3	45.192569	-73.349464	2.982
3	1	0	4	45.192486	-73.349582	3.093
1	1	1	3	45.192404	-73.349700	3.232
3	1	1	4	45.192234	-73.349923	3.047
2	1	0	3	45.192054	-73.350137	3.000
2	0	0	2	45.191914	-73.350341	1.315
2	1	1	4	45.192117	-73.350477	1.260
3	1	0	4	45.192153	-73.350429	1.269
1	1	1	3	45.192154	-73.350429	1.019
2	1	2	4	45.192294	-73.350226	2.436
4	1	1	6	45.192463	-73.350006	2.917
2	1	1	4	45.192630	-73.349753	3.001
2	2	1	5	45.192638	-73.349743	2.991