IMAGE-BASED MAPPING SYSTEM FOR TRANSPLANTED SEEDLINGS

by

Kyle McGahee

B.S., Kansas State University, 2014

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Mechanical and Nuclear Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2016

Approved by:

Major Professor
Dr. Dale Schinstock

Copyright

Abstract

Developments in farm related technology have increased the importance of mapping individual plants in the field. An automated mapping system allows the size of these fields to scale up without being hindered by time-intensive, manual surveying. This research focuses on the development of a mapping system which uses geo-located images of the field to automatically locate plants and determine their coordinates. Additionally, this mapping process is capable of differentiating between groupings of plants by using Quick Response (QR) codes. This research applies to green plants that have been grown into seedlings before being planted, known as transplants, and for fields that are planted in nominally straight rows.

The development of this mapping system is presented in two stages. First is the design of a robotic platform equipped with a Real Time Kinematic (RTK) receiver that is capable of traversing the field to capture images. Second is the post-processing pipeline which converts the images into a field map. This mapping system was applied to a field at the Land Institute containing approximately 25,000 transplants. The results show the mapped plant locations are accurate to within a few inches, and the use of QR codes is effective for identifying plant groups. These results demonstrate this system is successful in mapping large fields. However, the high overall complexity makes the system restrictive for smaller fields where a simpler solution may be preferable.

Table of Contents

List of Figures

List of Tables

Acronyms

APSC	Advanced Photo System Type-C
BGR	Blue Green Red
CSV	Comma Separated Value
DSLR	Digital Single-Lens Reflex
EDSDK	EOS Digital Software Development Kit
FPGA	Field Programmable Gate Array
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HSV	Hue Saturation Value
IMU	Inertial Measurement Unit
ISO	International Standards Organization
JPEG	Joint Photographic Experts Group
LED	Light Emitting Diode
LiDAR	Light Detection and Ranging
QR	Quick Response
ROS	Robot Operating System
RTK	Real Time Kinematic
SSE	Sum of Squared Errors
SSH	Secure Shell
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTM	Universal Transverse Mercator
UTC	Coordinated Universal Time
WGS-84	World Geodetic System

Acknowledgments

I would like to thank Dr. Jesse Poland and the National Science Foundation for the opportunity to complete this research. Dr. Dale Schinstock for being a great mentor and always keeping me pointed in the right direction on my studies. Ethan Wells for helping in the development of the data collection program. Dr. Kevin Wang for his work on the first version of the camera triggering interface. Josh Sharon for welding the top frame attached to the robot, and Byron Evers for helping me transport the robot to Salina. Most importantly I'd like to thank my loving parents and sisters, because without them I wouldn't be anywhere close to where I am today.

Chapter 1

Introduction

Recent research shows the growth rate of crop production must increase in order to meet the demand created by a quickly growing world population [1]. This increase in production is supported by new, automated technologies that allow both researchers and farmers to operate more efficiently. One such technology, which is investigated in this research, is the means to automatically develop a map of plant locations within a field.

An important point to consider with field mapping is that there are two different starting points. In some applications seeds are directly planted in the field, while in other applications plants are grown, typically in a greenhouse, and then transplanted into the field. The latter process is known as transplanting, and the plants are referred to as transplants. In addition, certain applications require similar plants to be identified as part of a larger plant group. These similarities could be based on crop type, treatments, or genetic parameters.

The task of mapping transplants has traditionally been performed by manually walking through the field and surveying the location of each plant. However, this is a tedious process that does not scale well to large fields containing tens of thousands of plants. Several automated mapping processes, such as [2], have been explored, but each of these methods have key drawbacks which are discussed in Section 2.1. The research presented in this thesis aims to address these shortcomings by developing a novel mapping system based on computer vision techniques. This image-based mapping system is an effective solution for locating and

identifying plants, especially when implemented with an automated robotic platform.

1.1 Mapping and Identification

The mapping process, which refers to creating a field map, is split into two smaller processes. The first of these is to determine the coordinates of each plant in the field with respect to one or more reference frames. Depending on the application this could be a local field reference frame or a global frame, such as one specified by latitude and longitude angles. Since the plants are fixed to the earth this is a two-dimensional mapping problem, and a third dimension, such as altitude, is not required. In addition to assigning coordinates, it's also useful to determine the row number in which each plant is found. This first sub-process is referred to as plant mapping.

Since plants in the field may not all be identical, for example they may have variations in their genetics, then this requires the mapping process be able to assign a unique group label to each plant. These labels are known prior to planting, and the challenge is to determine where each plant group ends up in the field. This process of assigning a plant to a group of plants is referred to as plant identification. The ability to handle this additional requirement is one of the key improvements of the proposed mapping solution over previous approaches.

1.2 Applications

There are many applications where a field map is useful. One such application is for perennial agriculture where plants remain for several years. Having a field map with accurate coordinates allows the same plants to be tracked over multiple years.

An extension of this application that applies to both perennial and annual agriculture is using the plant coordinates to automate maintenance tasks. For example, herbicides or pesticides can be applied more efficiently using knowledge of plant locations [3]. As well as other common tasks, such as intra-row tilling or cutting, can be made more effective by automatically avoiding plants [4].

Another application that applies to both types of agriculture is automatic plant lookup within a database. For example, if a researcher needs to record notes about a plant in an un-mapped field they would likely need to manually enter, or possibly scan, a plant number to retrieve that plant's entry within the database. However, if they are equipped with a RTK receiver they would be able to automatically retrieve the plant's entry based on their current location in the field.

Lastly, a field map can be combined with geo-tagged sensor measurements, such as soil moisture content or height readings, to associate the measurements with individual plants. This automated data collection greatly increases the amount of plant traits that can be consistently measured [5].

1.3 Mapping System Overview

The image-based mapping system developed here can be broken down into three parts. The first is the mapping equipment which includes the platform that traverses the field and the cameras used to collect images. The second part consists of items placed in the field that enable plant identification or make the overall mapping process more robust. The combination of these first two parts is referred to as the system design, which is described in Chapter 3.

The third part of the mapping system is the software used to convert the images into a useful field map. This involves using image-processing techniques to extract meaningful information from the images, such as the locations of plants, as well as other grouping and clustering algorithms. These algorithms are applied in sequential steps which collectively are referred to as the post-processing pipeline. This section of the mapping system is covered in Chapter 4.

The next two chapters of this thesis, 5 and 6, discuss application of the mapping process to a large-scale experiment conducted at the Land Institute in Salina, Kansas. The final chapter summarizes the findings of the research and covers the important lessons learned during this experiment.

Chapter 2

Background

This chapter begins with a brief summary of existing research that involves automated plant mapping. The main differences between these solutions and the proposed mapping system are highlighted. Next, the various coordinate systems used throughout the mapping process are defined, as well as the background theory on converting between camera pixels and world coordinates. This conversion and it's underlying assumptions are at the core of the image-based mapping system.

2.1 Similar Research

There are two different approaches to precision plant mapping. One is to create the map during the planting process, that is as the plants or seeds are being deposited into the ground. The other is to determine the plant locations after the planting is finished. This section presents an existing solution to the former approach and then discusses different methods for determining plant locations after they've been planted in the ground.

2.1.1 Planter Based Solutions

The research described in [6] evaluated two different methods for mapping tomato transplants during the planting process. The first was the use of an encoder attached to the planting

wheel in order to sense when a plant should be ejected from the planter. This wheel encoder had an angular resolution that corresponded to a ground resolution of 0.45 millimeters. There was naturally displacement between when the plant was released from the planter and where it actually ended up in the field due to the forward velocity of the planter. This displacement was experimentally calculated on a small set of plants and then accounted for during the mapping. The second method was to use an infrared optical beam-splitter to detect when the stem of a plant passed through the beam. This beam-splitter sensor was placed roughly 1 meter behind the planting wheel in order to give the plant time to settle once the soil has been pressed down around it.

The mapping platform consisted of a Holland model 1600 planter equipped with a centimeter level RTK system using a single antenna. In order to determine the planter heading, a Sum of Squared Errors (SSE) linear regression algorithm was used on three consecutive position measurements. To further increase the accuracy of the system, a dual axis inclinometer was used to take into account the non-zero roll and pitch angles of the planter. This sensor, along with the RTK receiver, encoder, and beam-splitter, were all connected to a National Instruments cRIO embedded computer with a Field Programmable Gate Array (FPGA) for parallel data logging.

In an experiment with 512 tomato plants, the beam-splitting sensor detected 491 objects, however only 249 of those corresponded to actual plants. The rest were due to soil clods or other debris. Many of the actual plants were missed due to being bent or planted at an angle and thus too low to pass through the beam. Unsurprisingly, these results were considered to be very poor and were attributed to the difficulty in finding a good height for the beam-splitter to be placed above the ground.

The second method, the encoder wheel, detected 527 planting events. The additional 15 events were due to plants not being placed in the planter wheel by the operator. After the offset calibration mentioned above, both the encoder wheel and beam-splitter sensor achieved average mapping accuracies of approximately 3 centimeters in the east direction and 1 centimeter in the north direction. The movement of the planter was primarily in the east direction which explains the larger errors. The results of the experiment are listed

Measurement	Infrared Sensor	Encoder Wheel
Mean Error - East (cm)	3.7	3.0
Mean Error - North (cm)	1.2	1.0
Std. Dev. - East (cm)	2.1	1.6
Std. Dev. - North (cm)	0.9	0.8

Table 2.1: Tomato mapping results.

in Table 2.1.

While the encoder wheel was effective in creating an accurate plant map, this approach has several drawbacks. One of these drawbacks is it requires close monitoring of the sensors' status. If an issue goes unnoticed then it could potentially result in a large section of the map being unusable, without a straightforward way to redo the mapping. Another disadvantage is it is difficult to extend this method to include differentiating between different groups of plants, which for the experiment described in this thesis was considered a requirement.

Also, it is worth noting that both [7] and [8] implemented similar systems using infrared seed detectors, and these sensors were much more successful at reliably determining plant locations than the beam-splitting method described above. The accuracy results of these systems were similar to those shown in Table 2.1.

2.1.2 Post-Planting Solutions

Many different methods in the past have been investigated for locating plants in the field. While this is an essential step in creating a field map for a post-planting solution, this area of research mainly focuses on using this information in real-time as opposed to the creation of a map. For example, in a tilling application the system would locate the plants as the platform is driving through the field rather than relying on a pre-existing map of plant locations. As a result, the research typically focuses on the effectiveness of finding plants instead of the accuracy of their coordinates.

These methods of locating plants primarily involve non-contact sensors such as cameras or depth sensors. For example, [9] used the distance and reflectance values from a Light

Detection and Ranging (LiDAR) type sensor to effectively differentiate between green plants and soil. [10] successfully used a stereo vision camera to map rows of soybean plants based on estimated height measurements from a three-dimensional reconstruction. In addition, [11] investigated combining color and near-infrared spectrum along with three-dimensional plant data to improve the robustness of locating plants. [12] used a watershed algorithm with morphological operators to extract a pre-known characteristic such as vein structure. This is especially useful in situations where plants and weeds are overlapping, and it's not easy to differentiate between them.

Each of these methods have their advantages, but it was determined that a solution based on monocular, color vision is preferred for the proposed mapping system. This is because plant height information is not useful right after transplanting, and single image solutions are inherently simpler and less expensive.

2.2 Coordinate Systems

A key step in any mapping process is the correct choice of coordinate systems. In the context of mapping, coordinates define the location of an item in the field with respect to a reference point. These coordinate systems, or frames of reference, are typically fixed to items that are significant in the mapping process, which include the Earth, field, platform, and cameras. The specific coordinate systems used in this research are discussed in the following sections.

2.2.1 Latitude and Longitude

A commonly used frame that is fixed to the Earth is a spherical coordinate system with the angles referred to as latitude and longitude. However, since the Earth is not a sphere these angles are projected onto an ellipsoid, called a datum. The datum that is used in this research is the World Geodetic System (WGS-84), which is the default datum used by the Global Positioning System (GPS) [13]. The third dimension is altitude and is measured relative to the surface of the ellipsoid. Coordinates in the frame are denoted (Φ, λ, A).

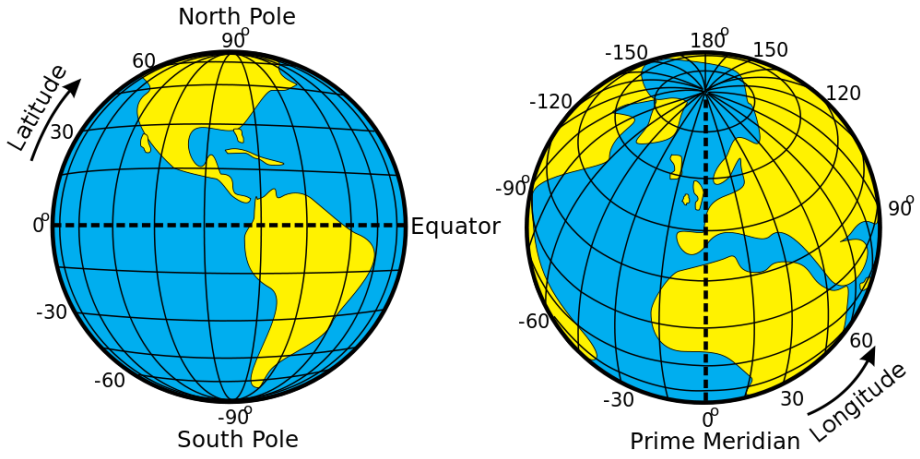


Figure 2.1: Illustration of Latitude (Φ) and Longitude (λ). Obtained under the Creative Commons License from the Wikimedia Commons.

2.2.2 Universal Transverse Mercator (UTM)

In order to describe points on the Earth's surface in a two-dimension plane, a projection model must be used on the latitude and longitude coordinates. The projection model used in this research is the Universal Transverse Mercator (UTM) which splits the Earth into 60 lateral bands, examples of which are shown in Figure 2.2. In each zone an item is described by its easting, northing and altitude coordinates. However, due to how the platform coordinate system is defined it's beneficial to have the Z axis in the downward direction. This requires the easting and northing be switched to maintain a right-handed coordinate system. Therefore, Universal Transverse Mercator (UTM) coordinates are described by (N, E, D). Another important modification is that the Z, or down, component is measured relative to the ground rather than the WGS-84 ellipsoid.

2.2.3 Field Coordinate System

One issue with the northing and easting described by UTM is that rows in a fields are not always planted north and south or east and west. Many of the post-processing steps benefit from removing this relative field orientation. A new field coordinate system is defined where the y-axis runs parallel to the rows and increases in the planting direction of the first row,

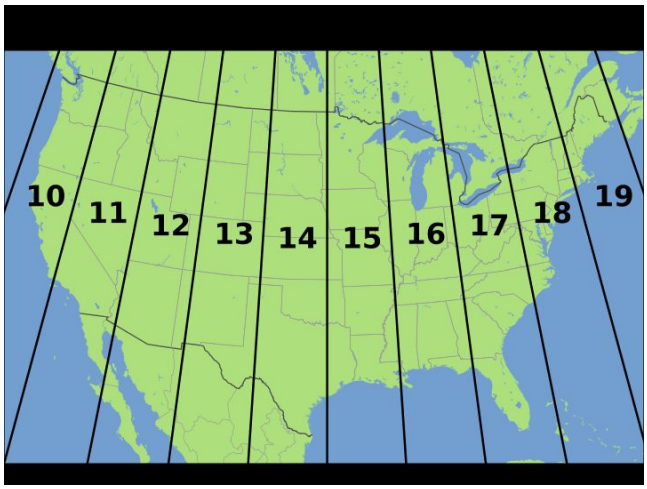


Figure 2.2: Visualization of UTM zones over the United States. Obtained under the Creative Commons License from the Wikimedia Commons.

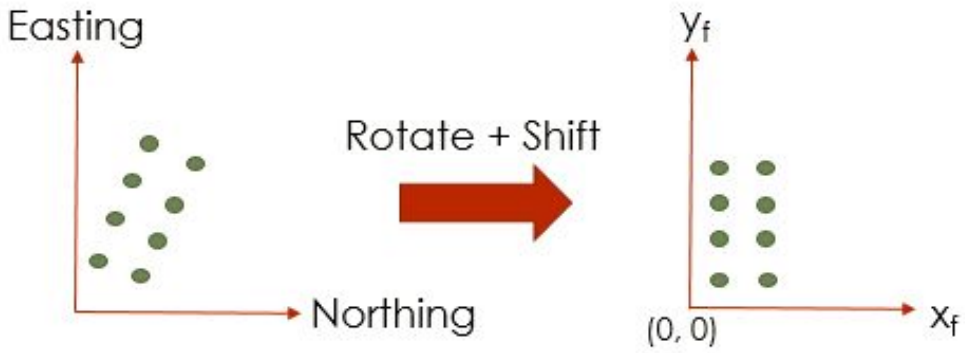


Figure 2.3: Conversion from modified UTM coordinate frame (left) to field coordinate frame (right).

and the x-axis increases in the direction of increasing row numbers. The origin is selected so that all items in the field have positive x and y coordinates. Similar to UTM, the units of this frame are meters. Coordinates in this frame are denoted (x_f, y_f, z_f).

2.2.4 Platform Coordinate System

Another necessary coordinate system is one fixed to the platform holding the cameras. This coordinate system allows the relative spacing and orientation between the cameras and the Global Navigation Satellite System (GNSS) antenna to be specified. In addition, this coordinate system defines how the platform's orientation is defined in terms of Euler angles,

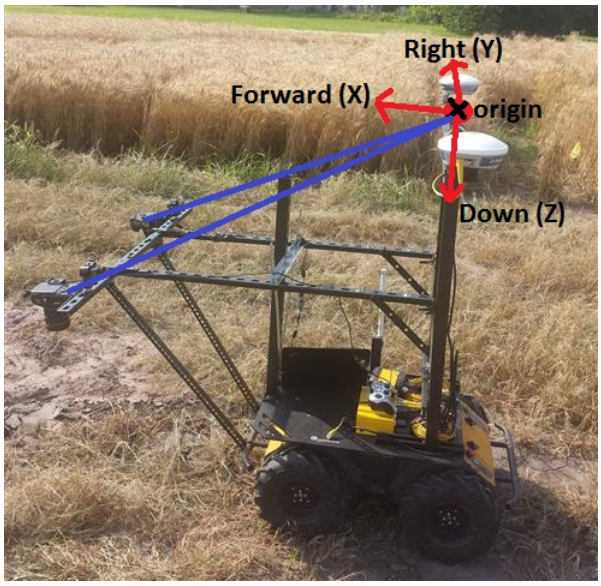


Figure 2.4: Forward-right-down platform coordinate frame. The origin is the average of the two antenna locations and is shown as a black X. Sensor offset vectors are shown in blue for the two cameras.

which is useful for accounting for non-level camera orientation. The axes of this coordinate system are displayed in Figure 2.4 which defines the x-axis out of the front of the platform, the y-axis out the right hand side and the z-axis orthogonal in the downward direction.

2.2.5 Camera Coordinate System

The camera coordinate system describes the location of objects in the world relative to the camera body. Typically, a camera coordinate system is defined by the x-axis out of the right hand side of the camera, the y-axis out of the bottom and the z-axis along the optical axis. For this application, however, an alternative camera coordinate system is defined which has the x-axis out of the top of camera, the y-axis out of the right side and the z-axis along the optical axis. This definition is used so that when the camera is mounted on the platform facing the ground with the top forward, the camera axes will align with the forward-right-down platform coordinate system. This makes the meaning of the standard roll-pitch-yaw Euler angles consistent and is also enforced by the data collection program. The origin is defined to be located at a distance f in front of the imaging sensor along the optical axis, where f is the focal length of the lens. This is discussed more in the following section.

Figure 2.5: Image showing camera coordinate axes.

2.3 Coordinate Projections

An important step when post-processing the images is converting between pixel and world coordinates, which is also the subject of many computer vision algorithms. Since pixels are described in $\mathbb{R}^2$ and world points in $\mathbb{R}^3$, this conversion requires a projection.

However, before this conversion can be defined a model must be chosen which describes how points in the world are projected onto the imaging sensor. This section first presents the model used in this research and then discusses two equivalent methods that can be used for converting coordinates.

2.3.1 Projection Model

A common choice for a thin, convex lens is the central perspective imaging model shown in Figure 2.6 [14]. This model is based on the concept of an ideal lens that is treated as an equivalent pin-hole. This implies the lens is perfectly focused and has no geometrical distortions. The ideal lens will have two focal points along the optical axis. One in the positive direction, or in front of the lens, and the other in the negative direction, or behind the lens. The central perspective model defines the image plane to be at the focal point in front of the lens so that the image is not inverted.

Points in the image plane can be described using two different coordinate frames. The

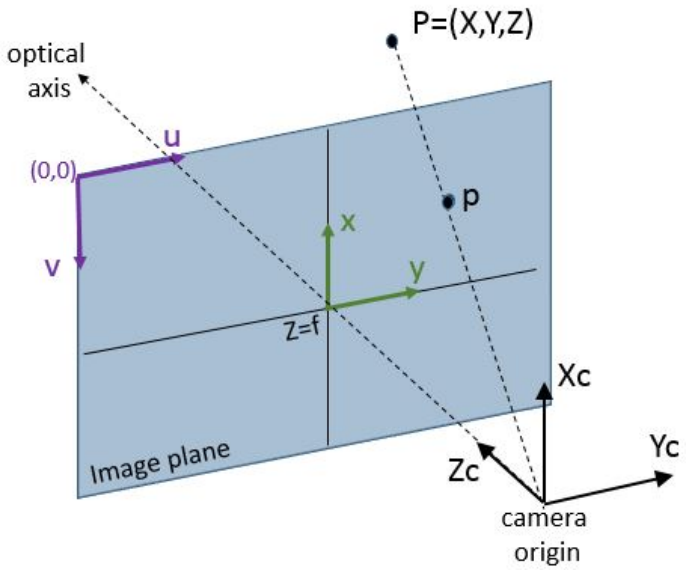


Figure 2.6: Central perspective imaging model showing the pixel frame in purple and the sensor frame in green. A point in the camera frame, P, is shown intersecting the image plane at point p.

first is the sensor frame and the second is the pixel frame. The origin of the sensor frame is located at the point where the optical axis intersects the image plane. A point in this frame is described by (x, y), where the axes are in the same direction as the camera frame and are shown in blue in Figure 2.6.

On the other hand, the origin of the pixel frame is located at the top left corner of the image plane, and the x-axis increases to the right and the y-axis downwards. A pixel's location is denoted as (u, v) rather than (x, y). The center of the image sensor is referred to as the principal point and is denoted (u_0, v_0). Another key difference between these frames is the units. The sensor frame has units of millimeters while the pixel frame has units of pixels.

2.3.2 Projection Methods

The conversion between pixel and world coordinates can go both ways, and both are used in the post-processing pipeline. However, going from pixel to world coordinates is used more often and is slightly more challenging. That is what is presented in this section. The conversion involves three frame transformations

$$\text{Pixel } (u, v) \rightarrow \text{Sensor } (x, y) \rightarrow \text{Camera } (X, Y, Z) \rightarrow \text{World } (N, E, D)$$

This is referred to as a backwards projection since it is projecting two-dimensional points out into the world. As mentioned above, this is more challenging because depth information is lost during the forward projection as the image is captured. Sometimes, multiple cameras or images are used to recover this depth information. Although in this application the height of objects being mapped are relatively small. For that reason, an assumption is made that every point in the world frame has a height of zero. This reduces the world to a plane and results in a $\mathbb{R}^2$ to $\mathbb{R}^2$ mapping which is possible with a single image.

There are two different methods for performing this backwards projection. The first method discussed operates in Euclidean space and is more intuitive to understand. The second method builds on this first method by instead working in Projective space. This produces a more efficient, but equivalent, conversion. Both of these methods use the central perspective model, and both make the same assumptions about the camera and lens. These are:

- The lens does not cause any distortion.
- There is no skew or displacement in how the image sensor is positioned within the camera.
- The focal length exactly matches the manufacturer's specification.

All of these assumptions are false to some extent. However, it is possible to correct them using certain camera calibration procedures [15]. For the system presented in this research, however, these effects are considered negligible.

Euclidean Space

The first transformation in the projection sequence is to convert pixel coordinates to sensor coordinates. Using the frame definitions shown in Figure 2.6, this can be represented in matrix form as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 & -\rho_h \\ \rho_w & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} \rho_h v_0 \\ \rho_w u_0 \end{bmatrix}$$

where ρ_w and ρ_h are the width and height of each pixel in millimeters, and u_0 and v_0 are the coordinates of the principal point in pixels. The next transformation is to convert from sensor coordinates to camera coordinates. It's clear from the projection model that if the sensor coordinates are combined with the focal length f as

$$(x, y, f)$$

then the result is a directional vector in the camera frame, but with units of millimeters rather than meters. The last frame transformation is to describe this vector in terms of the world frame. This can be accomplished by a rotation matrix R

$$\begin{bmatrix} n \\ e \\ d \end{bmatrix} = R * \begin{bmatrix} x \\ y \\ f \end{bmatrix} \tag{2.1}$$

where this rotation matrix is composed from the active, extrinsic rotations about the world's north, east and down axes in that order, by angles ϕ (roll), θ (pitch), and ψ (yaw). These are the Tait-Bryan angles of the camera frame with respect to the world frame. Technically, this is a passive rotation by negative angles since the vector is changing between bases, but using active rotations avoids many double negatives.

$$\begin{aligned} R &= R_z(\psi) * R_y(\theta) * R_x(\phi) \\ &= \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & cos(\phi) \end{bmatrix} \end{aligned} \tag{2.2}$$

These three rotation matrices can be multiplied together to result in a single rotation matrix.

$$\begin{bmatrix} \cos(\theta)\cos(\psi) & \sin(\phi)\sin(\theta)\cos(\psi) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & \cos(\phi)\sin(\theta)\sin(\psi) - \sin(\phi)\cos(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

Lowercase coordinates (n, e, d) in equation 2.1 are used to designate that these components are in the north, east, and down directions, but the units are still in millimeters meaning this does not represent the world point yet. The position of this vector in the world frame is given by the camera's world position

$$T = \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} N_{cam} \\ E_{cam} \\ D_{cam} \end{bmatrix} . \tag{2.3}$$

The actual world point can be found by calculating the intersection of this vector, (n, e, d), with the flat plane of the Earth given by the equation $D = 0$. This can be done by parameterizing the vector as a line with parameter S.

$$\begin{bmatrix} N \\ E \\ D \end{bmatrix} = S \begin{bmatrix} n \\ e \\ d \end{bmatrix} + T \tag{2.4}$$

Then plugging in the third component, D, into the plane equation and solving for

$$S = -T_z/d \ ,$$

which can be plugged back into equation 2.4. If d is zero then the position vector is parallel to the Earth's surface and will never intersect it.

While this method is easy to visualize, it requires multiple steps for each pixel that must be converted to world coordinates. A more efficient method is presented in the next section.

Projective Space

An ideal solution would be to describe the backwards projection from the pixel plane to the world plane as a single matrix that can be applied in one step. In order to do this, coordinates in each frame must be converted into the Projective space which adds an extra coordinate. This coordinate represents scale, and a set of coordinates in this space is referred to as homogeneous coordinates. There are many benefits of using homogeneous coordinates, including

1. performing translations and rotations in a single matrix.
2. avoiding unnecessary division in intermediate calculations, which is typically slower than other operations.
3. representing a coordinate at infinity with real numbers.

Homogeneous coordinates are followed by an apostrophe in order to differentiate them from regular Euclidean coordinates. For example, in the pixel frame coordinates can be described in Projective space as (u', v', w') where

$$u' = u * w' \text{ and } v' = v * w' \ .$$

In a backwards projection the pixel coordinates (u, v) are what is known, so w' is always set to 1. This is so it does not change the overall scale. In order to convert from pixel to sensor coordinates, the same relationship using the pixel sizes and principal point are used. However, when homogeneous coordinates are substituted this relationship can be described as a single matrix.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 0 & -\rho_h & \rho_h v_0 \\ \rho_w & 0 & \rho_w u_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix}$$

This is typically referred to as the inverse parameter matrix, or K^{-1}. The second transformation is to convert sensor coordinates to camera coordinates. A consequence of defining

the origin of the camera frame to be the location of the equivalent pin-hole is that all incoming rays of light converge to the origin of the frame. This leads to the simple relationship using the focal length f of

$$x = fX/Z \text{ and } y = fY/Z \ ,$$

which can easily be derived by similar triangles. When described as homogeneous coordinates this relationship becomes

$$x' = fX,\ y' = fY, \text{ and } z' = Z$$

or in matrix form as

$$\begin{bmatrix} X_u \\ Y_u \\ Z_u \end{bmatrix} = \begin{bmatrix} 1/f & 0 & 0 \\ 0 & 1/f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} .$$

The subscript u denotes that these coordinates in the camera frame are unscaled, similar to how (x, y, f) is not correctly scaled in the Euclidean method. Also note that (X_u, Y_u, Z_u) are not homogeneous coordinates. This matrix is referred to as the inverse camera matrix, or C^{-1}.

The last frame transformation is to go from the camera to world frame and correctly scale the position vector. This can be done in one step if homogeneous world coordinates (N', E', D', S') are used. In terms of the rotation matrix R and camera translation vector T defined in equations 2.2 and 2.3

$$\begin{bmatrix} R^{-1} & -T \end{bmatrix} \begin{bmatrix} N' \\ E' \\ D' \\ S' \end{bmatrix} = \begin{bmatrix} X_u \\ Y_u \\ Z_u \end{bmatrix} .$$

If each element in R is denoted by its row, i, and column, j, as R_{ij}, and since the inverse of a unitary matrix is equal to its transpose, then this is expanded to

$$\begin{bmatrix} r_{11} & r_{21} & r_{31} & -T_x \\ r_{12} & r_{22} & r_{32} & -T_y \\ r_{13} & r_{23} & r_{33} & -T_z \end{bmatrix} \begin{bmatrix} N' \\ E' \\ D' \\ S' \end{bmatrix} = \begin{bmatrix} X_u \\ Y_u \\ Z_u \end{bmatrix} .$$

This matrix is clearly not invertible and the assumption that $D = 0$ must be enforced. Since $D' = D/S'$ then D' is also zero, and it can be removed along with the third column of the matrix. After the matrix is inverted this results in

$$\tilde{p} = \begin{bmatrix} N' \\ E' \\ S' \end{bmatrix} = \begin{bmatrix} r_{11} & r_{21} & -T_x \\ r_{12} & r_{22} & -T_y \\ r_{13} & r_{23} & -T_z \end{bmatrix}^{-1} \begin{bmatrix} X_u \\ Y_u \\ Z_u \end{bmatrix} .$$

Even though an entire column of the rotation matrix is discarded, no information is lost as this column can be described as the cross product of the first and second columns. This inverted matrix is denoted ξ^{-1}, so that the entire projection can be represented as a single homography matrix

$$H = \xi^{-1} C^{-1} K^{-1}. \tag{2.5}$$

Once (N', E', S') is computed the actual northing and easting is simply the inhomogeneous coordinates

$$N = N'/S' \text{ and } E = E'/S' .$$

While it's not obvious, this result is equivalent to the one derived in the first method. This is verified symbolically in appendix A.

Chapter 3

System Design

A large part of the overall mapping system consists of the various components needed to capture, organize, and locate images of the field, as well as any additional field items needed for identifying plants. The proper design of this part of the mapping system is perhaps the most important step in the mapping process. Poor design leads to insufficient image quality, missing field coverage, or improperly geo-referenced images, which no amount of post-processing can correct.

The first part of the system discussed is the markers used for plant identification because the required size of these markers impose constraints on the rest of the system. Next, the base platform and additional equipment, such as the cameras, are presented along with reasoning about nominal parameters such as vehicle speed. This chapter concludes by describing additional field markers that are not strictly necessary but help improve the robustness of the mapping system.

3.1 Plant Identification

As mentioned in the introduction chapter, the mapping process involves not only determining plant coordinates but also assigning each plant to a group. Each plant group is referenced by a unique identifier (ID), such as 1035. The method used in this research is to encode the

Figure 3.1: Comparison of 2D barcode formats encoding the text 1035 with high error correction. From left to right: Quick Response (QR), Aztec, and Micro QR.

ID in a two-dimensional barcode that is placed at the beginning of each group of plants in the field.

If a plant group needs to be planted in different parts of the field then a repetition letter is appended to the group number. For example, three codes containing the text 1035A, 1035B, and 1035C all belong to plant group 1035. Since the two-dimensional barcodes are only placed at the beginning of the group, it's critical to know the direction of planting for each row in order to associate the correct set of plants with the correct ID. Codes could potentially be placed on both sides of each group to remove this added challenge, but this doubles the amount of code construction time and chance of missing a code during the post-processing. Instead, the row direction is encoded in row markers which are discussed later in this chapter.

3.1.1 Code Format

Quick Response (QR) codes were selected as the barcode format. This is a standardized format that was made publicly available by Denso Corporation over 20 years ago. It was first used for item tracking in the Japanese automotive industry and has most recently become well known for encoding Uniform Resource Locators (URLs) for websites [16]. Two important characteristics of this format are that it can be read from any orientation, and it can still be read if part of the code is damaged. Various other types of formats, such as Aztec or Micro QR, can encode information in smaller grids by restricting the character encoding, but offer less error correction. Also, at the time this research was conducted these alternate formats were not supported by any of the code reading programs that were investigated.

3.1.2 Size Constraint

For fields with thousands of different plant groups it's not feasible to place these QR codes by hand. Therefore, they must fit through the deposit cylinders of the transplanter which is shown later in the paper in Figure 5.2. In order to not get caught in the cylinders, the codes cannot be larger than 2.5 centimeters in diameter. Since there needs to be a white margin around the actual QR code, the code grid ends up being roughly 2 centimeters on each side. The QR format is split into different versions that define how many squares make up the code. The first, and smallest, version is a grid of 21 by 21 squares. An example of this first version is shown in Figure 3.1. This sized grid results in a maximum square size of only 1 millimeter.

3.1.3 Code Construction

The codes must be easy to produce due to the potentially large number of codes required for each field. The NiceLabel program can be used to generate all the QR codes at once, and a thermal printer, such as the Stover model SM4.25, can rapidly print the codes on pot labels. However, the pot labels need a solid base to stay upright and grounded during transplanting. The researchers at the Land Institute developed a degradable cement base into which the pot labels are inserted. This base consists of 2 parts perlite, 1 part water, and 1 part cement. An example of one of these pot label QR codes can be seen in Figure 3.2. While the plastic pot labels are not degradable, it's possible to re-use them between experiments if they are gathered up after the mapping is complete.

3.2 Platform Design

There are many types of platforms that could be used for mapping. Aerial vehicles have the benefit of autonomously traversing the field without the challenge of avoiding plants. However, they are unable to provide external lighting or shading which is important when post-processing the images. In addition, the size constraint on QR codes would require low

Figure 3.2: QR code printed on pot label inserted in cement base.

altitude flights at a constant altitude to keep the codes properly focused, which is not easy to achieve. Higher altitude flights would be possible with a telescopic lens, but this increases cost, weight, and most critically the effects of measurement error in camera orientation. For these reasons, only ground platforms are considered.

Two different types of ground platforms are investigated, a manual push-cart and a four-wheel robotic vehicle. The push-cart excels in its simplicity, however for this thesis only the robotic platform is discussed. The main benefits of a robot is the ability to drive at a constant speed and the option to operate autonomously. Driving at a consistent speed is important to ensure sufficient overlap between successive images, and a self-driving vehicle removes much of the tedious work associated with imaging large fields.

3.2.1 Robotic Platform

The selected robotic platform is the Husky A200 mobile robot made by Clearpath Robotics. The Husky is a four-wheeled, differential drive robot measuring 39 inches in length and 27 inches wide. It features a maximum speed of 1 meter per second, and it can carry up to 75 kilograms in optimal conditions. A custom C-channel structure was added to the top of the robot to enable it to image the field. This structure can be seen in Figure 3.3. Attached to the front of the structure are two Canon 7D Digital Single-Lens Reflex (DSLR) cameras. Using two cameras allows two rows to be mapped at the same time.

On the back of the C-structure are two white Trimble AG25 antennas that attach to a

Figure 3.3: Husky mobile robot equipped with two cameras.

Figure 3.4: Ag542 base station mounted on tripod.

Trimble BX982 GNSS receiver. This receiver is mounted to the top of the robot.

3.2.2 GNSS Receiver

The BX982 receiver provides centimeter level accuracy when paired with a fixed base receiver broadcasting RTK correction signals. For this application the fixed base is a Trimble Ag542 receiver with a Trimble Zypher Geodetic antenna, which is shown in Figure 3.4. This base station communicates with the SNB900 rover radio mounted on the robot over a 900 megahertz radio link. The dual antenna design allows the robot to determine its heading to within approximately 0.1 degrees. This accurate heading is important for geo-locating plants and QR codes within images, as well as allowing the robot to operate autonomously.

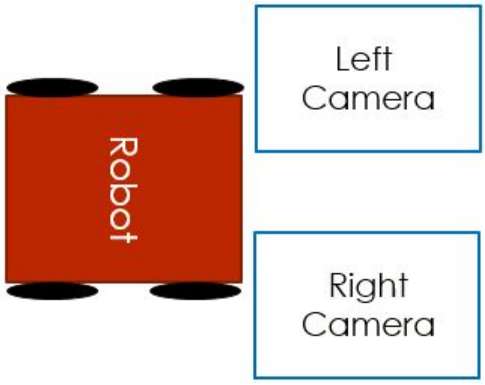


Figure 3.5: Depiction of camera field of views relative to robotic platform.

3.2.3 Cameras and Lighting

The Canon 7D features an 18 megapixel sensor and is fitted with a fixed 20 millimeter focal length, wide-angle lens. The camera contains an Advanced Photo System Type-C (APSC) sensor rather than a full frame 35 millimeter sensor. When paired with the wide-angle lens this gives a horizontal angle of view of 58.3 degrees and a vertical view of 40.9 degrees.

As shown in Figure 3.3, the cameras are mounted far out in front of the robot. This is so the wheels and front bumper do not show up in the image and effectively reduce the field of view. In addition, each camera is mounted with a 90 degree yaw offset with respect to the platform so that the longer side of the image is aligned with the forward movement of the robot as seen in Figure 3.5.

An item that is not pictured on the robot, but is shown in Figure 3.6, is a Light Emitting Diode (LED) bar. These 9 watt bars provide external lighting at night for consistent scene illumination. It is feasible to only use 2 bars, one for each camera, however using 4 bars provides enough light for the camera settings to be set conservatively which improves the robustness of the post-processing pipeline. Another benefit of using one bar on each side of each camera is it noticeably reduces image glare on the QR codes.

3.2.4 Determining Parameters

There are a number of parameters that need to be carefully chosen to ensure quality images and sufficient field coverage. The most important to determine first is the camera height, as

Figure 3.6: Canon 7D and external lighting bars.

that determines the resolution of the image. If the image resolution is too low then the QR codes will be unreadable.

If each pixel could correspond to exactly one square on the QR code then the minimum resolution would be 1 pixel per millimeter, since each square is roughly 1 millimeter in size. In reality, this is almost never the case, and the theoretical minimum is 3 pixels for one square. If only 2 pixels are used then the light from a black square could be split in half with the adjacent white square, and the result would be an unresolved gray square. However, all lenses also introduce some loss in contrast. The amount of contrast lost is a function of many things such as lens diffraction, aperture, and pixel position. Also, the camera's optical axis is not always perfectly orthogonal to the code. To account for these extra effects, the actual minimum resolution is set to 5 pixels per millimeter.

The 18 million pixels on the sensor are split into a 5184 by 3456 grid. The minimum resolution then requires a maximum image size of 1037 by 690 millimeters, which constrains the cameras to be no higher than 930 millimeters above the QR codes. In order to make the imaging process more robust, the cameras are mounted 700 millimeters above the ground which corresponds to an image size of 781 by 520 millimeters (or 31 by 21 inches), and a resolution of approximately 6.5 pixels per millimeter. Mounting the cameras lower than the maximum requirement also allows the external lighting to be more concentrated and requires smaller shading if the imaging is done during the day.

Another requirement that is added to make the mapping process more robust is each QR

Parameter	Value	Units
Camera Height	700	millimeters
Image Size	781 x 520	millimeters
Image Resolution	6.5	pixels / millimeter
Trigger Period	0.7	seconds / image
Robot Speed	0.4	meters / second

Table 3.1: Summary of platform parameters.

code must be in a minimum of two images. This helps solve temporary issues such as insects flying in front of the camera as well as offers multiple perspectives if the code is planted at an angle. In order to ensure this, the maximum spacing between successive images is given by the equation

$$\begin{aligned} \text{max spacing} &= (\text{image width} - \text{QR side length} - \text{pad})/2 \\ &= (781 - 25 - 75)/2 \\ &= 340 \text{ millimeters} \end{aligned}$$

where the 'pad' is extra spacing to account for variations in camera latency.

An additional constraint on the cameras that must be taken into account is the minimum trigger period. Many cameras, including the Canon 7D, are capable of exposing images rapidly and then buffering them before they are processed. However, the minimum trigger period considered in this section is the minimum amount of time for an image to be exposed, processed, and saved without continued buffering, as buffering can only be sustained for short periods. This was experimentally determined for the Canon 7D to be 0.7 seconds.

In order to satisfy the maximum image spacing of 340 millimeters, while not exceeding the minimum trigger period, the robot cannot drive faster than 0.5 meters per second. One downside of driving at this speed is that the cameras begin to noticeably shake when the field is not smooth, which can lead to blurry images. Therefore the nominal robot speed is set to 0.4 meters per second. These platform parameters are summarized in Table 3.1.

3.2.5 On-board Computers

There are a total of four computers used on the Husky.

Main Husky Computer - custom mini-ITX situated inside the main compartment of the robot running Ubuntu 12.04 along with the Robot Operating System (ROS). This computer contains all of the telemetry and guidance logic and is discussed more in Section 3.3.1.

Husky Microcontroller - Atmel ARM-based SAM7XC256 enclosed in the back of the robot chassis. This microcontroller receives linear and angular velocity commands from the main computer and reports feedback from the robot's encoders, battery, and motors.

Husky Interface Computer - Lenovo T400 laptop that is also running Ubuntu 12.04 along with ROS. This computer sits on top of the Husky and is used to send commands to the robot using a Secure Shell (SSH).

Data Collection Computer - Lenevo S431 laptop running Windows 7 that also sits on top of the Husky. This computer runs the program responsible for saving data from the cameras and GNSS receiver.

For simplicity, the output of the GNSS receiver is split to both the data collection computer as well as the main Husky computer. It would be ideal to run the data collection program on the Husky interface computer to eliminate the need for an extra computer. However, the data collection program requires a Windows based operating system to interact with the Canon cameras, and at the time of this research ROS was not stable on Windows.

3.3 Guidance and Control

The Husky arrived from Clearpath with basic driving functionality, but this was not sufficient for the mapping system. This section describes the additional functionality developed for

the robot that enables it to operate in autonomous or semi-autonomous modes.

3.3.1 Base Functionality

When the Husky was purchased the main computer came installed with the Robot Operating System, which is popular open-source framework that provides many of the same services as a traditional operating system, such as inter-process communication and hardware-abstraction. One major benefit is ROS allows different functionality to be split up into separate processes, referred to as nodes. This promotes code re-use and prevents one component from crashing the entire system. The nodes that were pre-installed on the Husky are listed below.

Teleop node - receives driving commands from the Logitech controller shown in 3.7. The default functionality is when the X button is held down then the left and right analog sticks command linear and angular velocity, respectively.

Husky node - in charge of sending the velocity commands over a serial port to the micro-controller that controls the motors.

IMU node - driver for the UM6 orientation sensor that came installed on the robot. This node is not used since the platform is stable, and the multiple GNSS antennas determine yaw.

3.3.2 Cruise Control

When driving through the field it's important that the robot maintains a constant speed to ensure all QR codes and plants are imaged. With the basic teleop functionality this was difficult to achieve while also keeping the robot centered in the middle of the row. To solve this issue, the teleop node was extended to include cruise control functionality that is commonly seen in consumer automobiles.

Cruise control can be enabled by either pressing both the Enable 1 and Enable 2 buttons at the same time or by the Preset button. The Preset button defaults to a certain configured speed, by default 0.4 meters per second. If the Override button is pressed then the linear

Figure 3.7: Logitech controller for Husky showing cruise control functionality.

speed is temporarily determined from the left analog stick, as is the case in the basic driving mode. The Resume button returns to the last speed, if there was one, and the up and down arrows on the D-pad vary the commanded speed in increments of 0.05 meters per second.

3.3.3 Automated Control

While this cruise control feature makes it feasible to manually drive the robot through the field, for large experiments this is a tedious task that requires ten or more hours of keep the robot centered between the rows.

ROS contains well-developed navigation functionality that allows the robot to convert odometry and sensor data into velocity commands. This navigation code, unfortunately, is primarily based around advanced functionality such as cost maps, detailed path planning, and map-based localization. All of which are unnecessary for this application. Therefore, the author decided to develop a simple guidance solution that is implemented in the following nodes:

GPS node - combines position and yaw data from the GNSS receiver and publishes this data to the rest of the ROS system.

Waypoint Upload node - allows the Husky interface computer to load a set of waypoints into to robot.

Guidance node - computes robot velocity needed to follow a set of waypoints.

3.4 Data Collection Software

A critical part of the mapping process is being able to accurately associate each image with the position and orientation of the camera at the time the image was taken. This process was accomplished using the Dynamic Sensing Program (DySense) which is an open-source data collection program that provides the means to obtain, organize, and geo-locate sensor data. This program was developed by the author in order to standardize data collection across various types of platforms and sensors. A screen shot of the DySense user interface can be seen in Figure 3.8.

Similar to how ROS splits different functionality into processes, DySense can split sensor drivers into processes which allows them to be written in any programming language. The camera sensor driver is implemented in *C#* and uses the EOS Digital Software Development Kit (EDSDK) to interact with the camera. This driver allows the images to be downloaded from the camera in real-time and assigns each one a unique file name and an estimated Coordinated Universal Time (UTC) stamp of when the image was exposed.

3.5 Additional Markers

In addition to the plant group QR codes, there are two other types of markers used in the mapping process. Row markers and plant markers.

3.5.1 Row Markers

Row markers are placed at the beginning and end of each row. Similar to identifying plant groups, these markers are also implemented using QR codes. The row codes store the row number and signify whether the code is the start ("St") or end ("En") of the row. As discussed in Section 3.1, it is critical to know the planting direction of each row because that

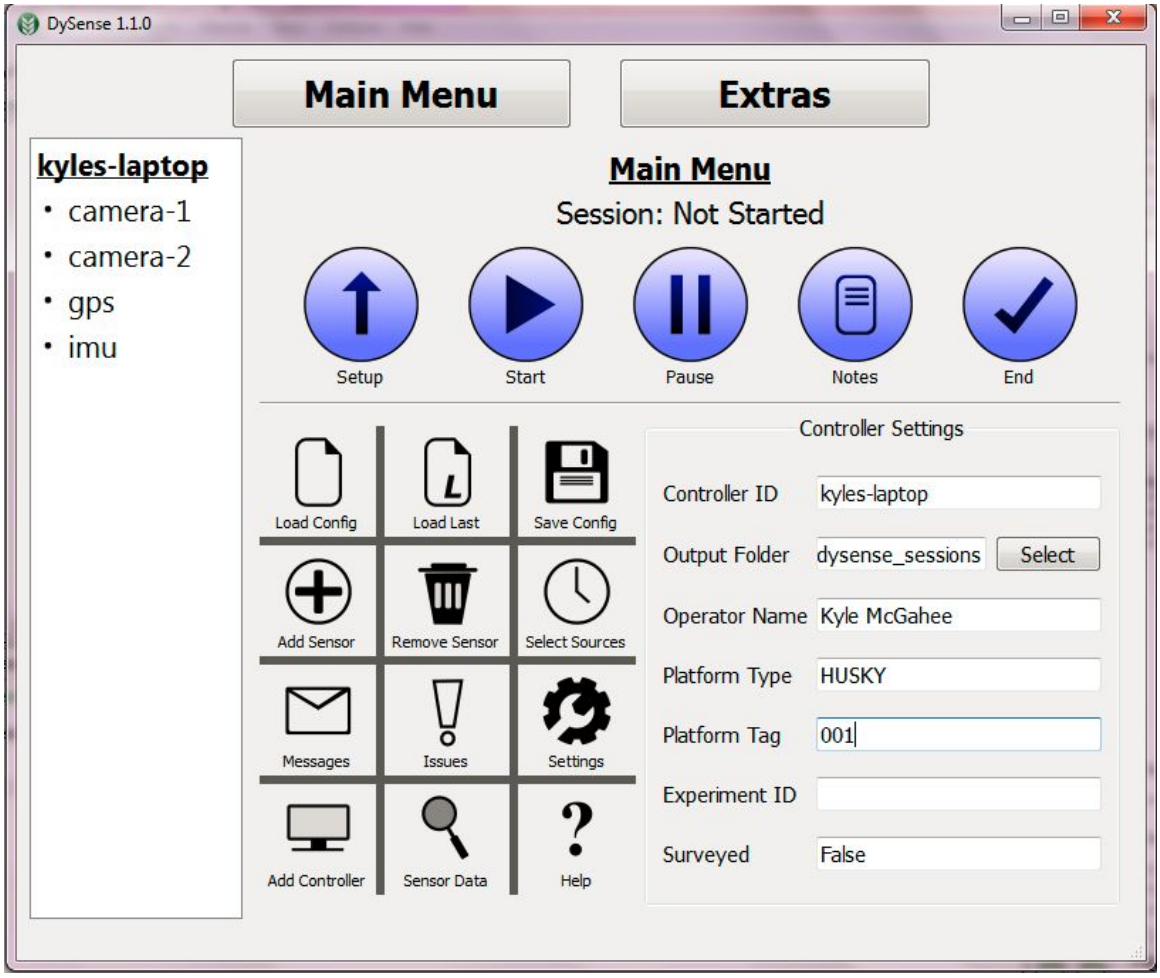


Figure 3.8: Screenshot of the data collection program.

Figure 3.9: QR codes marking the start and end of row 21.

defines which plants are associated with each group ID.

3.5.2 Plant Markers

The second type of marker is used to help distinguish plants from other debris that may be found in the field after tilling. This additional marker is optional, it but helps improve the robustness of locating plants. Depending on the size and quality the field these markers can be used on every plant or, for example, every 4 plants.

One type of plant marker is a blue dyed wooden stick approximately 5 inches in length which is placed in the center of each plant. The color blue is selected because it provides the largest difference between other hues likely found in the field, such as yellow/green in plants and red in soil. In addition to marking the plants, this stick also helps prevent the plants from flipping over when exiting the planter.

Another type of plant marker worth mentioning is a colored tag pierced through the top of an un-dyed wooden stick. This type of marker can provide addition information for manual plant inspection and is much more saturated than the dyed sticks, making it easier to detect in the post-processing pipelines.

Figure 3.10: Examples of blue stick (left) and colored tag (right)

Chapter 4

Post-Processing Pipeline

After the images are collected they must be converted into a field map. This task is accomplished by a set of scripts that are run in a sequential, or pipeline, fashion, where the output of one script is used as the input to the next. This chapter provides a general overview of the post-processing pipeline followed by a detailed explanation of each step.

4.1 Pipeline Overview

In this pipeline each script is referred to as a stage, where each stage accomplishes one specific task. The main reason the post-processing is split into separate stages is several stages take a significant amount of time to run, so it's beneficial to not re-run the entire pipeline when changes are made to one stage. The task that each stage accomplishes is:

Stage 0 Calculate the position and orientation of each image.

Stage 1 Find and read QR codes in all images.

Stage 2 Determine the row structure of the field using the QR codes.

Stage 3 Detect leaves and plant markers in each image.

Stage 4 Cluster plant parts from stage 3 into possible plants, and filter out unlikely plants.

Stage 5 Assign individual numbers to plants and save the final field map to a file.

The output of each intermediate stage consists of objects that directly relate to the field, for example QR codes, plants, or rows. These objects are serialized into a single output file which makes it trivial to pass these objects from one script to another.

Every stage of the pipeline is written in the Python programming language, and all of the image-processing algorithms are performed using the Open Source Computer Vision library, also known as OpenCV. The location of the post-processing code is listed in Appendix B.

4.2 Stage 0 - Calculating Camera State

The first step in the post-processing pipeline is to calculate the camera's position and orientation when each image was taken. This is performed by the data collection program since it is a general process that's useful for many other types of sensors in addition to cameras. However, after this calculation is done the format of the camera position is in latitude, longitude, altitude, and the format of the orientation is Euler angles. Therefore, this initial stage must convert the camera position to the modified UTM coordinates discussed in Section 2.2.2, as well as calculate the homography matrix defined by equation 2.5.

Even though the modified UTM coordinates use the ground reference for the z component, the height above the ellipsoid is also tracked for each image and item mapped in the field. This information could be used to estimate relative differences in water content in different parts of the field due to changes in elevation.

4.3 Stage 1 - Extracting QR Codes

The initial goal after calculating the position and orientation of each image is to detect and read all QR codes in the image set. This process consists of four steps which are applied to every image.

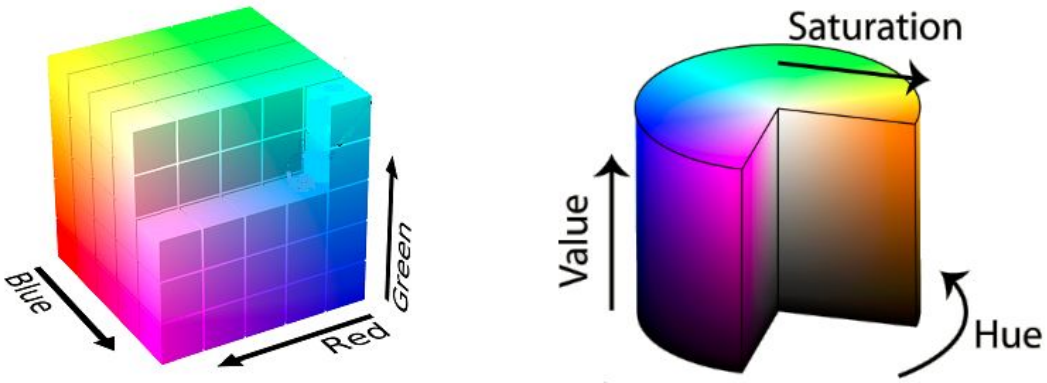


Figure 4.1: Visualization of BGR (left) and HSV (right) color spaces. Images obtained under the Creative Commons License from the Wikimedia Commons.

4.3.1 Converting Color-Spaces

The first step is to convert the image from the default color-space, which is Blue Green Red (BGR), to the Hue Saturation Value (HSV) color space. As can be seen in Figure 4.1, this HSV space is a cylindrical coordinate system which separates image intensity from color information. This makes colors more robust to changes in lighting, and the angular hue component is better suited for describing how humans perceive the color spectrum.

An important consideration is that most digital cameras perform a gamma-encoding when processing data from the imaging sensor. This is a non-linear operation which maximizes the amount of intensity information that can be stored for each pixel, as it pertains to human perception. However, human perception is not a valid concern in this post-processing application, and these non-linearities should be removed by properly decoding each color component. This was not realized until after the research was completed, and thus it is not implemented in the pipeline.

4.3.2 Thresholding

The second step is to separate the white QR codes from the rest of the image. This is accomplished by applying a range threshold for each of the HSV components. This threshold will output a 1 (white pixel) if the all three components are in the specified range, otherwise

Component	Min Value	Max Value	Notes
Hue	0	179	Include all hues
Saturation	0	65	Avoid saturated colors
Value	160	255	Avoid dark colors

Table 4.1: HSV range for detecting QR codes.

Figure 4.2: Binary image resulting from the HSV range filter. The overlaid red rectangle shows the rotated bounding box associated with QR code.

it will output a 0 (black pixel). In OpenCV hue is defined in the range of 0 to 179, and both saturation and value have a range of 0 to 255. The range that was experimentally determined for separating QR codes is shown in Table 4.1.

This threshold results in a binary image where pixels that are mostly white, such as QR codes, are all white, and everything else is all black. An example of a thresholded image can be seen in Figure 4.2.

4.3.3 Filtering by Bounding Boxes

The third step is finding the set of external contours, or outermost edges, of each object in the binary image. Each set of contours is then assigned a minimum bounding box which is the smallest rotated rectangle that encompasses the entire object. An example of a bounding box can be seen as a red rectangle in Figure 4.2. These bounding boxes are filtered to remove

Figure 4.3: Sequence of steps applied to objects that are potential QR codes. First a global or adaptive threshold is applied, and then the ZBar program returns the text stored in the code. In this case it would return 697.

Figure 4.4: Comparison showing how an adaptive, rather than a global, threshold can make a code readable by keeping the bottom right corner intact.

ones that are either too small or too large to possibly be a QR code.

4.3.4 Reading QR Codes

The final step is to use these bounding boxes to extract sections of the original image to run through the code reading program. From the open-source programs that were evaluated, the ZBar program provided the best results. However, the ZBar program requires a grayscale or binary image, and thus a threshold must be applied to the extracted color image.

The HSV range threshold is effective for finding possible codes, but it does not do a good job maintaining the grid of white and black squares that make up the QR code. Instead, the BGR image is converted to an intensity image, and if a simple global threshold does not result in a successful reading of the code then an adaptive threshold is tried instead. This results in a readable code even if there is noticeable image glare as seen in Figure 4.4.

The data returned by the ZBar program is used to determine if the code corresponds to a plant group or to a row marker. If no data, or bad data, is returned by the ZBar library then the extracted image is saved for the operator to review after the stage has completed. This makes it much easier to find misread or damaged codes and to manually add them into the final results. If the code is read successfully then its world coordinates are calculated and saved in a list of QR codes. If the same code is detected in multiple images then the world coordinates of each code reference are averaged to improve the mapping accuracy.

4.4 Stage 2 - Creating Field Structure

The second stage of the pipeline involves assigning row numbers to each QR code, and then creating plant groups that can span multiple rows. As an optional input to this stage the user can specify a file containing any codes from the previous stage that were not automatically detected.

4.4.1 Assigning Codes to Rows

This stage begins by pairing the row markers associated with the start and end of each row. Since the locations of the marker codes are calculated from the images, the average row heading can be calculated. This row heading is then used to transform the UTM coordinates into the field coordinate frame discussed in Section 2.2.3.

The next step is to assign each group QR code to a specific row based on its field coordinates. As some rows can span several hundred meters, it's not always possible to assign a code to the nearest row defined by a vector between the row start and end codes. Instead a sweeping algorithm is used. The idea is to sweep across the field, from left to right, and incrementally add QR codes to each row as shown in Figure 4.5. Once a code is added to a row it splits that row into smaller segments. A code is assigned to the row which has the closest segment based on the lateral distance to the code's field location.

This algorithm can effectively account for small amounts of curvature in the rows, but it

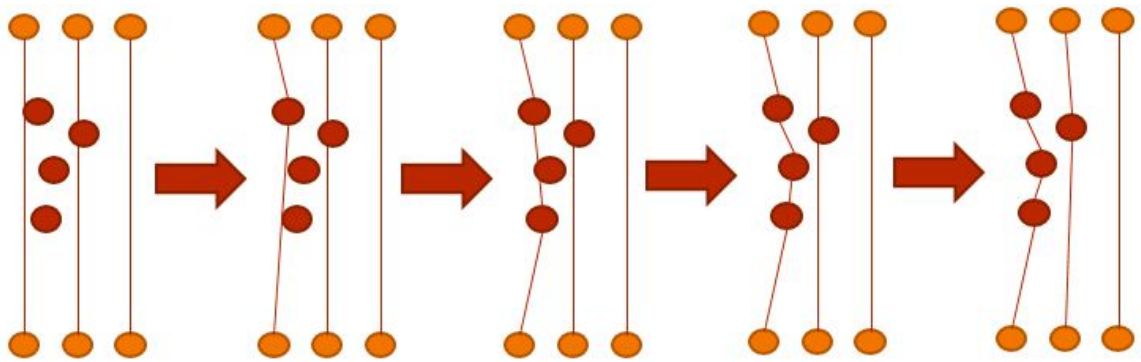


Figure 4.5: Example of the sweeping projection algorithm. Row codes are shown in orange and four group codes in red. At each step the left-most group code is assigned to the nearest segment, shown as a red line. The result is three codes belong to the left row, and the fourth group code belongs to the middle row.

is not designed to work on rows that are not planted in nominally straight lines.

4.4.2 Organizing Group Segments

Once all group codes are assigned to a row, it's possible to tell which codes come before and after one another. A group segment is defined by a beginning code and the next code that follows it in the direction of planting. This group segment is where the plants for a given code are located. It's possible, however, that when the transplanter reaches the end of a row that the current plant group isn't finished and continues into the next pass. A pass refers to the transplanter driving once down the field, so a 2-row transplanter would have a pass containing 2 rows. Therefore, the group segments at the end of corresponding rows are paired together into complete groups. An example of this is shown in Figure 4.6.

4.5 Stage 3 - Extracting Plant Parts

Similar to the process of extracting QR codes, this stage converts each image to the HSV color space, applies a range threshold, and filters the objects based on size. Since this stage is looking for plant leaves rather than white QR codes, it uses different ranges for the HSV components. These alternate ranges are defined in Table 4.2.

In addition to plant leaves, this stage also searches for plant markers. The values for the

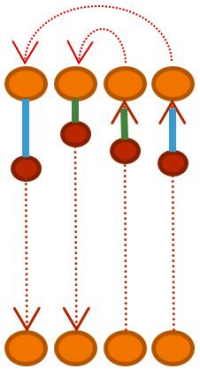


Figure 4.6: Example of four rows planted two rows at a time. The red dashed arrows show the direction of planting. There are four group segments, two blue and two green. The blue segments belong to the same plant group, and the green segments belong to a second plant group.

Component	Min Value	Max Value	Notes
Hue	35	90	Include green hues
Saturation	80	255	Include saturated colors
Value	20	255	Avoid very dark colors

Table 4.2: HSV range for detecting plant leaves.

two types of markers discussed in Section 3.5.2 are shown in the tables 4.3 and 4.4. The minimum saturation and value for the tag markers can be set much higher than the wooden blue sticks which leads to more reliable detection.

Similar to the QR codes these values are set based on experimentation. Unfortunately, good values for these thresholds are more likely to vary due to changes in external lighting or camera settings. This is because they depend on finding specific colors, rather than white.

Component	Min Value	Max Value	Notes
Hue	90	130	Include blue hues
Saturation	31	255	Include saturated colors
Value	16	255	Avoid very dark colors

Table 4.3: HSV range for detecting blue stick markers.

Component	Min Value	Max Value	Notes
Hue	15	45	Include yellow hues
Saturation	130	255	Include saturated colors
Value	100	255	Avoid dark colors

Table 4.4: HSV range for detecting yellow tags.

4.6 Stage 4 - Locating Plants

The most challenging aspect of the pipeline is reliably determining which plant parts found in the previous stage belong to the same plant, and which of those are actual plants that should be mapped. Plant parts refers to both the leaves of the plant as well as any plant marker associated with the plant. This is challenging because there is often unavoidable plant debris in the field that comes from the tilling right before planting. Plant markers, such as the blue sticks, help with this issue, but as discussed in Section 6.3 the blue sticks could not always be detected. In addition, for large experiments it may not always be feasible to have individual markers for every plant.

4.6.1 Hierarchical Clustering

The task of grouping plant parts into individual plants is done using a hierarchical clustering algorithm. In this application a cluster is represented by the minimum bounding rectangle of one or more plant parts. If two rectangles are clustered together the resulting cluster is represented by the smallest bounding rectangle that fits both the original rectangles. For simplicity the merged rectangle is a regular, non-rotated bounding rectangle.

This algorithm combines the nearest two clusters into a single cluster and keeps repeating this process until an end condition is met. The distance between two clusters is defined to be the smallest distance between any of the 4 rectangle corners. The end conditions are either (1) there is nothing left to cluster, or (2) the closest cluster is too far apart to be merged based on a user defined threshold. In addition, there is a maximum size limit on the clusters which is set to be the maximum expected plant size in the field. Finally, any tiny, unclustered plant parts are removed from the list of possible plants.

Figure 4.7: Example of hierarchical clustering. At each step the closest bounding boxes are merged together until there is one containing the entire plant.

4.6.2 Recursive Segment Splitting

After the clustering is completed for an entire plant segment, the list of possible plants is passed to a recursive splitting algorithm to filter out the real plants. This algorithm leverages information about the expected locations of plants as well as takes into account the plant markers found in stage 3. The splitting algorithm consists of the following steps for each segment:

Step 1 For each possible plant, calculate the lateral and projection distance to the segment.

Step 2 Remove any plants that do not fall within the segment based on the projection distance.

Step 3 Find the most likely plant based on various characteristics. This is described in more detail below. If there are no possible plants then create one in the next expected location based off the nominal transplanter spacing.

Step 4 Repeat the previous step, but starting at the end of the segment and find the next most likely plant by working backwards.

Step 5 Split the original segment into smaller segments using the most likely plants as new endpoints. If the new segments are too short to contain plants then the algorithm is finished. Otherwise, recursively go back to step 1 for each of the new segments.

The idea behind working from the both directions at the same time is the start and end QR codes are known locations in the field, and thus plants near them can be more reliably

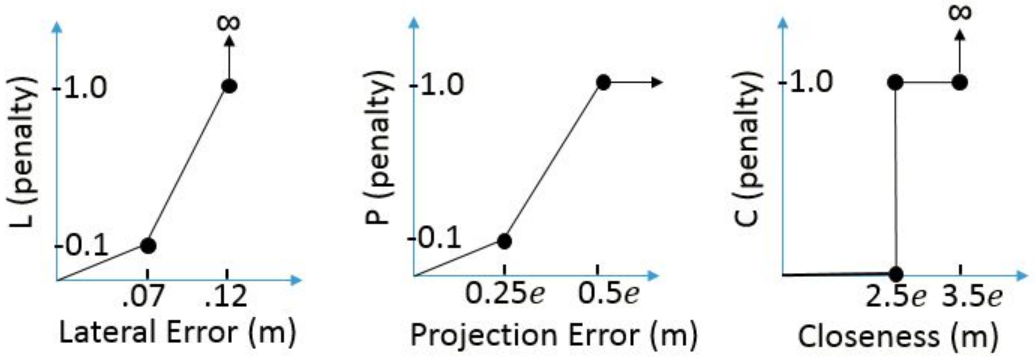


Figure 4.8: Piece-wise penalty functions. "e" represents the expected spacing between successive plants.

detected. In order to determine the most likely plant in step 3, each possible plant is assigned a penalty value. This value is calculated using

$$\text{Penalty} = (s_L\text{L} + s_P\text{P} + s_C\text{C}) \;/\; (s_B\text{B})$$

where the variables are described below.

Lateral Error (L) How far off the plant is from the expected line segment.

Projection Error (P) How far away the plant's projection onto the segment is from where the closest expected plant would be.

Closeness (C) How far away the plant is from the start/end of the segment, with the idea that the lateral and projection errors become less reliable the farther away the plant is from a known item's location.

Plant-Part Boost (B) Based on what types of plant parts are found in the plant cluster.

Scales (s_L, s_P, s_C, s_B) Relative weightings to change importance of each penalty.

The individual penalty components are calculated with the piece-wise linear functions displayed in Figure 4.8. The plant-part boost (B) is calculated as a product of additional scales for each plant part type. If a certain type of plant, for example a blue stick, is missing from a possible plant, then its scale is set to 1 so that it has no effect.

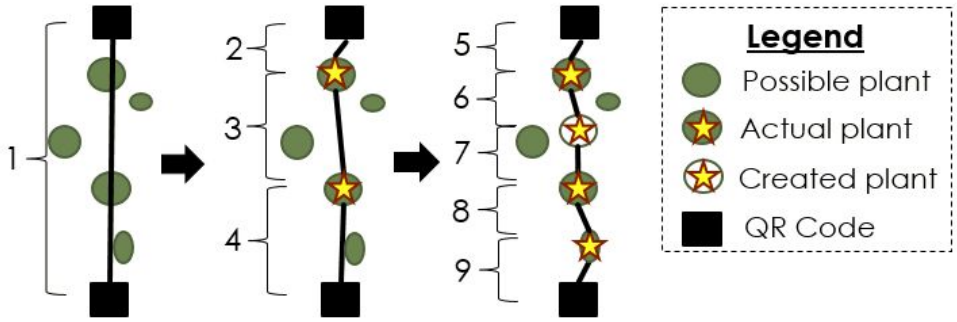


Figure 4.9: Example of the recursive splitting algorithm finding most likely plants within a group segment.

The lateral and closeness penalties are defined as infinity if the input exceeds a certain threshold. If this occurs then the overall penalty will also be set to infinity, and that plant will be removed from the list of possible plants. If this results in no plants to select from then a plant will be created and placed in the expected location. These are referred to as "Created Plants", and may occur due to a plant being dead, buried, or skipped during planting.

A simple example of the algorithm is shown in Figure 4.9. In the first image there is one segment between the two QR codes. The second image shows the result of the splitting algorithm running on segment labeled #1. It selects one plant based on the bottom code and one based on the top code. Segment #2 is determined to be too short to contain more plants, however the algorithm is recursively run again on segments #3 and #4. This results in the third image, which now contains five segments. One plant had to be created because the other two possible plants on segment #3 had too much lateral error to be considered.

Similar to the sweeping projection algorithm used to assign row numbers, this algorithm can effectively account for slight curves within a group segment since plants are incrementally determined starting at the end points.

4.7 Stage 5 - Saving Field Map

The final stage in the pipeline is generating a field map file. This stage begins by assigning every plant in the field a unique number so it can be easily referenced in a database. This numbering begins with plant #1 at the start of the first row and then follows a serpentine

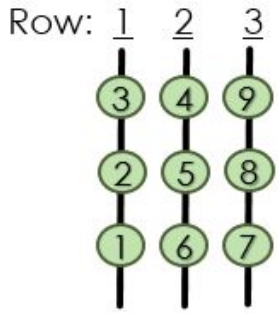


Figure 4.10: Example of serpentine numbering for nine plants.

pattern which can be seen in Figure 4.10. This numbering system is chosen because it represents how a person is likely to inspect plants when walking through the field.

One complication that is addressed in this final stage is that there may be a relative shift between the coordinates calculated by the mapping platform and the desired coordinates. This is due to the fact that RTK systems are only accurate relative to the base station. If the location of the base station isn't accurately surveyed then the field map will contain an offset in the both the northing and easting directions.

In order to account for this offset, a file can be provided as input to the stage that contains the coordinates of surveyed QR codes relative to the desired reference station. The average offset between these surveyed codes and the mapped codes is computed and every item in the map is translated by this amount. After this correction is complete all the codes and plants are written to a Comma Separated Value (CSV) file. This is a common type of file and can be used with many types of programs, or it can be easily imported into a database.

Chapter 5

Experimental Setup

The mapping system was evaluated at the Land Institute on a type of intermediate wheat grass, commonly referred to as Kernza. The Land Institute is a research organization located in Salina, Kansas, focused on using perennial crops to develop sustainable farming practices. The experiment, which was overseen by researcher Lee DeHaan, consisted of approximately 25,000 plants that were transplanted during the middle of October. Details about how the field was setup as well as different robot and camera settings are discussed in this chapter.

5.1 Field Setup

A two-row model 5000WD planter from Mechanical Transplanters was used to transplant the Kernza seedlings into the field. The transplanter works by a series of six deposit cylinders rotating at a constant rate. When a cylinder passes over the correct spot, its bottom flap is opened and the plant is ejected. The yellow water tanks seen in Figure 5.2 are used to automatically apply a small amount of water to the plant, and the rolling wheels help press the soil down.

The six deposit cylinders rotate so that each cylinder is spaced approximately 12 inches apart. For this experiment, plants were placed in every other cylinder which resulted in 24 inches between successive plants. The QR codes indicating the start of a new group are placed

Figure 5.1: Example of kernza seedling after transplanting. Image taken from the robot as it was driving through the field.

Figure 5.2: Two-row transplanter used for planting. Plants shown in the trays are a type of sunflower crop, not Kernza.

in the empty cylinders between two plants. This is so that the use of QR codes does not increase the overall size of the field. The spacing between the two rows on the transplanter was set to 36 inches, and this same spacing was used between each pass. Therefore, the entire field had consistent row spacing.

The plants were split into 97 individual rows, roughly 180 meters long. After the field was planted, the QR codes marking the start and end of each row were manually placed. The field contained a total of 4581 QR codes. Of these codes, 3619 corresponded to a plant grouping containing only a single plant. The other 962 codes corresponded to plant groupings containing, on average, 25 plants.

5.2 Mapping Time

An important decision for the mapping system is whether to collect images during the day or at night. The biggest challenge of mapping during the day is the ability to provide consistent lighting. Changes in scene lighting, for example between clouds and direct sun, can affect the color of plants and blue sticks and potentially cause under or over-exposure of QR codes. Daytime lighting can be made more consistent by using a shade cover over the field of view of the images. This, however, depends on the sun being high enough in the sky for the shading to work, which in turn limits the total amount of time images can be collected each day.

The shade size can always be increased or placed at an angle to account for the sun angle, but this increases cost, design complexity, and makes the platform more susceptible to wind gusts. This time-limited window to collect images can be problematic due to the potential for storms to affect the QR codes. Even if the codes are firmly planted in the ground and do not blow away, heavy rainfall can splash mud onto the codes decreasing the chance they'll be readable.

Since the experiment was performed in October, when the sun only reaches a maximum altitude angle of 42 degrees in Salina, the decision was made to test the mapping process at night. One disadvantage to mapping at night is the potential for the external lighting to attract insects which can fly into the camera's field of view. From preliminary tests the

LED bars did attract a few insects, but not enough to negatively impact the usefulness of the images.

5.3 Camera Setup

The correct choice of camera options is another important step in making the mapping system robust for a given experiment. Some of these options will vary in how they're determined between day and night-time mapping, because at night the available light is more restricted. Since this experiment was performed at night, that is what is discussed in this section.

The first option selected is the shooting mode, as that defines what other settings are available to change. The preferred shooting mode is Manual because that will allow the exposure time, aperture, and sensitivity to be set to constant values. Under controlled lighting conditions this will keep a consistent image intensity and prevent additional latency due to the camera calculating these settings before each image.

The maximum exposure time is determined based on the speed of the robot and maximum allowed translation in the QR codes. If the exposure time is set too high the black and white squares making up the codes will start to blend together, rendering the code unreadable. It was experimentally determined that an exposure time of 2.5 milliseconds, or 1/400 seconds, was sufficient for quality images. At a nominal speed of 40 cm/sec, and yaw rate of 10 degrees/sec, this results in a translation of 1.4 millimeters during the exposure time. Even though this is approximately the same size as the grid squares, there are enough pixels per square that the blending was negligible.

The aperture is set to a large diameter to maximize the amount of light reaching the sensor since the exposure time is relatively short for the amount of light available. When the lens is fully open (an f-stop of f/2.8) the depth of field is noticeable reduced, and it becomes more difficult to keep the QR codes in focus as the camera height slightly varies moving through the field. Therefore, an f-stop of f/4 is preferable as it provides a good trade-off between depth of field and exposed light. In addition, compared to f/2.8 an aperture of f/4 will have less noticeable lens effects such as distortion and vignetting which results in higher

quality images.

The light sensitivity of the sensor, commonly specified as an International Standards Organization (ISO) rating, is set last by inspecting the image in the field to achieve a desirable scene brightness. Using the two LED bars per camera required an ISO of 1000. If the ISO is set too high then sensor noise becomes significant. If too high of an ISO is required then the aperture can be made larger, or the external lighting can be increased.

The white-balance is set to a fixed setting chosen to match the same temperature of light as the LED bars, which is listed in the product description as 6000 kelvin. Many cameras offer both a Flash and Cloudy white balance that are both centered around 6000 kelvin. Preliminary field tests indicate both options produce extremely similar results, so the Flash setting was used. Auto-white balance mode should not be used, as most images will not contain a QR code for reference, and as a result many images of plants will vary in chromacity.

Finally, the image format is selected between raw and the Joint Photographic Experts Group (JPEG) format. Raw images store the pixel readings directly from the sensor, thus no information from the exposed image is lost. The JPEG format on the other hand typically merges adjacent pixels from the Bayer filter and applies a compression algorithm to reduce the file size. Even though the compression algorithm produces image artifacts that reduces the quality of the images, it was determined this does not reduce the effectiveness of the post-process pipeline. For the Canon 7D camera, raw images are around 15 megabytes larger then JPEG and require a conversion to a bitmap format before being useful for post-processing. This conversion requires approximately 5 to 10 seconds per image and adds a significant amount of additional processing time. For these reasons the JPEG format was used for this experiment.

5.4 Robot Operation

In Section 3.3.1 it was discussed that the robot can operate in either cruise control mode or a fully autonomous mode. In this experiment the robot was used in cruise control mode,

Setting	Value
Camera Mode	Manual
Aperture	f/4
Exposure	1/400 seconds
ISO	1000
White Balance	6000 K
Image Format	JPEG

Table 5.1: Camera settings for night-time mapping.

mainly because at the time there were not any tools available to view and edit the waypoints used by the robot. When generating a set of waypoints from the transplanter path, the waypoints at the row ends need to be edited to prevent the robot from driving in too rough of field conditions, as well as to prevent it from turning before the end of the row and running over plants. Even though the robot was operating in cruise control mode, the path of the robot was recorded, and it's possible to have the robot retrace this path in fully autonomous mode.

Chapter 6

Results and Analysis

The image collection step of the experiment was successful, as was the effectiveness of the post-processing pipeline. This chapter first discusses the final field map and then analyzes the different components of the post-process pipeline. This includes QR code detection as well as the recursive splitting algorithm for finding plants. The chapter concludes by looking at the accuracy of the mapped plants and codes and a timing analysis of the entire mapping process.

6.1 Field Map

The final field map produced from the Kernza experiment is displayed in Figure 6.1. This map is in the field coordinate system and shows the field is split into two different sections. The split in the field was intentionally left un-planted to allow vehicles to drive from one end to the other. The rectangle containing 15 rows labeled as 'special plants' refers to the plants that each have their own unique ID, as discussed in Section 5.1. These were planted first, followed by the right portion of the field, and finally the left section. This explains why the two rows on the far left are 50 meters shorter than the rest. In addition, the right part of the field was shifted upwards 10 meters to avoid low quality soil.

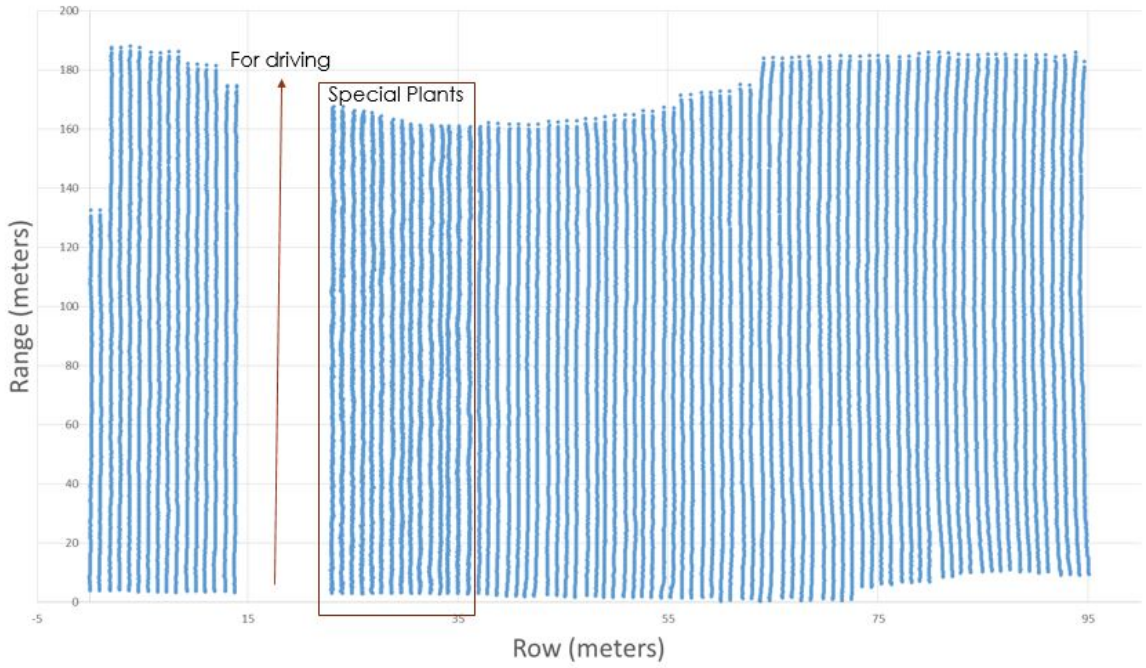


Figure 6.1: Field map of Kernza showing both plants and QR codes as small blue circles.

6.2 Code Detection

The QR code detection proved to be robust. From the 4581 codes in the field, all but 2 were automatically detected by the first stage of the post-processing pipeline. It is unknown why the 2 undetected codes were missed, but the most likely reasons are they were accidentally buried or did not appear in any images.

However, 43 codes were not able to be read by the ZBar program. Since each of the unreadable codes were placed in a special review directory it was straightforward to determine the reason each code couldn't be read. The most common cause was a printer error which resulted in a small section of the code to not be printed. This printer error affected 25 of the 43 unreadable codes. 9 codes were partially blocked by field debris or insects sitting on the code. 7 had dirt on the code, which most likely splashed up during transplanting. And finally, 2 codes had too much glare to be read successfully. Examples of each of these issues can be seen in Figure 6.2.

These unreadable codes were exported with their pixel coordinates and image name which made it trivial to include them in the final results. The codes affected by printer error could

Figure 6.2: Unreadable codes. From left to right on top row: printer error, blocked by debris, blocked by fly. On bottom row: dirt, glare.

be reduced by better inspection of the codes before planting or by using a different type of printer. The glare issue could be eliminated by using a polarizing lens filter, however this would have the potential to change the white balance setting and also possibly reduce image sharpness. Since the glare only affected 2 codes it is likely not worth the cost to address.

6.3 Plant Localization

The number of plants in each group were manually counted before transplanting. However, it is unknown exactly how many plants ended up in the field because some plants were discarded during the transplanting process. So the sum of the pre-counted plants, 25560, is the maximum number of plants that could be found in the field.

The plant localization algorithm detected 24269 plants in the images and created 335 plants where no plant was detected, but where one should have been. This resulted in a total of 24604 plants in the final map, which indicates 956, or roughly 1 out of every 25, plants were discarded during transplanting. These results were considered reasonable by the persons responsible for planting.

Blue sticks were only found in 23% of the plants that were detected in the images, but

all plants should have had a blue stick marker. The reason for this poor result was due to the lack of saturation in the blue sticks which made it difficult to find a threshold that didn't also detect the blue hues in the soil. Also, there were variations in the intensity of the blue dye color that made it difficult to find a single set of threshold values for detecting the markers.

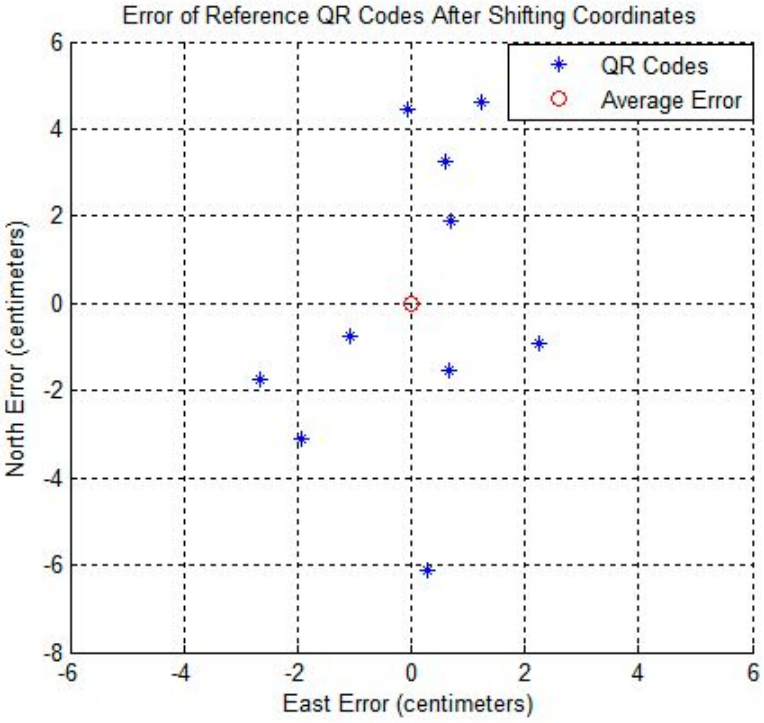
The assigned coordinates of the subset of the 335 created plants were projected back onto the images that contained those world coordinates, and an arbitrary 10 centimeter box was drawn on the image to indicate where the plant should have been. Fifty of these images were manually analyzed and 46 of them closely matched an actual plant in the image that was either dead or mostly buried under soil. The remaining 4 plants were either completely buried or a gap occurred in the field where nothing was planted.

6.4 Mapping Accuracy

As discussed in Section 4.7, the coordinates of all the plants and QR codes had to be offset into the coordinate frame used by the Land Institute. This was accomplished by manually surveying 10 codes using the Land Institute RTK base-station, and then calculating the northing and easting offsets that would make the average error zero.

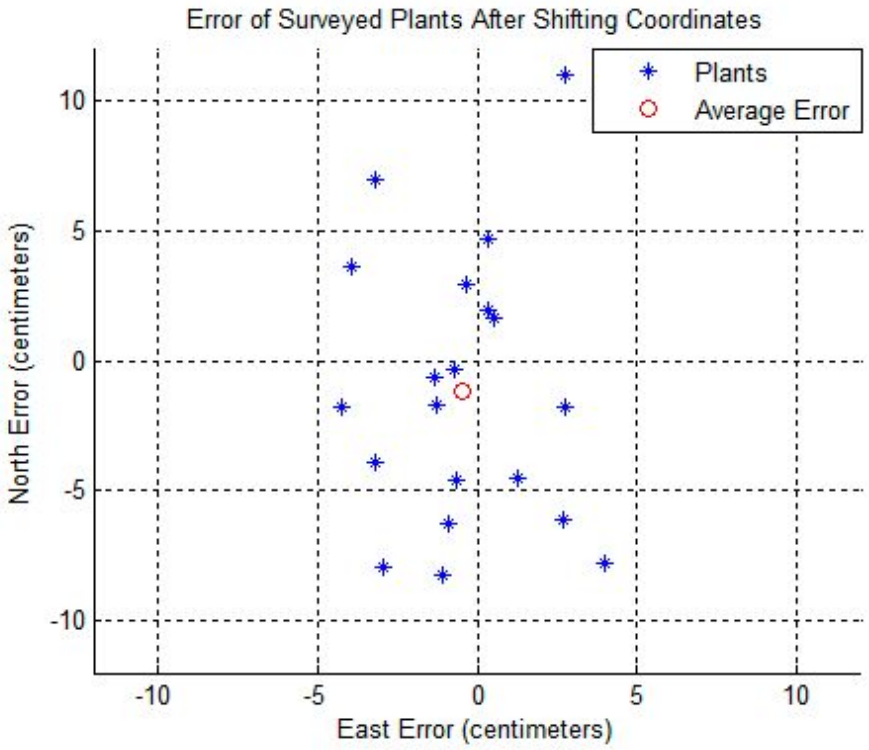
After this offset was applied, the errors of the 10 surveyed codes were calculated and plotted in Figure 6.3. The reason that codes were used to calculate the offset, rather than plants, is it's easier for the post-processing pipeline to consistently identify the center of the code. The non-absolute, average error is zero because that is how the shift in coordinates was calculated. An additional check would have been to use the robot's RTK base-station reference to also manually survey the codes, but this was not realized until afterwards.

An additional 20 plants were surveyed to verify that their mapped positions were accurate enough to distinguish them from neighboring plants. As with the surveyed codes, the plants were chosen randomly and evenly distributed throughout the field. A similar plot showing the errors for each surveyed plant can be seen in Figure 6.4. The average error is relatively small which shows the offsets calculated using the codes were not biased.



Error	East (cm)	North (cm)
Average	0	0
|Average|	1.4	3.4
Std. Dev.	1.5	3.5
|Maximum|	2.8	7.2
|Minimum|	0.5	0.8

Figure 6.3 & Table 6.1: Errors in reference QR codes after shifting to the Land Institute coordinate system.



Error	East (cm)	North (cm)
Average	-0.5	-1.2
|Average|	2.1	4.5
Std. Dev.	2.4	5.3
|Maximum|	4.3	10.9
|Minimum|	0.3	0.6

Figure 6.4 & Table 6.2: Error between mapped and manually surveyed plants.

The largest source of errors from these measurement were errors in geo-referencing each image. This was due to timing errors which were primarily caused by camera latency as well as latency in the GNSS receiver. Camera latency is the amount of time between when the data collection program commands the camera to take an image, and when the image is actually exposed. GNSS latency is the amount of time between when a position is calculated by the receiver, and when it is received by the data collection program.

From experimental testing the camera was determined to have a latency following a normal distribution with a mean of 65 milliseconds and a standard deviation of 30 milliseconds. The mean latency can be accounted for in the data collection program, but the variance can not. The GNSS receiver was configured to output at 50 hertz and a baud rate of 115,200 bits per second. This results in an estimated latency in the range of 20 to 30 milliseconds. Combining these latencies with the platform motion results in an error of:

$$\begin{aligned} \text{error} &= L(s + \dot{\psi} * d) \\ &= .06 \text{ sec} * (40 \text{ cm/sec} + 0.2 \text{ rad/sec} * 80 \text{ cm}) \\ &= 3.4 \text{ cm} \end{aligned}$$

where L is the estimated combined latency, s is the nominal platform speed, $\dot{\psi}$ is the average yaw rate needed to keep the platform between the rows, and d is the distance from the platform center of rotation to the camera. This assumes the camera's have a level orientation, which is the case for the robot. While this is a rough estimate it shows that these timings errors are a significant contribution to the overall error.

The contributions of error from vehicle speed is twice as large as that from the platform rotation. Forward motion of the robot was primarily in the northing direction which explains why these errors are around twice as large as the easting direction. Another contributing factor is the estimation error in the GNSS position and heading. The additional errors in the surveyed plants compared to the codes is likely due to a larger discrepancy between what the post-processing pipeline detects as the center of a plant and what a human measures as

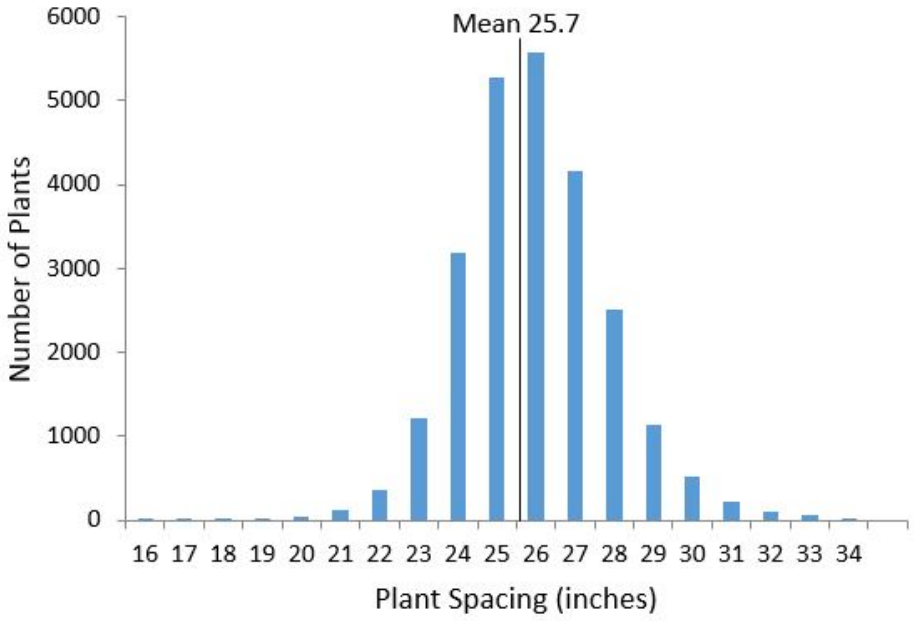


Figure 6.5: Histogram of spacing between successive plants.

the center of the plant.

6.5 Plant Spacing

An additional check to verify the results is that the plant spacing should be consistent with the output rate of the transplanter. The histogram in Figure 6.5 shows the distribution of plant spacing for all the plants. The mean plant spacing is 25.7 inches which is close to the expected 24 inches. However, the variances in plant spacing highlights the importance of mapping individual plants and not assuming the transplanter outputs at a consistent rate.

6.6 Time Analysis

Another consideration to the success of the mapping system is how much time it takes to collect and analyze all of the images. For this experiment, the robot speed and camera triggering rate were set as described in Section 3.2.4. This resulted in a total driving time of approximately 7 hours spread over 4 different nights. Each night required 30 minutes for

Stage	Time Per Image (sec)	Total Time (hours)
Extracting QR Codes	0.9	13.6
Creating Field Structure	-	0.05
Extracting Plant Parts	1.0	15.1
Locating Plants	0.005	3.0
Saving Field Map	-	0.05
Total	**1.9**	**31.7**

Table 6.3: Timing analysis for each stage of the post-processing pipeline.

setup and another 30 for tear-down, which added in a total of 4 additional hours.

During the 7 hours of driving, the robot collected 54,512 images. The timing analysis for the post-processing are presented in Table 6.3. These results were bench-marked on a Lenovo S431 ultra-book running a dual core i7-3687 at 2.6 gigahertz with 8 gigabytes of random access memory. One thing to note is it's straightforward to run the first stage in parallel for each night that images were collected. Also, if extracting plant parts were combined with the first stage, this would result in a large reduction in overall processing time.

While the post-processing adds a significant amount of additional time to the mapping process, it is almost completely automated. Therefore, only considering the 11 hours of manual work results in a mapping rate of 1.5 seconds per plant. Researchers at the Land Institute estimated that the previous method of manually surveying each plant required roughly 5 to 10 seconds per plant.

Chapter 7

Conclusion

The results of this research show that the image-based mapping system is a viable option for locating and identifying individual plants. The average absolute Euclidean error of 5.1 centimeters is comparable to manual surveying errors, and the worst measured longitudinal error of 10.9 centimeters is well within the maximum error allowed to differentiate adjacent plants. The group identification method successfully located all but 2 QR codes, which is a success rate of 99.96%. While the mapping process was successful, other important conclusions regarding the platform, plant markers, cameras, and system-complexities can be drawn from the research and are discussed below.

The image-based mapping is mostly independent of the base platform that is used, and the robot could easily be switched out for a simpler platform such as a manual push-cart. A robot was used in this experiment based on the potential to automate the image collection step. This research, however, indicated two benefits of having a human present in the field during the image collection. First is to detect if a camera stops taking pictures due to an internal error, which did occur several times during the experiment, and second is the ability to fix QR codes that are partially buried and covered by field debris. The first issue could be mitigated by providing automatic feedback between the robot, user, and data collection program, and the second could be solved by walking through the field and fixing these codes before the robot makes its passes. Both of these steps would be required for robust

automated image collection.

The least successful aspect of the experiment was the use of dyed blue sticks acting as plant-markers. These wooden sticks suffered from inconsistent intensity and didn't reflect enough light to appear saturated in the images. An alternative option is to use colored, plastic square markers that are pierced through un-dyed wooden sticks. To avoid large amounts of plastic being distributed in the field these markers would not need to be on every plant, but rather every 4 or 5 plants. If there is little field debris or weeds then these plant markers may not be necessary at all.

The limiting factor on how quickly images could be collected was the maximum trigger rate of the Canon 7D cameras. Additionally, these cameras had an indeterminate latency that led to positional errors and suffered from occasional triggering errors. A potentially better option would be to use a camera originally designed to transfer images over a Universal Serial Bus (USB) port and equipped with a high-resolution lens. These cameras typically have much higher capture rates and were intended to be used for scientific or robotic applications, unlike the Canon cameras which were primarily designed for photographers.

Perhaps the biggest conclusion drawn from the research is the importance of managing complexity in an application that is primarily intended to be used by researchers. Using images adds technical complexity in the fact that camera settings have to be setup carefully, and the thresholds in the post-processing have to be chosen to work the color spectrum of the images. These thresholds can vary over multiple experiments due to slight changes in environmental lighting, and it's difficult to find a set of values that work well for all scenarios.

Regardless of what type of equipment is selected to collect images, this thesis presents important first steps in creating a robust mapping system for not only mapping transplant coordinates but also assigning them to different groups. This functionality will potentially allow field sizes to scale up while also leveraging automated technologies needed to meet the ever growing demand for food.

Bibliography

[1] M. Tester and P. Langridge, "Breeding technologies to increase crop production in a changing world," *Science*, vol. 12, no. 327, pp. 818–822, 2010.

[2] M. Perez-Ruiz, D. Slaughter, C. Gliever, and S. Upadhyaya, "Tractor-based real-time kinematic-global positioning system (RTK-GPS) guidance system for geospatial mapping of row crop transplant," *Biosystems Engineering*, vol. 111, no. 1, pp. 64–71, 2012.

[3] J. Carballido, A. Rodrguez-Lizana, J. Agera, and M. Prez-Ruiz, "Field sprayer for inter- and intra-row weed control: Performance and labor savings," *Spanish Journal of Agricultural Research*, vol. 11, no. 3, pp. 642–651, 2013.

[4] T. Bakker, K. Asselt, J. Bontsema, J. Mller, and G. Straten, "Systematic design of an autonomous platform for robotic weeding," *Journal of Terramechanics*, vol. 47, no. 2, pp. 63–73, 2010.

[5] A. Ruckelshausen, P. Biber, M. Dorna, H. Gremmes, R. Klose, A. Linz, R. Rahe, R. Resch, M. Thiel, D. Trautz, and U. Weiss, "Bonirob: An autonomous field robot platform for individual plant phenotyping," pp. 841–847, 2009.

[6] H. Sun, D. Slaughter, M. Ruiz, C. Gliever, S. Upadhyaya, and R. Smith, "RTK GPS mapping of transplanted row crops," *Computers and Electronics in Agriculture*, vol. 71, no. 1, pp. 32–37, 2010.

[7] M. Nrremark, H. Sgaard, H. Griepentrog, and H. Nielsen, "Instrumentation and method for high accuracy geo-referencing of sugar beet plants," *Computers and Electronics in Agriculture*, vol. 56, no. 2, pp. 130–146, 2007.

[8] M. Ehsani, S. Upadhyaya, and M. Mattson, "Seed location mapping using RTK GPS,"

Transactions of the American Society of Agricultural Engineers, vol. 47, no. 3, pp. 909–914, 2004.

[9] D. Andjar, H. Moreno, C. Valero, R. Gerhards, and H. Griepentrog, "Weed-crop discrimination using LiDAR measurements," pp. 541–545, 2013.

[10] M. Kise, Q. Zhang, and F. Rovira Ms, "A stereovision-based crop row detection method for tractor-automated guidance," *Biosystems Engineering*, vol. 90, no. 4, pp. 357–367, 2005.

[11] B. Panneton and A. Bizeau, "Merging RGB and NIR imagery for mapping weeds and crop in 3D," vol. 5, pp. 3651–3657, 2014.

[12] P. Soille, "Morphological image analysis applied to crop field mapping," *Image and Vision Computing*, vol. 18, no. 13, pp. 1025–1032, 2000.

[13] A. Briney, "Geodetic datums." `http://geography.about.com/od/geographyintern/a/datums.htm`, 2016. [Online; accessed 2016-09-15].

[14] P. Coorke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, vol. 1st Ed. of *Tracts in Advanced Robotics*. Springer, 2013.

[15] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," vol. 1, pp. 666–673, 1999.

[16] Denso Corporation, "History of the QR code." `http://www.qrcode.com/en/history/`, 2014. [Online; accessed 2016-09-28].

Appendix A

Symbolic Projection Verification

Matlab Code

```
% Author: Kyle McGahee
% Description: Show that the two methods for coordinate
    projections are symbollically equivalent.

clear all

%% Method 1 - Euclidean Space

% Pixel coordinates and camera parameters.
syms u v f pw ph u0 v0

% Pixel to sensor frame
sensor = [0 -ph; pw 0]*[u; v] + [ph*v0; pw*u0];

% Rotation matrix
```

```
syms r11 r12 r13 r21 r22 r23 r31 r32 r33
R = [r11 r12 r13;
     r21 r22 r23;
     r31 r32 r33];

% Camera's postition in the world frame.
syms tx ty tz
Tr = [tx; ty; tz];

% Convert to camera frame by appending focal length and then
   rotate to world.
ned = R*[sensor; f];

% Convert this vector to a parameterized position vector.
syms S
pos_vec = S*ned + Tr;

% Solve for S so that the last row is zero (the item is on the
   flat world plane)
S = solve(pos_vec(3) == 0, S);

% Plug back into pos_vec to find actual actual world point.
NED = subs(pos_vec);

% Remove last component because it's zero.
NE_euclidean = NED(1:2);
```

```
%% Method 2 - Projective Space

% Convert from pixel to sensor frame in homogenous coordinates.
% sensor_p == sensor prime == sensor'
sensor_p = [0 -ph ph*v0; pw 0 pw*u0; 0 0 1]*[u; v; 1];

% Convert to camera frame
cam = [1/f 0 0; 0 1/f 0; 0 0 1]*sensor_p;

% 3x4 Transformation matrix
% From world to camera frame.
T = [transpose(R) -Tr];

% Modified transformation assuming D=0 so third column is removed.
Tm = [T(:,1:2) T(:,4)];

% Solve for world point.
world_p = inv(Tm)*cam;

% Convert back to non-homogenous coordinates by dividing by scale
   (S')
NE_projective = world_p(1:2) / world_p(3);

% Third row (R3) is cross product of first two.  Replace
   symbollically
% in 1st method since these terms won't show up in this method.
R3 = cross(R(1,:), R(2,:));
```

```
r31 = R3(1);
r32 = R3(2);
r33 = R3(3);
NE_euclidean = subs(NE_euclidean);

% This will print (1) (true) for each of the north and east
   symbollic equations if they are equivalent.
isAlways(NE_euclidean == NE_projective)
```

Matlab Output

```
ans =
   1
   1
```

Appendix B

Code Repositories

Post-Processing Pipeline

`https://github.com/Phenomics-KSU/HTMI`

Data Collection Program (DySense)

`https://github.com/DySense/DySense`

Extended Robot Functionality in ROS

`https://github.com/Phenomics-KSU/htp_auto`

Trimble GPS Driver in ROS (htp_auto branch)

`https://github.com/Phenomics-KSU/nmea_navsat_driver`