Development of a Quick-Install Auto-Steering System for Agricultural Vehicles

Antoine Pouliot

Department of Bioresource Engineering

Macdonald Campus of McGill University
Montréal, Québec
December 2016

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Master of Science

ABSTRACT

Navigation systems have become a crucial tool in large-scale agricultural operations. Such systems are available commercially; however, they are expensive and are not compatible with all models of tractors. The objective of this study was to develop an affordable quick-install camera system which was capable of acting as a guidance system capable of controlling the movement of a tractor for row crop cultivation and fertilization during the early stages of crop growth. Computer analysis of video streams is a technique which has only recently become an inexpensive method for obtaining guidance information. The technique has the particular advantage of utilizing the same equipment to sense a variety of different guiding features with minor user-level software changes. In this study, a computer-vision guidance system was developed to run on an Intel Atom D525 nanocomputer. A low-cost RGB-NIR camera was mounted to the front of the tractor, perpendicular to the ground and in line with the crop row to be followed to obtain a video stream of the plants passing underneath the vehicle. An application developed using the Python OpenCV platform was used to segment the crops from the soil and weeds, identify the lateral offset of the plant rows, and subsequently adjust the tractor's power-assisted steering accordingly via an adaptive PID controller. The adjustments were made by a stepper motor mounted on the front window of the tractor using suction cups. Unlike other similar systems, this mechanism that rests on the center of the steering wheel allows for quick and effortless installation and represents an inexpensive, entry-level precision agriculture solution. The computer vision system was tested successfully for travel speeds up to 11 km/h on tarmac under varying ambient light using a green garden hose to emulate a crop row.

ABRÉGÉ

Les systèmes de navigation sont devenus des outils cruciaux sur les grandes exploitations agricoles. Ces systèmes sont sur le marché, mais ils sont coûteux et ne sont compatibles qu'avec les tracteurs les plus récents. Par conséquent, l'objectif de cette étude était de développer un système de caméra abordable capable d'agir comme un système de guidage pour les cultures en rangs afin d'aider aux opérations de sarclage et d'application d'intrants durant les premiers stades de croissance des cultures. L'analyse par ordinateur des flux vidéo est une technique qui n'est que récemment devenue une abordable pour obtenir des informations de guidage. La technique a notamment pour avantage d'utiliser le même équipement pour détecter une variété de différents paramètres de guidage avec très peu d'implication de l'utilisateur, permettant ainsi à ce dernier de consacrer plus d'attention à d'autres tâches. Une caméra RVB (rouge, vert, bleu) à faible coût a été installée à l'avant d'un tracteur en ligne avec un rang de culture pour obtenir un flux vidéo des plantes passant sous l'équipement. La plate-forme Python OpenCV a été utilisée afin de développer une application pour distinguer les cultures du sol et des mauvaises herbes, de mesurer le décalage latéral du tracteur avec les rangées de plantes, puis d'ajuster la servodirection du tracteur en conséquence à l'aide d'une structure PID (proportionnelle, intégrale, dérivée) adaptative. Les ajustements étaient effectués par un moteur pas-à-pas installé au pare-brise du tracteur à l'aide de ventouses. Contrairement aux systèmes similaires, ce mécanisme qui repose sur le centre du volant du tracteur permet une installation rapide et facile, et représente une solution d'agriculture de précision d'entrée de gamme peu coûteuse. Le système de vision artificielle a été testé avec succès pour des vitesses de déplacement jusqu'à 11 km/h sur le tarmac en utilisant un boyau d'arrosage afin d'imiter un rang de culture.

ACKNOWLEDGMENTS

The author would like to acknowledge everyone who contributed at any level to the achievement of this thesis. Special thanks to Trevor Stanhope for his indispensable help with software development; thanks to Scott Manktelow for his friendly smile every morning, for the handcrafting lessons and his technical support in the construction of the various projects; and thanks to Paul Meldrum and the whole farm crew for their help with the field experiments. Thanks to Dr. Mark Lefsrud for his support and guidance as a committee member.

Partial funding for this research was provided through the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants Program.

I would like to thank my friends, and most importantly my family, for their unconditional support through my entire academic path. I am grateful to the research team colleagues that have made this journey rich and special, and I would especially like to thank my supervisor, Dr. Adamchuk, for his professional support. I am leaving behind good friends and colleagues, and taking unforgettable memories with me.

Je dédie ce document, mais surtout le travail et le cheminement qu'il représente,

à mes parents.

CONTRIBUTION OF AUTHORS

The author of this thesis was responsible for the invention of the quick-install auto-steering system and the development and evaluation of the optical guidance system. Trevor Stanhope, a master student under the supervision of Dr. Adamchuk, and the author are responsible for the elaboration of the entirety of the guidance software. The author designed and carried out the experimental and analytical work to meet the research objectives of this thesis. The author is also responsible for the preparation of the manuscript serving as a base for this thesis. Dr. Viacheslav Adamchuk, an Associate Professor in Department of Bioresource Engineering of McGill University, is the thesis supervisor. He created the idea for this research and offered scientific advice and technical guidance throughout the study. He is also responsible for editing and reviewing the prepared manuscript.

Dr. Adamchuk and Trevor Stanhope were coauthors on the Optical Guidance ASABE conference paper.

Publication related to the thesis

Pouliot, A., Stanhope, T.P., Adamchuk, V.I. A quick-install tractor guidance system relying on computer vision. 2015 New Orleans, Louisiana United States of America July 26-29, 2015.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND SYMBOLS

b	abscise of a linear function
BPPD	Band-Pass Plant Detection
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide-Semiconductor
CPR	Count Per Revolution
CS	Column Summation
df	degrees of freedom
e	lateral error or offset
FOV	Field of View
GLONASS	GLObalnaya NAvigatsionnaya Sputnikovaya Sistema
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HIS	Hue-Saturation-Intensity
IR	Infrared
LIDAR	a portmanteau of "light" and "radar"
NMEA	National Marine Electronics Association
PID	Proportional-Integral-Derivative
RGB	Red-Green-Blue
RMSE	Root Mean Square Average
RPM	Revolutions Per Minute
RTK	Real-Time Kinematic
S/N	Signal-to-Noise ratio
SURF	Speeded Up Robust Features

t	time
VRT	Variable Rate Technologies
θ	angular position
ω	angular velocity

1. INTRODUCTION

1.1 Overview

Steering agricultural machinery in the field to perform various farming operations is hard for machinery operators because of the long hours of tedious, repetitive tasks at low speeds. Among those tasks, guiding the tractor along crop rows without damaging plants requires much attention and is one of the leading causes of operators' fatigue. Becker et al. 1983 even reported that steering might be the most significant load on the mental capacities of an operator performing a field operation. With the adoption of attention-hungry equipment such as row crop cultivators (figure 1-1) or sprayers, it becomes necessary for producers wanting to increase their productivity to find alternatives to traditional vehicle guidance. Palmer & Matheson, 1988

Figure 1-1: Row crop cultivation on an organic operation.

determined that as much as 10% of crop production costs could be saved by improving the navigation accuracy of field equipment.

1.2 Justification

There is little doubt that big farming operations are embracing and massively adopting auto-guidance technologies. A survey by the Department of Agricultural Economics of Purdue University (Erickson & Widmar 2015) shows that GNSS guidance systems were adopted by 83% of the respondents in 2015 while only 61% had adopted them in 2013. The question here is whether smaller-scale, labour-intensive farms from developed and developing countries will ever adopt these technologies. For some, today's precision technology is still too expensive to be worth the cost (Dobbs 2013); however, as smaller, more modular equipment becomes widespread even the leanest operations may be able to change their practices.

When it comes to the automated guidance of agricultural vehicles, crop producers are faced with many options, such as global navigation satellite systems (GNSS) based guidance, or alternatives, including Light Detection and Ranging (LIDAR), ultrasonic, mechanical, and optical guidance systems. Among these, optical guidance has received significant attention as it is both relatively inexpensive and has an immense potential for further research and development, including different co-applications such as crop phenotyping.

Researchers have applied computer vision to guide agricultural vehicles for a wide diversity of field operations, with significant interest in cultivation, fertilization, spraying and harvest operations. In an optical guidance system, image-processing algorithms determine the position of the vehicle about the target (e.g. a crop row) and the vehicle trajectory is corrected by automatic steering control.

1.3 Research Objectives

The ultimate goal of this research was to equip farmers with a camera-based semi-automated steering system. The system has to be adaptable to different crops or tasks, be compatible with diverse agricultural vehicles equipped with power steering, be installed within five minutes without any pre-installation and lastly yet importantly, be inexpensive. The specific **objective** was to design and to test a commercially viable quick-install auto-steering system run by a simple control algorithm: a modified proportional-integral-differential (PID) control capable of guiding an unladen agricultural tractor to within a 95^{th} percentile lateral error within ± 5 cm of a desired path at speeds from 1 to 3 m/s on level tarmac or firm soil surfaces.

2. LITERATURE REVIEW

This section presents some of the literature related to guidance sensors for agricultural vehicles, automated steering systems and aids, and vehicle guidance algorithms to provide background information related to research presented in this thesis.

2.1 Posture sensors for agricultural vehicles

Inventions to help an agricultural vehicle driver space his/her planted rows uniformly started appearing in the 1870s (Heraud & Lange 2009) with patents regarding row markers, foam markers, aids for manual steering, automatic steering sensors, and auto steering systems. Posture (or positioning) sensors for agricultural vehicles with a few minor exceptions can be classified into four main categories: mechanical, ultrasonic, navigational, and optical.

Mechanical guidance systems are a contact type of guidance mechanisms that utilize existing features such as a crop row or a cultivation furrow. These systems are inexpensive as costs are restricted to the sensing and control devices. Snyder, 1885 invented a system called "furrow pilot" that automatically steered a tractor engine based on a previously formed furrow. The system and its successors were mostly used for tillage applications. Parish & Goering, 1970 developed a hydrostatic self-propelled vehicle with an automatic steering system based on a crop sensor. The latter consisted of two microswitches operated by the upward pressure of the hay. The system integrated an allowable dead band: if the edge of the standing crop lay outside the dead band, the switches sent a correction signal. Kirk, 1974 in his thesis described the development of a marker-follower guidance system for farm tractors that leaves a furrow-marker which is followed on the next pass. Field tests on a cultivated field showed the system to have good following characteristics on both straight and curved paths with maximum front wheel tracking error of

1 - 2" (2.54 – 5.08 cm). The system showed decent stability at speeds up to 6 miles/h on straight paths and was capable of following a 15-foot radius turn at 2.5 miles/h. This guidance system, with proper adaptation, could be used for most tillage and seeding operations. Examples of commercial products which have entered the industry in the past decade are the Claas Auto Pilot (CLAAS KGaA mbH, Harsewinkel, Germany) and the John Deere AutoTrac RowSense (Deere & Company, Moline, USA) (Figure 2-1). Mechanical systems are straightforward and affordable; however, they protrude from the vehicle and are a nuisance during headland operations and road transportation. Another considerable disadvantage lies in the fact that they require a physical feature that is strong enough to be sensed thus ruling out emerging row crops.

Ultrasonic devices calculate the distance from the time taken for an ultrasonic signal to reach and be reflected back from a target. Warner & Harries, 1972 have used them to detect the last

Figure 2-1: Claas Auto Pilot (left) (CLAAS KGaA mbH, 2016), and John Deere AutoTrac RowSense (right) (Deere & Company, 2016).

furrow for the next run during cultivation; however, an inadequate reflectance of ultrasound was obtained from soil, convincing the authors that it would be better to adopt optical methods for identifying the guideline to be followed. Mcmahon et al. 1987 used an array of ultrasonic sensors

to measure the position of apple trees to guide a harvester vehicle. Kirk & Krause, 1976 reported that ultrasonic sensors failed to obtain sufficient reflection from the crop edge to operate successfully and guide their swather. An infrared photoelectric sensor was found to be the best for their particular application.

The most recurrent approach to precise automated **navigational guidance** involves global navigation satellite systems (GNSS), and it was first proposed by Larsen et al. 1988. The American Global Positioning System (GPS) and its Russian equivalent GLONASS are worldwide radio-navigation systems that consist of constellations of 24 or more satellites and several ground stations. GNSS receivers have different levels of accuracy: inexpensive receivers ($100) such as the ones found in cellular devices can determine one's position to within 15 m (Heraud & Lange 2009). More expensive receivers that can use a second signal (L1/L2 bands) have a 1-m accuracy using differential GNSS. The most accurate GNSS receivers using a technique named real-time kinematic (RTK) can position within a few centimeters and are worth tens of thousands of dollars. The use of the latter was pioneered by O'Connor et al. 1996 for automatic steering of a tractor along straight lines. The accuracy of the RTK GNSS method is better than a 2.5 cm standard deviation 95% of the time(Easterly et al. 2010). Several patents were subsequently issued, leading to a fast adoption of navigational guidance that has had a significant effect on agricultural practices. The main drawback to GNSS guidance systems other than their prohibitive pricing is their "blindness" concerning existing features such as crop rows, residue rows, windrows or swaths, and furrows which could lead to errors in guidance unless the entirety of previous field operations were conducted using GNSS guidance.

Studies on **optical guidance systems** date as far back as the 1980s. Reid & Searcy, 1987 reported on the development of techniques that provide a well-segmented image of crop and soil background. They used near-infrared filters and an auto-iris lens, which kept the lighting level constant, to allow accurate segmentation. However, their work did not extend to vehicle guidance signals. Marchant & Brivot, 1995 investigated the problem of deriving vehicle guidance information (offset and heading angle) from images of crops grown in rows. They used the Hough transform to calculate the offset and orientation about the row structures. They combined information coming from three row segments to increase performance. Typical errors were 12.5 mm of offset and 1.0 degree of angle at a forward speed of 2 m/s. Billingsley & Schoenfisch 1995 used the chrominance signal (the color information of a picture) from a color camera to acquire an image representing the "greenness" in the image. To find line structures they used a cost function to find the best fit lines on several rows. A geared DC motor was used to steer a small tractor. They evaluated the performance of their system based on response to an initial error of 0.5 m at different speeds. Gerrish et al. 1997 installed a computer-vision guidance system on a full-size, modern tractor. A control point in the image was used as a reference to determine the angle at which the front steering wheels should be turned. They compared the performance of the system with steering by experienced human operators. In a test of skill, the human operators were more precise than automatic guidance at 12.9 km/h. At 4.8 km/h, however, automatic guidance was as skillful as the humans. Slaughter et al. 1999 demonstrated that color segmentation and an algorithm based upon the median of the spatial distribution of the seed line could be used with off-the-shelf computer-vision hardware to develop a real-time guidance system for row crop cultural practices such as cultivation. The precision of the system was comparable to that of a manually-guided cultivator and was demonstrated at a ground speed of up to 16 km/h under low weed loads.

Okamoto et al. 2002 developed an automatic guidance system for a weeding cultivator. A color charged-couple device (CCD) camera acquired the row crop images, and by processing the images in the computer, determined the offset between the machine and the target crop row. The weeding machine was steered through the crop row using an electro-hydraulic steering controller. Han et al. 2004 developed a row segmentation algorithm based on k-means clustering to segment crop rows. This information was next used to steer a tractor. The guided tractor was able to perform field cultivation in both straight and curved rows. However, quantitative performance evaluation of the automatically guided tractor had not been completed.

Another research was to develop a row detection algorithm for a stereovision-based agricultural machinery guidance system. The algorithm consisted of functions for stereo-image processing, elevation map creation and navigation point determination. The method developed first reconstructed a three-dimensional crop elevation map from a stereovision image of crop rows and searched for optimal navigation points from the map (Kise et al. 2005). Matias & Gil 2007 looked for a solution to guide an agricultural vehicle with independence of the task. Their system was based on a segmentation algorithm that used an optimum threshold function in terms of minimum quadratic value over a Fisher linear discriminant. This system has achieved positive results in the sense of segmentation in addition to guiding a vehicle in a real world environment successfully. However, conditions of non-uniform illumination raised the rate of errors; therefore, the system became confused because of the irregularities shadows of the terrain or external factors. It was also observed that the effect of irregular vegetation made the system inefficient when the difference of texture between the processed and non-processed areas was insignificant, as in tilling.

The common denominator among all the previous studies is the use of a fixed forward field of view ("far FOV") camera arrangement. The latter works adequately in the case of tall, mature

plants; however, it has its limitations with small plants and is inadequate when used for turning at the end of the row. For this purpose, Xue et al. 2012 implemented two FOV modes to complement the far FOV: the "near FOV" for small plants, and the "lateral FOV" used in turning a robot. A fuzzy logic control scheme was used to guide the robot. The method was tested while the vehicle successfully traveled through a distance of 30 m towards the end of a crop row in three replications. RTK-GNSS data was collected to evaluate de guidance performance and showed a maximum guidance error of 15.8 mm and stable navigational behavior.

Stanhope et al. 2014 developed an inexpensive webcam-based system which is capable of supplementing the mechanical guidance system for row crop cultivation during the early stages of crop growth. A computer-vision cultivator guidance system was developed for a 700 MHz ARM minicomputer to control a mechanical guidance system. The Python OpenCV platform was used to develop an application to identify the lateral offset of the plant rows and to adjust the hydraulic steering accordingly. The guidance system performed sufficiently for travel speeds up to 6 km/h in grain corn, green bean, and soybean fields under varying ambient light and crop conditions. Examples of commercial products which have entered the industry in the past years

are the Claas Cam Pilot (CLAAS KGaA mbH, Harsewinkel, Germany) and the John Deere AutoTrac Vision (Deere & Company, Moline, USA) (Figure 2-2).

Figure 2-2: Claas Cam Pilot (left) (CLAAS KGaA mbH 2016) and John Deere AutoTrac Vision (right) (Deere & Company 2016).

2.2 Automated steering systems and aids

Automated or aided guidance of agricultural vehicles can take various forms, including manual guidance with a lightbar, automatic steering systems that tap into the vehicle's steering electro-hydraulic valve or that use electric motor drives, depending on the investment. An example of a commercial lightbar used for guidance is the John Deere GreenStar Lightbar System (Deere & Company, Moline, USA) (Figure 2-3) which represents a straightforward and cost-effective guidance solution that comprises a GNSS receiver, a user interface, and path planning algorithms. It can readily be transferred between machines; however, it does require the operator to keep his/her eyes on the lightbar and hands on the steering wheel at all times. Typical accuracy for manual guidance ranges from 5 or 10 cm to 30 cm pass-to-pass, depending on the driver's coordination (Heraud & Lange 2009). Automatic steering, or auto-steer, is a much more complex system that typically requires a GNSS receiver, a user interface, path planning algorithms, vehicle steering actuators and manual override detectors, steering angle sensors, control

algorithms and controller, terrain compensation, and other sensors. Although automatic steering offers many benefits such as increased accuracy, increased operating speed, ability to concentrate on other tasks and reduced fatigue, it is nonetheless significantly more complex and costly than the lightbar system and is difficult to transfer between machines as it requires the latter to be equipped with electro-hydraulic steering valves. Examples of successful commercial auto-steer systems are the Trimble Autopilot (Trimble Navigation, Ltd., Sunnyvale, USA) and the John Deere AutoTrac (Deere & Company, Moline, USA). Electric motor drives are most commonly used in universal steering kits, typically for tractors that are not equipped with electro-hydraulic steering valves. In these systems, an electric motor actuator is used to move the steering wheel to steer the vehicle. Installation time is shorter due to the less intrusive nature of the systems, and they can therefore easily be transferred from one vehicle to another. Examples of successful commercial electric motor drive systems are the Trimble EZ-Steer (Trimble Navigation, Ltd., Sunnyvale, USA) (Figure 2-3), which consists of a friction wheel on the peripheral area of the steering wheel, and the John Deere AutoTrac Universal 200 (ATU) (Deere & Company, Moline, USA) (Figure 2-3), which consists of a simple belt mechanism.

Figure 2-3: John Deere Lightbar (left) (Deere & Company 2016), Trimble EZ-Steer (middle)(Trimble Navigation 2016), and John Deere ATU (right) (Deere & Company 2016).

2.2 Vehicle control algorithms

Control algorithms use the information relayed from the path-planning algorithms (lateral offset, heading error) to compute steering adjustments that must be accomplished by the vehicle steering system (Heraud & Lange 2009). Many control algorithms have been developed over the past decades, and some common methods are discussed here.

The PID algorithm (proportional-integral-differential) is a **tuning-based** control algorithm that has yielded practical results in previous works (Subramanian et al. 2006, Benson et al. 2003) and that assumes that the tractor's kinematics can be approximated as a first or second-order differential equation. In Benson et al. 2003, PID was used to calculate the actuator command signal based on the heading offset. The performance of the controller was comparable to that of manual steering. However, the maximum speed at which the controller could operate was 1.3 m/s. Fuzzy logic controllers ("fuzzy" refers to the fact that the logic involved can deal with concepts that can be expressed as "partially true" as opposed to "true" or "false") have also been used to control steering of agricultural vehicles (Cho & Ki 1999), and sometimes in conjunction with PID (Kodagoda et al. 2002). In Cho & Ki 1999, a fuzzy logic controller and computer-vision were used for guiding an autonomous sprayer vehicle through orchards. The input information to the fuzzy logic controller was given by both computer-vision and ultrasonic sensors. A combination of Fuzzy logic and PID control worked well for guiding a tractor through crop rows and repeatedly outperformed conventional PID schemes (Kodagoda et al. 2002).

Another approach to agricultural vehicle guidance algorithms is to use **model-based** control algorithms. The latter consists of developing a mathematical model of the vehicle's dynamics and using a feedback controller to control the mathematical model. To ensure these models are

independent of the make, type and size of the vehicle, a simple two-degree-of-freedom (DF) "bicycle model" approach is widely used throughout literature (Feng et al. 2004; Alleyne & DePoorter 1997; Derrick & Bevly 2009; Lenain et al. 2006; Gomez-Gil et al. 2011; Stombaugh et al. 1999; Choi et al. 1990; O'Connor et al. 1996). This model has one input (steering angle) and one output (lateral position) and requires the guidance dynamics of the vehicle to be quantified beforehand. Stombaugh et al. 1999 successfully tested a classical model-based controller that provided guidance of a two-wheel-drive agricultural tractor to within 16 cm of the desired path at speeds up to 6.8 m/s after experimentally quantifying the guidance dynamics of the tractor.

3. MATERIALS AND METHODS

3.1. Design of system components

3.1.1 Preliminary design

An outdoor USB CMOS camera (dust-tight, water-tight) with 640x480 pixel resolution (Shenzhen Yufei Technology Co., Ltd, Shenzhen, China) was mounted to the front of a 95 hp New Holland T5050 (New Holland Agriculture, Turin, Italy) tractor in-line with a crop row (381 mm to the left of the centerline of the tractor, and 730 mm to the front of the steering axle) to obtain a video stream of the plants under the tractor (Figure 3-1). It is to note that the system is calibrated by physically locking the camera above the area where the feature to be followed would normally pass. The camera had a lens with a 6-mm focal point, which translates to a 43°

Figure 3-1: Camera mounted to the bumper bar of the experimental tractor.

horizontal field of view and 32.3° vertical field of view at the standard 1.33 aspect ratio. This field of view provided approximately 1.15 mm/pixel resolution when mounted 1.0 m above the ground and oriented orthogonally to the direction of travel. A checkered board with 67 mm x 67 mm squares was used to validate the pixel to mm conversion (Figure 3-7).

The sensor was capable of automatically adjusting exposure and aperture settings for varying lighting conditions and could process images at a rate of approximately 25-30 frames/s. Although the software was optimized for the Intel Atom D525 processor (Intel Corporation, Santa Clara, USA) in a potentially marketable iteration, it was actually run on an Intel i7 powered VMC3501-K (NEXCOM International Co., Ltd, Taipei, Taiwan) with integrated touchscreen displaying a graphical interface to the tractor operator to ease the process of altering the code during tuning runs. Steering control was achieved with a Phidgets 1067 bipolar stepper controller (Figure 3-2) (Phidgets Inc., Calgary, Canada) and a 12V, 2.8A, 46.6 kg-cm geared bipolar stepper motor (Dongzheng Motor Co., Ltd, Dongyang City, China) equipped with an external optical rotary encoder that produced 300 cycles per revolution. An Arduino UNO microcontroller board (Figure 3-2) (Arduino, Somerville, USA) based on the Atmel ATmega328P microcontroller (Atmel, San Jose, USA) was used to read the signal from the rotary encoder. The stepper motor apparatus was fixed onto the tractor steering wheel via a RAM ball-grip positioning arm (National Products Inc., Seattle, USA) made of marine grade aluminum and capable of holding a mass of 4.5 kg. The positioning arm was mounted to the front window of the tractor using lock-to-grip suction cups capable of holding up to 60 kg (Figure 3-3). A steering wheel hub adapter was designed and fabricated using additive manufacturing. The installation of

the entire system took well under five minutes and could easily be done without tools in a commercially viable iteration.

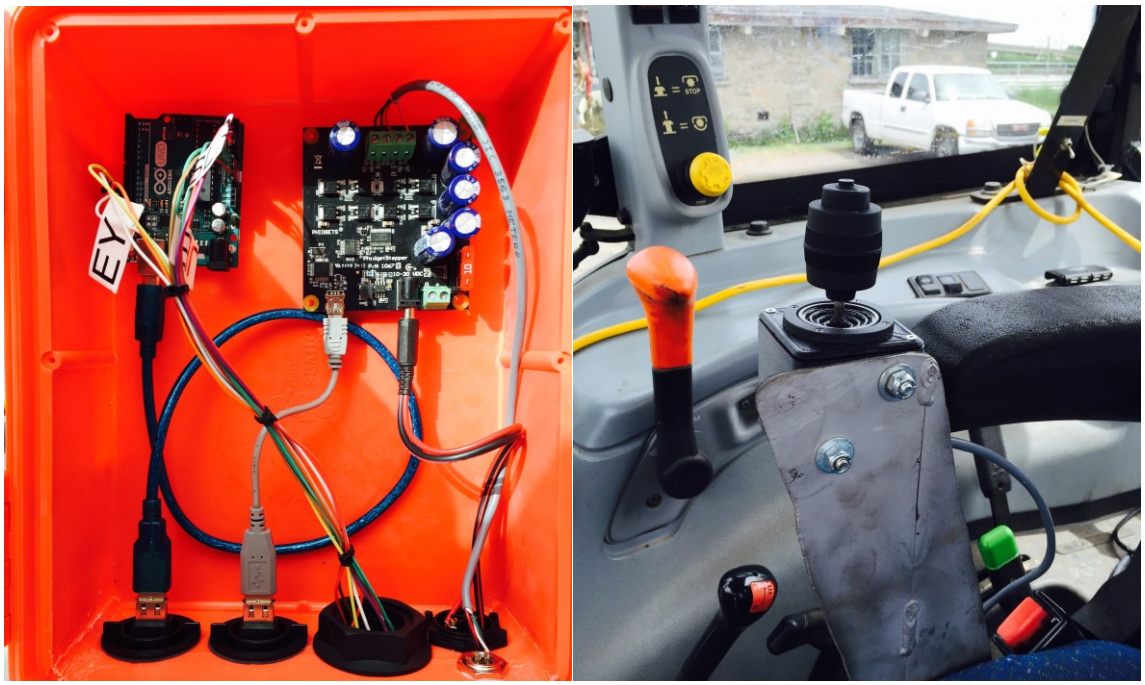

Figure 3-2: Stepper motor controller and microcontroller (left), and joystick (right).

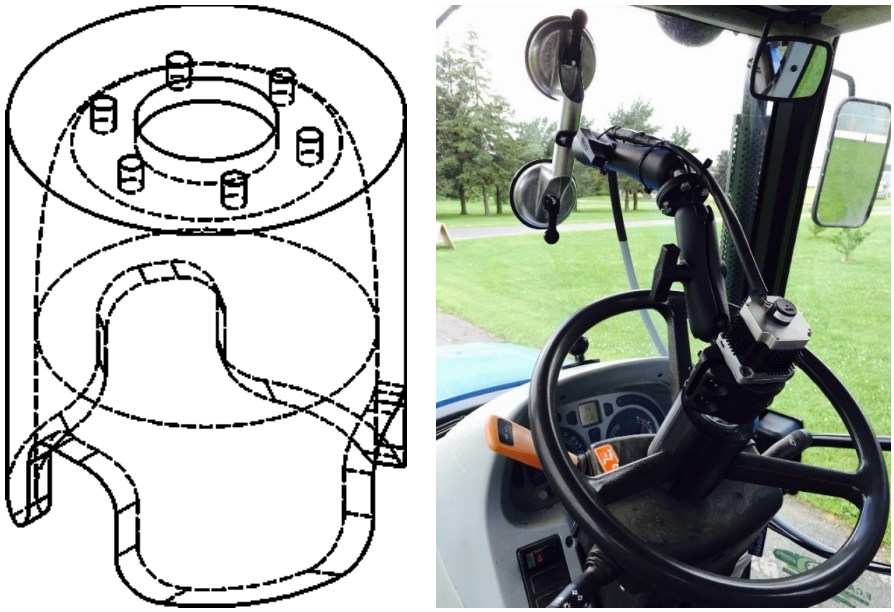

Figure 3-3: Drawing of the wheel hub adapter (left) and the quick-install steering mechanism (right).

The tailored ABS steering hub adapter offered many advantages such as light weight (less than 200 g), robustness, low cost, and adaptability to any steering wheel (Figure 3-4). A momentary

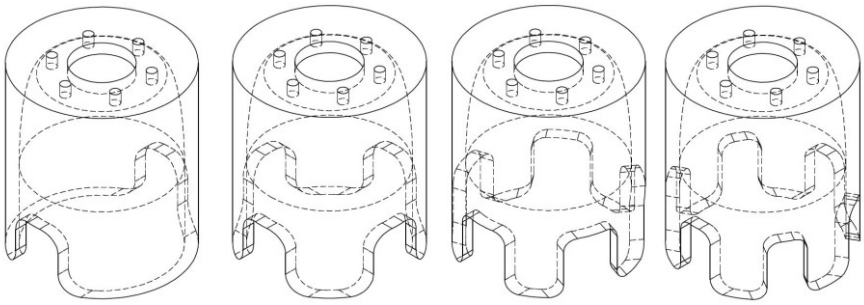

Figure 3-4: Additional steering hub configurations.

pushbutton switch mounted on a Tenco JH-D400X-R4 joystick (Tenco Technology Company Ltd., Shenzhen, China) allowed to switch easily between the automatic and manual modes of the steering system and offered the operator the ability to manually control the gain of the steering algorithm if needed. The ergonomic joystick, which relied on a 10 kΩ potentiometer, was used to steer the tractor manually when performing headland operations (Figure 3-2). The system was inexpensive compared with other guidance solutions already on the market. The price of the

guidance system as depicted in Figure 3-5 -omitting the GNSS receiver and the wheel angle sensor- was just under $1,100USD (Intel Atom CPU setup) (details in APPENDIX C).

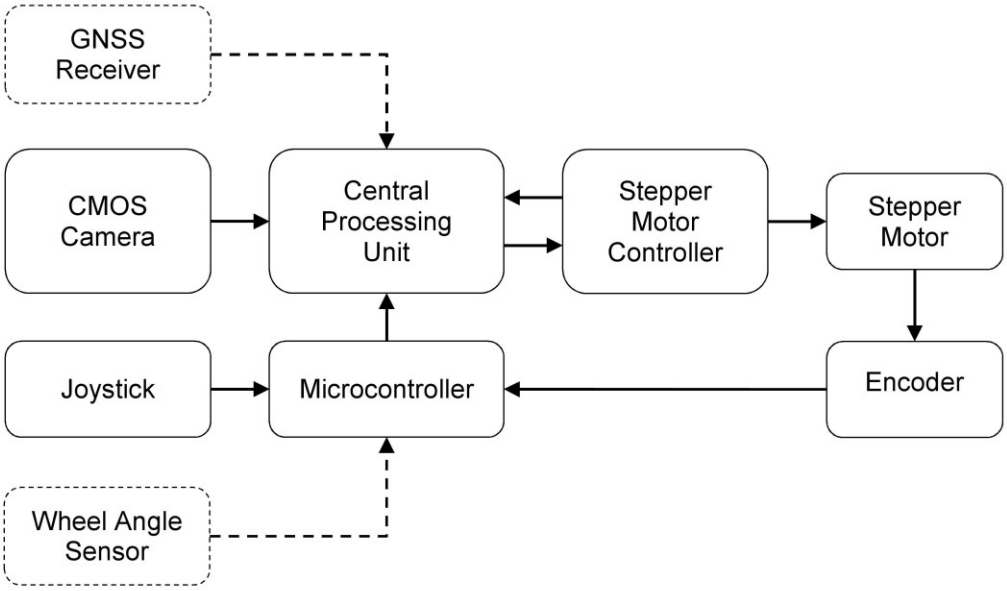

Figure 3-5: Steering system diagram

3.1.1.1 Instrumentation

Relative position measurements performed with the vision sensor were synchronized with geographic locations to validate measurements obtained during different passes. An RTK-level GNSS receiver located above the camera (1.52 m above the ground) was used to acquire geographic longitude and latitude (Figure 3-6), time, and GNSS signal quality parameters to serve as a comparison for the guidance system performance. The RTK-level GNSS receiver for both the rover and the base stations was a Trimble AgGPS 542 (Trimble Navigation, Ltd.,

Figure 3-6: RTK antenna mounted above the camera and king pin angle potentiometer.

Sunnyvale, USA), and the base station was located less than 500 m from the test track.

During development stage, a rotary potentiometer was used as a wheel angle sensor to validate the linear relationship assumption between steering wheel angle and actual wheel king pin angle (Figure 3-6). Figure 3-7 illustrates the relationship between steering wheel encoder value and king pin potentiometer value.

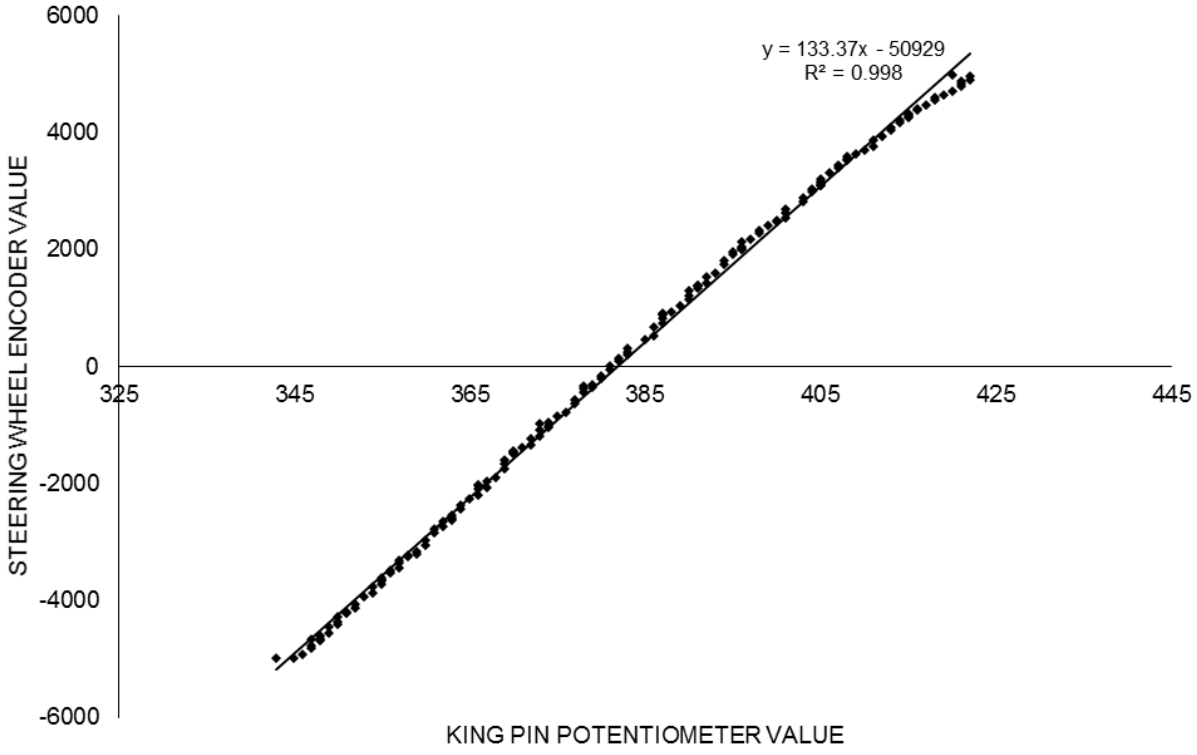

Figure 3-7: Steering wheel encoder value versus kingpin potentiometer value.

3.1.2 Final design

The outdoor USB CMOS camera (dust-tight, water-tight) with 640x480 pixel resolution (Shenzhen Yufei Technology Co., Ltd, Shenzhen, China) used for the Stage 1 experiment during the summer of 2015 was replaced by a Logitech HD Webcam C270 set to a 640x480 pixel resolution (Figure 3-8) (Logitech International S.A., Lausanne, Switzerland) to address issues

Figure 3-8: Logitech camera mounted to the bumper bar of the experimental tractor for Stage 2 experiment.

with degrading image quality over time. The new camera had a lens with a 4-mm focal point, which translates to a 60° field of view. This field of view provided approximately 1.38 mm/pixel resolution when mounted 1.0 m above the ground and oriented orthogonally to the soil surface. A checkered board with 67 mm x 67 mm squares was used to validate the pixel to mm conversion. (Figure 3-9). Once again, the sensor was capable of automatically adjusting exposure and

aperture settings for varying lighting conditions and could process images at a rate of approximately 25-30 frames/s.

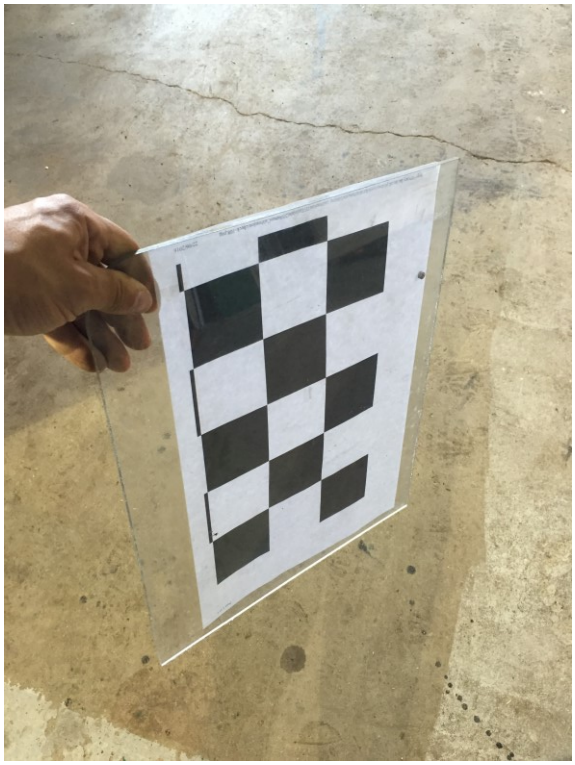

Figure 3-9: Checkered board used for pixel to mm conversion.

3.3. Guidance algorithm development

3.3.1 Crop row tracking

To detect the lateral offset of the crop row, a Python (Python Software Foundation, Delaware, USA) application based upon the research by Stanhope et al., 2014 was developed using the OpenCV image processing library. The plants were segmented from inorganic matter and crop residue in each image by converting from the RGB color space to the Hue-Saturation-Intensity (HSI) color space (OpenCV, 2014) to simplify color analysis and reduce the complexity of applying band-pass image filters. By converting to a de-correlated color-space, the image's hue, intensity, and color saturation were calculated according to:

$$H_{ij} = \begin{cases} 60 \cdot \left(\frac{G_{ij}-B_{ij}}{I_{ij}-min(R_{ij},G_{ij},B_{ij})}\right) & if\ I_{ij} = R_{ij} \\ 120 + 60 \cdot \left(\frac{B_{ij}-R_{ij}}{I_{ij}-min(R_{ij},G_{ij},B_{ij})}\right) & if\ I_{ij} = G_{ij} \\ 240 + 60 \cdot \left(\frac{R_{ij}-G_{ij}}{I_{ij}-min(R_{ij},G_{ij},B_{ij})}\right) & if\ I_{ij} = B_{ij} \end{cases} \quad (1)$$

$$S_{ij} = \begin{cases} 0 & if\ I_{ij} = 0 \\ \frac{I_{ij}-min(R_{ij},G_{ij},B_{ij})}{I_{ij}} & if\ I_{ij} \neq 0 \end{cases} \quad (2)$$

$$I_{ij} = \max(R_{ij}, G_{ij}, B_{ij}) \quad (3)$$

where I_{ij} = intensity, H_{ij} = hue, and S_{ij} = saturation parameters; R_{ij} = red, G_{ij} = green, and B_{ij} = blue value for a pixel in the i^{th} column and j^{th} row.

Based on these decolorized parameters, a Band Pass Plant Detection (BPPD) method was incorporated to segment plant foliage (pixels with hue from yellow-green to blue-green):

$$BPPD_{ij} = \begin{cases} 1 & if\ H_{ij} > H_{min} \wedge\ H_{ij} < H_{max} \wedge S_{ij} > mean(S) \wedge I_{ij} > mean(I) \\ 0 & otherwise \end{cases} \quad (4)$$

Where H_{min} = minimum hue (30, corresponding to yellow-green color); H_{max} = maximum hue (120, corresponding to blue-green color).

Once converted to the BPPD matrix, column summation (CS) in the direction of travel was calculated using:

$$CS_i = \sum_{j=0}^{n_j} BPPD_{ij} \quad (5)$$

where CS_i = i^{th} column (vertical pixel array) summation; n_j = the number of utilized rows (horizontal pixel arrays).

This summation results in a 1-D array representing the lateral distribution of BPPD values within the image. The lateral offset of the crop was determined by applying a high-pass filter to the CS array according to:

$$CI_i = \begin{cases} i & if\ CS_i \geq mean(CS) + 2 \cdot std(CS) \\ N/A & if\ CS_i < mean(CS) + 2 \cdot std(CS) \end{cases} \quad (6)$$

where CI_i = column index vector filled with column numbers i indicating columns with relatively high values of CS_i.

Lastly, the offset (in pixels) from the center of the camera's field of view was calculated using the median of valid column numbers according to:

$$Offset = \begin{cases} median(CI) - \frac{n}{2} & if\ count(CI) > 0 \\ i_{max(CS)} - \frac{n_i}{2} & if\ count(CI) = 0 \end{cases} \quad (7)$$

where Offset = the number of pixels shift of the crop row with respect to the center of the camera; n_i = the number of columns (image in pixels) (Figure 3-10).

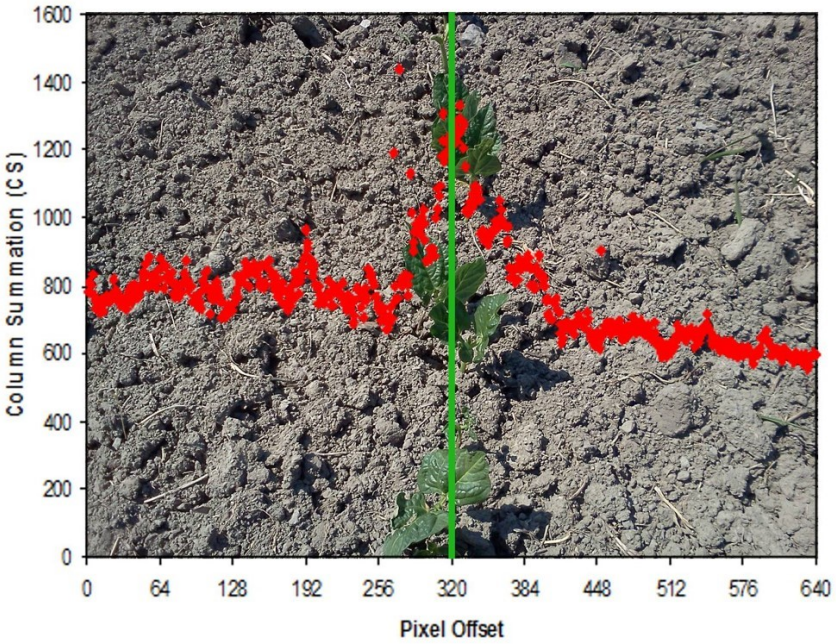

Figure 3-10: Row detection demonstration. Reprinted with permission by Stanhope (2014).

3.3.2 Ground speed estimation

To adjust the steering dynamics as a function of ground speed, a prototype computer-vision ground speed detection function was developed based on current work by Stanhope et al. The function was incorporated within the guidance software framework and runs in parallel to the row segmentation function. This approach allows the single camera to serve a dual purpose and removes the requirement of a GNSS receiver, radar device, or "fifth wheel." The ground speed algorithm uses two consecutive frames of the video stream to identify key points using the SURF algorithm (Speeded Up Robust Features) (Bay et al. 2006) (Figure 3-11). Next, k-means nearest neighbor matching on the 128-dimensional descriptors of each key-point finds matching key points between the two images (Figure 3-12). Lastly, the average velocity of the matching key

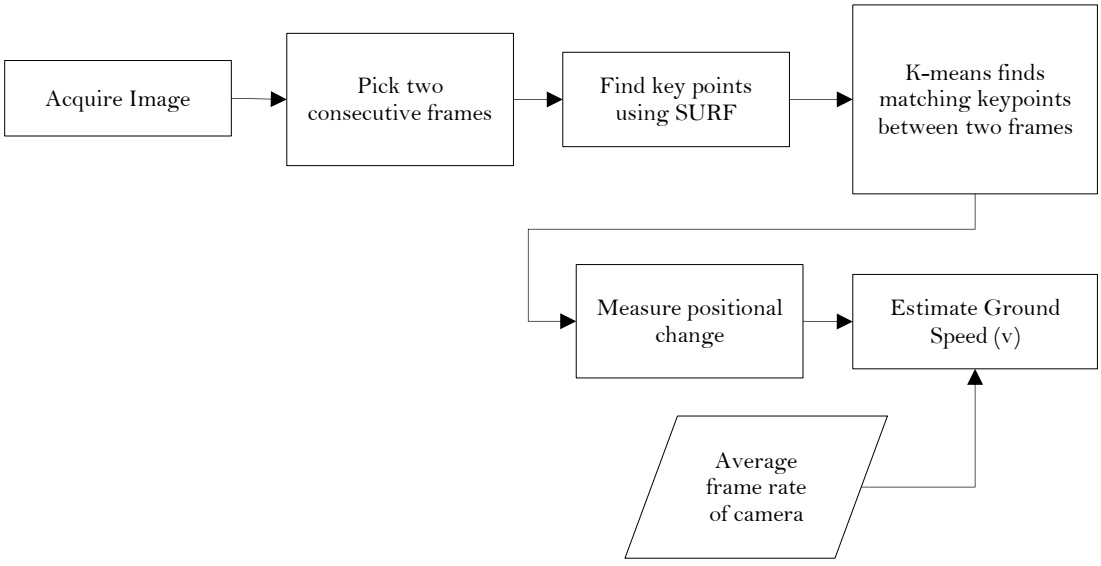

Figure 3-11: Flowchart of the ground speed estimation.

points was calculated by determining the positional change in projected (Okamoto et al. 2002) multiplied by the average frame-rate of the camera (Stanhope, 2015).

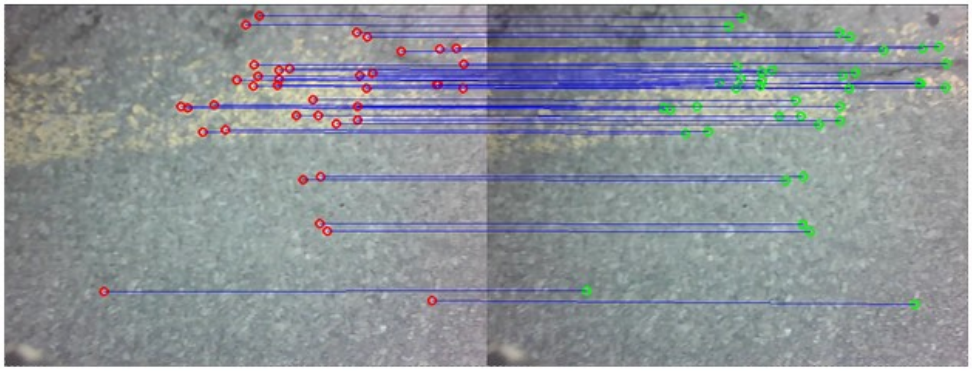

Figure 3-12: Keypoints matching on consecutive frames for ground speed estimation. Reprinted with permission by Stanhope (2014).

3.3.3 Preliminary control algorithm

For actuation of the steering wheel, an adaptive proportional-integral-derivative (PID) control system was designed. For this style of control, a suitable choice of proportional-integral-derivative (PID) gains and relationship between the steering amplitude and groundspeed was necessary in order to produce acceptable behavior of the steering system. The PID gains were established to provide similar steering behavior to that of a human operator by following a simple iterative testing protocol for travel speeds from 0.5 m/s to 2.5 m/s.

Firstly, a simple proportional control style was tested. Next, the groundspeed adaptive component was implemented with the gain K being linear and negatively proportional to the ground speed. Once basic functionality was achieved, derivative control based on a projected error was implemented to prevent overshoots (Figure 3-13). Lastly, a relatively small integral component was included to reduce residual steady-state lateral error (Figure 3-14). The desired steering wheel angle was obtained from:

$$\theta_w = \left(K_0 - K_v * v(t)\right) * \left(K_P \varepsilon(t) + K_I T_s \frac{\sum_{t-T_I}^{t} \varepsilon(i)}{T_I} + K_D T_s \frac{\sum_{t-T_D}^{t} \varepsilon(i) - \varepsilon(i-1)}{T_D}\right) \quad (8)$$

assuming: $\theta_a \propto \theta_w$

where θ_w= steering wheel angle, θ_a = actual wheel angle, K_0 = minimum gain (0.3), K_v = slope of gain (0.15); v = ground speed, K_P = proportional gain (2.5), K_I = integral gain (0), K_D = derivative gain (8.5), t = time or instantaneous time, e = lateral error or offset, T_S = time between samples, T_I = number of seconds used for averaging in the integral, T_D = number of seconds used for averaging in the derivative.

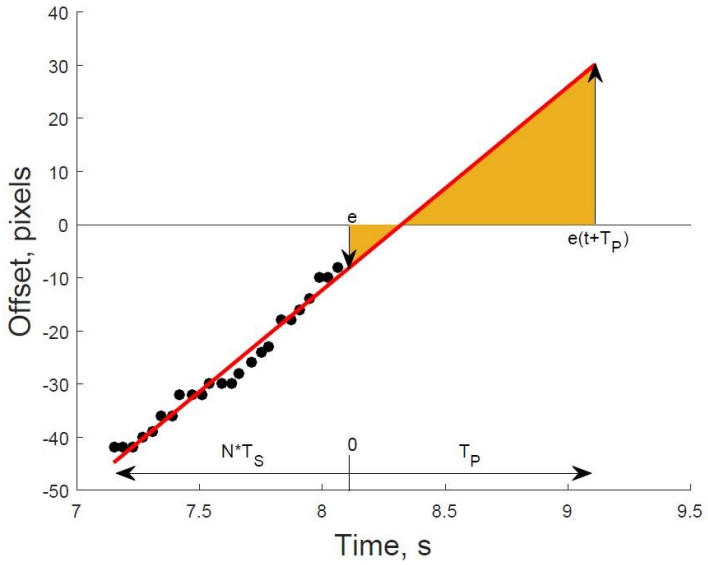

Figure 3-13: Stage 1 algorithm demonstration.

The stepper motor velocity was set to 8000 steps/s; the acceleration was set to 22000 steps/s^2 and the number of averages, N, was set to 5. The output min and output max were set to 0 and 20000 respectively.

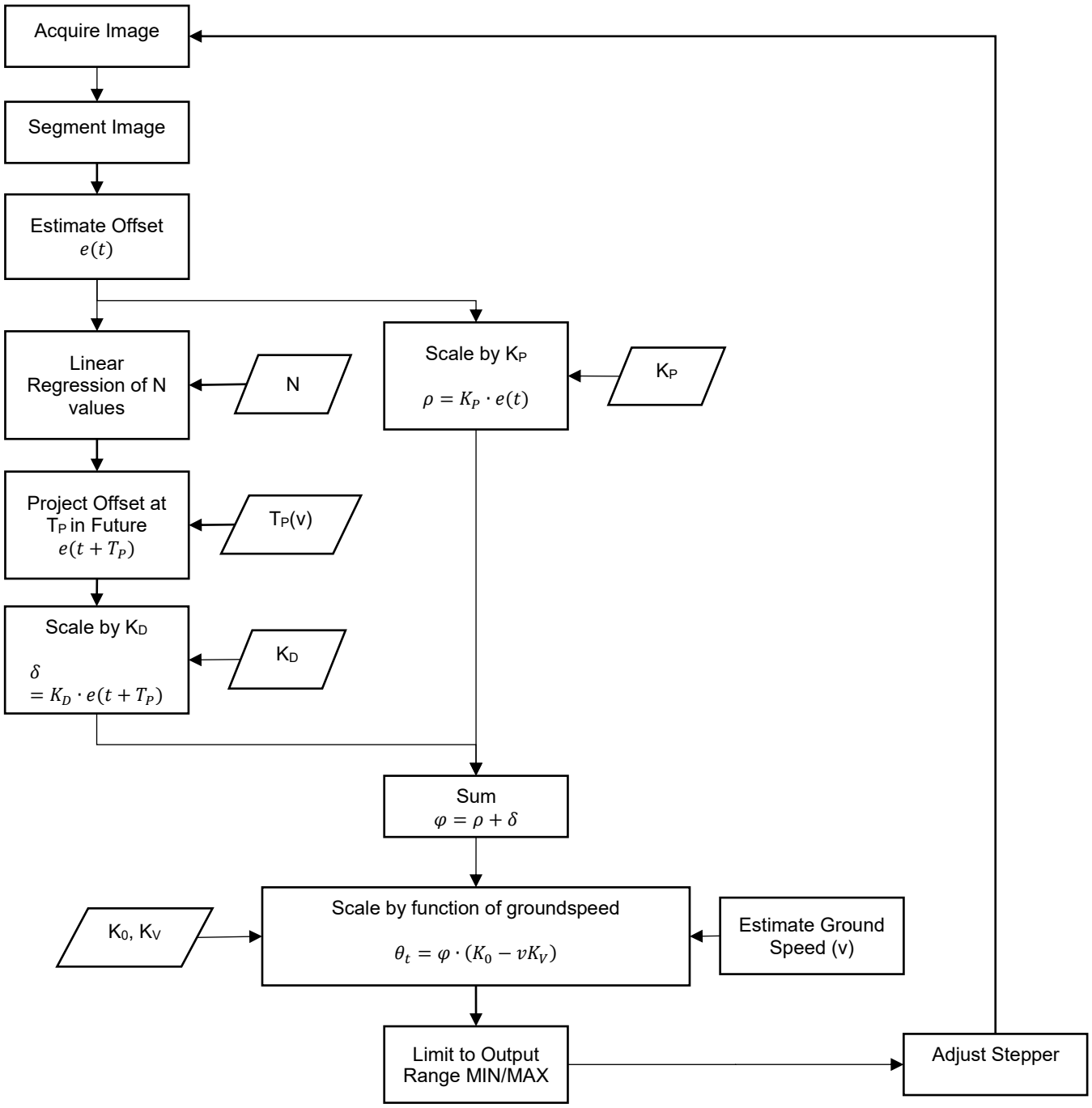

Figure 3-14: Flowchart of Stage 1 PID control algorithm.

3.3.4 Final control algorithm

Analysis of the results from Stage 1 experiment showed that the estimated lateral error data was noisy due to vibration effects and possibly required filtering to increase the signal-to-noise ratio and prevent jumpy behavior of the stepper motor. A second-order Savitzky-Golay filter was hence implemented in the program and applied to the lateral error estimation data points before the regression was conducted (Figure 3-15). The second-order Savitzky-Golay filter can only

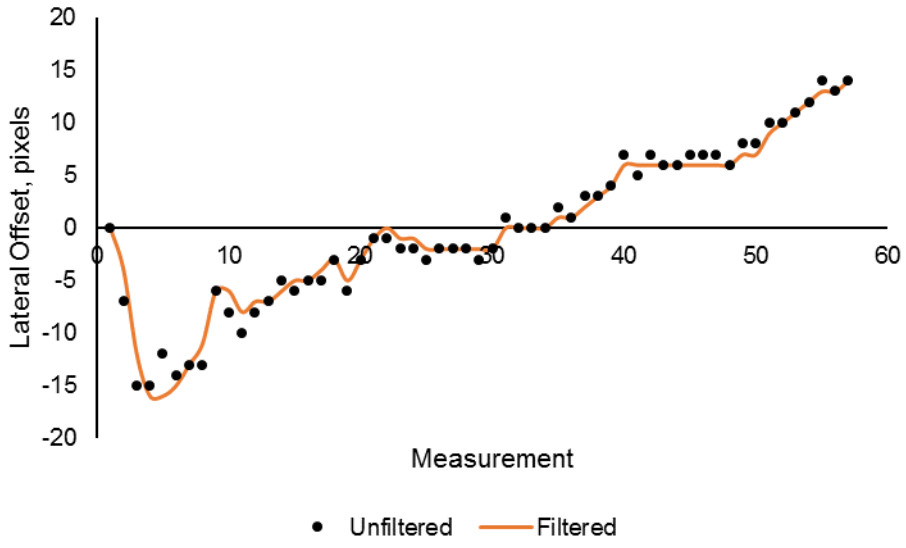

Figure 3-15: Illustration of the Savitzky-Golay filter.

take odd numbers of measurements; therefore, nine frames and 21 frames were chosen instead of 10 frames and 20 frames for assessing the effect of time projection length on the performance of the guidance system (in a commercial iteration, the control of time projection length would be available to the operator). Additionally, the proportional component of the PD control algorithm was dropped because deemed unnecessary to achieve stability and two dissimilar differential coefficients took turns whether the tractor was drifting away from its target or closing in towards it (Figure 3-16). The degree of the polynomial regression of preceding lateral error measurements was increased from 1 to 2 as it more accurately defines the motion of a vehicle.

Lastly, groundspeed estimation and scaling were taken out to fully assess the effect of projection time length. The two differential gains K_1 and K_2 were established by following a simple iterative testing protocol for travel speeds from 1 m/s to 3 m/s. A new approach to controlling the

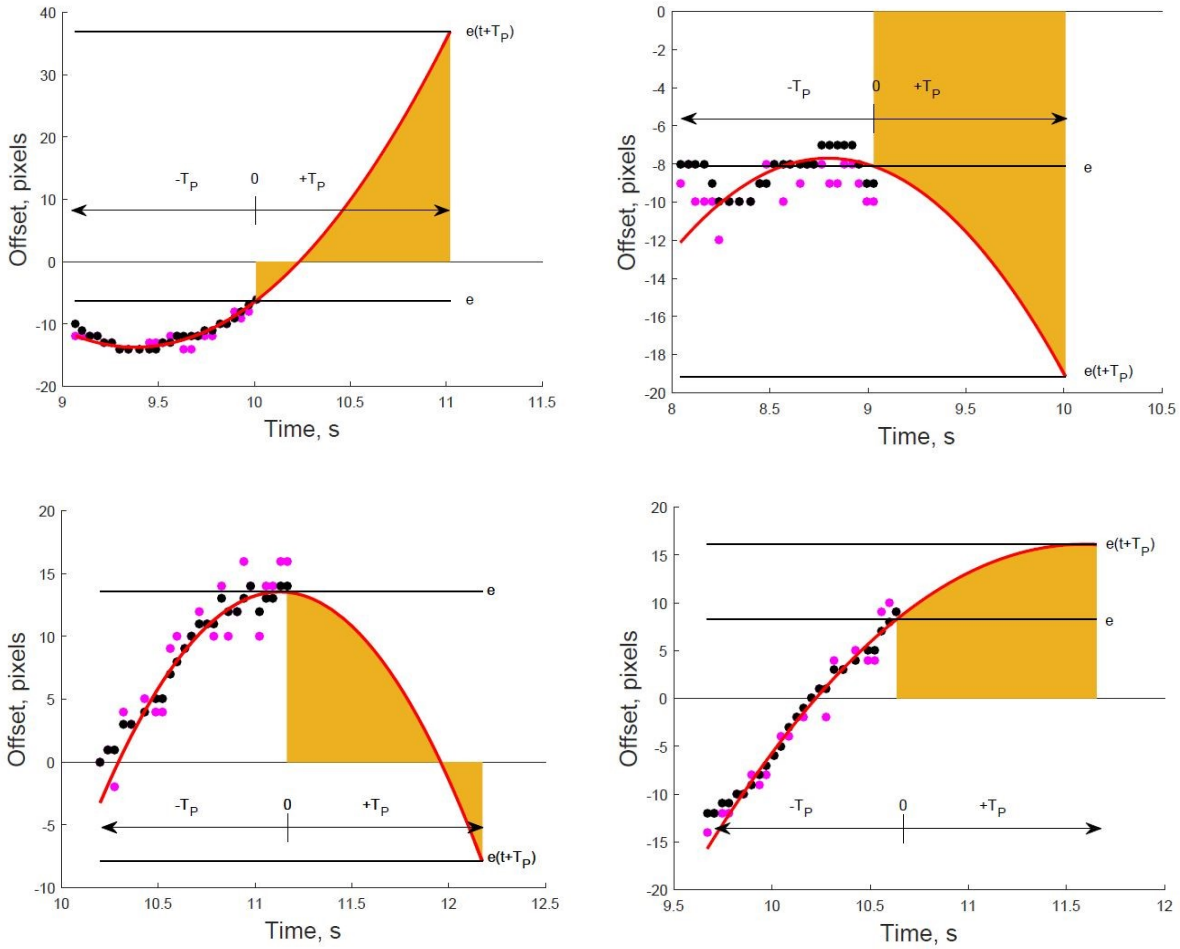

Figure 3-16: Projection algorithm illustrations. Top left: approaching from the right-hand side. Bottom left: approaching from the left-hand side. Top right: drifting towards the right-hand side. Bottom right: drifting towards the left-hand side. Magenta dots represent unfiltered data.

steering wheel was adopted for Stage 2: the controller now receives a target rotational velocity as a command rather than a positional change as this method was considered arbitrary because it had a different effect on the steering depending on the previous wheel position (Figure 3-17).

The desired steering wheel rotational velocity was obtained from:

$$\text{if } |P>I|: K_D = K_1$$

$$\text{else: } K_D = K_2$$

$$\text{if } P = 0: \omega_w = 0$$

$$\omega_w = K_D * (P - I) \tag{9}$$

$$\text{assuming: } \omega_a \propto \omega_w$$

where ω_w= steering wheel rotational velocity, ω_a = actual wheel rotational velocity, P = current smoothed offset, I = integral of projected values, K_D = derivative gain, K_1 = derivative approach coefficient (80), K_2 = derivative drift coefficient (160), t = time or instantaneous time, e = lateral error or offset, T_S = time between samples, T_D = number of steps used for averaging in the derivative (9 or 21 steps).

This time the stepper motor maximum velocity was set to 3000 steps/s and the acceleration to

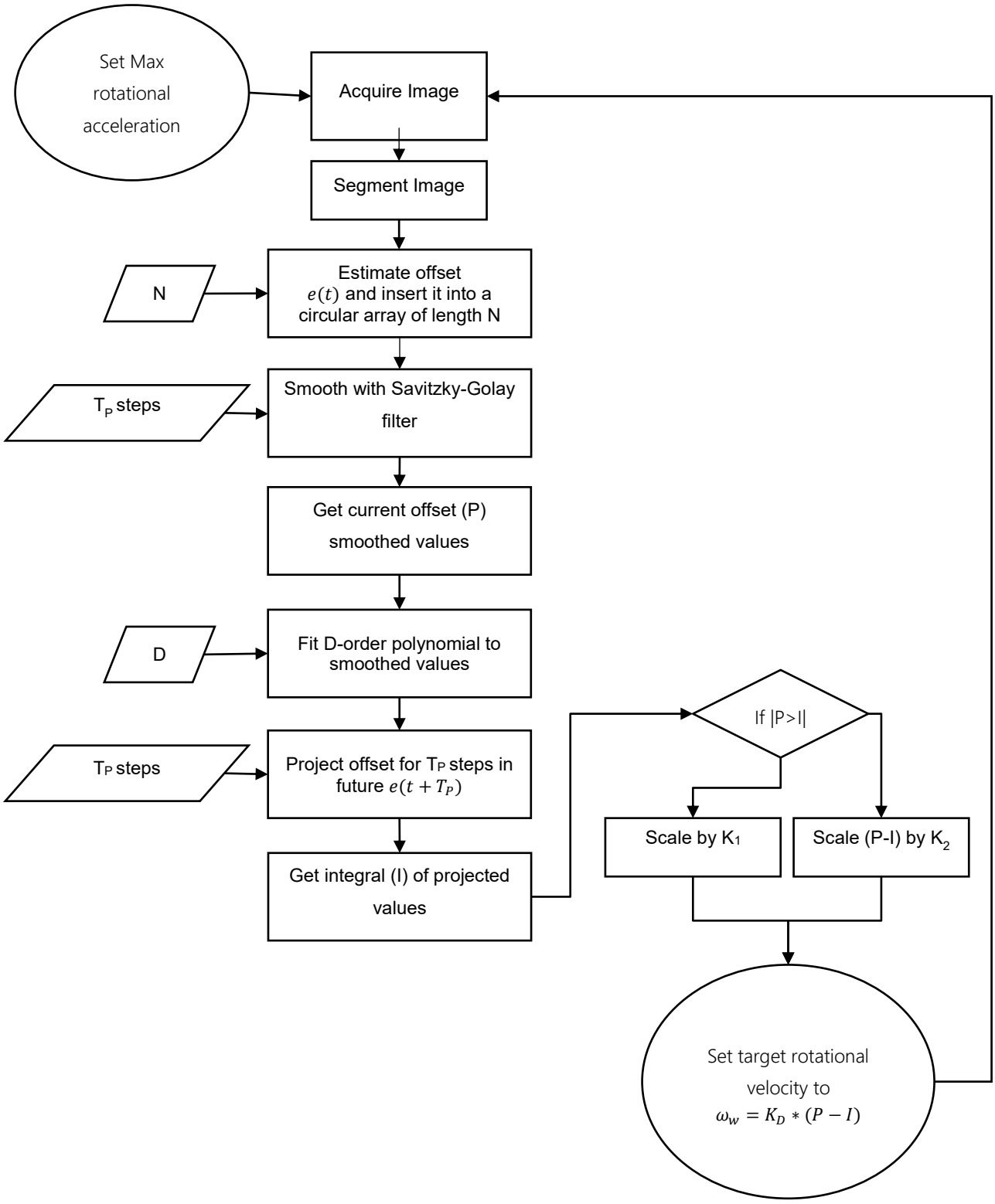

Figure 3-17: Flowchart of Stage 2 PID control algorithm.

15000 steps/s². The output min and output max were set to 0 and 1000000 respectively.

3.3 Experimental setup Stage 1

Two row arrangements were designed to test the guidance system on a straight path and a sinusoidal path (Figure 3-19). A green garden hose was used as an artificial row as it offered the possibility to be reconfigured on demand and the advantage of keeping the same appearance

Figure 3-18: Stage 1 test track with the experimental tractor at the far end.

throughout trials (Figure 3-18). It also made it easier to obtain repeated lateral error estimates. It was clamped down on the tarmac surface to ensure immovability during passes. The tractor was first positioned at the end of the path with the camera centered over the row and the front wheels turned straight forward. The tractor moved forward, and the auto-guidance and data

acquisition systems were engaged. The tractor traveled along the row until it reached the end of the segment. After entering the next pass, the auto-guidance system was engaged until the tractor reached the end. After that the operator again took control and manually steered the tractor for the headland maneuver. Test runs were conducted at a travel speed of 2,5 ± 0,2 m/s. Each combination of hose arrangement and direction was tested three times, resulting in 12 total test runs. All the test runs were completed within a two-hour period (a few minutes were necessary between tests to reposition the tractor). Data points were taken from each run as described above and saved to a .csv file in order to calculate the path mean absolute lateral error, the RMSE, and the 95th percentile. The RMSE was calculated per Eq.9:

$$RMSE = \sqrt{\sum_{i=1}^{N} \frac{e_i^2}{N}} \tag{9}$$

where e_i = lateral error based on camera offset detection.

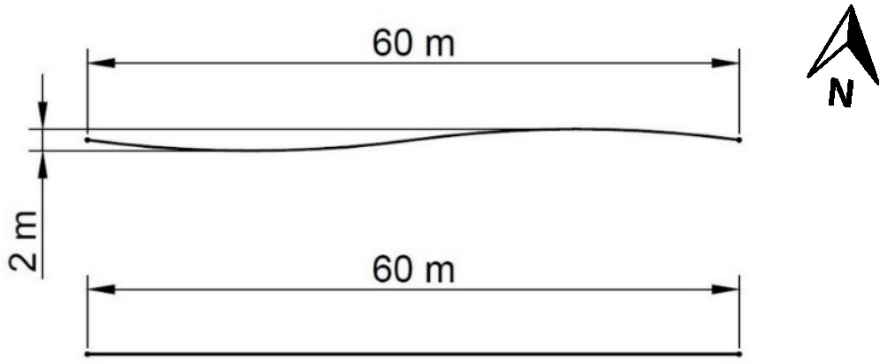

Figure 3-19: Arrangement of the two test layouts for the garden hose, on tarmac with the North arrow.

3.4 Experimental setup Stage 2

The two row arrangements were elongated to 120 m in order to obtain more data on each run and the amplitude of the curved track was increased to 4 m (Figure 3-20). The green garden hose was used again as an artificial row. Test runs were conducted at travel speeds of 1,0 ± 0,2 m/s, 2,0 ± 0,2 m/s and 3,0 ± 0,2 m/s. Each combination of hose arrangement, direction of travel, speed and projection time length was randomly tested four times, resulting in 96 total test runs. Two repetitions were completed on 29 June 2016 in the late afternoon under cloudy skies and two repetitions on 30 June 2016 in the morning under cloudless shiny skies (the light conditions were not controlled for).

Figure 3-20: Stage 2 curved and straight test tracks.

4. RESULTS AND DISCUSSION

4.1 Stage 1 test

The performance measures are shown in Table 1 for both the straight and curved paths at 2.5 m/s, which turned out to be the fastest speed at which the system was showing stable behavior and minimal hunting oscillation. Figure 4-1 demonstrates that the performance of the vision guidance system at a speed of 2.5 m/s in the straight and the curved path system was visually as good as a human driving the tractor. The Stage 1 vision guidance had a 95th percentile error over the straight track of 16.7 cm whereas the 95th percentile error on the curved track was 20.6 cm as shown in Table 1.

Based on this experiment, the machine vision algorithm clearly segmented the path to be traversed. However, the lateral offset data appeared noisy, and the use of a linear regression might have created aberrations in the predicted errors, resulting in unstable behavior. Control enhancements are implemented in Stage 2 to improve the guidance system performance further to attain the aforementioned 5 cm target.

Table 1: Results from Stage 1 test.

Trajectory	Direction of Travel	Trial	Number of Records	Mean Absolute Error (cm)	RMSE (cm)	95th Percentile (cm)
Straight	East-West	1	110	4.8	6.1	12.5
		2	110	5.9	7.1	13.7
		3	109	7.8	9.6	17.4
	West-East	1	111	10.9	12.5	20.9
		2	112	2.3	2.6	4.4
	Overall		552	6.9	8.6	16.7
Curved	East-West	1	98	15.2	16.4	27.5
		2	127	4.1	5.0	8.9
		3	129	4.6	5.7	10.6
	West-East	1	128	9.1	11.3	20.4
		2	129	9.8	11.8	19.6
		3	98	13.6	14.9	22.0
	Overall		709	9.2	11.3	20.6

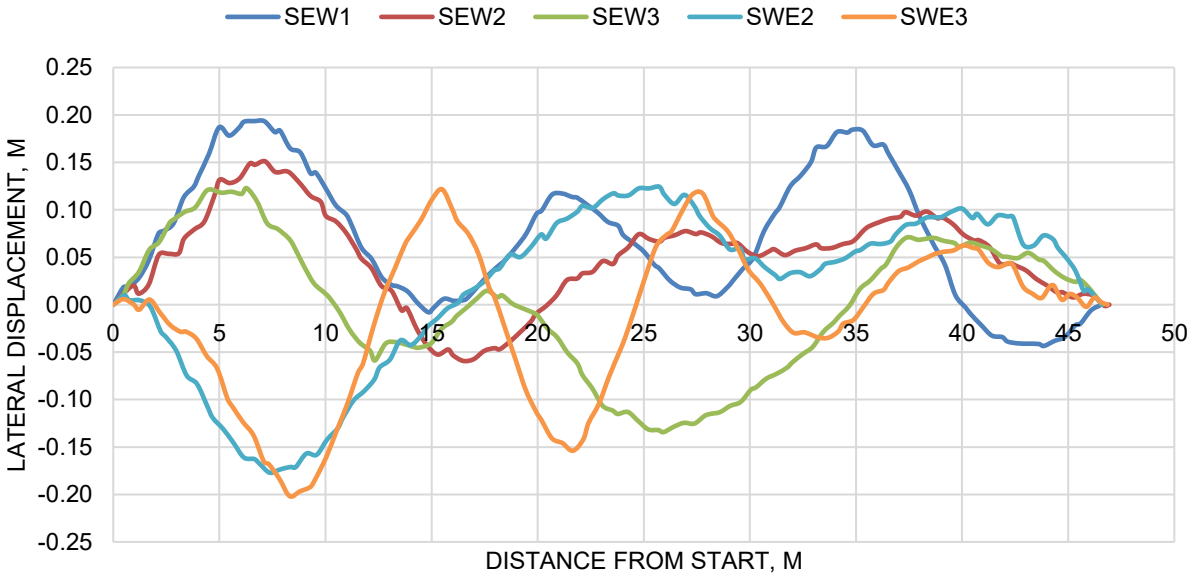

Figure 4-1: Performance of guidance system on the straight path.

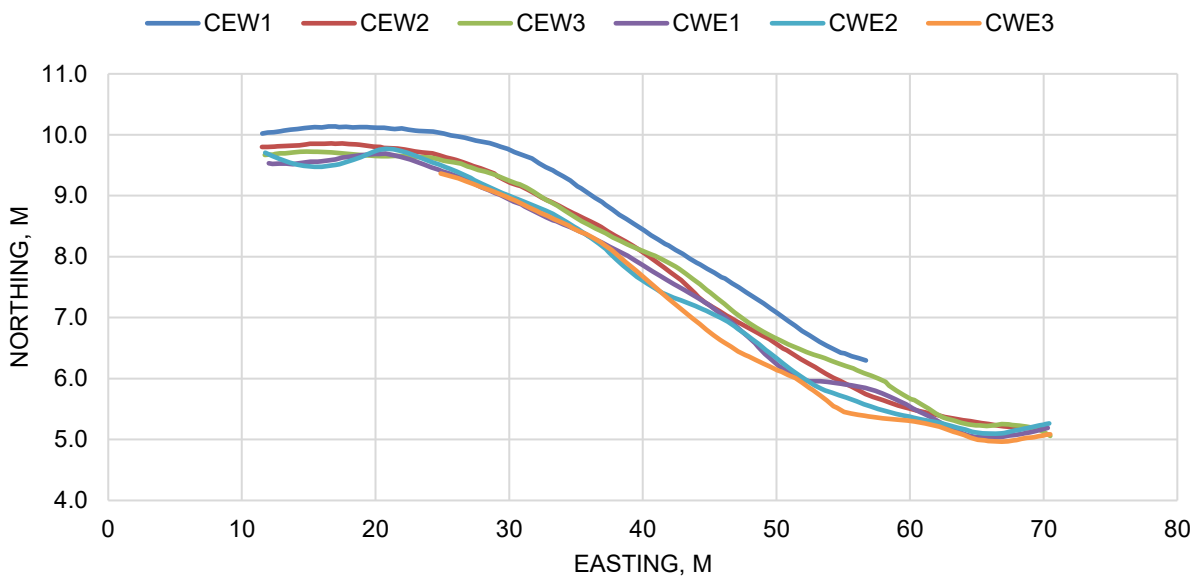

Figure 4-2: Performance of guidance system on the curved path.

4.2 Stage 2 test

Table 2 presents a summary of all 24 combinations of the 96 test runs (4 repetitions), including the number of records for both the straight and curved paths at speeds of 1 m/s up to 3 m/s, which turned out to be the fastest speed at which the system was showing stable behavior and minimal hunting oscillation. All 96 data sets were cropped to start and end at roughly the same geographical coordinates. The 95th percentile lateral errors were computed. Data was analyzed according to the GLM (Generalized Linear Model; ANOVA) procedure of SAS (SAS Institute, Cary, USA), as well as the MIXED procedure. The models used involved the trajectory factor (Straight or Curved), the direction factor (East-west or West-east), the speed the tractor was driven at (1 m/s, 2 m/s or 3 m/s), and the number of projection frames factor (9 frames or 21 frames). All the factors were treated as fixed. There were exactly four repetitions for each of the 24 different combinations of factors. The dependent variable in this experiment was the 95th percentile lateral error in centimeters for each of the 96 runs (see Table 3).

A standard Tukey's range test was performed to determine significant differences between combinations. Significance was determined with $\alpha \leq 0.05$. As can be seen, there was a noticeably smaller discrepancy between the straight and curved track results as compared with the Stage 1 experiment. In fact, the statistical analysis has shown that they are only significantly different at $\alpha = 0.05$ at a speed of 2 m/s. Trials at 1 m/s and 3 m/s showed no significant difference between the straight and curved configurations.

Table 2: Results from Stage 2 test.

Trajectory	Direction of Travel	Speed (m/s)	Projection Time Length (frames)	Number of Records	95th Percentile (cm)	Results with same letter are not significantly different
Straight	East-West	1	9	11930	7.0 ± 1.5	A B C
			21	11879	3.1 ± 1.5	A
		2	9	6476	11.1 ± 1.5	A B C D E F G
			21	6544	8.1 ± 1.5	A B C D E
		3	9	4416	15.3 ± 1.5	I J D E F G H
			21	4397	19.5 ± 1.5	I J H
	West-East	1	9	11950	6.7 ± 1.5	A B C
			21	11825	3.4 ± 1.5	A
		2	9	6652	9.0 ± 1.5	A B C D E F
			21	6556	7.5 ± 1.5	A B C D
		3	9	4305	14.7 ± 1.5	I J C D E F G H
			21	4233	20.6 ± 1.5	I J
Curved	East-West	1	9	12176	8.6 ± 1.5	A B C D E F
			21	11608	4.7 ± 1.5	A B
		2	9	6384	13.3 ± 1.5	I C D E F G H
			21	6411	13.8 ± 1.5	I J C D E F G H
		3	9	4509	16.0 ± 1.5	I J E F G H
			21	4265	21.6 ± 1.5	J
	West-East	1	9	12003	7.1 ± 1.5	A B C
			21	12001	3.5 ± 1.5	A
		2	9	6061	11.7 ± 1.5	B C D E F G H
			21	6395	13.4 ± 1.5	I C D E F G H
		3	9	4237	16.6 ± 1.5	I J F G H
			21	4414	17.7 ± 1.5	I J G H

Table 3: ANOVA of 95th percentile lateral error (cm) (Stage 2 experiment).

Source	df	MS	p-value
Trajectory	1	80.377031	0.0039
Direction	1	18.003383	0.1622
Speed	2	1200.649938	<.0001
Frames	1	0.00208	0.9879
Trajectory*Speed	2	32.16723	0.0335
Speed*Frames	2	125.924111	<.0001
Trajectory*Direction*Speed*Frames	14	4.916887	0.8977
Error	72	9.027846	

All three speeds yielded significantly different mean 95th percentile errors with 1 m/s just shy of attaining the 5-cm objective with 5.5 cm. 2 m/s had an 11.0 cm mean 95th percentile error, 3 m/s stood at 17.7 cm. A 17.7 cm 95th percentile error is acceptable under certain circumstances and is comparable to the performance of a manual guidance system (manual guidance ranges from 5 or 10 cm to 30 cm pass-to-pass according to Heraud & Lange 2009); nonetheless, it is imperfect and could certainly be improved further. It is to note; however, that the placement of the position sensor (camera) was the same as that of the control point as the camera performed both tasks. Having a second camera as a control point on the front axle of the tractor (730 mm behind the position sensor) might have reduced phase lag.

There was no statistically significant difference between East-West trials and West-East trials at $\alpha = 0.05$, which was expected.

The statistical analysis confirmed that there was a significant interaction between the speed the tractor was traveling at and the number of projection frames. Linear regressions of the 95th percentile lateral error as a function of ground speed on Figures 4-4 and 4-5 reveal that the performance of the guidance system was better at 1 m/s and 21 frames than at nine frames and that it was better at 3 m/s and nine frames than at 21 frames. There was no significant difference at 2 m/s between nine frames and 21 frames, suggesting that the actual number of projected frames at that speed should have been somewhere between nine and 21. These results clearly demonstrate the linear relationship between lateral error and speed and the necessity for an adjustable length of projection (number of projected frames) as a negative linear function of speed, which would easily be implemented using the groundspeed estimation algorithm.

During the tuning phase of the guidance system, it became evident that a slightly bigger steering actuator capable of faster accelerations would improve the performance of the system at higher speeds. A bigger motor would also allow the guidance system to work on smaller vehicles that are not equipped with power steering.

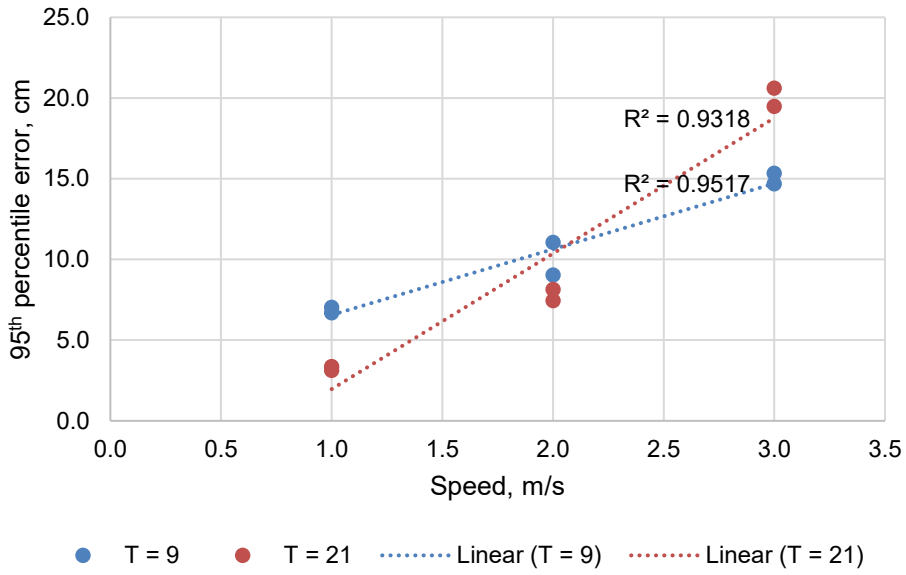

Figure 4-3: 95th percentile lateral error versus speed on the straight track.

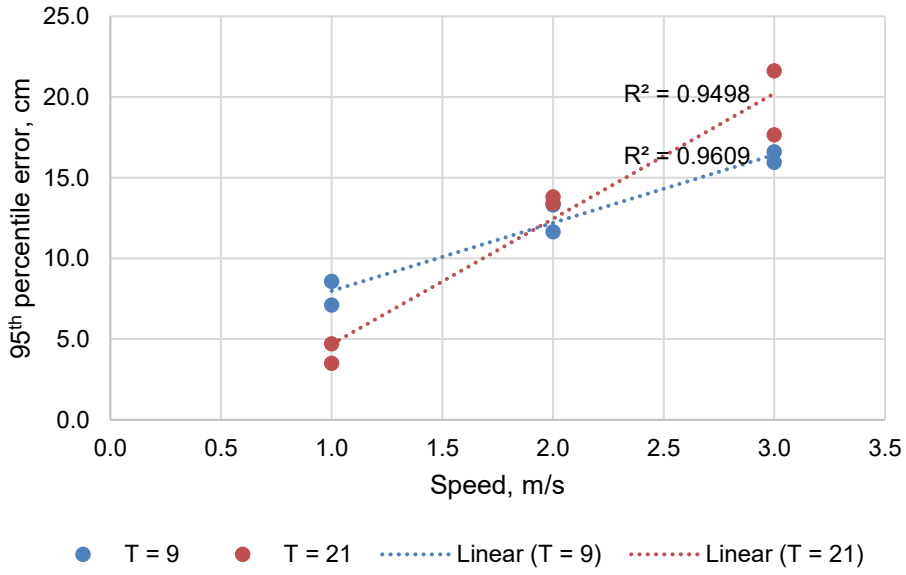

Figure 4-4: 95th percentile lateral error versus speed on the curved track.

4.3 Future improvements

Several uncontrollable variables such as soil conditions, terrain slope, and travel speed can have a substantial effect on agricultural vehicle dynamics. An inventive method for addressing the non-linear nature of the steering system and eliminating the necessity to constantly re-tune the guidance algorithm parameters (e.g., when switching between tractors) is to use an adaptive control system. Q-learning (Watkins & Dayan 1992) is a model-free reinforcement learning technique that would be suitable for this kind of control system. Q-learning is based on the principle of a reward mechanism meaning that, for a given action, the resulting behavior of a system is classified and subsequently rewarded or punished. A reward is attributed to the actions which resulted in positive behavior for a particular state of the system (e.g., adjusting gain offsets due to bias) whereas a punishment is attributed to the actions that led to negative behavior (e.g., high gains resulting in overshooting). Eventually, the learning process converges towards a non-linear response matrix which adapts to the current working environment of the system. Applications of adaptive controllers in agriculture have the potential to improve performance and reduce calibration when working with varying vehicle configurations and field conditions.

Future work will also bring the guidance system to the field to assess performance in presence of uneven ground conditions and while pulling implements. More compact designs will be evaluated, and improvements to ergonomics and manufacturability of the system will be implemented.

5. CONCLUSION

A universal quick-install computer vision guidance system was developed for following row crops. The camera that was used as a posture sensor was also used as a control point, and the system was tested on the tarmac with a garden hose, following an original evaluation protocol. The guidance system matched or surpassed commercially available systems regarding performance. However, only at 1 m/s did the guidance system reach the target 5 cm 95th percentile lateral error. Although the system does not offer the highest level of precision, its simple yet robust algorithm, its ergonomic features, its ease of installation and its versatility make it a serious contender in the arena of guidance systems. For the fertilizer and phytosanitary products application, it might increase profitability and protect the environment by reducing the overlaps, misses and damage to row crops.

REFERENCES

Alleyne, a. & DePoorter, M., 1997. Lateral displacement sensor placement and forward velocity effects\non stability of lateral control of vehicles. *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*, *3*(June), pp.1593–1597.

Bay, H., Tuytelaars, T. & Van Gool, L., 2006. SURF: Speeded up robust features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3951* LNCS, pp.404–417.

Becker, W.J., Shoup, W.D. & Sinden, J.V., 1983. Field measurement and analysis of operator speed-load stress. , (ASAE Paper No. 83-1628).

Benson, E.R., Reid, J.F. & Zhang, Q., 2003. Machine Vision-Based Guidance System for an Agricultural Small-Grain Harvester. *Transactions of the ASAE*, 46(4), pp.1255–1264.

Billingsley, J. & Schoenfisch, M., 1995. Vision-guidance of agricultural vehicles. *Auton Robot Autonomous Robots*, *2*(1), pp.65–76.

Cho, S.I. & Ki, N.H., 1999. Autonomous Speed Sprayer Guidance Using Machine Vision and Fuzzy Logic. *Transactions of the ASAE*, 42(4), pp.1137–1144.

Choi, C.H., Erbach, D.C. & R.J., S., 1990. Navigational Tractor Guidance System. *Transactions of the ASAE*, *33*(3), pp.699–706.

CLAAS KGaA mbH, 2016. Smart Equipment. Available at: http://www.claasofamerica.com/product/precision-farming/smart-equipment/cam-pilot [Accessed July 20, 2016].

Deere & Company, 2016. Guidance Systems. Available at: https://www.deere.ca/en_CAF/products/equipment/agricultural_management_solutions/guidance_systems/guidance_systems.page [Accessed July 20, 2016].

Derrick, J.B. & Bevly, D.M., 2009. Adaptive steering control of a farm tractor with varying yaw rate properties. *ROB Journal of Field Robotics*, *26*(6–7), pp.519–536.

Dobbs, T., 2013. Farms of the Future Will Run on Robots and Drones. *PBS Online*. Available at: http://www.pbs.org/wgbh/nova/next/tech/farming-with-robotics-automation-and-sensors/ [Accessed March 12, 2016].

Easterly, D.R. et al., 2010. Using a vision sensor system for performance testing of satellite-based tractor auto-guidance. *Computers and Electronics in Agriculture*, *72*(2), pp.107–118.

Erickson, B. & Widmar, D.A., 2015. *2015 Precision Agricultural Services: Dealership Survey Results*,

Available at: http://agribusiness.purdue.edu/files/resources/2015-crop-life-purdue-precision-dealer-survey.pdf.

Feng, L., He, Y. & Zhang, Q., 2004. Dynamic Trajectory Model of a Tractor-Implement System for Automated Navigation Applications. , 1(701), pp.66–88.

Gerrish, J.B. et al., 1997. Self-steering Tractor Guided by Computer-vision. , 13(517), pp.559–563.

Gomez-Gil, J. et al., 2011. Development and validation of globally asymptotically stable control laws for Automatic tractor guidance. *Applied Engineering in Agriculture*, 27(6), pp.1099–1108.

Han, S. et al., 2004. A guidance directrix approach to vision-based vehicle guidance systems. *Computers and Electronics in Agriculture*, 43(3), pp.179–195.

Heraud, J. a. & Lange, A.F., 2009. Agricultural automatic vehicle guidance from horses to GPS: How we got here, and where we are going. *ASABE Distinguished Lecture Series - Agricultural Automatic Vehicle Guidance from Horses to GPS: How We Got Here, and Where We are Going*, (913), pp.1–67. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-76749105599&partnerID=tZOtx3y1.

Kirk, T.., 1974. A furrow-following tractor guidance system. , 1974(c), pp.1–60. Available at: http://ecommons.usask.ca/handle/10388/etd-09032008-131205.

Kirk, T.G. & Krause, A.E., 1976. Swather Edge Guide Steering Control System.

Kise, M., Zhang, Q. & Rovira Más, F., 2005. A Stereovision-based Crop Row Detection Method for Tractor-automated Guidance. *Biosystems Engineering*, 90(4), pp.357–367.

Kodagoda, K.R.S., Wijesoma, W.S. & Teoh, E.K., 2002. Fuzzy speed and steering control of an AGV. *IEEE Trans. Contr. Syst. Technol. IEEE Transactions on Control Systems Technology*, 10(1), pp.112–120.

Larsen, W.E., Tyler, D.A. & Nielsen, G.A., 1988. Field navigation using the global positioning system (GPS).

Lenain, R. et al., 2006. High accuracy path tracking for vehicles in presence of sliding: Application to farm vehicle automatic guidance for agricultural tasks. *Autonomous Robots*, 21(1), pp.79–97.

Marchant, J.A. & Brivot, R., 1995. Real-Time Tracking of Plant Rows Using a Hough Transform. *Real-Time Imaging*, 1(5), pp.363–371.

Matias, P.M. & Gil, J.G., 2007. Automatic Guidance of a Tractor Using Computer Vision. *Lecture notes in computer science.*, (4815), pp.169–176.

McMahon, C.B., Tennes, B.R. & Burkhardt, T.H., 1987. System Using Ultrasonic Sensors. , pp.87–89.

O'Connor, M. et al., 1996. Automatic steering of farm vehicles using GPS. *Precision Agriculture*, pp.767–777.

Okamoto, H. et al., 2002. Automatic guidance system with crop row sensor. *Transactions of the Asabe*, (701), pp.307–316.

OpenCV, 2014. Open Source Computer Vision Library. , Ver. 2.4.7.

Palmer, R.J. & Matheson, S.K., 1988. Impact of navigation on farming. , (ASAE Paper No. 881602).

Parish, R.L. & Goering, C.E., 1970. Developing an Automatic Steering System for a Hydrostatic Vehicle.

Reid, J. & Searcy, S., 1987. Vision-based guidance of an agriculture tractor. *Control Systems Magazine, IEEE*, 7(2), pp.39–43.

Slaughter, D.C., Chen, P. & Curley, R.G., 1999. Vision Guided Precision Cultivation. *Precision Agriculture Precision Agriculture*, 1(2), pp.199–217.

Snyder, W.H., 1885. Furrow Pilot: Traction Engine.

Stanhope, T., Adamchuk, V. & Desperrier Roux, J., 2014. Computer vision guidance of field cultivation for organic row crop production. , 7004.

Stombaugh, T., Benson, E. & Hummel, J., 1999. Guidance control of agricultural vehicles at high field speeds. *Transactions of the ASAE*, 42(2), pp.537–544. Available at: http://elibrary.asabe.org/azdez.asp?JID=3&AID=13387&v=42&i=2&CID=t1999&T=2%5Cnhttp://cat.inist.fr/?aModele=afficheN&cpsidt=1241029.

Subramanian, V., Burks, T.F. & Arroyo, A.A., 2006. Development of machine vision and laser radar based autonomous vehicle guidance systems for citrus grove navigation. *Computers and Electronics in Agriculture*, 53(2), pp.130–143.

Trimble Navigation, L., 2016. EZ-Steer assisted steering system. Available at: http://www.trimble.com/agriculture/ez-steer.aspx?dtID=overview/ [Accessed July 20, 2016].

Warner, M.G.R. & Harries, G.O., 1972. An ultrasonic guidance system for driverless tractors. *Journal of Agricultural Engineering Research*, 17(1), pp.1–9. Available at: http://www.sciencedirect.com/science/article/pii/S0021863472800118.

Watkins, C.J. & Dayan, P., 1992. Q-learning. In *Machine learning*. pp. 279–292.

Xue, J., Zhang, L. & Grift, T.E., 2012. Variable field-of-view machine vision based row guidance of an agricultural robot. *Computers and Electronics in Agriculture*, 84(0), pp.85–91.

APPENDIX A: Python Code

```
"""
Row Assist
Precision Agriculture and Soil Sensing Group (PASS)
McGill University, Department of Bioresource Engineering

"""

__author__ = 'Trevor Stanhope'
__version__ = '1.0'

## Libraries
import Image
import cv2, cv
from src import CVGS
import serial
import json, ast
import numpy as np
import scipy.signal as sig
import thread
import gps
import sys, os, time
from datetime import datetime
from ctypes import *
from time import sleep
from Phidgets.PhidgetException import PhidgetErrorCodes, PhidgetException
from Phidgets.Events.Events import AttachEventArgs, DetachEventArgs,
ErrorEventArgs, InputChangeEventArgs, CurrentChangeEventArgs,
StepperPositionChangeEventArgs, VelocityChangeEventArgs
from Phidgets.Devices.Stepper import Stepper
sys.settrace

## Constants
CONFIG_FILE = "settings.json"

## Class
class RowAssist:

    def pretty_print(self, task, msg, *args):
        try:
            date = datetime.strftime(datetime.now(), "%H:%M:%S.%f")
            output = "%s\t%s\t%s" % (date, task, msg)
            print output
        except:
            pass

    def __init__(self, config_file):

        # Load Config
        self.pretty_print("CONFIG", "Loading %s" % config_file)
        self.config = json.loads(open(config_file).read())
        for key in self.config:
            try:
                getattr(self, key)
```

```
            except AttributeError as error:
                setattr(self, key, self.config[key])

        self.bgr1 = np.zeros((640, 480, 3), np.uint8)
        self.bgr2 = np.zeros((640, 480, 3), np.uint8)
        self.cv_speed = 0.0

        # Initializers
        self.run_threads = True
        self.init_log() # it's best to run the log first to catch all events
        self.init_camera()
        self.init_display()
        self.init_cv_groundspeed()
        self.init_stepper()
        self.init_controller()
        self.init_pid()
        self.init_gps()
        self.init_qlearner()

    ## Initialize Display
    def init_display(self):
        thread.start_new_thread(self.update_display, ())

    ## Initialize Ground Speed Matcher
    def init_cv_groundspeed(self):
        """
        Initialize CV GroundSpeed
        """
        self.pretty_print('GSV6', 'Initializing Groundspeed Matcher ...')
        try:
            self.CVGS = CVGS.CVGS()
            thread.start_new_thread(self.update_groundspeed, ())
        except Exception as e:
            raise e

    # Initialize QLearner
    def init_qlearner(self):
        """
        Initialize Q-Learner
        """
        self.pretty_print('QLRN', 'Initializing Q-Learner ...')
        try:
            """
            X: STEERING_WHEEL POSITION
            Y: ERROR
            """
            self.qmatrix = np.zeros((self.OUTPUT_MAX, self.CAMERA_WIDTH, 3),
np.uint8)
        except Exception as e:
            raise e

    # Initialize Camera
    def init_camera(self):
        """
        Initialize Camera
        """
        # Setting variables
```

```
        self.pretty_print('CAM', 'Initializing CV Variables ...')
        self.pretty_print('CAM', 'Camera Width: %d px' % self.CAMERA_WIDTH)
        self.pretty_print('CAM', 'Camera Height: %d px' % self.CAMERA_HEIGHT)
        self.pretty_print('CAM', "Camera Rotated: %s" %
str(self.CAMERA_ROTATED))
        if self.CAMERA_ROTATED:
            self.CAMERA_CENTER = self.CAMERA_HEIGHT / 2
        else:
            self.CAMERA_CENTER = self.CAMERA_WIDTH / 2
        self.pretty_print('CAM', 'Camera Center: %d px' % self.CAMERA_CENTER)
        self.pretty_print('CAM', 'Camera Depth: %d cm' % self.CAMERA_DEPTH)
        self.pretty_print('CAM', 'Camera FOV: %f rad' % self.CAMERA_FOV)
        self.pretty_print('CAM', 'Ground Width: %d cm' % self.GROUND_WIDTH)
        self.pretty_print('CAM', 'Error Tolerance: +/- %d cm' %
self.ERROR_TOLERANCE)
        self.PIXEL_PER_CM = self.CAMERA_WIDTH / self.GROUND_WIDTH
        self.pretty_print('CAM', 'Pixel-per-cm: %d px/cm' %
self.PIXEL_PER_CM)
        self.PIXEL_RANGE = int(self.PIXEL_PER_CM * self.ERROR_TOLERANCE)
        self.pretty_print('CAM', 'Pixel Range: +/- %d px' % self.PIXEL_RANGE)

        # Set Thresholds
        self.threshold_min = np.array([self.HUE_MIN, self.SAT_MIN,
self.VAL_MIN], np.uint8)
        self.threshold_max = np.array([self.HUE_MAX, self.SAT_MAX,
self.VAL_MAX], np.uint8)

        # Attempt to set each camera index/name
        self.pretty_print('CAM', 'Initializing Camera ...')
        self.camera = None
        self.bgr = None
        self.mask = None
        self.camera_idx = 0
        try:
            while self.camera == None:
                cam = cv2.VideoCapture(self.camera_idx)
                cam.set(cv.CV_CAP_PROP_SATURATION, self.CAMERA_SATURATION)
                cam.set(cv.CV_CAP_PROP_FRAME_HEIGHT, self.CAMERA_HEIGHT)
                cam.set(cv.CV_CAP_PROP_FRAME_WIDTH, self.CAMERA_WIDTH)
                for i in range(10):
                    (s, bgr) = cam.read()
                if s:
                    self.camera = cam
                elif self.camera_idx > 3:
                    print "NO CAMERA ATTACHED!"
                    exit(1)
                else:
                    self.camera_idx += 1
        except Exception as error:
            self.pretty_print('CAM', 'CAM ERROR: %s' % str(error))

    # Initialize PID Controller
    def init_pid(self):
        self.pretty_print('PID', 'Initializing Control System')
        # Initialize variables
        try:
            self.estimated = 0
```

```
            self.projected = 0
            self.pretty_print('PID', 'Default number of samples: %d' %
self.NUM_SAMPLES)
            self.offset_history = [0] * self.NUM_SAMPLES
            self.pretty_print('PID', 'Setup OK')
        except Exception as error:
            self.pretty_print('PID', 'ERROR: %s' % str(error))

    # Initialize Log
    def init_log(self):
        self.pretty_print('LOG', 'Initializing Log')
        self.LOG_NAME = datetime.strftime(datetime.now(), self.LOG_FORMAT)
        self.pretty_print('LOG', 'New log file: %s' % self.LOG_NAME)
        gps_params = ['gps_lat', 'gps_long', 'gps_speed', 'gps_alt',
'gps_quality', 'gps_satellites']
        nongps_params = ['time', 'hz', 'offset', 'est', 'proj', 'diff',
'velocity', 'steps', 'encoder']
        if self.GPS_ON:
            self.log_params = gps_params + nongps_params
        else:
            self.log_params = nongps_params
        try:
            self.log = open('logs/' + self.LOG_NAME + '.csv', 'w')
            # Write config settings
            for k,v in self.config.iteritems():
                self.log.write(k + ',' + str(v) + '\n')
            # Write headers
            self.log.write(','.join(self.log_params + ['\n']))
            self.pretty_print('LOG', 'Setup OK')
            self.vid_writer = cv2.VideoWriter('logs/' + self.LOG_NAME +
'.avi', cv.CV_FOURCC('M', 'J', 'P', 'G'), self.CAMERA_FPS,
(self.CAMERA_WIDTH, self.CAMERA_HEIGHT), True)
        except Exception as error:
            raise error

    # Initialize Controller
    def init_controller(self):
        self.pretty_print('CTRL', 'Initializing controller ...')
        try:
            self.pretty_print('CTRL', 'Device: %s' % str(self.SERIAL_DEVICE))
            self.pretty_print('CTRL', 'Baud Rate: %s' %
str(self.SERIAL_BAUD))
            self.controller = serial.Serial(self.SERIAL_DEVICE,
self.SERIAL_BAUD, timeout=0.05)
            self.angle = 0
            self.angle_rate = 0
            self.encoder = 0
            self.encoder_rate = 0
            self.encoder_rate_prev = 0
            thread.start_new_thread(self.update_controller, ())
            self.pretty_print('CTRL', 'Setup OK')
        except Exception as error:
            self.pretty_print('CTRL', 'ERROR: %s' % str(error))
            exit(1)

    # Initialize Stepper
    def init_stepper(self):
```

```
        # Constants
        self.pretty_print('STEP', 'Output Minimum: %d' % self.OUTPUT_MIN)
        self.pretty_print('STEP', 'Output Maximum: %d' % self.OUTPUT_MAX)

        # Create
        try:
            self.pretty_print("STEP", "Creating phidget object....")
            self.stepper = Stepper()
        except PhidgetException as e:
            self.pretty_print("STEP", "Phidget Exception %i: %s" % (e.code,
e.details))

        # Open
        try:
            self.pretty_print("STEP", "Opening phidget object....")
            self.stepper.openPhidget()
        except PhidgetException as e:
            self.pretty_print("STEP", "Phidget Exception %i: %s" % (e.code,
e.details))

        # Settings
        try:
            self.pretty_print("STEP", "Configuring stepper settings ...")
            self.stepper.setOnAttachHandler(self.StepperAttached)
            self.stepper.setOnDetachHandler(self.StepperDetached)
            self.stepper.setOnErrorhandler(self.StepperError)

self.stepper.setOnCurrentChangeHandler(self.StepperCurrentChanged)
            self.stepper.setOnInputChangeHandler(self.StepperInputChanged)

self.stepper.setOnPositionChangeHandler(self.StepperPositionChanged)

self.stepper.setOnVelocityChangeHandler(self.StepperVelocityChanged)
        except PhidgetException as e:
            self.pretty_print("STEP", "Phidget Exception %i: %s" % (e.code,
e.details))

        # Attach
        try:
            self.pretty_print("STEP", "Attaching stepper motor ...")
            self.stepper.waitForAttach(1000)
        except PhidgetException as e:
            self.pretty_print("STEP", "Phidget Exception %i: %s" % (e.code,
e.details))
            try:
                self.stepper.closePhidget()
            except PhidgetException as e:
                self.pretty_print("STEP", "Phidget Exception %i: %s" %
(e.code, e.details))
            exit(1)
        else:
            self.DisplayDeviceInfo()

        # Engage
        try:
            self.pretty_print("STEP", "Engaging stepper motor ...")
```

```
            self.output = (self.OUTPUT_MIN + self.OUTPUT_MAX) / 2
            self.stepper.setCurrentPosition(0, (self.OUTPUT_MIN +
self.OUTPUT_MAX) / 2)
            self.stepper.setTargetPosition(0, (self.OUTPUT_MIN +
self.OUTPUT_MAX) / 2)
            self.stepper.setEngaged(0, False)
            self.stepper.setVelocityLimit(0, self.VELOCITY)
            self.stepper.setAcceleration(0, self.ACCELERATION)
            self.stepper.setCurrentLimit(0, self.AMPS)
        except Exception as error:
            self.pretty_print("STEP", "ERROR: %s" % str(error))
            exit(2)

    # Initialize GPS
    def init_gps(self):
        """ Initialize GPS """
        self.pretty_print('GPS', 'Initializing GPS ...')
        self.gps_latitude = 0
        self.gps_longitude = 0
        self.gps_altitude = 0
        self.gps_speed = 0
        self.gps_quality = 0
        self.gps_satellites = 0
        try:
            self.gps = serial.Serial(self.GPS_DEVICE, self.GPS_BAUD)
            thread.start_new_thread(self.update_gps, ())
            self.pretty_print("GPS", "GPS connected")
        except Exception as err:
            self.pretty_print('GPS', 'WARNING: GPS not available! %s' %
str(err))

    ## Update Learner
    def update_learner(self, ph1, e, group):
        self.qmatrix[ph1,e,:] = self.qmatrix[ph1,e,:] + group
        return group

    ## Capture Image
    def capture_image(self):
        """
        1. Attempt to capture an image
        2. Repeat for each capture interface
        """
        try:
            (s, bgr) = self.camera.read()
            if s is False:
                self.pretty_print('CAM', 'ERROR: Capture failed')
                bgr = None
        except KeyboardInterrupt:
            raise KeyboardInterrupt
        except Exception as e:
            raise e
        self.bgr2 = self.bgr1
        self.bgr1 = bgr # Update the BGR (raw)
        if self.CAMERA_ROTATED:
            return np.rot90(bgr)
        else:
            return bgr
```

```
    ## Plant Segmentation Filter
    def plant_filter(self, bgr):
        """
        1. RBG --> HSV
        2. Set minimum saturation equal to the mean saturation
        3. Set minimum value equal to the mean value
        4. Take hues within range from green-yellow to green-blue
        """
        if bgr is not None:
            try:
                hsv = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)
                self.threshold_min[1] = 128 #np.percentile(hsv[:,:,1], 100 *
self.SAT_MIN / 256 # overwrite the saturation minima
                self.threshold_min[2] = np.percentile(hsv[:,:,2], 100 *
self.VAL_MIN / 256) # overwrite the value minima
                self.threshold_max[1] = 255 #np.percentile(hsv[:,:,1], 100 *
self.SAT_MAX / 256) # overwrite the saturation minima
                self.threshold_max[2] = np.percentile(hsv[:,:,2], 100 *
self.VAL_MAX / 256) # overwrite the value minima
                mask = cv2.inRange(hsv, self.threshold_min,
self.threshold_max)
                #kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
                #mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
            except KeyboardInterrupt:
                raise KeyboardInterrupt
            except Exception as e:
                raise e
        return mask

    ## Find Plants
    def find_offset(self, mask):
        """
        1. Calculates the column summation of the mask
        2. Calculates the 95th percentile threshold of the column sum array
        3. Finds indicies which are greater than or equal to the threshold
        4. Finds the median of this array of indices
        5. Repeat for each mask
        """
        if mask is not None:
            try:
                column_sum = mask.sum(axis=0) # vertical summation
                centroid = int(np.argmax(column_sum) - self.CAMERA_CENTER +
self.CAMERA_OFFSET)
                idx = centroid
            except KeyboardInterrupt:
                raise KeyboardInterrupt
            except Exception as error:
                self.pretty_print('OFF', '%s' % str(error))
        else:
            idx = self.CAMERA_OFFSET
        return idx

    ## Best Guess for row based on multiple offsets from indices
    def estimate_error(self, idx):
        """
        Calculate errors for estimate, average, and differential
```

```
        """
        try:
            t = range(0, self.T1_COEF) # the time frame in the past
            t_plus = range(self.T1_COEF + 1, self.T1_COEF + self.T2_COEF) #
the time frame in the future
            self.offset_history.append(int(idx)) # add most recent raw value
to history
            while len(self.offset_history) > self.NUM_SAMPLES:
                self.offset_history.pop(0) # trim history to specified length
            smoothed_values = sig.savgol_filter(self.offset_history,
self.T1_COEF, 2)

            # Estimated
            e = int(smoothed_values[-1]) # get latest
            if e > self.CAMERA_CENTER: e = (self.CAMERA_CENTER) - 1
            elif e < -self.CAMERA_CENTER: e = -self.CAMERA_CENTER

            # Projected
            spline_filtered = np.polyfit(t, smoothed_values[-self.T1_COEF:],
deg=self.REGRESSION_DEG)
            projected_vals = np.polyval(spline_filtered, t_plus)
            de = np.mean(np.gradient(smoothed_values[-self.T1_COEF:]))
            ie = int(np.mean(projected_vals)) # integral error
            if ie > self.CAMERA_CENTER: ie = (self.CAMERA_CENTER)
            elif ie < -self.CAMERA_CENTER: ie = -self.CAMERA_CENTER
        except Exception as error:
            self.pretty_print("ROW", "Error: %s" % str(error))
        return e, ie, de

    ## Calculate Output (Supports different algorithms)
    def calculate_output(self, e, ie, de, v, d_phi, d_phi_prev):
        """
        Calculates the PID output for the stepper
        Arguments: est, proj, diff, speed
        Requires: OUTPUT_MAX, OUTPUT_MIN, CENTER_OUTPUT
        Returns: output
        """
        # Version 1 (Two-Stage D)
        if self.VERSION == 1:
            if e < 0:
                if e < ie:
                    output = self.OUTPUT_MAX
                    velocity = abs(e - ie) * self.VELOCITY *
self.D_APPROACH_COEF
                else:
                    output = self.OUTPUT_MIN
                    velocity =  abs(e - ie) * self.VELOCITY *
self.D_DRIFT_COEF
            elif e > 0:
                if e > ie:
                     output = self.OUTPUT_MIN
                     velocity = abs(e - ie) * self.VELOCITY *
self.D_APPROACH_COEF
                else:
                    output = self.OUTPUT_MAX
                    velocity = abs(e - ie) * self.VELOCITY *
self.D_DRIFT_COEF
```

```
            else:
                velocity = 0
                output = self.stepper.getCurrentPosition(0)

        # Version 2
        elif self.VERSION == 2:
            if e < 0:
                if e < ie:
                    output = self.OUTPUT_MAX
                    velocity = abs(e - ie) * self.D_APPROACH_COEF
                else:
                    output = self.OUTPUT_MIN
                    velocity =  abs(e - ie) * self.D_DRIFT_COEF
            elif e > 0:
                if e > ie:
                    output = self.OUTPUT_MIN
                    velocity = abs(e - ie) * self.D_APPROACH_COEF
                else:
                    output = self.OUTPUT_MAX
                    velocity = abs(e - ie) * self.D_DRIFT_COEF
            else:
                velocity = 0
                output = self.stepper.getCurrentPosition(0)

        # limit velocity to VELOCITY_MAX
        if velocity > self.VELOCITY: velocity = self.VELOCITY

        return int(output), int(velocity)

    ## Read Controller
    def update_controller(self):
        """ Get info from controller """
        a = time.time()
        b = time.time()
        while self.run_threads:
            event = None
            try:
                s = self.controller.readline()
                event = json.loads(s)
                b = time.time() # end timer
                if event is not None:
                    self.encoder = event['a']
                    self.angle = event['b']
                    self.encoder_rate = (event['a'] - self.encoder) / (b - a)
                    self.encoder = event['a']
            except Exception as error:
                print str(error)
            a = time.time() # reset timer

    ## Set Stepper
    def set_stepper(self, output, velocity):
        """ Set Stepper, returns the dead-reckoning number of steps/position
"""
        phi_current = self.stepper.getCurrentPosition(0)
        try:
            if self.stepper.isAttached():
```

```
            # Set Target Position
            self.stepper.setTargetPosition(0, output)

            # Set Velocity
            self.stepper.setVelocityLimit(0, abs(velocity))

    except KeyboardInterrupt:
        raise KeyboardInterrupt
    except Exception as e:
        self.close_stepper()
        raise e
    return phi_current

## Get Groundspeed
def get_groundspeed(self, images):
    """ Get the current groundspeed """
    return self.cv_speed # Return current speed

## Write to Log
def write_to_log(self, sample):
    """
    Write results to the log
    """
    a = time.time()
    try:
        data = [str(sample[k]) for k in self.log_params]
        newline = ','.join(data + ['\n'])
        self.log.write(newline)
    except KeyboardInterrupt:
        raise KeyboardInterrupt
    except Exception as e:
        self.pretty_print("LOG", str(e))
        raise Exception("Failed to write to file document")
    b = time.time()

## Update GPS
def update_gps(self):
    """
    1. Get the most recent GPS data
    2. Set global variables for lat, long and speed
    """
    while self.run_threads:
        try:
            sentence = self.gps.readline()
            sentence_parsed = sentence.rsplit(',')
            nmea_type = sentence_parsed[0]
            if nmea_type == '$GPVTG':
                self.gps_speed = float(sentence_parsed[7])
            elif nmea_type == '$GPGGA':
                self.gps_latitude = float(sentence_parsed[2])
                self.gps_longitude = float(sentence_parsed[4])
                self.gps_altitude = float(sentence_parsed[9])
                self.gps_quality = float(sentence_parsed[6])
                self.gps_satellites = float(sentence_parsed[7])
        except Exception as e:
            self.gps_latitude = 0.0
            self.gps_longitude = 0.0
```

```
                self.gps_altitude = 0.0
                self.gps_speed = 0.0
                self.gps_quality = 0
                self.gps_satellites = 0

    ## Estimate groundspeed (THREADED)
    def update_groundspeed(self, wait=0.05, hist_length=3):
        """ Needs: bgr1 and bgr2 """
        self.speed_hist = [0] * hist_length
        while self.run_threads:
            time.sleep(wait) # don't run too fast
            try:
                bgr1 = self.bgr1
                bgr2 = self.bgr2
            except Exception as e:
                raise e
            try:
                if not np.all(bgr1==bgr2):
                    cv_speed = 0 #self.CVGS.get_velocity(bgr1, bgr2,
fps=self.CAMERA_FPS)
                    self.speed_hist.reverse()
                    self.speed_hist.pop()
                    self.speed_hist.reverse()
                    self.speed_hist.append(cv_speed)
                    self.cv_speed = np.mean(self.speed_hist)
            except Exception as error:
                self.pretty_print('CVGS', 'CV001: %s' % str(error))

    ## Update Display (THREADED)
    def update_display(self):
        """ Flash BGR capture to user """
        try:
            cv2.namedWindow("test")
            while self.run_threads:
                try:

                    # Draw Display
                    bgr = np.dstack((self.mask, self.mask, self.mask))
                    bgr[:, self.CAMERA_CENTER,:] = 255
                    bgr[:, self.CAMERA_CENTER - self.PIXEL_RANGE, 0] = 255
                    bgr[:, self.CAMERA_CENTER + self.PIXEL_RANGE , 0] = 255
                    # draw estimated position
                    bgr[:,self.estimated + self.CAMERA_CENTER, 0] = 0
                    bgr[:,self.estimated + self.CAMERA_CENTER, 1] = 255
                    bgr[:,self.estimated + self.CAMERA_CENTER, 2] = 0
                    # draw projected position
                    bgr[:,self.projected + self.CAMERA_CENTER, 0] = 0
                    bgr[:,self.projected + self.CAMERA_CENTER, 1] = 0
                    bgr[:,self.projected + self.CAMERA_CENTER, 2] = 255
                    cv2.imshow("test", bgr)
                    if cv2.waitKey(5) == 27:
                        pass

                    # Grab the raw image to write to video
                    if self.CAMERA_ROTATED:
                        self.vid_writer.write(self.bgr1)
                except Exception as error:
```

```
                self.pretty_print('DISP', 'ERROR: %s' % str(error))
                time.sleep(1)
        except Exception as error:
            self.pretty_print('DISP', 'ERROR: %s' % str(error))

    ## Close Controller
    def close_stepper(self):
        """
        Close Controller
        """
        self.pretty_print('SYSTEM', 'Closing Stepper ...')
        try:
            self.stepper.setEngaged(0, False)
            self.stepper.closePhidget()
        except Exception as error:
            self.pretty_print('STEP', 'ERROR: %s' % str(error))

    ## Close
    def close(self):
        """
        Function to shutdown application safely
        1. Close windows
        2. Disable stepper
        3. Release capture interfaces
        """
        self.pretty_print('SYSTEM', 'Shutting Down Now!')
        self.run_threads = False
        try:
            self.close_stepper() ## Disable stepper
        except Exception as error:
            self.pretty_print('STEP', 'ERROR: %s' % str(error))
        try:
            self.controller.close() ## Disable Arduino
        except Exception as error:
            self.pretty_print('ARD', 'ERROR: %s' % str(error))
        try:
            self.camera.release() ## Disable camera
        except Exception as error:
            self.pretty_print('CAM', 'ERROR: %s' % str(error))
        try:
            self.vid_writer.release() ## Safely close video writer
        except Exception as error:
            self.pretty_print('VID', 'ERROR: %s' % str(error))
        cv2.destroyAllWindows() ## Close windows

    ## Run
    def run(self):
        """
        Function for Run-time loop
        1. Get initial time
        2. Capture image
        3. Generate mask filter for plant matter
        4. Calculate indices of rows
        5. Estimate row from image
        6. Get number of samples
        7. Calculate lateral error after filtering
        8. Send output response to stepper
```

```
        9. Throttle to desired frequency
        10. Log results to DB
        11. Display results
        """
        start_time = time.time()
        iterations = 0
        while self.run_threads:
            iterations += 1
            if time.time() - start_time > self.DURATION:
                self.close()
            try:
                a = time.time()
                bgr = self.capture_image()
                mask = self.plant_filter(bgr)
                self.bgr = bgr
                self.mask = mask
                if (bgr is not None) and (iterations > self.START_DELAY):
                    self.stepper.setEngaged(0, True)
                    cv_speed = self.get_groundspeed(bgr)
                    offset = self.find_offset(mask)
                    (est, proj, diff) = self.estimate_error(offset)
                    encoder = self.encoder
                    encoder_rate = self.encoder_rate
                    encoder_rate_prev = self.encoder_rate_prev
                    angle = self.angle
                    output, velocity = self.calculate_output(est, proj, diff,
cv_speed, encoder_rate, encoder_rate_prev)
                    self.encoder_rate_prev = encoder_rate
                    steps = self.set_stepper(output, velocity)

                    # Cleanup for Logging
                    if output == self.OUTPUT_MIN: velocity = -1 * velocity
                    elif output == self.OUTPUT_MAX: velocity = velocity

                    self.estimated = est
                    self.projected = proj
                    b = time.time()
                    hz = (1 / float(b-a))
                    sample = {
                        'offset' : offset,
                        'est' : est,
                        'proj' : proj,
                        'diff' : diff,
                        'angle' : angle,
                        'encoder' : encoder,
                        'encoder_rate' : encoder_rate,
                        'velocity' : velocity,
                        'steps' : steps,
                        'time' : datetime.strftime(datetime.now(),
self.TIME_FORMAT),
                        'hz' : hz,
                        'cv_speed' : cv_speed,
                        'gps_long' : self.gps_longitude,
                        'gps_lat' : self.gps_latitude,
                        'gps_alt' : self.gps_altitude,
                        'gps_speed' : self.gps_speed,
                        'gps_quality' : self.gps_quality,
```

```
                        'gps_satellites' : self.gps_satellites
                    }
                    if self.LOGFILE_ON:
                        self.write_to_log(sample)
                    if hz < 20: print "WARNING! LOW CAMERA FPS!"
                    if self.VERBOSE:
                        self.pretty_print("STEP", "%d Hz\t%d px\t%2.1f
px\t%2.6f,%2.6f DMS" % (hz, est, proj, self.gps_latitude,
self.gps_longitude))
                else:
                    time.sleep(0.01)
            except KeyboardInterrupt as error:
                self.run_threads = False
                self.close()
                break
            except UnboundLocalError as error:
                print "RUN " + str(error)
            except Exception as error:
                print "RUN " + str(error)

    # Information Display Function
    def DisplayDeviceInfo(self):
        self.pretty_print("STEP", "%8s, %30s, %10d, %8d" %
(self.stepper.isAttached(), self.stepper.getDeviceName(),
self.stepper.getSerialNum(), self.stepper.getDeviceVersion()))
        self.pretty_print("STEP", "Number of Motors: %i" %
(self.stepper.getMotorCount()))

    # Event Handler Callback Functions
    def StepperAttached(self, e):
        attached = e.device
        self.pretty_print("STEP", "Stepper %i Attached!" %
(attached.getSerialNum()))

    def StepperDetached(self, e):
        detached = e.device
        self.pretty_print("STEP", "Stepper %i Detached!" %
(detached.getSerialNum()))

    def StepperError(self, e):
        try:
            source = e.device
            self.pretty_print("STEP", "Stepper %i: Phidget Error %i: %s" %
(source.getSerialNum(), e.eCode, e.description))
        except PhidgetException as e:
            self.pretty_print("STEP", "Phidget Exception %i: %s" % (e.code,
e.details))

    def StepperCurrentChanged(self, e):
        source = e.device
        #self.pretty_print("STEP", "Stepper %i: Motor %i -- Current Draw:
%6f" % (source.getSerialNum(), e.index, e.current))

    def StepperInputChanged(self, e):
        source = e.device
        #self.pretty_print("STEP", "Stepper %i: Input %i -- State: %s" %
(source.getSerialNum(), e.index, e.state))
```

```
    def StepperPositionChanged(self, e):
        source = e.device
        #self.pretty_print("STEP", "Stepper %i: Motor %i -- Position: %f" %
(source.getSerialNum(), e.index, e.position))

    def StepperVelocityChanged(self, e):
        source = e.device
        #self.pretty_print("STEP", "Stepper %i: Motor %i -- Velocity: %f" %
(source.getSerialNum(), e.index, e.velocity))

## Main
if __name__ == '__main__':
    session = RowAssist(CONFIG_FILE)
    session.run()
```

APPENDIX B: Arduino Code

```
#define MODE_PIN 13
#define STEERING_PIN A1
#define WHEEL_ANGLE_PIN A2
#define SENSITIVITY_PIN A0
#define BIAS_PIN A3
#define ENCODER_A_PIN 2
#define ENCODER_B_PIN 3
#define BUFFER_LENGTH 128
#define BAUD 9600

int steering_min = 100;
int steering_max = 500;
int steering = 330;
int sensitivity = 330;
int bias = 330;
int angle = 512;
int mode =0;
int mode_counter = 0;
int mode_limit = 0;
char output[BUFFER_LENGTH];
int encoder = 0;

void setup() {
  Serial.begin(BAUD);
  pinMode(MODE_PIN, INPUT);
  digitalWrite(MODE_PIN, HIGH);
  attachInterrupt(0, counter, CHANGE); // set encoder interrupt
  pinMode(STEERING_PIN, INPUT);
  pinMode(SENSITIVITY_PIN, INPUT);
  pinMode(BIAS_PIN, INPUT);
  pinMode(WHEEL ANGLE PIN, INPUT);
}

void loop() {
  if (!digitalRead(MODE_PIN)) {
    if (mode_counter > mode_limit) {
      mode_counter = 0;
      if (mode) {
        mode = 0;
      }
      else {
        mode = 1;
      }
    }
    else {
      mode_counter++;
    }
  }
  else {
    mode_counter = 0;
  }
  steering = analogRead(STEERING_PIN);
  if (steering < steering_min) {
```

```
    steering = -1;
  }
  else if (steering > steering_max) {
    steering = 1;
  }
  else {
    steering = 0;
  }
  sensitivity = analogRead(SENSITIVITY_PIN);
  bias = analogRead(BIAS_PIN);
  angle = analogRead(WHEEL_ANGLE_PIN);
  sprintf(output, "{'mode':%d, 'steering':%d, 'sensitivity':%d, 'bias':%d,
'encoder':%d, 'angle':%d}", mode, steering, sensitivity, bias, encoder,
angle);
  Serial.println(output);
  Serial.flush();
};

void counter(void) {
  if (digitalRead(ENCODER_A_PIN) == HIGH) {
    if (digitalRead(ENCODER_B_PIN) == LOW) {
      encoder++; // CW
    }
    else {
      encoder--; // CCW
    }
  }
  else   // must be a high-to-low edge on channel A
  {
    // check channel B to see which way encoder is turning
    if (digitalRead(ENCODER_B_PIN) == HIGH) {
      encoder++; // CW
    }
    else {
      encoder--; // CCW
    }
  }
}
```

APPENDIX C: Cost (USD) of the guidance system components.

Quantity	Component	Unit Price USD	Amount
1	2 POS Cable	$17.50	$17.50
1	2 POS Panel	$11.40	$11.40
1	Arduino UNO	$25.79	$25.79
1	Ball Grip Positioning Arm Component	$19.41	$19.41
1	Ball Grip Positioning Arm Component	$56.07	$56.07
2	Ball Grip Positioning Arm Component	$16.66	$33.32
1	Ball Grip Positioning Arm Component	$40.86	$40.86
1	Ball Grip Positioning Arm Component	$25.04	$25.04
1	Brackets and Fasteners	$50.00	$50.00
1	Camera	$35.95	$35.95
1	Camera Bracket	$12.95	$12.95
1	Intel Atom CPU	$339.85	$339.85
1	Waterproof Enclosure	$28.91	$28.91
1	ABS Hub Adapter	$5.00	$5.00
1	Joystick	$32.99	$32.99
1	Laser Cutting of Enclosure	$30.00	$30.00
1	Stepper Motor	$44.00	$44.00
1	Stepper Motor Controller	$95.00	$95.00
1	Suction Cups	$53.92	$53.92
2	USB Cables	$15.90	$31.80
1	USB Connector Female	$11.52	$11.52
1	USB Connector Male	$23.94	$23.94
2	USB Panel Mount Connectors	$15.00	$30.00
1	Wiring	$10.00	$10.00
		Total Cost:	**$1,065.22**