Applications of computer-vision for agricultural implement feedback and control

Trevor Stanhope

McGill University

November 22, 2015

Contents

Chapter 1

Introduction

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 2

Evaluation of inter-row cultivator guidance system

Abstract

For organic row crops, frequent cultivation is required for weed control, and these implements require precise guidance systems to assure proper positioning of the working tools. Legacy systems have made use of mechanical guiding rods, but these systems perform poorly in the earliest stages of crop growth. Modern techniques based on RTK GPS are available commercially but are prohibitively expensive for small-scale operations. Therefore, the objective of this study was to develop a low-cost CCD camera system which is capable of supplementing the mechanical row detection during inter-row cultivation. A computer-vision guidance system was developed for the Intel Atom architecture to interface with an electro-hydraulic steering hitch system. Two redundant CCD cameras were mounted to the cultivator toolbar in-line with crop rows to obtain a video stream of the plants passing beneath the implement. The OpenCV platform was used to develop an algorithm for identifying the lateral offset of the plant rows and adjust the hydraulic steering accordingly via PID control. The computer-vision guidance system was tested successfully without GPS RTK assistance at travel speeds of 6, 8, 10, and 12 km/h in corn and soybean fields under varying ambient light and crop conditions.

Figure 2.1: Mechanical guiding rods.

2.1 Introduction

Unlike conventional farming where herbicide application is the primary method for weed prevention, organic farmers must use tillage implements such as inter-row cultivators. Inter-row cultivation is a field operation which requires precise control of the implement in order to maximize the tillage area without causing damage to the crops. Often, cultivation is conducted at speeds of up to 12 km/h with an error tolerance of ± 5 cm. Many organic farmers are not equipped with RTK-guided tillage implements and prefer to use older, mechanically-guided systems. A common method for mechanically-guided cultivators employs guiding rods (also known as brushes) which are mounted to a rotating voltage divider (**??**). These rods make contact with the crop stems and their position approximates that of the crop row. The resulting reference signal is fed to a hydraulic control system which adjusts the steering mechanism of the implement accordingly. Although these mechanically-guided systems are rugged and maintainable, the guiding rods perform poorly with seedlings. During the early stages of growth guiding rods have the potential to cause damage to the crop or lose track of the row, forcing operators to travel at speeds of less than 6 km/h. To address this issue, modern systems implement non-contact detection methods, such as a secondary RTK GPS antennae or cameras, to estimate the lateral offset of the cultivator.

The objective of this study was to develop a low-cost, embedded system which extends computer-vision row detection functionality to existing electro-hydraulic implement guidance systems. To be considered effective, the system should be capable of 4 cm precision 95 % of the time for travel speeds up to 12 km/h and for crops up to 20 cm in height.

2.2 Literature Review

Demand for high precision tractor control has produced significant research interest over the past two decades. Prior studies by Hague et al., Kise et al., and Astrand et al. have demonstrated that by capturing a video stream of the crop rows passing by tractor, the lateral offset of vehicle relative to the crop rows can be estimated with a high degree of consistency. Several different computer-vision methodologies have been proposed for detecting the offset of the crop row: Stereo-vision, distribution of crop foliage, feature-based extraction, In order to identify the location of the crop row, the distribution of plant foliage within the captured images must be determined, a process known as segmentation. During this process, varying ambient light conditions is one of the major limiting factors for successfully implementing computer-vision guidance for weed control. Variations in the ambient color temperature occur naturally with changes in weather and time of day, as well as with the particular camera model being used. Additionally, irregular lighting intensity, also known as non-diffuse lighting, in the camera's field of view, such as from shadows cast by the cultivator, is also a major concern. Non-diffuse lighting results in greater difference in the intensity of pixels and reduces the effectiveness when differentiating between color hues.

Several different image filtering methods can be utilized for segmentation of plants from soil. Using unfiltered RGB data is not ideal due to the high correlation between the three channels. Therefore, conversion to an alternate color-space is necessary for plant identification.

An approach proposed by Meyer et al. (1998) demonstrates a contrast filter known as Excess Green (ExG) calculated using the following equation:

$$\left[\mathbf{X} + \mathrm{a} \;\geq\; \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.1}$$

where nx = image width (pixels); ny = image height (pixels).

The transformed image is passed through a low-pass filter to remove bright spots before a constant threshold value of 20 on a scale of 0 to 255 is applied. This method was tested in diffuse natural light conditions in a greenhouse environment between 11:30am and 12:30pm and resulted in a 100 successful rate of discrimination between plants and soil.

Alternative approaches have utilized HSV transform

After segmentation of the plants within the image, it is necessary to determine the lateral offset of the crop row. Methods for determining the lateral offset can be grouped into two classes based on whether the camera's angle of inclination is either equal to zero, or greater than zero; these classes are referred to as orthogonal or perspective, respectfully.

Orthogonal methods rely on a camera which faces vertically downward and is directly aligned with a single crop row of the cultivator. A basic approach proposed by Olson et al. (1995) for detecting the lateral offset of the crop relies on taking the sum of the pixel elements grey values in the direction of travel. The resulting curve represents the likelihood of the row's position for each x-index within the image. To isolate the most probable offset, two separate methods were compared: 1) a least squares regression of a sinusoidal wave, and 2) a Fourier Fast Transform (FFT) low-pass filter. Both filtration methods were effective in cereals to within an error of 10 mm, but performed poorly on sugar beets due to their characteristically large leaf volume. In a similar study by Slaughter et al. (1995), an algorithm for detecting the lateral offset of the row using individual segmentation of plants in the image was proposed. For each plant, a histogram of the intensities was calculated which was then used to find the median offset of each plant. If a plant's median was significantly different than the other plants in the image, it was considered a

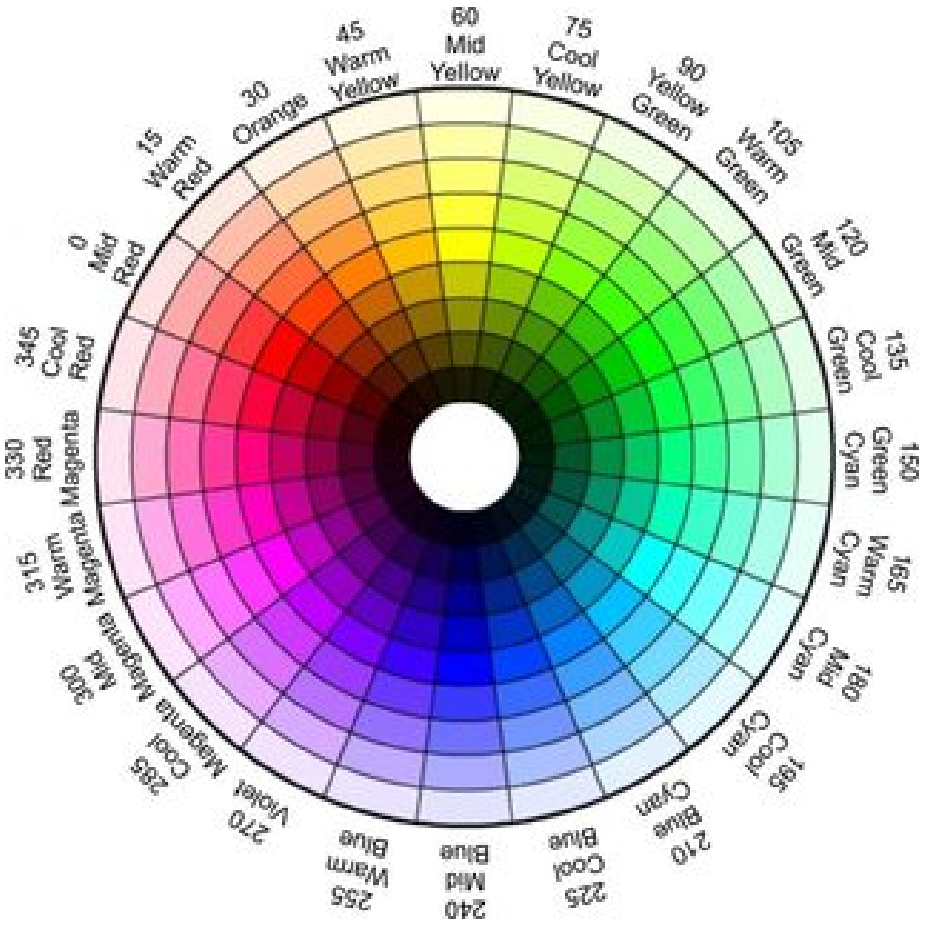

Figure 2.2: HSV Transform

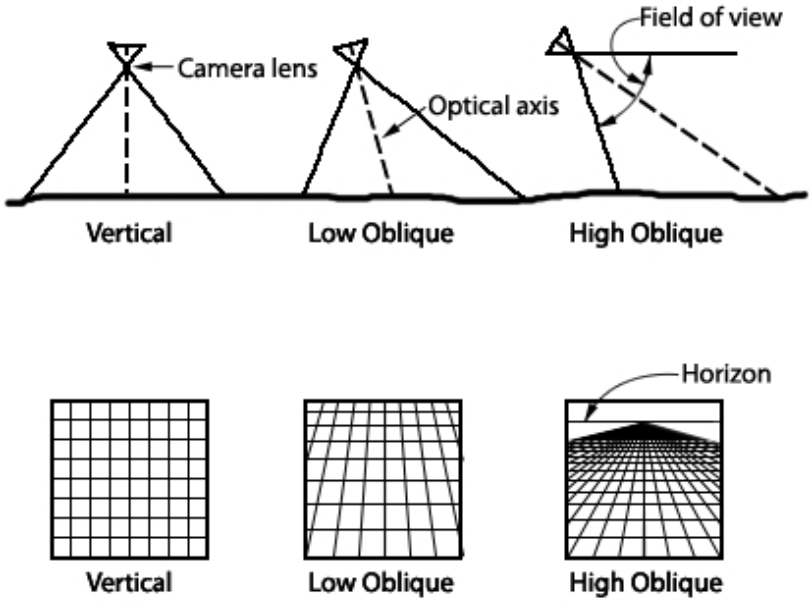

Figure 2.3: Perspective vs. orthogonal viewing angles.

weed and disregarded. The row offset was then calculated based on the medians of the remaining plants. This method was tested on lettuce and tomatoes for use with a band sprayer operating at 8 km/h and performed successfully with standard error of 9 mm and within 12 mm 95th of time.

Conversely, perspective methods rely on a camera with a positive angle of inclination with multiple rows in the field of view. One approach for perspective guidance utilizes the Hough Line Transform (HLT) algorithm to detect linear rows in cauliflower, sugar beets, and wide-spaced wheat. In a study by Pla et al. (1997), a system using HLT performed with an error of 18 mm, which was considered sufficient by the researchers. A similar perspective approach using a band-pass filter proposed by Hague et al. (2001) based on prior knowledge of the spacing of the crop rows was developed for use on cereals and beets. Supported by the British Beet Research Organization the developed system was capable of 3 cm precision at speeds of up to 10 km/h. The project was considered highly successful and was commercialized in 2001 in partnership with Garford Farm Machinery and Robodome Electronics under the name RoboCrop1.

When comparing the orthogonal and perspective methods, both have advantages and disadvantages. The perspective approach is less sensitive to missing plants and high weed density. However, perspective methods rely on prior knowledge of the crop spacing and linear rows with low curvature. Comparatively, orthogonal systems optimize resolution, in pixels-per-centimeter, and require very only basic calibration. Lens distortion is an issue for perspective systems. To compensate for increased subject distance, perspective methods require a higher resolution camera, resulting in greater computational requirements and costs. The reduced field of view for orthogonal systems is a concern when there are significant gaps in the crop rows. To address the issues inherent to the orthogonal approach, a system with two or more cameras may provide sufficient redundancy.

Actuation of the implement position can be achieved by several methods, such as vehicular steering, pivoting hitches, side-shift hitches, or stabilizer steering. Of these, vehicular steering has received tremendous interest however is not applicable in all environments. Some providers have produced mountable steering but no computer-vision versions have been successfully produced. Pivoting hitch systems have also been manufactured. For light cultivation, the side-shift style hitch has been demonstrated to be effective. However, this method causes problems when mounted to a heavy cultivator (deep harrows or coulters) due to "jumping", i.e. the effect of the hitch shifting the tractor. To address this requires either removing tools or dramatically increasing the weight of the tractor, both of which are undesirable to the farmer. Stabilizer steering has applications in heavier tillage.

A hydraulic hitch positional control system can be expressed as a linear system. The output is the lateral error of the vehicle relative to the crop row and the input is the position of the steering mechanism. Additionally, state-information of the system is required, and it is assumed the angular speed is constant and travel speed is within acceptable bounds. To control this system, the voltage signal to the electrohydraulic steering can be modulated to adjust the lateral position of the cultivator.

2.3 Methodology

For field evaluation, a twelve row Hiniker cultivator equipped with a Sukup Auto-Guidance system mounted to a Fendt Vario 850 was used throughout the testing period. The cultivator toolbar was configured to a row-spacing of 30 inches. Steering actuation of the cultivator was achieved via two 75 cm stabilizers mounted 165 cm behind the cultivator toolbar and spaced 147.8 cm apart.

Figure 2.4: Diagram of the cultivator implement mounted to the tractor.

The hydraulic actuation of the stabilizers had maximum angular travel of ± 1.31 rad and angular velocity of 0.4 rad/s.

The computer-vision guidance system was installed alongside the existing mechanical guidance system to act as a replacement for the guiding rod potentiometer. The remaining components of the Sukup Auto-Guide system, including the electrohydraulic controller, center-pivot potentiometer, manual adjustment inputs, and hydraulic solenoids, were unmodified. This configuration allowed easily switching between the two modes of operation during field trials.

Images of the plants passing beneath the cultivator are captured by two weather-proof CCD cameras mounted via C-brackets to he toolbar. In a compromise between the orthogonal and perspective methods, the two cameras were mounted at a 30° inclination from vertical and a subject depth of 100 cm. This approach provides additional longitudinal field of view without significantly contributing to image distortion. To provide row estimation redundancy in the event of regions of high weed density or gaps in the crop rows, the two cameras were installed on the 3rd and 9th row of the cultivator.

An embedded Linux system based on the Debian 7.8 operating system (Linux Kernel 3.2) was developed for the Intel Atom D525 architecture. A 1.8GHz processor was used, with a 32 GB SSD, and 1 GB of RAM (Jetway). A run-time application was developed for the system using the Python programming language (v. 2.7.6) which operated as a local microwebserver. A high-speed database server (MongoDB 32-bit) was implemented for ultra-low latency storage and retrieval of data. This robust platform provided sufficient computing power for real-time image analysis at a relatively low

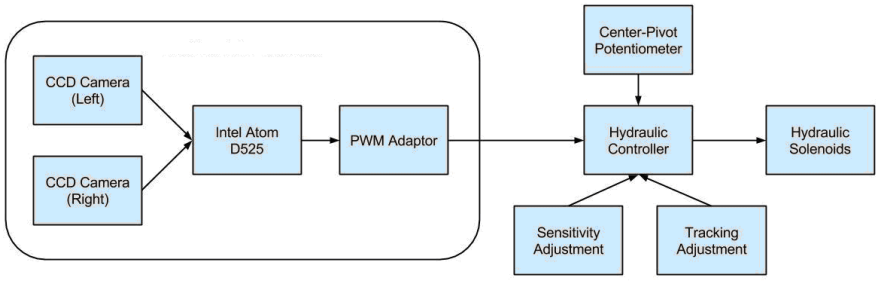

Figure 2.5: Diagram of the computer-vision system integrated with hydraulic controller.

cost.

To generate the output voltage signal of the control system, an ATmega328P microcontroller was used as an 8-bit PWM generator. The microcontroller was integrated as a Universal Serial Bus (USB) peripheral device with the microprocessor as the host.

To aid user operation, a graphical user interface was implemented which allows the operator to check the performance of the guidance system in real-time.

2.3.1 Plant Segmentation

Video of the crop rows were captured in real-time from the two CCD cameras. The cameras used for this study had a native resolution and frame-rate of 640x480 and 25 frames-per-second, respectively. However, were hardware downscaled to a resolution of 320x240 and 16 frames-per-second. After capturing an image, the image matrix was transformed from the Red-Green-Blue (RGB) color-space to the de-correlated Hue-Saturation-Value (HSV) color-space in order to simplify band-pass filtering operations (2.2).

$$\left[\mathbf{X} + \mathrm{a} \;\geq\; \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.2}$$

After transforming the image to the HSV color-space, a band-pass plant detection filter (BPPD) was applied to isolate pixels which could represent plant foliage. This filter selects for pixels with hue ranging from yellow-green to blue-green, saturation above the mean saturation, and value (i.e. brightness) between the extremes of under- and over-exposed. Threshold values were determined empirically using a training set of sample images in varying light and crop conditions. During this process, it was observed that the cameras experienced significant blue-shifting of crop foliage in very bright or low light, so the upper threshold for hue was set well into the cyan-blue region. This change did not have any noticeable negative impact on performance due to the relative lack of blue-tones in soil.

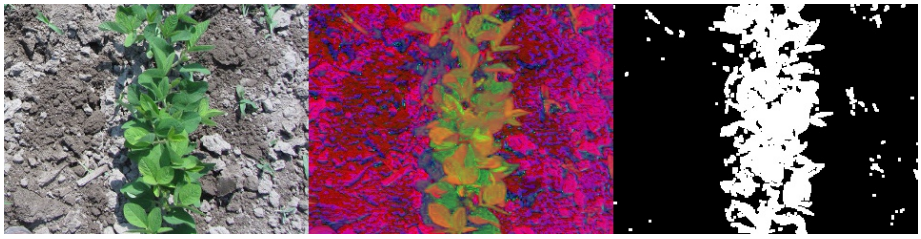

Figure 2.6: BPPD Normal

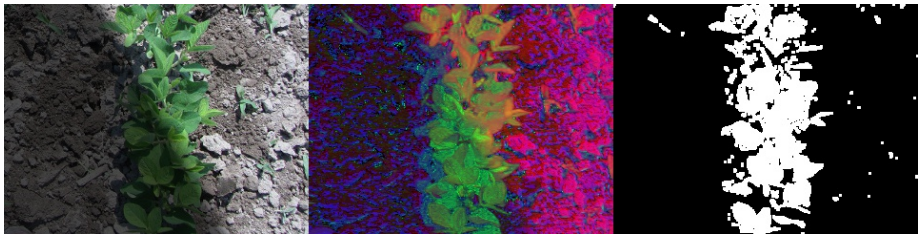

Figure 2.7: BPPD Shadow

$$\left[\mathbf{X} + \mathrm{a} \;\geq\; \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C\right] \tag{2.3}$$

where Hmin=45 lower hue threshold (yellow-green), Hmax=105 upper hue threshold (blue-green).

The BPPD filter utilizes the linear interpolation percentile function to calculate the upper and lower thresholds of the Value and Saturation bands. This approach eliminates the need for static limits, reducing false-positive classification of pixels as green under varying lighting conditions.

As a final post-processing step, morphological opening with a 3 by 3 kernel was applied to the BPPD mask to reduce any remaining noise while preserving the structure of crop foliage:

$$\left[\mathbf{X} + \mathrm{a} \;\geq\; \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C\right] \tag{2.4}$$

where K is a 3 by 3 kernel.

This three step process is computationally non-intensive, yet produces sufficient segmentation in diverse lighting conditions. Notably, the percentile-based band-pass filters of saturation and intensity produced reliable masks in scenarios of extreme exposure and shadows.

2.3.2 Row Estimation

After the plant foliage mask (M) has been produced, the column summation of the mask was calculated in the direction of travel, resulting in an array (C) representing the lateral distribution of plant foliage within the image.

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.5}$$

where H=240

Indices of the C-array with low values suggest bare-soil, moderate values suggest sparsely distributed weeds, and higher values suggest presence of the crop row due to the longitudinal alignment of the plant foliage. Using this distribution, the centroid of the crop row was estimated by applying a high-pass filter to select for indices which are significantly greater than others:

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.6}$$

where α=0.95

The array (p) consists of all indices of the image which are most likely to represent the crop row. The estimated centroid of the crop row was determined by taking the weighted mean of the probable indices, where weight of each index was the normalized value of its corresponding column summation:

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.7}$$

where N is the number of elements in p, and x is position the estimated centroid in pixels.

To compensate for errors in the detection process inherent to single camera systems, the row centroid estimation process was repeated for images, producing two column summation arrays (C1 and C2) and two estimated row centroids (x1 and x2). After calculating the estimated row centroid for each camera, the centroid and column summation values are compared to determine the final estimated offset of the crop row:

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.8}$$

where E is the maximum acceptable error tolerance, W is the width of the camera in pixels.

This approach prioritizes centroid estimations which are in agreeance and in the event of disagreeance between the two cameras the dominant centroid was taken as the row. This provided a simplistic means for reducing errors due to weeds.

For the sake of performance evaluation, the error in pixels may be converted to centimeters using the relationship between the cameras' field-of-view of 64 cm, measured width-wise along the center-line, at a subject depth of 100 cm. For a resolution of 320 px in width, this results in a resolution of 0.2 cm/px.

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.9}$$

where fov is the field-of-view of the camera (in centimeters), W is the camera width (in pixels).

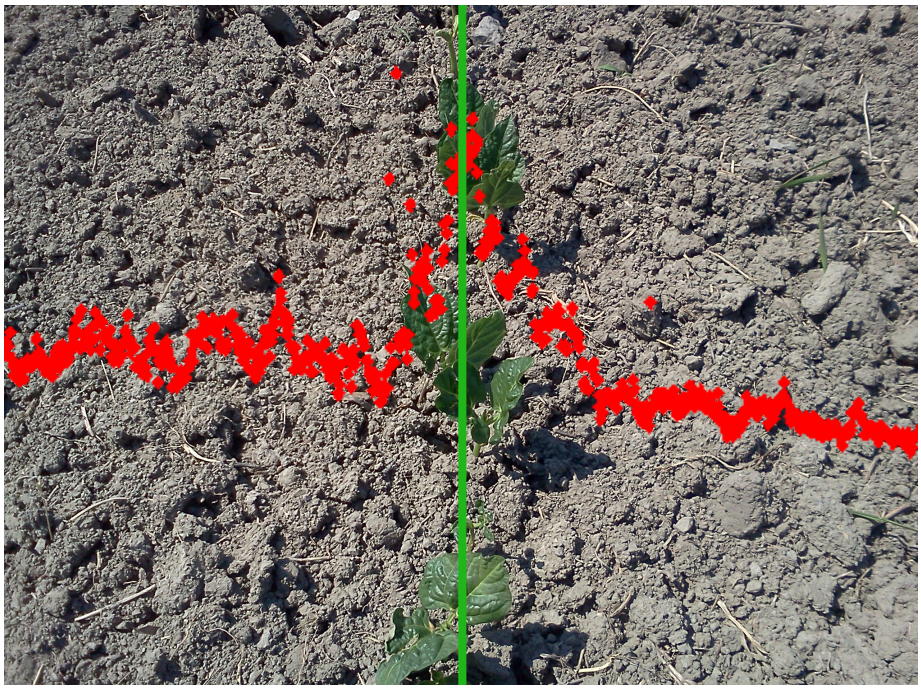

Figure 2.8: Demonstration of offset detection.

Figure 2.9: Rotating stabilizers

2.3.3 Electro-hydraulic Control

The rotary stabilizer mechanism was operated via bang-bang hydraulic solenoid controller whose actuation was based on the voltage differential between the feedback signal from the row detection system, i.e. the guiding rods or computer-vision module, and the potentiometer mounted to the rotary mechanism on the hitch.

To generate a smooth output signal, a signal conditioning based on a Proportional-Integral-Derivative (PID) feed-back controller was implemented. Coefficients were initially chosen to provide similar response to that of the guiding rods and were subsequently modified by trial and error.

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.10}$$

where $K_P = 1.0$, $K_I = 4.0$, $K_D = 0.5$, $N = 15$ is the number of integral samples, $M = 3$ is the number of differential samples.

The output value was transmitted via a weatherproof (IP68) USB connection to the microcontroller which was interfaced with the hydraulic solenoid controller. The microcontroller generates an analog output signal via pulse-width modulation (PWM) to produce a voltage range from 0.0 V to 5.0±0.05 V. However, the operating range of the Sukup Auto-Guide system is from 0.10±0.02V to 8.0±0.02V with a fixed supply voltage of 9.70±0.02V. To account for this discrepancy, the PWM output range was scaled linearly and a logic level converter (LLC) circuit was used to rescale and shift the PWM signal to the required range.

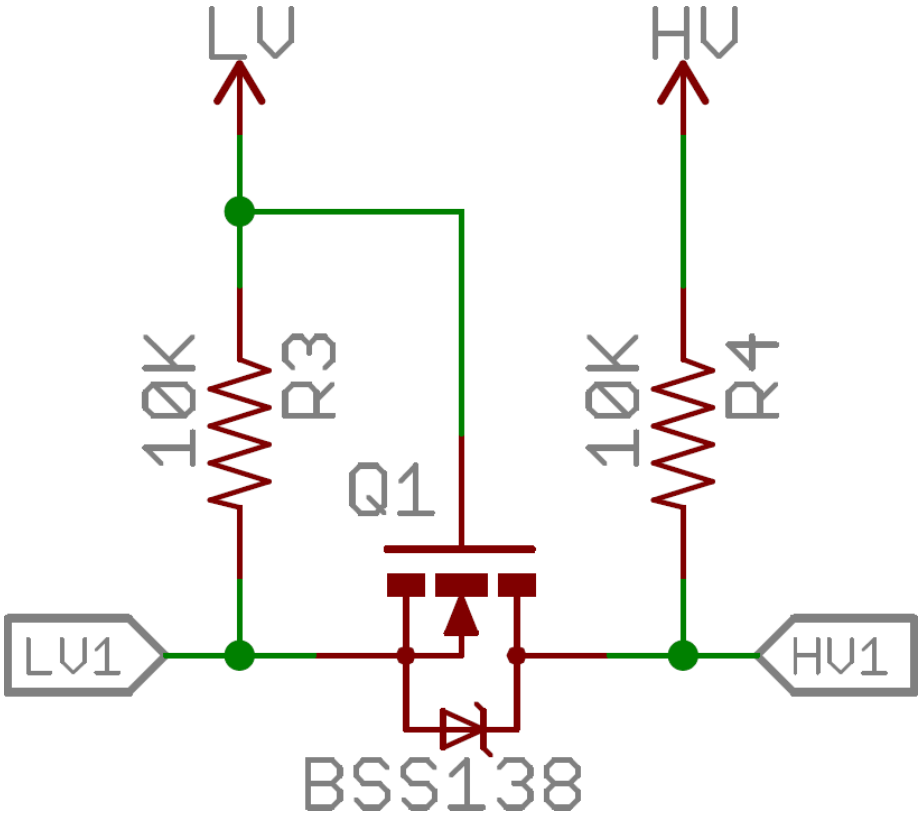

Figure 2.10: Rotating stabilizers

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.11}$$

where $V_{min} = 0.10$, $V_{max} = 8.00$, $V_{HV} = 9.70$, $V_{LV} = 5.0$

To produce this output, a signal condition circuit was employed making use of a MOSFET logici-level converter.

This final output voltage represents the desired set-point for angle of the steering stabilizers to be reached by hydraulic solenoid controller. The mapping between output voltage and position was found to be a linear, second order system, with saturation.

$$\left[\mathbf{X} + \mathrm{a} \ \geq \ \underline{\hat{a}} \sum_{i}^{N} \lim_{x \to k} \delta C \right] \tag{2.12}$$

where K=0.3275 rad/V at a sensitivity setting of 10.

Table 2.1: RMSE and 95th percentile with respect to travel speed.

Team	P	W	D	L	F	A	Pts
Manchester United	6	4	0	2	10	5	12
Celtic	6	3	0	3	8	9	9
Benfica	6	2	1	3	7	8	7
FC Copenhagen	6	2	1	3	5	8	7

2.3.4 Calibration

At the beginning of each day of field trials, a camera calibration procedure was followed to ensure proper alignment of the cameras: (1) the cultivator was aligned with the crop rows which was verified by measurements with a tape measure at the working tools; (2) lateral adjustments were made to the camera bracket to ensure the vertical center-line of each camera was aligned with the crop row; (3) vertical adjustments made to the camera bracket to ensure a subject depth of 100 cm from the camera lens to the soil surface.

Additionally, the Sukup Auto-Guide system provides basic user tuning in the form of sensitivity and tracking adjustment inputs. The sensitivity adjustment effectively changes the mapping between voltage and radial resolution of the stabilizers, with a range of 1 to 10 resulting in mappings from 0.1350 rad/V to 0.3275 rad/V, respectively. Similarly, the tracking adjustment offsets the zero position of the stabilizers on a scale of -3 to +3 corresponding to -0.435 rad to +0.435 rad, respectively. Therefore, at the beginning of each set of trials the following settings were ensured: (1) the sensitivity was set to 10 out of 10, and (2) the tracking adjustment was set to 0.

2.3.5 Trials

Field tests of the system took place over the summer of 2014 from June to August on straight-drilled corn and soybean crops. Only organic cultivars were considered for this study, therefore no pesticides were applied to the fields and weeds were present during testing. All fields used during testing were maintained by Agri-Fusion 2000 Inc., a 4000 ha organic farming operation in St-Polycarpe, Quebec.

Trials were classified into four stages of crop development: $\leq$10 cm, 10 - 15 cm, 15 - 20 cm, and $\geq$20 cm. For each run, five representative locations along the row were selected and the height of the crop from the soil surface was observed using a tape measure, and the resulting average determined the height stage of the trial.

To test the reliability of the two systems at differing speeds of operation, trials were conducted at four approximate travel speeds: 6, 8, 10, and 12 km/h. The travel speed of the tractor was set via the automatic speed controller of the vehicle.

2.4 Results

2.5 Discussion

With respect to RMSE, the two systems performed comparatively, with an average RMSE of less than 3.5 cm for both the computer-vision and the guiding rod systems for all crop stages 3.2.

Table 2.2: RMSE and 95th percentile with respect to crop stage.

Team	P	W	D	L	F	A	Pts
Manchester United	6	4	0	2	10	5	12
Celtic	6	3	0	3	8	9	9
Benfica	6	2	1	3	7	8	7
FC Copenhagen	6	2	1	3	5	8	7

However, with respect to the 95th percentile, usage of the guiding rods resulted in greater error than the computer-vision system for crops at the $\leq$10 cm, 10-15 cm, and 15-20 cm stages. Notably, the guiding rods resulted in an average 95th percentile of 6.57 cm and 6.53 cm for the $\leq$10 cm and 10-15 cm stages, respectively. Comparatively, the computer-vision system had an average 95th percentile of less than 4 cm for all four crop stages. The accuracy of the guiding rods increased dramatically as the plants matured, resulting in a significant decline in average 95th percentile and average RMSE at the 15-20 cm and $\geq$20 cm stages. The computer-vision system showed significantly lower error than the guiding rods for crop stages less than 20 cm in height, but a slight increase in error for plants greater than 20 cm.

There was no apparent correlation between error and travel speed for both systems, either with respect to RMSE or 95th percentile. Notably, the computer vision guidance system outperformed the mechanical system at all travel speeds; however, this affect is likely due to the interaction with crop height, not travel speed.

2.5.1 Future Improvements

Due to the fact that the cameras used for this study were intended for security applications, the cameras were designed for usage in a wide range of ambient lighting. Specifically, the photosensor of the cameras was sensitive to infrared (IR) light and featured a built-in array of 24 IR LEDs. Although this feature theoretically extended the daily hours of operation, IR-sensitivity proved to have a negative affect on color-quality under intense ambient light, i.e. $\geq$50000 Lux, resulting in over-exposure and blue-shifting of green tones in the HSV color-space. To correct this, applying contrast-limited adaptive histogram equalization (CLAHE) as pre-processing prior to the BPPD filter may reduce the negative affects of IR-sensitivity without requiring hardware changes (e.g. non-IR sensitive cameras or usage of an IR filtering lens).

In this study, since row offsets were determined by analyzing the plant foliage mask in the direction of travel with a relatively small subject distance, no perspective or radial (also referred to as lens-distortion) correction was implemented. However for alternate usage cases, e.g. cameras aligned at the mid-point between rows, it would be necessary for perspective and lens-distortion correction to be implemented. Due to the nature of how these systems are installed, a versatile method for in-field camera calibration demonstrated by Lee et al. in 2002 can be utilized. This method makes use of a checkerboard pattern which is placed in the camera's field of view. An image is captured and the positions of the corners are identified. Using the known size of the squares, the calibration coefficients can be calculated which can be used on new images to rectilinearize the image, thus correcting for perspective and radial distortion.

This study focused on a rotating stabilizer hydraulic steering system and therefore the orientation of the crop rows relative to the cultivator (and by extension the cameras) was effectively orthogonal. This approach is appropriate for rotating stabilizer and center-shift steering systems;

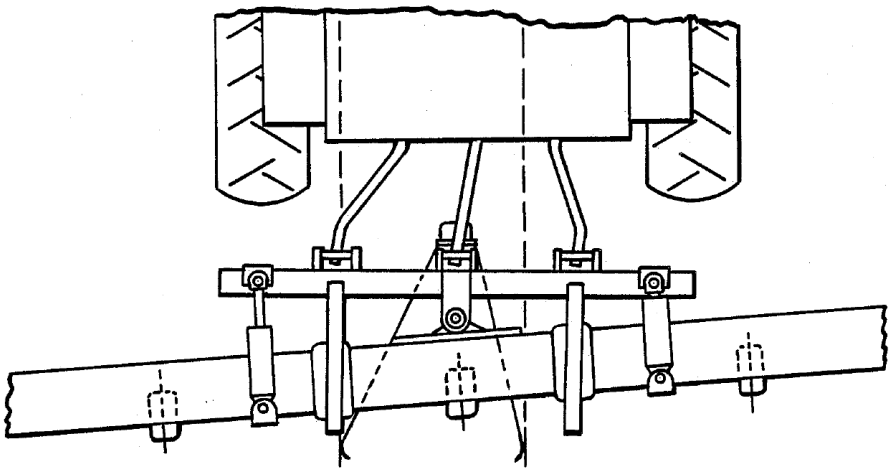

Figure 2.11: Pivoting hitch hydraulic steering system.

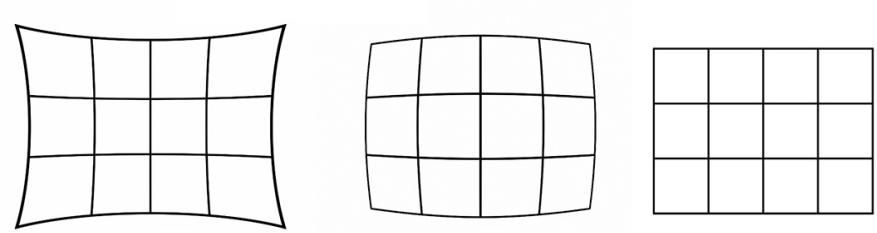

Figure 2.12: Lens distortion

however several commercially available hydraulic steering system designs make use of a pivoting hitch system (2.11).

Such pivoting hitch steering systems may rotate the cultivator toolbar as much as ±5 degrees and would therefore significantly reduce the accuracy of row estimation. To compensate for this effect, knowledge of the camera orientation and its position relative to the pivot point is required. Motion estimation of the cameras using feature-based keypoint tracking may be a robust solution which does not require integration with the hitch controller and is therefore system-agnostic.

2.6 Conclusion

The BPPD method worked well in various lighting conditions and crop conditions. The computer-vision system achieved an average 95th percentile of ¡ 4.0 cm for all crop stages, and significantly outperformed the guiding rods at the earliest stages of ¡10 cm and 10-15 cm. Calibration procedure was simple

2.6.1 Acknowledgements

This study was supported in part by funding provided through the National Science and Engineering Research of Canada (NSERC) Engage program with assistance from Jean Cantin of the Quebec Ministry of Agriculture, Fisheries and Food.

Chapter 3

Comparison of feature-based motion estimation algorithms

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.2 Literature Review

Emerging applications for computer-vision on agricultural implements Row-crop cultivation Strip tillage Post-harvest spraying Motion feedback is very useful for such applications Orientation Responsiveness Incorporating speed detection into these systems can be problematic RTK GPS can be expensive Low-cost GPS units are less reliable Fifth-wheels can suffer from slippage

3.3 Methodology

3.3.1 Camera Model

An outdoor (dust-tight, water-tight) CMOS camera with 640x480 pixel resolution was mounted to the front of a New Holland T5050 (New Holland Agriculture, Turin, Italy) tractor in-line with a crop row to obtain a video stream of the plants passing beneath the equipment. The camera lens had an aperture of 2.4, a 6 mm focal point, and a 4:3 aspect ratio, which translated to a $43^{i}rc$ horizontal field of view and $32.9^{c}irc$ vertical field of view. For the specifications, the camera provided approximately 1.15 mm/px resolution when mounted 1.0 m above the ground and oriented orthogonally to the testing surface. In order to maximize the overlap between consecutive images at higher travel speeds, the camera was oriented width-wise in the direction of travel. The camera sensor was capable of automatically adjusting white-balance for varying lighting conditions and could capture images at a rate of approximately 25 frames per second. Additionally, the CMOS sensor of the camera was not IR-filtered and the camera enclosure was equipped with 24 infrared LEDs. This allowed the device to be used effectively in a wide range of ambient lighting conditions and was rated to a minimum illumination of 0.5 Lux.

In order to determine the ground speed of the vehicle using computer-vision, an application was developed with the Python OpenCV platform using the SURF (Speeded Up Robust Features) keypoint matching algorithm (Bay, 2006). The application was developed to run on an Intel i7 powered VMC3501-K (Nexcomm, Taiwan). A Trimble AgGPS RTK receiver position directly above the camera was used to obtain geographic longitude and latitude, time, and GNSS signal quality for each data point produced by the computer-vision algorithm. To account for variance in the subject depth, the distance from the camera to the surface, an ultrasonic sensor with 1 cm accuracy was mounted at the same vertical height as the camera.

Using the checkboard pattern, the lens and projection distortion was corrected.

Figure 3.1: Camera Bracket

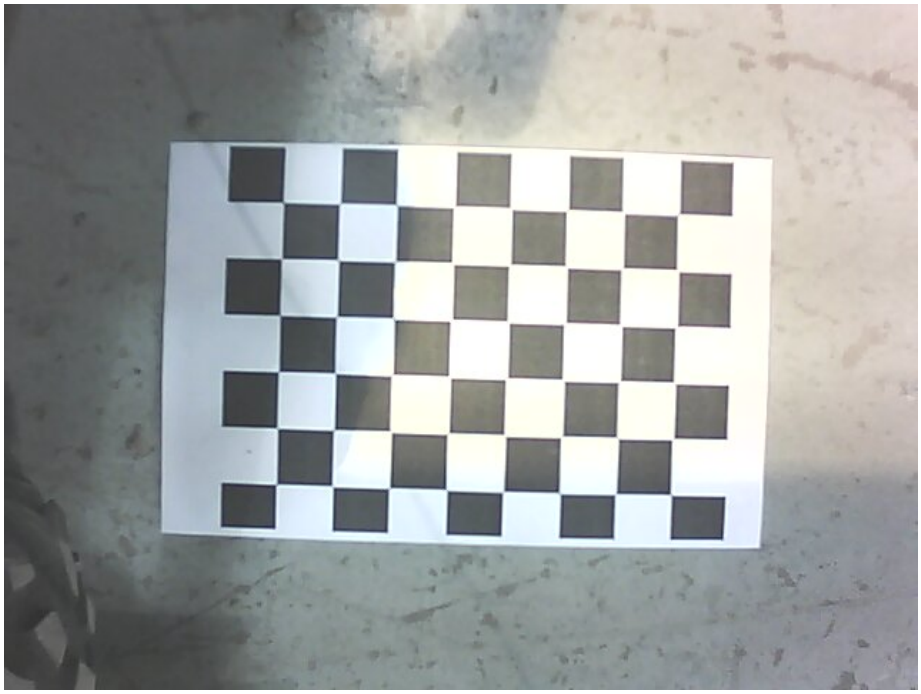

Figure 3.2: Checkerboard camera calibration

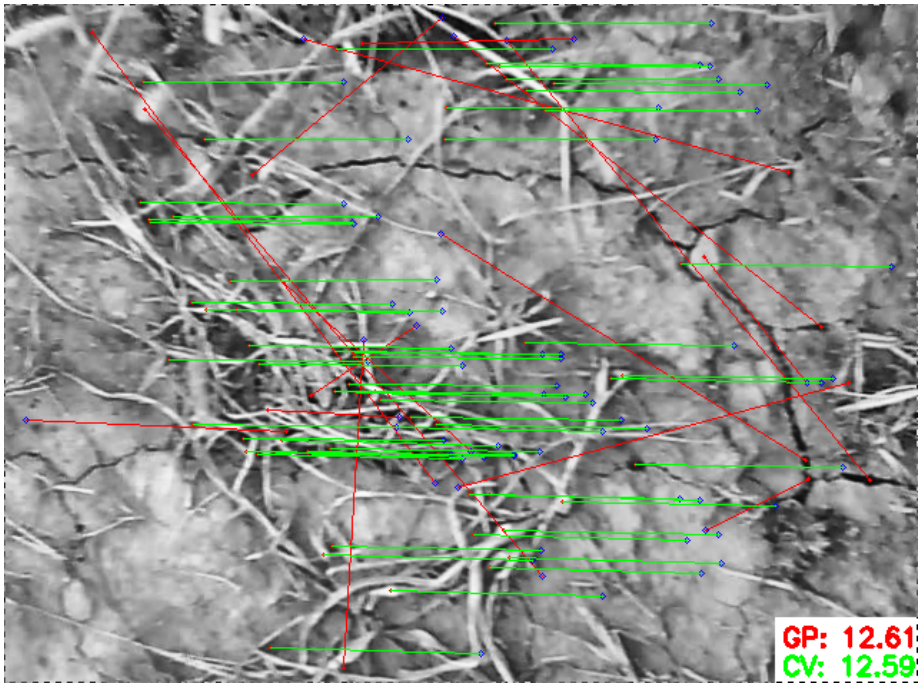

Figure 3.3: Keypoint tracking.

3.3.2 Image Processing

To determine the ground speed of the tractor using computer vision, the proposed algorithm captures two consecutive RGB images from the camera, M1 and M2, at times t1 and t2, respectively. Each image matrix is then transformed to grayscale by the following formula:

Contrast-limited adaptive histogram equalization (CLAHE) was used to optimize the gray-scale images. If any histogram bin exceeds the specified contrast limit of 40, those pixels are clipped and distributed uniformly to other bins before applying histogram equalization. After equalization, to remove artifacts in tile borders, bilinear interpolation is applied.

3.3.3 Keypoint Matching

Both sets of descriptors, D1 and D2, are then matched using k-Nearest Neighbors brute-force matching, where each of the keypoint descriptors of M1 are compared to those of M2. For the regression, two best matches were found for each query descriptor. This process results in a set consisting of pairs of possible keypoint matches and the corresponding error terms:

To filter the array of possible matches for good matches, for each pair of possible matches the error of each match is compared to the other. If the match has a first error term (eA) which is less than half of the second error term (eB), then the point is considered a good match and is stored to set Q.

For each good match in set Q, the key point's coordinates in M1 and M2, i.e. (x1,y1) and (x1,y1), are projected from pixel space to real-space with the following transformation:

$$f = 2.0*tan(\alpha/2.0)\lambda = W/fK = 0.95r = sqrt(X^2+Y^2)Y = y*K*(1+c1*r^2+c2*r^4+c3*r^6)X = x*K*(1+c1*r^2+c2*r^4+ \tag{3.1}$$

Where α is the horizontal field-of-view (0.75 radians), W is the image width (640 px), S is the subject depth (1000 mm), and ρ is the pitch (0.0 radians).

The velocity of each keypoint between the two consecutive images was calculated by taking the hypotenuse between the coordinates in M1 and M2 divided by the rate of image capture (f):

$$v = f * sqrt((X2 - X1)^2 + (Y2 - Y1)^2)\theta = arctan2((X2 - X1), (Y2 - Y1)) \tag{3.2}$$

The set of vectors was filtered by clustering by histogram and selecting for dominant vector angle within a tolerance of 0.05 radians. Consistently removes stationary objects and false pairs without being computationally intensive. Additionally, since the dominant vector angle is used, despite its simplicity this approach is orientation-agnostic.

3.3.4 Field Trials

To ensure precise estimation of speed, the following procedures were conducted prior to testing: Ensure orientation of camera within 1 degree. Ensure pitch of camera within 1 degree Ensure height of camera at 1000 mm to concrete surface Six (6) surface types Pavement Gravel Soil / Residue Turf Grass Hay Grass Mature Soy For each surface type, five (5) videos were collected 0 km/h to 18 km/h to 0 km/h in 45 seconds

3.4 Results

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Table 3.1: RMSE and 95^{th} Percentile with respect to crop-stage (cm).

	RMSE (cm)				95^{th} Percentile (cm)			
	≤10	10-15	15-20	≥20	≤10	10-15	15-20	≥20
Computer-vision	0.50 ± 6.43	19.36 ± 8.11	63.69 ± 15.01	65.67 ± 16.49	38.86 ± 0.23	38.9 ± 0.29	145.55 ± 15.61	143.69 ± 16.17
Guiding rods:	21.33 ± 6.78	20.92 ± 7.97	66.33 ± 16.07	66.15 ± 16.71	38.39 ± 0.81	38.35 ± 0.79	133.34 ± 18.60	131.83 ± 15.94

3.5 Discussion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget,

Table 3.2: RMSE and 95^{th} Percentile with respect to travel speed (km/h).

	RMSE (cm)				95^{th} Percentile (cm)			
	6	8	10	12	6	8	10	12
Computer-vision	0.50 ± 6.43	19.36 ± 8.11	63.69 ± 15.01	65.67 ± 16.49	38.86 ± 0.23	38.9 ± 0.29	145.55 ± 15.61	143.69 ± 16.17
Guiding rods:	21.33 ± 6.78	20.92 ± 7.97	66.33 ± 16.07	66.15 ± 16.71	38.39 ± 0.81	38.35 ± 0.79	133.34 ± 18.60	131.83 ± 15.94

consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.5.1 Applications

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.6 Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Chapter 4

Conclusion

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Appendix A

Python Application

```
#!/usr/bin/env python
"""
Agri-Vision
Precision Agriculture and Sensor Systems Group (PASS)
McGill University, Department of Bioresource Engineering
"""

__author__ = 'Trevor Stanhope'
__version__ = '3.02'

## Libraries
import cv2, cv
import serial
import pymongo
from bson import json_util
from pymongo import MongoClient
import json
import numpy as np
from matplotlib import pyplot as plt
import thread
import gps
import time
import sys
from datetime import datetime
import ast
import os
import threading
import cherrypy
from cherrypy.process.plugins import Monitor
from cherrypy import tools

## Constants
try:
    CONFIG_FILE = '%s' % sys.argv[1]
```

```
except Exception as err:
    cwd = os.path.dirname(os.path.realpath(__file__))
    config_path = os.path.join(cwd, 'settings.cfg')
    CONFIG_FILE = open(config_path).read().rstrip()

# External Functions
def normalize_by_histogram(gray):
    """
    Normalize gray-scale by histogram
    """
    hist, bins = np.histogram(gray.flatten(), 256, [0,256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max()/ cdf.max()
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
    cdf = np.ma.filled(cdf_m, 0).astype('uint8')
    gray_norm = cdf[gray] # Now we have the look-up table
    return gray_norm
def is_number(s):
    try:
        float(s)
        return True
    except ValueError:
        return False

## Class
class app:
    def __init__(self, config_file):

        # Path
        self.CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))

        # Load Config
        if config_file is None:
            self.default_settings()
        else:
            self.load_settings(config_file)

        # Initializers
        self.init_log() # it's best to run the log first to catch all
            events
        self.init_cameras()
        self.init_controller()
        self.init_pid()
        self.init_db()
        self.init_gps()
        self.init_display()
        self.init_webapp()

        # Thread states
```

```
84          self.running = False
85          self.updating = False
86
87      # Initialize Webapp
88      def init_webapp(self):
89          self.log_msg('HTTP', 'Initializing Webapp Tasks')
90          try:
91              self.run_task = Monitor(cherrypy.engine, self.run, frequency
                    =1/float(self.config['CAMERA_FPS'])).subscribe()
92              if self.config['GPS_ENABLED']: self.gps_task = Monitor(
                    cherrypy.engine, self.update_gps, frequency=1/float(self.
                    config['GPS_HZ'])).subscribe()
93              self.cameras_task = Monitor(cherrypy.engine, self.
                    check_cameras, frequency=5).subscribe()
94              self.controller_task = Monitor(cherrypy.engine, self.
                    check_controller, frequency=5).subscribe()
95              self.save_task =  Monitor(cherrypy.engine, self.save_image,
                    frequency=1).subscribe()
96          except Exception as error:
97              self.log_msg('ENGINE', 'Error: %s' % str(error), important=
                    True)
98
99      # Initialize Camera(s)
100     def init_cameras(self):
101
102         # Setting variables
103         self.log_msg('CAM', 'Initializing CV Variables')
104         self.log_msg('CAM', 'Camera Width: %d px' % self.config['
                CAMERA_WIDTH'])
105         self.log_msg('CAM', 'Camera Height: %d px' % self.config['
                CAMERA_HEIGHT'])
106         self.log_msg('CAM', 'Camera Depth: %d cm' % self.config['
                CAMERA_DEPTH'])
107         self.log_msg('CAM', 'Camera FOV: %f cm' % self.config['CAMERA_FOV'
                ])
108         self.log_msg('CAM', 'Error Tolerance: +/- %1.1f cm' % self.config[
                'ERROR_TOLERANCE'])
109
110         # Attempt to set each camera index/name
111         self.log_msg('CAM', 'Initializing Cameras')
112         self.cameras = []
113         self.images = []
114         for i in range(self.config['CAMERAS']):
115             try:
116                 cam = self.attach_camera(i) # returns None if camera
                        failed to attach
117                 self.cameras.append(cam)
118                 self.images.append(np.zeros((self.config['CAMERA_HEIGHT'],
                        self.config['CAMERA_WIDTH'], 3), np.uint8))
119                 self.log_msg('CAM', 'OK: Camera #%d connected' % i)
```

```
120             except Exception as error:
121                 self.log_msg('CAM', 'ERROR: %s' % str(error), important=
                        True)
122
123     ## Attach a camera capture
124     def attach_camera(self, i):
125         try:
126             self.log_msg('CAM', 'WARN: Attaching Camera #%d' % i)
127             cam = cv2.VideoCapture(i)
128             cam.set(cv.CV_CAP_PROP_FRAME_WIDTH, self.config['CAMERA_WIDTH'
                    ])
129             cam.set(cv.CV_CAP_PROP_FRAME_HEIGHT, self.config['
                    CAMERA_HEIGHT'])
130             cam.set(cv.CV_CAP_PROP_SATURATION, self.config['
                    CAMERA_SATURATION'])
131             cam.set(cv.CV_CAP_PROP_BRIGHTNESS, self.config['
                    CAMERA_BRIGHTNESS'])
132             cam.set(cv.CV_CAP_PROP_CONTRAST, self.config['CAMERA_CONTRAST'
                    ])
133             cam.set(cv.CV_CAP_PROP_FPS, self.config['CAMERA_FPS'])
134             if cam.isOpened():
135                 self.log_msg('CAM', 'OK: Camera successfully opened')
136                 return cam
137             else:
138                 self.log_msg('CAM', 'ERROR: Failed to attach camera!')
139                 cam.release()
140                 return None
141         except:
142             self.log_msg("CAM", "Failed to create camera object!",
                    important=True)
143             return None
144
145     # Initialize Database
146     def init_db(self):
147         self.LOG_NAME = datetime.strftime(datetime.now(), self.config['
                LOG_FORMAT'])
148         self.MONGO_NAME = datetime.strftime(datetime.now(), self.config['
                MONGO_FORMAT'])
149         self.log_msg('DB', 'Initializing MongoDB')
150         self.log_msg('DB', 'Connecting to MongoDB: %s' % self.MONGO_NAME)
151         self.log_msg('DB', 'New session: %s' % self.LOG_NAME)
152         try:
153             self.client = MongoClient()
154             self.database = self.client[self.MONGO_NAME]
155             self.collection = self.database[self.LOG_NAME]
156             self.log_msg('DB', 'Setup OK')
157         except Exception as error:
158             self.log_msg('DB', 'ERROR: %s' % str(error), important=True)
159
160     # Initialize PID Controller
```

```
161     def init_pid(self):
162         self.log_msg('PID', 'Initialing Electro-Hydraulics')
163         try:
164             self.log_msg('PID', 'N Samples: : %s' % self.config['N_SAMPLES
                ')
165             self.log_msg('PID', 'P Coefficient: : %s' % self.config['
                P_COEF'])
166             self.log_msg('PID', 'I Coefficient: : %s' % self.config['
                I_COEF'])
167             self.log_msg('PID', 'D Coefficient: : %s' % self.config['
                D_COEF'])
168             self.offset_history = [ 0.0 ] * int(self.config['N_SAMPLES'])
169             self.log_msg('PID', 'Setup OK')
170         except Exception as error:
171             self.log_msg('PID', 'ERROR: %s' % str(error), important=True)
172 
173     # Initialize Log
174     def init_log(self):
175         try:
176             self.LOG_NAME = datetime.strftime(datetime.now(), self.config[
                'LOG_FORMAT'])
177             error_log_path = os.path.join(self.CURRENT_DIR, self.config['
                LOG_DIR'], self.config['LOG_MESSAGES'])
178             self.error_log = open(error_log_path, 'w')
179             self.log_msg('LOG', 'Message Setup OK')
180             if self.config['DATALOG_ON']:
181                 data_log_path = os.path.join(self.CURRENT_DIR, self.config
                    ['LOG_DIR'], self.LOG_NAME)
182                 self.data_log = open(data_log_path, 'w')
183                 headers = ['time', 'estimated', 'sig', 'pwm', 'volts']
184                 self.data_log.write(','.join(headers) + '\n')
185             self.log_msg('LOG', 'Datalog Setup OK')
186         except Exception as error:
187             self.log_msg('ERROR', str(error), important=True)
188 
189     # Initialize Controller
190     def init_controller(self):
191         self.log_msg('CTRL', 'Initializing controller ...')
192         self.calibrating = False
193         try:
194             self.log_msg('CTRL', 'Device: %s' % self.config['SERIAL_DEVICE
                ')
195             self.log_msg('CTRL', 'Baud Rate: %d' % self.config['
                SERIAL_BAUD'])
196             self.log_msg('CTRL', 'PWM Resolution: %d' % self.config['
                PWM_RESOLUTION'])
197             self.log_msg('CTRL', 'Voltage: %d' % self.config['PWM_VOLTAGE'
                ])
198             self.controller = self.attach_controller()
199             if self.controller is None:
```

```
200                 self.log_msg('CTRL', 'ERROR: Could not locate a controller
                        !')
201             else:
202                 self.log_msg('CTRL', 'Setup Succesful')
203         except Exception as error:
204             self.log_msg('CTRL', 'ERROR: %s' % str(error), important=True)
205             self.controller = None
206 
207     ## Attach Controller
208     def attach_controller(self, attempts=5):
209         try:
210             self.log_msg('CTRL', 'Attaching controller ...')
211             for dev in self.config['SERIAL_DEVICE']:
212                 for i in range(attempts):
213                     dev_num = dev + str(i)
214                     self.log_msg('CTRL', 'Trying: %s' % dev_num)
215                     try:
216                         ctrl = serial.Serial(dev_num, self.config['
                                SERIAL_BAUD'])
217                         return ctrl
218                     except Exception as err:
219                         self.log_msg('CTRL', 'WARN: %s' % str(err),
                                important=True)
220             return None
221         except Exception as error:
222             raise Exception("Serial connection failed!", important=True)
223 
224     # Initialize GPS
225     def init_gps(self):
226         self.log_msg('GPS', 'Initializing GPS ...')
227         self.latitude = 0
228         self.longitude = 0
229         self.speed = 0
230         try:
231             self.log_msg('GPS', 'Enabing GPS ...')
232             self.gpsd = gps.gps()
233             self.gpsd.stream(gps.WATCH_ENABLE)
234         except Exception as err:
235             self.gpsd = None
236             self.log_msg('GPS', 'WARNING: GPS not available! %s' % str(err
                    ), important=True)
237 
238     # Display
239     def init_display(self):
240         self.log_msg('DISP', 'Initializing Display')
241         try:
242             self.output_image = np.zeros((self.config['CAMERA_HEIGHT'],
                    self.config['CAMERA_WIDTH'], 3), np.uint8)
243             if not self.config['DISPLAY_ON']:
244                 self.log_msg('DISP', 'WARN: Display disabled!')
```

```
245             except Exception as error:
246                 self.log_msg('DISP', 'ERROR: %s' % str(error), important=True)
247 
248     ## Rotate image
249     def rotate_image(self, bgr):
250         bgr = cv2.rotate(bgr)
251         return bgr
252 
253     ## Check Cameras
254     def check_cameras(self):
255         """ Checks cameras for functioning streams """
256         if self.config['VERBOSE']: self.log_msg('CAM', 'Checking Cameras
                ...')
257         for i in range(self.config['CAMERAS']):
258             if self.cameras[i] is None:
259                 self.cameras[i] = self.attach_camera(i)
260             elif not self.cameras[i].isOpened():
261                 self.log_msg('CAM', 'Camera #%d is offline' % i)
262                 self.cameras[i] = None
263             else:
264                 self.log_msg('CAM', 'Camera #%d is okay' % i)
265 
266     ## Capture images from multiple cameras #!TODO: fix halting
267     def capture_images(self):
268         a = time.time()
269         if self.config['VERBOSE']: self.log_msg('CAM', 'Capturing Images
                ...')
270         images = [None] * self.config['CAMERAS']
271         for i in range(self.config['CAMERAS']):
272             self.log_msg('CAM', 'Attempting on Camera #%d' % i)
273             try:
274                 cam = self.cameras[i]
275                 if cam is not None:
276                     (s, bgr) = self.cameras[i].read()
277                     if s:
278                         if np.all(bgr==self.images[i]): # Check to see if
                            frame is frozen
279                             self.log_msg('CAM', 'WARN: Frozen frame')
280                             self.cameras[i].release()
281                         else:
282                             self.log_msg('CAM', 'Capture successful: %s' %
                                    str(bgr.shape))
283                             images[i] = bgr
284                     else:
285                         self.log_msg('CAM', 'WARN: Capture failed')
286                         self.cameras[i].release()
287                 else:
288                     pass
289             except:
290                 pass
```

```
291         if all(v is None for v in images): raise Exception("No images
                captured!", important=True)
292         self.images = images
293         b = time.time()
294         if self.config['VERBOSE']: self.log_msg('CAM', '... %.2f ms' % ((b
                - a) * 1000))
295         return images
296
297
298     ## Plant Segmentation Filter
299     def bppd_filter(self, images):
300         """
301         RBG --> HSV
302         Set saturation/value equal to BPPD(S, V)
303         Take hues within range from green-yellow to green-blue
304         """
305         if self.config['VERBOSE']: self.log_msg('BPPD', 'Filtering for
                plants ...')
306         if images == []: raise Exception("No input image(s)!", important=
                True)
307         a = time.time()
308         masks = []
309         threshold_min = np.array([self.config['HUE_MIN'], self.config['
                SAT_MIN'], self.config['VAL_MIN']], np.uint8)
310         threshold_max = np.array([self.config['HUE_MAX'], self.config['
                SAT_MAX'], self.config['VAL_MAX']], np.uint8)
311         for bgr in images:
312             if bgr is not None:
313                 try:
314                     hsv = cv2.cvtColor(bgr, cv2.COLOR_BGR2HSV)
315                     threshold_min[1] = np.percentile(hsv[:,:,1], 100 *
                            self.config['SAT_MIN'] / 255.0)
316                     threshold_min[2] = np.percentile(hsv[:,:,2], 100 *
                            self.config['VAL_MIN'] / 255.0)
317                     threshold_max[1] = np.percentile(hsv[:,:,1], 100 *
                            self.config['SAT_MAX'] / 255.0)
318                     threshold_max[2] = np.percentile(hsv[:,:,2], 100 *
                            self.config['VAL_MAX'] / 255.0)
319                     mask = cv2.inRange(hsv, threshold_min, threshold_max)
320                     mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, np.ones
                            ((self.config['KERNEL_X'],self.config['KERNEL_Y'])
                            , np.uint8))
321                     masks.append(mask)
322                     self.log_msg('BPPD', 'Mask #%d was successful' % len(
                            masks))
323                 except Exception as error:
324                     self.log_msg('BPPD', str(error), important=True)
325             else:
326                 self.log_msg('BPPD', 'WARN: Mask #%d is blank' % len(masks
                    ), important=True)
```

```
327                 masks.append(None)
328         b = time.time()
329         if self.config['VERBOSE']: self.log_msg('BPPD', '... %.2f ms' % ((
                b - a) * 1000))
330         return masks
331 
332     ## Find Plants
333     def find_offset(self, masks):
334         """
335         Finds indicies which are possible centroids
336         Note: Repeated for each mask
337         """
338         if self.config['VERBOSE']: self.log_msg('BPPD', 'Finding offset of
                row ...')
339         if masks == []: raise Exception("No input mask(s)!", important=
                True)
340         a = time.time()
341         offsets = []
342         sums = []
343         for mask in masks:
344             if mask is not None:
345                 try:
346                     colsum = mask.sum(axis=0) # vertical summation
347                     T = np.percentile(colsum, self.config['
                        THRESHOLD_PERCENTILE'])
348                     probable = np.nonzero(colsum >= T) # returns 1 length
                        tuble
349                     num_probable = len(probable[0])
350                     best = int(np.median(probable[0]))
351                     s = colsum[best]
352                     c = best - self.config['CAMERA_WIDTH'] / 2
353                     offsets.append(c)
354                     sums.append(s)
355                 except Exception as error:
356                     self.log_msg('OFF', '%s' % str(error), important=True)
357         b = time.time()
358         if self.config['VERBOSE']: self.log_msg('OFF', '... %.2f ms' % ((b
                - a) * 1000))
359         return offsets, sums
360 
361     ## Best Guess for row based on multiple offsets from indices
362     def estimate_row(self, indices, sums, tol=32):
363         """
364         If within tolerance, check error between indices from both cameras
365         If similar, take mean, else take dominant estimation
366         1. Estimate the weighted position of the crop row (in pixels)
367         """
368         if self.config['VERBOSE']:self.log_msg('ROW', 'Smoothing offset
                estimation ...')
369         if sums == []: raise Exception("No input sum(s)!", important=True)
```

```
370         if indices == []: raise Exception("No input indice(s)", important=
                True)
371         if len(sums) != len(indices): raise Exception("len(x) must equal
                len(y)", important=True)
372         a = time.time()
373         try:
374             e = np.abs(np.mean(np.diff(indices)))
375             sums = np.array(sums)
376             if e > self.config['ERROR_TOLERANCE'] * self.config['
                    CAMERA_WIDTH'] / self.config['CAMERA_FOV']:
377                 indices = np.array(indices)
378                 cur =  indices[np.argmax(sums)]
379             else:
380                 cur = np.mean(indices)
381         except Exception as error:
382             self.log_msg('ROW', 'ERROR: %s' % str(error), important=True)
383             cur = self.config['CAMERA_WIDTH'] / 2
384 
385         # Insert current into local memory
386         self.offset_history.append(cur)
387         lim = int(np.ceil(self.config['N_SAMPLES'] / float(self.config['
                AGGRESSIVENESS'])))
388         while len(self.offset_history) > lim:
389             self.offset_history.pop(0)
390 
391         # Smooth
392         n = int(np.ceil(self.config['N_SAMPLES'] / float(self.config['
                AGGRESSIVENESS'])))
393         m = int(np.ceil(self.config['M_SAMPLES'] / float(self.config['
                AGGRESSIVENESS'])))
394         est =  int(self.offset_history[-1])
395         avg = int(np.mean(self.offset_history[-n:]))
396         diff = int(np.mean(np.diff(self.offset_history[-m:])))
397         b = time.time()
398         if self.config['VERBOSE']: self.log_msg('ROW', '... %.2f ms' % ((b
                - a) * 1000))
399         return est, avg, diff
400 
401     ## Control Hydraulics
402     def calculate_output(self, est, avg, diff):
403         """
404         Calculates the PID output for the PWM controller
405         Arguments: est, avg, diff
406         Requires: PWM_VOLTAGE, PWM_RESOLUTION, MAX_VOLTAGE, MIN_VOLTAGE
407         Returns: PWM signal
408         """
409         if self.config['VERBOSE']: self.log_msg('PID', 'Calculating PID
                Output ...')
410         a = time.time()
411         try:
```

```
412             p = est * self.config['P_COEF']
413             i = avg * self.config['I_COEF']
414             d = diff  * self.config['D_COEF']
415             sig = self.config['SENSITIVITY'] * (p + i + d) - self.config['
                    CAMERA_OFFSET'] # scale and offset signal
416             self.log_msg('PID', "sig = %.1f + %.1f + %.1f" % (p,i,d))
417         except Exception as error:
418             self.log_msg('PID', 'ERROR: %s' % str(error))
419             sig = 0
420         b = time.time()
421         if self.config['VERBOSE']: self.log_msg('PID', '... %.2f ms' % ((b
                - a) * 1000))
422         return sig
423
424     ## To Volts
425     def to_volts(self, pwm):
426         volts_at_ctrl = (pwm * self.config['PWM_VOLTAGE']) / float(self.
                config['PWM_RESOLUTION'])
427         volts = round(np.interp(volts_at_ctrl, [0, self.config['
                PWM_VOLTAGE']], [0, self.config['SUPPLY_VOLTAGE']]) / 1000.0,
                2)
428         self.log_msg('PID', 'PWM = %d (%.2f V)' % (pwm, volts))
429         return volts
430
431     ## Control Hydraulics
432     def set_controller(self, sig):
433         """
434         1. Get PWM response corresponding to average offset
435         2. Send PWM response over serial to controller
436         """
437         a = time.time()
438         if self.config['VERBOSE']: self.log_msg('CTRL', 'Setting
                controller state ...')
439         try:
440             if self.controller is None: raise Exception("No controller!")
441             v_zero = (self.config['MAX_VOLTAGE'] + self.config['
                    MIN_VOLTAGE']) / 2.0
442             pwm_zero = np.interp(v_zero, [0, self.config['SUPPLY_VOLTAGE'
                    ]], [0, self.config['PWM_RESOLUTION']])
443             pwm_max = np.interp(self.config['MAX_VOLTAGE'], [0, self.
                    config['SUPPLY_VOLTAGE']], [1, self.config['PWM_RESOLUTION
                    ']])
444             pwm_min = np.interp(self.config['MIN_VOLTAGE'], [0, self.
                    config['SUPPLY_VOLTAGE']], [1, self.config['PWM_RESOLUTION
                    ']])
445             if self.calibrating == True:
446                 return pwm_zero
447             else:
448                 pwm = sig + pwm_zero # offset to zero
449                 if pwm > pwm_max:
```

```
450                     pwm = pwm_max
451                 elif pwm < pwm_min:
452                     pwm = pwm_min
453                 if self.config['PWM_INVERTED']:
454                     pwm = pwm_max - pwm + pwm_min
455                 self.controller.write(str(int(pwm)) + '\n') # Write to PWM
                        adaptor
456                 res = self.controller.readline()
457                 if int(res) != int(pwm):
458                     self.log_msg('CTRL', 'WARN: Controller returned bad
                            value!', important=True)
459                 else:
460                     self.log_msg('CTRL', 'OK: Set controller successfully'
                            )
461                 return pwm
462         except Exception as error:
463             self.log_msg('CTRL', 'ERROR: %s' % str(error), important=True)
464             #!TODO add bit here to automatically kill controller object
                    now?
465             try:
466                 self.controller.close()
467             except:
468                 pass
469             self.controller = None
470         b = time.time()
471         if self.config['VERBOSE']: self.log_msg('CTRL', '... %.2f ms' % ((
                b - a) * 1000))
472
473     ## Log to Mongo
474     def log_db(self, sample):
475         """
476         Log results to the database
477         Returns: Doc ID
478         """
479         if self.config['VERBOSE']: self.log_msg('MDB', 'Writing to
                database ...')
480         try:
481             if self.collection is None: raise Exception("Missing DB
                    collection!")
482             doc_id = self.collection.insert(sample)
483             self.log_msg('MDB', 'Doc ID: %s' % str(doc_id))
484         except Exception as error:
485             self.log_msg('MDB', 'ERROR: %s' % str(error), important=True)
486         return doc_id
487
488     ## Log to File
489     def log_data(self, data):
490         """
491         Open new text file
492         For each document in session, print parameters to file
```

```
493         """
494         try:
495             if self.data_log is None: raise Exception("Missing data
                    logfile!")
496             t = datetime.strftime(datetime.now(), self.config['TIME_FORMAT
                    '])
497             data_as_str = [str(d) for d in data]
498             self.data_log.write(',',.join([t] + data_as_str + ['\n']))
499         except Exception as error:
500             self.log_msg('LOG', 'ERROR: %s' % str(error), important=True)
501 
502     ## Log Messages
503     def log_msg(self, header, msg, important=False):
504         """
505         Saves important messages to logfile
506         """
507         try:
508             if self.error_log is None: raise Exception("Missing error
                    logfile!")
509             date = datetime.strftime(datetime.now(), self.config['
                    TIME_FORMAT'])
510             formatted_msg = "%s\t%s\t%s" % (date, header, msg)
511             self.error_log.write(formatted_msg + '\n')
512             if self.config['VERBOSE'] or important: print formatted_msg
513         except Exception as error:
514             print "%s\tLOG\tERROR: Failed to log message!\n" % date
515 
516     ## Display window briefly
517     def flash_window(self, img, title=''):
518         cv2.namedWindow(title, cv2.WINDOW_NORMAL)
519         if self.config['FULLSCREEN']: cv2.setWindowProperty(title, cv2.
                WND_PROP_FULLSCREEN, cv2.cv.CV_WINDOW_FULLSCREEN)
520         cv2.imshow(title, img)
521         if cv2.waitKey(5) == 0:
522             time.sleep(0.05)
523             pass
524 
525     ## Draw Lines
526     def draw_lines(self, img, offset):
527         """
528         Draw tolerance, offset, and center lines on image
529         """
530         (h, w, d) = img.shape
531         try:
532             TOLERANCE = int((self.config['CAMERA_WIDTH'] / self.config['
                    CAMERA_FOV']) * self.config['ERROR_TOLERANCE'])
533             cv2.line(img, (TOLERANCE + w/2 + self.config['CAMERA_OFFSET'],
                     0), (TOLERANCE + w/2 + self.config['CAMERA_OFFSET'], h),
                    (0,0,255), 1)
```

```
534             cv2.line(img, (-TOLERANCE + w/2 + self.config['CAMERA_OFFSET'
                    ], 0), (-TOLERANCE + w/2 + self.config['CAMERA_OFFSET'], h
                    ), (0,0,255), 1)
535             cv2.line(img, (offset + w/2, 0), (offset + w/2, h), (0,255,0),
                     2) # Average
536             cv2.line(img, (w/2 + self.config['CAMERA_OFFSET'], 0), (w/2 +
                    self.config['CAMERA_OFFSET'], h), (255,255,255), 1) #
                    Center
537             return img
538         except Exception as e:
539             raise e
540
541     ## Run Calibration
542     def run_calibration(self, delay=1.0, sweeps=2):
543         """
544         Execute calibration routine (if necessary) for hydraulic
                controller
545         """
546         self.calibrating = True
547         self.log_msg('LOG', 'WARN: Running calibration sweep ...',
                important=True)
548         v_zero = (self.config['MAX_VOLTAGE'] + self.config['MIN_VOLTAGE'])
                 / 2.0
549         pwm_zero = np.interp(v_zero, [0, self.config['SUPPLY_VOLTAGE']],
                [0, self.config['PWM_RESOLUTION']])
550         pwm_max = np.interp(self.config['MAX_VOLTAGE'], [0, self.config['
                SUPPLY_VOLTAGE']], [1, self.config['PWM_RESOLUTION']])
551         pwm_min = np.interp(self.config['MIN_VOLTAGE'], [0, self.config['
                SUPPLY_VOLTAGE']], [1, self.config['PWM_RESOLUTION']])
552         try:
553             for i in range(sweeps):
554                 # Hold at min
555                 self.controller.write(str(int(pwm_min)) + '\n') # Write to
                         PWM adaptor
556                 self.controller.readline()
557                 time.sleep(delay)
558                 # Sweep up
559                 for pwm in range(int(pwm_min), int(pwm_max)):
560                     self.controller.write(str(int(pwm)) + '\n') # Write to
                             PWM adaptor
561                     self.controller.readline()
562                 # Hold at max
563                 self.controller.write(str(int(pwm_max)) + '\n') # Write to
                         PWM adaptor
564                 self.controller.readline()
565                 time.sleep(delay)
566                 # Sweep down
567                 for pwm in range(int(pwm_max), int(pwm_min)):
568                     self.controller.write(str(int(pwm)) + '\n') # Write to
                             PWM adaptor
```

```
569                     self.controller.readline()
570         except Exception as error:
571             self.log_msg('LOG', 'ERROR: %s' % str(error), important=True)
572         self.calibrating = False
573 
574     ## Generate Output Image
575     def generate_output(self, images, masks, avg, volts):
576         """
577         Generates a single composite image of the multiple camera feeds,
                with lines drawn
578         """
579         output_images = []
580         if self.config['HIGHLIGHT']:
581             if self.config['VERBOSE']: self.log_msg('DISP', 'Using mask
                    for output images')
582             for mask in masks:
583                 try:
584                     if mask is None: mask = np.zeros((self.config['
                            CAMERA_HEIGHT'], self.config['CAMERA_WIDTH'], 1),
                            np.uint8)
585                     img = np.array(np.dstack((mask, mask, mask)))
586                     output_images.append(self.draw_lines(img, avg))
587                 except Exception as error:
588                     raise error
589         else:
590             if self.config['VERBOSE']: self.log_msg('DISP', 'Using RGB for
                     output images')
591             for img in images:
592                 try:
593                     if img is None: img = np.zeros((self.config['
                            CAMERA_HEIGHT'], self.config['CAMERA_WIDTH'], 3),
                            np.uint8)
594                     output_images.append(self.draw_lines(img, avg))
595                 except Exception as error:
596                     raise error
597         output = np.vstack(output_images)
598 
599         # Add Padding
600         pad_y = self.config['CAMERA_HEIGHT'] * 0.15
601         pad = np.zeros((pad_y, self.config['CAMERA_WIDTH'], 3), np.uint8)
                # add blank space
602         output = np.vstack([output, pad])
603 
604         # Offset Distance
605         distance = round((avg - self.config['CAMERA_OFFSET']) / (self.
                config['CAMERA_WIDTH'] / self.config['CAMERA_FOV']), 2)
606         self.log_msg('DISP', 'Offset Distance: %d' % distance)
607         if avg - self.config['CAMERA_WIDTH'] / 2 >= 0:
608             distance_str = str("+%2.1f cm" % distance)
609         elif avg - self.config['CAMERA_WIDTH'] / 2 < 0:
```

```
610             distance_str = str("%2.1f cm" % distance)
611         else:
612             distance_str = str("  .  cm")
613         cv2.putText(output, distance_str, (int(self.config['CAMERA_WIDTH']
                * 0.04), int(self.config['CAMERAS'] * self.config['
                CAMERA_HEIGHT'] + pad_y / 1.5)), cv2.FONT_HERSHEY_SIMPLEX,
                0.75, (255,255,255), 2)
614
615         # Output Voltage
616         volts_str = str("%2.1f V" % volts)
617         cv2.putText(output, volts_str, (int(self.config['CAMERA_WIDTH'] *
                0.72), int(self.config['CAMERAS'] * self.config['CAMERA_HEIGHT
                '] + pad_y / 1.5)), cv2.FONT_HERSHEY_SIMPLEX, 0.75,
                (255,255,255), 2)
618
619         # Arrow (Directional)
620         if avg - self.config['CAMERA_WIDTH'] / 2 >= 0:
621             p = (int(self.config['CAMERA_WIDTH'] * 0.45), int(self.config[
                    'CAMERAS'] * self.config['CAMERA_HEIGHT'] + pad_y / 2))
622             q = (int(self.config['CAMERA_WIDTH'] * 0.55), int(self.config[
                    'CAMERAS'] * self.config['CAMERA_HEIGHT'] + pad_y / 2))
623         elif avg - self.config['CAMERA_WIDTH'] / 2 < 0:
624             p = (int(self.config['CAMERA_WIDTH'] * 0.55), int(self.config[
                    'CAMERAS'] * self.config['CAMERA_HEIGHT'] + pad_y / 2))
625             q = (int(self.config['CAMERA_WIDTH'] * 0.45), int(self.config[
                    'CAMERAS'] * self.config['CAMERA_HEIGHT'] + pad_y / 2))
626         color = (255,255,255)
627         thickness = 4
628         line_type = 8
629         shift = 0
630         arrow_magnitude=15
631         cv2.line(output, p, q, color, thickness, line_type, shift) # draw
                arrow tail
632         angle = np.arctan2(p[1]-q[1], p[0]-q[0])
633         p = (int(q[0] + arrow_magnitude * np.cos(angle + np.pi/4)), #
                starting point of first line of arrow head
634         int(q[1] + arrow_magnitude * np.sin(angle + np.pi/4)))
635         cv2.line(output, p, q, color, thickness, line_type, shift) # draw
                first half of arrow head
636         p = (int(q[0] + arrow_magnitude * np.cos(angle - np.pi/4)), #
                starting point of second line of arrow head
637         int(q[1] + arrow_magnitude * np.sin(angle - np.pi/4)))
638         cv2.line(output, p, q, color, thickness, line_type, shift) # draw
                second half of arrow head
639         return output
640
641     ## Update the Display
642     def update_display(self, images, masks, volts, avg):
643         """
644         Check for concurrent update process
```

```
645         Draw graphics on images
646         Output GUI display
647         """
648         if self.config['VERBOSE']: self.log_msg('DISP', 'Updating display
                ...')
649         a = time.time()
650         if self.updating:
651             return # if the display is already updating, wait and exit (
                    puts less harm on the CPU)
652         else:
653             self.updating = True
654             try:
655                 output_small = self.generate_output(images, masks, avg,
                        volts)
656                 self.output_image = output_small
657                 output_large = cv2.resize(output_small, (self.config['
                        DISPLAY_WIDTH'], self.config['DISPLAY_HEIGHT']))
658 
659                 # Draw GUI
660                 self.log_msg('DISP', 'Output shape: %s' % str(output_large
                        .shape))
661                 self.flash_window(output_large, title='Agri-Vision')
662             except Exception as error:
663                 self.updating = False
664                 self.log_msg('DISP', str(error), important=True)
665             self.updating = False
666         b = time.time()
667         self.log_msg('DISP', '... %.2f ms' % ((b - a) * 1000))
668 
669     ## Update GPS
670     def update_gps(self):
671         """
672         Get the most recent GPS data
673         Sets: Global variables lat, long and speed
674         """
675         if self.config['GPS_ENABLED']:
676             if self.gpsd is not None:
677                 try:
678                     self.gpsd.next()
679                     self.latitude = self.gpsd.fix.latitude
680                     self.longitude = self.gpsd.fix.longitude
681                     self.speed = self.gpsd.fix.speed
682                     self.log_msg('GPS', '%d N %d E' % (self.latitude, self
                            .longitude))
683                 except Exception as error:
684                     self.log_msg('GPS', 'ERROR: %s' % str(error),
                            important=True)
685 
686     ## Close
687     def close(self, delay=0.5):
```

```
688         """
689         Function to shutdown application safely
690         1. Close windows
691         2. Disable controller
692         3. Release capture interfaces
693         """
694         self.log_msg('SYS', 'Shutting Down!')
695         if self.controller is None:
696             self.log_msg('WARN', 'Controller already off!')
697         else:
698             try:
699                 self.log_msg('CTRL', 'Closing Controller ...')
700                 self.controller.close() ## Disable controller
701             except Exception as error:
702                 self.log_msg('CTRL', 'ERROR: %s' % str(error), important=
                        True)
703         for i in range(len(self.cameras)):
704             if self.cameras[i] is None:
705                 self.log_msg('CAM', 'WARN: Camera %d already off!' % i)
706             else:
707                 try:
708                     self.log_msg('CAM', 'WARN: Closing Camera %d ...' % i)
709                     self.cameras[i].release() ## Disable cameras
710                 except Exception as error:
711                     self.log_msg('CAM', 'ERROR: %s' % str(error),
                            important=True)
712         if self.config['DISPLAY_ON']:
713             cv2.destroyAllWindows() ## Close windows
714 
715     ## Check Controller
716     def check_controller(self):
717         """ Checks controller for functioning stream """
718         if self.config['VERBOSE']: self.log_msg('CTRL', 'Checking
                controller ...')
719         if self.controller is None:
720             try:
721                 self.controller = self.attach_controller()
722             except Exception as e:
723                 self.log_msg('CTRL', 'ERROR: Failed to attach controller!'
                        , important=True)
724 
725     ## Run
726     def run(self):
727         """
728         Function for Run-time loop
729         1. Capture images
730         2. Generate mask filter for plant matter
731         3. Calculate indices of rows
732         5. Estimate row from both images
733         4. Get number of averages
```

```
734             5. Calculate moving average
735             6. Send PWM response to controller
736             7a. Log results to DB
737             7b. Display results
738             """
739             if self.running:
740                 return # Don't do anything
741             else:
742                 self.running = True
743                 try:
744                     images = self.capture_images()
745                     masks = self.bppd_filter(images)
746                     offsets, sums = self.find_offset(masks)
747                     (est, avg, diff) = self.estimate_row(offsets, sums)
748                     sig = self.calculate_output(est, avg, diff)
749                     pwm = self.set_controller(sig)
750                     volts = self.to_volts(pwm)
751                     self.log_msg("SYS", "err:%s, cams:%d, pwm:%d, volts:%2.2f"
                            % (str(offsets), avg, pwm, volts), important=True)
752                     if self.config['MONGO_ON']:
753                         sample = {'sig':sig, "pwm":pwm, "volts":volts, "est":
                                est, "sums":sums, "offsets":offsets}
754                         doc_id = self.log_db(sample)
755                     if self.config['DATALOG_ON']: self.log_data([est, sig, pwm
                            , volts])
756                     if self.config['DISPLAY_ON']:
757                         try:
758                             os.environ['DISPLAY']
759                             threading.Thread(target=self.update_display,
760                                 args=(images, masks, volts, avg),
761                                 kwargs={}
762                             ).start()
763                         except Exception as error:
764                             self.log_msg('SYS', 'ERROR: %s' % str(error),
                                    important=True)
765                     else:
766                         self.output_image = self.generate_output(images, masks
                                , avg, volts)
767                 except KeyboardInterrupt as error:
768                     self.shutdown()
769                 except UnboundLocalError as error:
770                     self.log_msg('SYS', 'ERROR: %s' % str(error), important=
                            True)
771                     self.shutdown()
772                 except IOError as error:
773                     self.log_msg('SYS', 'ERROR: %s' % str(error), important=
                            True)
774                     self.shutdown()
775                 except Exception as error:
776                     self.log_msg('SYS', 'ERROR: %s' % str(error))
```

```
777                 try:
778                     for i in range(self.config['CAMERAS']):
779                         self.attach_camera(i)
780                 except Exception as error:
781                     self.log_msg('SYS', 'ERROR: %s' % str(error),
                        important=True)
782             self.running = False
783 
784     ## Save Image
785     def save_image(self, filename='out.jpg', subdir='agionic/www'):
786         """ Save image to output file (to be loaded by webserver) """
787         try:
788             if self.config['VERBOSE']: self.log_msg('HTTP', 'Saving output
                    image to file')
789             filepath = os.path.join(self.CURRENT_DIR, subdir, filename)
790             cv2.imwrite(filepath, self.output_image)
791         except Exception as error:
792             self.log_msg('SYS', 'ERROR: %s' % str(error), important=True)
793 
794     ## Save Settings
795     def save_settings(self, keyval, filename='custom.json', subdir='modes'
        ):
796         """ Save current config to a custom config file """
797         keys = keyval.keys()
798         vals = [int(v) for v in keyval.values()]
799         new_settings = dict(zip(keys, vals))
800         self.log_msg('CONF', str(new_settings))
801         self.config.update(new_settings)
802         filepath = os.path.join(self.CURRENT_DIR, subdir, filename)
803         with open(filepath, 'w') as jsonfile:
804             jsonfile.write(json.dumps(self.config, indent=4, sort_keys=
                True))
805 
806     ## Default Settings
807     def default_settings(self, filename='default.json', subdir='modes'):
808         """ Resets config to default settings file """
809         filepath = os.path.join(self.CURRENT_DIR, subdir, filename)
810         print "WARNING: Loading %s" % filepath
811         if os.path.exists(filepath):
812             with open(filepath, 'r') as jsonfile:
813                 self.config = json.loads(jsonfile.read())
814         else:
815             print "FATAL ERROR: No config found!"
816             exit(1)
817 
818     ## Load Settings
819     def load_settings(self, config_file, subdir='modes'):
820         """ Load config from input file """
821         filepath = os.path.join(self.CURRENT_DIR, subdir, config_file)
822         print "WARNING: Loading %s" % filepath
```

```
823         if os.path.exists(filepath):
824             with open(filepath, 'r') as jsonfile:
825                 self.config = json.loads(jsonfile.read())
826         else:
827             print "ERROR: Specified config not found! Trying default!"
828             self.default_settings()
829 
830     ## Render Index
831     @cherrypy.expose
832     def index(self, indexfile="index.html"):
833         """ Render index file for webhooks"""
834         indexpath = os.path.join(self.CURRENT_DIR, self.config['
            CHERRYPY_PATH'], indexfile)
835         with open(indexpath) as html:
836             return html.read()
837 
838     ## Handle Posts
839     @cherrypy.expose
840     def default(self, *args, **kwargs):
841         """ This function is basically the API """
842         try:
843             url = args[0] #!TODO: select action depending on request type/
                url
844             #self.save_image()
845             if url == 'update':
846                 # Parse JSON object to pythonic dict
847                 post = kwargs.keys()
848                 data = post[0]
849                 unicode_config = json.loads(data)
850                 new_configs = dict([(str(k), v) for k, v in unicode_config
                    .items()])
851                 self.save_settings(new_configs)
852             elif url == 'config':
853                 self.log_msg("HTTP", "Caught /config request")
854                 return json.dumps(self.config)
855             elif url == 'default':
856                 self.log_msg("HTTP", "Caught /default request")
857                 self.default_settings()
858                 return json.dumps(self.config)
859             elif url == 'calibrate':
860                 self.log_msg("HTTP", "Caught /calibrate request")
861                 self.run_calibration()
862         except Exception as err:
863             self.log_msg('ERROR', str(err), important=True)
864         return None
865 
866     ## CherryPy Reboot
867     @cherrypy.expose
868     def shutdown(self):
869         cherrypy.engine.exit()
```

```
870
871 ## Main
872 if __name__ == '__main__':
873     session = app(CONFIG_FILE)
874     cherrypy.server.socket_host = session.config['CHERRYPY_ADDR']
875     cherrypy.server.socket_port = session.config['CHERRYPY_PORT']
876     conf = {
877         '/': {'tools.staticdir.on':True, 'tools.staticdir.dir':os.path.
                join(session.CURRENT_DIR, session.config['CHERRYPY_PATH'])},
878         '/js': {'tools.staticdir.on':True, 'tools.staticdir.dir':os.path.
                join(session.CURRENT_DIR, session.config['CHERRYPY_PATH'], 'js
                ')},
879         '/logs': {'tools.staticdir.on':True, 'tools.staticdir.dir':os.path
                .join(session.CURRENT_DIR, 'logs')},
880     }
881     cherrypy.quickstart(session, '/', config=conf)
882     session.close()
883     #if session.reboot: os.system("reboot")
```