Technical University of Denmark

Aske Bay Jakobsen

Crop row navigation for autonomous field robot

Master's thesis, April 2015

DTU Electrical Engineering
Department of Electrical Engineering

Crop row navigation for autonomous field robot
Navigation i rækkeafgrøder med autonom markrobot

Report written by:
Aske Bay Jakobsen

Advisor(s):
Jens Christian Andersen

DTU Electrical Engineering
Automation and Control
Technical University of Denmark
Elektrovej
Building 326
2800 Kgs. Lyngby
Denmark
Tel : +45 4525 3576

studieadministration@elektro.dtu.dk

Project period:	24/11/2014 – 24/04/2015
ECTS:	30
Education:	MSc
Field:	Electrical Engineering
Class:	Public
Remarks:	This report is submitted as partial fulfilment of the requirements for graduation in the above education at the Technical University of Denmark.
Copyrights:	

Abstract

In an effort to increase the productivity of modern agriculture a host of robotic platforms and machinery has seen the light of day. This thesis explores the possibilities for developing a system that enables a small mobile robot to navigate in crop rows using visual feedback. To increase the reusability of the software developed, it has been implemented as a plug-in for Mobotware. It utilizes OpenCV functions to accomplish the necessary image analysis used for navigation. To control the robot a scripting language is used that contains useful premade functionalities that can be readily applied to an array of hardware setups. System verification is based on tests performed using the robot in an emulated field environment.

Contents

List of Figures

Chapter 1

Introduction

Since the industrial revolution, automation has served to raise the productivity of societies, increased the availability of goods and has heightened the standard of living. From the early beginnings until modern day, more and more industries have been benefiting from a growing amount of automation, not least the agricultural sector.
In the last couple of decades many advances have been made in computing and sensor technologies, and the price of such devices have gone down. This means that automation need not to be mechanical muscle alone, but can also act as mechanical minds. This fact has opened up new possibilities for utilizing automation in areas not previously thought viable.
This thesis deals with the systems needed for enabling an autonomous field robot to navigate in rows of crops using visual feedback. The robot does so by analysing the data obtained from an array of sensors, most notably a color camera.

1.1 Problem Formulation

The objectives of this thesis is to develop, and verify through experimentation, a system that will enable the robotic platform *Vibro Crop Robotti* from *Kongskilde*, to navigate a field of crops planted in rows using visual feedback.

To achieve the goals the robot must be able to operate robustly in the field, dealing with situations where plants, or even entire rows are missing. Further more the system must be able to detect upcoming headland, and, once reached, be able to safely navigate it and return to the next row.
The implementation of the system is to be made in the form of a *Mobotware* plug-in and an associated MRC-script.

1.2 Thesis Structure

The thesis is divided into a number chapters that are structured in the following way:

Overview
Contains information about the hardware platform used in the project as well as an overview of the project objectives in general.

Image Segmentation
In this chapter all the steps used in the image segmentation process is described. The result of the steps is a black and white image that forms the foundation for the next chapter.

Hough Transformation
This part describes how line features are extracted from the black and white image. First a brief explanation of the theory behind the method and then some examples from the practical implementation.

Implementation
This chapter explains how the developed software is implemented and then used to navigate the robotic platform.

Results
The data and results acquired from test with the full system on two separate hardware platforms is discussed.

Further Work
Contains recommendations for improvements on the system that could be achieved with further work.

Conclusion
Concludes on the results obtained throughout project.

Chapter 2

Overview

This section presents an overview of the thesis and contains a brief explanation of the overall goals and chapters to come. It also consists of a description of the hardware platform used in this project.

2.1 The robotic platform

The robotic platform is the *Kongskilde Vibro Crop Robotti* developed by Kongskilde in collaboration with The University of Southern Denmark. It consists of two individual belt-driven units that can be equipped with an array of different tools. The primary tool intended to be used with the systems developed in this project, is a cultivator. Its purpose is to cut the roots of weed growing between the rows of crops, thus reducing the need for herbicides.

Each of the drive units is powered by a $48V$ $5kW$ brushless DC motor. The motors are controlled by two Roboteq HBL1650 Brushless DC Motor controllers, each capable of providing $150A$.
The robot is equipped with a total of ten $12V$ lead-acid car batteries. One battery in each drive unit is reserved for the motor controllers, computer and other electronics. This brings the weight of the robot up to roughly $400kg$.
The computing power comes from a *Robotech Controller 701* by Conpleks. It features an Intel *Core i7* processor as well as options for connecting peripheral devices.

The primary sensor used to detect the crop rows is an *Xbox 360 Kinect*. A Kinect contains not only an RGB camera but also a three dimensional depth sensor. The depth sensor works by analysing the pattern of an infrared light emitted by the Kinect, thus making it susceptible to other sources of infrared. The Sun is a powerful source of such light, making it difficult to use the sensor outside. For this reason the use of the depth sensor will not be utilized in this project. This entails that all information about the location of crops, comes solely from the RGB camera. The position of the Kinect on the Robotti platform can be seen in figure 2.2

Figure 2.1: Vibro Crop Robotti by Kongskilde

2.2 Objective

The Robotti platform is part of the Kongskilde Vibro Crop family. The series aims offer intelligent technological solutions in agriculture. The existing guidance system for the Robotti is based on a *Real Time Kinematic gps* (RTK-gps). This system is very accurate, but has a few drawbacks. First, it is a costly solution as the RTK-gps unit itself is very expensive. Secondly, it relies on having a priori knowledge of the position of every plant in the field, down to the same accuracy as the system itself. This is not necessarily a problem though - if the same field was planted with a precision farming device, this information is readily available. The third problem however, is the fact that an RTK-gps need better reception than an average gps. If this can not be obtained, the system doesn't work reliably.
Another negative side effect, of relying on pre-acquired field data, is a decreased demonstrability. Potential buyers of the system can not have it demonstrated in their own field, unless they are already users of precision farming.

For these reasons, a system that can act as a back-up, or a stand alone solution, is to be developed. As such, the primary objective of the project is to enable the Robotti to work autonomously in a field using visual navigation. The way this has been envisioned is illustrated in figure 2.3. In the left half of the image the expected field layout is displayed. It consists of rows of plants, and the headland can be any of two types - crossing rows or bare soil. The blue trapezoid represents the field of view.
The right half of the image shows the robot path through the field. The *black ar-*

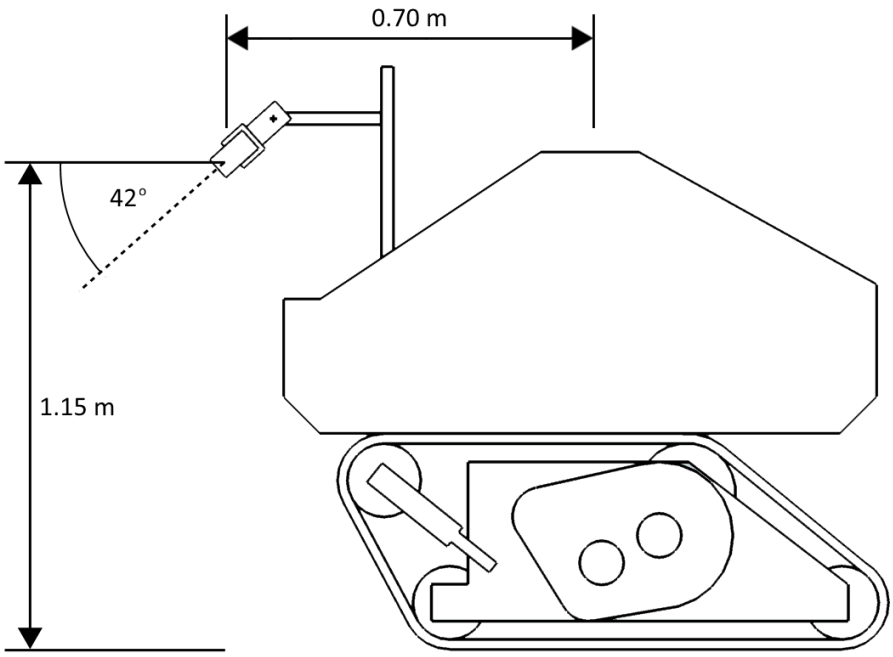


Figure 2.2: The Robotti platform with the Kinect sensor attached.

rows represent sections of the field where the robot is guided by visual feedback. The *red arrows* represents the part where robot is navigating the headland. In the headland, the robot uses odometry to keep track of its position. The arrows also illustrate the pattern that is used to ensure that the robot get through all the rows.

2.3 Visual feedback

When the robot is driving in the field, it is guided by visual feedback. To get a sense of the environment, images from the Kinect camera are analysed in such a way that the robot can detect the individual rows of plants. From the information about the rows, a *guide point* can be found that will aim the robot in such a way that it steers clear of the rows themselves.
The steps needed to analyse the image are as follows:

- Convert the image to the desired color space
- Mask the image using color thresholding
- Filter the mask to remove noise
- Blur the image if necessary

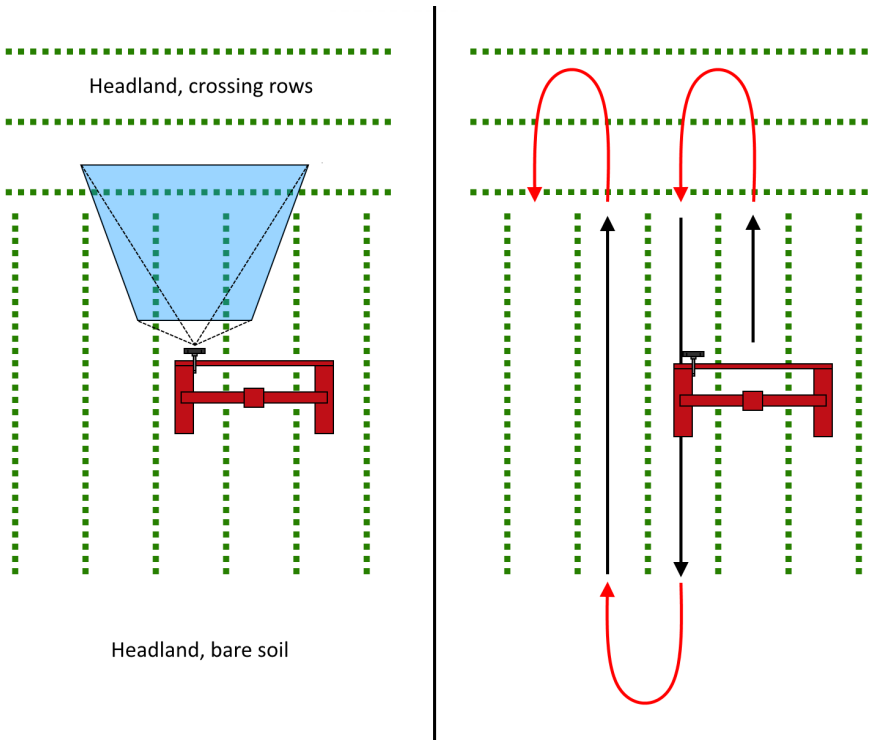


Figure 2.3: Overview of expected solution.

- Extract line features using the hough transform

2.4 Implementation

To realize the objective, two pieces of software are made. The first piece, and the largest by far, is the software responsible for the image analysis. It is implemented in the form of a plug-in for *Mobotware*. The plug-in gets an image from the Kinect and performs all the necessary steps to calculate the guide point. Many of the functions used are from the *Open CV*-library.
The second piece of software is a script, written using the *SMRCL* language. The script receives the information from the plug-in and uses it to manipulate the robot hardware accordingly.

2.5 Regarding Plants

As one of the major parts of this project deals with the visual identification of plant material, it would be natural to assume that some actual plant material has been examined. However, due to the months of the year during which the project has been undertaken, the correct type of plants have been exceedingly

difficult to acquire.
To remedy this unfortunate situation, the following steps have been taken.

- Images of corn fields have been supplied by Kongskilde and have been used for testing the image analysis. They form a large part of the foundation used during development of the plug-in. The images suffer from being taken at angles and positions that do not match the setup used in the final solution.
- A futile attempt was made at growing live corn plants. They ended up only serving as a reference for creating fake plants.
- Fake plants have been used to get verification that the system can detect something that resembles real plants under testing conditions.
- For larger scale testing, 20mm blue nylon rope has been used to represent rows of plants. To justify this solution the assumption has been made that, while different in appearance, other characteristics, such as the distance between rows and overall field layout, can be replicated accurately enough to substitute actual crop rows.

While these solutions are not optimal, they have proved useful for testing during the project. Figure 2.4 shows the development of fake plants used in the project.

Figure 2.4: Development of fake corn plants using a living plant as reference.

Chapter 3

Image Segmentation

Image segmentation is the process of dividing an image into regions of interest that can be interpreted by a computer. As humans we use our sight all the time and detecting objects and features can seem trivial. For a computer, however, getting consistent and robust results is often a rather resource-demanding task. Each possible variation in the analysis of the image, be it changes in lighting, colors etc, adds complexity to the way specific regions are found.
In this chapter the methods used for extracting line features made of crops growing in soil, is examined. The main concept is based on segmentation by thresholding the color spectrum of the image.

3.1 Color Space

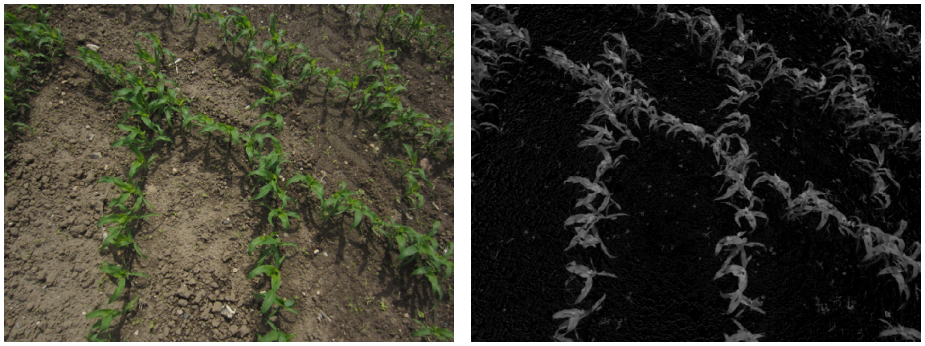
The color space of a digital image is the way color information is stored and represented. A common way represent colors is in the RGB format, where R stands for red, G for green and B for blue. The image is stored in three separate layers, each layer containing the information of its respective color. It is the combination of the layers that express the final image.
This is generally a convenient representation since most displays operate on the same principle, having each pixel consist of a red, green and blue component. It also has the benefit of being the format of the images captured by the kinect camera. However, other color spaces exist, each having their own advantages and disadvantages given the situation.
In this section two color spaces will be investigated, RGB and YUV. They will be evaluated based on their suitability for separating colors using an automatic threshold algorithm, the algorithm is explained further in section 3.2.2.

3.1.1 RGB

As metioned the RGB image is comprised of three layers, all containing color information. As the goal is to find green colored plants, a mask is applied that generates a single grayscale image highlighting green regions of interest.

A common way to make this mask is to use what is known as the *excess green vegetation index*, or simply ExG (Amir, R. 2014). The mask is applied on every pixel and uses the following operation:

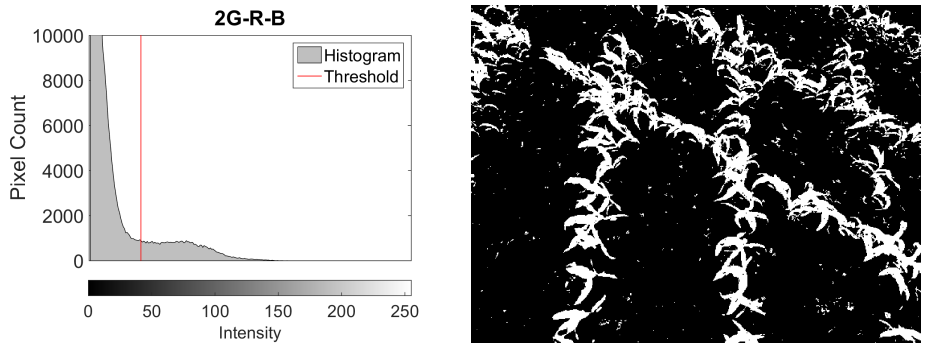
$$I = 2G - R - B \tag{3.1}$$



(a) Raw image to be analyzed (b) Image after ExG mask is applied

Figure 3.1

The result of this operation is demonstrated in figure 3.1. As expected the background is darkened and the plants are highlighted. To obtain a binary image, an intensity threshold is applied. A good estimate of how well this threshold separates regions of interest can be determined by looking at the image histogram.



(a) RGB-image histogram (b) Binary image

Figure 3.2

The method for finding the threshold value assumes a bimodal distribution, the details of which are described in section 3.2. As such, a better segmentation should result from more distinct peaks in the histogram.
In figure 3.2a it can be seen that the histogram is dominated by a large peak in the low intensity area. This peak represents the background of the image while the smaller, less distinct peak with high intensity, represents the plants. The

threshold is correctly placed between the peaks, and the binary image obtained by applying the threshold can be seen in figure 3.2b.

3.1.2 YUV

The YUV color space differs from the RGB format in the way that only two layers have color information. The Y layer only contains the luminance - the intensity of the image and the U and V layers contains the chromaticity - the color information. Figure 3.3 shows how the colors are mapped.

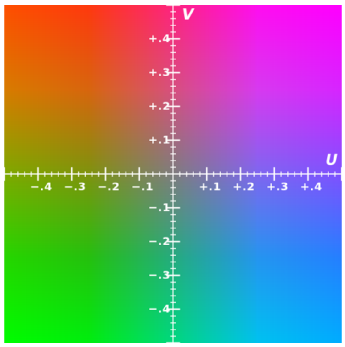
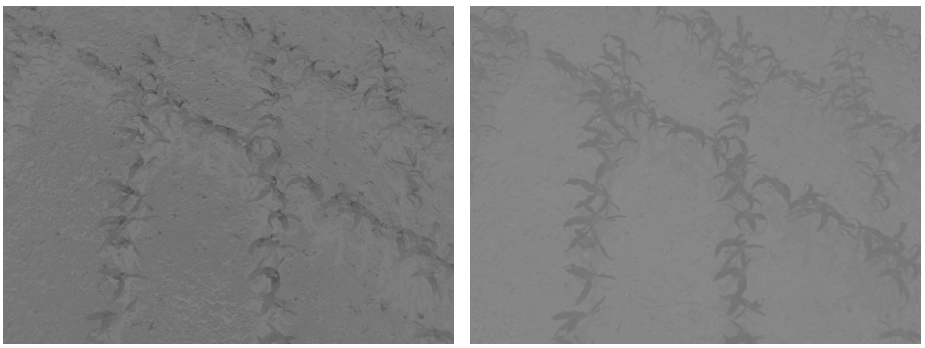


Figure 3.3: The color-map of the U and V layers

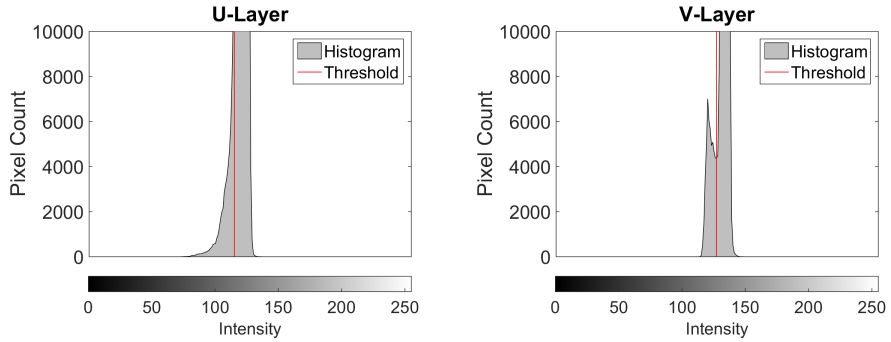
Since each of the color layers, U and V, can be considered a grayscale image on its own, there is no need for a transformation like equation 3.1, as was the case for the RGB image, however, a threshold needs to be found for both of the layers.
By using the raw image in figure 3.1a as a starting point and then converting it to YUV, the following images can be obtained:



(a) U-Layer (b) V-Layer

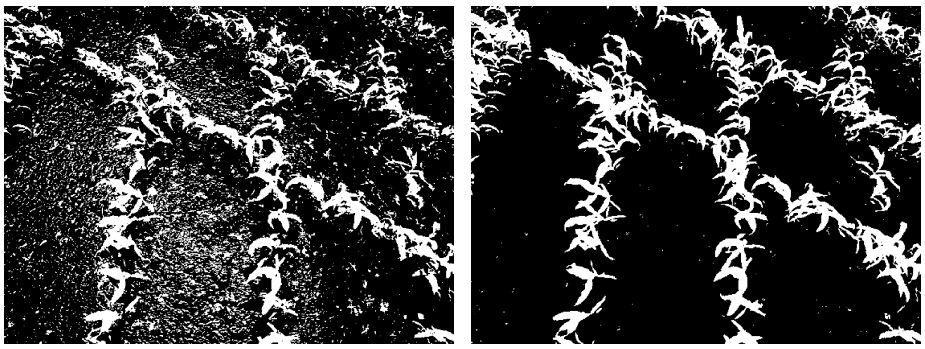
Figure 3.4: U and V layers as grayscale images

It is noticeable that the histograms in figure 3.5 occupies a much narrower band of intensities than was the case with the RGB-format. And while there is no two separate peaks in the U-layer, very distinct peaks exist for the V-layer. This is also apparent in the binary images in figure 3.6, where the U-layer contains lots



(a) U-Layer histogram (b) V-Layer histogram

Figure 3.5: Histograms of the U and V layers



(a) U-Layer binary (b) V-Layer binary

Figure 3.6: U and V layers as binary images

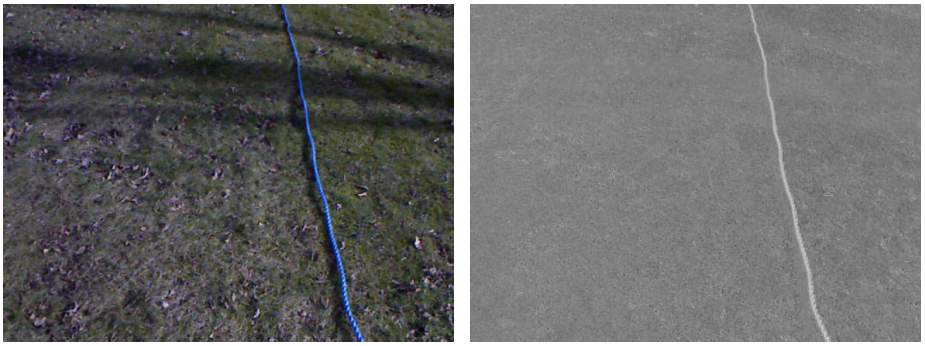
of unwanted noise while the V-layer has good separation.
Which of the two layers that has the best separation differs depending of the initial image, so to further clear up unwanted noise, the binary images are combined using a pixel-wise logical AND operation. The resulting image has a more clear separation, regardless of which of the individual layers created the best result. Such a combination of figure 3.6a and 3.6b can be seen in figure 3.7.

An additional feature, gained by using the YUV color space, comes from the fact that there is a clear separation between color and intensity. Even though the intensity layer is not used for any analysis on its own, the very nature of the colorspace should result in a superior tolerance to changes in the lighting conditions, when compared to the RGB format. The argument for this behaviour arises from the inherent separation of the layers - while an object may appear to change color as more or less light hits it, the underlying information in the color layers should not change, only the intensity.
The idea that colors are completely independent from intensity, however, is not always possible to recreate under test conditions. A case where color is maintained reasonably well, with varying lighting, is illustrated in figure 3.8. As can be seen on the color image, the shadow of a tree covers a significant part of the

grassy background. By looking at the grayscale image though, it reveals very little trace of the shadow.
Figure 3.9 on the other hand, demonstrates how shadows may leave severe artifacts in the color layers. The issue is particularly problematic in this case, as the shadow is cast by the robot itself. This means that it is likely to interfere for quite some time.

A reason for the inconsistency in the perceived color is likely due to something as novel as the color of the sky. On clear days, when shadows are obviously more prevalent, the light is tinted slightly blue. This blue tint interferes in different ways, depending on the color it falls on. The examples above would imply that green colors are less susceptible to tinting than the more reddish nuances.

Figure 3.7: U and V layers combined using pixel-wise logical AND



(a) Full color image (b) The V color layer

Figure 3.8: The effect of shadows cast on a green background

3.2 Color Thresholding

The main method for segmenting out the regions of interest is based on color thresholding, that is, finding objects in an image based on their respective colors. Many different ways of finding the threshold values and applying them to the image exits. In this project two methods for obtaining thresholds are exam-

(a) Full color image (b) The V color layer

Figure 3.9: The effect of shadows cast on a red background

ined and compared against each other. The first method is manual thresholding, the second uses an algorithm for automatically finding a value and it is known as Otsu's method. The threshold values found by both of these methods are applied to the image globally.

The task of segmenting by colors can be made significantly easier by having some prior knowledge about scenes to be analysed. In this project a number of such assumptions are made about the images that will be segmented.

- All regions of interest consist primarily of one color
- The background consists primarily of one color - different from the regions of interest
- The colors form a bimodal distribution

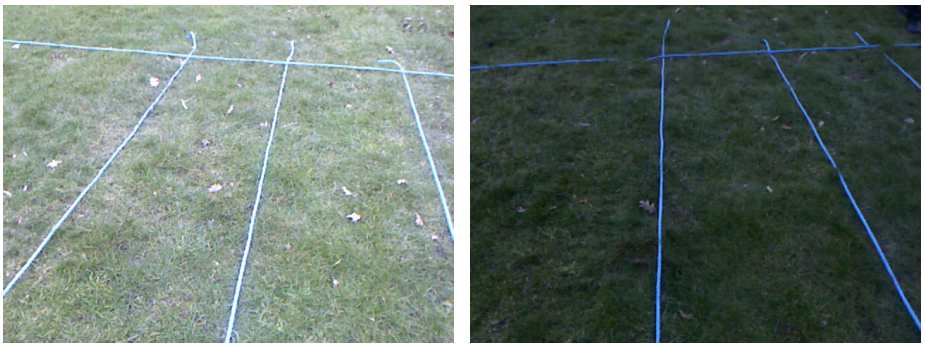
These assumptions have some practical implications that will be further discussed in the sections to come.

3.2.1 Manual Threshold

Manual thresholding is, as the name implies, the use of a human operator to manually determine the threshold value needed to segment out a specific region of interest. When using manual thresholding one is leveraging the superior capabilities of the human mind to perceive patterns and understand context. This can give very good results, but also poses a host of problems - especially for systems that are supposedly autonomous.

In its simplicity the threshold values are found by looking at the masked image that appears after the current threshold has been applied. Since the color space used is the YUV format, two threshold values are needed, one for each of the color layers as described in section 3.1.2.
By tweaking the thresholds and evaluating the resulting mask, very solid segmentation can be obtained. However, this approach can be problematic for

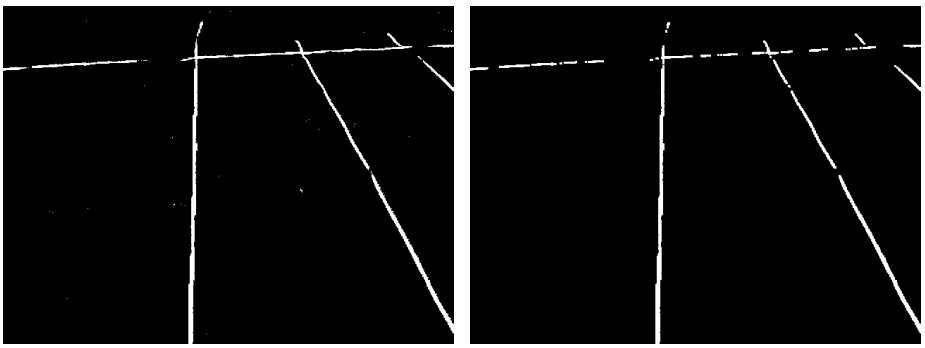
several reasons. In this project the system responsible for the image analysis is wholly contained on the mobile platform. This means that any threshold can only be set before the platform is running and, consequently, are only based on the initial conditions. This can be perfectly fine in a static environment, but as the intended work space for the Robotti is outdoor fields and areas, static conditions are unlikely to occur most of the time.



(a) Rope on grass, clear skies (b) Rope on grass, clouded skies

Figure 3.10: These two pictures are taken from the same Robotti test-run within a few meters of each other. The scenery is the same except from the different angles and lighting conditions.

Figure 3.10 show the difference in lighting that arises from the sky being clouded or clear. Not only is the clear image lighter overall, it also changes the appearance of the rope from blue to a more white color.
This kind of change in lighting conditions is one of the problems that manual thresholding is ill suited to deal with.

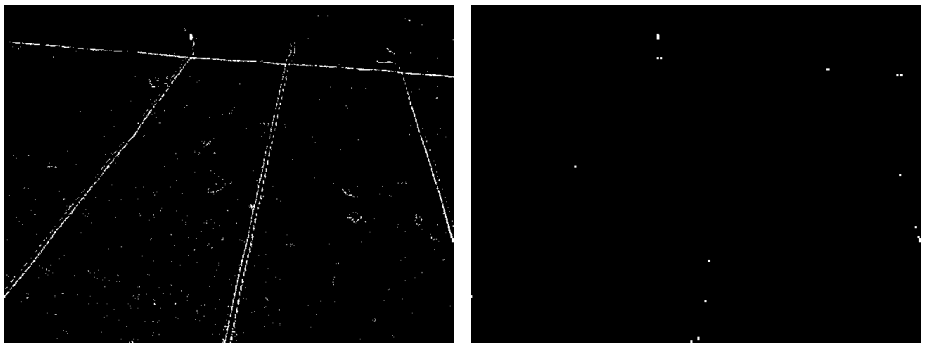


(a) Unfiltered (b) Filtered

Figure 3.11: Manual thresholds found using the dark image (figure 3.10b)

Figure 3.11 shows the filtered and unfiltered masks that results from applying a manually found threshold to the dark image in figure 3.10b. The blue rope stands out from the background very clearly, and the filtering process does not change the image by a large amount. From the perspective of an operator no additional tweaking of the thresholds would seem necessary.

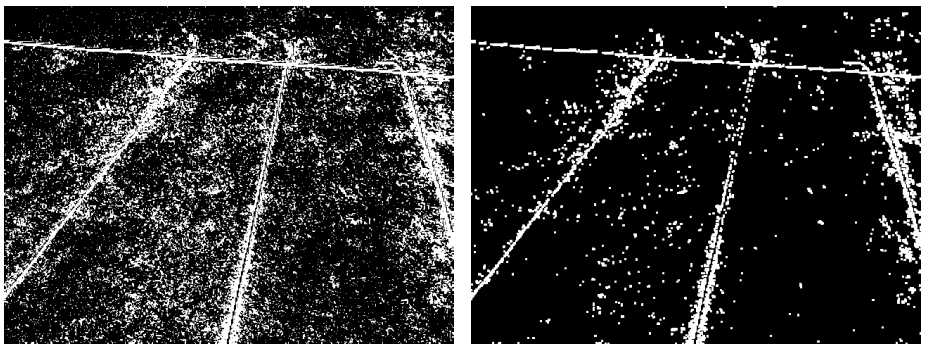
Figure 3.12 shows the effect of the same threshold values being applied to the light image in figure 3.10a. While the unfiltered mask retains some information about the rope, it is insufficient to survive the subsequent filtering process. The filtered image holds no features usable for guiding the robot.



(a) Unfiltered (b) Filtered

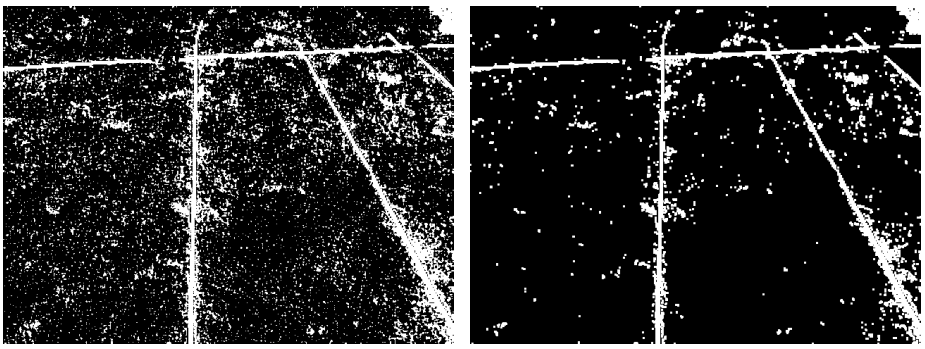
Figure 3.12: The masks of the light image created with threshold values found using the dark image

In the case where the threshold values are first found using the dark image, the same values could not be used successfully on the lighter image. To demonstrate the unpredictable nature of static thresholds used under varying lighting conditions, the following example will not have its starting point in darker image. The same operations will be performed, but the manual threshold values will be found using the light image.
Figure 3.13 presents the masks found by applying the manual threshold. As can be seen, the image is a lot noisier but does retain more information after the filtering.
Figure 3.14 shows the masks for the dark image. Again the images contain lots of noise but also here it is possible to extract line features.



(a) Unfiltered (b) Filtered

Figure 3.13: Manual thresholds found using the light image (figure 3.10a)



(a) Unfiltered (b) Filtered

Figure 3.14: The masks of the dark image created with threshold values found using the light image

3.2.2 Automatic Threshold

Automatic thresholding is algorithmic approach to finding threshold values. Instead of relying on a human operator and static conditions, an algorithm analyses every frame and finds a threshold suitable for one image alone. This should, in theory, indicate a more dynamic ability to extract useful features under varying conditions. As with anything however, this approach comes with its own set of challenges. The algorithm relies on certain assumptions to be true and if those can not be met, problems may arise.
The algorithm used to find thresholds in this project is known as *Otsu's Method*. It is used to create the binary masks from the grayscale intensity images that are extracted from the color layers of a captured image.

Otsu's method works by iteratively going through all possible threshold values and for each step calculate the spread of pixels on either side of the threshold. One side of the threshold represents the background the other represents the region of interest. The threshold value that gives the smallest within-class variance is assumed to be the one that best separates the foreground and background. To give good results this method relies on the assumption stated in the beginning of this chapter - that the color histogram forms a bimodal distribution.
The within-class variance is found using the following equations

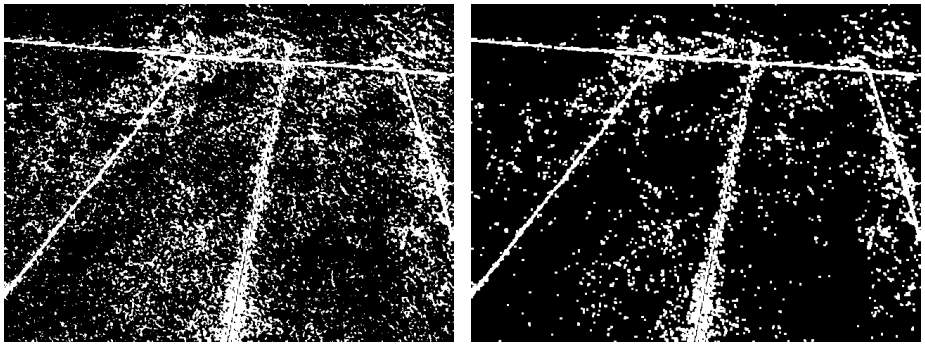
$$\sigma_W^2 = \omega_b \sigma_b^2 + \omega_f \sigma_f^2 \tag{3.2}$$

$$\omega_b = \sum_{0}^{t} p(i) \tag{3.3}$$

$$\omega_f = \sum_{t}^{N} p(i) \tag{3.4}$$

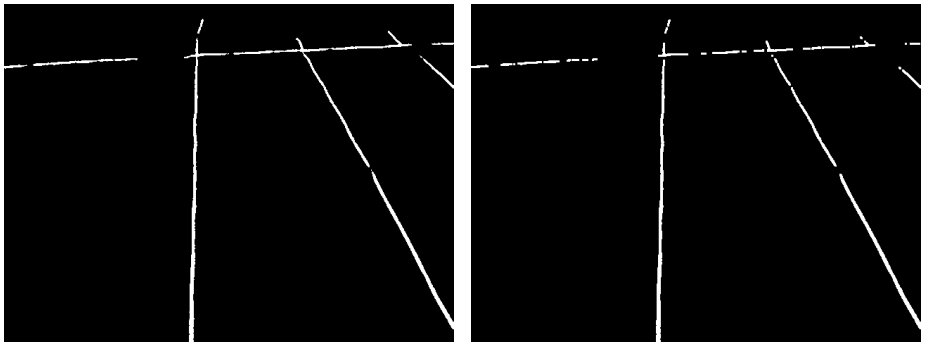
σ_W^2 is the within-class variace. ω_b is the probability of the background pixels, which is all pixels below the threshold t. $p(i)$ is thus the probability of each of the intensity-levels i in the grayscale image. ω_f represents the probability of the foreground pixels, that is the pixels from t to N where N is the highest intensity level, in the case of 8-bit images $N = 255$. σ_b^2 and σ_f^2 are the variances on either side of the threshold respectively.

To demonstrate the ability of the method, the algorithm will be tested on the light and dark image in figure 3.10a and 3.10b respectively.



(a) Unfiltered (b) Filtered

Figure 3.15: The masks of the light image created with threshold values found using Otsu's method



(a) Unfiltered (b) Filtered

Figure 3.16: The masks of the dark image created with threshold values found using Otsu's method

From figure 3.15 and 3.16 it can be seen that the thresholding algorithm manages to give results that are comparable to those found using manual thresholds. The difference is that, while manuals threshold did not necessarily work on subsequent images, this method calculates a new threshold, for every image, that matches the current conditions.

While the use of Otsu's method has some distinct advantages over manual thresholding, it is not without its flaws. Some of those flaws become apparent when the initial assumptions are not met. The following example will show the effects of having an unevenly colored background and little plant material to segment out.
The basic premiss of bi-modality entails that the color histogram should have two peaks, one for the background and one for the regions of interest, in this case plants. If the background consists of more than one color, these will start to form peaks of their own. As Otsu's method only distinguishes between the two most dominant intensity levels, there is no way to guarantee that the plants are the regions that will be segmented out. This effect is enhanced even further if the amount of visible plant material in the image is very small.

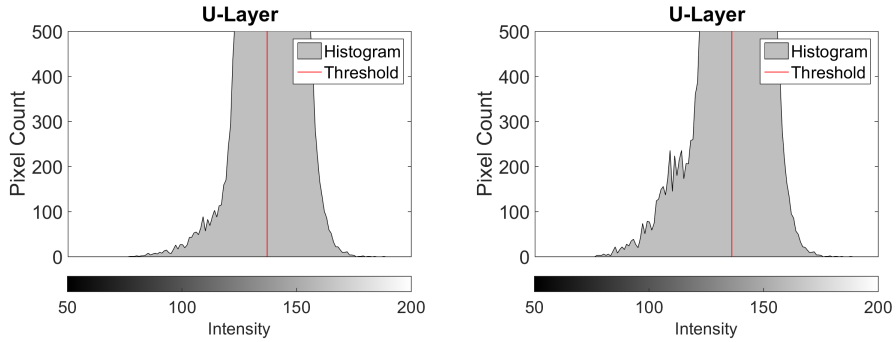
Figure 3.17 displays two images, very similar each other. The only difference is where one image has one row of plants, the other has three. The effects that arise from this difference is rather dramatical in terms of usable features in the final image.
Notice subtle difference in the histograms in figure 3.18. The left peak on the 3-row-histogram is slightly more distinct than is the case with the single row. This small change is enough to shift the threshold and create a vastly different mask. The mask for both the 1-row and 3-row image is shown in figure 3.19. From those it can be seen that in the 1-row case, the varying colors of the background are a more dominating feature than the plants. This results in details in the background being highlighted, while the plants have all but disappeared. When 3 rows of plants are present, they form a large enough peak in the histogram to shift the threshold in a manner that ensures that the plants are indeed the regions highlighted.

(a) One row of plants (b) Three rows of plants

Figure 3.17: The color images of crop rows on cobble stone background.

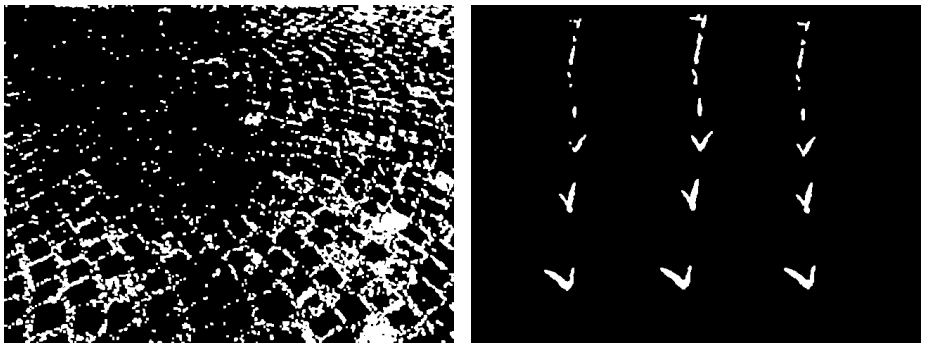
3.2.3 Summary

This section has described two methods for finding threshold values, manual and automatic. The advantages and disadvantages, as well as possible pitfalls, has been explained for both methods. Manual thresholding could provide good results under wide variety conditions, as long as those conditions remain static.



(a) One row of plants (b) Three rows of plants

Figure 3.18: The color histograms of crop rows on cobble stone background.



(a) One row of plants (b) Three rows of plants

Figure 3.19: The black and white masks of crop rows on cobble stone background.

Changes, however, in lighting or other parameters, may give an unpredictable outcome. Automatic thresholding, on the other hand, has a good ability to adapt to changes in the image. This adaptability comes with a price though, as the thresholds need to be calculated for every single frame. This increases the computational demands. Further more, the automatic threshold algorithm relies heavily on a set of assumptions about the image to be met.
While both manual and automatic thresholding have been used during testing, the long term goal should be to use a stable and robust automatic method. One could even argue that until that is the case, the system cannot be considered fully autonomous.

3.3 Filtering

Filtering comes in many forms and can serve a multitude of purposes. In this project the filtering process is applied to the binary black and white mask images that are the outcome of using the found threshold values. The purpose is to reduce the noise in the images. This noise comes in the form specks of back-

ground protruding through the threshold, and smaller pieces of plant material such as weeds. Two morphological operations are used to achieve this noise reduction, they are known as *erosion* and *dilation*.

3.3.1 Erosion

Erosion, as the name implies, reduces the size of white clusters in the image by eroding away at the edges of the cluster. If a cluster is small enough, it can be eroded completely away. This fact is used to remove small, unwanted regions in the image while maintaining larger features, although they do suffer from a slight reduction in size.
The erosion filter works by convoluting the image with a *kernel*. The kernel used is a 3-by-3 square matrix with an *anchor point* in the center. By scanning the anchor point over the source image, the corresponding position of the anchor point in the destination image is given the minimium value covered by the kernel. This operation is illustrated in figure 3.20.
The amount of erosion performed on the image is dependent, not only on the size and shape of the kernel, but also on how many times the filter is applied. The implementation of the filter in this project uses a fixed size kernel, but the number of iterations can be set to any integer value.
Figure 3.21 shows the before and after effects of the erosion process. Notice how any small features in the original mask has disappeared in the eroded image. The larger features remain and will be subject to the subsequent dilation process.

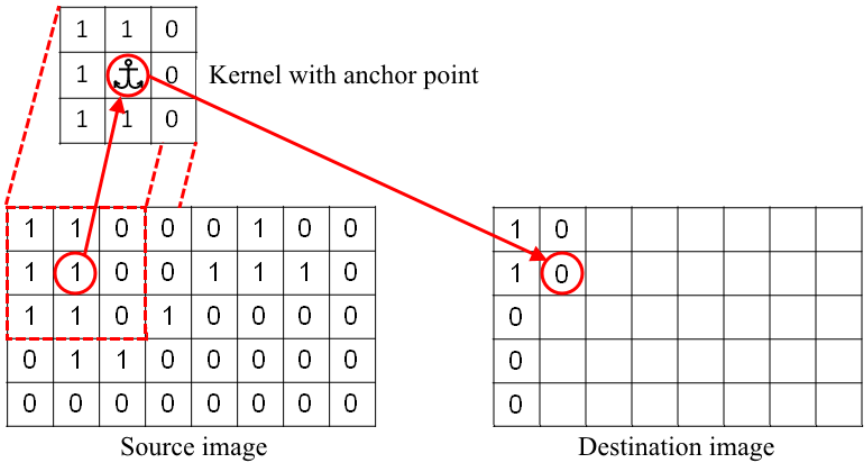
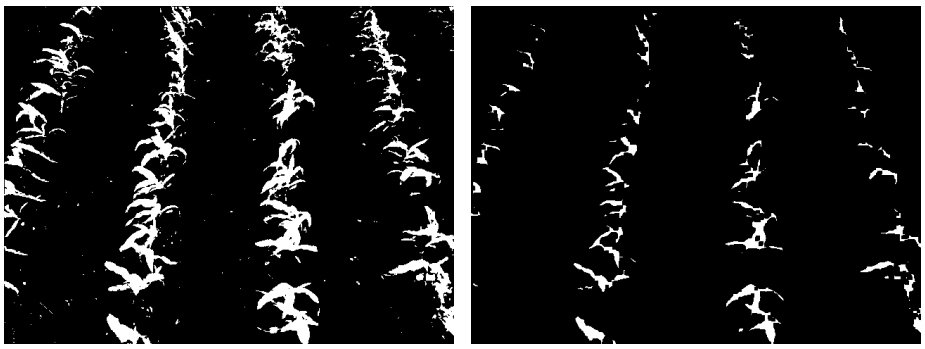


Figure 3.20: The erosion process performed using a 3-by-3 kernel with its anchor point in the center

3.3.2 Dilation

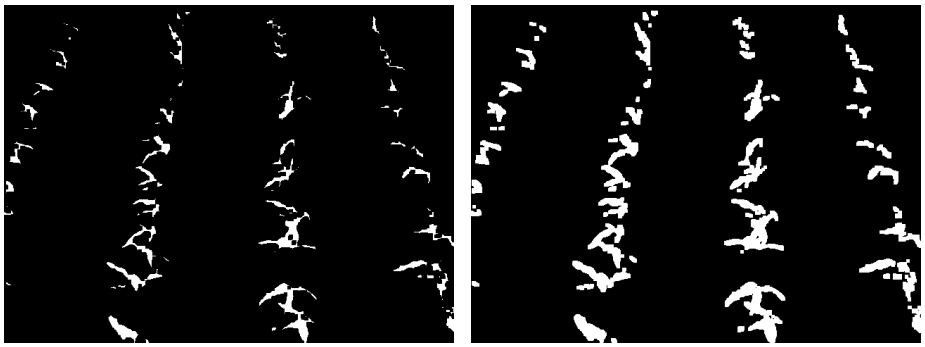
Dilation is closely related to erosion and can be said to have the opposite effect. It is applied in the same manner as erosion, but where the erosion process



(a) Original masked image (b) Mask after the erosion process

Figure 3.21: The before and after of the erosion process

carried over the smallest value covered by the kernel, dilation uses the maximum value. This means that instead of shrinking the visible clusters, it grows them outwards. This morphological operation is performed after the erosion to restore the remaining features to something closer to their original size. As with the erosion, the amount of dilation is controlled by the number of times the filter is applied, as the kernel size is fixed.
The effect of dilation can be seen in figure 3.22.



(a) Eroded mask (b) Mask after the dilation process

Figure 3.22: The before and after of the dilation process

3.4 Blur

Blurring is the act of averaging out an image. By making all pixels appear more like their neighboring pixels, the image is smoothed out. This effect can be used for several reasons, but in the context of this project it has a very specific purpose. As mentioned in section 3.2.2 about automatic thresholds, there exists an inherent sensitivity to color variations in the background. In that section it was described how increasing the ratio of plants to background could alleviate this problem. That is not the only thing that can be done however. By blurring

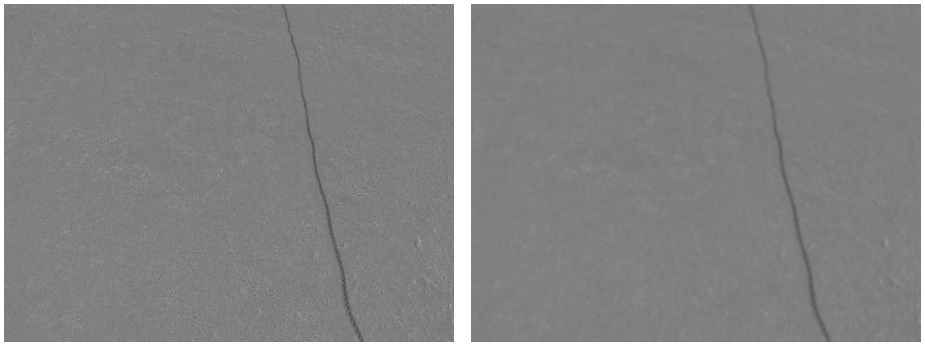
the image, subtle differences in color blends together, creating a more uniform layer. This makes it easier, using Otsu's method, to distinguish between the background and foreground.

The way blurring is applied very similar to the way erosion and dilation is applied. A kernel is convolved with the source image and the pixel values for the destination image can be calculated. But where both erosion and dilation works on binary images, blurring is used on grayscale images. For 8-bit images, this means that the values can range from 0 to 255. Additionally, the kernel itself is different. It is also a 3-by-3 matrix, but the value in the anchor point is calculated as the average value of all the elements covered, including the anchor point itself. The kernel size is fixed, but the blurring can be applied multiple times. The kernel used is given by:

$$Kernel = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.5}$$

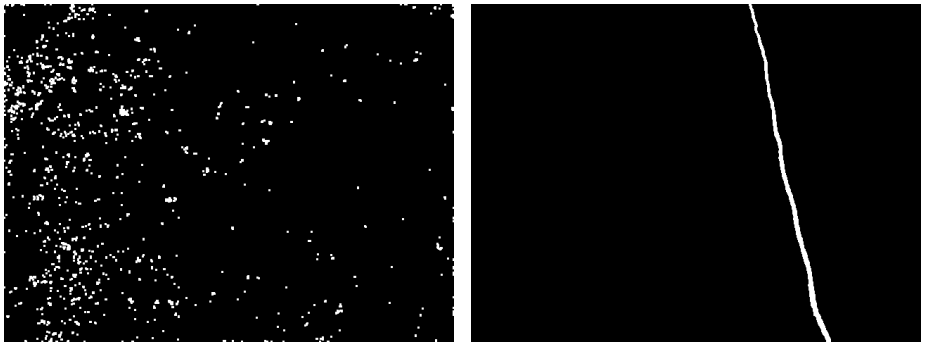
To demonstrate the effects of blurring the image in figure 3.23 will be used. It shows a scene where a blue rope is to be separated from a grassy background. The problem in this situation is that the grass is not uniformly green, and it is littered with brown leaves. To simulate a scenario where the regions of interest occupies a very limited part of the overall picture, no more rope is added. Instead it can be established how blurring the image can help mitigate the negative effects of the uneven background.
Figure 3.24 shows the U-layer in a blurred and non-blurred version. While it can be difficult to see major differences between them, the effect when finding the threshold is very apparent. Figure 3.25 shows the resulting binary masks. The one based on the unblurred image has picked up details in the background, and the rope is completely lost. The blurred image, on the other hand, clearly separates the rope from the background.

Figure 3.23



(a) Zero blurring (b) Three iterations of blur

Figure 3.24: The U color layer with zero and three iterations of blur



(a) Zero blurring (b) Three iterations of blur

Figure 3.25: The resulting binary mask with zero and three iterations of blur

Chapter 4

Hough Transformation

After the segmentation of the image is complete, what is left is a single black and white binary image. This image, if all went well, contains the plant material in white and everything else has been removed and is colored black. For humans it has been relatively easy through most, if not all, of the steps to recognize the plants and find that they form rows. For a computer however, even at this point, it is completely oblivious to any patterns that might occur in the image. The steps so far has merely served to form a foundation that will enable the computer to search for such patterns. Since the goal is to find rows of plants, an algorithm that searches for line features will be used. This algorithm is the *Hough transform*.

4.1 Principle of operation

The hough transform works by assessing each white pixel in the image. For every point a line is drawn through it, expressed in polar coordinates using ρ and θ. θ is the angle of the line and ρ is the shortest distance from the origin to the line and is calculated using the following formula:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{4.1}$$

This is done multiple times, each time with a new angle as shown in figure 4.1. By storing the values of ρ and θ in what is called the accumulator space, a sinusoidal curve will appear for every point. Where multiple points lie on the same line, the sinusoids will cross each other in the accumulator space. This is demonstrated in figure 4.2. In the accumulator space two curves representing each of the points is present. The crossing point, highlighted with a red square, is at exactly 0 degrees. This is what would be expected since only a vertical line goes through both points. To expand a bit further on the principle, figure 4.3 shows an example with three lines. One vertical line and two lines at 45 and -45 degrees respectively. The areas with the highest intensity - the most line crossings, are highlighted with red squares.

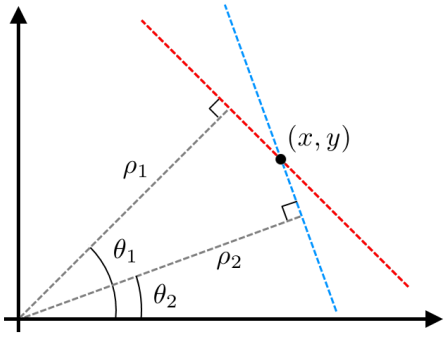
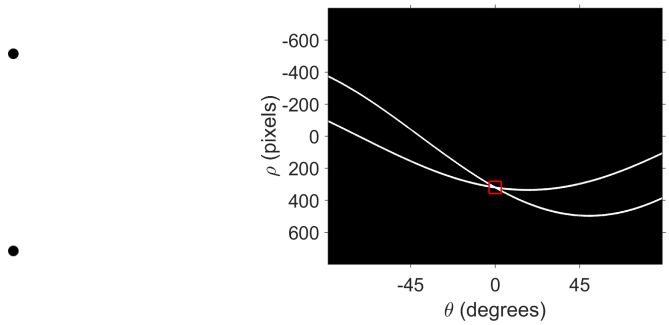


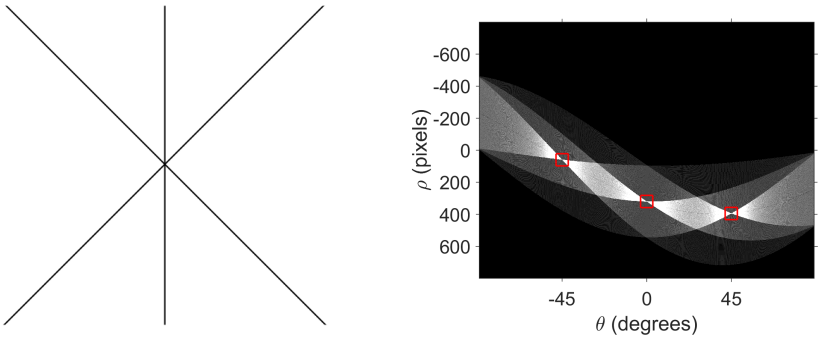
Figure 4.1: The ρ and θ parameters represent the line in polar coordinates.



(a) Points to be transformed (b) Hough accumulator space

Figure 4.2: Each of the points in the image corresponds to a sinusoidal curve in the accumulator space. The parameters where two such curves cross each other indicates a line going trough both points.

The strength of the hough transform comes from the fact that if something resembling a line exists in the image, it can usually be found - even in very noisy conditions. The range and resolution of the θ parameter is used to determine how finely the image is transformed. If the lines are relatively thick, as is usually the case when looking at crop rows, the resolution can be reduced, thus lowering the computational demands.
As the hough transform is an algorithm that performs several actions, multiple times, on every white pixel in the image, it is generally a rather computationally intense process. The total computational time increases with the number of pixels that has to be examined. This relationship between the number of pixels and the computation times proves to be almost linear. This can be seen in the chart in figure 4.4. In that example the hough transform were used to find three lines of increasing thickness.



(a) Lines to be transformed (b) Hough accumulator space

Figure 4.3: As more and more points are found to belong to the same line, the intensity at the crossings increase. The most intense areas are highlighted with a red square.

4.2 Practical Implementation

As the transformation completes, the most distinct lines in the image makes up highest peaks in the accumulator space. By plotting the lines defined by the parameters at these peaks, something potentially unfortunate reveals itself. Each row of crop does not correspond to a single peak in the accumulator space, but rather a collection of such peaks. This is illustrated in figure 4.5.
A collection of lines are clustered on the most visible crop rows. To simplify subsequent calculations, it would preferable to have a single line represent the crop row it covers. One solution to this problem would be to determine which line belong in what cluster. Once the clusters had been identified an average line could be calculated.
Another solution however, and the one used in this project, is to change the underlying image by removing the pixels that make up the crop row. Once the strongest line has been identified, all white pixels, within a certain width of the line, are painted black. Now the hough transform can be applied again, finding the second strongest line. This process can be repeated either until exhaustion, or until a specified number of lines have been found. The effect can be seen in figure 4.6. Notice how the strongest peak in the accumulator, highlighted by the red square in figure 4.6a, has disappeared in the second iteration.

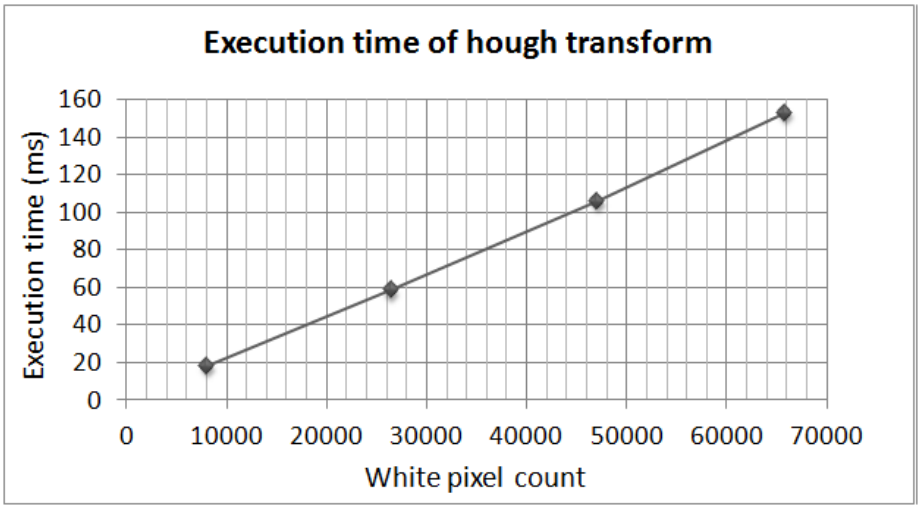
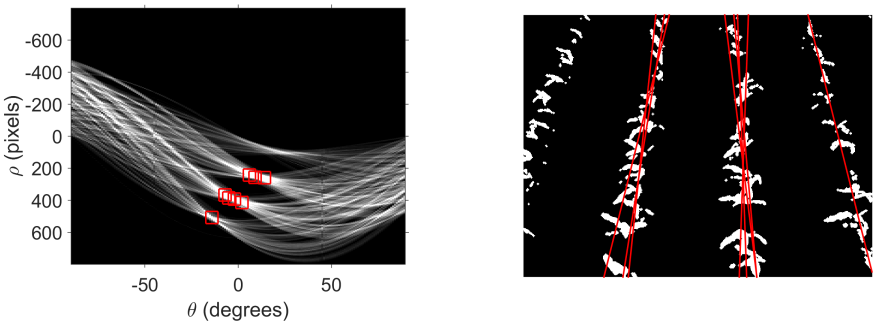


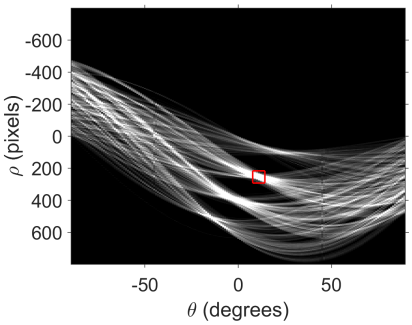
Figure 4.4: The relationship between number of white pixels and execution time.



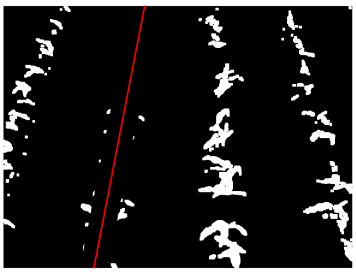
(a) Hough accumulator space

(b) Resulting lines

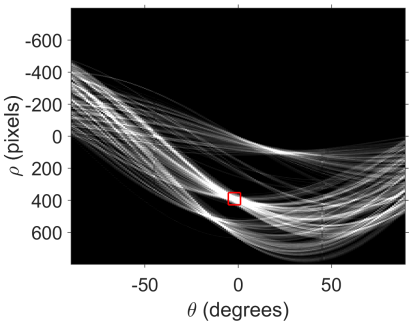
Figure 4.5: Multiple lines found on each row of crops



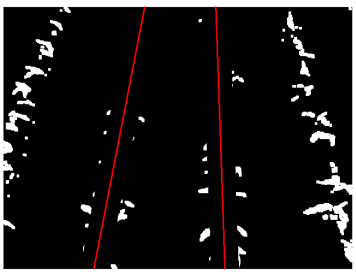
(a) Accumulator space. 1. iteration



(b) Resulting lines. 1. iteration



(c) Accumulator space. 2. iteration



(d) Resulting lines. 2. iteration

Figure 4.6: Crop rows removed as lines are found

Chapter 5

Implementation

All the preprocessing steps required for the image segmentation, as well as the line feature extractor, realised using the hough transform, are made available by implementing them in the form of a Mobotware plug-in. That plug-in however, contains other functionalities, most notably an *end of row*-detector as well as the means for finding a reference point to steer the robot after. And while all this essential image analysis is carried out solely by the plug-in, it outputs only the resulting values. The actual utilization of the data obtained is done separately in an MRC-script. This script receives the data from the camera server and uses it to control the robotic hardware platform.
These additional functionalities, as well as the MRC-script, will be explained in this chapter.

5.1 Following The Rows

So far the focus has been on enabling the system to detect the crop rows and estimating their position using a line. The purpose of this, of course, is to ultimately use the information gained to facilitate the navigation of a robotic platform between the rows, without damaging the plants. The calculations necessary for finding such a *guide point* are based on a set of simple rules.

- If exactly two rows are present, the guide point is assumed to be in the center between the rows.
- If three or more rows are present, the two rows closest to the center of the image are used.
- If only one row is present the guide point is placed a predetermined distance from the row. If the row is in the left half plane of the image, the guide point is placed right of it and vice versa.

These rules ensure that the guide point is placed as expected in most situations. It is not perfect though, and in certain scenarios wrongly placed guide points are

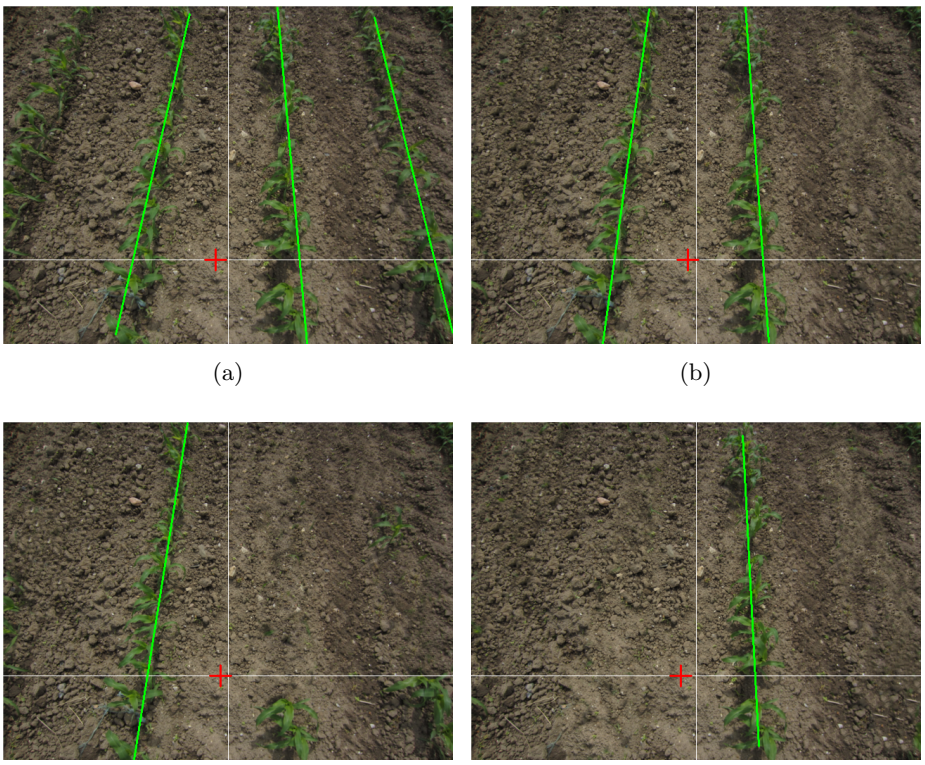
the unfortunate outcome. Both cases will be addressed in following examples.

Figure 5.1 shows four different scenarios, all of them yielding acceptable results. The guide point is placed safely between the rows when there are multiple present. In the case of single rows, the guide point is placed at a distance similar to that seen with more rows, and the side to which the point is shifted is also correct.
On the other hand, figure 5.2 illustrates the kind of situation where the rules creates an undesirable guide point placement. It simulates a case where the center right crop row is missing enough plants to remain undetected. The plug-in now calculates a new placement for the guide point according to the rules - in the center between rows. In the best case scenario the robot runs over a few plants, but otherwise returns to the correct row once the plants appear again. It might however end up in the wrong row, which means either the robot is skipping a lane, or doing one twice. Both of these things should be avoided.

In an effort to reduce the risk of placing the guide point wrongly, in the case of missing rows, a simple feature has been implemented in the plug-in. Instead of taking all the lines found in the image into account, when calculating the guide point, a limitation can be imposed. The limitation is implemented as a maximum distance from the center of the image to the line. If a line lies outside of this distance, it will not contribute to the calculation of the guide point. The point on the line which is used to determine its distance to the center, is the intersection between the line itself and a horizontal line placed at a certain height in the image. The horizontal line is represented in white on figure 5.1 to 5.3.
Figure 5.3 shows the imposed limitations drawn in as red vertical lines. It can be seen that the outermost row is now ignored, and the guide point is placed at a more reasonable position between the rows.
This approach works well when the robot is driving straight and the rows are in relatively predictable positions. However, if the angle of attack is skewed in either direction, a line might unintentionally be cut out, resulting in poorer performance. This situation is illustrated in figure 5.4. Notice how the guide point is shifted slightly to the right when the margins of the limitation is widened to include the extra row.

5.2 Detecting Headland

Headland is a piece of land surrounding all sides of a field. It is there to ensure that the machinery used in the field can turn around at the end of the rows, without disrupting the rows themselves. This means that for a robotic platform to perform multiple runs through a field, it needs to be able to detect the headland. Once the headland has been detected, the robot would know it is time turn around.
One of the main challenges lie in the fact that the headland can be any one type out of many. Some farmers prefer to leave it as bare soil, others plant rows of a different orientation while some choose to plant a completely different crop altogether. The goal in this project is to be able to detect two such types. The



(a) (b)

(c) (d)

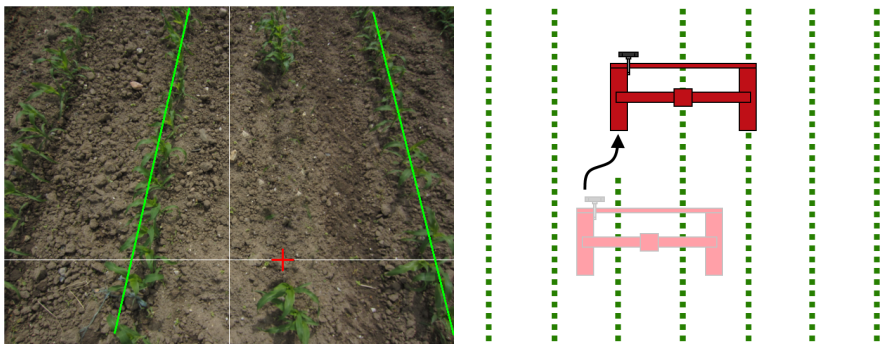
Figure 5.1: Different scenarios with good guide point placement

types chosen are the bare soil, and crossing rows.

5.2.1 Crossing rows

In this type of headland rows of crops are planted around the perimeter of the field. They planted perpendicular to the rows in the main field , and the headland is typically a few meters wide. The advantage of this type of headland, with respect to an automated detection system, is the fact that the same line feature extraction scheme that calculated the guide points, can be reused. Instead of searching for vertical lines to steer the robot, horizontal lines are found to indicate the end of the rows.

Two examples of detecting crossing rows can be seen in figure 5.5. The first image shows a single crossing line being detected, while the second shows two. In the case of multiple lines, the one closest to the robot is assumed to be the start of the headland.



(a) Missing row. (b) Robotti running over crops.

Figure 5.2: Wrong placement of the guide point can mean plants will be damaged or the robot may get lost.

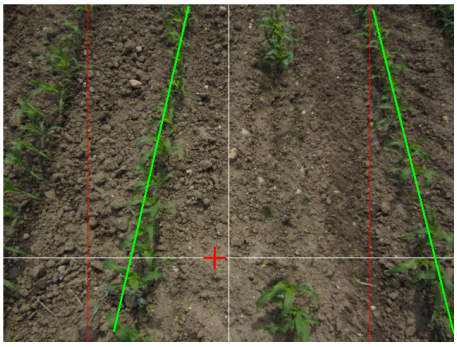


Figure 5.3: Limiting the area in which lines contribute to calculating the guide point.

5.2.2 Bare Soil

In this type of headland, as the name suggests, the rows end in nothing but bare soil. A simple approach to detect the beginning of this type of headland, would be to simply set a minimum length of the lines found, and when that length has been reached, assume the end of the row has been found. This solution is perfectly viable so long as the rows in the field are fairly uniform. An example where this could work is displayed in figure 5.6a. The estimated lines end where the crop rows does, and so the end points would be a good indicator of the headland. Unfortunately though, completely uniform rows, with no missing plants, cannot be guaranteed. Figure 5.6b shows an example where the end point of the estimated lines would not necessarily be a good indicator of the headland.

The method used for detecting bare soil headland in this project is slightly more intricate than the one described above. Instead of simply using the line end points in conjunction with line length, it uses the lines to overlay a series of virtual rectangles on top of them, spanning the entire height of the image. Inside each of the rectangles, the total number of white pixels is counted. Once

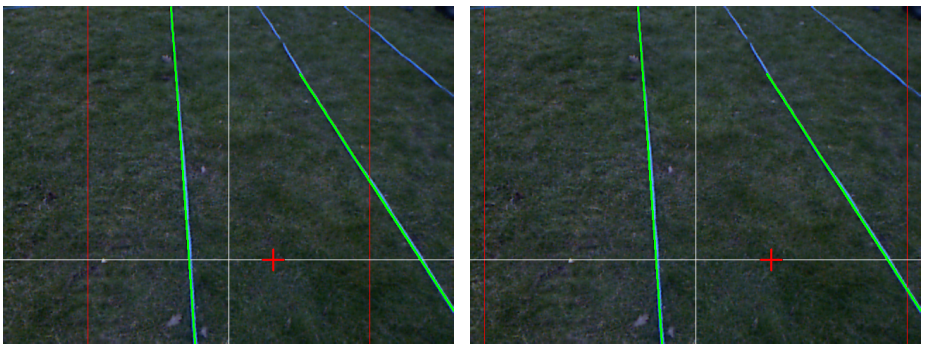
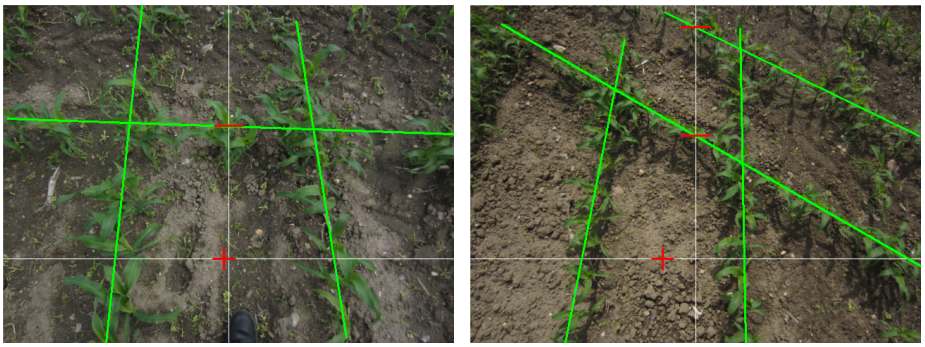


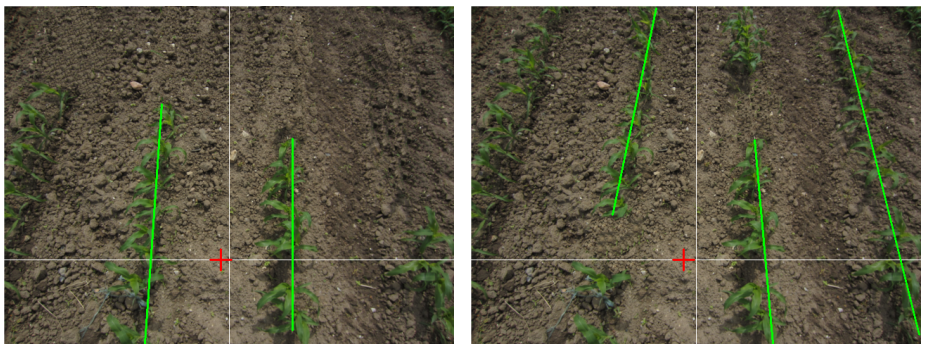
Figure 5.4: The effects of skewed angle of attack using distance limitations.



(a) Single crossing row (b) Multiple crossing rows

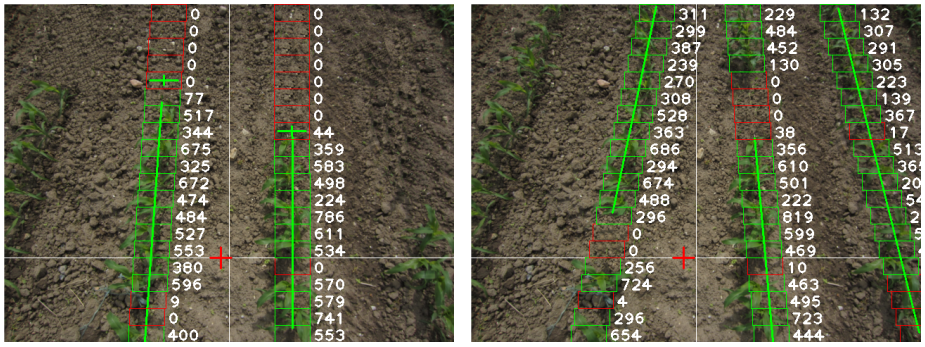
Figure 5.5: Detection of both single and multiple crossing rows.

this have been counted for all of them, an average amount of pixels is calculated. By comparing the number of pixels inside each individual rectangle with the average value, it can be determined whether or not it is covering enough plant material to be part of the row.
Figure 5.7 shows the same two images as in figure 5.6, but with the rectangles drawn in along with the number of pixels inside. Green rectangles contain plant material, whereas the red ones are considered empty.
By setting a requirement on the number of consecutive empty rectangles needed before a row is considered to have ended, it is possible to avoid the problems faced by missing plants in the row.
A green cross is drawn inside the rectangle that signifies the end of the row. In figure 5.7a it can be seen that two such crosses are present and placed correctly. In figure 5.7b, on the other hand, it is assumed that the row continues despite the apparent holes in the rows.
The effect is achieved because the array of rectangles does not end with the estimated line, but rather continues along the path for the entire image. This gives a more robust indication of when to expect the headland.



(a) (b)

Figure 5.6: Using the end points of the estimated lines may or may not be a good indicator of the headland.



(a) (b)

Figure 5.7: Headland detection using pixel counts in discrete rectangles.

5.3 Navigating Headland

When the robot is operating in the field, the steering is constantly being corrected by means of visual feedback. The crop rows it is driving between provides a reference point for navigation. This is true up until the point where the headland is detected. Due the many different types of headland, a predictable and suitable reference for visual guidance cannot be assumed to be present.
This means that from moment the robot enters the headland an alternative method for navigating the robot is necessary. The method chosen for this task is odometry. Odometry can give an estimated position relative to some starting point, but has the unfortunate side effect of being incredibly susceptible to accumulative errors. The reason this can still be an acceptable solution is due to the short distances over which it is used.
Figure 5.8 shows the path that the robot has to follow in order to return to the field and be situated in the next row.

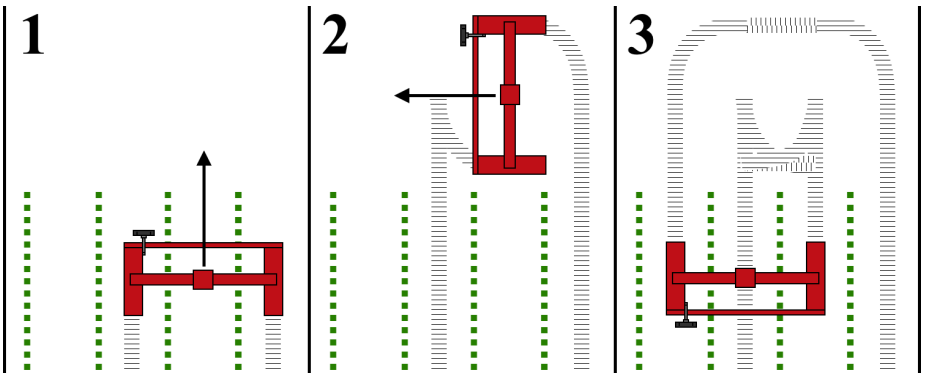


Figure 5.8: Robotti navigating the headland

5.4 The MRC-script

As mentioned in the introduction of this chapter, the image analysis is carried out by the plug-in while the MRC-script manages the behaviour of the robot. The script receives information from the plug-in and uses it to steer the robot. It is based around a main loop that has three primary functionalities:

- Calculate target coordinates relative to the robot position
- Check if headland has been detected
- Issue *drive* commands

The drive command function is part of Mobotware, and it works by steering the robot towards a line defined by a set of coordinates and an angle. When the script is running the coordinates for the guide point are received from the plug-in. Those coordinates are are then manipulated by the script to form a line that goes from the robot itself to the guide point. This is illustrated in figure 5.9. The blue arrow represents the line that the robot tries follow. It points towards the guide point that was received from the plug-in. As the robot closes in on the line, the position of the guide point is updated - reflecting the new situation.

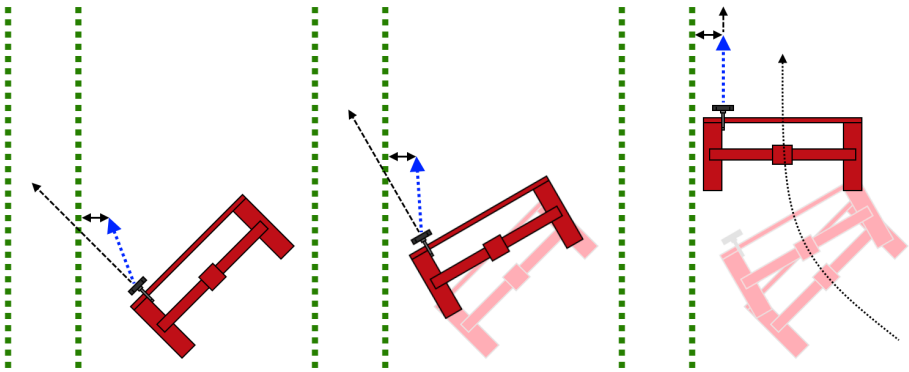


Figure 5.9: Robotti approaching the crops

Chapter 6

Results

This chapter contains some additional results obtained through test runs with two robotic platforms. One is the *iRobot* platform the other is the *Vibro Crop Robotti*. Both platforms have been set up to be as similar to each other as possible. This is particularly true for the camera placement, as that is the primary source of data, as well as it being somewhat sensitive to position changes. Table 6.1 below shows the position of the camera on both platforms.

		iRobot	Robotti
X position	[m]	0.65	0.70
Y position	[m]	0.0	0.0
Z position	[m]	1.15	1.15
Rotation ω	[rad]	0.0	0.0
Rotation ϕ	[rad]	0.733	0.733
Focal Length	[pixel]	551	551

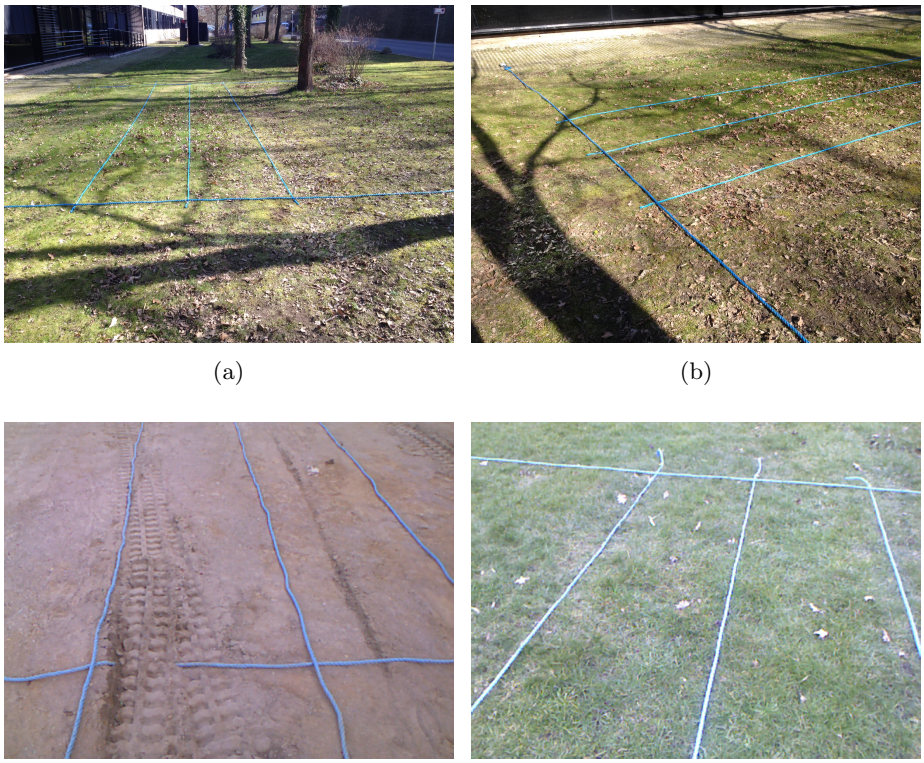
Table 6.1: Camera parameter of two platforms

The similar camera positions would imply that the behaviour of the plug-in, and the calculation of the guide point, would also be very similar.

6.1 The Field

The purpose of the system developed during this project is to enable a robot to navigate autonomously in a field using visual feedback. To test the system and demonstrate its ability to fulfil this purpose, an actual field with the proper crops growing in it would be an optimal testing site. However, due to a lack of transportation options for the robots, as well as seasonal challenges with regard to the growth state of certain plants, it has not been possible to verify the system under such optimal conditions.

The solution to this predicament has been to substitute planted rows of crops with something that may be different in appearance, but should be similar in characteristics.
Blue ropes are used to make up the crops, and are placed with the correct distance between each row, in this case $0.75m$. The background has been either grass or a combination of sand and gravel.
Figure 6.1 displays the field layout on various backgrounds.



(a) (b)

(c) (d)

Figure 6.1: Field layout with different background

6.2 iRobot

The iRobot is the platform that has seen the most extensive testing. This is mainly due to its availability in the early stages of the project where the Robotti was not in a working state. Further more, its smaller size allows for a single person to carry out the tests, and to do so on a much smaller testing area.

The test data obtained is comprised of odometry data from the robot itself and from camera data gained by the plug-in. A GPS-module is also mounted on each of the robots, however, the data retrieved has proven unhelpful. The reasons for this, be it poor connection or the short distances over which the robots have been driving, has not been pursued further.

Figure 6.2 shows an xy-plot of the odometry data from a run with three rows. The test field used can be seen in figure 6.1a. The starting point of the robot is in $(0, 0)$ and, as expected, the path is rather skewed due to odometric errors accumulating, but otherwise represents the field reasonably well.
The overlaid camera data is not meant as a precise portrayal of the calculated guide points with respect to the odometry, but is supposed to give an overview of where certain spikes occur during the run. The reason it can not be deemed accurate is because the angle and heading of the robot has not been taken into consideration. From the overview it can be seen that the largest spikes takes place right before the robot enters the rows again. At this point the robot has been driving and turning using only odometry, and as such this phenomenon is not entirely unexpected.

To get more insight into the camera data obtained from the plug-in, figure 6.3 holds all the samples from the run. The blank spaces appear when no suitable line information could be found. This happens at the end of each row and continues until the robot is facing towards the field again.
To get a sense of the accuracy of the robot while it is driving along straight paths, which it will be doing most of the time, a frequency distribution plot has been made which can be seen in figure 6.4. To avoid the spiky outliers near beginning of the rows, the plot only contains the samples within -0.1 and 0.1. This corresponds to 91.2% of all the samples available. It can be seen that most of them lie well within these boundaries.

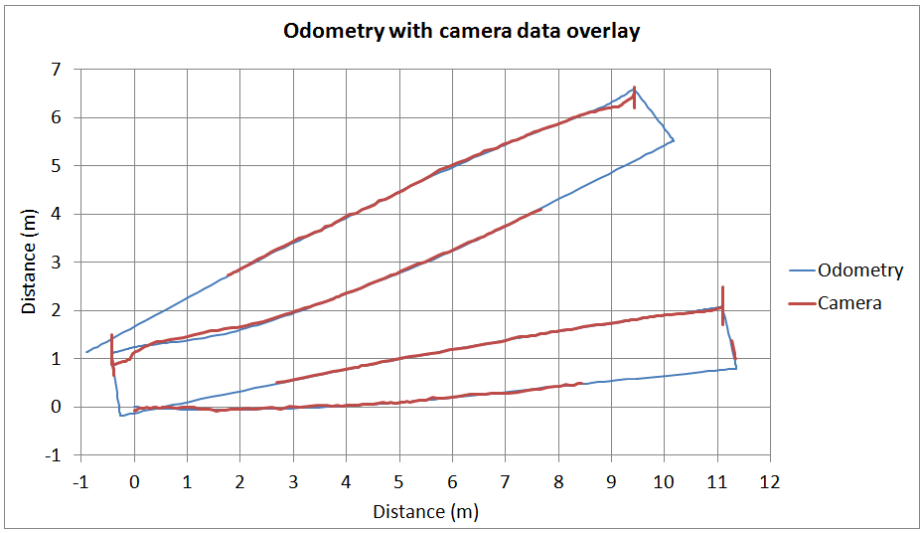


Figure 6.2: xy-plot of odometry data with camera data overlay.

To get an even more detailed look of the performance at the straight part of the path, the samples from the beginning of the run until the first turn have been plotted in figure 6.5. From the plot it can be gathered that the *mean value* is $0.00258m$ and the *mean absolute deviation* is a mere $0.02301m$.
Figure 6.6 is meant to demonstrate how the robot settles in during the initial

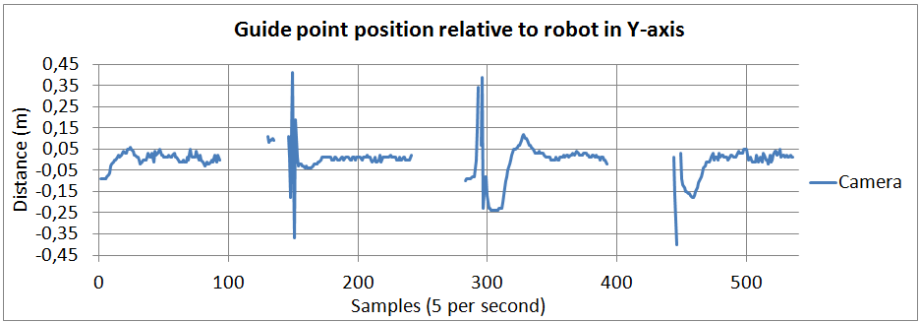


Figure 6.3: Camera data from the plug-in.

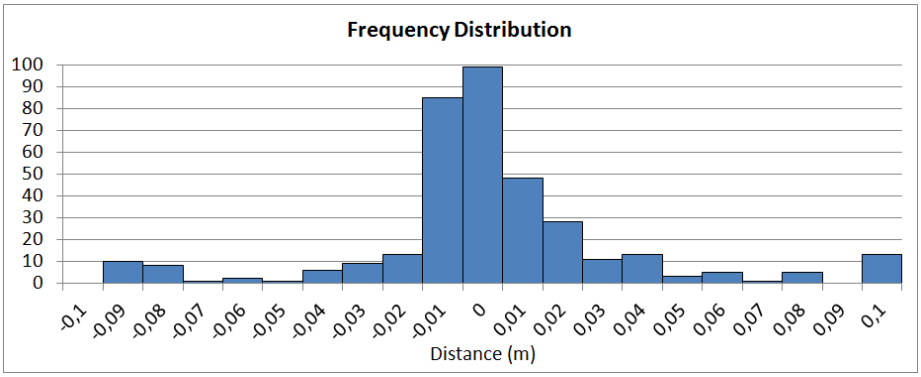


Figure 6.4: Frequency distribution from -0.1 to 0.1 meters (91.2% of samples).

spiky measurements when facing the row after a turn.

The plug-in handling the data from the camera dose not only yield a guide point, but also outputs the distance from the robot to the end of the row when possible. Figure 6.7 shows the detection of the end of the row overlaid on the guide point values. Notice how the end is detected prior to each turn.
Figure 6.8 shows the progression of the end of the row before the first turn. The end of row is first detected at a distance of $2.83m$ in front of robot, and is lost again at $1.27m$.

6.3 Robotti

The Robotti is the platform for which the guidance system has been developed. While the data obtained by the iRobot is interesting in its own right and has served as proof of concept, it is the performance of the system on the Robotti that matters the most. Unfortunately, due to the unwieldy nature of the platform as well as host of hardware complications, the testing has been somewhat limited.

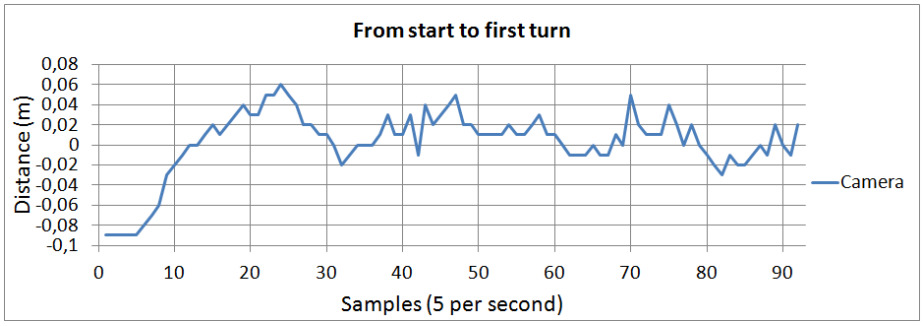


Figure 6.5: Measurements from the starting point until the first turn.

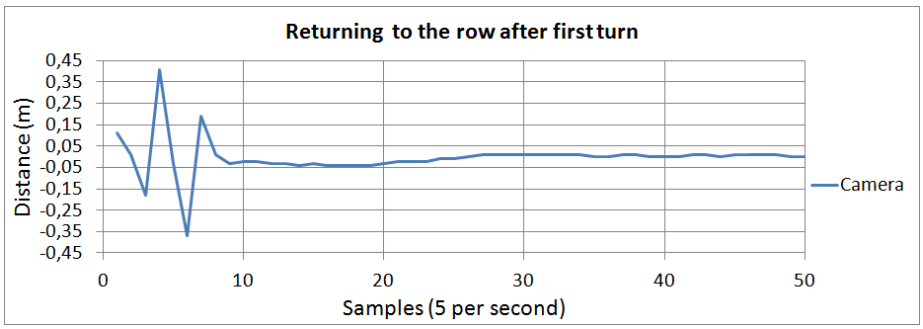


Figure 6.6: Demonstration of how the guide point values, and the robot, settles in after a turn using odometry.

In order to get a sense of how the Robotti responds to the guidance system, a test was performed that had the robot drive between two straight rows. The rows were approximately 20 meters long, and was made up of two ropes. Two separate test sessions were carried out, each with a different starting position. In the first run, the robot was placed at an angle such that it had to make an initial turn in order to end up between the rows. At the start of the second run, the robot was positioned as close as possible to the center of the rows.

For a comparison between the two runs, the camera data from the guidance system have been plotted for both in figure 6.9. In the first run it can be seen that the system starts out by estimating that the robot should aim at a point offset by 30 cm. While this distance is quickly reduced, an overshoot is introduced that peaks at -13 cm.
In the second run, the initial offset is 8 cm and this is reduced with no overshoot. There is no apparent difference between the runs once they have both reached steady state.

To get a better view of what happens during the settling time of the first run, see figure 6.10 that contains two plots. Plot (a) displays the odometry data from the entire path, and while the effects of the overshoot is visible, more details is present in plot (b). In plot (b) it can be seen that, even though the overshoot, as it is determined by the camera data, is -13 cm, the overshoot in

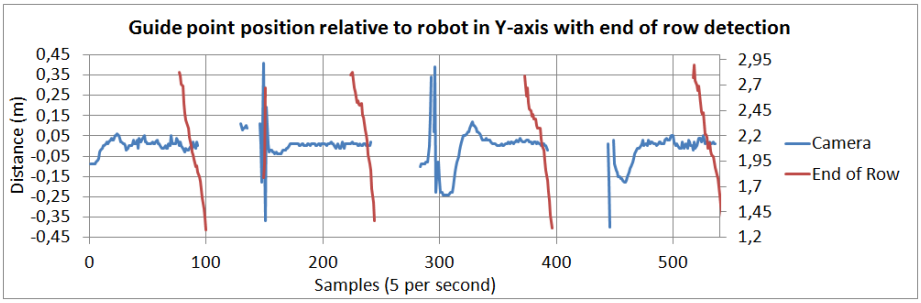


Figure 6.7: Camera data from the plug-in including end of row detection.

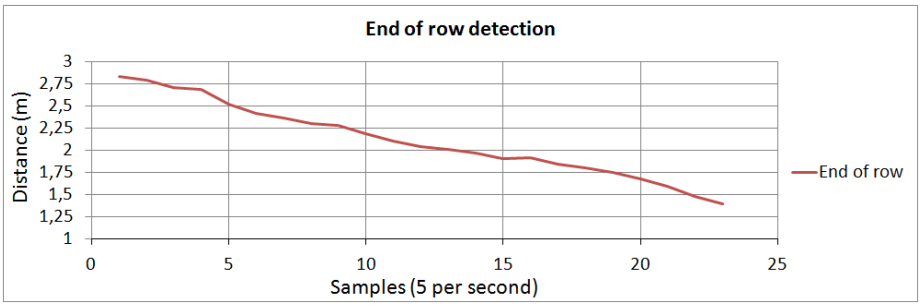


Figure 6.8: First end of row detection

the odometry appears to be much smaller. This observation can be reinforced further by looking at video evidence found on the included CD-ROM.

To get an insight of the steady state performance, a subset of samples have been extracted from the camera data. The subset starts at the 100'th sample, and continues to the end. At the starting point of the set it is assumed that the robot is fully in steady state for both runs. The set can be seen in figure 6.11, and the frequency distribution of the set is seen in figure 6.12.
To get an overview of the performance, the data gathered have been collected in table 6.2.

	Mean value	Mean absolute deviation
Run 1	0.0075 [m]	0.0135 [m]
Run 2	0.0088 [m]	0.0127 [m]
Combined	0.0082 [m]	0.0131 [m]

Table 6.2: Accuracy parameters of the two runs.

The Robotti platform was also tested under conditions that more accurately resembles an actual field. Again, ropes was used to represent the crop rows, and the layout can be seen in figure 6.1c. An xy-plot of the odometry data can be seen in figure 6.13. The starting point is in $(0, 0)$ and while the lines appear

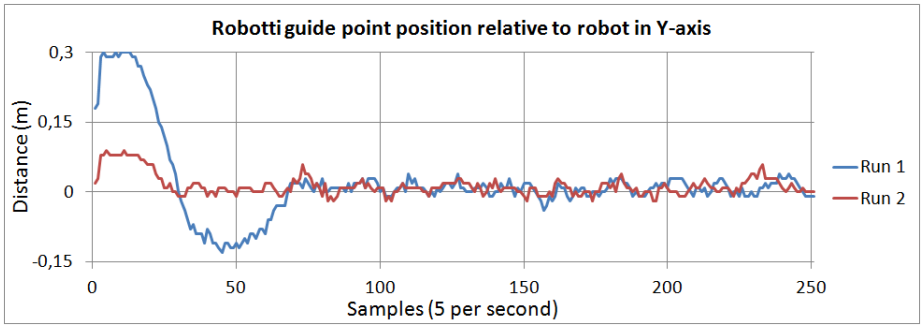
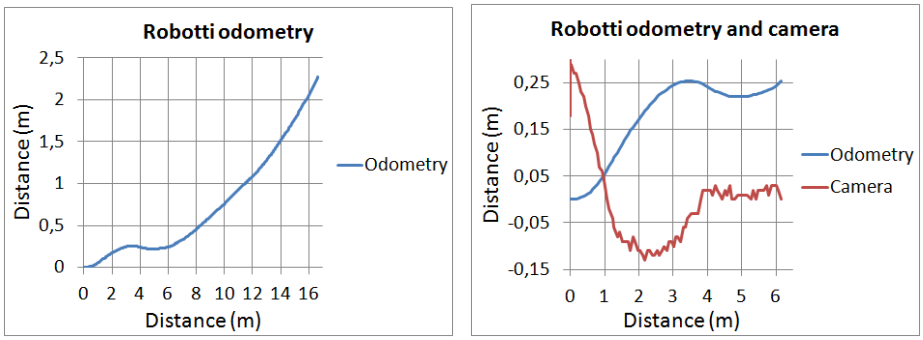


Figure 6.9: Measurements along a 20 meter straight path.



(a) (b)

Figure 6.10: Plot (a) is the odometry for all the samples. Plot (b) is the first 100 samples of both odometry and camera data.

skewed, it is possible to see that a row was fully circumnavigated.
Due to a series of unfortunate circumstances, the data provided by the guidance system during the test is regrettably not available. As such, no conclusions of its performance can or will be drawn. A video of the test can be found on the included CD-ROM.

6.3.1 Final thoughts

While it is possible to have the Robotti running with Mobotware and all the features it include (all the servers, modular plug-in's, joystick control, etc.), the operation of the hardware is still in a state that leaves much to be desired. The interface with the Roboteq controllers, in particular, seems to be causing some problems. A variety of techniques have been tried to increase the robustness, among those is position control, closed loop speed control and a combination of the two, but none of them resulted in a truly satisfactory behaviour.
This is also evident in the video material, where the robot can be seen struggling with completing turns. The robot hardware is capable of speeds in excess of 12 km/h and should have plenty of power for the tasks performed during testing, but this potential is never realised. Overall, the hardware was able to be used

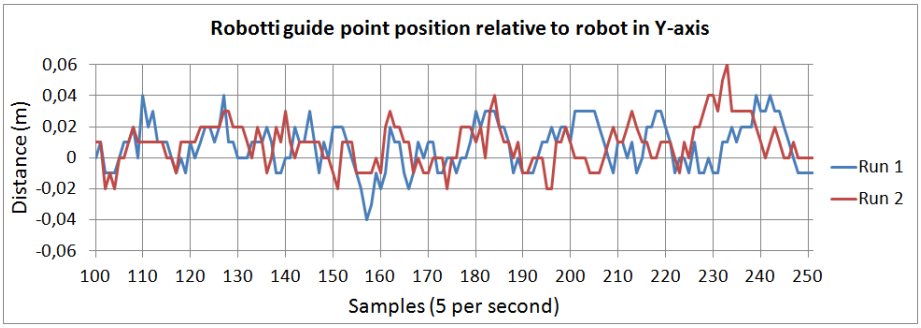


Figure 6.11: Steady state measurements (from the 100th sample and onwards).

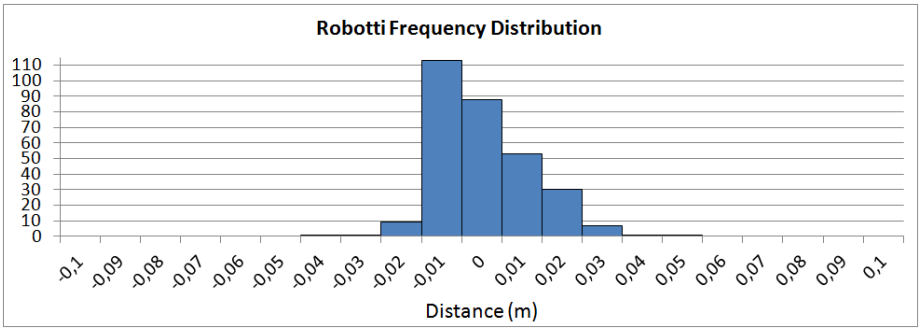


Figure 6.12: Frequency distribution of the steady state measurements.

in order to validate the visual navigation system developed for it, but there is lots of room for improvement.

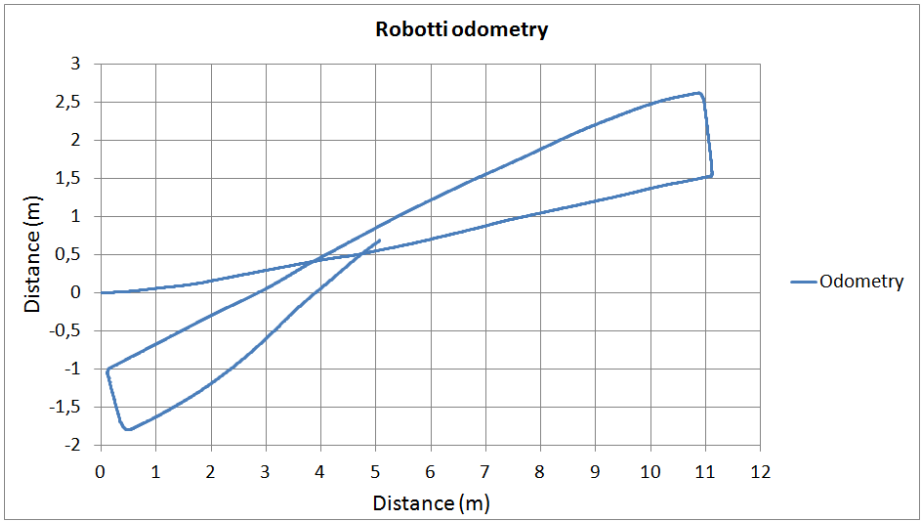


Figure 6.13: xy-plot of Robotti odometry data from a test run on field layout.

Chapter 7

Further Work

This chapter will contain a brief discussion of the further work that would benefit the overall system and implementation. It is not meant as an exhaustive list, nor is it definitive.

Image Segmentation

The image segmentation process is a large part of the project and therefore also has a lot of potential to become more refined. The overall goals of being able to detect crop rows is fulfilled, but the robustness can be improved. Particularly the automatic thresholding, using Otsu's method, is a feature that could benefit from further development.

Another area that could increase the practical application of the system, if improved, is the headland detection. Two types of headland have been covered in this project, but many other types exist.

Implementation

The implementation of the image analysis using a plug-in is a very agile approach. The MRC-script on the other hand, suffers from being written in a language that, at times, can seem utterly inadequate for the task at hand. While it does contain some useful features, primarily the *drive* command, and is not impossible to use, a more reasonable means of programming the robot must be conceivable.

Testing

Testing of the system has been performed using a multitude of plant and crop row substitutes. These solutions are great during development, but they can not quite make up for testing in an actual environment in which the robot is to work. Therefore, validation of the system under real conditions would be an obvious next step.

Chapter 8

Conclusion

The overall goal of developing a system that enables a mobile robotic platform to navigate autonomously in a field of crop rows using visual feedback, is considered to be fulfilled. The degree and quality of the fulfilment will be explained in more detail below.

The navigational system has been implemented through two pieces of software, one for analysing image data and one for controlling the robot hardware. The part responsible for the image analysis is made as a plug-in for Mobotware. The task of segmenting the image, with the goal of extracting line features from crop rows, has been accomplished through a series of steps. While each of the steps used for the segmentation has proved useful, some are more prone to failure than others. The aggregate of the steps is a program that is indeed able to extract the sought after lines, but lacks the reliability to be considered a final solution.
The MRC-script which is responsible for interacting with the robot itself has mainly been developed with testing purposes in mind. For that it works great as it is easy to use the in-built functions, but for any large scale programming it quickly becomes tedious to work with.

Regarding the hardware itself, the Vibro Crop Robotti, several issues were had with it when trying to utilize the hardware interface tools from Mobotware. The robot was able to complete the test trials on the field layout, and gave fine results in tests designed demonstrate its accuracy. Alas, from an outside perspective it did not perform as gracefully as was anticipated.

Bibliography

[1] Amir, R. (2014), Crop Row Navigation with 3D sensor

[2] Andersen, J. C., Andersen, N. & Ravn, O., Autonomous Rule-Based Robot Navigation in Orchards

[3] Xue, J. (2014), Guidance of an agricultural robot with variable angle-of-view camera arrangement in cornfield

[4] Xue, J., Grift, T. E. (2011), Agricultural Robot Turning in the Headland of Corn Fields

[5] Siegwart, R., Nourbakhsh, I. R., Scaramuzza, D., Introduction to Autonomous Mobile Robots, Second Edition, The MIT Press Cambridge, 2011

[6] http://docs.opencv.org

Appendix A

The MRC-Script

```
v = 0.5                    % robot speed
driveDist = 0.2            % distance driven in the 'driveon' command
fwdDist = 1.5              % distance driven past the end of line
odoDist = 0                % total distance driven using odometry
driveCount = 0             % number of driveon-cmd's since last turn
driveMin = 5               % minimum number of driveon-cmd's before
                           % "endOfLine" is called again
angle = 90
angleNeg = -angle
endRow = 0                 % distance to end of row, set to $vis4 in main
endRowMin = 2.3            % endRow must be smaller than this value
                           % to call "endOfLine"
rowWidth = 0.75
rowDist = 0;               % distance driven towards next row
right = 0                  % determines if the robot turns right
turns = 0                  % number of turns performed so far
turnLimit = 1              % number of turns before resettting
heading = 0                % average heading from the last 3 meters

vision "poolpush img=18 i=1 cmd='linefinder img=18 device=18 smrcl'"
wait 2

label "main"
        if($vis2 == -1)"halt"
        y = $vis2
        x = $vis3
        xy = sqrt(x*x+y*y)
        th = atan2(y, x)

        xCalc  = cos($odoth+th)*xy+$odox
        yCalc  = sin($odoth+th)*xy+$odoy
        thCalc = $odoth + th

        endRow = $vis4
        heading = $vis5
```

```
        %-----------------------------------------------------
        eval x
        eval y
        eval endRow
        eval heading
        %-----------------------------------------------------

        if(endRow < endRowMin & endRow != -1 & driveCount ...
        ... > driveMin) "endOfLineLeft"

        driveon xCalc yCalc thCalc "rad" @v v ...
        ... : ($drivendist > driveDist)

        driveCount = driveCount + 1
        eval driveCount
goto "main"


label "endOfLineLeft"
        driveCount = 0
        odoDist = endRow + fwdDist
        if(right == 1) "endOfLineRight"
        call "swap"
        turns = turns + 1
        if(turns == turnLimit)"resetLeft"
        label "backLeft"

        %-----------------------------------------------------
        dth = heading - $odoth
        x = odoDist
        y = sin(dth)*x

        xCalc  = cos($odoth+dth)*x+$odox
        yCalc  = sin($odoth+dth)*x+$odoy
        thCalc = $odoth + dth

        drive xCalc yCalc thCalc "rad" @v v : ($targetdist < 0.0)
        %-----------------------------------------------------

        x1 = $odox
        y1 = $odoy
        th1 = $odoth

        turn angle

        x2 = $odox
        y2 = $odoy

        dx = abs(x2-x1)
        dy = abs(y2-y1)
        d = sqrt(dx*dx+dy*dy)
        rowDist = (cos(th1)*d) + rowWidth

        %-------------------------------------------
```

```
        eval rowDist
        %-----------------------------------------------

        fwd rowDist
        turn angle
goto "main"


label "endOfLineRight"
        call "swap"
        turns = turns + 1
        if(turns == turnLimit)"resetRight"
        label "backRight"

        %-----------------------------------------------------
        dth = heading - $odoth
        x = odoDist
        y = sin(dth)*x

        xCalc  = cos($odoth+dth)*x+$odox
        yCalc  = sin($odoth+dth)*x+$odoy
        thCalc = $odoth + dth

        drive xCalc yCalc thCalc "rad" @v v : ($targetdist < 0.0)
        %-----------------------------------------------------

        x1 = $odox
        y1 = $odoy
        th1 = $odoth

        turn angleNeg

        x2 = $odox
        y2 = $odoy

        dx = abs(x2-x1)
        dy = abs(y2-y1)
        d = sqrt(dx*dx+dy*dy)
        rowDist = (cos(th1)*d) + rowWidth

        %-----------------------------------------------
        eval rowDist
        %-----------------------------------------------

        fwd rowDist
        turn angleNeg
goto "main"


label "resetLeft"
        turns = 0
        call "swap"
goto "backLeft"
```

```
label "resetRight"
        turns = 0
        call "swap"
goto "backRight"


label "swap"
        % swaps the value between 0 and 1
        switch(right)
                right = 1
        case 1
                right = 0
        endswitch
return


label "halt"
        idle
        wait 1
goto "main"
```