Hyperspectral seed detection with supervised learning classification system

Seed planting control

Author:
Christian Liin Hansen

University:
University of Southern Denmark - The Technical Faculty

Instructors:
Henrik Midtiby
Lars-Peter Ellekilde

Project owner:
ImproSeed ApS

Course:
Thesis for the degree M.Sc.E in robotic systems, 40 ECTS points

Project period:
September 1st 2014 - June 1st 2015

Todo list

Abstract

English

Nothing yet

Danish

Nothing yet

Contents

1 Preface

Nothing yet

2 Reading manual

Nothing yet

3 Introduction

In collaboration between the company ImproSeed ApS and the University of Southern Denmark, ideas for a master thesis project has been developed. The company ImproSeed ApS is working with the forestry planting process to improve the efficiency of fully growing threes. At the moment an efficiency of the planting success lies between 60-75%, i.e. minimum 25% lost in profit. Therefore ImproSeed ApS is interested to maximize the efficiency of the seeds sprouting process, the germination. The idea was to use a vision based solution, where the system is able to differentiate the seeds into three categories as followed:

- *Green*. Seeds are ready for planting. Send to to planting process.
- *Yellow*. Seeds are not ready for planting. Keep them in current process.
- *Red*. Seeds are not valid for planting. Discard the seeds from current process.

Additionally a learning component should be in place for letting the vision system learn how a specific type of seed looks in the different categories.
In this case, it would ideally means that no seeds would be planted in the ground if they where not in a *good* condition. That would contributed to a higher efficiency. Due to uncontrolled environment, like bad soil, wildlife and weather conditions etc. an efficiency of 100% would never be realistic.

3.1 Project description

The setup of the system contains a conveyor belt that runs with a continuous velocity, where a mounted camera from above is sensing the incoming seeds. The image processing system needs to classify the seeds due to the features. One feature would be to look if the seeds is starting sprouting and in this case, how much has the germination reached. An other feature is, if the seed has sprouted, what is the condition of the sprouts. If a sprout is bended or even cracked then this seed should perhaps be sorted out.
A block diagram shown in figure 3.1, is where the different components in the system is indicated.

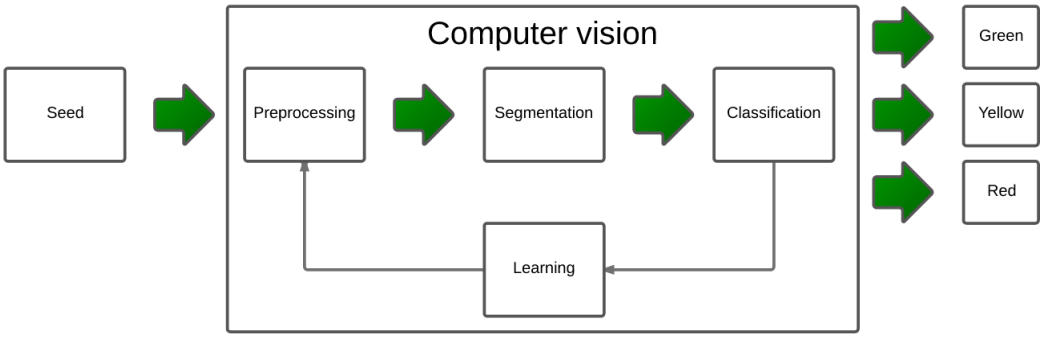

Figure 3.1: Blockdiagram of the computer vision system

- *Seed*. Incoming seeds of a specific type on a conveyor belt is random distributed among the categories. These are monitored by a line spectrometer, that is creating a hyperspectral cube
- *Preprocessing*. Each image from the hyperspectral cube contains important information about the seed

- *Segmentation.* Sorting images and after finding contours, finding central coordinates of each seed.
- *Classification.* Categorize each seed into three buckets due to how their quality and how far their germination is.
- *Learning.* Doing classification, information can be memorized on how a specific seed type looks. This will be used to have a more universal system, which can handle different type of seeds.

3.1.1 Sensor for vision system

By using a normal digital cameras, the option of extracting information outside the spectrum of visible light seems limiting. However from literature search, it is possible to extract information of materials in their near-infrared spectrum (NIF) by using hyperspectral imaging. This techniques has been used in many different application, like detecting the quality of wine-grapes [13], rice cultivar identification [8] and many others like mineral exploration, agriculture, and forest production. [14]. Therefore to classify the seeds into the three categories a line spectrometer will be used. The hypothesis is that there is a significant difference in the reflected wavelength for each pixel, when the objects is seed or a sprout. However if the seeds or sprouts are covered with resin, experiments in how this would change the electromagnetic spectral signature needs to be investigated. Thinking about the water contain can perhaps lead to a feature extraction.

3.1.2 Learn how other type of seeds looks in different categories

The system needs to detect the difference of the seeds, which is related to how far in the sprouting process each has reached. Having a system, that can handle only one kind of seed type is not enough. Therefore the system should be able to learn how different seeds looks like by supervised learning and thereby not only be limited to one kind of seed. The idea is that the user can take a portion of manually sorted seeds in a given condition and use that for training data. The result would be a more universal system, which can classify different types of seeds into the three introduced categories, *Green*, *Yellow*, *Red*.
An example will explain the flow:

- Example 1
  - Manually sorted seeds of type A, category *Green* is placed on a running conveyor belt
  - Each seed is segmented and classified as type A, category *Green*
  - Repeat training with type A, category *Yellow* and *Red*
  - After training procedure, a handful of mixed categories type A seeds is placed on the running conveyor belt. The system should be able to recognize the category for each seeds and thereby sorting the seeds correctly.

3.2 Deliverables

At the end of this Master Thesis project, A MSc. report documenting the following will be delivered:

- A seed detecting method based on computer vision, where the main methods is compared together alternative methods
- A learning method based on AI, where the main methods is compared together with alternative methods

- A description of the project management, i.e. timetable and logbook

3.3 Learning goals

The learnings goals for this Master Thesis project is defined [3]. The learning outcome is:

3.3.1 Knowledge

The student

- is able to account for relevant engineering skills based on the highest level of international research within the subject area of the programme
- has a good understanding of - and be able to reflect on - relevant knowledge within the subject area of the programme
- is able to identify relevant scientific problems within the subject area of the programme

3.3.2 Skills

The student

- is able to assess, select and apply scientific methods, tools and competencies within the subject area of the course
- is able to present novel analysis and problem-solving models
- is able to explain and discuss relevant professional and scientific problems
- is able to communicate in writing in a clear and understandable manner

3.3.3 Competence

The student

- is able to manage work and development situations that are complex and unforeseen and require new solution models
- is able to independently initiate and carry out discipline-specific and crossdisciplinary cooperation and to assume professional responsibility
- is able to independently take responsibility for his/her own professional development and specialization is able to disseminate research-based knowledge

Furthermore the student is able to demonstrate engineering skills in

- Understand and explain how a line spectrometer works.
- Interpret the hyperspectral data from the line spectrometer and create an images that can be handled for segmentation of seeds.
- Implementing a seed detection algorithm that detects seeds out from the image that is created by the data from the line spectrometer
- Implementing a learning algorithm that can use training data for a specific type of seed and be able to classify the seeds based on the training data.

4 Related work

Using hyperspectral imaging, information beyond the visible spectra is extracted. Using a line spectrometer a column of pixel will be ready for processing at each scan. The idea is to extract the information in the near-infrared (NIR) part of the spectrum to get the spectral signature of each image. As the line spectrometer scan across the seeds, many grayscale image can be created, where each grayscale image represent a small wavelength band. All the images is stacked on top of each other to create a hyperspectral cube. This cube is defined at having the x and y axis represent the spatial coordinates and the z axis represent the spectral dimension.
A figure from the reference [7] is illustrating the principle in figure 4.1

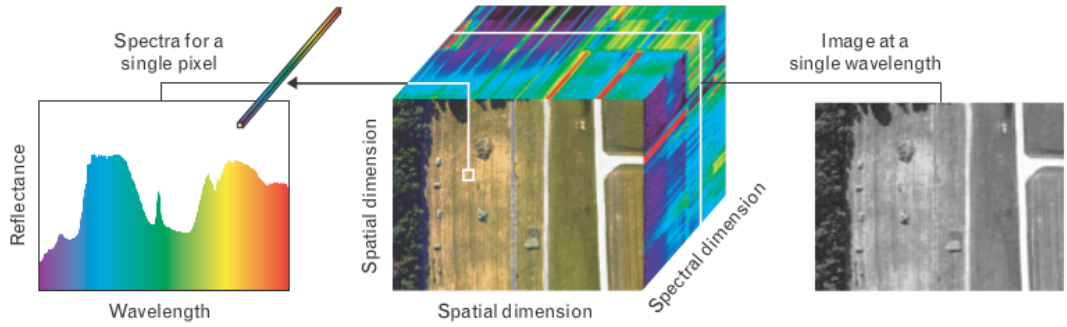

Figure 4.1: Hyper spectral imaging [7]

Evt have referencer her på, hvordan man selv vil kunne lave sit eget hyperspektralt setup

5 Division of work

This chapter gives an overview of how the project was structured from start to end. Decision has been made based on gained experience from implementing different proof of concept. The different proof of concepts is described in details in chapter 6. The overall structure in the project is described in section 5.1. A short conclusion for each proof of concept is described in section 5.2.

5.1 Project methods

As described in section 3.1, the overall task is to have a vision system that is able to detect, analyse and classify each seed on a moving conveyor belt. The output is a 3D coordinate, which must be calibrated in both intrinsic and extrinsic parameters, before transferred as a motion command to a ABB Flexpicker.

In the start of the project, the author and supervisor agreed that it was important to get the chain of the project with different components up and running first. This is illustrated in figure 5.1. As soon all the component is implemented it is possible to optimize each component individual and independent of the system. Using this approach, compared to start out by interfacing a hyper spectral camera, will lead to faster implementation time. The pitfall by using the other approach, is the risk of not having the data flow establish in time before hand-in of the project. Working with growing seeds is time consuming and can not be fully control. This can lead to time consumption and delay in the project. By using the described project method, this minimize the risk of delays in the project. I.e. if a component is inhibited since seeds are not ready, the possibly of optimizing other component, like the classifier component exist. In this way the project are agile and is not locked on hold.

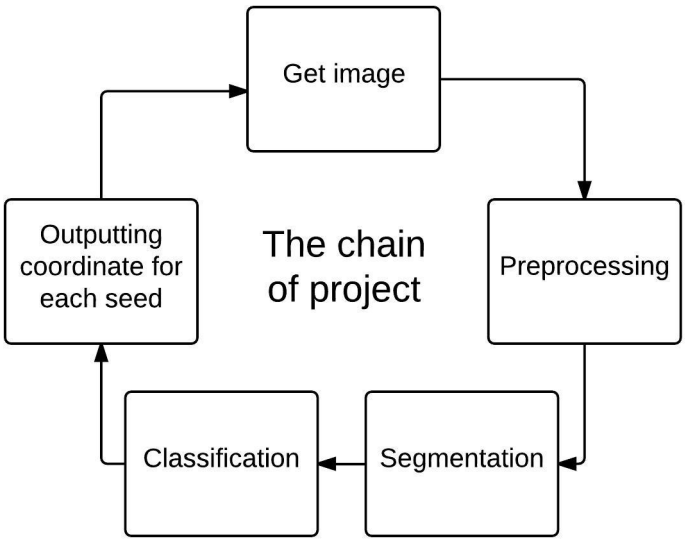

Figure 5.1: The chain of the project

The start of figure 5.1 goes with feeding image into the system, which is handled by the *Get image* component. From there each image will go through the components *Preprocessing*, *Segmentation*, *Classification* and finally the *Outputting coordinate for each seed* component in that given order. Then a new image will be loaded and the same process repeats.

After the structure for the working process was setted, the implementing of the chain of the project could begin. The implementation would be based on proof of concept. The different implementations is described further in chapter 6.

Add ref to repositories where all the proof of concept script is availabel

5.2 Implementation the chain of project

In this section a short overview of the first proof of concepts in order to finish the chain of project is explained. All the proof of projects through this thesis is explain in chapter 6. All the Python code can be found in the following Github repository:
In the next section, and short overview of the optimization of different components is explained.

5.2.1 Black Pepper detection

The result of implementing the Black Pepper detection program was insight in using different function from the OpenCV library in Python. Many function will be used later in other proof of concept implementations. Examples are the FindContour function and area checking. This is shown in the Python listing 5.1. The actually code, and arguments for the function FindContour has been reduced for better visualization. The hardcoded value of 50 will in later implementation be substituted with a statistically estimate. A in depth description of the Black Pepper detection program is descriped in section 6.1.

```
1  import cv2
2  ...
3
4  # Finding the countours in a morphed image
5  contours = cv2.findContours(morphedImg)
6
7  for contour in contours:
8          area = cv2.contourArea(contour)
9
10         # If area is too small, then skip this contour and
11         # take the next one in the list
12         if area < 50:
13             continue
14 ...
```

Listing 5.1: Using area checking for each contour

5.2.2 Classify square and circle objects

The result of implementing a square and circle classification system, was gained experience in classification of objects and principle of supervised learning. Having two training sets with squares and circles, the system was taught. Feature extraction was performed and a classifier was implemented. The result was a simple linear classifier, the Perceptron. This was able to classify the test image, which contains a mix of circles and squares. A more detailed description of the choice of feature, description of the classifier and result is described in section 6.2. Additionally classification result relies on extracting good features. In the simple case with circles and squares the solution was easy. The compactness feature was a perfect for detecting circles, which is indicated in figure 6.7. When it comes to testing with real seeds, new features must be investigated, however the principle of feature extracting and find the separation line or plane remains. However it is not guaranteed to get features that is linear separable. This means that the classifier component properly needs to be optimized. More features will properly be needed when dealing with real seeds.

5.3 Optimizing component in the chain of the project

It was concluded after the implementation described in subsection 5.2.2 that the chain of project has been implemented, i.e referring to figure 5.1 all the component is completed. However now the optimization process is started. The first component that must be optimized is the *Get image*, since the input images now must be of real seed and not ideal noise free square and circles.

5.3.1 Classification of seeds with two features - Digital camera

The result of implementing the seed classification, bases on images taken from a digital camera, was one step closer to the final implementation. In this implementation, new features was found by analysing the criteria for a good and bad seeds. Overall seeds is categorizes as follow:

- Good seeds has length between 1-3 mm
- Good seeds has a white or yellowish sprout tip

If the sprouts has a strong yellow or brown color nuance in the sprout, the seeds is bad. The tip of the sprout is very important.

5.3.2 Classification of seeds with two features - Web camera

In the end of the project the implementation was trying to have real time images taken from with the setup. At this point, the project was at a crossroad. Which camera is needed? The choice was between using a high-end digital camera or using a webcamera. In section **??** it was decided to go with the webcamera. The result of this showed that the white balance is not realistic in the image processing. At the moment, this is still to be investigated if the hue-mean value is significant different (by using the ANOVA statistical tool) compared to the hue value from seeds that in reality was brown.

5.3.3 Classification with two features using SVM

5.3.4 Classification with three features using SVM

5.3.5 Classification from video stream

5.3.6 Outputting 3D coordinates using pin hole model

6 Proof of concept

In this section the different proof of concept approaches through the project is described.

6.1 Black pepper detection

The chain of the project was to have several components ready, including a classifier component. It was decided to break down the task to complete the tracking of objects only, without using any classifier. They way the program was strucked is illustrated in figure 6.1. The green boxes indicate components that was implemented and the red box indicate the component that was skipped for this given implementation.

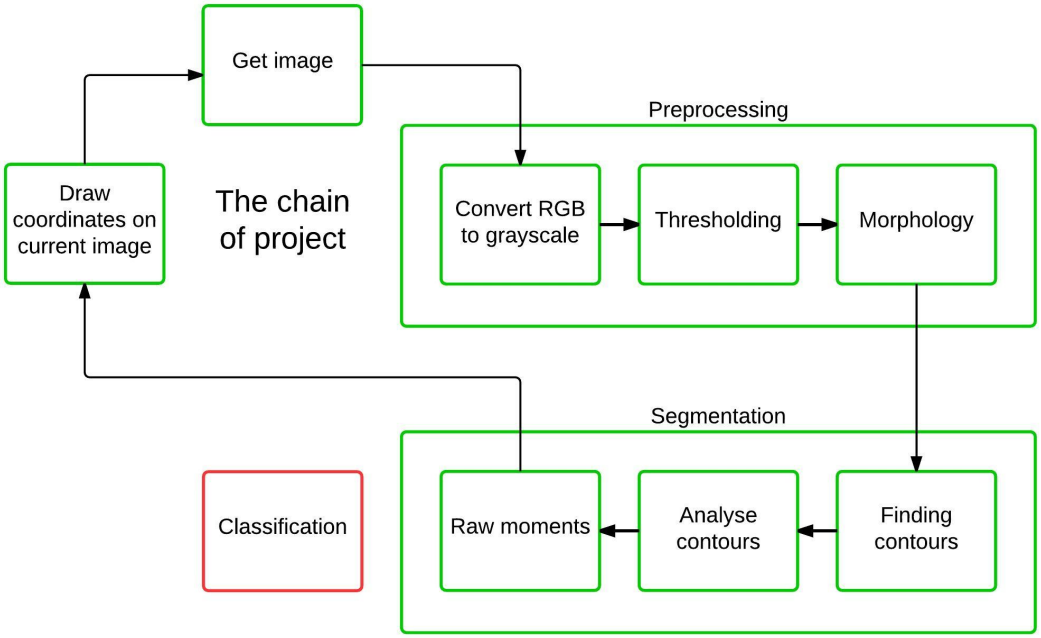

Figure 6.1: Testing proof of concept without classifier component

The seed detection algorithm was tested on grain of black peppers. This is illustrated in figure 6.2. The next step is to classify between different shapes, like squares and circles.

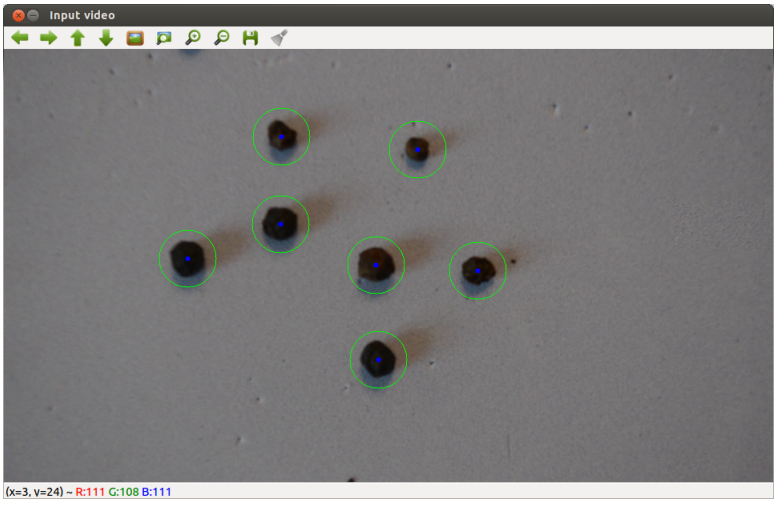

Figure 6.2: Proof of concept - Detecting peppers on a flat surface. Inspired from source [12]

6.2 Classify square and circles objects

The main task for this project is to be able to automatically sort seeds in different groups, based on the principle of supervised learning. Supervised learning is a way of learning a system to perform a given task, where training data is presented which has already been classified [9]. The idea is to use training data that correlate with the unseen testing data and hopefully will be able to classify the testing data properly with the learning gained from the training data. The concept is illustrated in figure 6.3, where the training data can consist of several data files, which are usually different training images. The testing data is usually a test image or a video stream.

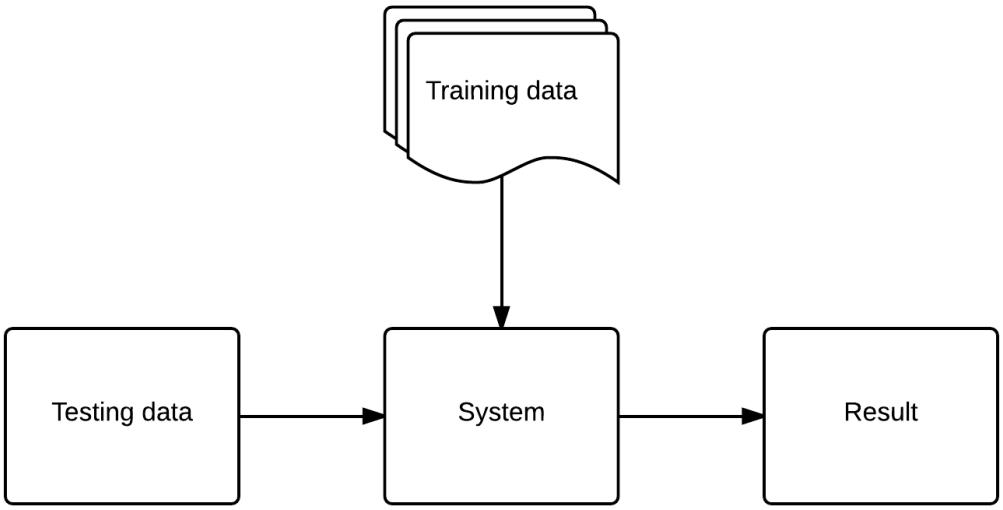

Figure 6.3: Supervised learning concept with training data and testing data

A minimum vision system was needed to be establish in order to improve different components of the system. The idea was to start out with a vision system that was able to detect simple objects and classify each object in the image to a certain category. This assignment includes the following task:

- Load training into the system
- Do segmentation and feature extraction from the training data
- Do classification of the training data and hence let the system learn from the training data
- Load testing data into the system
- Let the system classify the testing data with respect to what the system had learned from previous training data

6.2.1 Features

The task is to let the system be able to classify between circles and rectangles in an image. The reason for using objects like circles is that these objects has at least one strong feature that can be used for a relative easily classification result. The strong feature is known as compactness [11].

$$\frac{4\pi \cdot area}{perimeter^2} \tag{6.1}$$

where ideal circles has a compactness value of 1. This result is derived, when inserting the area and perimeter for a circle. This is shown in equation 6.2.This feature has the nice property of being invariant to scale, translation and rotation and hence is a good feature for circles versus rectangles detection.

$$\frac{4\pi \cdot area}{perimeter^2} \Rightarrow \frac{4\pi \cdot \pi r^2}{(2\pi r)^2} \Rightarrow \frac{4\pi^2 r^2}{4\pi^2 r^2} \Rightarrow 1 \tag{6.2}$$

Due to quantification error, a compactness of a circle in the image processing will be approximately 1.

6.2.2 Training and testing data

The training data is two images, where the one contains rectangles and the other contains circles. The testing data is an image where a mixture of rectangles and circles is represented. This images is shown in figure 6.4, 6.5 and 6.6 respectively.

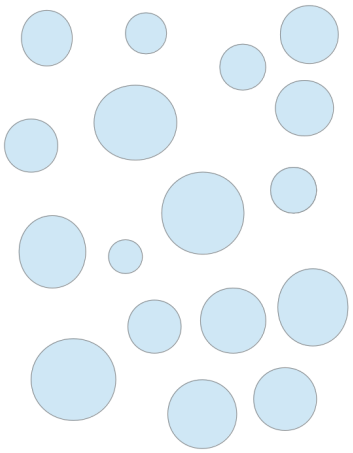

Figure 6.4: Training data with circles

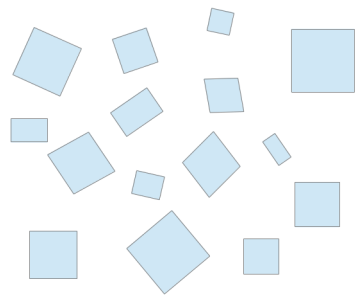

Figure 6.5: Training data with squares

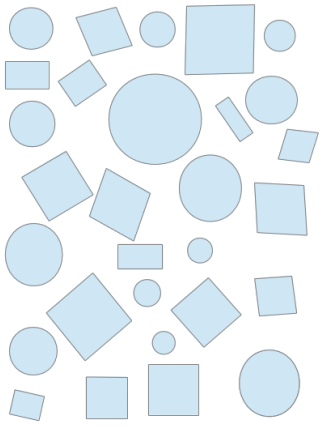

Figure 6.6: Testing data

After the training data was loaded into the system using OpenCV library, it is possible to find the contours for each image and calculate the compactness and area for each contour. As described earlier, the compactness of circles will have higher values compared to squares. These features is illustrated in the feature space plot in figure 6.7, where the red feature point represent data from figure 6.4 and the blue represent data from figure 6.5.
The result in figure 6.7 shows that circles and rectangles can be linear separated by using the shape feature compactness. The area is less important, but however used to plot the feature space. With the training data and testing data available, the system can begin to make a classification based on the training data.

6.2.3 The Perceptron

It was chosen to use a simple classifier like the Perceptron to get hands-on with classifiers. The Perception is a simple neurale network that will find a solution if the data is linear separable [1]. The Perceptron is relatively simple to implement compared to other classifiers, like e.g. Support Vector Machines. This will be referred as SVM in the rest of the documentation. It was decided to implement the Perceptron for better understanding of the classification procesedure. However for future implementation with SVM, using OpenCV SVM library will be used [4]. Using features that is linear separable is needed, when using the simple version of the Perceptron. However the Percetron can be extended to handle non-linear separable data by using the so called *Kernel trick* [1], but is out of the scope within this thesis.

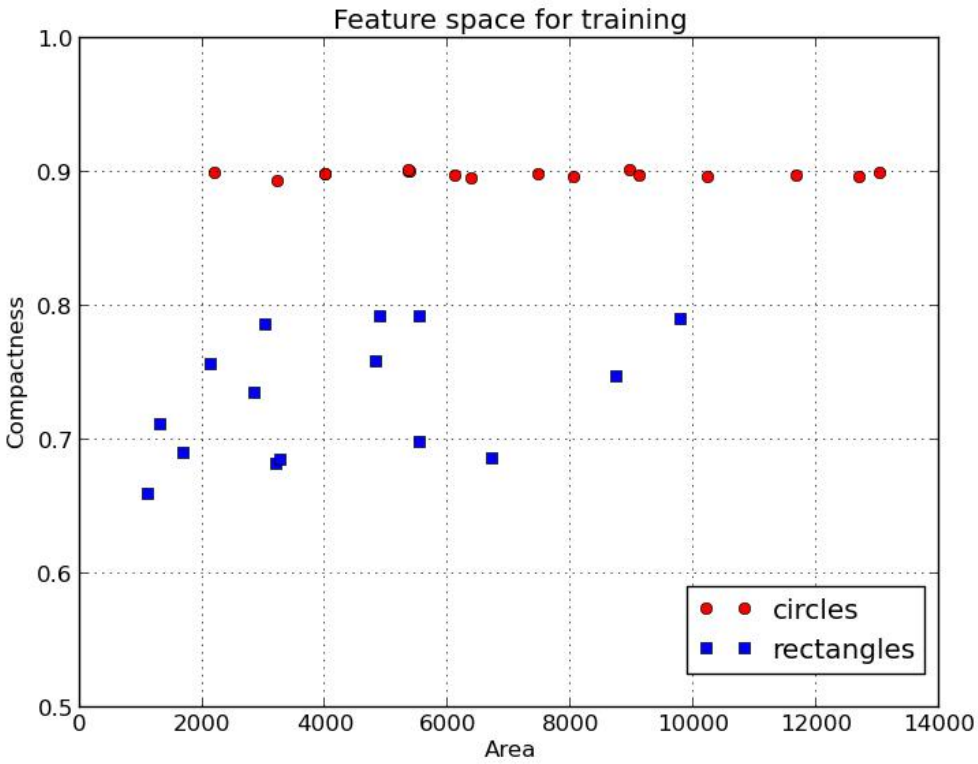

Figure 6.7: Feature space of compactness and area of each object in the training data

6.2.3.1 Feed forward neurale network

The Perceptron is a single layer, feed forward neural network. The network consist of two input neurons with weights, bias input and one output neuron. The activation function of the output is a step function. The Perceptron is implemented using the Perceptron Learning rule, which classify each output to either 1 or -1. With the update weighs and bias the linear classification line is defined [1] in equation 6.3 as followed:

$$y = \frac{w_0}{-w_1}x + \frac{b}{-w_1} \tag{6.3}$$

The classification separation line is illustrated in figure 6.11 with a blue line. The pseudo code for the Perceptron implementation is shown in algorithm 1

Input: Pre classified training images with circles and rectangles separately
Output: Updated weights and bias to be used for making the classification line
Initialization: Set weights and bias to zero: $w_0 = w_1 = b = 0$

```
while runFlag is true do
    Initialize errorCounter
    for data in trainingData do
        dot product between weights and input.
        if dot product + bias >= 0 then
            result = 1
        else
            result = -1
        end
        error = pre classfied class - result;
        if error is not zero then
            Update the weights and bias
            Increment errorCounter
        end
    end
    if errorCounter is zero then
        set runFlag to false
    end
end
```

Algorithm 1: Pseudo code of the implemented Perceptron classifier

6.2.3.2 Result of the Perceptron

The result of the Perceptron shows that sometimes the classifier do misclassification. This is illustrated in figure 6.8 and indicated by two red arrows, where two objects has been classified as rectangles, but are obviously circles. The blue dots represent rectangles and red dots represent circles. Sometimes the classification is a success which the result shows in figure 6.9.

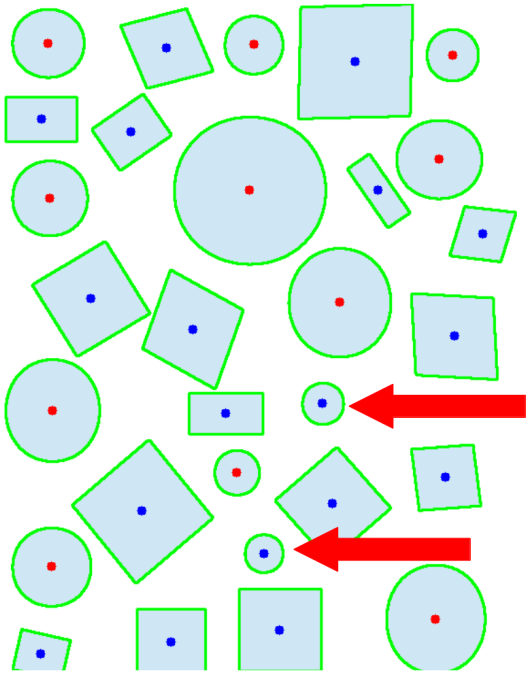

Figure 6.8: Wrong classification - test1

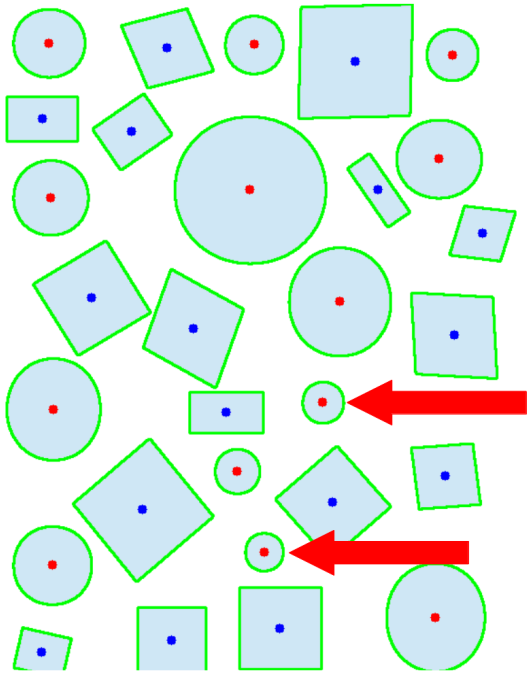

Figure 6.9: Correct classification - test2

6.2.3.3 Discussion and conclusion of Perceptron

The algorithm of the Perceptron runs until there is no error. If the training data is linear separable, the Perceptron will find a solution. When the algorithm stop and classify the training data correct, this do not guarantee full correct classification, since testing data might differ in feature values. The result of the misclassification with two objects, as shown in figure 6.8 is explained by investigating the testing data. In figure 6.10 the convergence of the weight vector, bias term and the error is shown. The algorithm terminates after approximately 160 iterations. In figure 6.11 the Perceptron finds a solution for the classifier, by using the training data. In figure 6.12 this solution misclassify the testing data with two objects. In figure 6.13 the misclassification is two objects with a relative high compactness and small area. This explains the result in figure 6.8. The conclusion is that using the Perceptron as classifier are sensitive to small offsets in feature space. To eliminate this drawback, implementation of SVM as classifier would be a solution.
As stated before sometimes the Perceptron do classify correct, which was shown in figure 6.9. In figure 6.14 the convergence of the weight vector, bias term and the error is shown. The algorithm terminates after approximately 180 iterations. In figure 6.15 the Perceptron finds a solution for the classifier, by using the training data. In figure 6.16 this solution separates the features. This result in a correct classification of the testing data which is shown in figure 6.17.

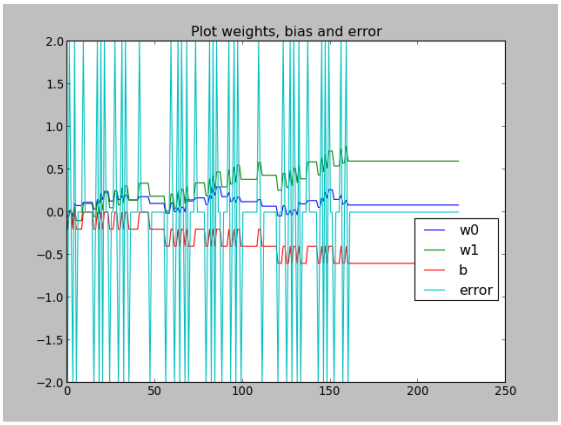

Figure 6.10: Convergence of weights, bias and error first run - test1

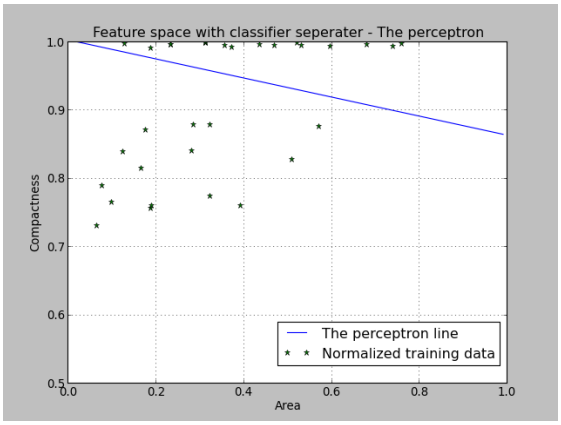

Figure 6.11: Classification line with training data - test1

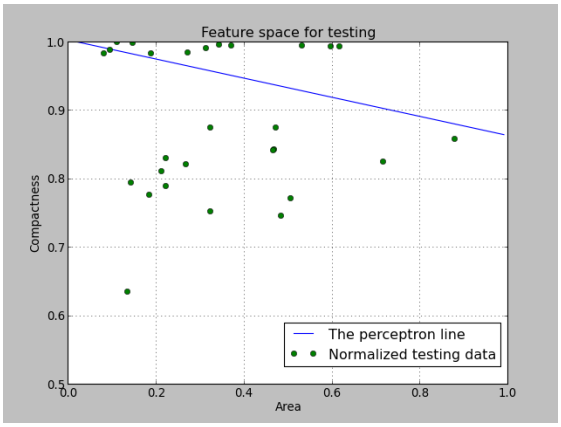

Figure 6.12: Classification line with testing data - test1

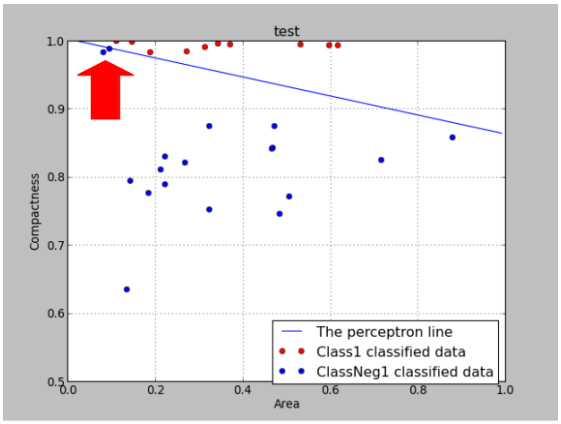

Figure 6.13: Final classification with classification line - test1

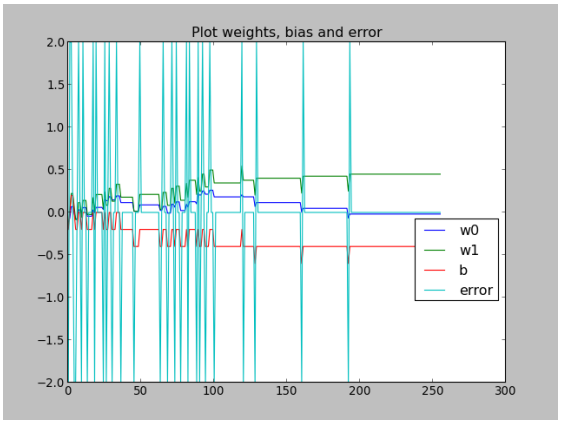

Figure 6.14: Convergence of weights, bias and error first run - test2

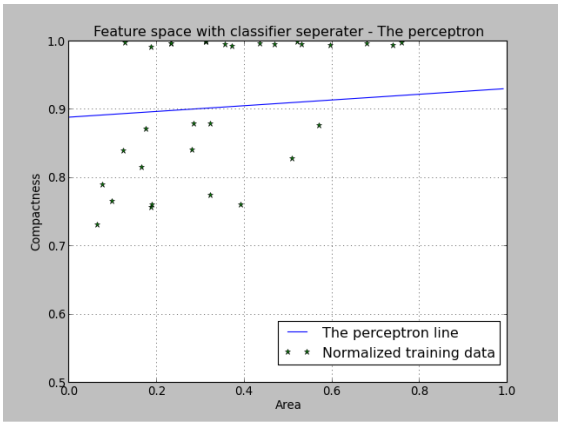

Figure 6.15: Classification line with training data - test2

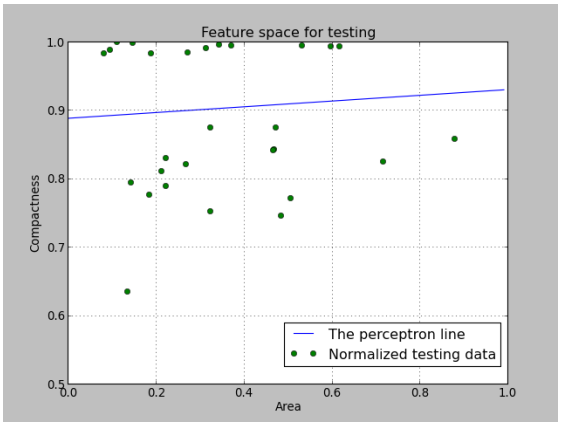

Figure 6.16: Classification line with testing data - test2

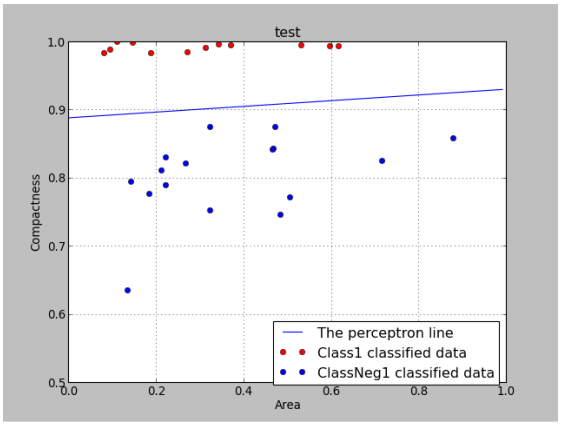

Figure 6.17: Final classification with classification line - test2

6.3 Classify real seeds with digital camera

The vision system was expanded to handle images of real seeds on a conveyor belt. These images was taken at the scene at the company ImproSeed.The image was taken with an taken with an Olympus XZ-1 digital camera with a resolution of 3648 x 2736, 1/15 exposure time and ISO speed rating of 200.
The images was taken with a high resolution digital camera. In this chapter, the image processing procedure will be explain. In section 6.4.1 arguments for choosing the web camera is listed. In section 6.4.2 the setup with the web camera is explained.
Here we place all the stuff where we use nice images...

6.3.1 Preprocessing

Using the VideoCapture class from OpenCV, which is shown in listing 6.1, the RGB image is available for image preprocessing. The preprocessing is needed in order to segmentate the image, i.e. find objects. The preprossing part takes the RGB image as input and outputs a front ground image, a sprout image and a combined seed and sprout image. The process is as shown in figure 6.18.
In order to produce the front ground image, the RGB input image is converted into a grayscale image and then thresholded. This is convenient since the conveyor belt in the setup is relatively darker than the seeds and sprouts. This result in a binary image, where the background is black and the front ground which include the seed and sprout is gray. Normally a binary 8 bit image is divided into two binds, i.e. pixel intensities with 0 and 255, where 0 value and 255 value represent black and white pixels respectively. However the sprout pixels must be added later to form the *Seed and sprout image* regarding figure

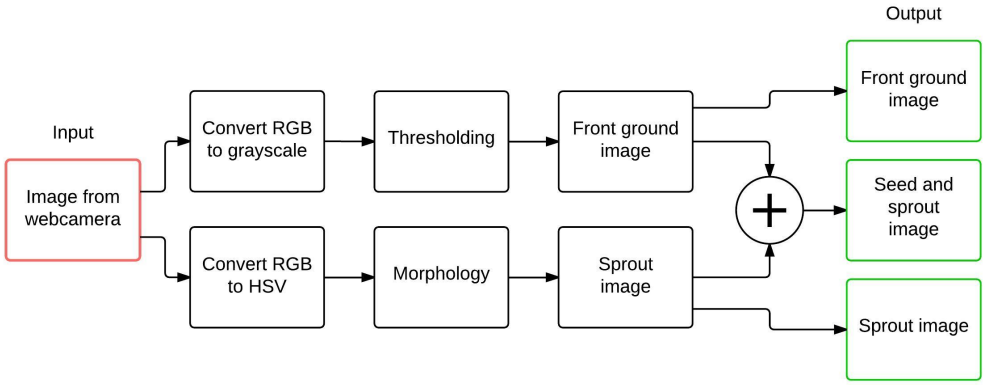

Figure 6.18: Preprocessing flow in order to generate the frontground , sprout and the combined seed and sprout image

6.18. In order to separate the sprouts from the rest of the front ground, this intensity value is reserved to be 255. This is the reason for setting the upper intensity value to 128 under the thresholding process.

6.3.2 Result of preprocessing

The input image, the front ground image, the sprout image and the seed and sprout image is shown in figure **??**, **??**, **??**, **??** respectively.
The input image shown in figure **??** was taken with an Olympus XZ-1 digital camera with a resolution of 3648 x 2736, 1/15 exposure time and ISO speed rating of 200. In section 6.4.2 the Logitech C930e web camera with a resolution of 1920 x 1080 is used. The image quality of the two camera is different, however the principle of preprocessing process is the same, despite the difference in quality of the images.

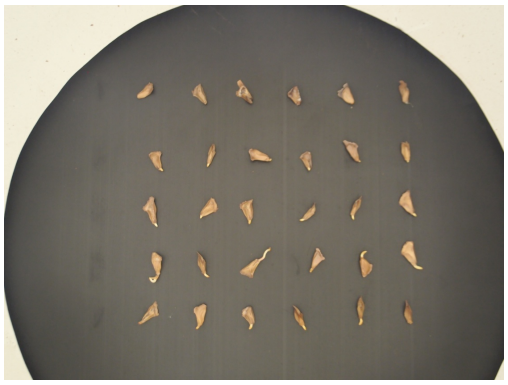

Figure 6.19: Input RGB image

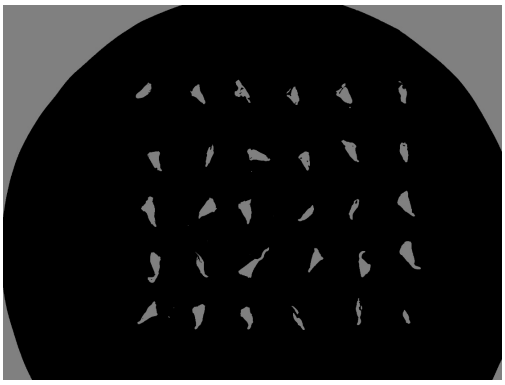

Figure 6.20: Front ground image

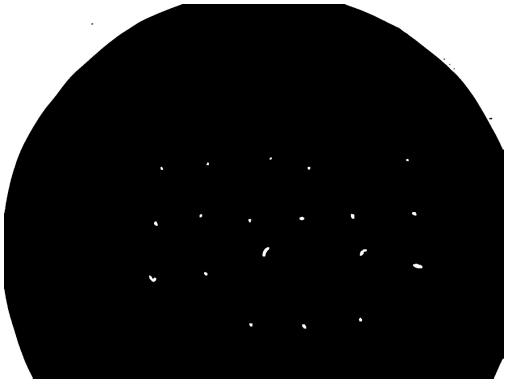

Figure 6.21: Sprout image

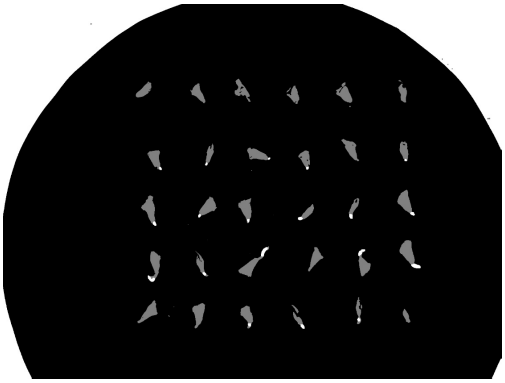

Figure 6.22: Seed and sprout image

6.3.3 Segmentation

In the segmentation component, the images from the proprocessing compoennt is used. The main idea in the segmentation part is to use the FindContours function from OpenCV. The contours is the pixels edge around an object which was wound in the thresholded or Front ground image. The contour is drawn, by using the drawContour function from OpenCV. The result is shown in figure

Here tell what the argument was for FindContour etc

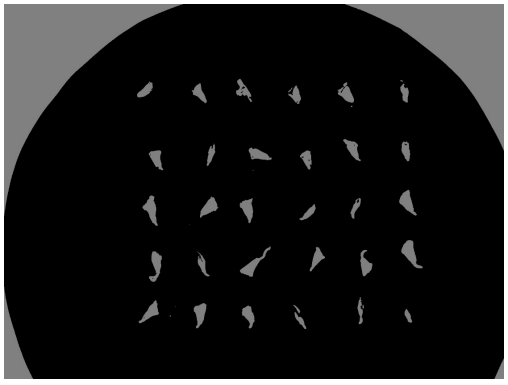

Figure 6.23: Front ground image

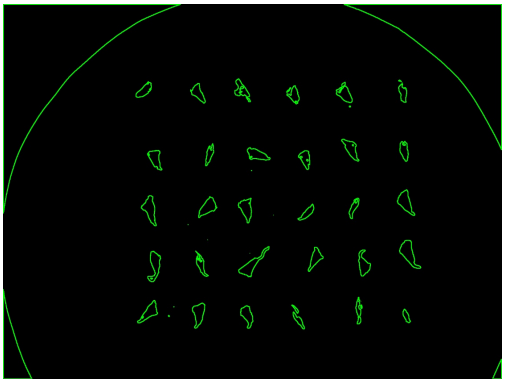

Figure 6.24: Contours found

For each contour, the analysing starts. This is shown with the following Python listing, where each contour in the list of contours, contains a list of x,y coordinate pixels. From this each single seedobject is analysed in the way, that the pixel pr pixel is analyzed. If the pixel is white, it is a sprout pixel. If the pixel is gray then it is a seed pixel. After

the list of seed and sprout is generated for each object, the feature extraction starts. A rotated boundingbox is fitted in order to get the length and width of this one. Together with minRectArea from OpenCV, the center of mass cordinate can be extracted. Then the ratio width/length is calculated, and together with hue values a list of featues is generated out from each image. This is displayed in figure, where it sees how the list is constructed.

6.3.4 Classification

In this subsection, the classification procedure is explain. The training images contains of a image where the seeds have too long sprouts and training image where the sprouts have an acceptable length. This is shown in figure 6.25 and figure 6.26 respectively. In figure 6.27 and 6.28 the detection of the seeds is illustrated.

Figure 6.25: Too long sprouts

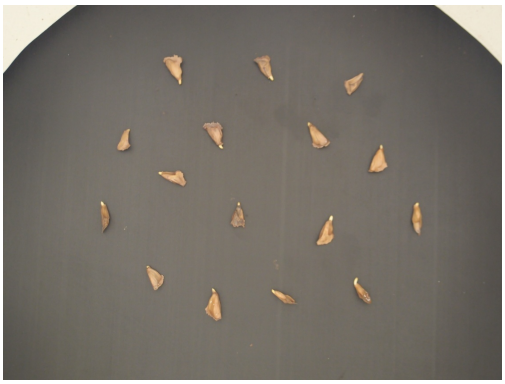

Figure 6.26: Acceptable length of sprouts

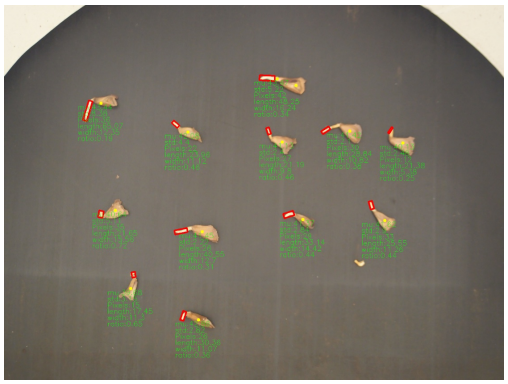

Figure 6.27: Detecting of sprouts

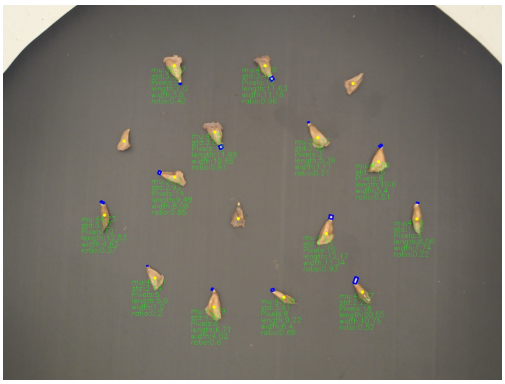

Figure 6.28: Detecting of sprouts

The feature plot is shown in figure 6.30 where the x-axis is the length of the sprouts oriented bounding box, and the y-axsis is the number of sprout pixels with each sprou. The result shows features that is non separable, which calls for the implementation of an other classifier as the Perceptron.
A quick test with the Perceptron is however possible. In order to let the Perceptron classify out from the training data, the data needs to be separable. In order to do this, part of the data must be removed. The most intuitive data point that can be removed is the

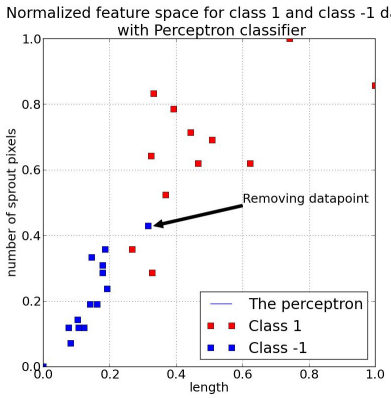

Figure 6.29: Feature plot showing non separable data

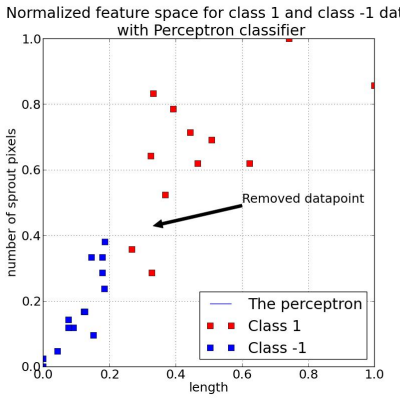

Figure 6.30: Feature plot made separable by removing a datapoint

6.3.5 Learning

The idea is to use a supervised learning approach, i.e. the system will be supplied with training data, where all the seeds are in a one of the three categories, like *Green*, *Yellow*, *Red*. At the same time the wavelength from the line spectrometer will be applied, so the system would be able to learn how a specific type of seed will look like for each categories.
At the moment one option is to use support vector machine for the supervised learning approach or the deep learning approach.

6.4 Mounting webcamera to the setup

In this chapter, the image processing procedure will be explain. In section 6.4.1 arguments for choosing the web camera is listed. In section 6.4.2 the setup with the web camera is explained.

6.4.1 The choice of camera

Define a suitable camera for this project is a challenged task. For choosing a camera several parameters exists. In this project the three most important parameter is estimated to be the complexity of interfacing, the cost and the availability of the camera.
As described in section **??**, the approach is to have the the chain of the project up and running before any component can be optimized. It was decided, in collaboration with the supervisor, to start out with the a relatively cheap and easy interfacing camera. Alternatively a high-end camera could be a possibility. However the high-end camera was reserved for an other project, had a price range of 10 times the web camera and was interfaced with Ethernet and external power supply. By experience from previous courses on SDU, interfacing the USB Logitech C930e web camera is straightforward process by using OpenCV library. The following Python code listing, shown in listing 6.1 shows how to load images from the web camera and display them on the screen. [5]. The argument to the OpenCV VideoCapture class indicate the connected camera device index.

```
1  import cv2
2  cap = cv2.VideoCapture(1)
3
4  while(1):
5
6      # Capture frame-by-frame
7      ret, frame = cap.read()
8
9      # Display the resulting frame
10     cv2.imshow('frame',frame)
11
12     # If the user hit the ESC bottom, the program ends...
13     k = cv2.waitKey(30) & 0xff
14     if k is 27:
15         print("User closed the program...")
16         break
17
18 # When everything done, release the capture
19 cap.release()
20 cv2.destroyAllWindows()
```

Listing 6.1: Using VideoCapture class from OpenCV

Here I have to show the image, where the sprouts are white in real life, i.e. where the hue_mean does not have any signicifant difference compaired to the image that is brown in real life.

Later in the project, the webcamera will perhaps revield its limitations. However as a starting point, the webcamera has been the chosen camera. Therefore the argument for choosing the USB webcamera boils down to this:

- Plug-and-play USB interface
- Simply Python code using OpenCV to get access to images
- Several Logitech cameras was available at SDU
- The price of the web camera is less than 1000 DKK [10].

6.4.2 The camera setup

The chosen camera, the Logitech C930e was extended with a wooden stick which is shown in figure 6.31. This was done in order to mount the camera inside a white sphere with a uniform light distribution. The camera is interfaced to the nearby PC by a USB connection. The software running on the PC is a Python script. This is shown in figure 6.32.

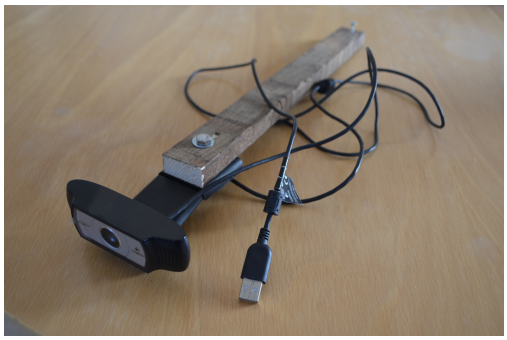

Figure 6.31: Webcamera extended with wooden stick

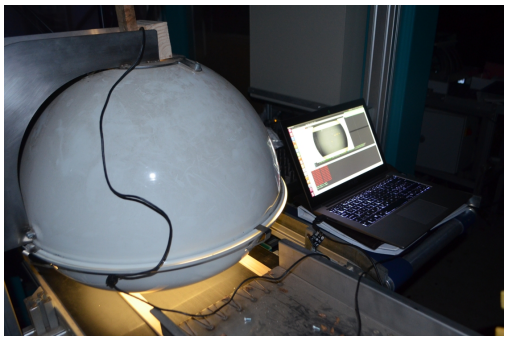

Figure 6.32: Webcamera mounted inside sphere and connected to PC

In order to let the cameras field of view cover the area where the seed could be placed doing operation with the shaking chute, the camera was need to be place further away perpendicular to the conveyor belt. It is still to be discussed, how much the distance from the camera lens to the conveyor belt, or zoom, can be tolerated. The more zoom or the less the distance is, the more seeds will be out of the cameras field of view. A mechanical solution would be to narrow the metal chute, which will force the seeds to be dumped more at the middle of the conveyor belt. However this will increase the area density of the seeds and increase the risk of having seed touching each other or clustered together. If a seed is touching or occluded by another seed, it creates a need to makes the vision system more complex. Dealing with seeds that is not free from other seeds is not within the scope of this project. However ideas for solution this this problem will be discussed later.

To measure how far away the camera must be mounted in order to cover the cameras field of view, two wooden sticks was placed on the conveyor belt, which is shown in figure 6.33. The camera was thereafter mounted by having the two wodden sticks inside the cameras field of view, as shown in figure 6.34. The white area in the figure is the white sphere seen from inside. The black boarder is added for better visualization and is not part of the original image data.

Figure 6.33: Wooden sticks used to align distance

Figure 6.34: Webcamera mounted inside sphere and connected to PC

6.4.3 The camera settings

In order to have better control over the web camera, a Python scrip was implemented, where different camera settings was available. These camera settings are listed below:

- Disable auto focus and auto exposure
- Focus
- Sharpness
- Exposure
- Cropping area

In order to control the parameters, the video for Linux version 2 control package was installed, since OpenCV camera setting support was limited. From a Ubuntu terminal, the command *v4l2-ctrl -list* displays all the available settings in range and steps. In this way, different commands could be executed through the Python scrip by using *OS* command. A Python code snippet is shown in listing 6.2, where the autofocus and auto exposure is disable and manually adjusted together with the sharpness parameter.

```
1 import os
2
3 os.system('v4l2-ctl -d 0 -c focus_auto=0')
4 os.system('v4l2-ctl -d 0 -c exposure_auto=1')
5 os.system('v4l2-ctl -d 0 -c focus_absolute=40')
6 os.system('v4l2-ctl -d 0 -c exposure_absolute=250')
7 os.system('v4l2-ctl -d 0 -c sharpness=200')
```

Listing 6.2: Calling an OS command from the Python script to adjust camera settings

The reason for disable the autofocus is to avoid any uncontrolled behaviour of the web camera while the system runs. Since the distance from the seed on the conveyor belt and to the camera lens do not change over time, the argument for having a fixed zoom exist. If the autofocus is not disabled, the camera could potentially begin to autofocus process while the seeds are in the field of view. This could result in wrong segmentation and therefore extra control would be needed. Therefore to keep the Python script as simple as possible, the autofocus was disabled and manually focus was used instead together with the sharpness parameter. A video demonstrating the Python script with the given parameters is available on the following Youtube video:

`https://www.youtube.com/watch?v=jUG6IO6ayv4&feature=youtu.be`

6.4.3.1 Test at ImproSeed with manually adjust parameters

The Python script described in section **??** was tested at the location of Improseed. The parameters was adjusted by trial and error until the most satisfied result was archived, which is shown in figure 6.38. This figure has a green boarder around the image to aid the reader. The boarder is not part of the image data. All the images was cropped equally. The caption of each figure contains the parameter. The auto focus and auto exposure has been disabled. The manual adjusting value is referred as absolute values. The abbreviation list is as followed:

- Absolute focus = AF
- Absolute exposure = AE
- Sharpness = S

Figure 6.35: AF=20, AE = 250, S = 200

Figure 6.36: AF=30, AE = 250, S = 200

Figure 6.37: AF=40, AE = 250, S = 200

Figure 6.38: AF=40, AE = 260, S = 200

Figure 6.39: AF=40, AE = 260, S = 255

Figure 6.40: AF=50, AE = 260, S = 200

Figure 6.41: AF=45, AE = 260, S = 200

Figure 6.42: AF=37, AE = 260, S = 200

7 Final implementation of vision system

This is the section, where the focus is on test and results. How good did we do classification? What is the procentance. How good was it compaired to the old system? Which classifier was best? Random Forrest vs, SVM?

Is that really important? There is a lot of papers out there that has investigated this before....

8 Robotics

An future extension of the Master Thesis is to interface the ABB Flexpicker robot, which is shown in figure 8.1, with the computer vision system. The task of the ABB Flexpicker is to grasp each of the classified seed and place them physically in a place according to their category. At the moment the focus in the Master Thesis is the computer vision and AI component. These components needs to complete the final test, before any more time can be invested in the robotic part.

Figure 8.1: The ABB Flexpicker robot that do the pick-and-place task after seeds has been classified by the vision system

9 System test

Nothing yet

10 Discussion

Nothing yet

11 Conclusion

Nothing yet

12 Future work

This is the future work section

13 Bibliography

[1] Lasse Simmelsgaard Boerresen.
Comparison of algorithms for seedling classification.
Bachelor thesis, University of Southern Denmark, The Maersk Mc-Kinney Moller Institute, June 2013.

[2] Sen-Ching S. Cheung and Chandrika Kamath.
Robust techniques for background subtraction in urban.
http://computation.llnl.gov/casc/sapphire/pubs/UCRL-CONF-200706.pdf, available: 4-9-2014.

[3] Master Thesis course description.
http://fagbesk.sam.sdu.dk/study/fagbasen/fagprint.shtml?fag_id=27506&print=1, available 15-9-2014.

[4] OpenCv dev team.
Opencv - support vector machines.
http://docs.opencv.org/modules/ml/doc/support_vector_machines.html, available 4-12-2014.

[5] OpenCV dev team.
Capture video from camera.
http://docs.opencv.org/trunk/doc/py_tutorials/py_gui/py_video_display/py_video_display.html, available: 4-3-2015.

[6] OpenCV development team.
Finding contours in your image.
http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html, available 15-9-2014.

[7] David Marden Dimitris Manolakis and Gary A. Shaw.
Hyperspectral image processing for automatic target detection applications.
https://www.ll.mit.edu/publications/journal/pdf/vol14_no1/14_1hyperspectralprocessing.pdf, available: 4-9-2014.

[8] Wenwen Kong.
Rice seed cultivar identification using near-infrared hyperspectral imaging and multivariate data analysis.
http://www.mdpi.com/1424-8220/13/7/8916, available: 4-9-2014.

[9] Bing Liu.
Web Data Mining - Exploring Hyperlinks, Contents, and Usage data.
Springer, second edition, 2011.

[10] Logitech.
Logitech c930e web camera.
http://www.logitech.com/da-dk/product/webcam-c930e-business, available: 4-3-2015.

[11] Henrik Skov Midtiby.
Object recognition.
http://henrikmidtiby.github.io/downloads/2014-11-05featurespresentation.pdf, available: 4-9-2014.

[12] Stackoverflow mirosval.
Filled circle detection using cv2 in python.

http://stackoverflow.com/questions/21612258/filled-circle-detection-using-cv2-in-python, available 16-9-2014.

[13] Francisco J. Rodríguez-Pulido.
Grape seed characterization by nir hyperspectral imaging.
http://www.sciencedirect.com.proxy1-bib.sdu.dk:2048/science/article/pii/S0925521412002116, available: 4-9-2014, DOI: 10.1016/j.postharvbio.2012.09.007.

[14] Center for Space Research The University of Texas at Austin.
Hyperspectral remote sensing.
http://www.csr.utexas.edu/projects/rs/hrs/hyper.html, 2014.

A Title of Appendix A

Text of Appendix A is Here

B Title of Appendix B

Text of Appendix B is Here