Tree Seed Classification System based on Artificial Intelligence

Seed planting control

Author:
Christian Liin Hansen

University:
University of Southern Denmark - The Technical Faculty

Instructors:
Henrik Skov 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 1^{st} 2014 - June 1^{st} 2015

Todo list

Abstract

English

Nothing yet

Danish

Nothing yet

Contents

Sikre at referencehenvisning ikke bare er tal, men forfatter og årstal

Check denne url http://www.imm.dtu.dk/~janba/MastersTh pdf

Se ...*TestingIr* Her ligger to frø for tæt på hinanden og påvirker den cropped boundingbox. Her skal der laves et check på hvilke hvide pixel giver mening at se på.

Fedt. Med OtSus optimale threshold så er frontgrond image bedre og kræver mindre

1 Preface

Nothing yet

2 Reading manual

An overview of this project can be seen from the table of contents. However in order to aid the reader of the documentation, this reading manual has been added to give an extra explanation of the coming chapters:

- Chapter 3 - Introduction: This chapter introduce the company ImprooSeed ApS and explain the problem statement. A proposed solution is described in the project description.
- Chapter 4 - Related work: This chapter describes the status of seed detection today and what approached other people have done

Need to finish this

- Chapter 5 - Division of work: This chapter describes how the project was structure explains the process from start to end.
- Chapter 6 - Computer vision system: This chapter describes the final implemented computer vision system with results, discussion and conclusion

The other chapters are self explainable. Furthermore the reference in the documentation are shown in brackets, like this: [1] which corresponds to papers, websites or books.

This documentation can be downloaded as PDF file from the following repository:
`https://github.com/ChristianLiinHansen/Master-thesis-doc`

All the source code, written i Python, can be downloaded at Github at the following repository:
`https://github.com/ChristianLiinHansen/MasterThesis`.

The time management work can be downloaded from Google Drive at the following shared link:
`https://goo.gl/fTMY94`

In the following documentation the word *seed* is an object that may or may not contain a *sprout*. In figure 2.1, image (a) is a seed without a sprout. Image (b) is a sprout with a white sprout and image (c) is a seed with a white and longer sprout.

(a) Seed with no sprout (b) Seed with sprout (c) Seed with longer sprout

Figure 2.1: Define seed and sprouts.

3 Introduction

In collaboration between the company ImproSeed ApS and the University of Southern Denmark (SDU), 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 two categories as followed:

- *Green - good condition.* Seeds are ready for planting. Send to to planting process.
- *Red* - bad condition. 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

In order to maximize the efficiency, which can only be evaluated after many years, a fully working system including a vision system and robot systems is needed. The focus in this Master Thesis is to implement the vision system, that is able to classify each seed as either a good or bad. A clear interface to a robotic system will be described, and if time allows, work on the robotic system will be made.

The setup contains a conveyor belt where a camera, which is mounted above and perpendicular to the belt, is sensing the incoming seeds. The vision system will be based on different components such as preprocessing, segmentation and classification of each seeds using features. Example of features would be, if there is a sprout, how far the sprout is reaching out of the seed. An other feature is the color of the sprouts. Additionally if a sprout is bended or even cracked this gives information about the seed and can be used as a feature as well. A block diagram shown in figure 3.1 illustrate the input and output of the vision system. Each component is shortly explained on the following page.
The input to the vision system is images from the camera. The inputs comes in two folds. First the training data is fed to the system, which teach the system. Then after the testing data is fed to the system.

- Inputs
  - Training seed: These seed are labelled data, i.e. the whole image shows seeds with one specific feature. An other training data would contains seeds with an other single feature. Example is training image 1 contains seeds where the sprouts are too long. Training image 2 contains seeds where there is no sprouts at all.
  - Testing seed: The testing data contains seeds that is not labelled and must be classified by the system as either good or bad seed, based on what the system has learned from the training data.

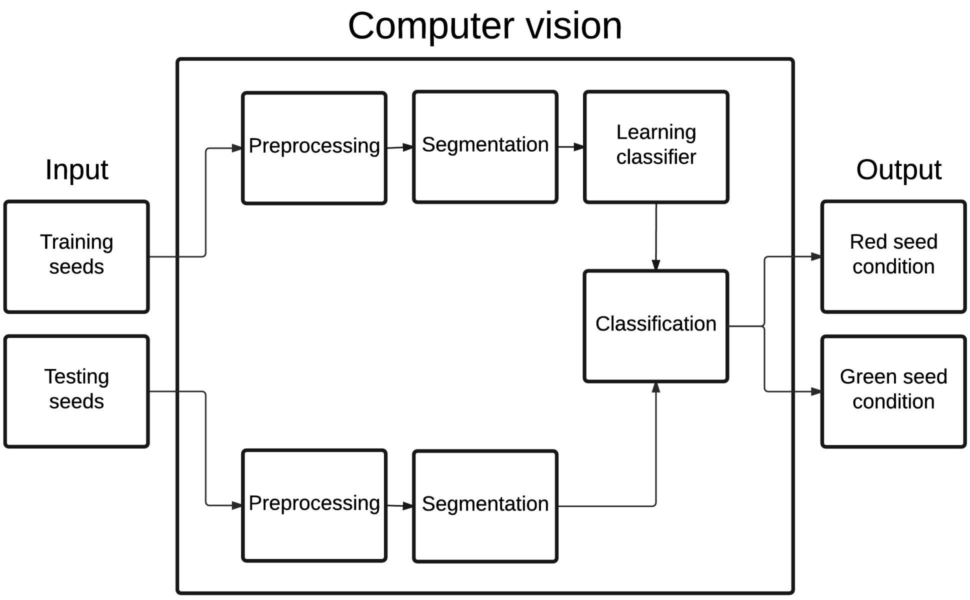


Figure 3.1: Blockdiagram of the computer vision system

- Computer vision system
  - *Preprocessing.* Prepare the image for further analyse by converting and filtering the image.
  - *Segmentation.* This component track and extract features for each seed.
  - *Classification.* Teach the system with training data and classify the testing data due to the learning of the system.
- Output
  - Red seed conditions: A seed with center of mass location and categorized as bad seed.
  - Greeb seed conditions: A seed with center of mass location and categorized as good seed.

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 [26], rice cultivar identification [18] and many others like mineral exploration, agriculture, and forest production. [29]. The hypothesis is that there is a significant difference in the reflected wavelength for each pixel, when the pixels comes form the seed or 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 containment can perhaps be a feasible feature which is only possible with hyperspectral camera [19]. In the period of this Master Thesis, the argumentation for choosing a specific type of camera will be explained.

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 and 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.

3.2 Deliverables

At the end of the Master Thesis project, the following source code and documenting 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
- A MSc. rapport documenting deliverables and learning goals

TODO: So did I compaire my main methods to alternative methods? What could be alternative detection methods? One would be for intead of preprocess the input image, I use semisupervised learning to find out what is my background, what is my seed layer and what is my sprouts. And again I also implement point 2, with a learning methods, i.e. semi-supervised learning, where I compare this with the supervised learning methods.

This is the last implementation I need to do.

3.3 Learning goals

The learnings goals for this Master Thesis project is defined by the SDU [10]. 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 cross disciplinary 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

- Implementing a seed detection algorithm, which feeds images from a camera.
- Implementing an artificially intelligence (AI) methods that performs classification of detected seeds in images.

4 Related work

Write something that match to related work here.

Seed detection has been going on for a while...
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 [14] is illustrating the principle in figure 4.1

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

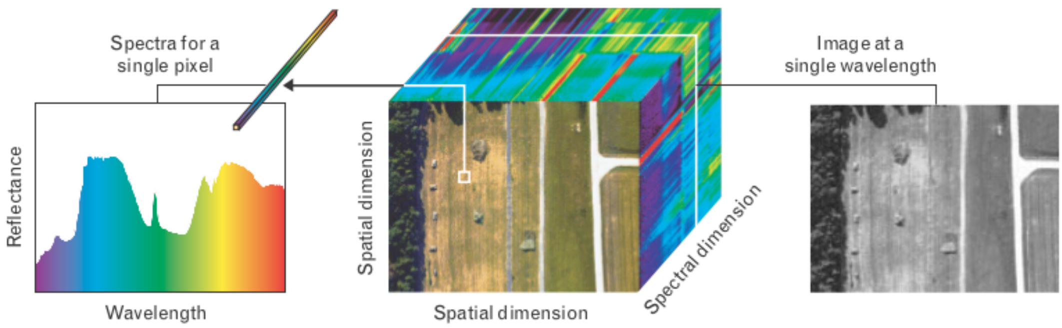


Figure 4.1: Hyper spectral imaging [14]

5 Division of work

One of the first task in the project was to make a project description, which is located in section 3.1. This has been iterative modified while the project went on. Another task was to decide coding language for implementation. The two candidate were C++ and Python. One argument for using Python is to have higher abstraction level, than C++ and access to quality 2D plotting library such as matlibplot and the data analysis library scikit-learn. With Python there exist in general a higher computation time and more RAM use compared to C++, but time performance within natural limits is not a critical factor. Therefore Python has been the selected programming language and hence the following toolboxes has been used for this project: Python 2.7.3 [2], OpenCV 2.4.9 [8], Numpy 1.6.1 [28], Matlibplot 1.1.1rc [16], SciPy 0.9.0 [17] and Sklearn 0.16b1 [25].

5.1 Project methods

As described in section 3.1, the main goal is to have a vision system which feeds images of seeds on conveyor belt and then be able to detect, analyse and classify each seed regarding their condition. The output is a x, y, z 3D coordinate, which is transferred to a robotic system for grasping. The vision system contains different component, such as *Get image*, *Preprocessing*, *Segmentation*, *Classification* and *Outputting*. These are described as chapter 6.

The chosen method was to build up the *chain of project* of simple components with room for optimization. The principle of the chain of project is illustrated in figure 5.1 for respectively the vision and the robotic system. The vision system is the main focus in this project. By using this approach data flow in the system is better established and secured compared to other methods. An example of an other methods would be to complete a component with and use additionally time in optimization and increase robustness. The pitfall is to not have data flow established in time. Additionally working with natural growing seeds is time consuming and cant be fully controlled, which can lead to delays and set the entire project on hold. By using the first method, the project is more flexible by changing to another task. E.g. if a component is inhibited for development, the task of optimizing other component exist. By using the latter methods, this option would not be feasible since components already has been optimized.

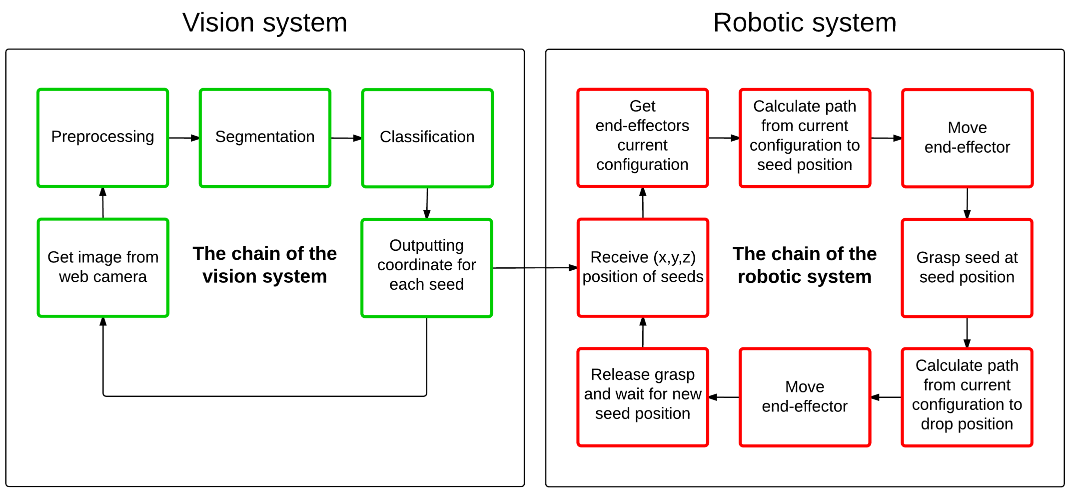


Figure 5.1: The chain of the project

5.2 Proof of concepts

Following the *chain of project* methods, described in section 5.1, the first step was to learn how to use Python with OpenCV. The chain of project was splitted up into two proof of concepts, where the first part is the black pepper detection, which is described in subsection 5.2.1. The second part is the Perceptron classifier, which was implemented in order to learn how classifier and supervised learning works. This is described in subsection 5.2.2.

5.2.1 Black pepper detection

In order to learn Python and the OpenCV libray, a black pepper detection python script was implemented. This script was without the classifier component, which is illustrated in figure 5.2. Knowledge of using preprocessing tools such as thresholding and morphology together with the segmentation tools of finding contours and their center of mass location was gained in this proof of concept.

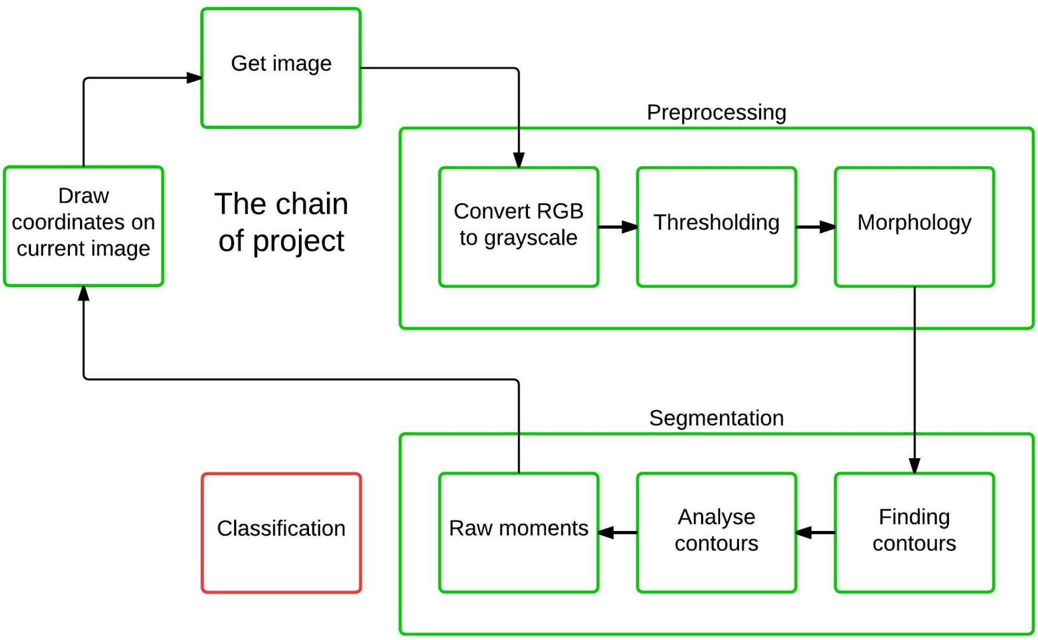


Figure 5.2: Testing proof of concept without classifier component.The green boxes indicate components that was implemented and the red box indicate the component that was skipped for this given implementation.

The system was tested on grain of black peppers.. Inspired from source [24]. The result is illustrated in figure 5.3. The next step is to classify between different shapes, like squares and circles. This is described in subsection 5.2.2.

5.2.2 Square and circle detection

The main task for this project is to classify seeds, based on the principle of supervised learning. Supervised learning is a learning technique that teach a system with training data that has been labelled [20]. The idea is that the data should correlate with the unseen testing data and hence the system will be able to classify the testing data properly with the learning gained from the training data. The concept is illustrated in figure 5.4, where the training and testing data in this project consist of images data.

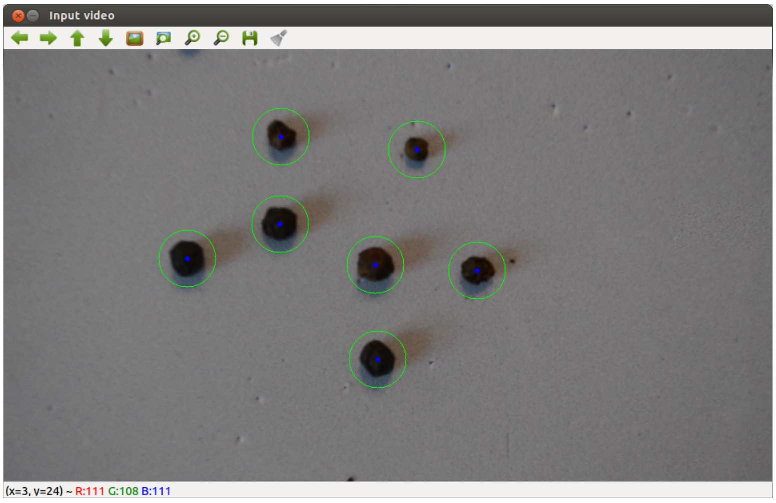


Figure 5.3: Proof of concept - Detecting peppers on a flat surface.

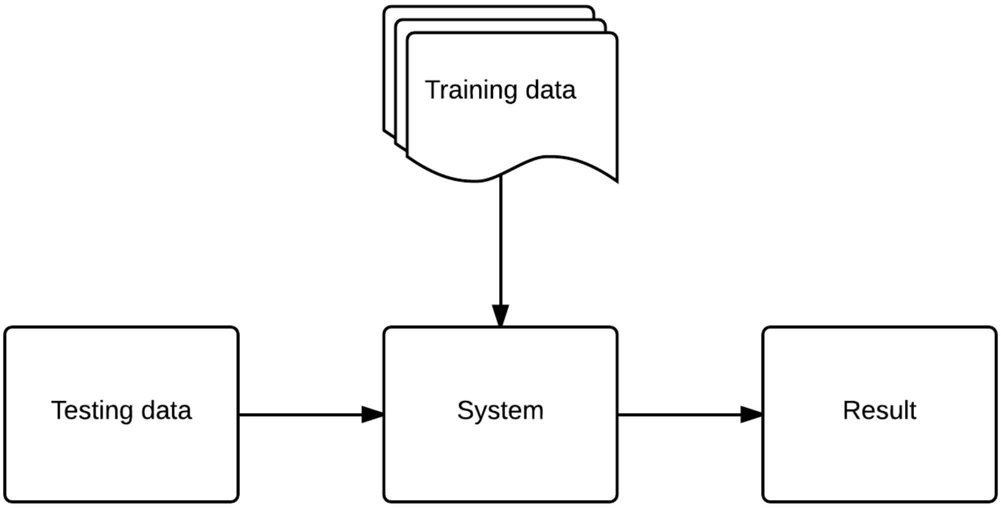


Figure 5.4: Supervised learning concept with training data and testing data

Training and testing data

In this proof of concept the focus in on implementing the classification component, based on supervised learning. Simple training and testing data has been selected and hence his proof of concept is a square and circle detection system. 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 B.1.

Features

In order to let a system learn, information from the training data must be extracted. These information is called features, which describes the training data. From previous experience and knowledge [23] the shape features such as *compactness* is effective, when it comes to detecting different shapes of objects. With this feature, circular object will have higher values compared to squares. The equation for the i-th object is shown in equation 5.1.

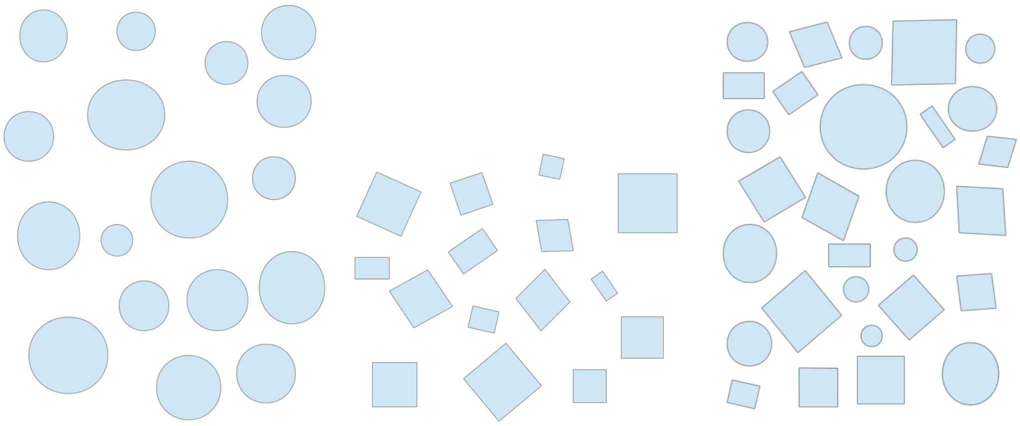


Figure 5.5: Two training images a) and b) with circle and squares respectively). Testing image c) with mix of square and circles.

$$compactness_i = \frac{4\pi \cdot area_i}{perimeter_i^2} \tag{5.1}$$

This feature has the property of being invariant to scale, translation and rotation and hence is a good feature for circles versus rectangles detection. Each object in the image will have a compactness feature value between 0 and 1, where ideal circles has a compactness value of 1 and squares will be lower. This is derived, when inserting the area and perimeter for a circle and square in equation 5.2 and equation 5.3 respectively.

$$compactness_{circle} = \frac{4\pi \cdot area_{circle}}{perimeter_{circle}^2} \Rightarrow \frac{4\pi \cdot \pi r^2}{(2\pi r)^2} \Rightarrow \frac{4\pi^2 r^2}{4\pi^2 r^2} = 1 \tag{5.2}$$

$$compactness_{square} = \frac{4\pi \cdot area_{square}}{perimeter_{square}^2} \Rightarrow \frac{4\pi \cdot l^2}{(4l)^2} \Rightarrow \frac{4\pi^2 l^2}{16 l^2} \approx 0.785 \tag{5.3}$$

After the training data is 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. The result is a feature plot, which is shown in 5.6, where the red feature point represent data from figure 5.5(a) and the blue represent data from figure 5.5(b).
The result in figure 5.6 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 classify the testing data based on the generalization of the training data.

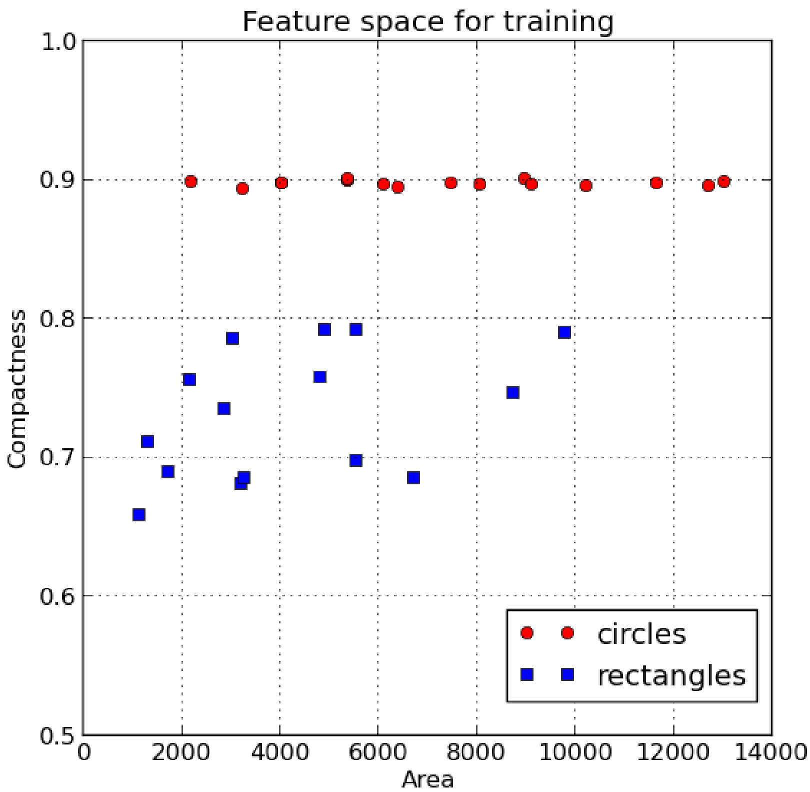


Figure 5.6: Feature space of compactness and area of each object in the training data

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 [7]. The Perceptron is relatively simple to implement compared to other classifiers, like e.g. Support Vector Machines. 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 mapping the data into higher dimension, which makes the the data linear and then map data back again. The computation time will increase when mapping from $\mathbb{R}^2 \rightarrow \mathbb{R}^3$ but can be reduced with the *Kernel trick* [7]. However this is out of the scope in this project.

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 [22], which classify each output to either 1 or -1. With the update weighs and bias the linear classification line is defined [7] in equation 5.4 as followed:

$$y = \frac{w_0}{-w_1}x + \frac{b}{-w_1} \tag{5.4}$$

The classification separation line is illustrated in figure 5.8(b) with a blue line. The pseudo code for the Perceptron implementation is shown in algorithm 1

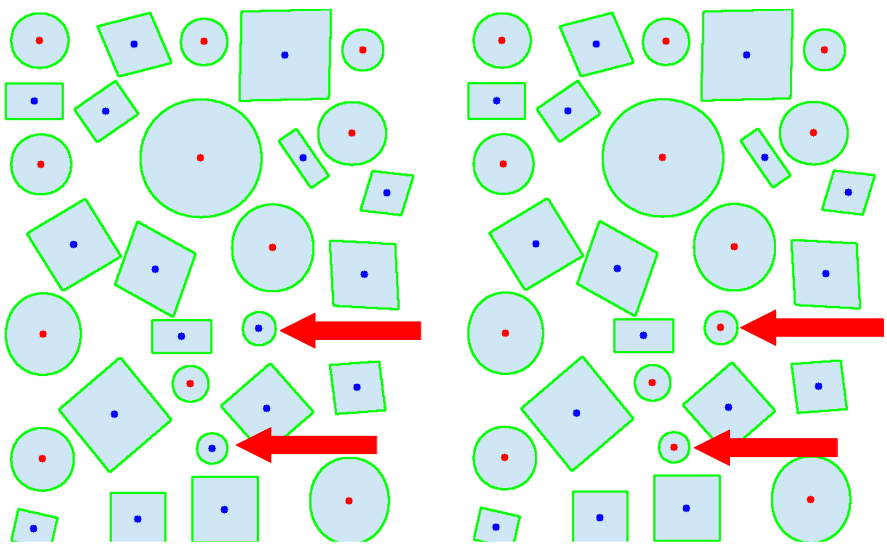
Input: Pre classified training images with circles and rectangles separately
Output: Updated weights and bias to be used for making the classification line
Initialization: Set weights and bias to zero: $w_0 = w_1 = b = 0$

```
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

Result of the Perceptron

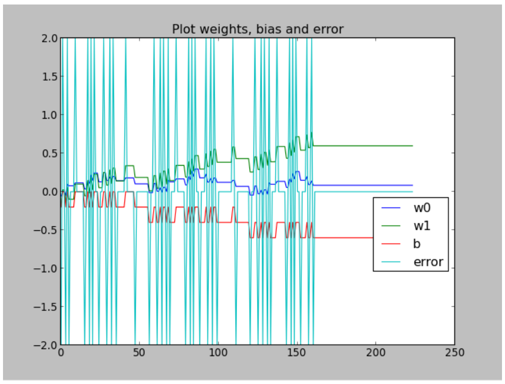
The result of the Perceptron shows that sometimes the classifier do misclassification. This is illustrated in figure 5.7(a) and indicated by two red arrows, where two objects has been classified as rectangles, but are obviously circles. The blue dots represent rectangles and red dots represent circles. Sometimes the classification is a success which the result shows in figure 5.7(b).



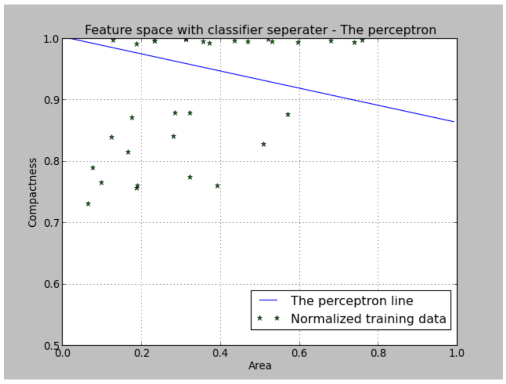
(a) (b)

Figure 5.7: Figure a) and b) with wrong and correct classification respectively

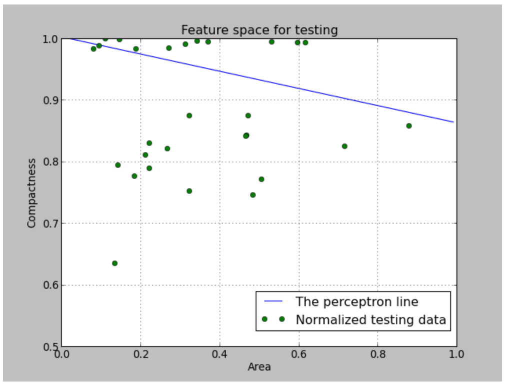
The algorithm of the Perceptron runs until there is no error. If the training data is linear separable, the Perceptron will find a solution. When the algorithm stop and classify the training data, this do not guarantee full correct classification, since testing data differ in feature values. The result of the misclassification with two objects, as shown in figure 5.7(a) is explained by investigating the testing data. In figure 5.8(a) the convergence of the weight vector, bias term and the error is shown. The algorithm terminates after approximately 160 iterations. In figure 5.8(b) the Perceptron finds a solution using the training data. In figure 5.8(c) this solution do misclassification of two objects. In figure 5.8(d) the misclassification is two objects with a relative high compactness and small area, which explains the result in figure 5.7(a).



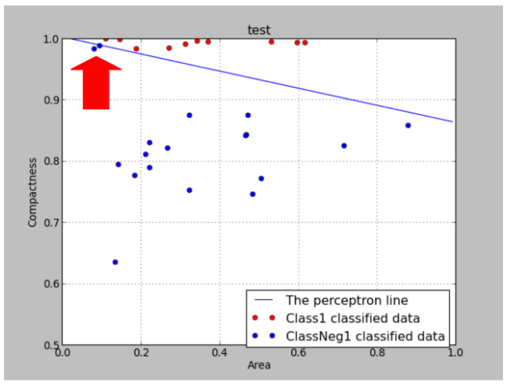
(a)



(b)



(c)



(d)

Figure 5.8: Figure a) show the convergence of the weights, bias and error. Figure b) show the separation line based on training data. Figure c) shows the testing data with the separation line. Figure d) shows the classification result, with two misclassified objects

Sometimes the Perceptron do classify correct, which is shown in figure 5.7(b). Again this can be explained by investigating the classifier. In figure 5.9(a) the convergence of the weight vector, bias term and the error is shown. The algorithm terminates after approximately 180 iterations. In figure 5.9(b) the Perceptron finds a solution for the classifier, by using the training data. In figure 5.9(c) this solution separates the features. The result is a correct classification of the testing data which is shown in figure 5.9(d). However the result shows that if the separation line is placed to close to training clusters, a high risk of misclassification exists of the testing data. To minimize the risk, a need for maximizing the separation margin is needed. The Support Vector Machine (SVM) classifier is preferable [7] for such task and hence will be integrated in the final project using the a SVM library [25].

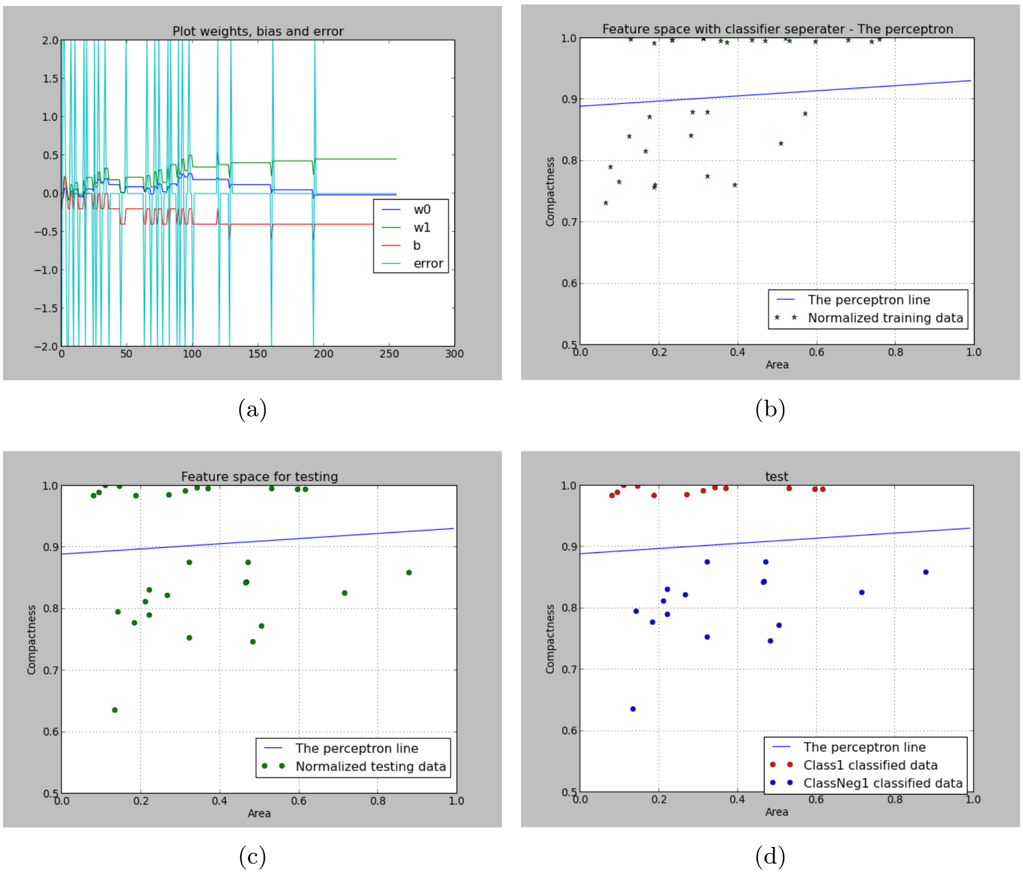


Figure 5.9: Figure a) show the convergence of the weights, bias and error. Figure b) show the separation line based on training data. Figure c) shows the testing data with the separation line. Figure d) shows the classification result, with no misclassified objects

Discussion and conclusion of Perceptron

To summarize the result of implementing a square and circle classification system is 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. Additionally classification result relies on extracting good features. In the simple case with circles and squares the solution was easy. The compactness feature is a perfect for detecting circles. 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

It was concluded after the implementation described in subsection 5.2.2 that the chain of project for the vision system, shown in figure 5.1 was implemented. The next step of optimizing was to use real seeds instead of squares and circles. The first component for optimization was the *Get image*, since the input images now must be of real seed and not ideal noise free square and circles.

5.3.1 Detection and classification of real seed

The vision system was expanded to handle images of real seeds on a conveyor belt, with the use of functionality gained from the proof of concepts, described in section 5.2. The chain of project was completed with fully working components, i.e. each image went through a preprocessing, segmentation and classification component. At the moment the classifier was based on the Perception, even that a solution would be sensitive to noise. These images was taken at the scene at the company ImproSeed.The image was taken 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 this chapter, the preprocessing, segmentation and classification procedure will be explain.
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. Together with the project owner it was agreed that seeds is categorizes as follow:

- Good seeds has length between 1-3 mm.
- Good seeds has a white sprout tip.
- Seed starts to become bad, if their sprout tips turns into yelow and brow color.
- Bad seeds have broken sprouts.

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.

5.3.2 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 6.39 and figure 6.40 respectively. In figure 5.12 and 5.13 the detection of the seeds is illustrated.
The feature plot is shown in figure 5.15 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 5.10: Too long sprouts

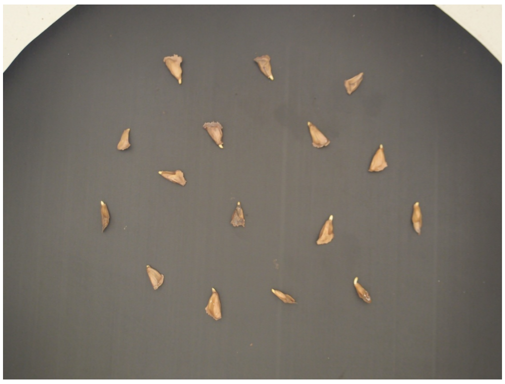


Figure 5.11: Acceptable length of sprouts

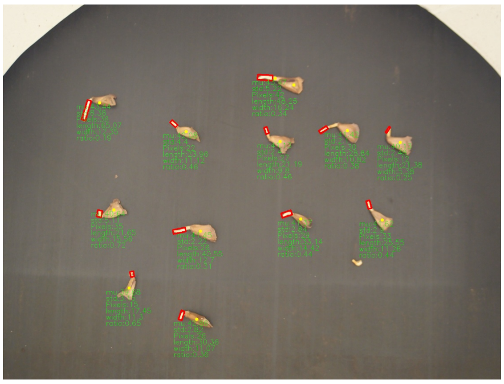


Figure 5.12: Detecting of sprouts

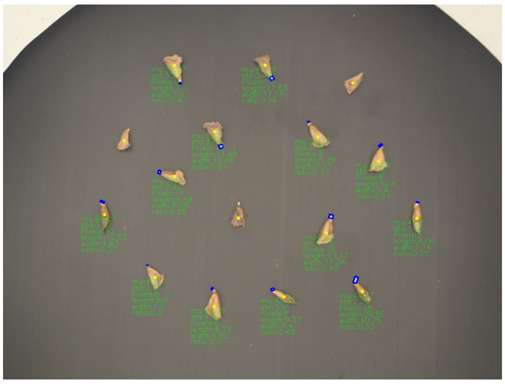


Figure 5.13: Detecting of sprouts

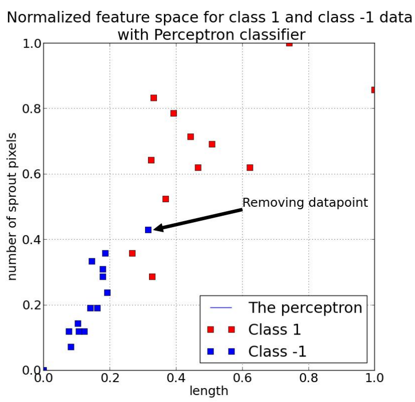


Figure 5.14: Feature plot showing non separable data

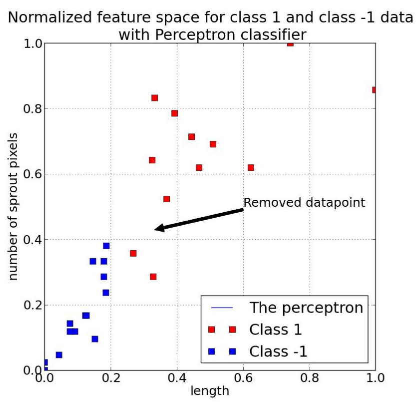


Figure 5.15: Feature plot made separable by removing a datapoint

5.3.3 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.

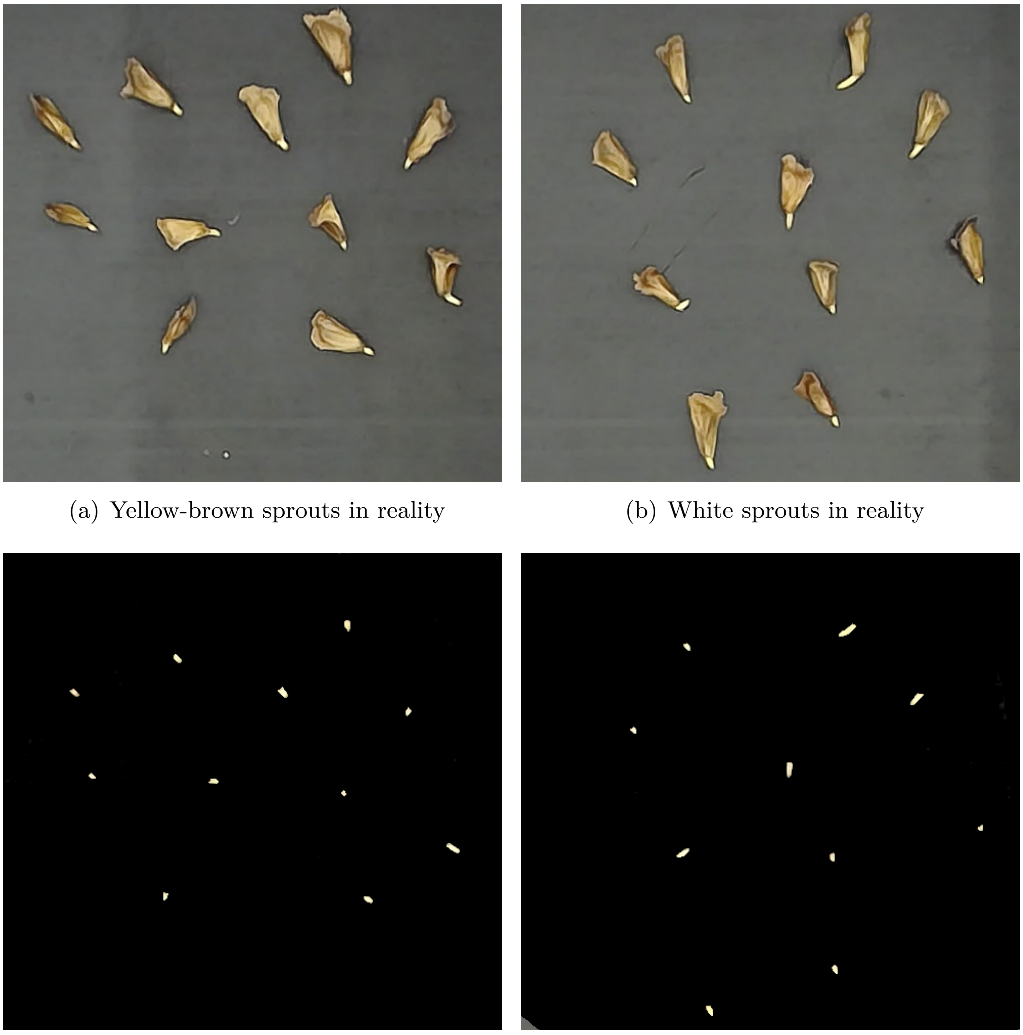
5.3.4 Choice of camera

Using a hyper spectral camera in the project was considered. A hyperspectral camera, compared to a normal RGB camera, gives spectral and spatial data simultaneously. Data from a hyperspectral camera is typically structured in a datacube, where the spatial data is the Y-X plane and the spectral is the Z direction [29]. This is useful for exploring the image response at given wavelength in the electromagnetic spectrum. All materials reflect light differently, i.e. the radiance varies in wavelength when the material change. This has been very useful to detect camouflage vehicles in open areas [14], classify different rice cultivars [18] and finding grape seed characters [26].

Many companies that sells hyperspectral camera [3], [1], [6] and [5] only show the price of their camera through quotes. It has not been possible to find a price list of hyperspectral cameras. Therefore it is assumed that a hyper spectral camera is in the high-end regarding cost. Define a suitable camera for this project is a challenged task, since several parameters exists. In this project the three most important parameter is the complexity of interfacing, the cost and the availability of the camera.
As described in section 5.1, the approach is to have the the chain of the project up and running before any component can be optimized. It was decided to start out with a relatively cheap and easy interfacing camera. By experience from previous courses on SDU, interfacing the USB Logitech C930e web camera is a straightforward process by using OpenCV library. A code example in appendix **??** show how to stream images from a USB web camera.
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
- Logitech camera was available at SDU
- The price of the web camera is currently 1199 DKK [21].

However using a webcamera showed later in the project that sprout would have a more whiter color in the image compared to the human perception. Images (a) and image (b) in figure 5.16 shows seeds with a natural white sprout color, but it is only the left image (a) that had the white color in reality. The seeds in the right image (b) were too yellow and brown. At first glance the right image (b) do not show any significant difference in RGB values than the left image (a), which is a major fault. In order to conclude that the RGB webcamera fails in see the difference, the images were further analysed.
The sprout was extracted for image (a) and (b) and the result is image (c) and (d) respectively in figure 5.16. The extraction of sprouts was performed using the GNU Image Manipulator Program (GIMP).
A 3D plot of the RGB value for image (c) and image (d) was constructed. THe RGB values is in range 0 - 255. The black background pixels, with zero value, were filtered to insure proper mean and standard deviation calculation. A blue and red star is added to



(a) Yellow-brown sprouts in reality (b) White sprouts in reality

(c) Yellow-brown sprouts extracted (d) White sprouts extracted

Figure 5.16: Image (a) with a yellow-brown sprout color. Image (b) with a white sprout color. Image (c) and (d) shows the extracted sprouts for image (a) and (b) respectively.

the 3D plot to indicate the mean of the samples for the yellow-brown and white sprouts respectively. The 3D plot is shown in figure 5.17. The calculated rounded mean and standard deviation is shown in table 5.1.

	Yellow/brown	White sprouts
mean (R, G, B)	(232, 226, 186)	(244, 239, 193)
std (R, G, B)	(28, 29, 28)	(17, 19, 19)

Table 5.1: Table showing the mean and standard deviation of the the yellow-brown and white sprouts

From the result from the 3D plot in figure 5.17 and table 5.1, a great variance is shown among the data. One reason for this can be the manual selection of pixel through GIMP. Another reason can be the light condition and a third reason can be camera parameter setting. A final step before conclude that there is no significant difference between and hence the webcamera do not with the current settings give a useful difference between white and yellow-brown sprouts, the statistical tool analyse of variance can be used. However using such tool the assumption is that the data has a normal distributed and the data is independent. The first assumption do not hold, as shown in the histogram in figure 5.18. Secondly the sprout data is depended, since the value of the sprouts

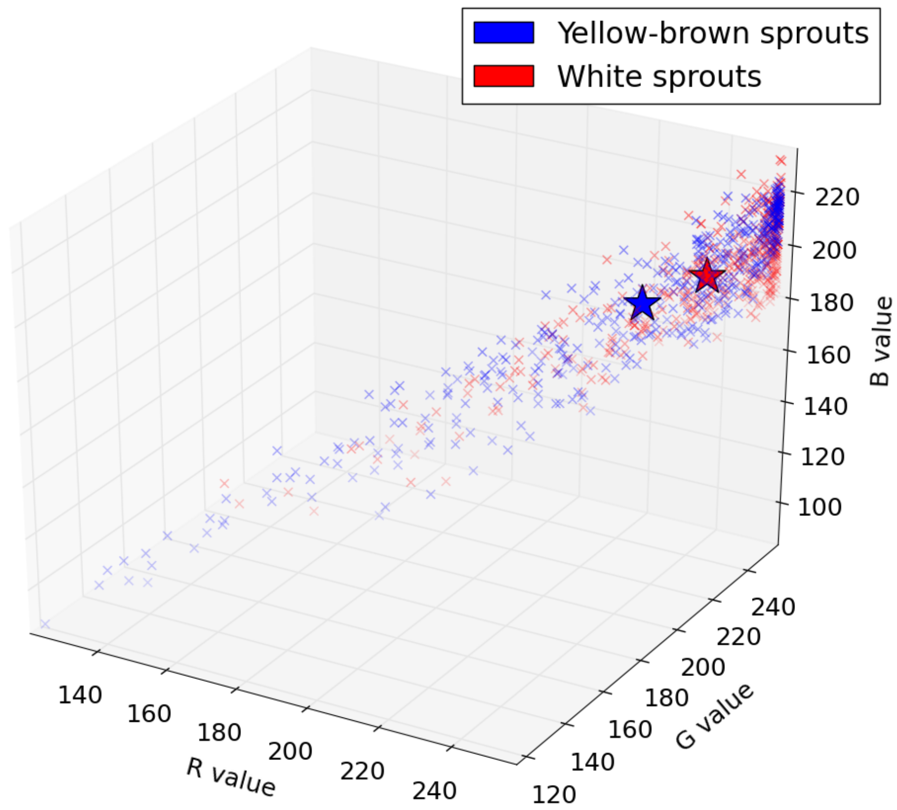


Figure 5.17: Plotting the RGB values for yellow-brown and white sprouts, togeter with their mean (Star

are closely related to the neighbour pixels. This conclude that there is no significant difference between the sprout pixels in figure 5.16 and hence the current settings with the webcamera is unsuccesful in producing RGB pixel values that correspond to ground truth.

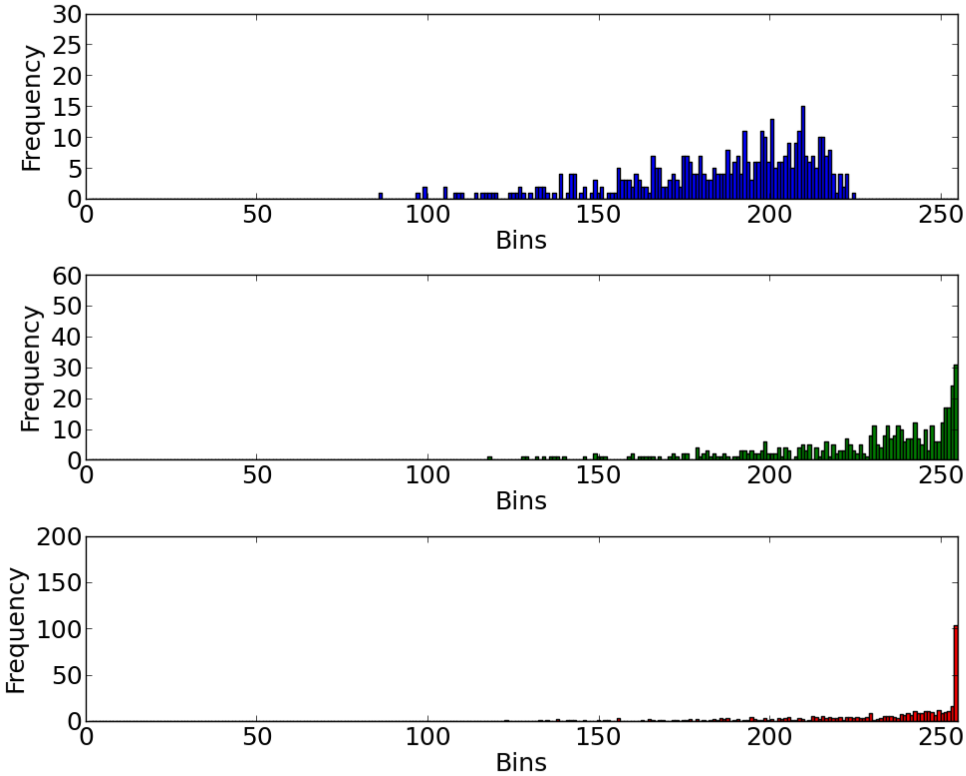
5.4 Time management

To insure progress in the project, a 14 days project meeting interval was held, where the author and supervisor were present. For each meeting a meeting agenda was created by the author with status, questions and schedule for the following 14 days work. Additionally the author created an logbook with headlines summing up the work of the day. All has been shared with the supervisor on Google Drive. The link is available in chapter 2.

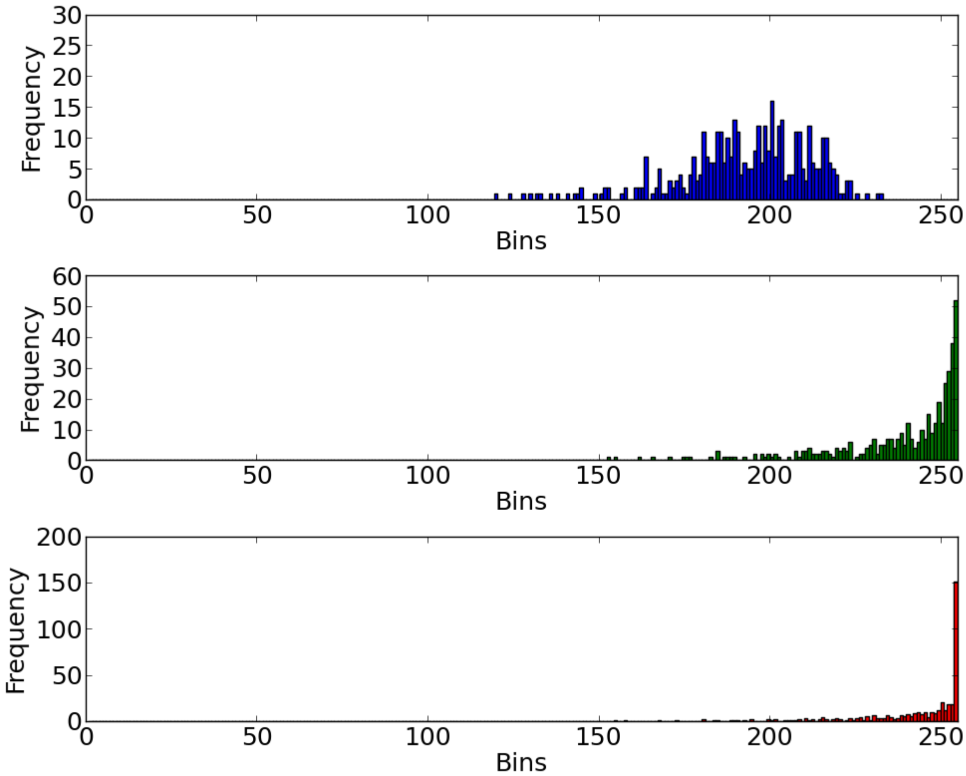
5.4.1 Company contact

Within this project, company visits has been made. First to see how the setup looks on site at the company ImproSeed ApS. Secondly to performed tests on site, in order to create image data. The communication between the author and project owner was established early in the project, in order to involve project owner as much as possible. It was agreed between the project owner and author to have a work plan ready after the project ends. This workplan is described in appendix C.

Hør efter møde med Henrik, hvad jeg gør med den plan



(a) RGB histogram of the yellow-brown sprouts



(b) RGB historam of the white sprouts

Figure 5.18: RGB histogram of (a) yellow-brown and (b) white sprouts.

6 Computer vision system

In this chapter, the computer vision system is described separately in more details. This system contains different components, which is illustrated in figure 6.1. These component are further described in the following sections.

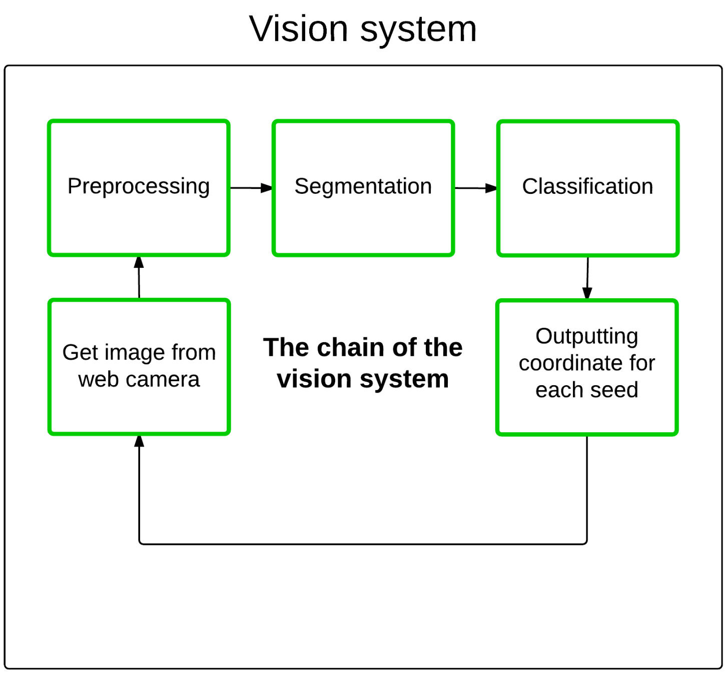


Figure 6.1: The chain of the project for the vision system

6.1 Get image from web camera

Argument for choosing the web camera, Logitech C930e is described in 5.3.4. The process of getting the image from the web camera is performed using VideoCapture class from the OpenCV library. A code example is shown in appendix **??**.

CameraSettings

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 6.1, where the autofocus and auto exposure is disable and manually adjusted together with the sharpness parameter.

```python
import os

os.system('v4l2-ctl -d 0 -c focus_auto=0')
os.system('v4l2-ctl -d 0 -c exposure_auto=1')
os.system('v4l2-ctl -d 0 -c focus_absolute=40')
os.system('v4l2-ctl -d 0 -c exposure_absolute=250')
os.system('v4l2-ctl -d 0 -c sharpness=200')
```

Listing 6.1: 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`

6.2 Preprocessing

When an input RGB images from the web camera is loaded into the system, within the preprocessing component, a *front ground* image, a *sprout* image and a combined *seed and sprout* image is produced as three outputs images. The flow is shown in figure 6.2.

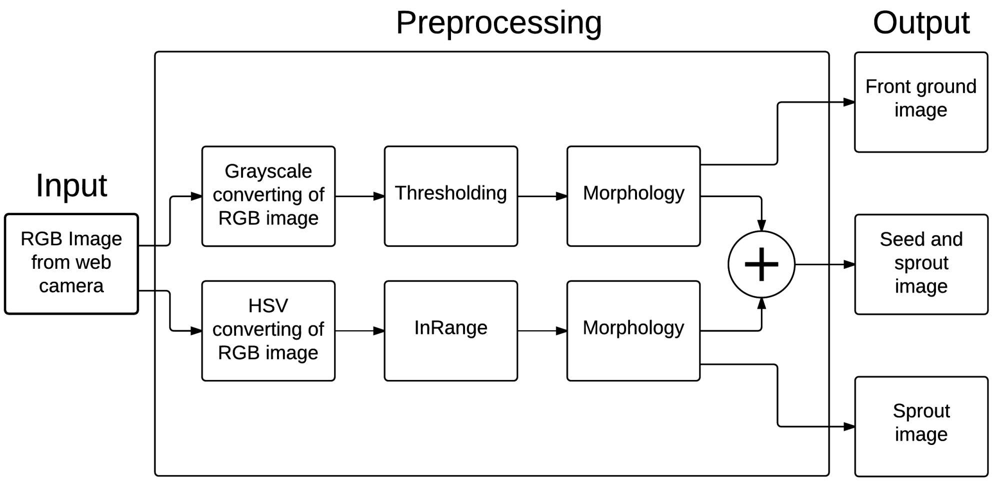


Figure 6.2: Preprocessing flow in order to generate the front ground, sprout and the combined seed and sprout image.

The front ground image is produced by converting the input image to grayscale, threshold it and finally apply the morphology process *closing* in order to repair the seed structures after the threshold. The conveyor belt, i.e. the background is relatively darker than the seeds and sprouts, which make a simple threshold sufficient. The threshold value was chosen to be 128. A sprout image is produced by converting the RGB image to a HSV image, filter out pixels using the inRange function from the OpenCV library and finally use morphology to repair the sprout structures. At the end the seed and sprout image is produced by adding the sprout image with the front ground image. The *Closing* morphology process is applied in order to close the gabs within each seed. This is performed by first dilate the image which expand the front ground pixel and thereby fills out holes and afterwards shrink the structure back to original state by erosion. A dense 3 x 3 kernel is used. In figure 6.7 and figure 6.8 shows an example of a front ground image before and after the morphology respectively.

An example of an RGB input image is shown in figure 6.3. Within this image, a red and a green rectangle are placed. This is not part of the original image, but is added in order to indicate the regions of interest (ROIs), which is described in subsection 6.2.1.The three output images is described as followed:

- Front ground image, figure 6.4 is a binary image, where a whole seed with sprout has pixels intensity values of 128. The rest of the pixels are black.
- Sprout image, figure 6.5 is a binary image, where the sprout pixels has intensity value of 255. The rest of the pixels are black.
- Seed and sprout image, figure 6.6 is the combination of the front ground image, figure 6.4 and the sprout image, figure 6.5. The result is an image where background pixels are black, seed pixels are gray and sprout pixels are white.

Ideally all gray pixels belongs to the seed and all white pixels belongs to the sprout. However scenarios happens, where seed pixels is is processed as sprout pixels. This is discussed further in section **??**. .

Ref to where we discuss the fact that there is some unertainscy

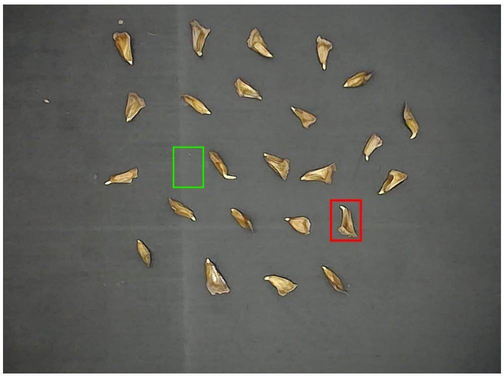


Figure 6.3: Input RGB image

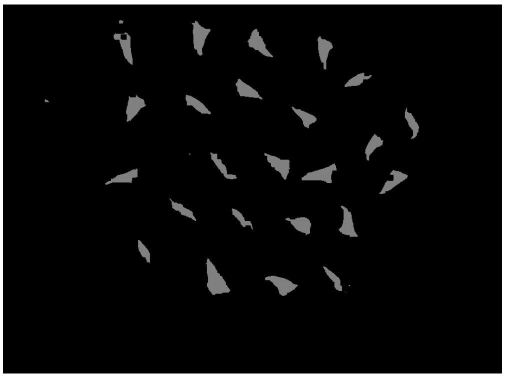


Figure 6.4: Front ground image

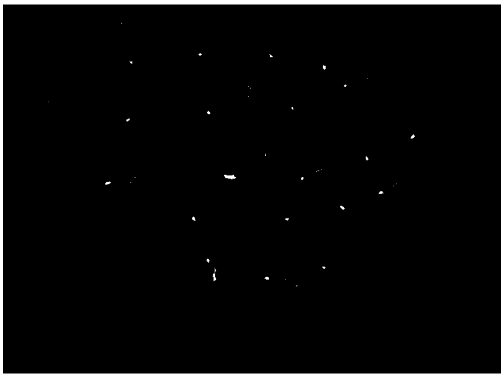


Figure 6.5: Sprout image

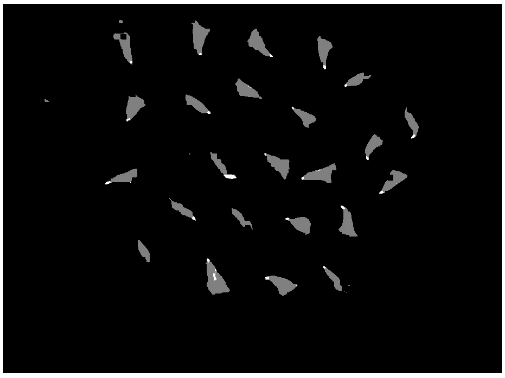


Figure 6.6: Seed and sprout image

Figure 6.7: Front ground before morphology

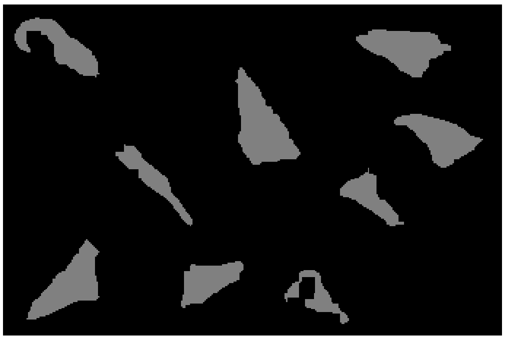


Figure 6.8: Front ground after morphology

6.2.1 Choice of using HSV compared to RGB method

As described in section 6.2, the RGB pixel was converted to the HSV colormap in order to find the sprout pixels. Instead of using HSV, the sprout pixels could be extracted by setting the RGB parameters directly. In order to see if this makes any difference, a HSV vs RGB test was initiated. For simplification a ROI of a single seed with sprout was cropped out of the test image. The dimension is 55 x 74 pixels. In order to have a portion of background pixels, another ROI of the background was created. These ROIs are indicated by a red and a green rectangle respectively in figure 6.3. The test included the following images:

- ROI of seed with sprout, figure 6.9
- ROI of background, figure 6.10:
- Seed image, where non-seed pixels was set to 0, figure 6.11
- Sprout image, where non-sprout pixels was set to 0, figure 6.12

A 3D plot in figure 6.14 and figure 6.13 shows the color map of RGB and HSV respectively. The result heavily depends on the accuracy in the pixel selection. From the 3D plots non of the color mapping approaches can be claimed to perform better than the other. From literature [27], the HSV method separate the intensity from the color information and makes the HSV colormapping invariant to certain types of highlights, shading, and shadow in the image. This makes the HSV more practical for the human interpretation [15]. Therefore the HSV method is the chosen approach in the preprocessing part.

Figure 6.9: Cropped image

Figure 6.10: Cropped background image

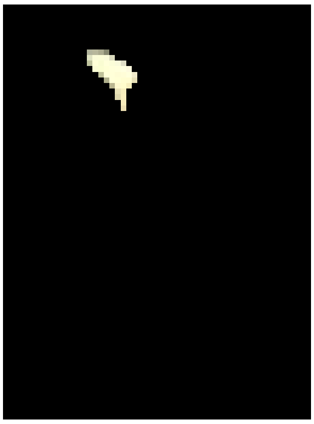


Figure 6.11: Sprout pixels only

Figure 6.12: Seed pixels only

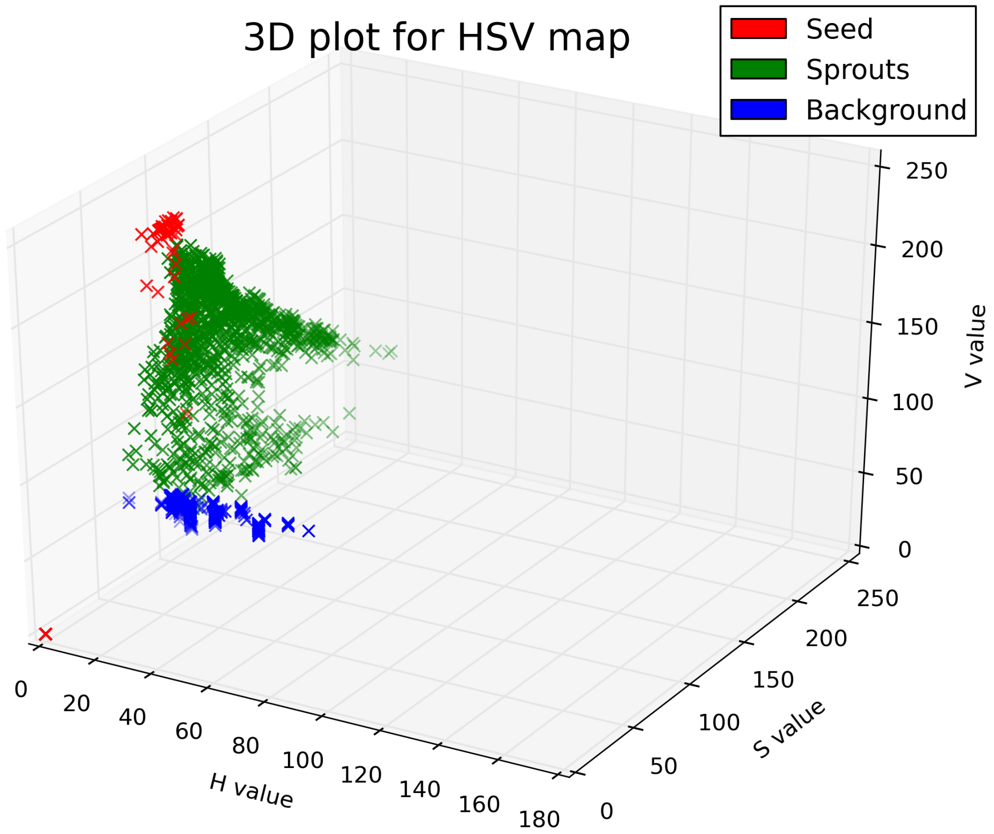


Figure 6.13: Preprocessing flow in order to generate the frontground , sprout and the combined seed and sprout image

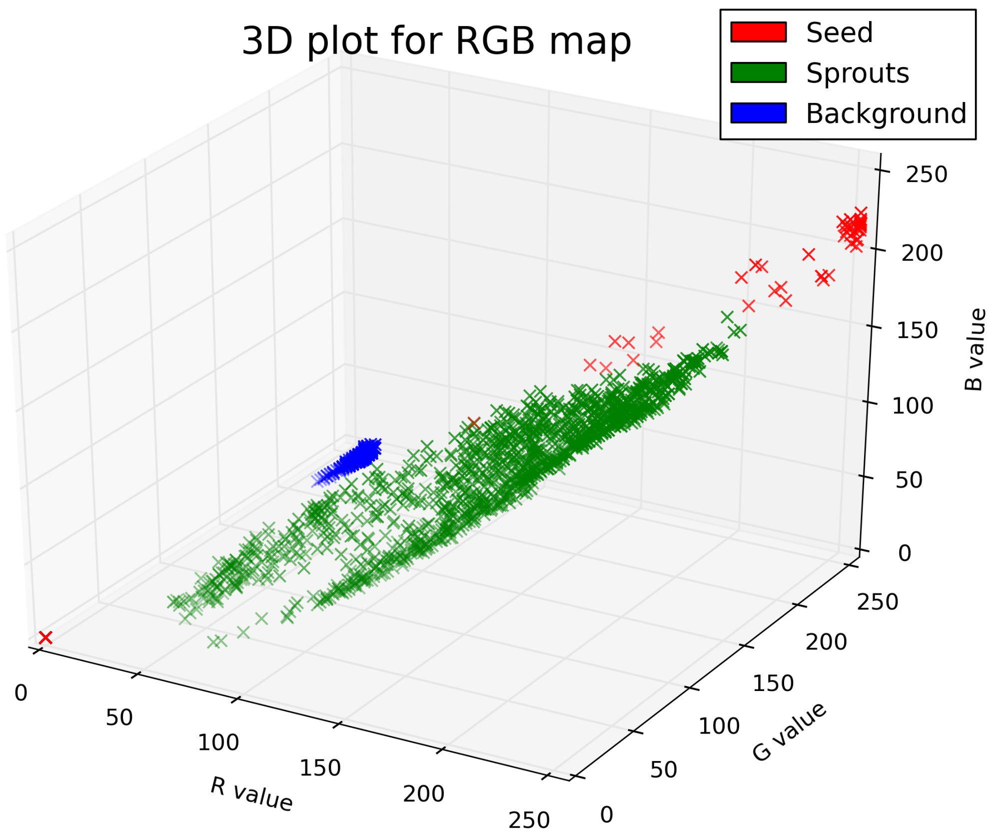


Figure 6.14: Preprocessing flow in order to generate the frontground , sprout and the combined seed and sprout image

6.3 Segmentation

The three output images *Front ground image*, *Sprout image* and *Seed and sprout image* from the preprocessing component described in 6.2 is used as input in the segmentation component. The output is a list with features extracted for each seed in the RGB image. The flow of the segmentation process is shown in figure 6.15.

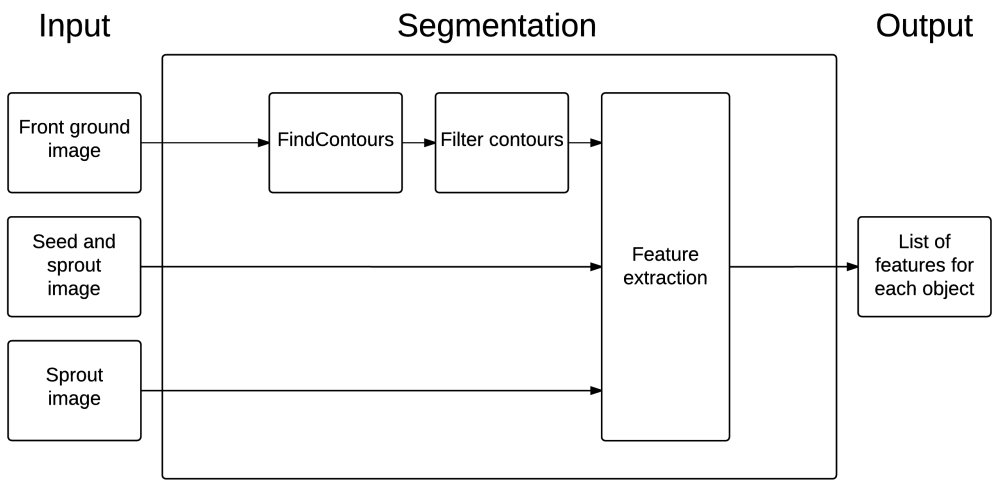


Figure 6.15: Segmentation flow in order to generate the list of features for each object in the RGB image.

6.3.1 Filter contours

The front ground image is segmented using the function FindContours from the OpenCV library. This function segment a binary image into contours and returns a list of all the contours found in the binary image, including any left over noise blobs, which were not removed by morphology in the preprocessing component. A contour contains the (x,y) location of edge pixels for any given shape. To filter out the noise contours, which e.g. could be an artefact or a piece of any material from the conveyor belt, a contour area threshold is implemented. This is done by using the findContourArea function from the OpenCV library.

In order to find a threshold value of minimum and maximum contourarea, a test image was analysed, which is shown in figure 6.41. This test image has been produced and edited in Gimp (GNU Image Manipulator Program) in order to have a mix of different types of objects, i.e. some objects with long sprout, some with small sprouts and last some without sprouts.
To analyse the test image, a histogram of the contour area for all contours in the test image is shown in figure 6.17. By inspecting the histogram from 0 to 157 on the x-axis, it is expected that noise blobs are the reason for five detected contours with a contour area from 0 to 157 square pixels. The maximum contour area is 1574 square pixels. A minimum 200 square pixels seems reasonable for cutting off the noise contours in the images. A upper threshold is set to 2000 square pixels.
Regarding the histogram in figure 6.17, five noise blobs were found in the test image which goes into the bin between 0 and 157 square pixels. In order to visualize these objects that were below the 200 minimum contour area threshold, the noise contours is illustrated in figure 6.18. The list of contour areas is: 2.0, 18.5, 0.0, 127.5, 29.5 square pixels. The result is by calling the *findContourArea* from the OpenCV library. From the reference, the area return from the function will almost always differ from number of

ref this: http://answ of-a-single-pixel-object-in-opencv/

Figure 6.16: Test image with a mix of 44 seeds.

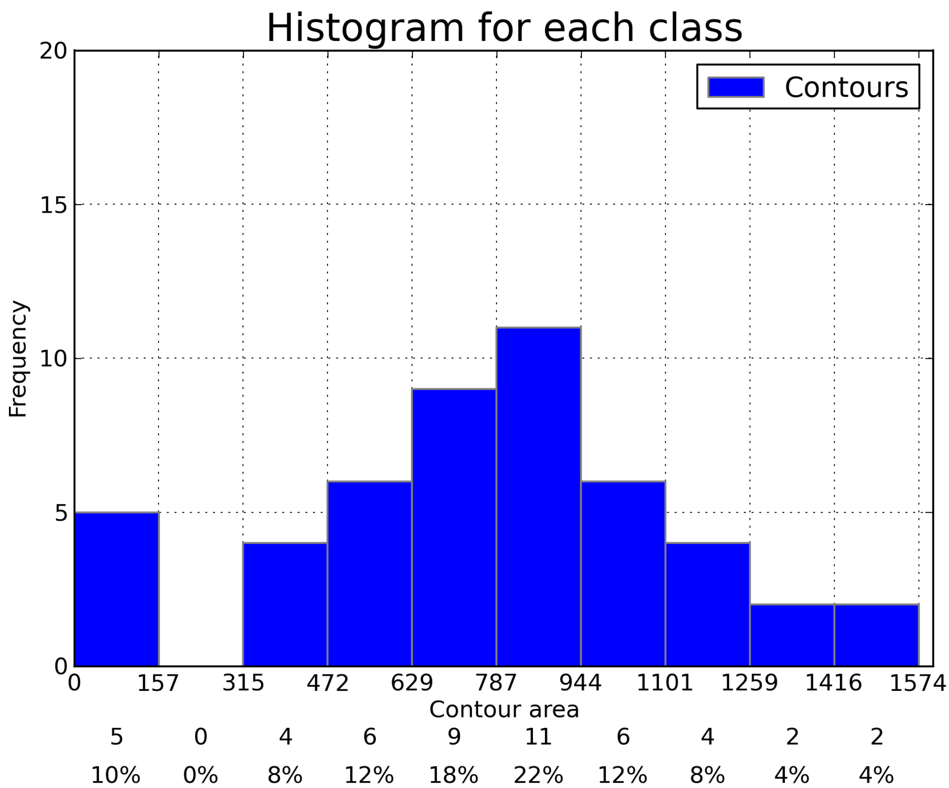


Figure 6.17: Histogram of contour area in the test image with 44 seeds and 5 noise blobs

white pixels. From the reference, if a contour area is 0.0, it means that the given contour has only one pixel. If a contour has an area of 0, the center of mass coordinate is placed in the (0,0), i.e. the upper left location in the image.

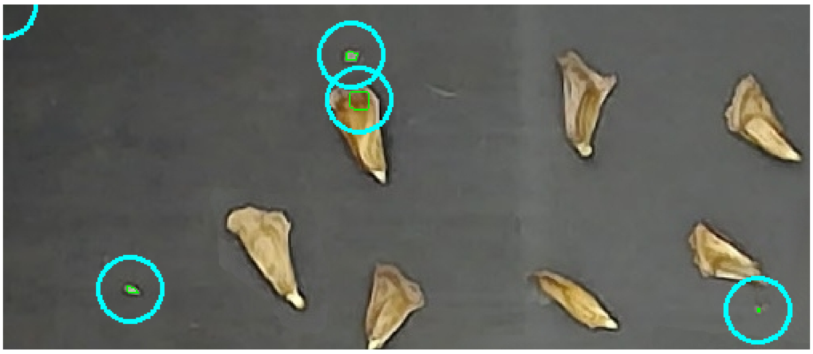


Figure 6.18: Indication of contours with area below 200 square pixels

In order to see the effect of the min and max contour area threshold range, a test was carried out. For better visualization the effect of the min and max contour area threshold, a ROI of For better visualization the effect of the min and max contour area threshold was cropped out from the input images in figure **??**. The test included the following images:

- ROI of input image, figure 6.19
- ROI of front ground image after the morphology process, figure 6.20
- ROI of drawn contours before filtering, figure 6.21
- ROI of drawn contours after filtering, 6.22

The test shows how the smaller noise contours is removed, by setting the minimum contour area to 200 square pixels. The effect of having a maximum contour area of 2000 is not illustrated in the test. However from the histogram in figure 6.17 the maximum contour area from the input image is 1574 square pixels, hence the upper threshold is set to 2000 square pixels given the current camera setup. Is the camera mounted closer to the conveyor belt, this maximum threshold needs to be adjusted. Additionally if two objects are touching each other, the contour that covers two or more objects will be filtered out. This topic is further discussed in the section

Here disucss that 2000 square pixels are perhaps a little to little if e.g two objects are touching each other. Or we have the camera mounted closer to the conveyor belt. Important topics!

6.3.2 Feature extraction

The input for the feature extraction is a list of filtered contours, the sprout image and the combined seed and sprout image. Each single seed is analysed pixel by pixel. If the pixel is white, it is a sprout pixel and if the pixel is gray, it is a seed pixel. After the list of seed and sprout is generated for each seed, 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.
It is a difficult task to categorizing an object, i.e. a seed with or without as either good or bad.
There exist uncertainty in the classification even for a human supervisor. However the classification is based on rules of thump which are described as followed:

- An object is categorized as bad if:

Figure 6.19: ROI of input image

Figure 6.20: ROI of thresholded image

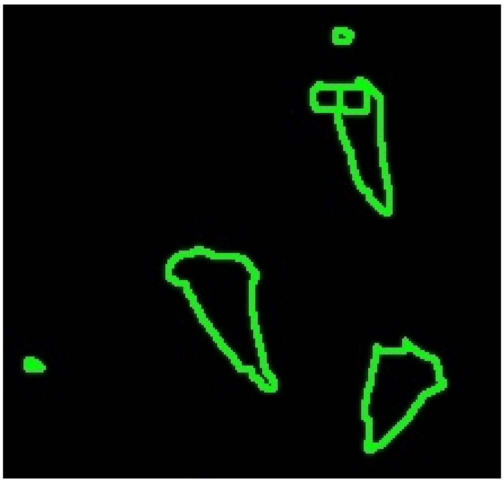


Figure 6.21: Founded contours with min contour area is zero

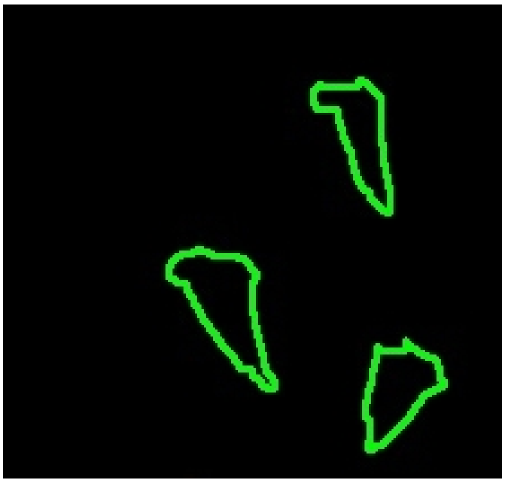


Figure 6.22: Founded contours with min contour area is 200

  - No sprout exist within the object, figure 6.23.
  - The length of the sprout is longer than 3 mm, figure 6.24.
  - The sprout is curved or twisted, figure 6.25.
  - The color of the sprout is yellow or brownish, figure 6.26.

- An object is categorized as good if:
  - The length of the sprout is between 1 to 3 mm, figure 6.27
  - The color of the sprout is white, figure 6.28

If a condition which categorize an object as good is present while a condition which categorize the object as bad, the object is categorized as bad. E.g. if the sprout of an object is white, but longer than 3 mm, then the object is bad. A problem in defining when a sprout is yellow exist. This problem is discussed in the section

Here describe the problem with seeds that should be brown is sampled as white pixels. Perhaps make a plot between pixel that are "brown" and pixel that are white. It would be two histograms where we have different bins of hue values on x axis and frequency on y-axis.

Figure 6.23: No sprout

Figure 6.24: Sprout too long

Figure 6.25: Curved sprout

Figure 6.26: Brownish sprout

Figure 6.27: Sprout OK

Figure 6.28: Sprout OK

Picking the right features is a hard task. However the main factor in deciding the category for an object is to analyse the sprout. The sprout information is collected by finding the oriented boundingbox (OBB) of the sprout pixels. In the following figures 6.29, a ROI has been cropped out for better visualization. In figure 6.30 the OBB around the sprout pixels is drawn. The red pixels is not a part of the image data.

Figure 6.29: Cropped out object from RGB input image

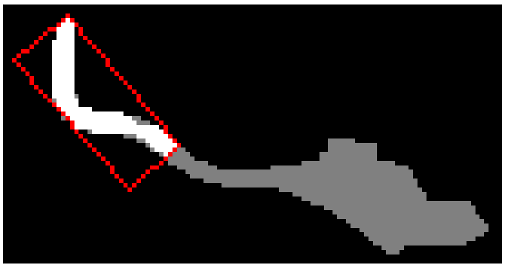


Figure 6.30: A red OBB drawn around the sprout pixels

For each OBB, the length, width, ratio and number of sprout pixel is extracted. In order to differentiate between sprout that has a white nuance compared to yellow or brownish nuance, the mean and standard deviation of the sprout pixels is analysed. All in all it boils down to the following list of features, that is extracted within the segmentation component:

- Length of OBB
- Width of OBB
- Ratio $\frac{Width_{OBB}}{Length_{OBB}}$
- Number of sprout pixels within the OBB
- The mean hue value for the sprout pixels
- The standard deviation for the sprout pixels

The OBB is found by using the *minAreaRect* function from OpenCV library. This function returns the center of mass (COM), the width, the height and the orientation of a contour. The attributes are not invariant due to rotation, hence extra functionality is implemented to insure the attributes *length* and *width* is always the longest and shortest side of a bonding box respectively. The COM coordinate is available within the attributes of the function *minRectArea*. However calculating the center of mass is available through the use of moments. To see the difference between the two methods, a comparison is shown in figure 6.31. The red dots in the figure indicate the contours COM using the *minRectArea* and the green dots indicate the COM calculated by using moments. Taking into account, that the objects in the image is between 10-15 mm long, the difference between red and green COM do not play any important role. It all comes down to the type of grasping, which in this case will be performed by an pneumatic suction end-effector. At the moment the methods of using moments is used, since this was implemented before using the *minRectArea* function. Argument for using the the *minRectArea* would be to save the computational time by using moments. However this is a is a task for future work.

I really dont have any arguments for stick with the moments. So at the end write that I change to use use COM from the minRectArea function and that saved this amount of time

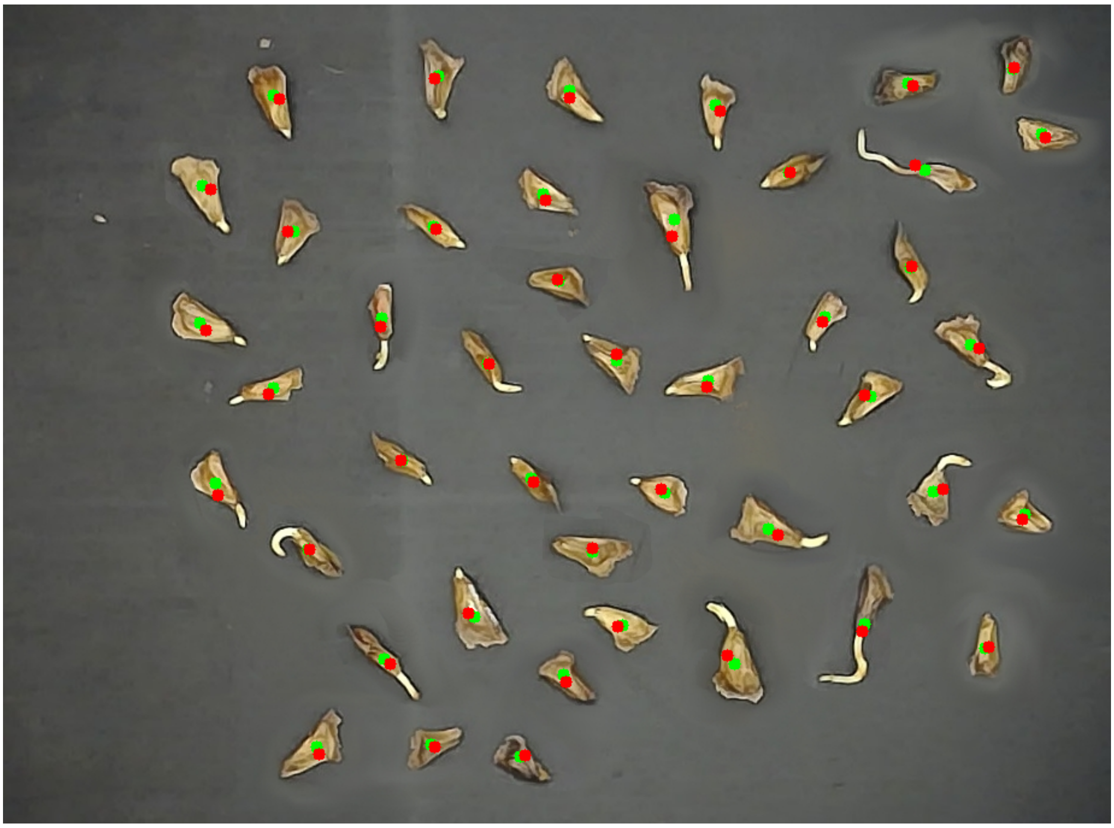


Figure 6.31: Red dots indicate COM using *minRectArea*. Green dots indicate COM using moments

Write a little about moments here. See doc from Summerkursus or individual projects

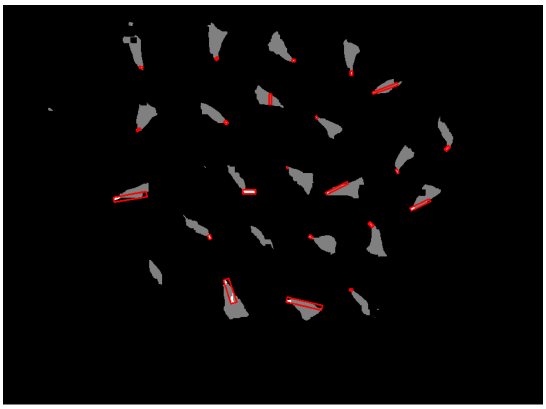
6.3.3 Clustering

As described in section 6.3.2 the feature extraction is based on the OBB that span the sprout pixels for each contour. The data from the feature extraction is later used in the classification module in section 6.4. In order to maximize the classification rate, it is important to minimize the false negatives and false positive, i.e .type I and type II errors. It is hypothesized that each object in a image is good. A type I error will occur if an object is classified as bad, but confirmed by a supervisor as good. Vice versa a type II error will occur if an object is classified as good, but the supervisor confirms the object is bad.

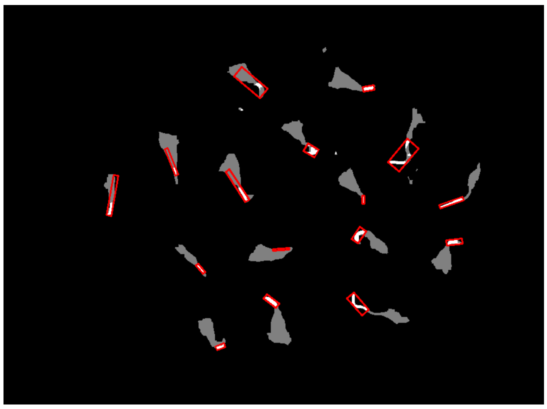
In order to measure the type I and type II errors, two tests is carried out. Test 1 contains an RGB image with good objects. Test 2 contains an RGB image with bad objects. These are shown respectively as (a) and (b) in figure 6.32. Both dimensions of the images is 920 x 680 pixels. Doing the test, each image (a) and image (b) was preprocessed. The preprocessing component is described in section 6.2. The result of the preprocessing component of image (a) and (b) is shown in the same figure as image (c) and (d) respectively.

(a)

(b)



(c)



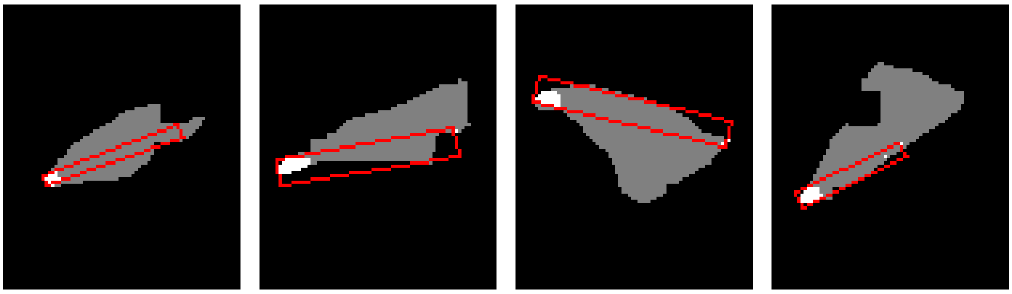
(d)

Figure 6.32: Two RGB images. The left image (a) contains good seeds and the right image (b) contains bad seeds, since the sprouts are too long

Perhaps (c) and (d) should be removed or placed in the appendix together with (a) and (b) since a big image needs big space to see the details. This is not true with (c) and (d)

Doing analyse of the sprouts for each object, the bounding boxes is extracted and drawn with a red rectangle. The drawing is performed by using the *drawContours* from the OpenCV libray. In figure 6.30 the OBB is fitted correctly around the sprout pixels by comparing the white sprout pixels with the RGB image, i.e. no type I or type II error. However this is not always the case. An example with four images in figure 6.33 shows incorrectly OBB. All the images is size 74 x 89 pixels and is cropped out from an original *Seed and sprout* image . By observing the OBB is spanning white pixel, that do not belong to the sprout area of an object.

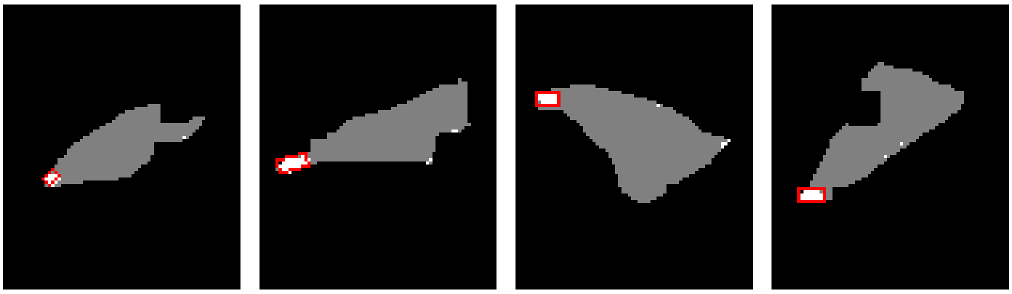
Store all the big images in appendix, that we dont show but still is referred to



(a) (b) (c) (d)

Figure 6.33: Main figure caption

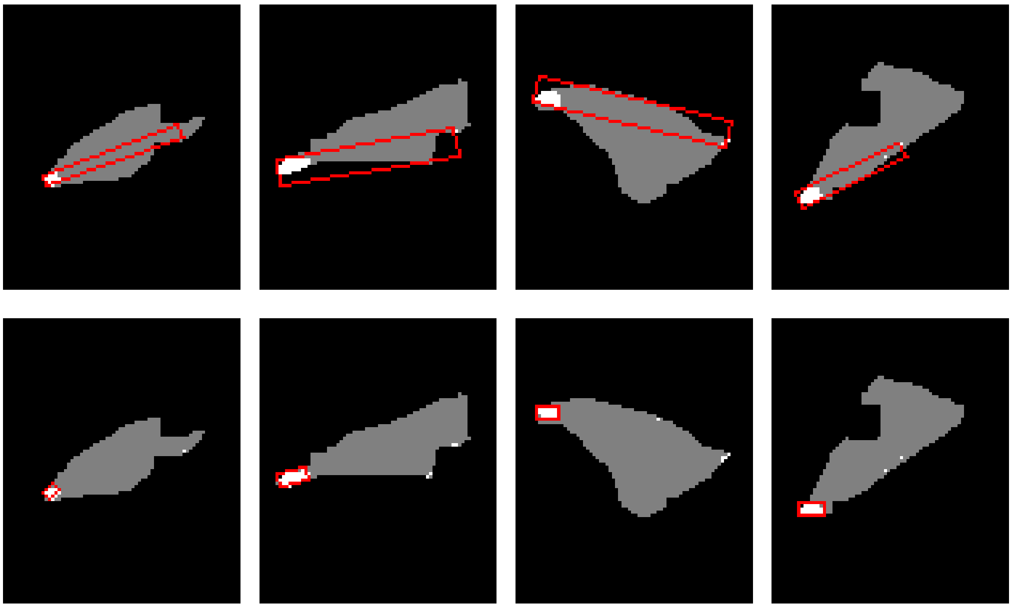
By observing the drawing of the bounding box, sometimes an extracted bounding box is spanning white pixel, that do not belong to the sprout area of an object. This is shown in the following figure



(a) (b) (c) (d)

Figure 6.34: Main figure caption

The image in **??** show how the potential errors will accor, if the seeds look like as above. Here a more sophisticated algorithm is needed. An extra check on how the neiboorhood is. figure **??** If the founded "real" sprout list is surrounded only or almost by gray pixel then take the other. It is hard to tell what the solution should be.



(e) Angry Smiley (f) Angry Smiley (g) Angry Smiley (h) Angry Smiley

Figure 6.35: Main figure caption

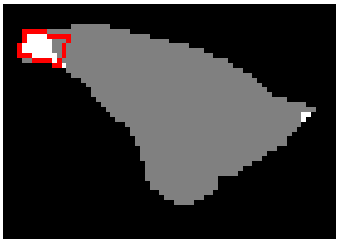


Figure 6.36: Class 1

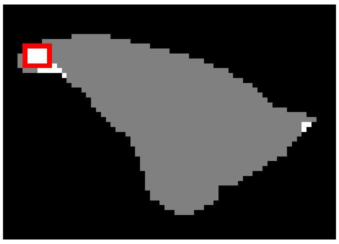


Figure 6.37: Class -1

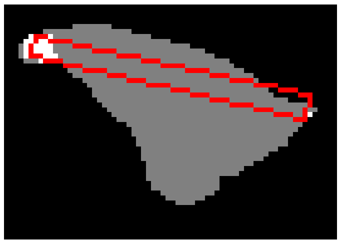


Figure 6.38: Class 0

6.4 Classification

Three images, where the first two are training data and the last is testing data, has been analysed to find a proper minimum contour area threshold. Each image is explained as followed:

here add some part of the AI stuff with training data etc

- Training data, class 1: Seeds with optimal sprouts, figure 6.39
- Training data, class -1: Seeds with too long sprouts, figure 6.40
- Testing data, class 0: Seeds with different sprout length, figure 6.41

Figure 6.39: Class 1

Figure 6.40: Class -1

Figure 6.41: Class 0

6.5 Outputting coordinates

7 Robotic system

8 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....

9 Robotics

An future extension of the Master Thesis is to interface the ABB Flexpicker robot, which is shown in figure 9.1, 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 9.1: The ABB Flexpicker robot that do the pick-and-place task after seeds has been classified by the vision system

The interface between the vision and robotic vision is a ROS topic, where a 3D pose of seeds for grasping would exist.

10 System test

Nothing yet

11 Discussion

Nothing yet

12 Conclusion

Nothing yet

13 Future work

This is the future work section

14 Bibliography

[1] Hyspex, .
URL http://www.hyspex.no/index.php.
[Online; accessed 2015-05-16].

[2] Python software foundation. python language reference, version 2.7, .
URL http://www.python.org/.
[Online; accessed 2015-05-11].

[3] Resonon, .
URL http://www.resonon.com/products_imagers_main.html.
[Online; accessed 2015-05-16].

[4] Ros-industrial support for abb manipulators(metapackage), .
URL http://www.http://wiki.ros.org/abb/.
[Online; accessed 2015-05-13].

[5] Surfaceoptics, .
URL http://surfaceoptics.com/.
[Online; accessed 2015-05-16].

[6] Ximea, .
URL http://www.ximea.com/en/products/xilab-application-specific-oem-cus hyperspectral-cameras-based-on-usb3-xispec.
[Online; accessed 2015-05-16].

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

[8] G. Bradski.
Dr. Dobb's Journal of Software Tools, 2000.

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.

[15] Rafael C. Gonzalez and Richard E. Woods.
Digital image processing.
Pearson Education, 2008.

[16] J. D. Hunter.
Matplotlib: A 2d graphics environment.
Computing In Science & Engineering, 9(3):90–95, 2007.

[17] Eric Jones, Travis Oliphant, Pearu Peterson, et al.
SciPy: Open source scientific tools for Python, 2001–.
URL http://www.scipy.org/.
[Online; accessed 2015-05-11].

[18] 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.

[19] Lei Li, Qin Zhang, and Danfeng Huang.
A review of imaging techniques for plant phenotyping.
Sensors, 14(11):20078–20111, 2014.
ISSN 1424-8220.
doi: 10.3390/s141120078.
URL http://www.mdpi.com/1424-8220/14/11/20078.

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

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

[22] Poramate Manoonpong.
Ai2 lecture in neurale network, 2014, available: 14-5-2015.
User: student, Password: ai2lecture.

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

[24] 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.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay.
Scikit-learn: Machine learning in Python.
Journal of Machine Learning Research, 12:2825–2830, 2011.

[26] 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.

[27] Gang Qian Shamik Sural and Sakti Pramanik.
Segmentation and histogram generation using the hsv color space for image retrieval.
http://www.cs.jhu.edu/~hwang/papers/Color_Imag_Segm_Dicta03.pdf, available: 19-3-2015.
10.1109/ICIP.2002.1040019.

[28] S. Chris Colbert Stéfan van der Walt and Gaël Varoquaux.
The NumPy Array: A structure for efficient numerical computation, computing in science and engineering, 2001–.
URL http://www.numpy.org/.
[Online; accessed 2015-05-11].

[29] Center for Space Research The University of Texas at Austin.
Hyperspectral remote sensing.
http://www.csr.utexas.edu/projects/rs/hrs/hyper.html, 2014.
[Online; accessed 2015-05-13].

[30] Hanzi Wang and David Suter.
Color image segmentation using global information and local homogeneity.
http://www.cs.jhu.edu/~hwang/papers/Color_Imag_Segm_Dicta03.pdf, available: 19-3-2015.

A OpenCV VideoCapture class

This appendix shows how to stream video from a USB web camera, by using the VideoCapture class from OpenCV libray. The argument *cameraIndex* needs to be corrected to the proper number, i.e. 0, 1, 2 ... depending on how many video devices the computer running the Python script is connected to. A simple method in Ubuntu to verify the index for the USB web camera is to launch the VLC media player and go to the *Capture Device* menu. Here select the proper index under *Device Selection* by testing the video output of the selected video device.

```python
import cv2
cap = cv2.VideoCapture(1)

while(1):

    # Capture frame-by-frame
    ret, frame = cap.read()

    # Display the resulting frame
    cv2.imshow('frame',frame)

    # If the user hit the ESC bottom, the program ends...
    k = cv2.waitKey(30) & 0xff
    if k is 27:
        print("User closed the program...")
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Listing A.1: Using VideoCapture class from OpenCV

B Test of webcamera parameters

The Python script described in section 5.3.4 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 B.1(b). 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

(a) AF=40, AE = 250, S = 200

(b) AF=40, AE = 260, S = 200

(c) AF=40, AE = 260, S = 255

(d) AF=50, AE = 260, S = 200

Figure B.1: Testing the parameters AF, AE and S. The best result is figure (b)

C Handleplan for Improseed ApS

Dansk
Status for projektet er som følgende:

- Vision system:
  - Det er muligt at anvende et USB webcamera, Logitech C930e til detektering af længde og bredde af frøenes spire. Pris via Logitechs hjemmeside ligger ved dags dato på 1199 kr.[21]
  - Det er ikke lykkes at opnå tilfredsstillende billeder ved brug Logitech C930e webkameraet til detektering af spirernes farvenuance. Ønskes denne feature, må der investeres tid til at afsøge markedet for potentielle kamera løsninger. Indkøbes der kamera med andet interface end USB, skal der laves nyt interface mellem kamerea og PC program.
- Robot system:
  - ROS-industri er igang med at udvikle en MoveIt pakke, til interfacing af ABB robotter.[4]. For yderlig information kontakt
    Shaun Edwards via email: `swri-ros-pkg-dev@googlegroups.com`

Ønskes der videre arbejde med kildekoden for projektet, kan dette hentes via Github på følgende adresse: `https://github.com/ChristianLiinHansen/MasterThesis`.

Dette PDF dokument kan findes på følgende adresse: `https://github.com/ChristianLiinHanse` `Master-thesis-doc`.

English
This is the workplan in English