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 1st 2014 - June 1st 2015

Todo list

Ryd op efter ord som "to" ift "two" og where i stedet for were. 7
Make figures like **??** 7
Hue moments? 7
Lav CD ? 7
Sikre at referencehenvisning ikke bare er tal, men forfatter og årstal 7
Check denne url `http://www.imm.dtu.dk/\protect\unhbox\voidb@x\penalty\@M\{}janba/MastersThesisAdvice.pdf` 7
Se ...*TestingInRoboLab*...*SegmentationWork*/*imgDrawClass*0.*png*. 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. 7
Fedt. Med OtSus optimale threshold så er frontgrond image bedre og kræver mindre morphology. Dette er nice optimering, da jeg før kørte 3 dilate + 3 erode for at fjerne huller i front ground image 7
Beskriv et sted, at hvis man havde det oprindelig setup inde i RoboLab ville projektet være kommet længere. Eksempelvis hvis den hvide omvendte webergrill blev afmonteret og taget fra Sunds til Odense. Så spildte man ikke tiden med at have en åndssvag trækasse med forkert lys i. 7
Print the whole document out and look at the figure. Do the look nice? Can you see what you should see? 7
Beskriv det faktum at med en hue color segmentering, så hvis de brune seeds ikke kommer med, så er hue featuren egentlig ikke relevant at bruge. Fordi hvis spireren er for brun, jamen så ses spiren slet ikke og dermed er der ikke nogle hvide pixel i hsv segmenteringen 7
Være sikker på at jeg forklare mit valg af features og dermed tester andre features 7
Det skal være således at læseren vil kunne se de problemer jeg har stødt på, og hvordan jeg har løst dem. Eksemplet med Perceptronen skal også fremgå, da jeg ikke vidste nok om classificering og på den måde var perceptronen er god måde at blive klogere på det med klassificering 7
Beskriv det med hvordan systemet virker, når der skal laves træningsdata 7
Test af hvilke features der var gode og hvilke andre features, der ikke var gode . . 7
See hvilke ting der skal rettes fra mødet d. 29. april. Bl.a. learnings goals og deliverables 7
VIGTIG ved aflevering. Læs `http://www.sdu.dk/Om_SDU/Institutter_centre/Mmmi_maersk_mckinney_moeller/Studerende/Vejledninger/Aflevering+af+specialerapport` 7
Need to finish this 9
Write something that match to related work here. 14
Evt have referencer her på, hvordan man selv vil kunne lave sit eget hyperspektralt setup 14
Draw this in google SketchUp? 29
ref this: http://answers.opencv.org/question/58/area-of-a-single-pixel-object-in-opencv/ 40
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! 42
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. 43
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 45

Write a little about moments here. See doc from Summerkursus or individual projects 45
Write that in discussion, that this will not be good, if two seeds are really close to each other. Since then two sprout clusters are present, in the image 48
Is that really important? There is a lot of papers out there that has investigated this before.... 55

Abstract

English

Nothing yet

Danish

Nothing yet

Contents

1 Preface **8**

2 Reading manual **9**

3 Introduction **10**
3.1 Project description . . . 10
3.1.1 Sensor for vision system . . . 11
3.1.2 Learn how other type of seeds looks in different categories . . . 11
3.2 Deliverables . . . 12
3.3 Learning goals . . . 12
3.3.1 Knowledge . . . 12
3.3.2 Skills . . . 12
3.3.3 Competence . . . 12

4 Related work **14**

5 Division of work **15**
5.1 Project methods . . . 15
5.2 Proof of concepts . . . 16
5.2.1 Black pepper detection . . . 16
5.2.2 Square and circle detection . . . 16
5.3 Optimizing . . . 23
5.3.1 Choice of camera . . . 23
5.3.2 Detecting real seeds . . . 23
5.3.3 2 class classification of real seed with Perceptron . . . 27
5.3.4 A new setup . . . 28
5.3.5 Optimizing the preprocesssing and segmentation component . . . 31
5.4 Time management . . . 34
5.4.1 Company contact . . . 34

6 Computer vision system **35**
6.1 Get image from web camera . . . 35
6.2 Preprocessing . . . 36
6.2.1 Choice of using HSV compared to RGB method . . . 38
6.3 Segmentation . . . 40
6.3.1 Filter contours . . . 40
6.3.2 Feature extraction . . . 42
6.3.3 Clustering . . . 45
6.4 Classification . . . 49
6.4.1 Result of SVM with different kernels . . . 50
6.5 Outputting 3D coordinates . . . 53

7 Robotic system **54**

8 Discussion **55**

9 Conclusion **56**

10 Future work **57**
10.1 Seeds that is close to each other . . . 57
10.2 Seed that is touching each other . . . 58
10.3 Semi-supervised learning . . . 58

11 Bibliography **59**

A Setup at ImproSeed Aps **62**

B Webcamera parameters **63**

C Test of webcamera parameters **64**

D Light bulb Kelvin test **65**

E OpenCV VideoCapture class **66**

F Workplan for Improseed ApS **67**

G Testing γ parameter for the RBF kernel **68**

H Testing C parameter for the RBF kernel **69**

Ryd op efter ord som "to" ift "two" og where i stedet for were.

Make figures like **??**

Hue moments?

Lav CD ?

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

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

Se *...TestingIn* Her ligger to frø for tæt på hinanden og påvirker den cropped boundingbox. Her skal der laves et check på hvilke

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, this reading manual has been added to give an extra explanation of the 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.
- Chapter 5 - Division of work: This chapter describes how the process of the project was formed from start to end. This chapter includes test which is used to argue for the chosen approach.
- Chapter 6 - Computer vision system: This chapter describes in more details how the different component in the computer vision system works. Result at test of the different component is described.

Need to finish this

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 defined as an object that may or may not contain a *sprout*. Image (a) in figure 2.1, is a seed without a sprout. Image (b) is a seed with a white sprout and image (c) is a seed with a longer sprout.

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

Figure 2.1: Defining seed and sprout.

3 Introduction

In collaboration between the company ImproSeed ApS (IPS) and the University of Southern Denmark (SDU), ideas for a master thesis project has been developed. IPS is working with forestry planting process to improve the efficiency of growing threes. At the moment an efficiency of the planting success lies between 60-75%, i.e. minimum 25% lost in profit. Therefore ISP has interest in maximizing the efficiency by sorting the seeds before planting process. The idea is to use a computer vision solution, where the system is able to differentiate the seeds into two categories as followed:

- *Green - good condition.* Seeds are ready for planting. Send seeds to planting process.
- *Red* - bad condition. Seeds are not valid for planting. Discard the seeds from current process.

Additionally a learning component should be implemented for letting the vision system be able to learn how a specific type of seed looks in the different categories. Improving this will make the system be able to classifier different kind of tree cultivars.
Implementation of this means that no seeds would be planted in the ground if they were 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. The classification component will use features to classify each seed. Example of features: If there is a sprout, how far is the sprout reaching out of the seed. An other feature is the color of the sprouts. Additionally if a sprout is bended or even cracked 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.

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 which share the same characteristic. An other training data would contains seeds with an other characteristic. Example: 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.

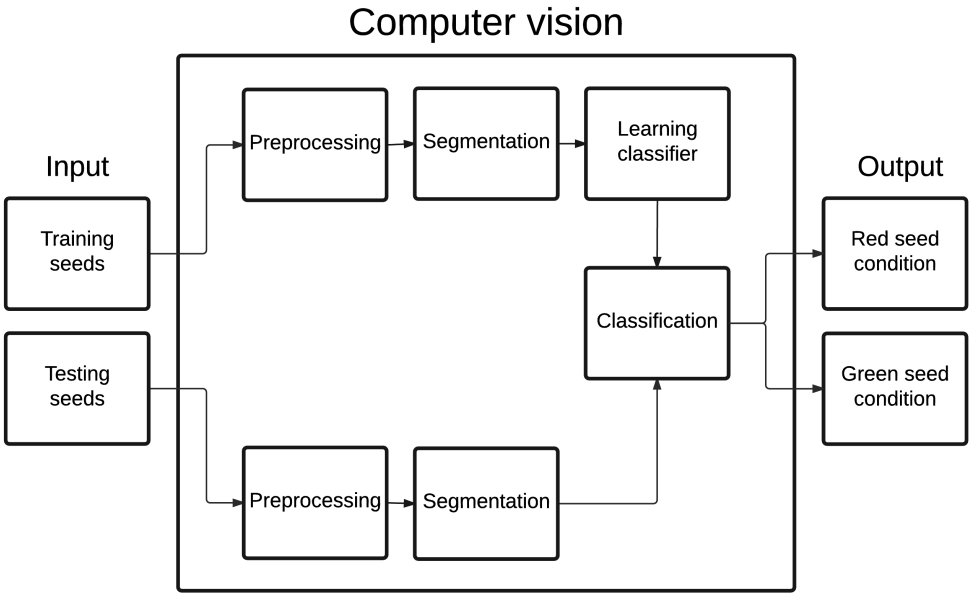


Figure 3.1: Blockdiagram of the computer vision system

- 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.
  - Green 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 camera. This techniques has been used in many different application, like detecting the quality of wine-grapes [31], rice cultivar identification [23] and many others like mineral exploration, agriculture, and forest production. [34]. 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 [24]. 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

3.3 Learning goals

The learnings goals for this Master Thesis project is defined by the SDU [15]. 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 [19] is illustrating the principle in figure 4.1

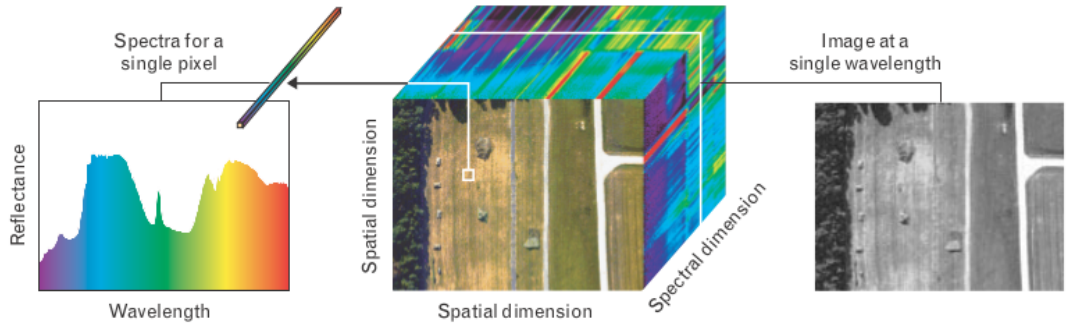


Figure 4.1: Hyper spectral imaging [19]

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

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 [5], OpenCV 2.4.9 [13], Numpy 1.6.1 [33], Matlibplot 1.1.1rc [21], SciPy 0.9.0 [22] and Sklearn 0.16b1 [30].

5.1 Project methods

As described in section 3.1, the main goal is to have a vision system which feeds images of seeds placed on a conveyor belt and then be able to detect, analyse and classify each seed regarding their condition. The output of the vision system is a 3D coordinate (x,y,z), which has been transform from the image frame to a robot frame. Then the coordinate is transferred to a robotic system in order to let a robot perform grasping at the location of the coordinate seen in the robot frame. 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 and use additionally time to optimize and increase the robustness of the given component. The pitfall is to not have data flow established in time. Additionally working with natural growing seeds is time consuming and can not 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.

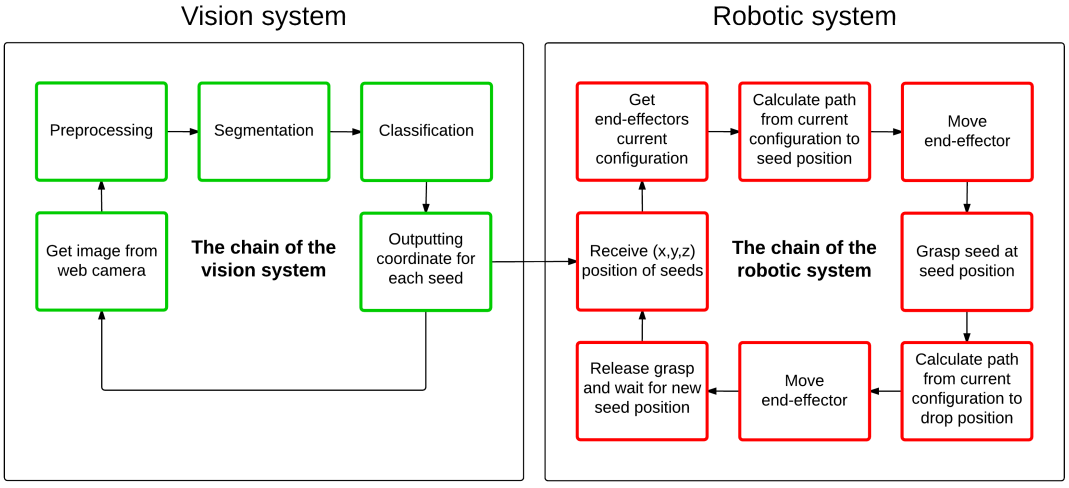


Figure 5.1: The chain of the project

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 illlustrated in figure 5.2. Knowledge of using preprocessing tools such as threshold, morphology together with the segmentation tools of finding contours, areas of contours and their center of mass location was gained in this proof of concept.

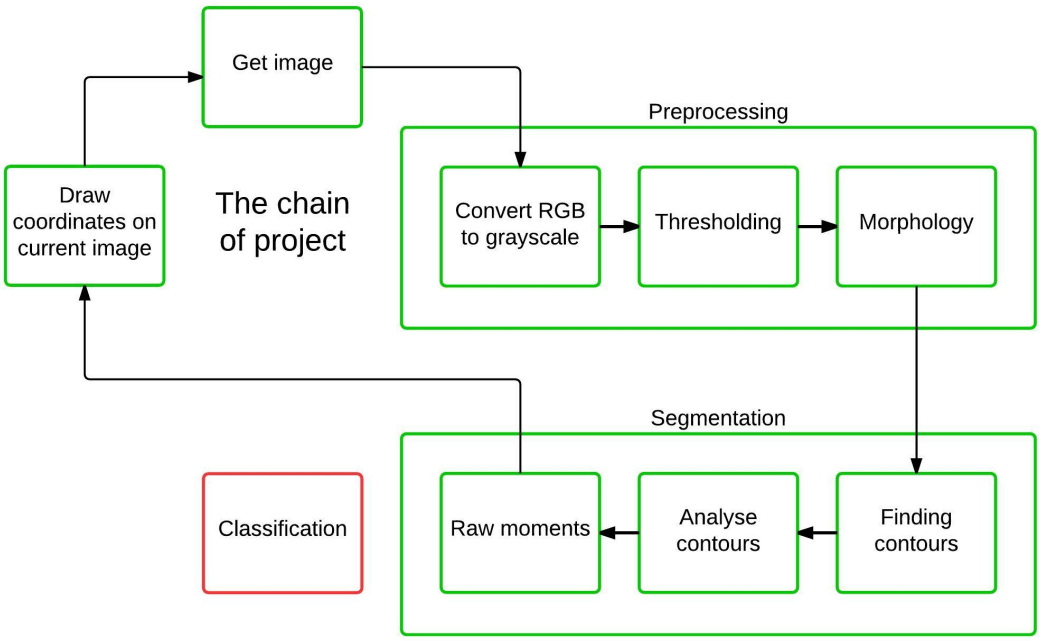


Figure 5.2: Testing proof of concept without classifier component. The green boxes indicate components that was implemented and the red box indicate the component that was skipped for this given implementation.

The system was tested on grain of black peppers. Inspired from source [29]. 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 [25]. 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.

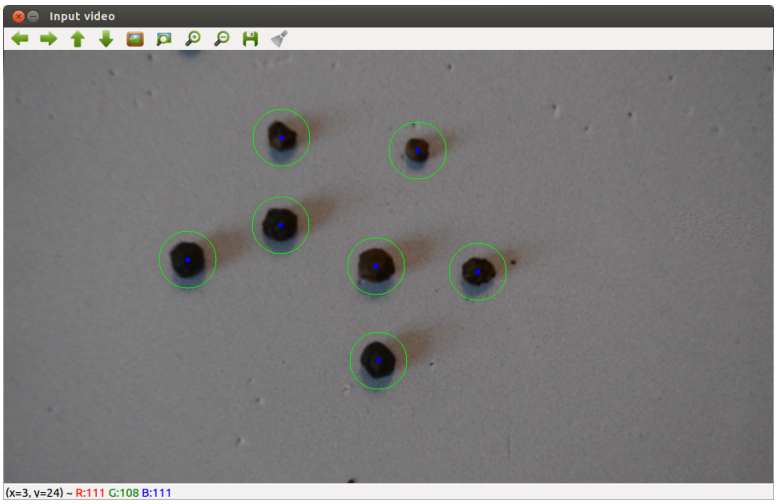


Figure 5.3: Proof of concept - Detecting peppers on a flat surface.

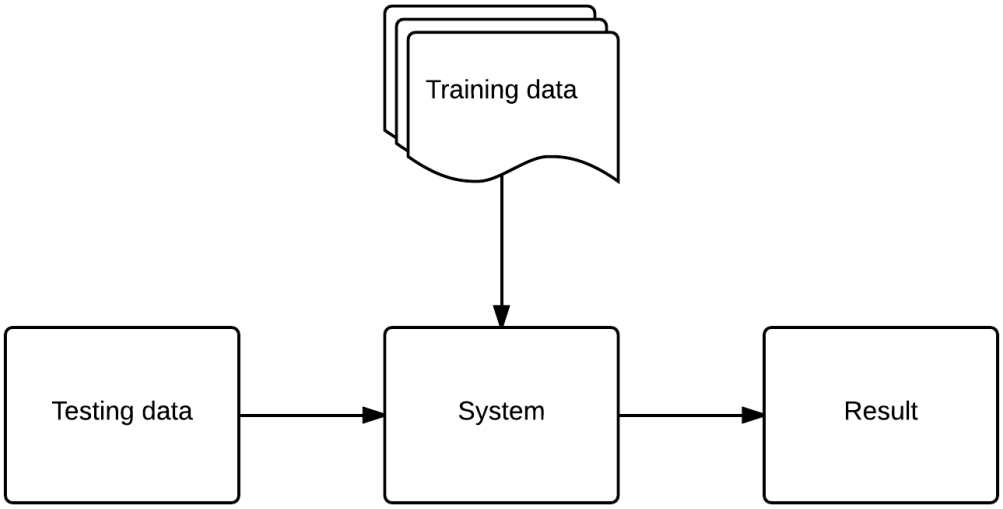


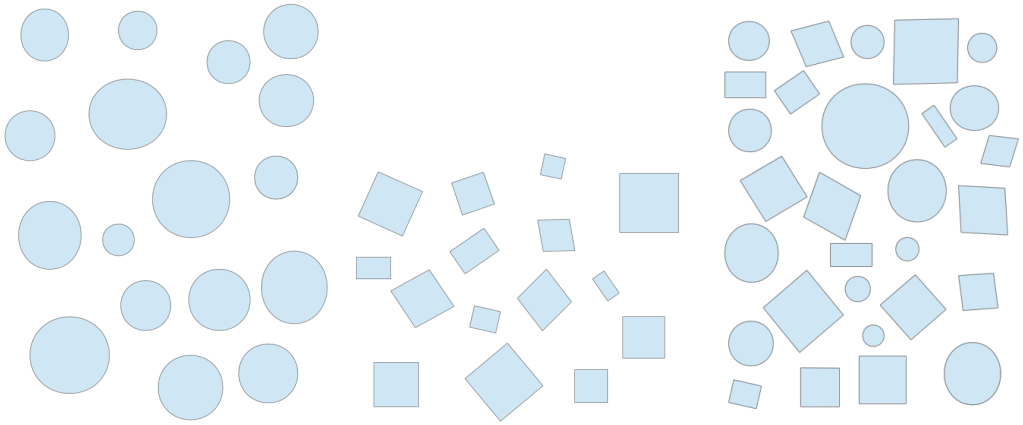
Figure 5.4: Supervised learning concept with training data and testing data

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 5.5.

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 [28] 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.



(a) (b) (c)

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.

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

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

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 figure 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 shows that circles and rectangles can be linear separated by using the 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.

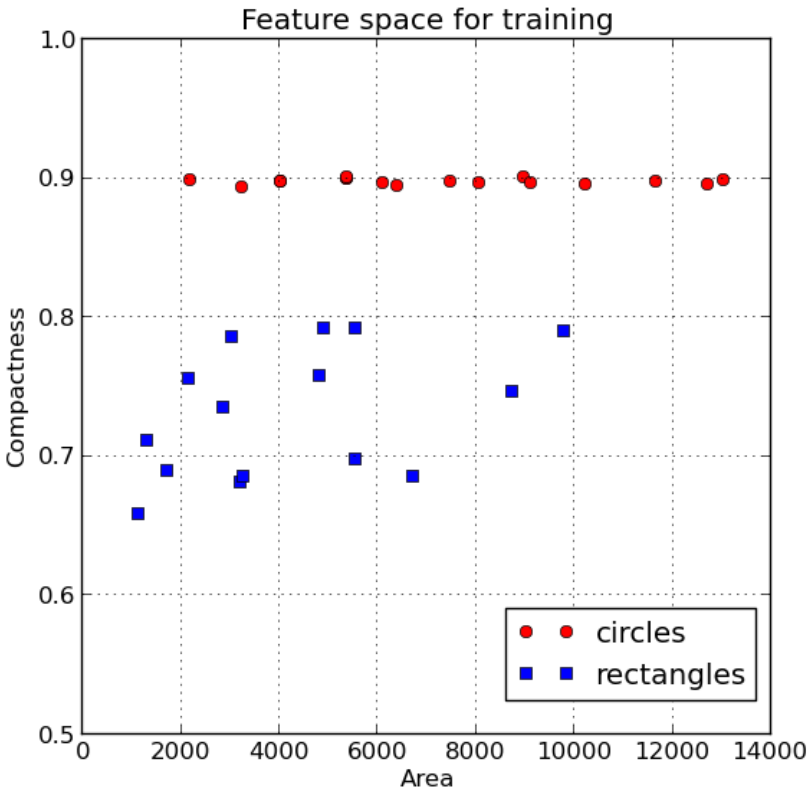


Figure 5.6: Feature space of compactness and area of each object in the training data

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 [12]. 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* [12]. 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 [27], which classify each output to either 1 or -1. With the update weighs and bias the linear classification line is defined [12] 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

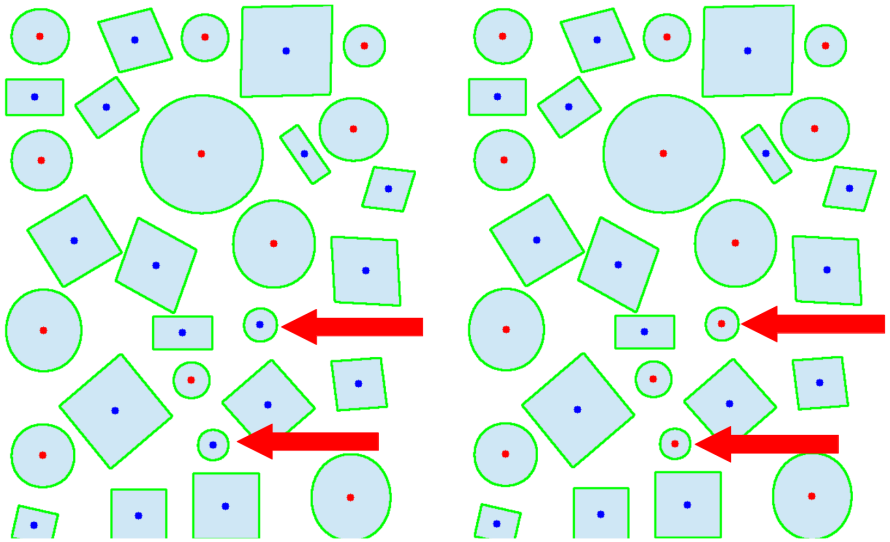
Input: Pre classified training images with circles and rectangles separately
Output: Updated weights and bias to be used for making the classification line
Initialization: Set weights and bias to zero: $w_0 = w_1 = b = 0$

```
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).

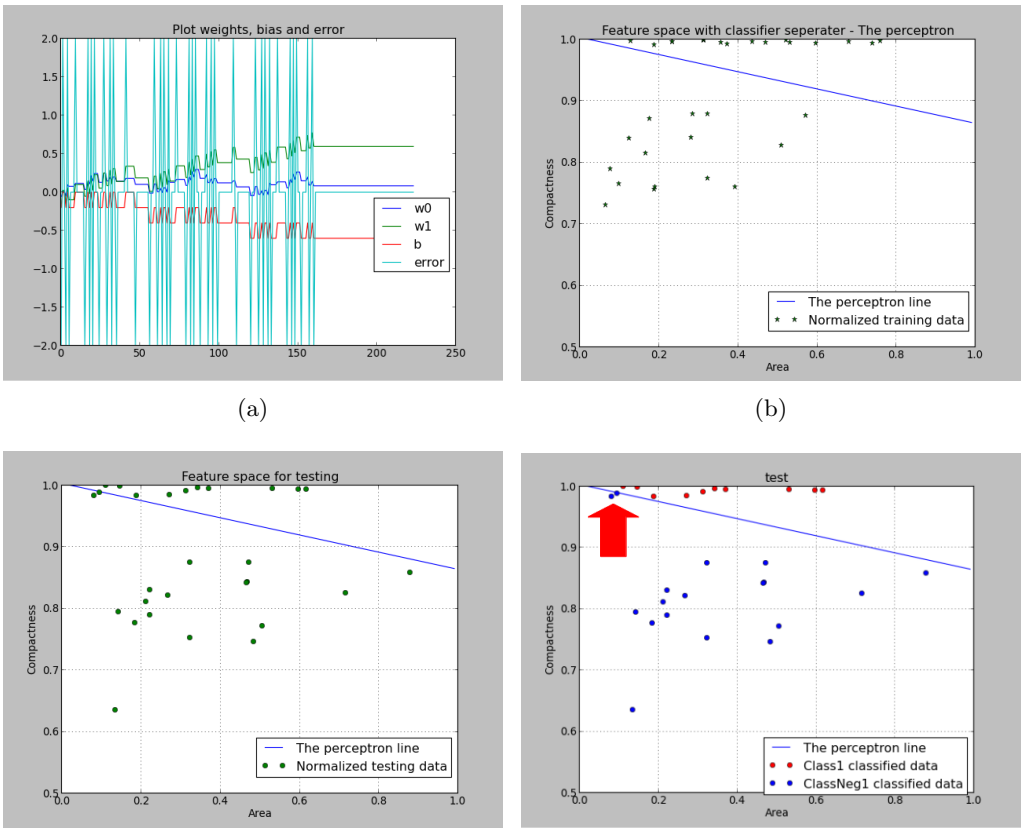


Figure 5.8: Figure a) show the convergence of the weights, bias and error. Figure b) show the separation line based on training data. Figure c) shows the testing data with the separation line. Figure d) shows the classification result, with two misclassified objects

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 too 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 [12] for such task and hence will be integrated in the final project using the a SVM library [30].

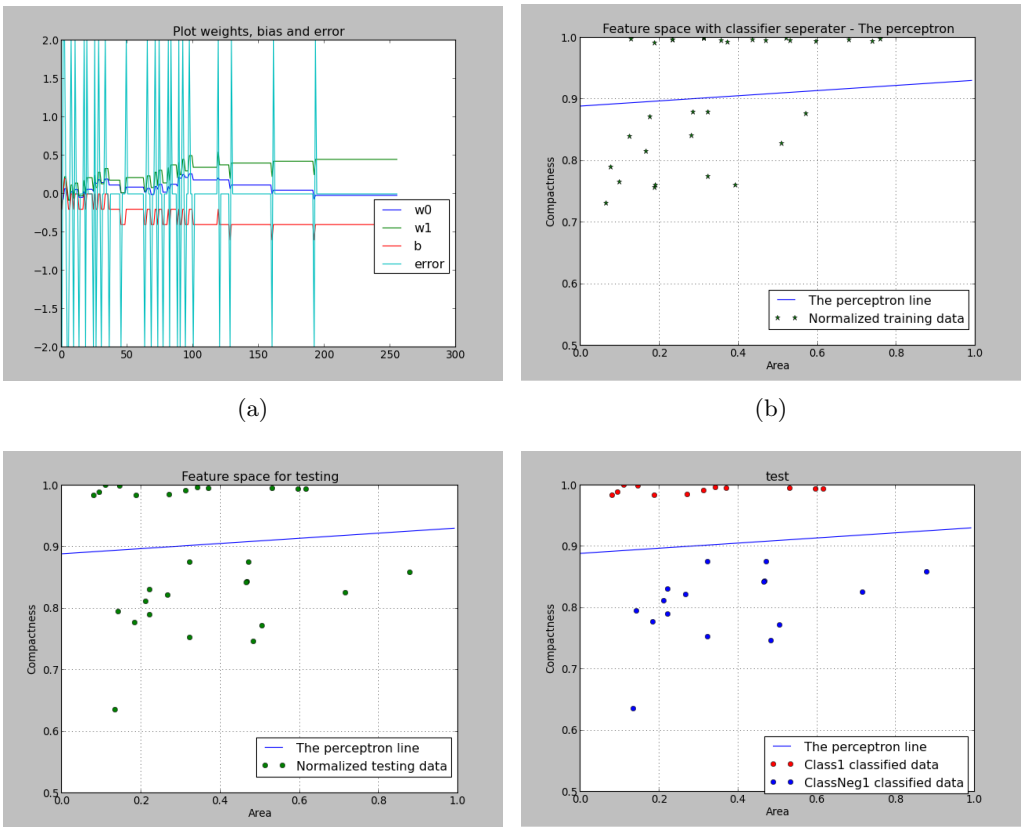


Figure 5.9: Figure a) show the convergence of the weights, bias and error. Figure b) show the separation line based on training data. Figure c) shows the testing data with the separation line. Figure d) shows the classification result, with no misclassified objects

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 completed. The next step was to optimize with real seeds instead of squares and circles. However in order to use real seeds, the choice of camera was considered.

5.3.1 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 [34]. 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 [19], classify different rice cultivars [23] and finding grape seed characters [31].

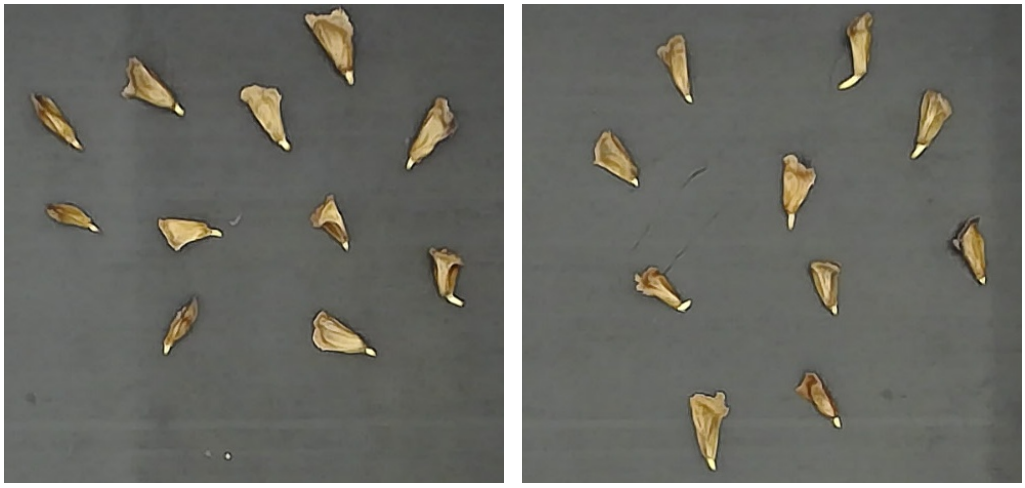
Many companies that sells hyperspectral camera [6], [4], [11] and [10] only gives the price of their camera through quotes. It has not been possible to find a price list of hyperspectral cameras. Therefore it is assumed that a hyper spectral camera is in the high-end regarding cost. Define a suitable 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 E 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 at no cost
- The price of the web camera is currently 1199 DKK [26].

5.3.2 Detecting real seeds

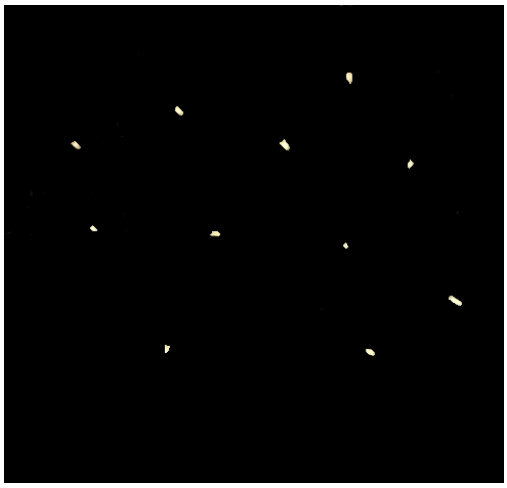
The vision system was ready to be tested with real seed using the Logitech C930e webcamera. A company visit to IPS was arranged in order to collect training data. Description of the setup on site is described in appendix A. The result created a need to trim different parameters of the camera manually focus, sharpness and exposure. Therefore a image script was implemented in order to capture the proper training data. This implementation is described in appendix B. After some test, the best parameters for the camera was found. This is described in appendix C.

However using the webcamera showed later in the project that sprouts would have a more whiter color in the image compared to the human perception. Images (a) and image (b) in figure 5.10 shows seeds with a natural white sprout color, but it is only the left image (a) where the sprouts have a white color in reality. The sprouts 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 critically. In



(a) Yellow-brown sprouts in reality

(b) White sprouts in reality



(c) Yellow-brown sprouts extracted from (a)

(d) White sprouts extracted from (b)

Figure 5.10: (a) Yellow-brown sprouts.(b) White sprouts. Image (c) and (d) shows the extracted sprouts for image (a) and (b) respectively.

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 5.10. 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 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 5.11. 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 5.11 and table 5.1, a great variance is shown

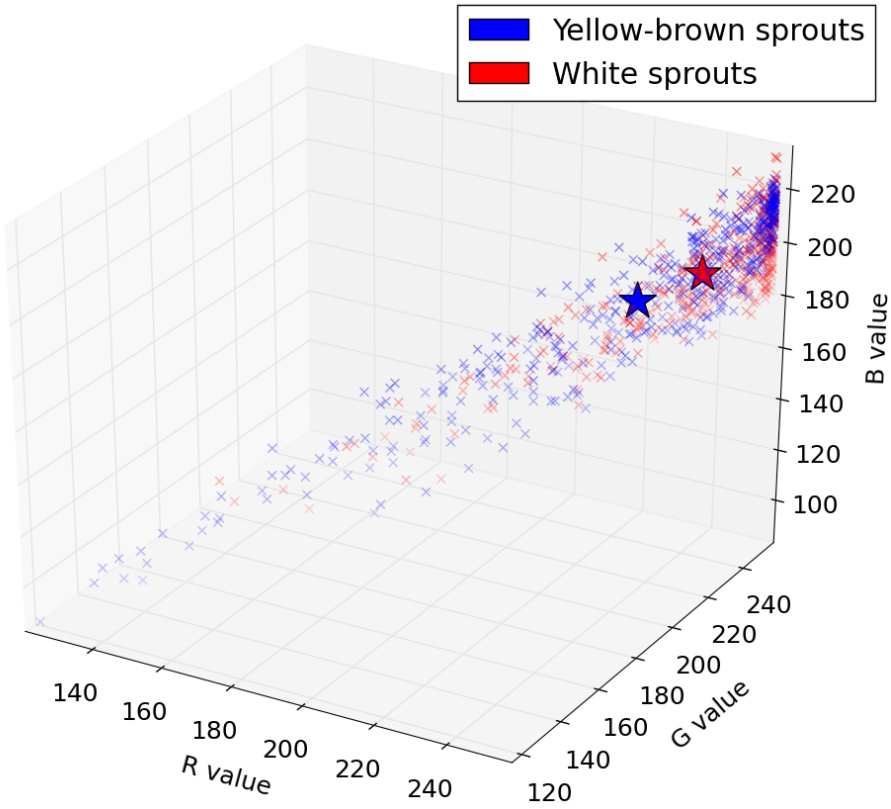
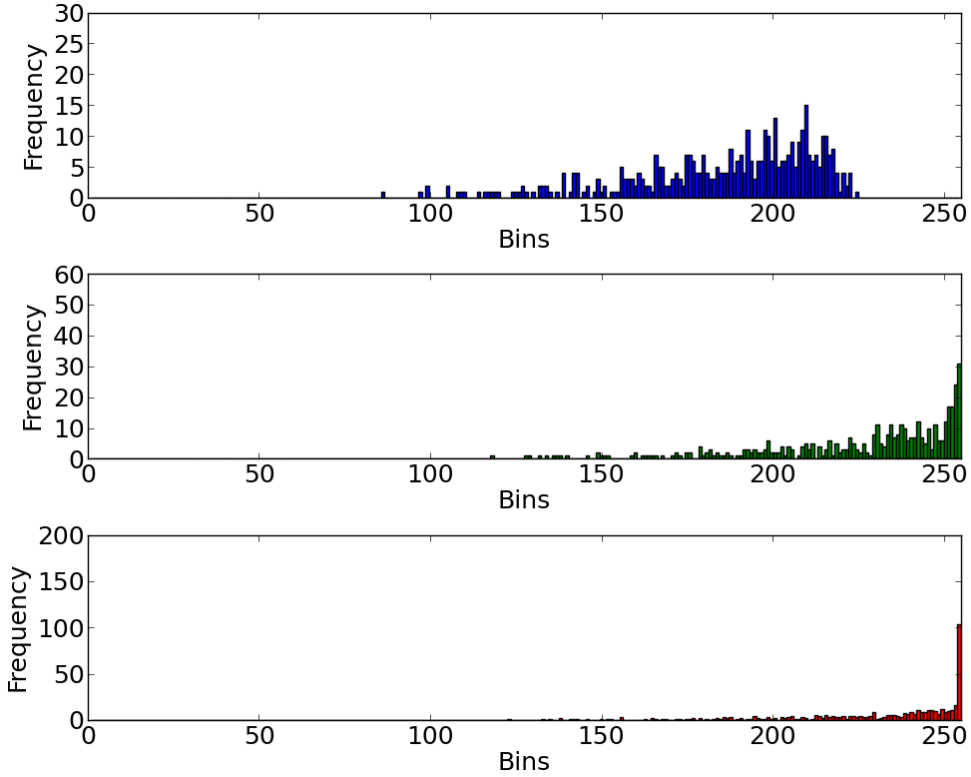
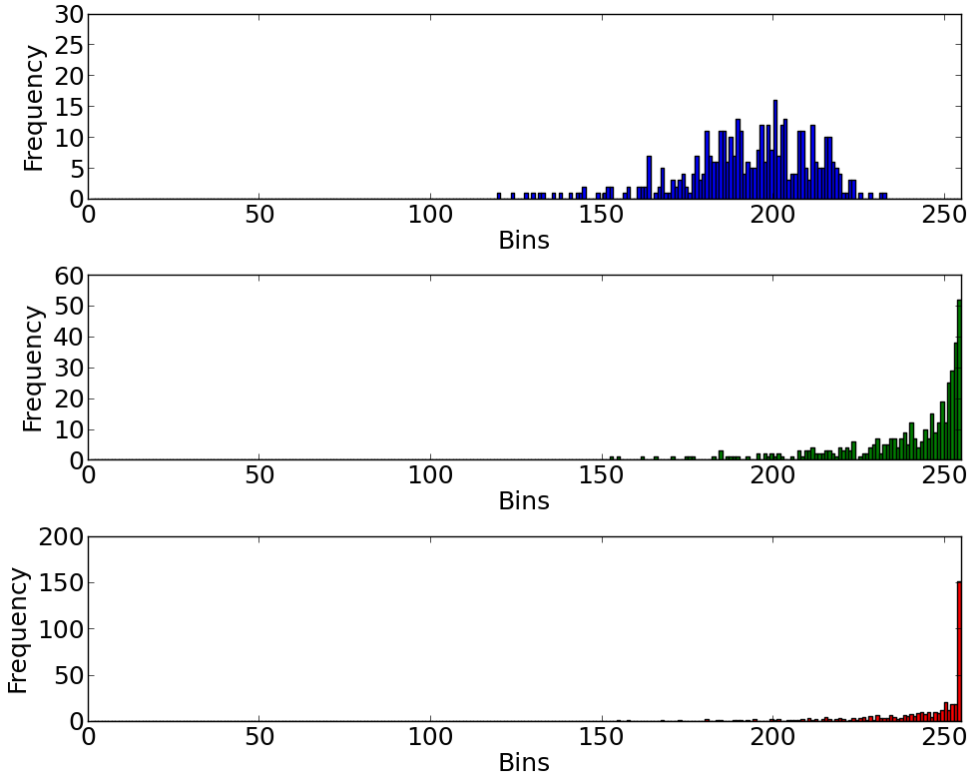


Figure 5.11: Plotting the RGB values for yellow-brown and white sprouts, togeter with their mean (Star

among the data. One reason for this can be the manual selection of pixel with GIMP. Another reason can be the light condition and a third reason can be camera parameter setting. With the current settings for the camera, there is no significant difference between white and yellow-brown sprouts. The statistical tool Analyse of Variance (ANOVA) could be used to verify this, but the assumption for ANOVA is that data is normal distributed and independent. The first assumption do not hold, as shown in the histogram in figure 5.12. Secondly the sprout data is depended, since the value of the sprouts are closely related to the neighbour pixels. Therefore there is no significant difference between the sprout pixels in figure 5.10 and with the current settings of the webcamera, it has been unsuccessful to producing RGB pixel that correspond to ground truth by using the web camera. However the RGB webcamera can still be used to detect the length and width of the sprouts. Therefore the arguments for continue with a webcamera, which is described in subsection 5.3.1 still remains.



(a) RGB histogram of the yellow-brown sprouts

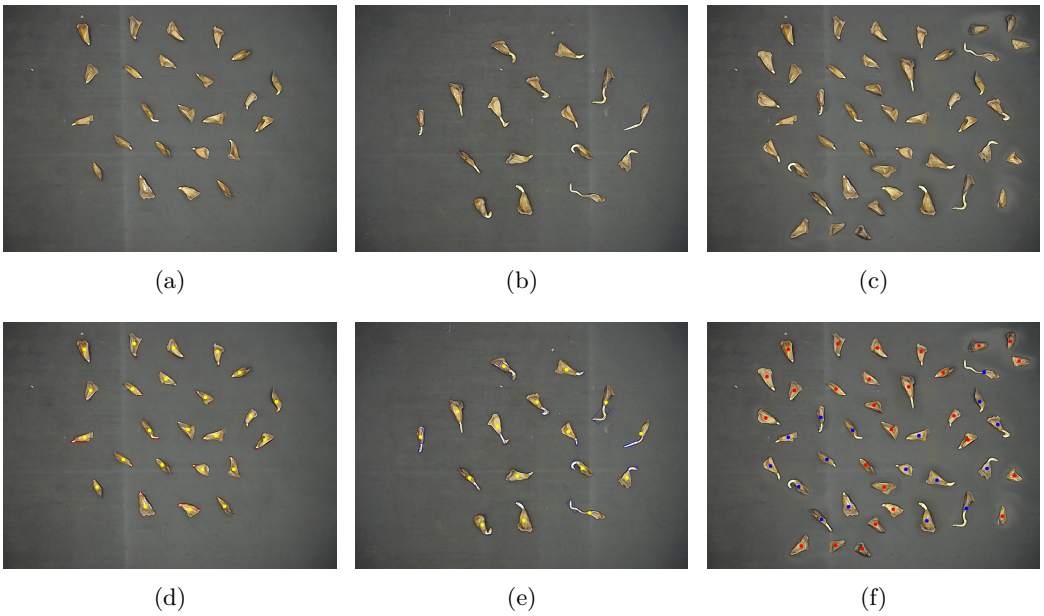


(b) RGB historam of the white sprouts

Figure 5.12: RGB histogram of (a) yellow-brown and (b) white sprouts.

5.3.3 2 class classification of real seed with Perceptron

The training data described in subsection 5.3.2 was labelled as class1, class-1 and class0. The class1 data contained sprouts that had length less than approximately 1 mm. The second training data class-1 contained sprouts longer than 1 mm. The testing data, named class0, was generated using the training data. These data is shown in image (a), (b) and (c) in figure 5.13.



(a) (b) (c)

(d) (e) (f)

Figure 5.13: (a) Training data class 1 with OK sprout length. (b) Training data class -1 with too long sprout length. (c) Testing data. (d) Features extracted from (a). (e) Features extracted from (b). (f) Classification result of (c) based on training data (a) and (b)

The feature extraction for the training data is plotted in image (a) in figure 5.14. Here it is shown, as predicted in section 5.2.2 that the data is not linear separable. By using the Perceptron a upper iteration limit was implemented in order to stop the algorithm, otherwise the Perceptron would run forever with non-linear separable data. The seperation classification line is shown in image (b). The featureplot of the testingdata is shown in image (c). The classification result in featureplot is shown in image (d).

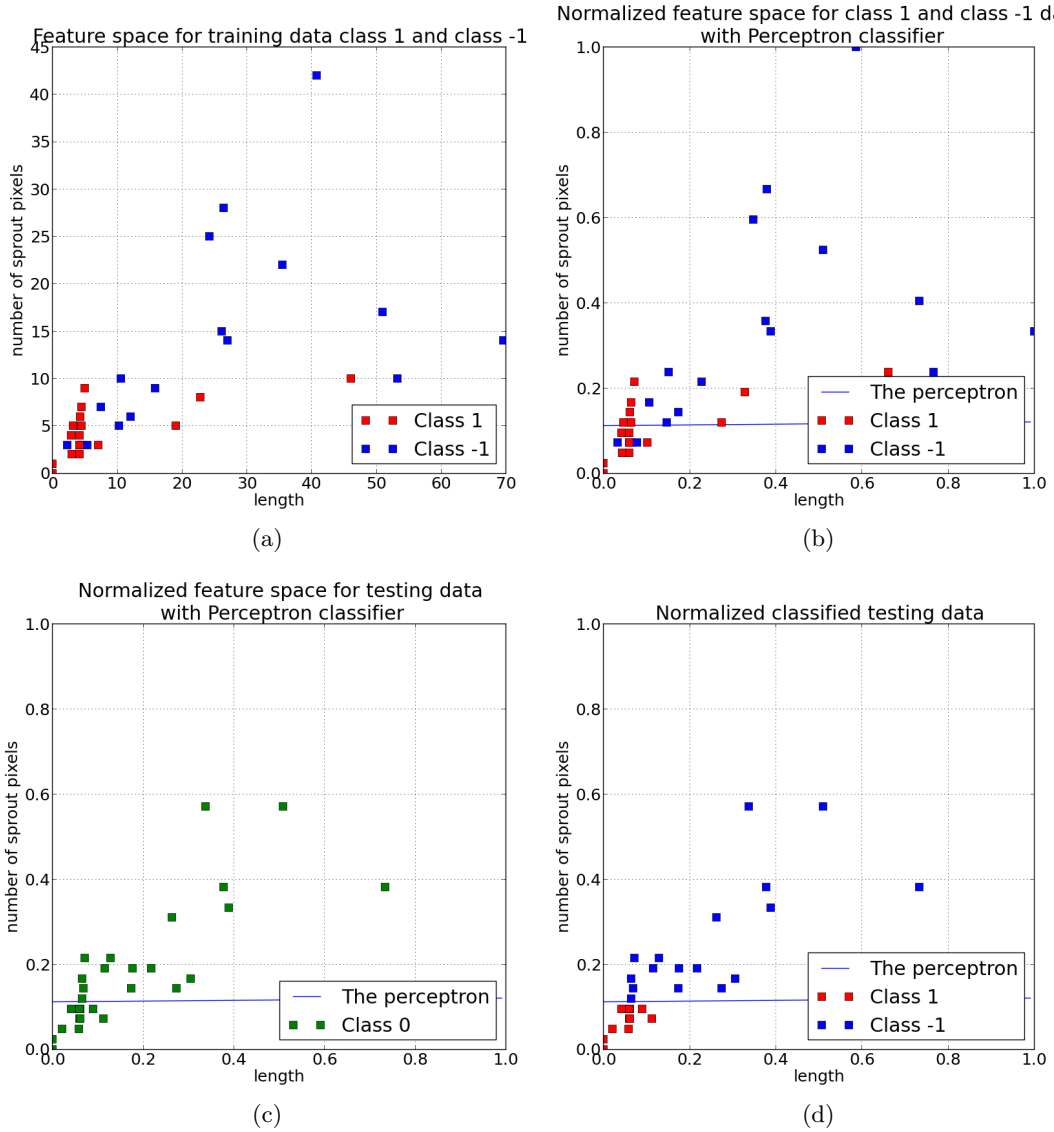
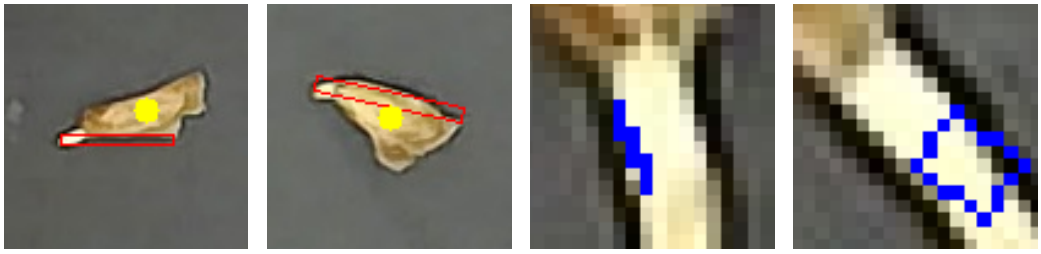


Figure 5.14: (a) Training data class 1 with OK sprout length. (b) Training data class -1 with too long sprout length. (c) Testing data. (d) Features extracted from (a) and (b)

As described in subsection 5.2.2, the classifier do not optimize the margin and the Perceptron will run until a solution is found, unless it is forced to break after counted iterations. This is not desired and time for more sophisticated classifiers is needed.
From literature [12] the Support Vector Machines (SVM) in contrast to the Perceptron, has parameters that can be used to give different solution. I.e. with a radial base function kernel (RBF) the cost parameter C and the γ parameter can be changed to give different classification region. These parameters are described further in section 6.4. Therefore it is was chosen to not spend more time on the Perceptron classifier.

5.3.4 A new setup

A interesting experience with real seeds, showed how the the segmentation and processing components needed optimization. Some seeds in figure 5.13 in image (d) and image (e) have false features, i.e. the bounding box, which the feature extraction is based on, was wrongly segmented. Zoom-in example of this is illustrated in figure 5.15. In image (a) and (b) the bounding box stretches around all white pixels. In image (c) and (d) the bounding box is too small and inaccurate. This is handled and described further in subsection 6.3.3.



(a) Class 1 (b) Class 1 (c) Class -1 (d) Class -1

Figure 5.15: Image (a) and image (b) shows boundingboxes which are too long. Image (c) and (d) shows boundingboxes that do not cover the real sprout pixels

To summarize, several things changed the project. First of all, the preprocessing and segmentation component needed to be adjust in order to better reproduce the sprouts from the images. Secondly the SVM should be integrated as the new classifier with three classes. At the current state, ideas of testing the vision system together with a robotic system was present. In order to have full time access to the robot, a new setup in Robolab at SDU was implemented. This is shown in figure 5.16.

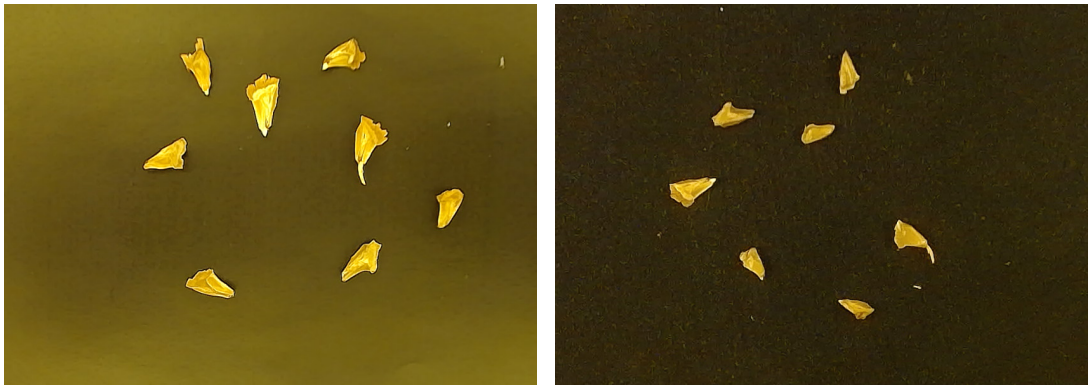
(a) (b)

Figure 5.16: (a)Images of new setup in RoboLab at SDU. (b) The camera Logitech C930e is mounted inside the wooden box with the optical axis perpendicular to the plane with seeds.

A full working setup would include the vision system, conveyor belt and a robot, where the vision system track seeds the conveyor belt. In order to calculate the position of the grasping location the speed of the conveyor belt must be known. Without any encoder this is not an easy task, since the speed estimation of the conveyor belt must be implemented as an extra vision component. From experience from previous course, it was a troublesome process to insure a robust speed estimation, bases on computer vision and having a buffer that only contains valid coordinates. Therefore it was chosen to simplify the setup. In order to have a more simple setup with a robot, the conveyor belt was removed. In RoboLab the UR-6 robot has been used previously to draw sketches on A4 papers. With the list of classification results and center of mass coordinate for each seed, the idea was to let the robot, in the robot work frame draw at each seed location, the classification result. I.e. the robot gets the location of a seed, moves to that location in the robotic workframe and either draw a checkmark or X depending on the classification result.

Draw this in google SketchUp?

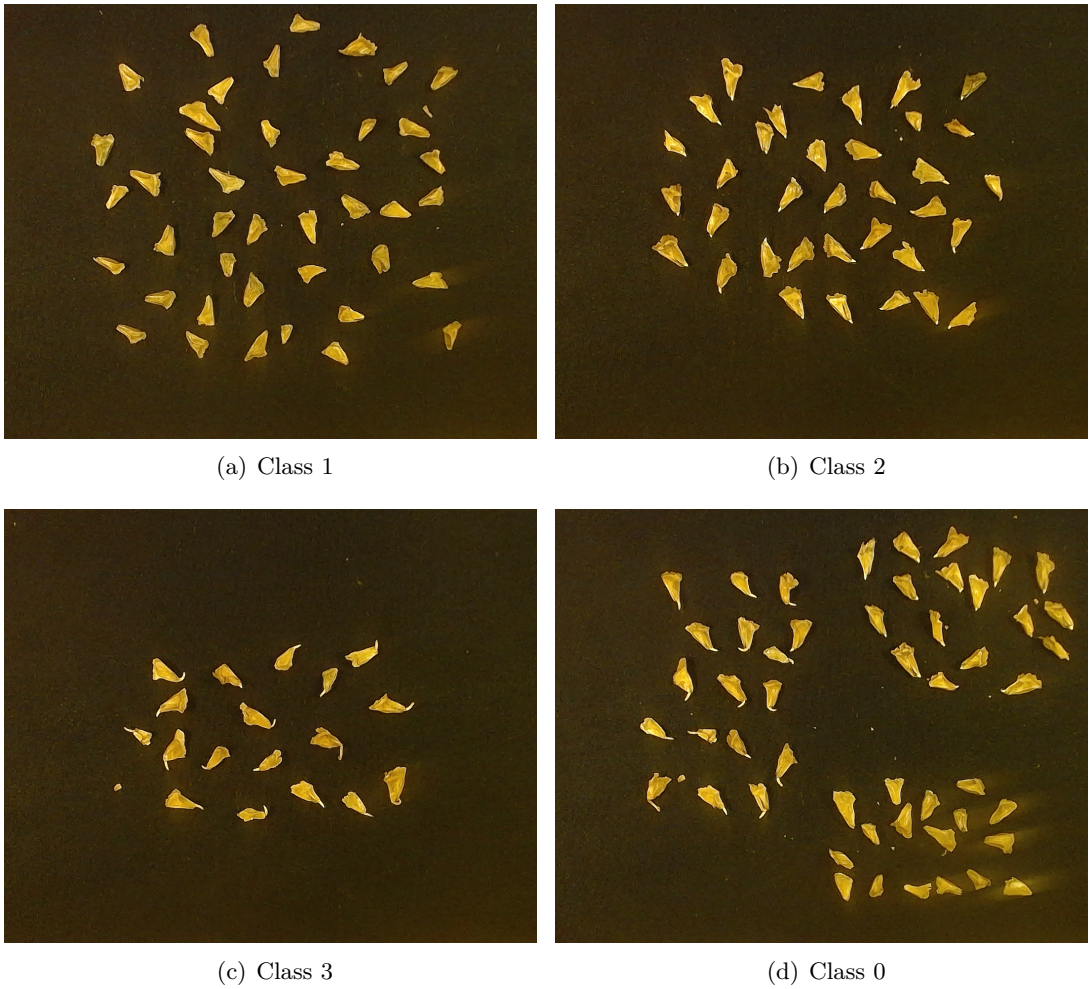
To start out with the new setup, the camera was mounted in a wooden box, which is shown in figure 5.16. Using the wooden box in RoboLab, showed that the light conditions plays an important part. The light bulb that was available were illuminating *warm light*, which has a temperature of 2700K. Additional shielding of the wooden box was attached and a new background with less reflectance was added.The result is shown in 5.17.



(a) (b)

Figure 5.17: (a) Initial test of new setup. (b) Changing the background with less reflectance

The temperature in Kelvin plays an important role. A test was carried out in order to see the difference between warm white (2700K) and a cool white light (4000K). This test is described in appendix D.
Since the environment was changed, new training data and testing data was created. It was decided to have three classes of training data in order to make a three decision regions. This is shown in figure 5.18.



(a) Class 1 (b) Class 2

(c) Class 3 (d) Class 0

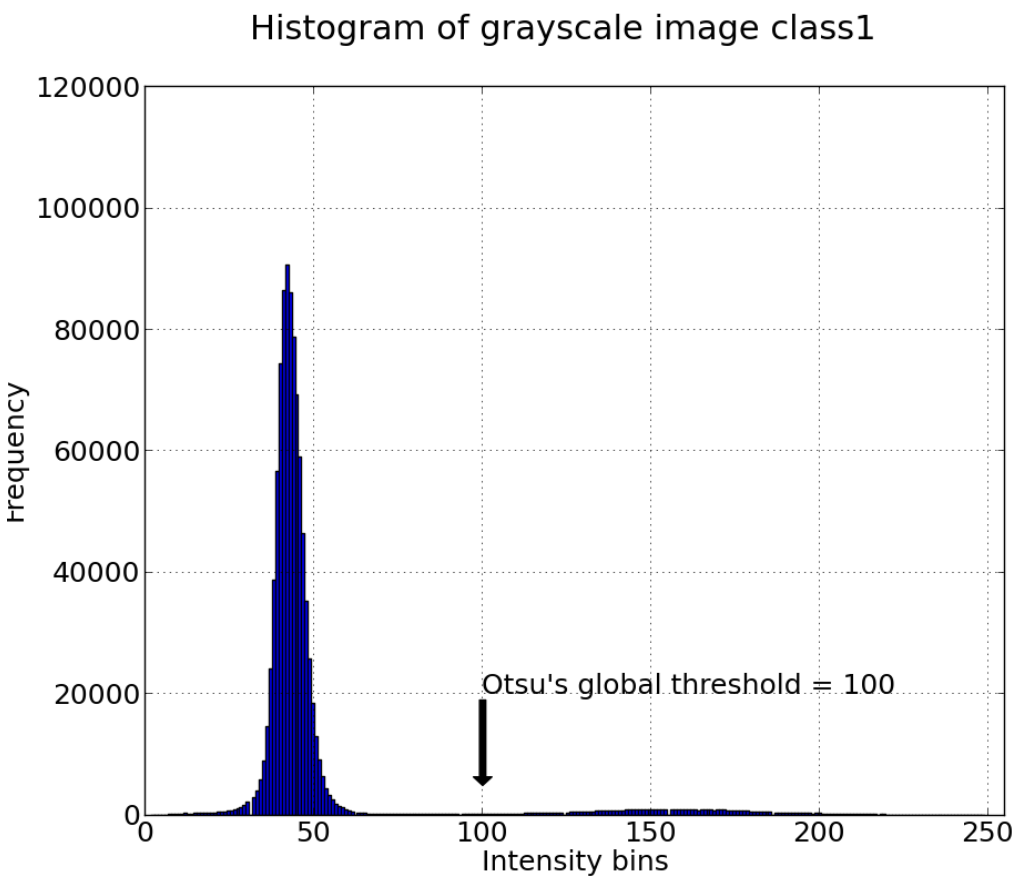
Figure 5.18: (a) Training data class 1 where seeds has no sprouts. (b) Training data class 2 where length of sprouts are OK. (c) Training data class 3 where length of sprouts is too long. (d) Class 0 shows seed with different sprout length.

At the moment of the project it was chosen to focus on one kind of feature i.e. the length of the sprouts, due to the limitations with the color perception, which is described in subsection 5.3.2. The three criteria is as followed:

- If there is no sprout, the seed is bad
- If the length of the sprout is between 1-3 mm it is a good seed
- If the length of the sprout is longer than 3 mm, it is a bad seed

5.3.5 Optimizing the preprocesssing and segmentation component

The first step was to determine if the threshold value could be optimized. Previously the threshold value was picked to 128, to separate between background and front ground pixels. In order to repair the structures, which is described in section 6.2 three iterations of morphology was used. This have the downside of possible bridging seed that are close to each other. Using the OTSU optimal threshold value [2], the seeds was less damaged and fewer morphology iterations was need. The result is shown in figure 5.19, where (a) shows the histogram and locate the OTSU threshold. Image (b) shows the result of class 1 before using the OTSU threshold value together with three morphology iterations. Image (c) shows the result for class 1 after using the OTSU threshold together with only one morphology iteration.



(a)

(b) (c)

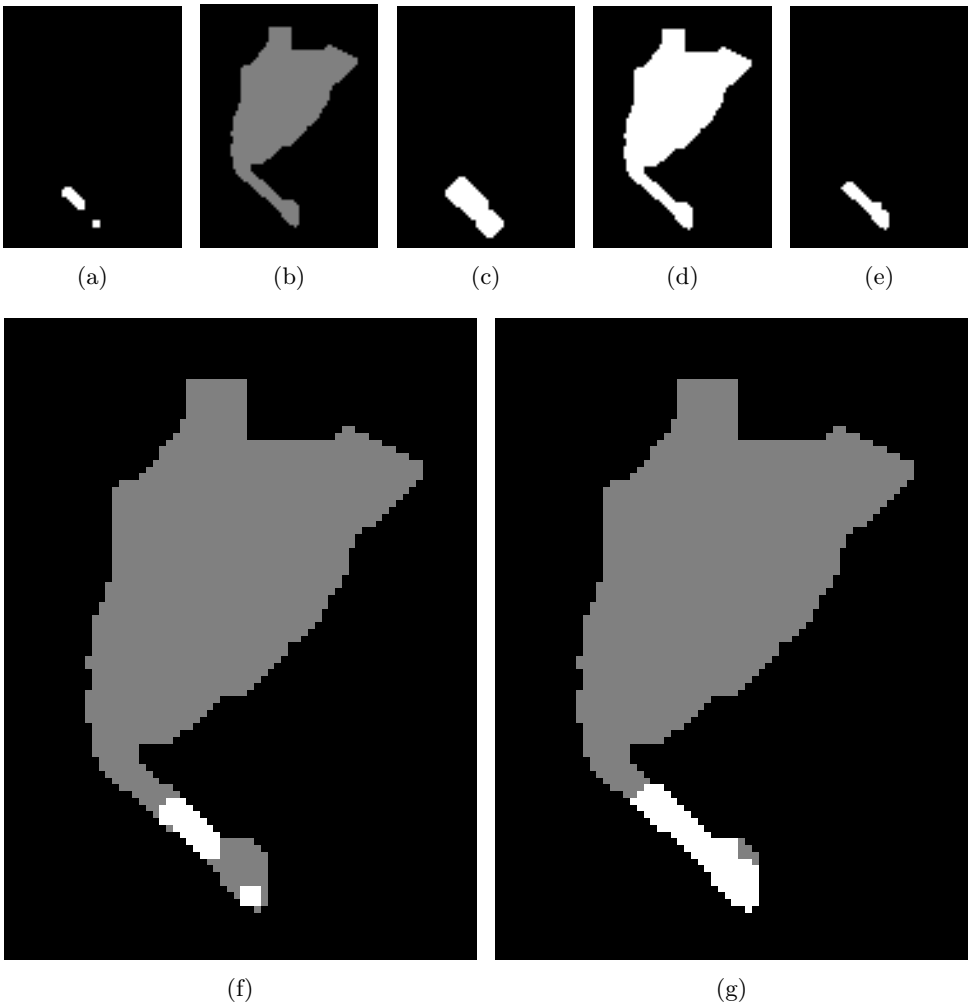
Figure 5.19: (a) Training data class 1 where seeds has no sprouts. (b) Training data class 2 where length of sprouts are OK. (c) Training data class 3 where length of sprouts is too long. (d) Class 0 shows seed with different sprout length.

The next step was to repair the sprouts with morphology using the AND operation. The process is described as followed:

- Dilate the sprout image, $I_{dilateSprouts} = \text{dilate}(I_{sprouts})$

- Add the front ground with it self to get a temporary front ground image of white pixels, $I_{whiteFrontground} = I_{grayFrontground} + I_{grayfrontground}$.
- Do at bitwise AND operation between the white front ground image and the dilated image, $I_{sproutRepaired} = I_{dilateSprouts}$ & $I_{whitefrontground}$
- Add the repaired sprout image to the original front ground image to get the new seed and sprout image. $I_{newSeedAndSprout} = I_{sproutRepaired}$ & $I_{grayFrontground}$

Example of the described approach is shown in figures 5.20.



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

(f) (g)

Figure 5.20: (a) Sprout before. (b) Front ground image. (c) Sprout dilated. (d) Added (b) twice. (e) Bitwise AND operation between (c) and (d). (f) Result before, with (a) and (b) added. Result after, with (e) and (b) added.

The final result is shown in figure 5.21. Image (a) shows how the preprocessing output, before the implementation, gave bad feature extraction and hence the features clustered more or less randomly together. Image (b) shows the result of the implementation in how the features is more spread out. Image (c) shows the features after a normalization which enhance the result.

(a) (b) (c)

Figure 5.21: (a) Featureplot before optimization. (b) Featureplot after optimization. (c) Featureplot after optimization and normalization.

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 F.

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.

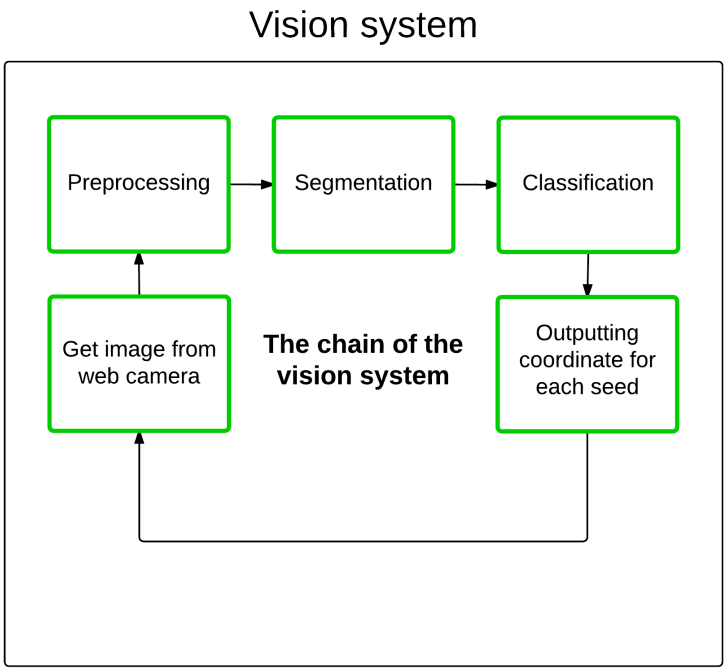


Figure 6.1: The chain of the project for the vision system

6.1 Get image from web camera

Argument for choosing the web camera, Logitech C930e is described in 5.3.1. 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 E. The webcamera has different parameters. These is further described in appendix B.

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.

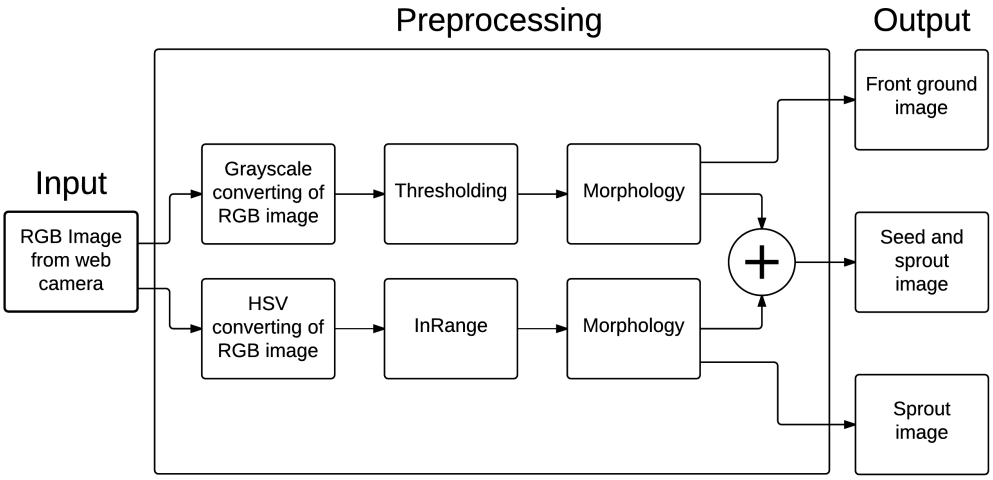


Figure 6.2: Preprocessing flow in order to generate the front ground, sprout and the combined seed and sprout image.

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 described further in section 6.3.3.

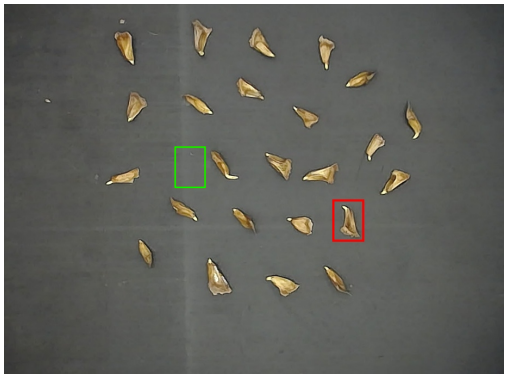


Figure 6.3: Input RGB image

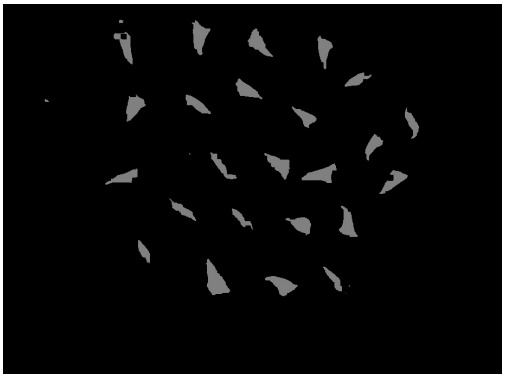


Figure 6.4: Front ground image

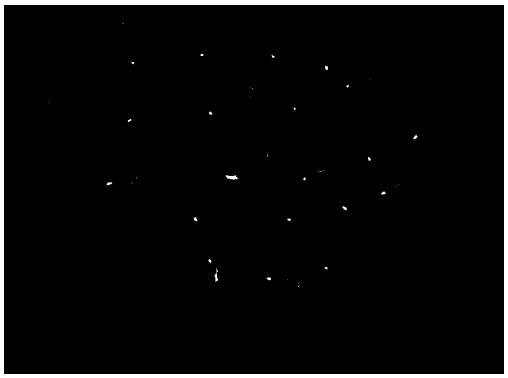


Figure 6.5: Sprout image

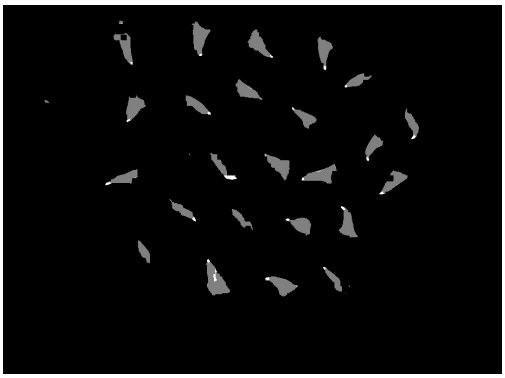


Figure 6.6: Seed and sprout image

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 [32], 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 [20]. Therefore the HSV method is the chosen approach in the preprocessing part.

Figure 6.9: Cropped image

Figure 6.10: Cropped background image

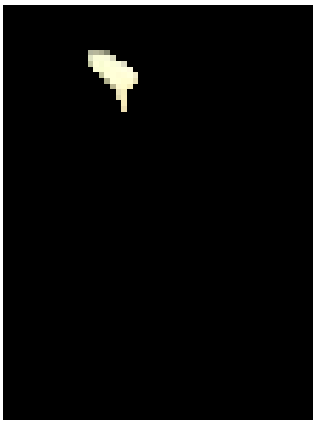


Figure 6.11: Sprout pixels only

Figure 6.12: Seed pixels only

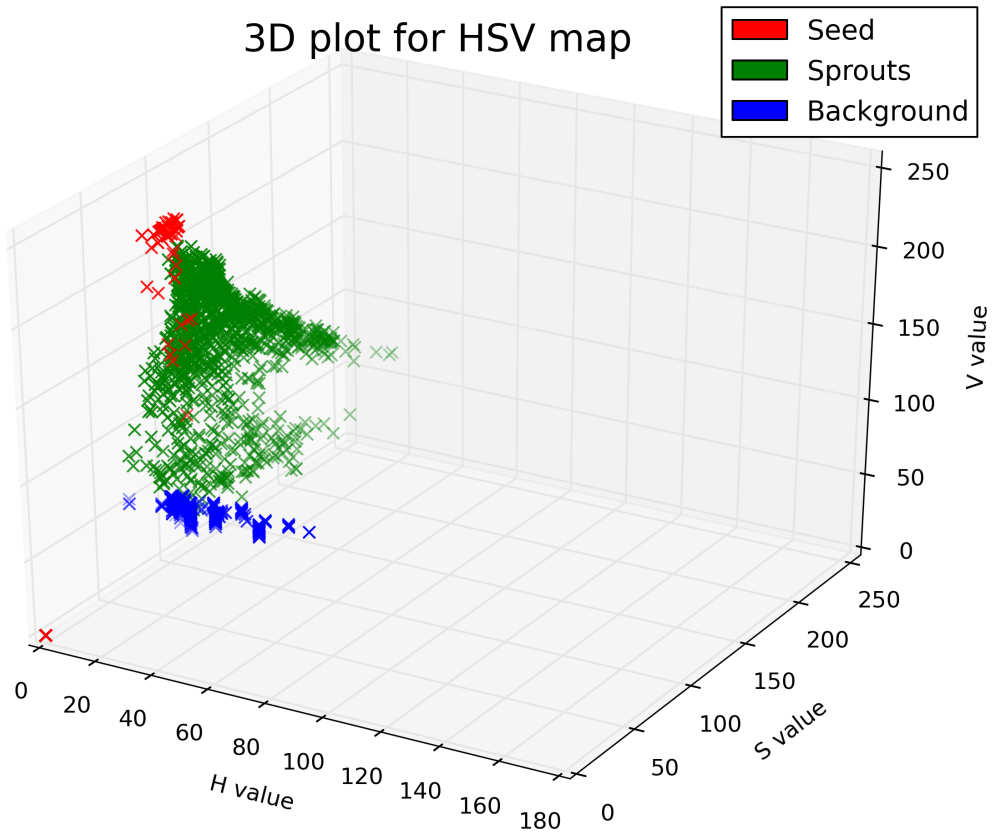


Figure 6.13: Preprocessing flow in order to generate the frontground , sprout and the combined seed and sprout image

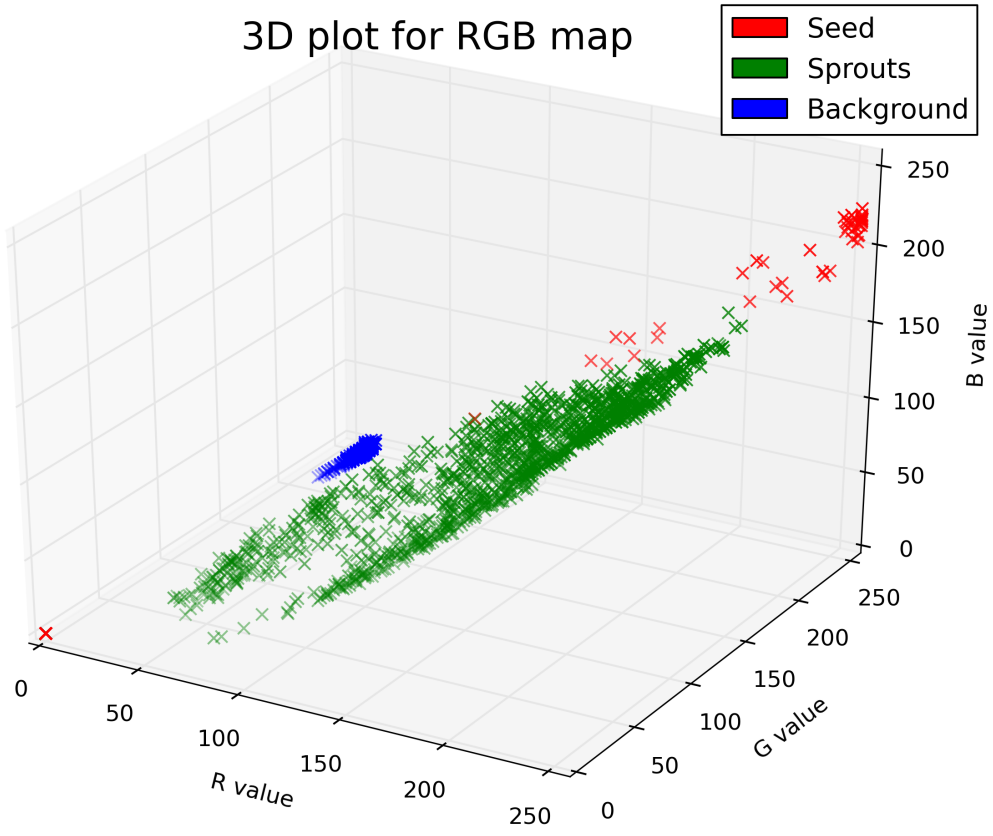


Figure 6.14: Preprocessing flow in order to generate the frontground , sprout and the combined seed and sprout image

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.

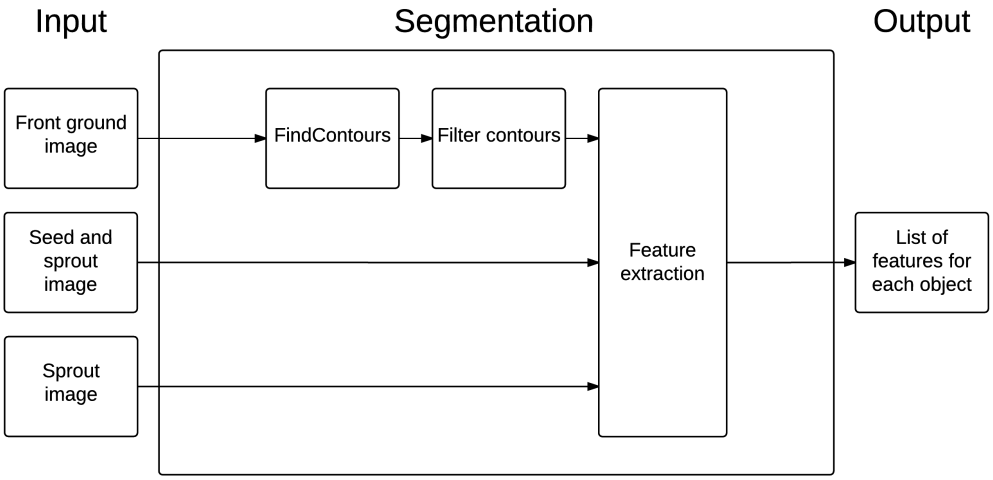


Figure 6.15: Segmentation flow in order to generate the list of features for each object in the RGB image.

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 6.16. 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.

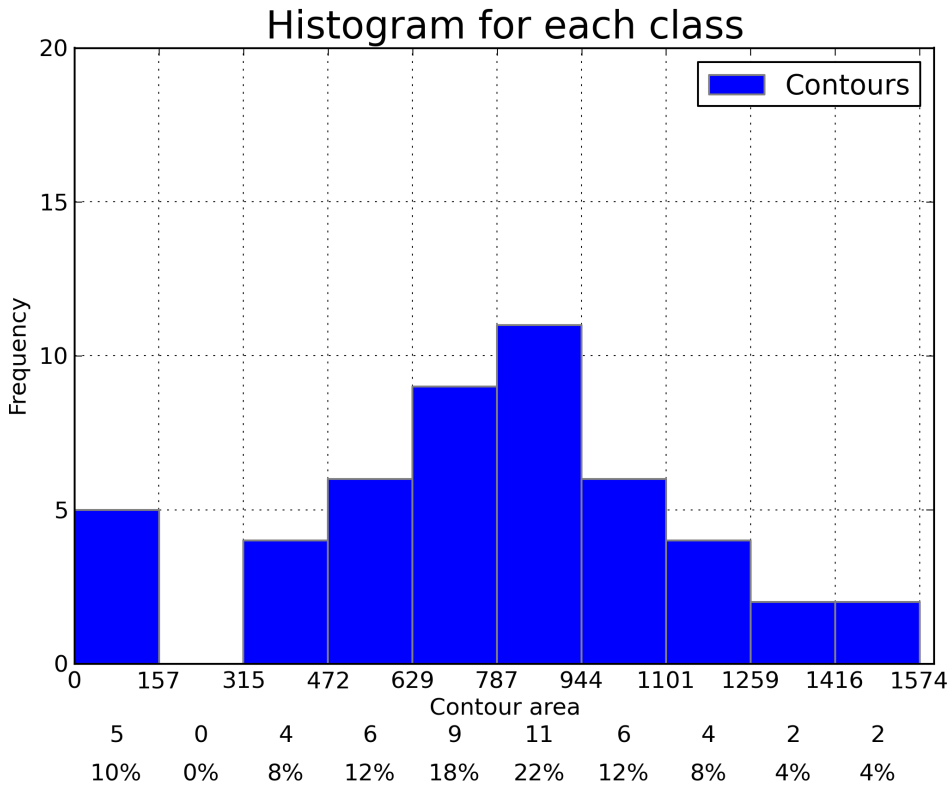


Figure 6.17: Histogram of contour area in the test image with 44 seeds and 5 noise blobs

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.

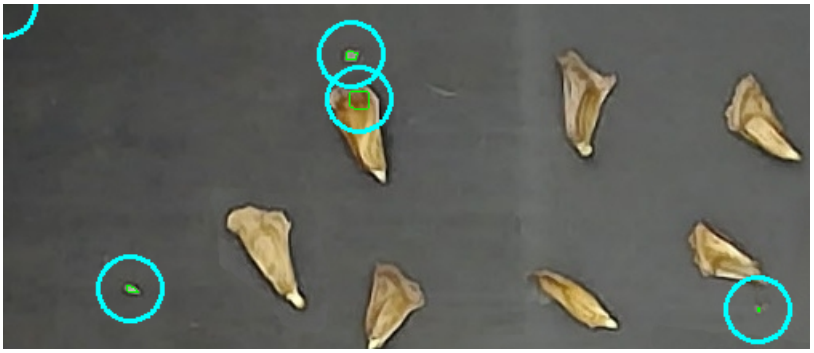


Figure 6.18: Indication of contours with area below 200 square pixels

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(a)
- ROI of front ground image after the morphology process, figure 6.19(d)
- ROI of drawn contours before filtering, figure 6.19(c)
- ROI of drawn contours after filtering, **??**

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 oriented boundingbox is fitted in order to get the length and width of this one. Together with minRectArea from OpenCV, the center of mass coordinate can be extracted. Then the ratio width/length is calculated, and together with hue values a list of features is generated out from each image. 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:
  - No sprout exist within the object, figure 6.20(a).
  - The length of the sprout is longer than 3 mm, figure 6.20(b).

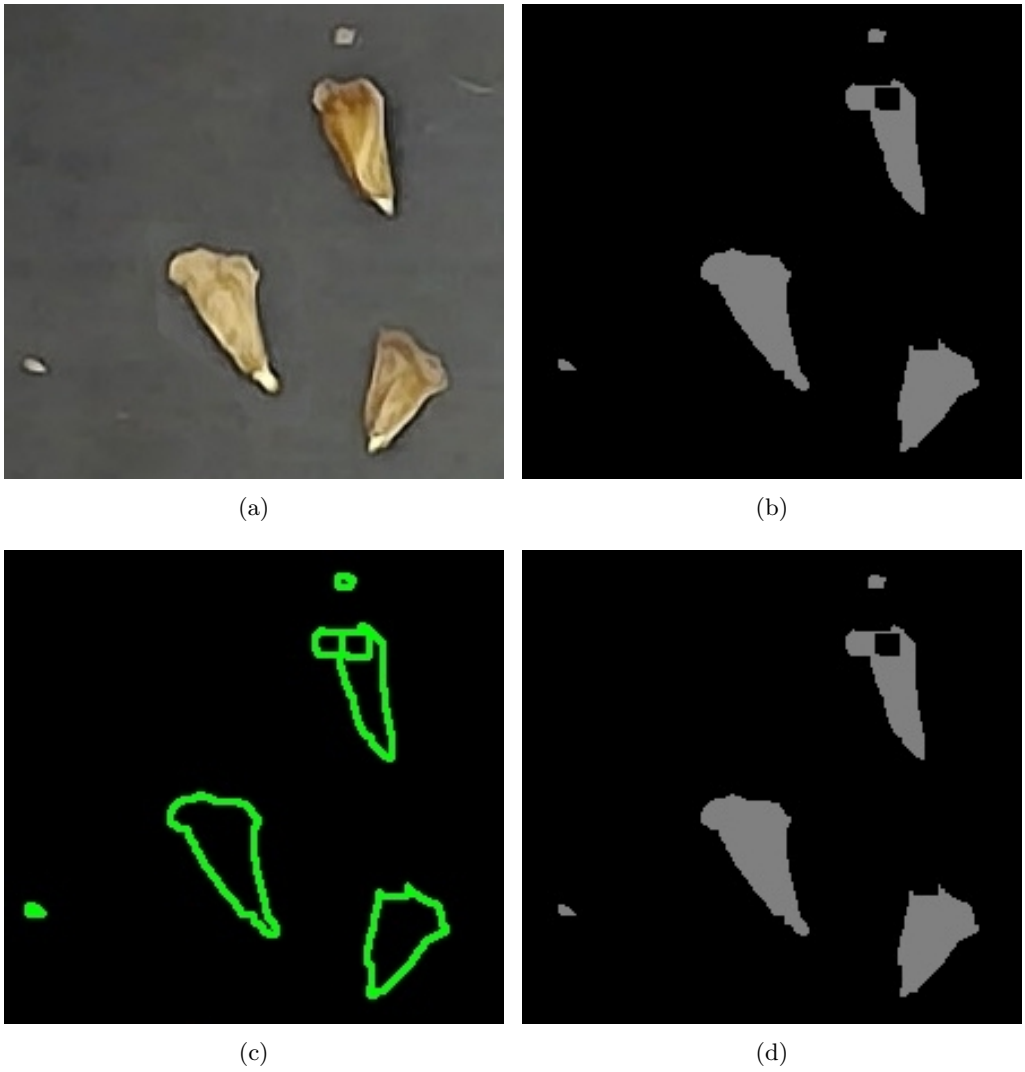


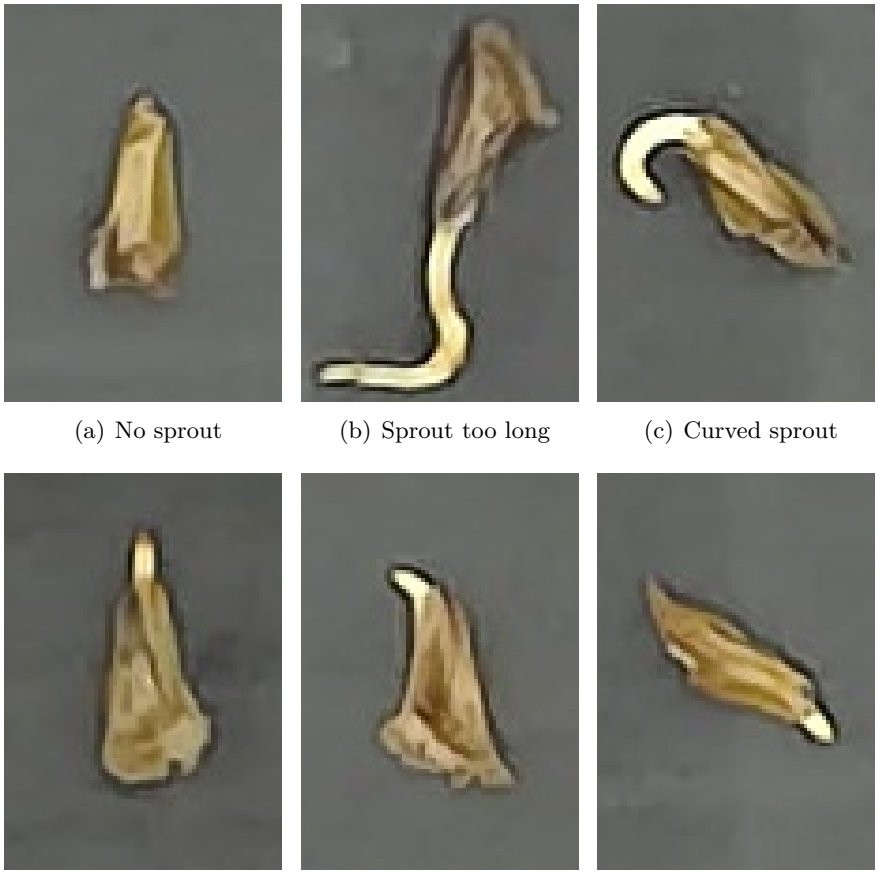
Figure 6.19: Image (a) Roi of input image. Image (b) ROI of thresholded image. Image (c) Founded contours with minimum contour area is zero. Image (d) Founded contours with minimum contour area is 200.

  - The sprout is curved or twisted, figure 6.20(c).
  - The color of the sprout is yellow or brownish, figure 6.20(d).

- An object is categorized as good if:
  - The length of the sprout is between 1 to 3 mm, figure 6.20(e)
  - The color of the sprout is white, figure 6.20(f)

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
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 6.21(a), a ROI has been cropped out for better visualization. In figure 6.21(b) the OBB around the sprout pixels is drawn. The red pixels is not a part of the image data.
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

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



(a) No sprout (b) Sprout too long (c) Curved sprout

(d) Brownish sprout (e) Sprout OK (f) Sprout OK

Figure 6.20: Showing different conditions of the sprouts

(a) (b)

Figure 6.21: Image (a) Cropped out object from RGB input image. Image (b) A red OBB drawn around the sprout pixels

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 6.22. 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

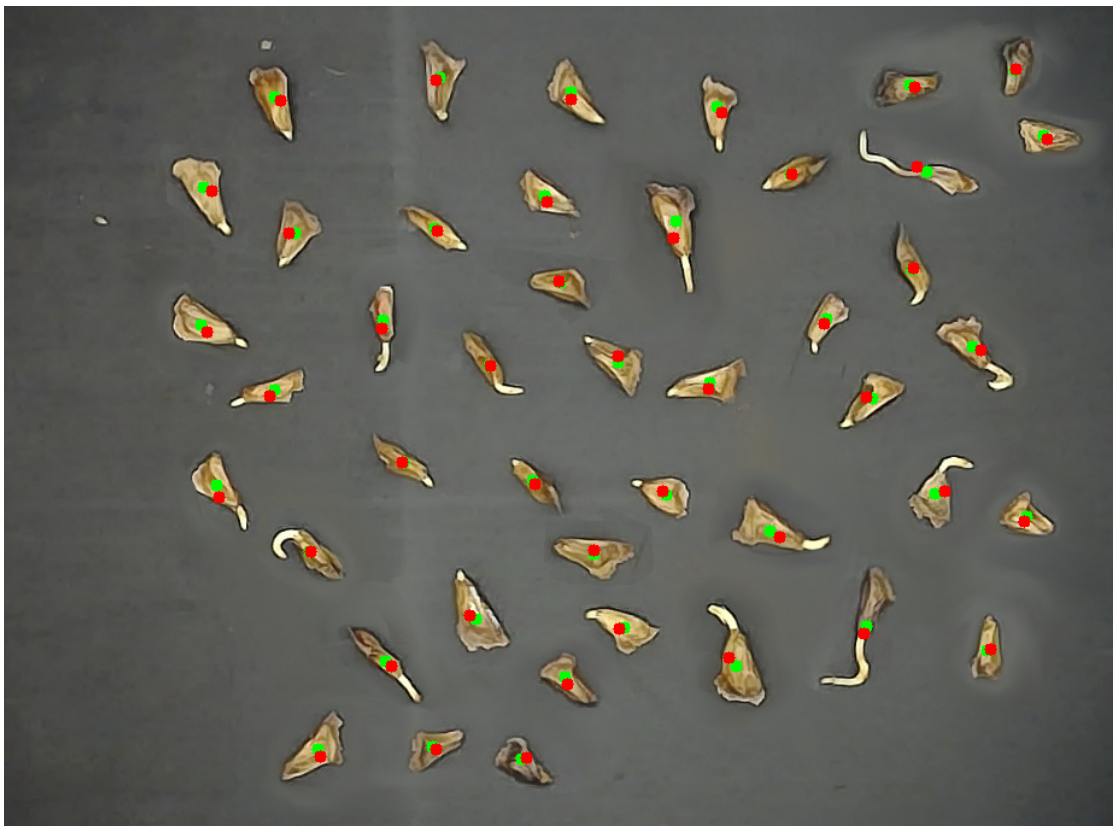


Figure 6.22: Red dots indicate COM using *minRectArea*. Green dots indicate COM using moments

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

6.3.3 Clustering

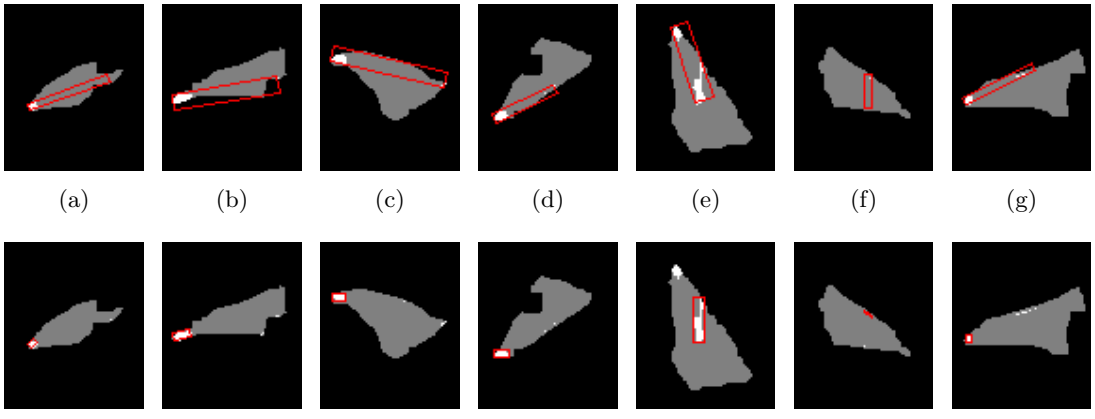
As described in section 6.3.2 the feature extraction is based on the OBB that span the sprout pixels for each contour. The data from the feature extraction is later used in the classification module in section 6.4. In order to maximize the classification rate, it is important to minimize the false negatives (Type I error). It is hypothesized that each seed in a image shares a good sprout. A type I error will occur if an object is classified as bad, but confirmed by a supervisor as good. In order to measure the type I errors, a tests is carried out. The test contains an RGB image with good labelled seeds. These

are shown image (a) in figure 6.23. The dimensions of the image is 920 x 680 pixels. Doing the test image (a) was preprocessed. The preprocessing component is described in section 6.2. The result of the preprocessing component of image (a) is shown in the same figure as image (b). Red bounding boxes is drawn to indicate features.

(a) (b)

Figure 6.23: (a) RGB image with good seeds. (b) Result of image (a) that went through the preprocessing and segmentation component.

Doing analyse of the sprouts for each object, the bounding boxes is extracted and drawn with a red rectangle. 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. There are 24 seeds with 7 false negatives. Type I error rate is at $\frac{7}{24} = 29.16\%$. The seven examples has been cropped out with a dimension of 74 x 89 pixels and is shown from image (a) to (g) in figure 6.24. In order to deal with the false negatives bounding boxes a the K-means cluster algorithm was implemented. The result of the K-means cluster algorithm is two lists. One list with "true" and one list with "false" sprout pixels. It is assumed that the majority of white pixels is wrong the "true" sprout pixels and the "false" sprout pixels is the minority of white pixels. Hence chosen "true" cluster is the list with most pixels.



(a) (b) (c) (d) (e) (f) (g)

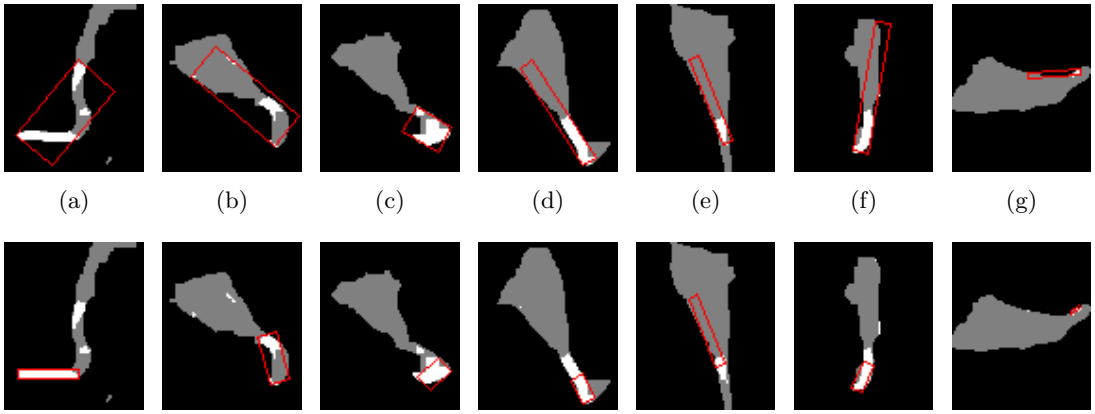
(h) (i) (j) (k) (l) (m) (n)

Figure 6.24: (a) - (g) shows boundingboxes before clustering and (h) - (n) shows after clustering with the use of the K-means clustering algorithm for training data class 1. K was equal 2

Overall implementing the K-means cluster algorithm helped to reduce false negatives, which is shown in image (h) to (n) in figure **??**. The type I error has been reduced to down to $\frac{2}{24} = 8.33\%$. However the assumption of having two lists after using the K-means cluster algorithm and use the list with most pixels is not always the truth.

(a) (b)

Figure 6.25: (a) RGB image with bad seeds. (b) Result of image (a) that went through the preprocessing and segmentation component.



(a) (b) (c) (d) (e) (f) (g)

(h) (i) (j) (k) (l) (m) (n)

Figure 6.26: (a) - (g) shows boundingboxes before clustering and (h) - (n) shows after clustering with the use of the K-means clustering algorithm for training data class -1. K was equal 2

Image 6.24(l), which is still a false negatives, shows the result of assuming that the biggest cluster contains true sprout pixels. Image 6.24(m) is also false negatives, but the amount of white sprout pixels where limited and hence the result which is shown in figure 6.24(f). Additional test was performed in order to evaluate the effectiveness of the K-means cluster algorithm. Training data where sprout were too long was used. The result is shown in figure 6.26. This shows that the cluster algorithm depends on the result of the preprocessing result. Image (a) and (c) has reduced their bounding box, due to "holes" in the sprout. Image (d) and (e) gives wrong bonding boxes. This concludes that the K-means algorithm is not the best solution in order to fix the preprocessing problems.
Another scenario is when two sprouts are close to each other. This is illustrated in figure **??** 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 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.

Pseudo code of the K-means cluster algorithm

The pseudo code of the K-means cluster algorithm is shown in algorithm 2. The K-means cluster algorithm is only executed if the number of contours for each ROI is >= 2. The

pro and cons of implementing the K-means cluster algorithm is described in the overall discussion in section.

Write that in discussion, that this will not be good, if two seeds are really close to each other. Since then two sprout clusters are present, in the image

Input: List of all white "sprout" pixels
Output: Two clustered lists: One with "true" sprout pixels and one with "false" sprout pixels.
Initialization: Randomly pick two different samples from the input list and use these as center for each cluster, $c1_{current}$ and $c2_{current}$

---

while *until algorithm breaks* **do**
  for *each pixel in input list* **do**
    d1 = Euclidean distance from pixel to center of cluster 1
    d2 = Euclidean distance from pixel to center of cluster 2
    if *d1* $<$ *d2* **then**
      Append pixel to *cluster 1 list*
    else
      Append pixel to *cluster 2 list*
    end
  end
  $c1_{new}$ = mean of *cluster 1 list*
  $c2_{new}$ = mean of *cluster 2 list*
  if $c1_{new}$ *is equal to* $c1_{current}$ *and* $c2_{new}$ *is equal to* $c2_{current}$ **then**
    break
  else
    Update current center for each cluster
    $c1_{current} = c1_{new}$
    $c2_{current} = c2_{new}$
  end
end

Algorithm 2: K means algorithm for K = 2

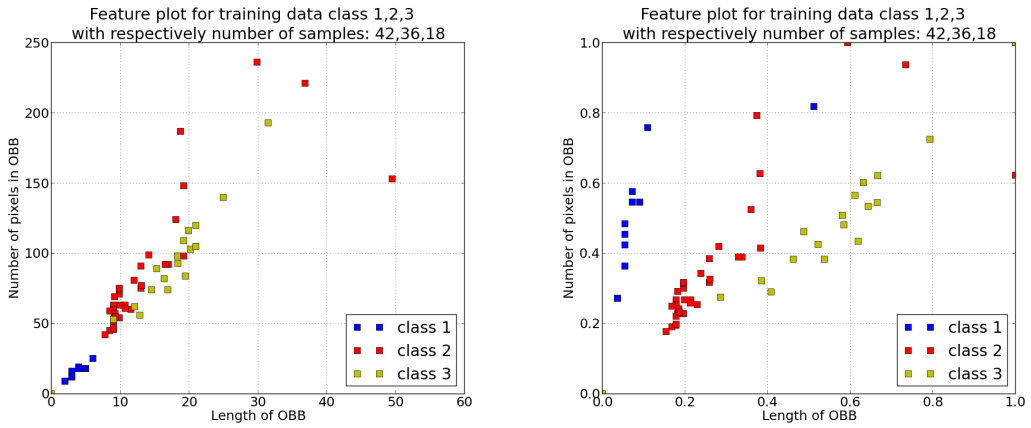
]

6.4 Classification

In the beginning of the project, the Perceptron classifier was implemented without using any libraries in order to learn how a classifier works. After gaining experience with classifier, a need for better classifier was needed. The argument is described in subsection 5.3.3. The SVM was integrated by using the off-the-shelf SVM library from skLearn [30]. The new training data (a), (b) and (c), which is shown back in figure 5.18, were fed into the preprocessing component and further into the segmentation component with the feature extraction. The output is the feature vectors, v_x and v_y. The selected features were the length and number of pixels in the oriented boundingbox. These is labelled as x and y axis respectively and shown in figure 6.27. The result shows in image (a) that the features is clustered more together. In order to cope with that, the two vectors were normalized as followed:

$$\hat{\vec{v_i}} = \frac{\vec{v_i}}{\max \vec{v_i}} \tag{6.1}$$

The result of this is shown in figure 6.27. where image (a) is before normalizing and image (b) is after. The next step is to create a classifier based on the training data set using different kernels.



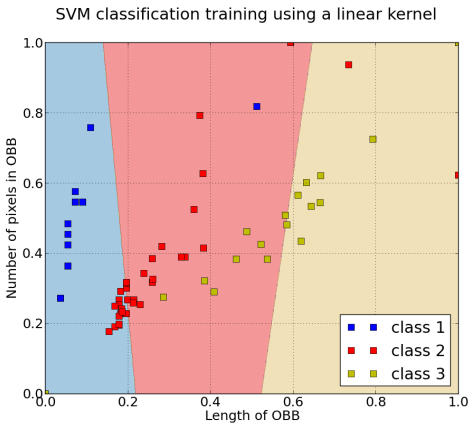
(a) Class 1

(b) Class 2

Figure 6.27: (a) Feature plot of training data class 1, 2 and 3. (b) Feature plot of training data class 1,2 and 3 normalized.

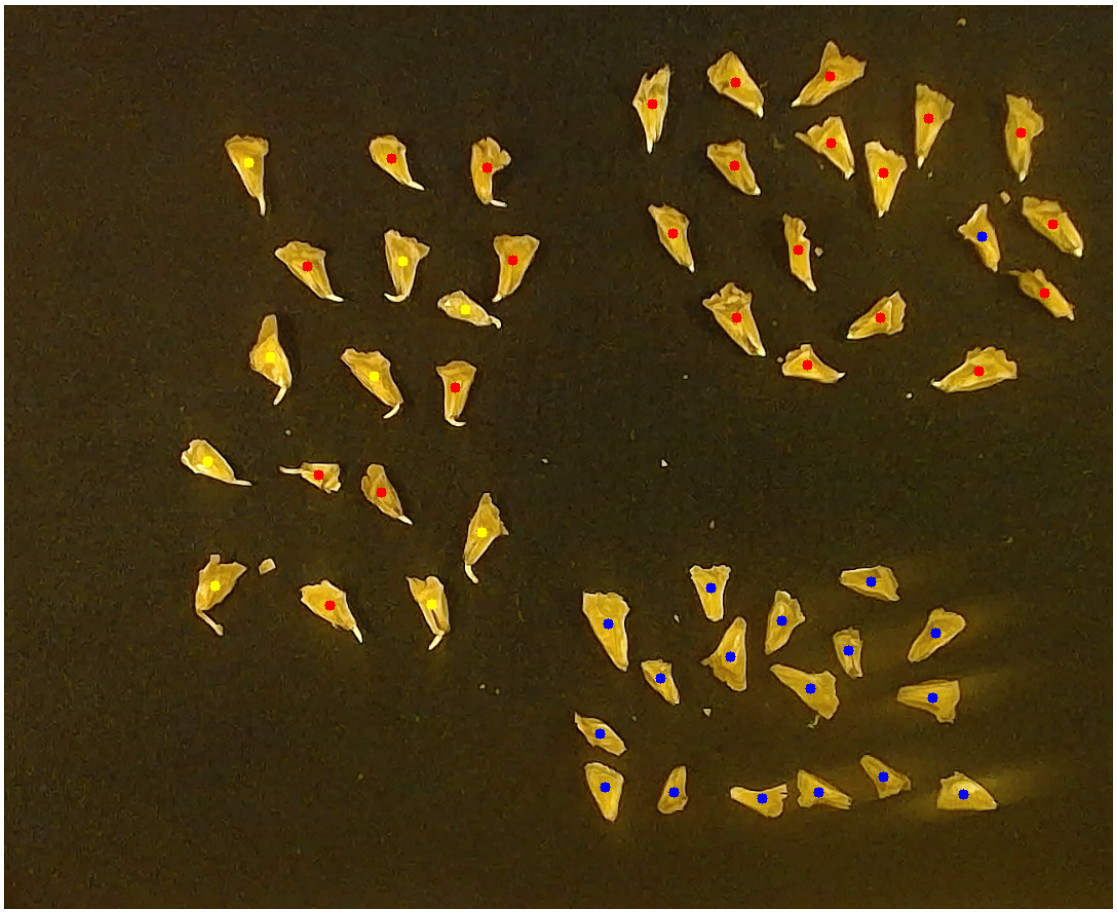
6.4.1 Result of SVM with different kernels

One thing that was not experienced regarding implementation of the Perceptron classifier was the influence of using different kernels and parameters. Using an example from SciKit learning [9] tree kernels were introduced. These are the linear, radial-base function and polynomial kernel. The parameter for the polynomial kernel is the number of degree. However the result were not useful and with an degree of 3 and above were a too heavy burden for the PC. Therefore the linear and radial base function (RBF) kernel were compared. The result with a linear kernel is shown in figure 6.28. The result using a radial base function, is shown in figure 6.29.
Two parameter was adjusted using the RBF kernel. These were the γ and C parameter [8]. The test of different parameters is shown in appendix G and appendix H. Observing the result the chosen parameters is using $\gamma = 0.8$ and $C = 1.0$. The training data in the appendix is sligtly different than the final result. Reason is that when the appendix was created, the preprocessing and segmentation optimization, which is described in subsection 5.3.5.
Observing the result in image (c) in figure 6.28, there is 9 out of 51 testing seeds that is misclassified. With no misclassification all the seeds in the right lower corner would be marked with a blue dot, seeds in the upper right corner would be marked with a red dot and finally seeds in the left side would be marked with a yellow dot. This is not the case and hence there exist 9 misclassified seeds in the image (c). This gives a classification rate of $1 - \frac{9}{51} = 82.35\%$. Using a RBF kernel, which the result is shown in figure 6.29, the classification rate with 6 out of 51 misclassified seed is $1 - \frac{6}{51} = 88.24\%$.



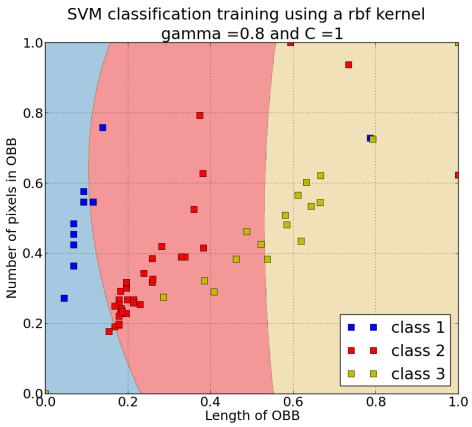
(a) Class 1

(b) Class 2

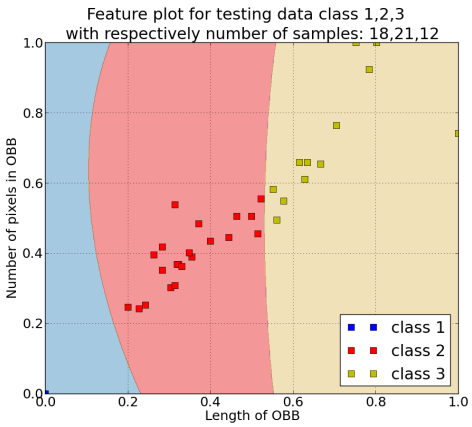


(c) Class 2

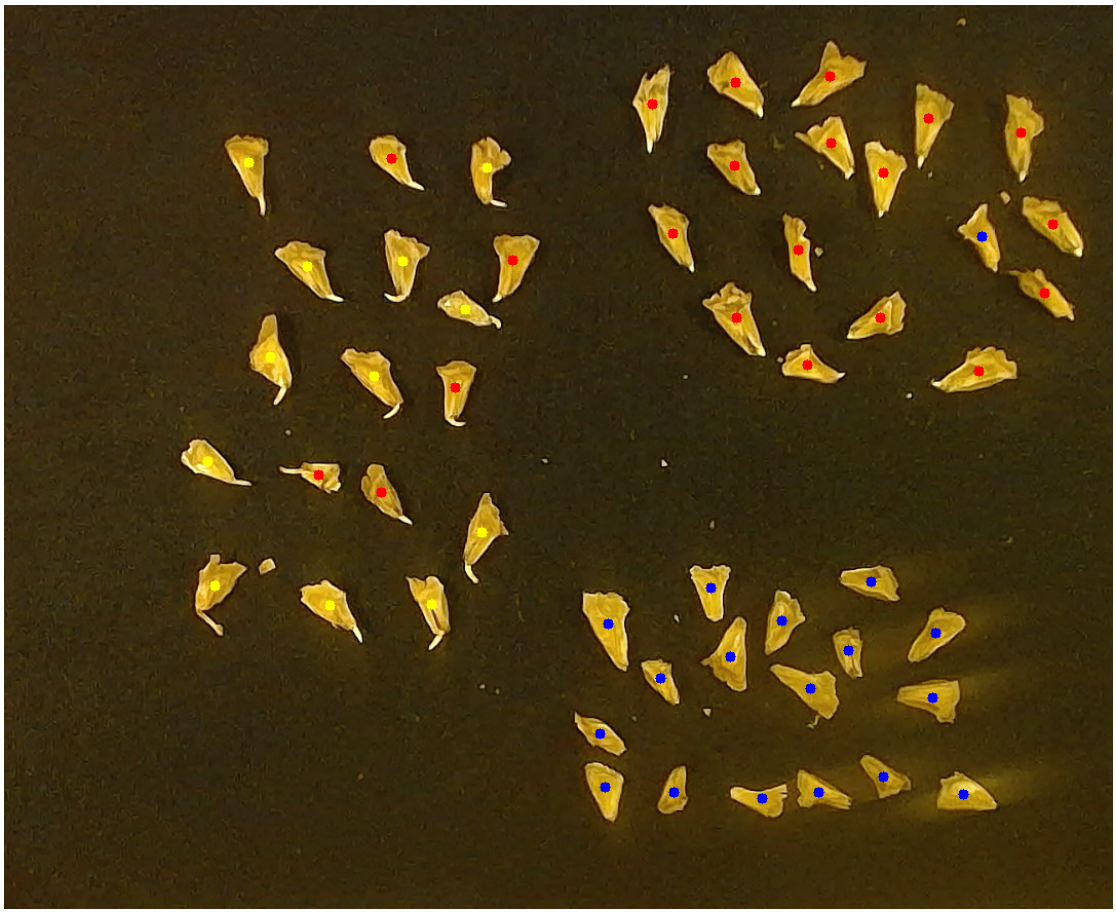
Figure 6.28: (a) Training data class 1 where seeds has no sprouts. (b) Training data class 2 where length of sprouts are OK. (c) Training data class 3 where length of sprouts is too long. (d) Class 0 shows seed with different sprout length.



(a) Class 1



(b) Class 2



(c) Class 2

Figure 6.29: (a) Training data class 1 where seeds has no sprouts. (b) Training data class 2 where length of sprouts are OK. (c) Training data class 3 where length of sprouts is too long. (d) Class 0 shows seed with different sprout length.

6.5 Outputting 3D coordinates

The output of the classification component is two lists, where the first list contains the center of mass for all the good classified seeds and the second list contains the center of mass for all the bad classified seeds. In order to have a robotic system to be able to pick the seeds, the center of mass coordinate (u,v) must be converted into a 3D location, which is relative to the camera frame. Later the extrinsic parameters must be calibrated together with the speed of the conveyor belt taking into account in order to have a robot grasping the seeds. Since the camera is mounted stationary and is perpendicular to the conveyor belt, the pinhole model can be used to map from a 2D (u,v) point into a 3D (x,y,z) location. It is required to know the focal length of the camera. This was obtained from previous coarse by running the a ROS */camera_info* topic and readout the focal length. These were $f_x = 1195$ and $f_y = 1193$ pixels for x and y respectively. The equation to map from 2D (u,v) to 3D (x,y,z) with a the known distance z from the conveyor belt to the camera, is as followed:

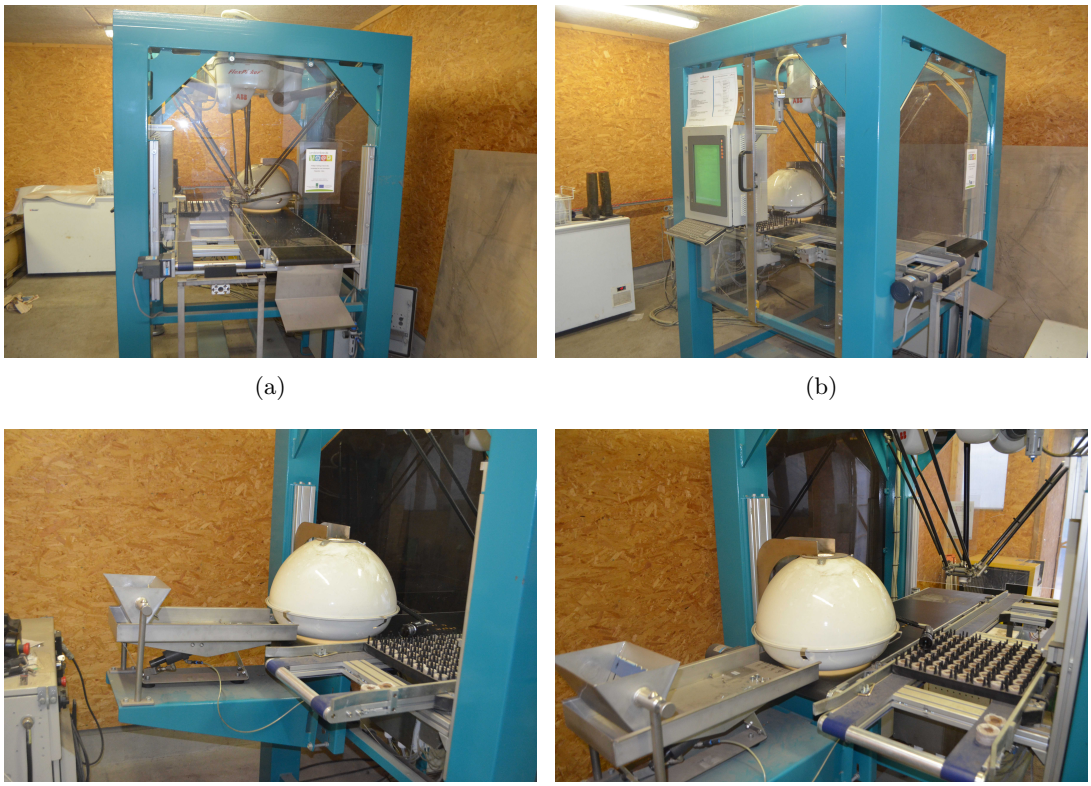
$$x = \frac{\left(u - \frac{columns}{2}\right) \cdot z}{f_x} \tag{6.2}$$

$$y = \frac{\left(v - \frac{rows}{2}\right) \cdot z}{f_y} \tag{6.3}$$

The OpenCV library defines the 0,0 location of an image, to be at the upper left corner. In order to take this offset into account, the offset is subtracted from the (u,v) pixel location in equation 6.2 and 6.3 respectively.

7 Robotic system

An future extension of the Master Thesis is to interface the ABB Flexpicker robot, which is shown in figure 7.1, 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 system. These components needs to complete a final test, before any more time can be invested in the robotic part.



(a) (b)

(c) (d)

Figure 7.1: (a)-(d). Setup with the ABB Flexpicker robot. The white sphere is the closed environment where a camera is mounted inside.

However for future work, the interface between the vision and robotic system is recommended to be implemented in the open source tool Robot Operating System (ROS) [3]. The *Chain of the robotic system* is described in section 5.1 in figure 5.1.

8 Discussion

Nothing yet
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 Conclusion

Nothing yet
Skriv noget a la, hvad det sværeste har været i dette project. En af tingene er det med at sikre at man træffer de rigtige beslutninger, som hvor man vælger at bruge sin tid på. Sparing til en makker kan være svært, når man arbejder alene med projektet. Det har hjulet rigtig meget, og jeg føler mig priviligeret at min vejleder holdte fast i et 14 dages møde gennem hele forløbet. Havde dette været skåret ned, havde jeg brugt min tid mere forkert.
En anden ting er forholdet mellem skole og Improoseed. Skolen ønsker at man tester sine metoder af med alternative metoder. Firmaet ønsker at ting bare virker. Dette er kontrovielt.
En anden ting er prioriteringen mellem projectarbejde, prioritering af andre fag sammentidig med projektet og privatlivets gang derhjemme. Det skal ikke betragtes som nogen undskyld, med hensyn til at jeg har kæreste og barn. Der findes altid en måde for at planlægge sig ud af tingene. Dermed sagt at det er ikke sværere at lave et speciale project når man har kæreste og barn. Dette har jeg bevist ved aflevering af denne documentation. Men det er sandt at specialet ikke bliver nemmere med kæreste og barn. Dette sætter bare endnu større krav til struktureringen af ens hverdag, hvilket jeg værdsætter at have oplevet. Dette har udrustet mig endnu mere i livets dagelige jungle.
Summa sumaroum har dette givet mig endnu mere erfaring i projektarbejde. Der er ting der skal afprøves og det gælder om at minimere spildtid. Dette er også derfor man i proffesionel sammenhæng har et project team og ikke ene mands projekter. Dog skal man være selvstændig og selv finde løsninger, men det er svært at vide om ens løsning eller fremgangsmåde er den rigtige, når man ikke dagligt kan spare med en makker.
En anden ting er rent dokumentationskrivning. Det hele har været en stor process, og derfor uden en projekt makker kan det være svære at holde den røde tråd gennem hele projektet. Dokumentation undervejs er blevet gjort, men meget er også ændret. For at sikre et mål med at have en classifier op at køre, men at man så ændre hele setup'et, så ændre man også hele koden og dokumentationen. Ændringer i projektet har altid en pris i form af ressourer.
Det at seed i forhold til LegoKlodser er noget ganske andet og øger kompleksiteten i projektet er vigtigt at pointere. Test og udvikling gøres svære.

10 Future work

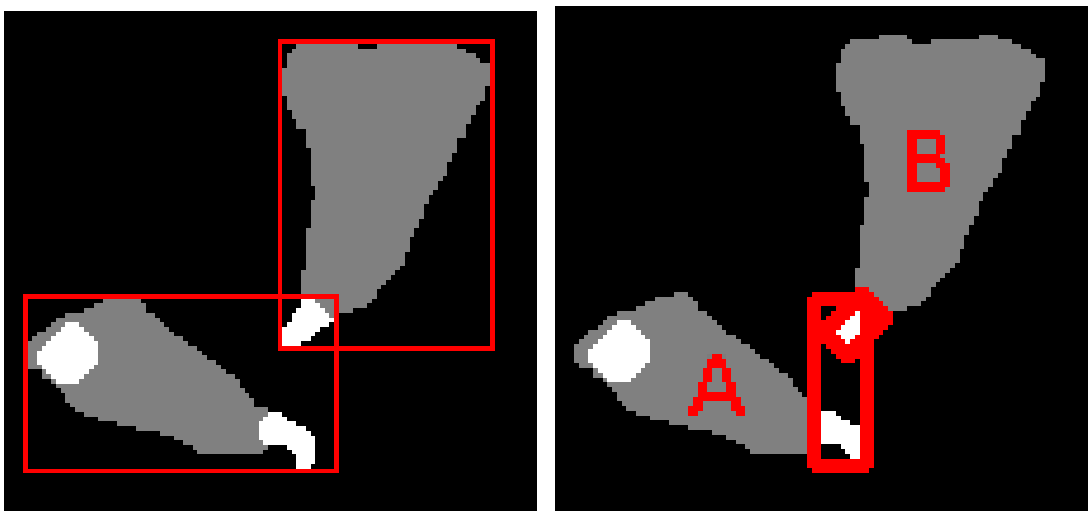
In this chapter different task for future work is explained. The overall future work is obviously to have a robust vision system, which is interfaced to a robotic system through ROS. However the future work described in this chapter is focus on the vision system only.

10.1 Seeds that is close to each other

A problem exist if the seeds width sprouts is too close to each other. Example of this is shown in figure 10.1. With the final implementation, each element from the front ground image is cropped out using a bounding box. In image (a) the two seeds are overlapping. In this case the seed labelled "A" in image (b), have an extra blob of white pixels which is processed by the K-means algorithm. This result in a too long sprout boundingbox which generates a false feature for this given seed. An idea of how to minimize this kind of problem is stated as followed:

- Get the seeds center of mass (COM) location, $center_{seed}$.
- Detect the different blobs in the ROI and calculate their individual COM, i.e. $center_1$, $center_2$,..., $center_n$.
- For each COM location, get a lines to the $center_{seed}$. This line contain all the pixel between the two points. The higher number of black pixels within the spanned line, the higher the probability will be of having a neighbour blob and not the real blob of sprout pixels.

If no sprouts is overlapping, but there exist multiple sprout blobs, calculation the Euclidean distance can be used to detect neighbour blob. This is based on the assumptions that each sprouts grow out from a pointy edge of the seed and not in the middle of the seed. Additional for each blob, check the pixel neighbours by expanding the blob with a dilation and then subtract the original blob. This will create a surrounding mask that gives pixel intensity values about the neighbour pixels. Taking the average of the ring with neighbours pixels will indicate if the blob is surrounded by black or gray pixels.



(a) Class 1 (b) Class 2

Figure 10.1: (a) Two seeds are close to each other which gives overlapping with the seed boundingbox. (b) Result gives false feature for one of the seed

10.2 Seed that is touching each other

Seeds that is touching each other can be a hard task to cope with. One option is adjust the speed of the conveyor belt or place an mechanically "shaker", which in general gives more spread to the seeds. However a test is needed to state how often this scenario happens. No matter what it creates problems for the grasping process. The best opportunity is to detect both seeds and let the robot pick one of them and let the other pass. Idea to detect the COM for seeds that is touching each other is stated as followed:

- In the front ground image, compare all the contour areas. If one object in the image has an area, which is outside of the normal distribution for seeds, then create a ROI around that object
- Within the ROI do a edge detection and subtract the edge from the ROI. This will perhaps clear out the touching points, but also ruin the contour of the sprouts.

Again this scenario can be very complex to deal with. First thing is to conclude how often this problem happens, before doing any implementing.

10.3 Semi-supervised learning

Instead of using thresholding and HSV mapping in order to create the front ground image and the seed and sprout image, different approached could be interesting. By the use of semi-supervised learning the front ground image could be created by having training data of background pixels and front ground pixels. This should be extended to find sprout pixels as well.

11 Bibliography

[1] Ros - camera calibration, .
URL http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration.
[Online; accessed 2015-05-19].

[2] Otsu threshold - opencv, .
URL http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html?highlight=threshold#cv2.threshold.
[Online; accessed 2015-05-23.

[3] Robotic operation system - ros, .
URL http://www.ros.org/.
[Online; accessed 2015-05-23.

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

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

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

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

[8] Svm rgb parameters, .
URL http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#example-svm-plot-rbf-parameters-py.
[Online; accessed 2015-05-21].

[9] Svm example, .
URL http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html#example-svm-plot-iris-py.
[Online; accessed 2015-05-21].

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

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

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

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

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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].

[23] 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.

[24] 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.

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

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

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

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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].

[34] 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].

[35] 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 Setup at ImproSeed Aps

The chosen camera, the Logitech C930e was extended with a wooden stick which is shown in figure A.1. 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. This is shown in figure A.2.

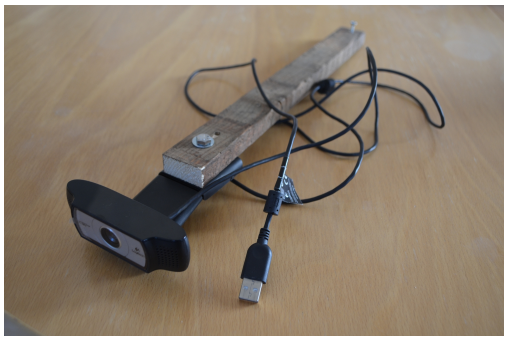


Figure A.1: Webcamera extended with wooden stick

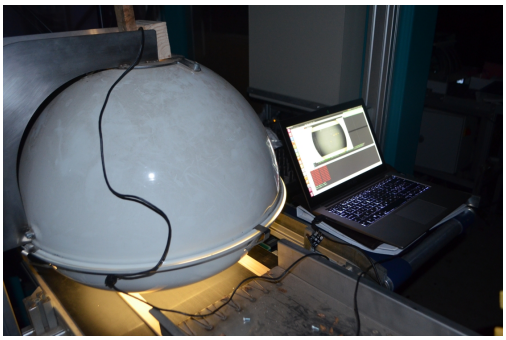


Figure A.2: Webcamera mounted inside sphere and connected to PC

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 needed 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 is discussed in future work, chapter 10.

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 A.3. The camera was thereafter mounted by having the two wodden sticks inside the cameras field of view, as shown in figure A.4. 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.

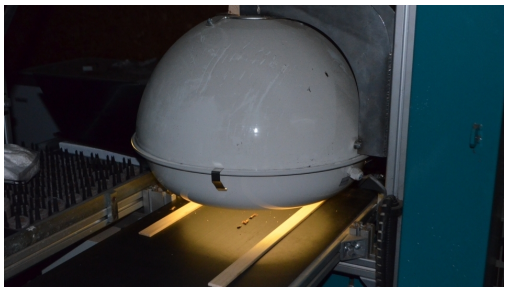


Figure A.3: Wooden sticks used to align distance

Figure A.4: Webcamera mounted inside sphere and connected to PC

A quick way to insure more space in the field of view would be to rotate the camera 90 degrees.

B Webcamera parameters

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 B.1, 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 B.1: Calling an OS command from the Python script to adjust camera settings

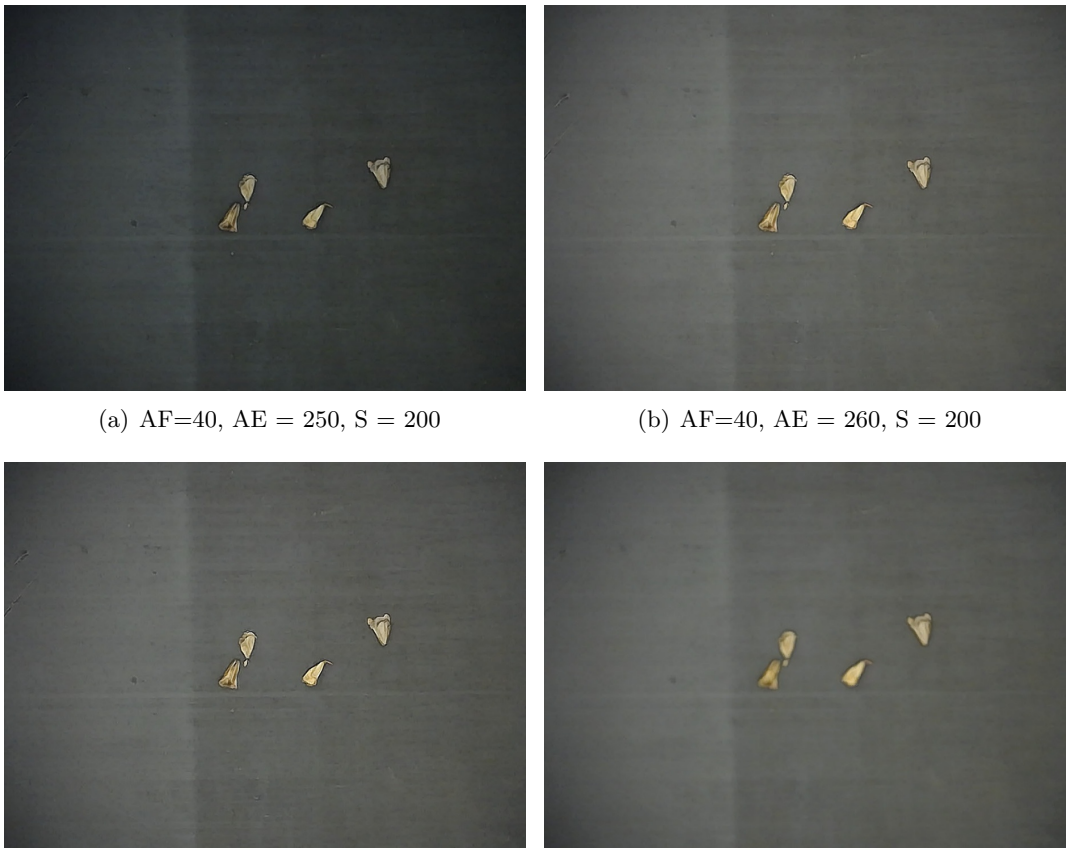
The reason for disable the autofocus is to avoid any uncontrolled behaviour of the web camera while the system runs. Since the distance from the seed on the conveyor belt and to the camera lens do not change over time, the argument for having a fixed zoom exist. If the autofocus is not disabled, the 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`

C Test of webcamera parameters

The Python script described in section 5.3.1 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 C.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

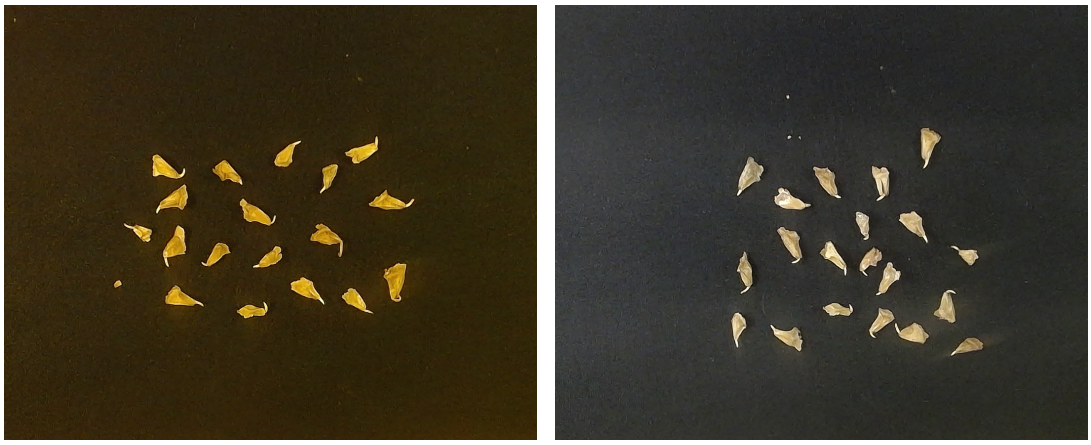
(d) AF=50, AE = 260, S = 200

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

D Light bulb Kelvin test

A test was carried out to see the result of using light bulb with different Kelvin temperature. The higher temperature in Kelvin, the colder the light is. The two kind of light bulbs which was tested has the following specifications:

- IKEA Light bulb, 2700K.
  - 7W, 315Lm, 60mA, E27. Model no: ES0806G7.
  - Current version is obsolete.
- RS LED Light bulb, 4000K.
  - 4W, 240Lm, E27, Item no: 786-9184.
  - Available through RS-online with item number.



(a) At 2700K (b) At 4000K

Figure D.1: (a) Image of seeds with 2700K. (b) Image of seed with 4000K.

From the test in figure D.1, the color seems to be more true with a higher Kelvin temperature. The background in image (a) has a dark green color, where the background in image (b) is more gray. However the light condition in both cases has not been ideal and with the current setup at ImprooSeed, see appendix A, the light condition is better.

E 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.

```
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 E.1: Using VideoCapture class from OpenCV

F Workplan for Improseed ApS

Dansk
Status for projektet efter aflevering er som følgende:

- Vision system:
  - Der er udviklet et software system, der kan skelne frø fra en baggrund ved brug af et Logitech C930e USB webcamera. Dette virker robust.
  - System kan lokalisere spirer på et frø på baggrund af farve. Dette er ikke robut nok.
  - Systemet kan trænes på baggrund af håndsorterede frøeksempler og klassifice nye frø der bliver indført i systemet.
- Robot system:
  - Der er ikke implementeret et interface mellem vision systemet og robot systemet.
  - ROS-industri er dog igang med at udvikle en MoveIt pakke (OpenSource) til interfacing af ABB robotter.[7]

Et standard web kamera ikke er velegnet til opgaven, men den udviklede software ser ud til at virke fint med billeder fra et DSLR eller kompakt kamera.

Slagplan for videre udvikling er som følgende:

a. Afsøge marked for kamera, der har bedre farvegengivelse af spire.

b. Tilpas frø segmentering med nyt kamera.

c. Tilpas spire segmenteringen med nyt kamera.

d. Gentræne klassificeringkomponentet med gode og dårlige frø som ny træningsdata med nyt kamera.

e. Kalibrere kameraets interne parameter. Brug evt tutorial med ROS kamera kalibrering [1].

f. Kalibrere kameraets eksterne parameter, dvs. mapping mellem kameraets workspace og robottens workspace.

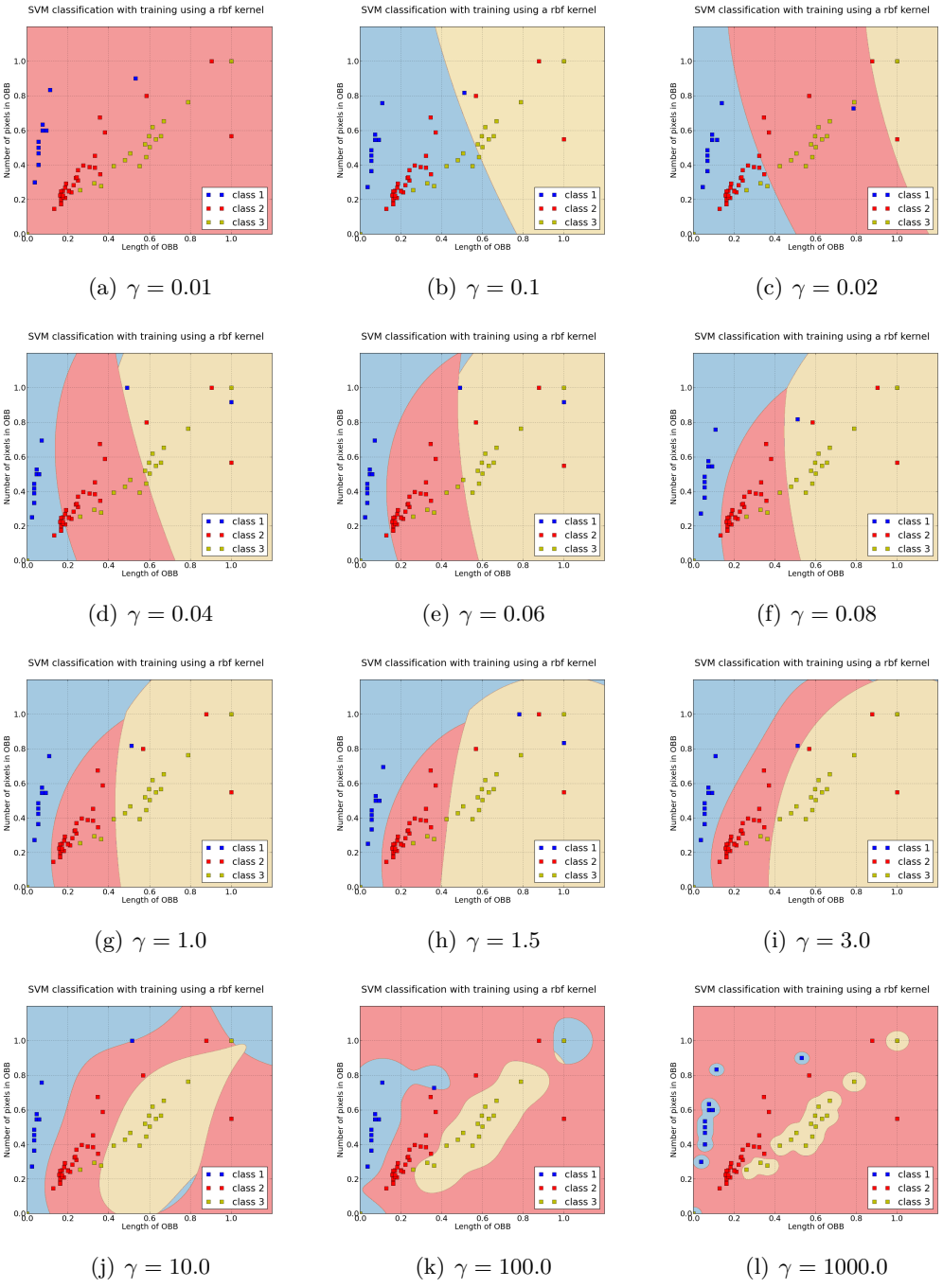
g. Udarbejde interface to robotten ved brug af ROS systemet.
   - Fastlæg hvilket interface der ligger til ABB Flexpicker robotten.
   - Opbyg ROS node, der publisher et 3D punkt og teste om dette kan modtages i robotsystemets buffer, samt at robotten flytter sig til det ønskede 3D punkt.

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/ChristianLiinHansen/Master-thesis-doc`.

G Testing γ parameter for the RBF kernel

This appendix show the result when charging the γ parameter. The correction parameter C is fixed at 1.0.



(a) $\gamma = 0.01$ (b) $\gamma = 0.1$ (c) $\gamma = 0.02$

(d) $\gamma = 0.04$ (e) $\gamma = 0.06$ (f) $\gamma = 0.08$

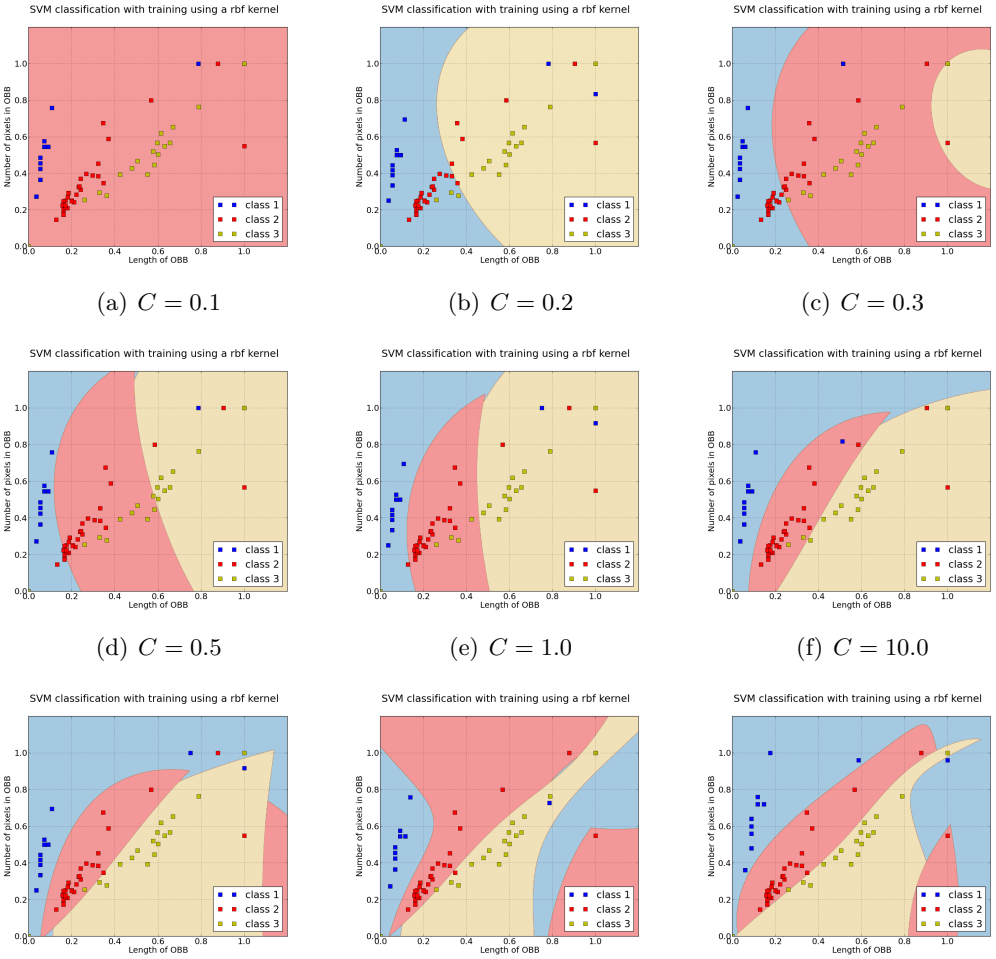
(g) $\gamma = 1.0$ (h) $\gamma = 1.5$ (i) $\gamma = 3.0$

(j) $\gamma = 10.0$ (k) $\gamma = 100.0$ (l) $\gamma = 1000.0$

Figure G.1: Images where the γ goes from 0.01 (a) to 1000 (m).

H Testing C parameter for the RBF kernel

This appendix show the result when changing the C parameter. The gamma parameter γ is fixed at 0.8.



(a) $C = 0.1$ (b) $C = 0.2$ (c) $C = 0.3$

(d) $C = 0.5$ (e) $C = 1.0$ (f) $C = 10.0$

(g) $C = 100.0$ (h) $C = 1000.0$ (i) $C = 10000.0$

Figure H.1: Images where the C goes from 0.1 (a) to 10000 (l).