Tree Seed Classification System based on Artificial Intelligence

Træfrø Klassificeringssystem baseret på Kunstig Intelligens

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 M.Sc.E degree in robotic systems, 40 ECTS points

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

Todo list

Abstract

English

This report documents a master thesis, which has been carried out during the 4th semester of the Master degree in Robotic System Engineering at the University of Southern Denmark (SDU) in Odense. The project has been developed in cooperation with the company ImproSeed ApS, which is working with forestry planting process in order to improve the efficiency of growing trees. At the moment the efficiency is between 60-75% and hence an improvement of the efficiency is the overall goal. Therefore the subject of the project is sorting tree seeds using a robotic system with an ABB FlexPicker robot and a computer vision system. The focus in this project is the implementation of the vision system. The computer vision system is able to detect and classify tree seeds, based on the features of their sprouts, as either good or bad. The input to the vision system is RGB images which is fed from a web camera. The seeds lie on a flat surface and the camera is mounted perpendicularly above the seeds inside a box or sphere to keep an uniform light distribution. The system is training a support vector machine classifier using the principle of supervised learning, where training data contains images of seeds with different characteristics. The computer vision system has a classification rate of of 88.24%, which is above the current efficiency of growing trees.

Danish

Denne rapport dokumenterer et specialeforløb, som er blevet gennemført i løbet af fjerde semester på kandidatoverbygningen i robotteknologi ved Syddansk Universitet i Odense. Projektet er udviklet i samarbejde med firmaet ImproSeed ApS, som arbejder med plantningsproces inden for skovbrug for at forbedre effektiviteten af voksende træer. I øjeblikket ligger effektiviteten mellem 60-70%, hvor forøgning af effektiviteten det overordnede mål. Derfor er emnet sorter træfrø ved hjælp af et robotsystem indeholdende en ABB FlexPicker robot og et computer vision system. Fokus i dette projekt er implementeringen af visionsstemet. Dette er i stand til at finde og klassificere træfrø, baseret på de egenskaber af spirerne, som enten er gode eller dårlige. Input til visionsystemet er RGB-billeder, som tilføres fra et web-camera. Frøene ligger på en plan flade, og kameraet er monteret vinkelret ovenover frøende inde i en boks eller kugle for at opretholde en ensartet lysfordeling. Systemet træner en support vector machine classifier ud fra princippet i supervised learning, hvor træningsdata indeholder billeder af frø med forskellige karakteristika. Computer vision systemet har en klassificeringsrate på 88.24%, hvilket er over den nuværende effektivitet for voksende træer.

Contents

1 Preface

I participated in the field robotics summer coarse at SDU in 2014. From here I grew a personal interest in field robotic and computer vision. In one of the coarse lectures, Henrik Skov Midtiby gave introductions to different aspect in computer vision used in the field. After the lecture a discussion with Henrik Skov Midtiby developed ideas for a master thesis. Later a project description was handed in, which was the start of this master thesis project.

I would like to thank Henrik Skov Midtiby for being supervisor on the master thesis and for his support and interest in the project. Also I thanks fellow students and the personnel at RoboLab for their interest and ideas to the project. Finally I thank Peter Benfeldt, the project owner for the collaboration in the project.

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 for extra explanation of the following 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 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 has been implemented. Result at test of the different component is described.

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 Github repository:
`https://github.com/ChristianLiinHansen/Master-thesis-doc`

All the source code, written i Python, can be downloaded from the following Github 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*. In figure 2.1, image (a) 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 trees. 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 category - good condition.* Seeds are ready for planting. Send seeds to planting process.
- *Red category* - bad condition. Seeds are not valid for planting. Discard 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. Implementing this will make the system be able to classifier different kind of tree cultivars. 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 main goal in this master thesis is to have a vision system, that is able to classify each seed as either a good or bad. 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 feeds images of seeds from the conveyor belt and is 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 is transformed from the image frame to a robot frame. The coordinate is transferred to a robotic system in order to let a robot perform grasping of the seeds. If time allows, work on the robotic system will be made.

The system needs to classify seeds, based on the sprouting process for each seed. Having a system, that only handles one cultivar is limiting. Therefore the system should be able to learn how different seeds cultivars looks like. The idea is based on the principle of supervised learning [35], where the user has labelled data, i.e. seeds for different states in the sprouting process and hence can uses that as training data to teach the system. After the systems knows how different cultivars looks, it can classified new incoming seeds based on what the system has learned. The result would be a more universal system, which can handle multiple tree cultivars.

The inputs images from the camera comes in two folds. First the training data is fed to the system, which teach the system. Then testing data is fed to the system and each seeds category and center of mass is extracted. A block diagram illustrating the input and output of the vision system is shown in figure 3.1.

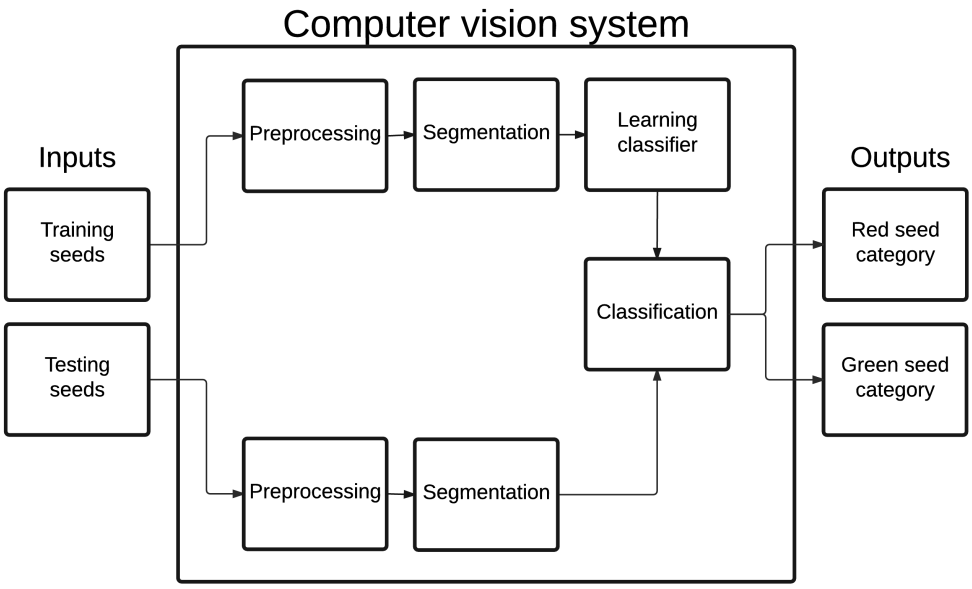


Figure 3.1: Block diagram of the computer vision system.

- Inputs
  - Training seed: These seed are labelled data, i.e. the image includes seeds which shares the same characteristic. An other training data contains seeds with 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.
- Computer vision system
  - *Preprocessing.* Process input RGB image into binary images to prepare images for segmentation.
  - *Segmentation.* Tracking and extract features for each seed.
  - *Classification.* Teach the system with training data and classify the testing data.
- Outputs
  - Red category: A bad classified seed with center of mass.
  - Green category: A good classified seed with center of mass.

3.1.1 Sensor for vision system

It is important to decide which sensor to use in the project. A literature survey has been carried out in order to get information about what other people have done, which relates to the project. This is explained in chapter 4.

3.2 Deliverables

At the end of the master thesis, the following source code and documenting will be delivered:

- A seed detecting method based on computer vision, which can classify each seed.
- A learning method based on AI.
- 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 [25]. 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

Hyperspectral camera

A camera is used to sense the incoming seeds. By using generel RGB cameras, such as DSLR, compact and webcamera the reflectance of red, green and blue response is extracted. This is within the visible spectrum from 400 to 700 nano meters. Limiting the information range within the visible light for the human perception is limiting. From literature [29] a hyperspectral camera generates data simultaneously in spatial and spectral dimension. This makes it possible to extract the reflectance outside the visible light range where other electromagnetic signatures is available. Work from other people shows how a hyperspectral camera has been used in order to extract information in the near-infrared (NIR) part of the spectrum. This techniques has been used in different application, like detecting the quality of wine-grapes [41], rice cultivar identification [33] and many others like mineral exploration, agriculture, and forest production. [44]. From literature [34], the amount of water containment and chlorophyll in biomass can be extracted, also from the NIF spectrum. This can perhaps be used to describe the status of the seed in general. It is assumed that there exists a significant difference in the reflected wavelength for each pixel, if the pixels comes form the seed or sprout.

Seedling classifier

The bachelor project by Lasse Simmelsgaard (LSB) [22] describes the Perceptron, Support Vector Machine and Random Forrest classifiers. The approach by LSB was to learn about classifiers by implementing the Perceptron by himself and then later user more powerful state-of-the-art classifiers like the Support Vector Machine (SVM) and Random Forrest (RF). LSB states that with Legendre features, a SVM classifier with a radial basis function kernel (RBF) is the best choice for classifing real world seedling samples. This is highly relevant for the master thesis. Implementing the Perceptron to learn about classifiers will be completed. Using other classifiers as the SVM will be completed.

5 Development process

This chapter describes the development process through the project.

5.1 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 a 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.

5.1.1 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.

5.1.2 Coding language and tools

An important task was to decide coding language for implementation. C++ has been the primary coding language through the master programme, but Python [15] was considered due to its higher abstraction level. Both language supports OpenCV [5], which is a open-source computer vision library. However Python give access to 2D plotting library Matlibplot [31] and data analysis library Scikit-learn [40]. 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, OpenCV 2.4.9, Numpy 1.6.1 [43], Matlibplot 1.1.1rc , SciPy 0.9.0 [32] and Sklearn 0.16b1 .

5.2 Project methods

It is important to have a project methods in order to structure the project. The chosen method, named *Chain of Project*, includes building up a chain of simple components with room for individual optimization. The vision system in this project contains different component, such as *Get image*, *Preprocessing*, *Segmentation*, *Classification* and *Outputting*. The principle is illustrated in figure 5.1 for the vision and the robotic system. The boxes with green and red boarder in the figure, indicate which component that got implemented and not implemented respectively. By using this *Chain of Project* method data flow is better established and secured compared compared to the alternative methods. An example of alternative methods would be to complete a component and use additionally time to optimize that given component. The risk is to not have a flow up and running before the project deadline. Working with natural growing seeds can be more time consuming and lead to delays and potentially sets the project on hold. By using the *Chain of Project* method, the project becomes more agile, due to tasks can be more independent and flexible of each other.

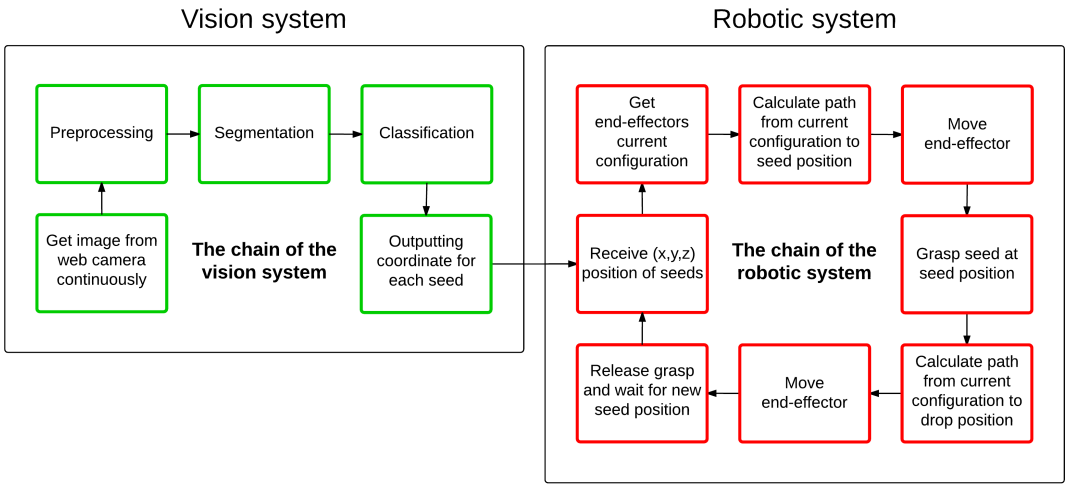


Figure 5.1: The chain of the project for the vision and robotic system. The green and red boxes indicate which component that was implemented or not in the project.

5.3 Proof of concepts

Following the *Chain of Project* method, described in section 5.2, the approach was to implement a minimum of functionality for each component to faster establish the data flow. The chain of project was divided up into two proof of concepts. The first proof of concept, *Black Pepper Detection System*, was implemented to have basic functionality of the preprocessing and segmentation component and get familiar with OpenCV and Python. This is described in subsection 5.3.1. The second proof of concept was implemented with relatively simple Perceptron classifier to learn how classifier and supervised learning works, by implementing it from scratch. This is described in subsection 5.3.2.

5.3.1 Black pepper detection

This system was implemented without the classifier component, which is illustrated in figure 5.2. The preprocessing component is mapped the RGB input image into a grayscale image and a threshold function is applied to get a binary image, where the black peppers were the foreground. The binary image was filtered with morphology to reduce the contours in the segmentation component. In the segmentation component the function *FindContours* from OpenCV was used to get the contours for each black pepper. Using the function *FindContourArea* from OpenCV, together with the central moments, the center coordinate was extracted and illustrated in the RGB image, which is shown in figure 5.3. This results shows a success in having the preprocessing and segmentation component implemented. The implementation was inspired from source [39]

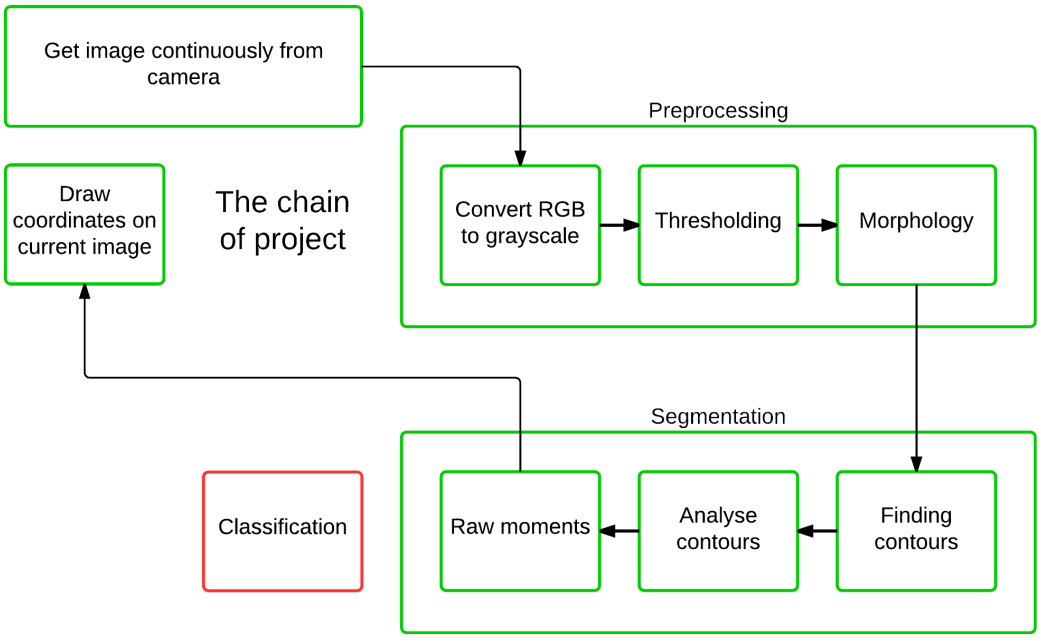


Figure 5.2: Testing proof of concept without classifier component. The green boxes indicate components that was implemented and the red box indicate the component that was skipped for this given implementation.

The next step is to classify between different shapes, like squares and circles. This is described in subsection 5.3.2.

5.3.2 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

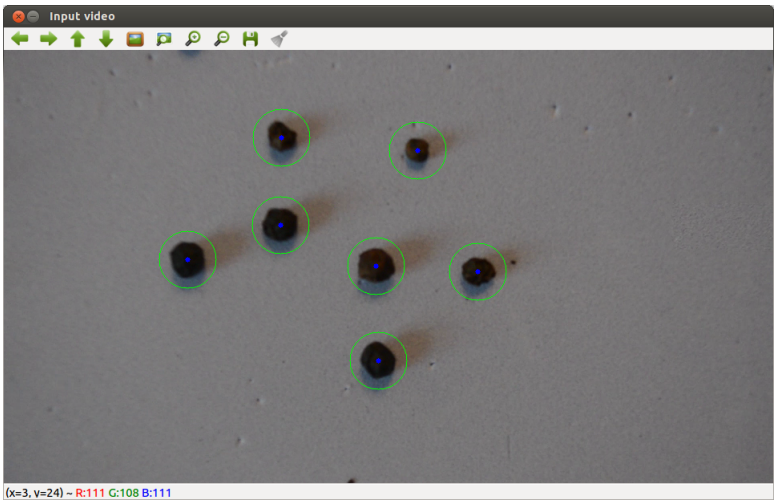
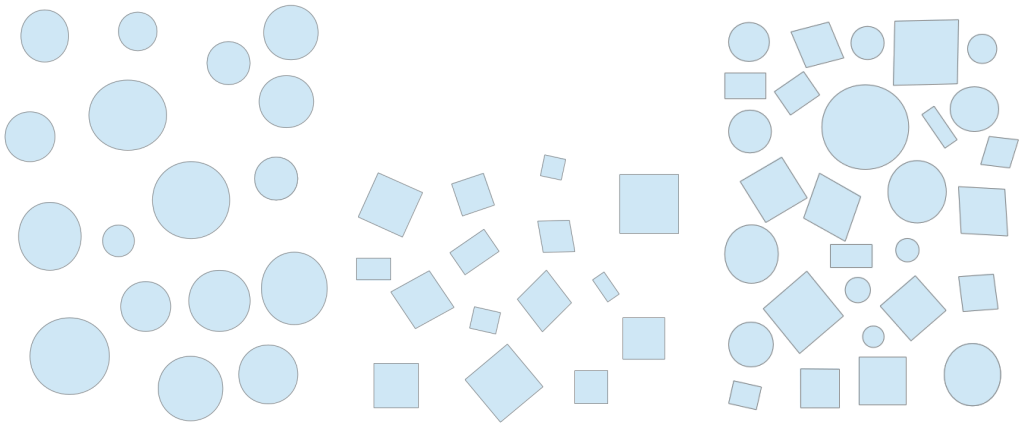


Figure 5.3: Black pepper detection system. Detecting peppeers and indicating each peppers center of mass with a blue dot. The green ring around the seeds is added for better visualization of tracking.

data that has been labelled [35]. 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.

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 5.4.



(a) (b) (c)

Figure 5.4: Two training images a) and b) with circle and squares respectively). Testing image c) with mix of square and circles.

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 [38] 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.

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

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 5.5, where the red feature point represent data from figure 5.4(a) and the blue represent data from figure 5.4(b).

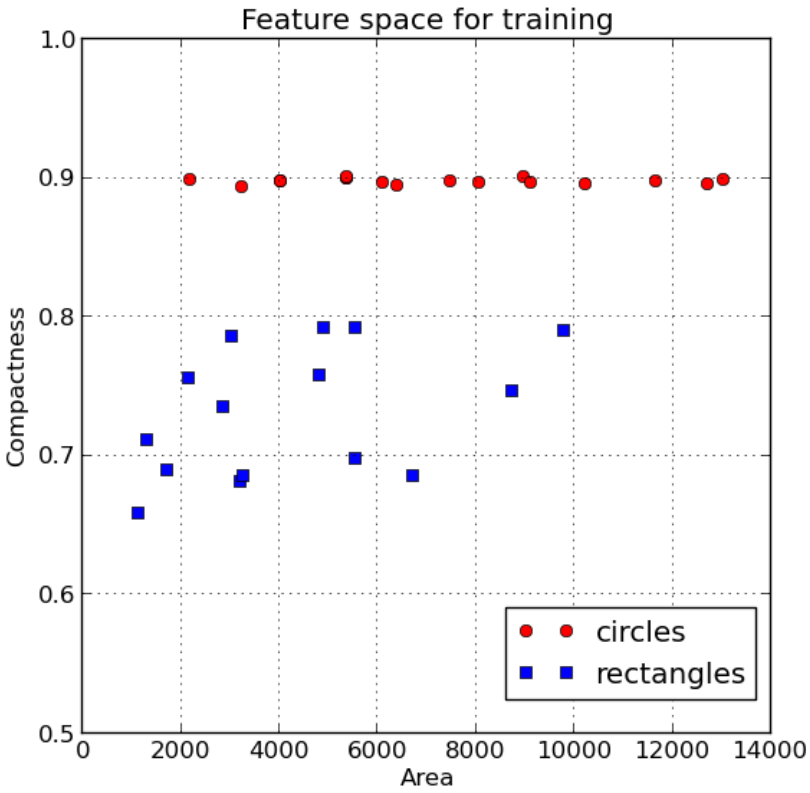


Figure 5.5: Feature space of compactness and area of each object in the training data

The result shows that circles and rectangles can be linear separated by using the feature *compactness*, which is the most important feature here. 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.

The Perceptron

It was chosen to implement the Perceptron classifier to get hands-on with classifiers in general. The Perceptron will find a solution if the data is linear separable [22]. The Perceptron is relatively simple to implement compared to other classifiers, like e.g. Support Vector Machines. In order to let the Perceptron find a solution, features needs to be linear separable. Otherwise the Perceptron can be extended to handle non-linear data by mapping the data into higher dimension, which makes the data linear and then map the data back again to non-linear representation. The computation time will increase when mapping to higher dimension, but this can be handled by using a kernel function, which reduce the number of multiplications[22], page 10. This is also known as the *Kernel trick*.

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 [37], which classify each output to either 1 or -1. With the update weighs and bias the linear classification line is defined [22] 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 5.7(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: $w_0 = w_1 = b = 0$

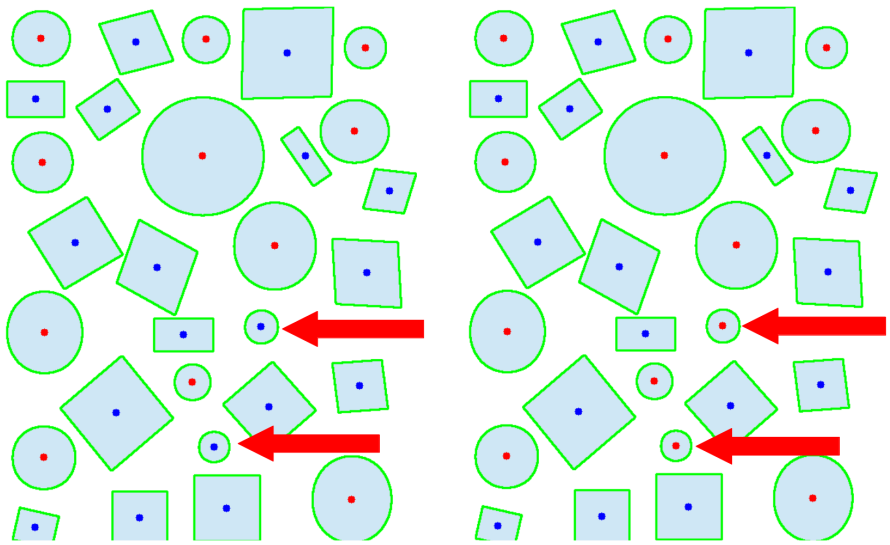
```
while runFlag is true do
    errorCounter:= 0
    for i data in trainingData do
        if ⟨w_i, x_i⟩ + b >= 0 then
            result:= 1
        else
            result:= -1
        end
        error:= desired output - result;
        if error is not zero then
            Update the weights and 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 5.6(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 5.6(b).



(a) (b)

Figure 5.6: 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 5.6(a) is explained by investigating the testing data. In figure 5.7(a) the convergence of the weight vector, bias term and the error is shown. The algorithm terminates after approximately 160 iterations. In figure 5.7(b) the Perceptron finds a solution using the training data. In figure 5.7(c) this solution do misclassification of two objects. In figure 5.7(d) the misclassification is two objects with a relative high compactness and small area, which explains the result in figure 5.6(a).
Sometimes the Perceptron do classify correct, which is shown in figure 5.6(b). Again this can be explained by investigating the classifier. In figure 5.8(a) the convergence of the weight vector, bias term and the error is shown. The algorithm terminates after approximately 180 iterations. In figure 5.8(b) the Perceptron finds a solution for the classifier, by using the training data. In figure 5.8(c) this solution separates the features. The result is a correct classification of the testing data which is shown in figure 5.8(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. From literature [22] the two state-of-the-art classifiers, Support Vector Machine (SVM) and Random Forrest (RF) is presented. The SVM is based on the Perceptron algorithm and hence it was a natural choice to replace the Perceptron with a Support Vector Machine (SVM), using the SVM library [40]. It would be interesting to compare the result of SVM with the RF classifier, but this was not archived in the project due to other priorities, such as optimizing the preprocessing and segmentation components

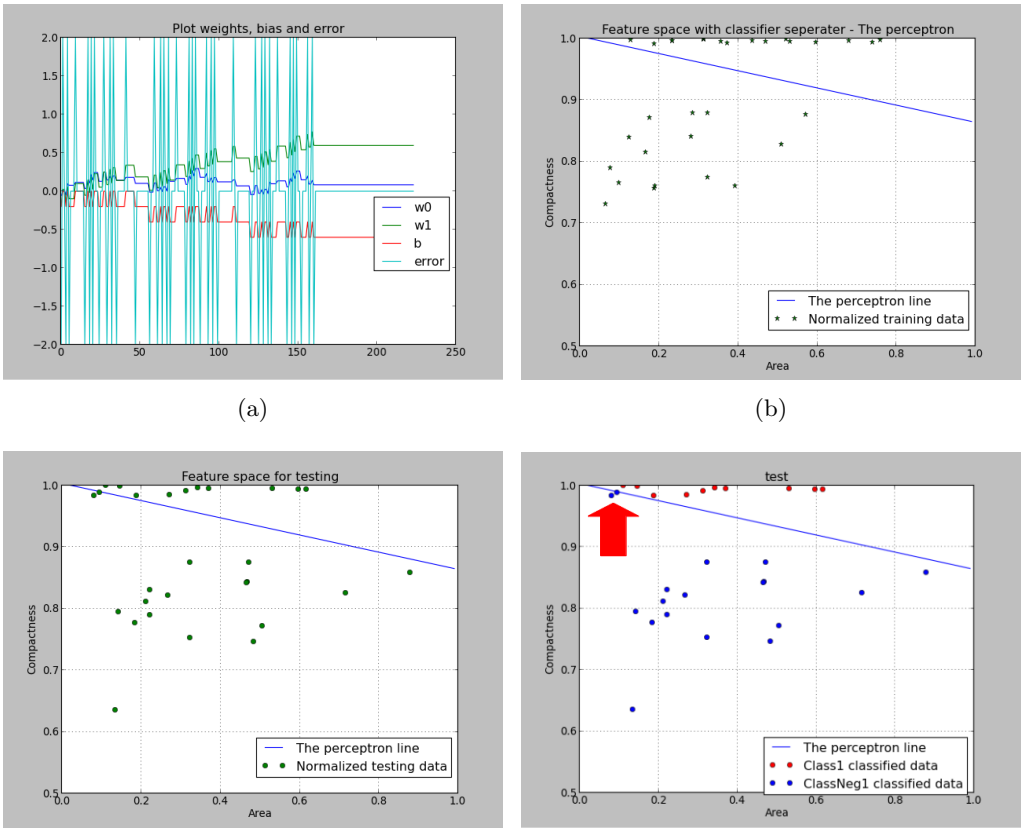


Figure 5.7: Figure a) show the convergence of the weights, bias and error. Figure b) show the separation line based on training data. Figure c) shows the testing data with the separation line. Figure d) shows the classification result, with two misclassified objects

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.

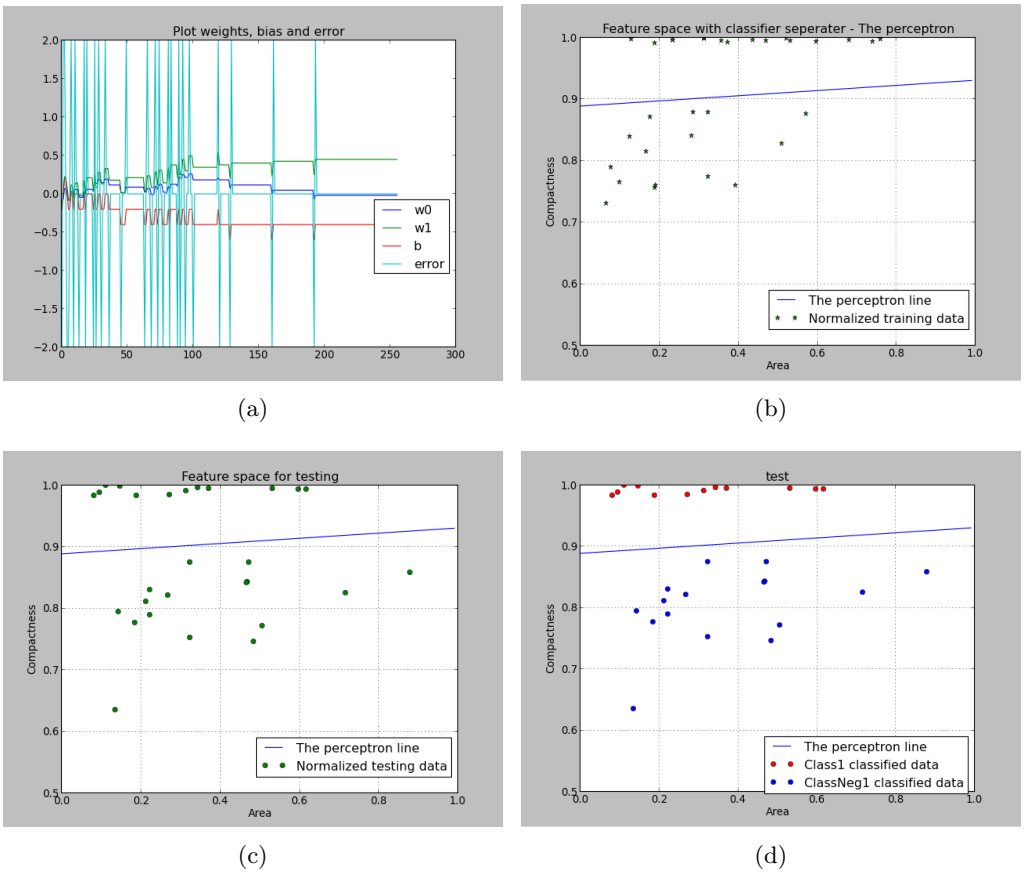


Figure 5.8: Figure a) show the convergence of the weights, bias and error. Figure b) show the separation line based on training data. Figure c) shows the testing data with the separation line. Figure d) shows the classification result, with no misclassified objects

5.4 Optimizing

It was concluded after the implementation described in subsection 5.3.2 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.

5.4.1 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 [44]. 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 [29], classify different rice cultivars [33] and finding grape seed characters [41].

Many companies that sells hyperspectral camera [16], [14], [21] and [20] 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 a high-cost camera. A medium-cost solution [1] with Ethernet interface is an option. However to get started with a Basler camera, the minimum buying of equipment is:

- Camera: Scout Series GigE Camera. 659 x 494 resolution = 4240 DKK
- Lense :Computar M0814-MP, 8mm, F1.4 = 1858 DKK
- Frame grabber: NI PCIe-8235, Quad GigE Vision Board = 6950 DKK
- Ethernet cable 1 m: 67 DKK
- I/O Cable: 12pin Hirose to 2x 12pin Hirose (Y-Cable) = 2825 DKK
- Total sum for the absolute minimum equipment is **16535** DKK

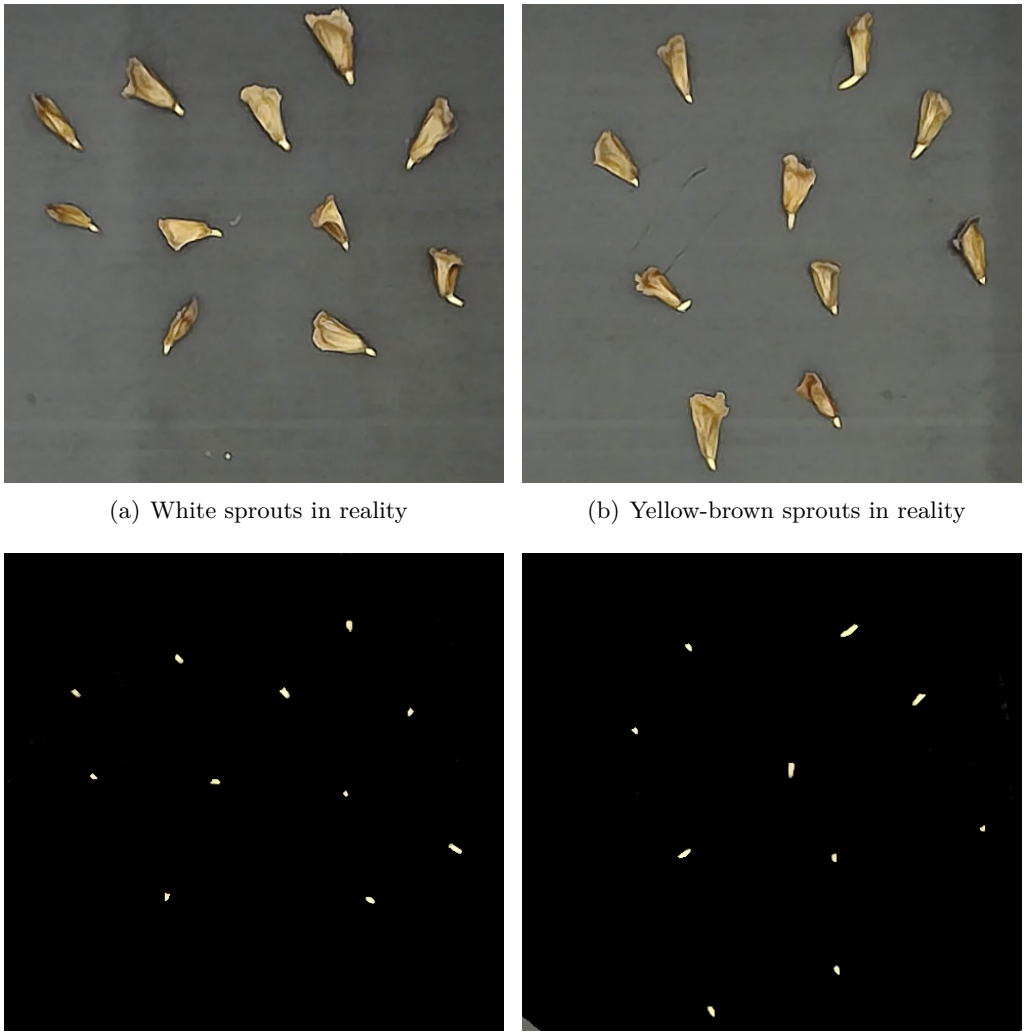
Define a suitable camera for this project is a challenged task, since several parameters exists. In this project the three most important parameter is the complexity of interfacing, the cost and the availability of the camera. As described in section 5.2, the approach is to have the the chain of the 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 an 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.

5.4.2 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 an image script was implemented in order to capture the proper training data. This implementation together with a video link of the demonstration 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 have a more whiter color in the image compared to the human perception. Images (a) and image (b) in figure 5.9 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 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 5.9. 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 5.10. The calculated rounded mean and standard deviation is shown in table 5.1.



(a) White sprouts in reality

(b) Yellow-brown sprouts in reality

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

(d) White sprouts extracted from (b)

Figure 5.9: (a) Yellow-brown sprouts.(b) White sprouts. Image (c) and (d) shows the extracted sprouts for image (a) and (b) respectively.

From the result from the 3D plot in figure 5.10 and table 5.1, a great variance is shown 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 5.11. 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 5.9 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 5.4.1 still remains. The length of a sprout is an important factor for classifying seeds and is used as primary feature in this project. The number of white pixels can be used as a second feature to verify the bounding box. If a sprout bouding box has a relatively long length but few number of white pixels, then this bounding box properly spans white pixels that is not part of the real sprouts.

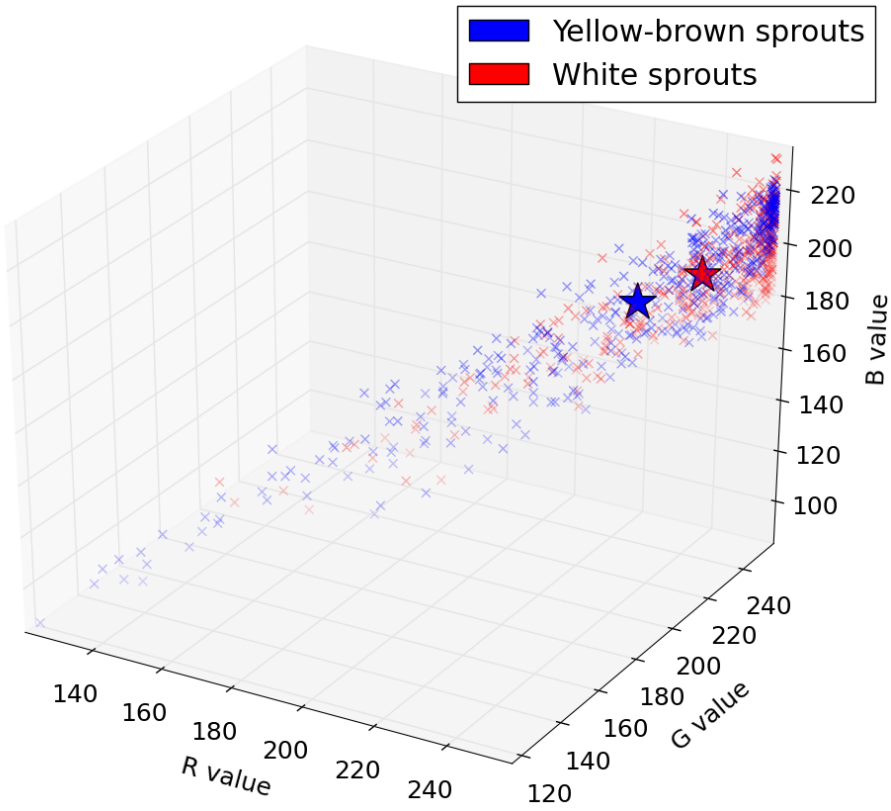
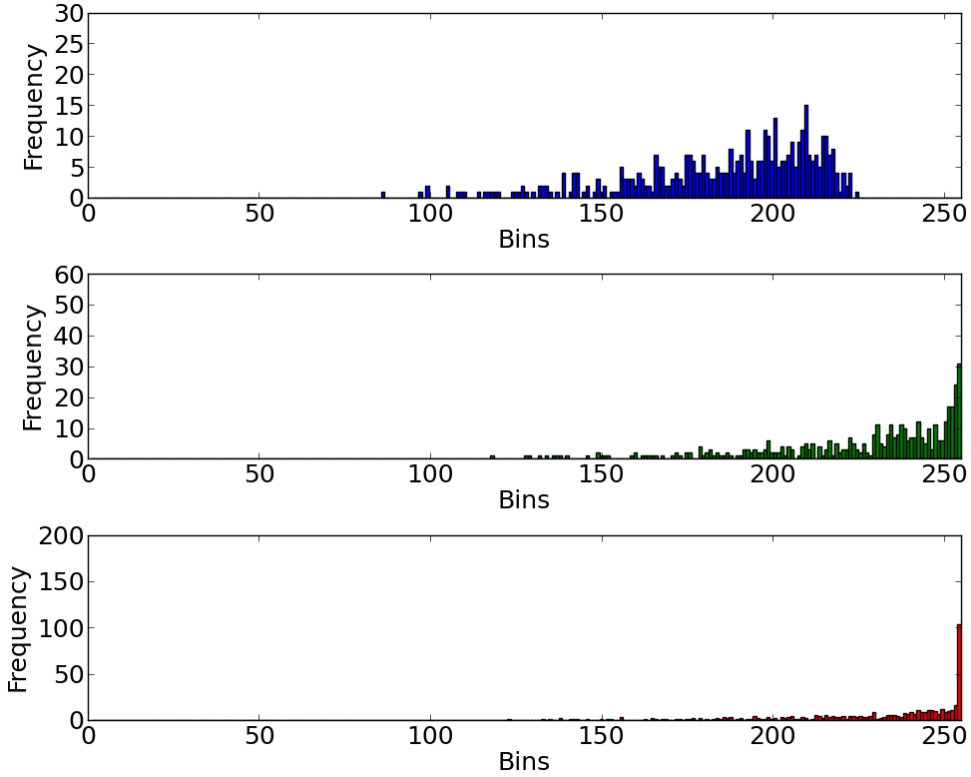


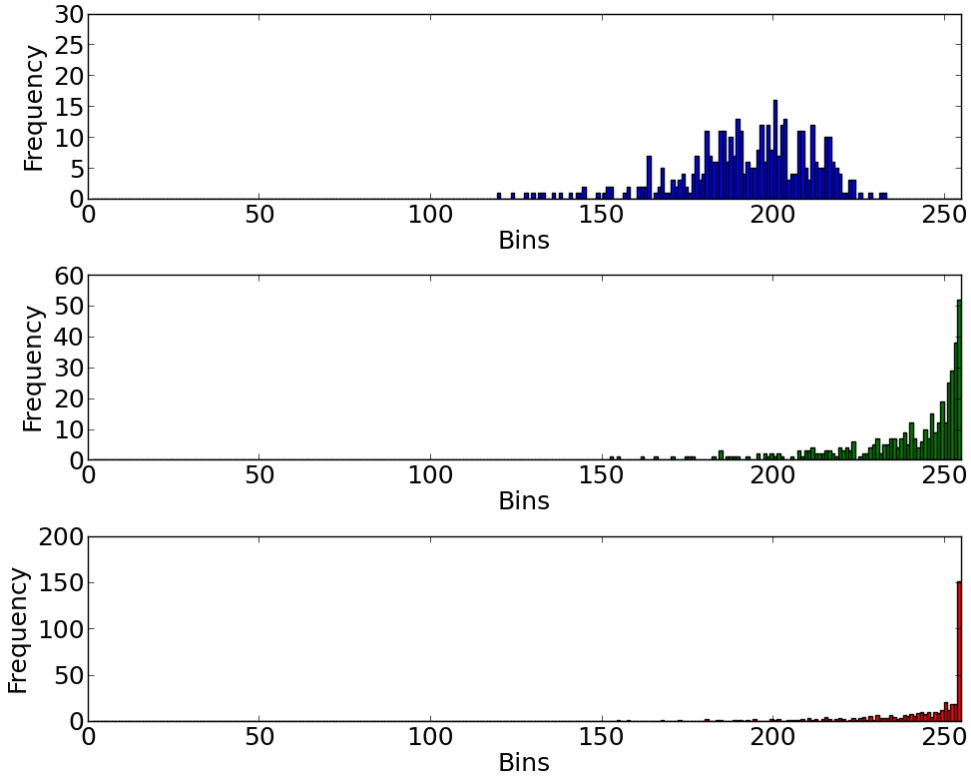
Figure 5.10: Plotting the RGB values for yellow-brown and white sprouts, togeter with their mean (Star

	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



(a) RGB histogram of the yellow-brown sprouts

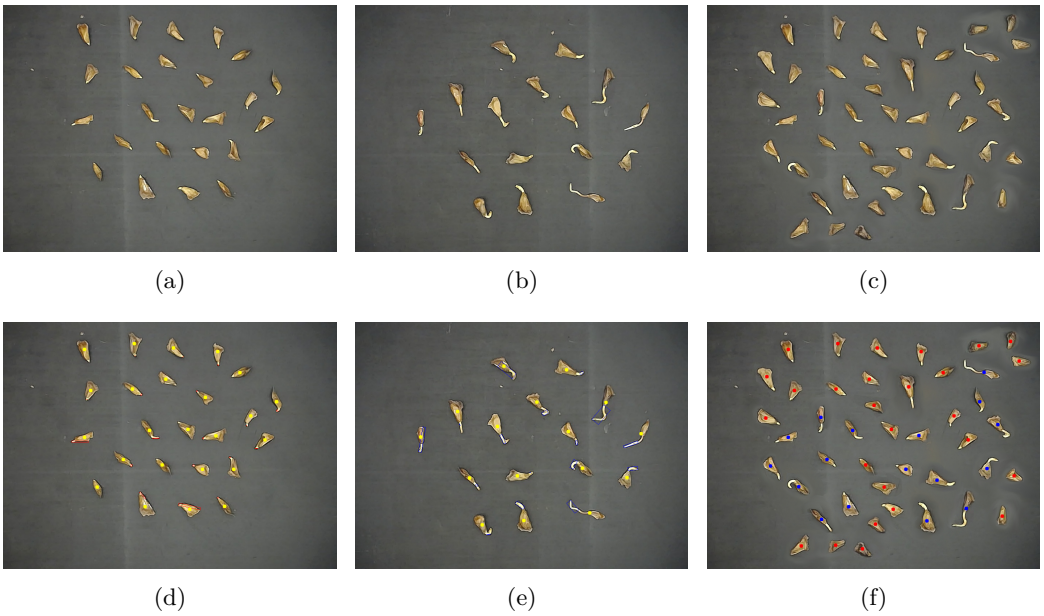


(b) RGB historam of the white sprouts

Figure 5.11: RGB histogram of (a) yellow-brown and (b) white sprouts.

5.4.3 2 class classification of real seed with Perceptron

The training data described in subsection 5.4.2 class1 and class-1 is labelled data. The class1 data contained sprouts that had length less than approximately 1 mm. The other training data class-1 contained sprouts longer than 1 mm. The unlabelled testing data, named class0 is a mix of the training data and other data. These data is shown in image (a), (b) and (c) in figure 5.12.



(a) (b) (c)

(d) (e) (f)

Figure 5.12: (a) Training data class 1 with sprout length < 1 mm. (b) Training data class -1 with sprout length > 1 mm. (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 5.13. Here it is shown, as predicted in section 5.3.2 that the data is not linear separable. By using the Perceptron an 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). The result in classifying the real seeds i shown in figure 5.12(f). This shown the principle of using the classifier, but a third classification region must be added to classify the seeds correctly. With only two classes the sprouts are either too small or too long. Therefore a third region is needed.

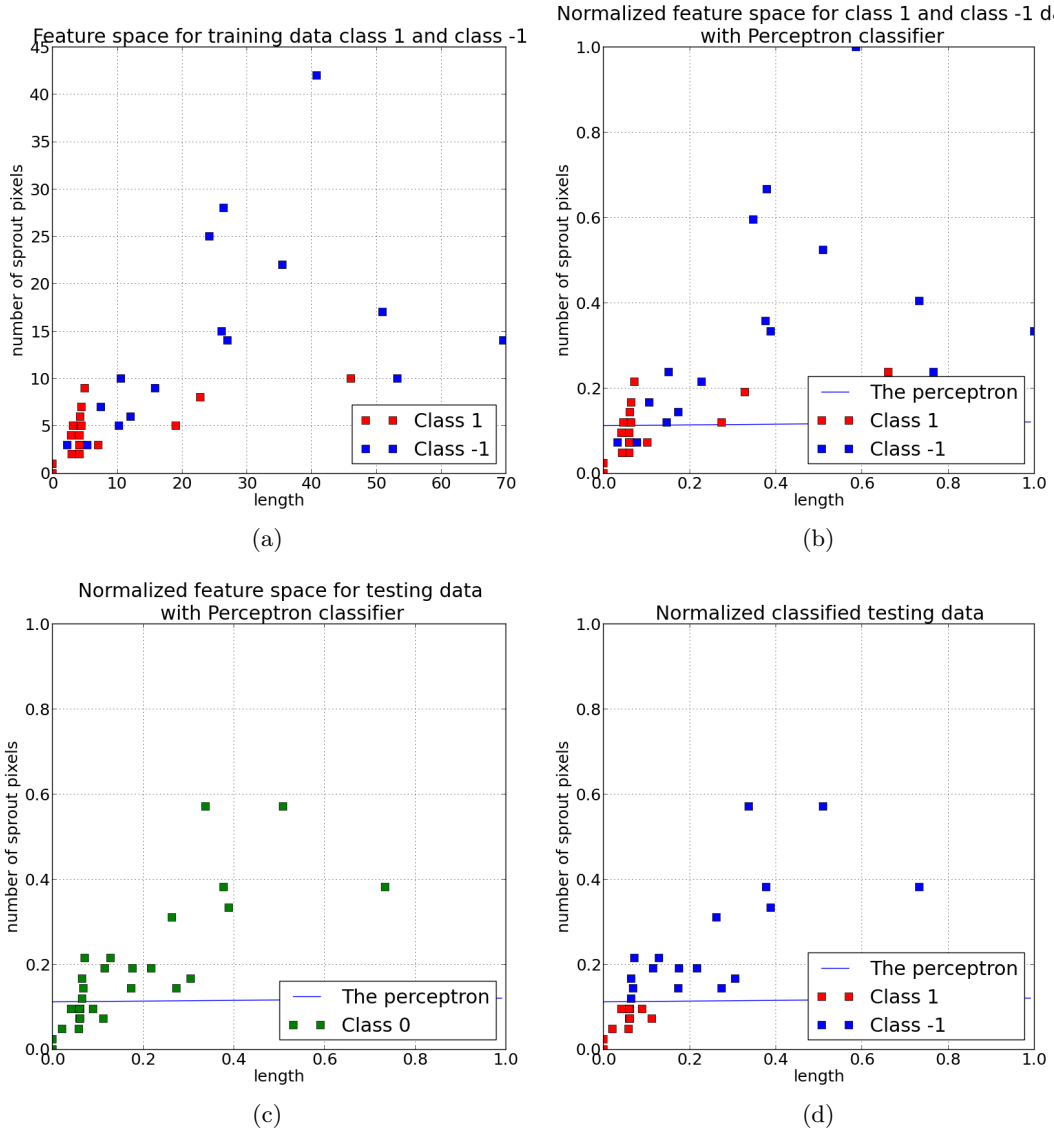


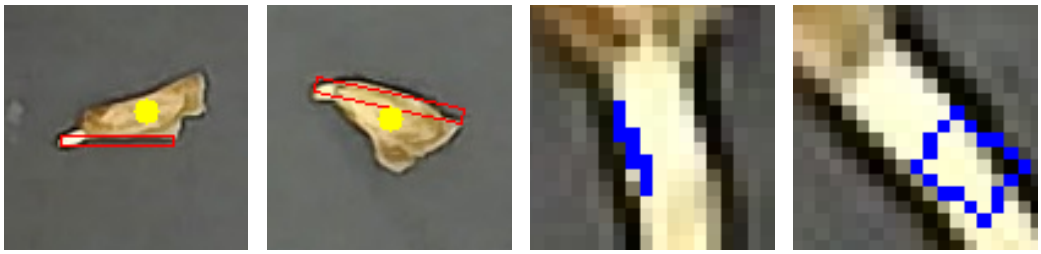
Figure 5.13: (a) Training data class 1 with OK sprout length. (b) Training data class -1 with too long sprout length. (c) Testing data. (d) Features extracted from (a) and (b)

As described in subsection 5.3.2, 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 [22] 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. By using the Sci-kit learn library, the SVM library was available, where a SVM was implemented using the training and testdata described in this section. The result is shown in the following video:
`https://www.youtube.com/watch?v=L7oyjrSUAxM`.

The video shows how the training data is used to create the two separation regions and how the testing data is classified. Some of the testing features points is jumping in values and hence gives different classification results for some seeds. This is due to the effect of the K-means algorithm that runs in the background. This is further described in subsection 6.3.3.

5.4.4 A new setup

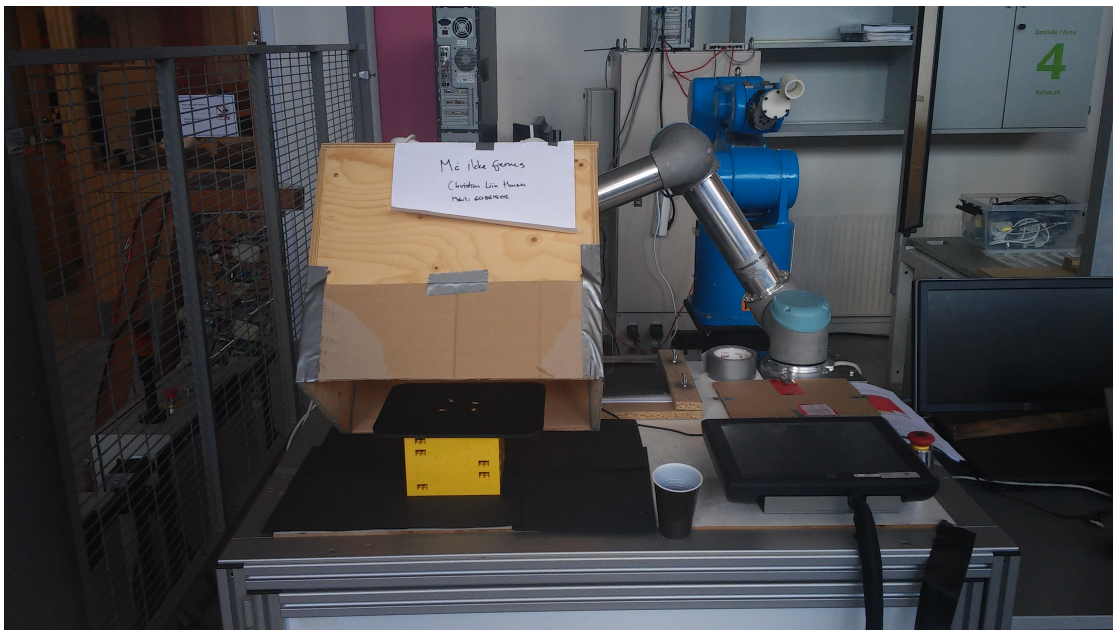
A interesting experience with real seeds, showed how the the segmentation and processing components needed optimization. Some seeds in figure 5.12 in image (d) and image (e) have false features, i.e. the bounding box, which the feature extraction is based on, was wrongly segmented. Zoom-in example of this is illustrated in figure 5.14. 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 with a K-means clustering algorithm and described further in subsection 6.3.3.



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

Figure 5.14: 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. The setup included an UR-6 robot, wooden box with camera mounted pendicular to a surface with seed on. This is shown in figure 5.15.

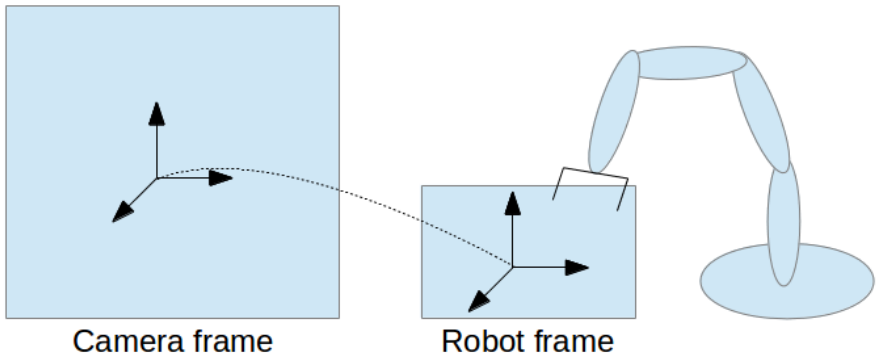


(a)

Figure 5.15: (a) Images of new setup in RoboLab at SDU. 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 on the conveyor belt. In order to calculate the position

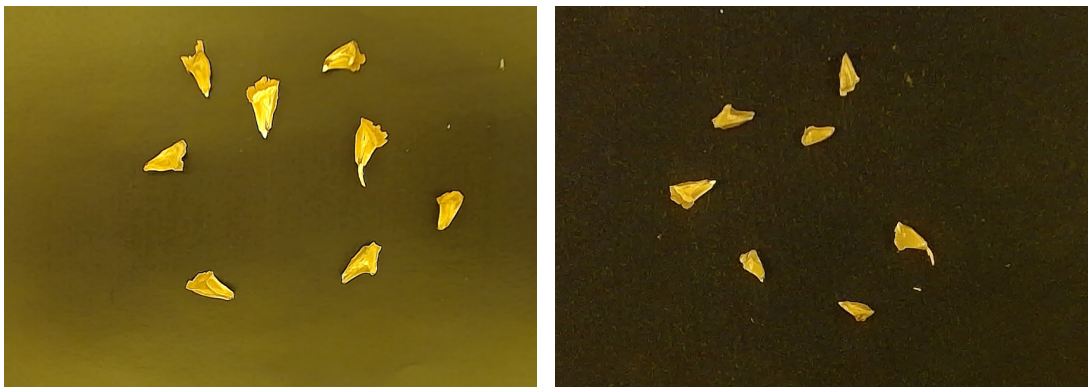
of the grasping location the speed of the conveyor belt must be known. Without any encoder this is not an easy task, since the speed estimation of the conveyor belt must be implemented as an extra vision component. By experience from previous course, it was a troublesome process to insure a robust speed estimation implementation. Additionally a buffer system must be implemented in the robot system in order to insure one center of mass per seed. In order to have a more simple setup with a robot, the conveyor belt was removed. In RoboLab the UR-6 robot was used previously to draw sketches on a stack of A4 papers. The idea was to let the robot draw the classification result for each seed, at center of mass location seen in the robot frame. I.e. the robot moves the end-effector where a pen i attached to the center of mass location seen in the robot frame. Then a check mark or "X"-mark is drawn for a good or bad seed respectively. The principle is illustrated in figure 5.16.



(a)

Figure 5.16: (a) Schematic illustration of a simplified setup. The vision system outputs a 3D location relative to the camera frame, where the robotic system interpreters the coordinate and draw the result on A4 paper relative in the robot frame.

To start out with the new setup, the camera was mounted in a wooden box, which is shown in figure 5.15. Using the wooden box in RoboLab, showed that the light conditions plays an important part. The light bulbs that were available were illuminating *warm light*, which have 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 figure in 5.17.

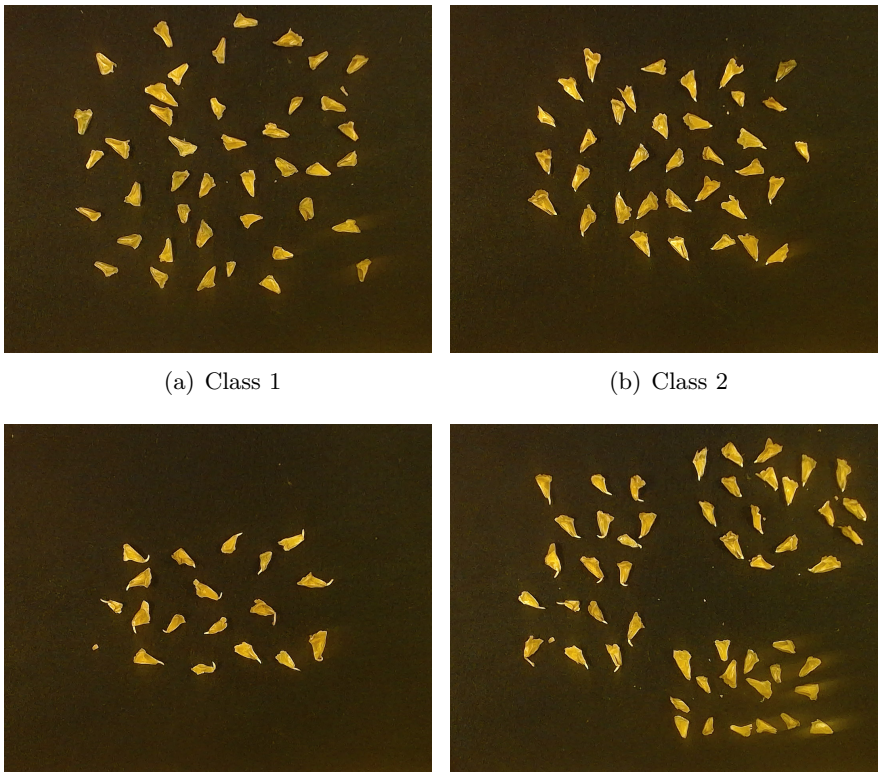


(a) (b)

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. The unlabelled testing data is noted as *class0* in the documentation, but is not a class.



(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 5.4.2. The three criteria is as followed:

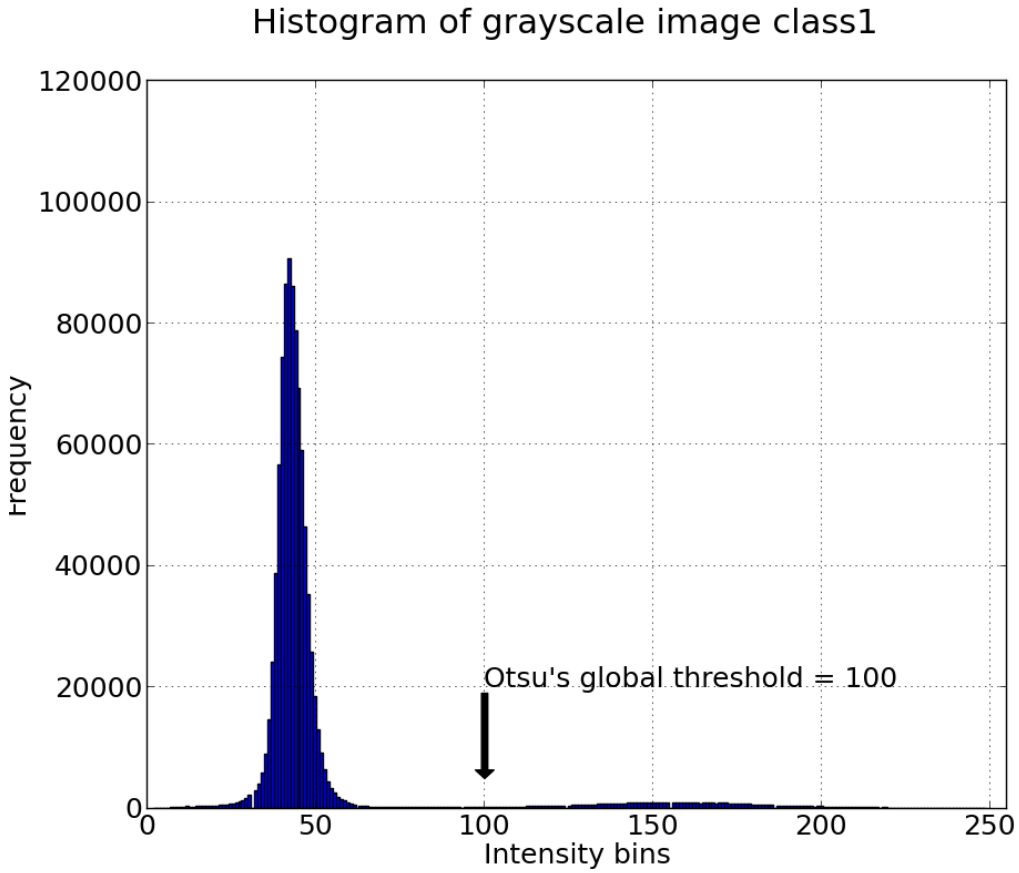
- If there is no sprout, the seed is bad
- If the length of the sprout is between 1-3 mm it is a good seed
- If the length of the sprout is longer than 3 mm, it is a bad seed

With the new training data, the preprocessing and segmentation component were adjusted and optimized. This is further described in subsection 5.4.5.

5.4.5 Optimizing the preprocesssing component

Previously the threshold value for the preprocessing component, described in section 6.2 was picked to 128 to separate between background and foreground pixels. In order to repair the structures, 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 [6], 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) Histogram of class 1 training data, indicating with the use of OTSU' algorithm from OpenCV, the optimal threshold value. (b) ROI of foreground image repaired with old tresholdvalue = 128 and three iterations of morphology which creates bridging when seeds are close. (c) ROI of foreground image with optimal threshold value and one iteration of morphology with prevents bridging

The preprocessing process gave sprouts that were not reproduced correctly, which lead to false features. A solution of repairing the sprouts was feasible. The step was to used morphology and the bitwise AND operation. The process is described as followed:

- Dilate the sprout image, $I_{dilateSprouts} = \text{dilate}(I_{sprouts})$
- Add the foreground with it self to get a temporary foreground image of white pixels, $I_{whiteForeground} = I_{grayForeground} + I_{grayForeground}$.
- Do at bitwise AND operation between the white foreground image and the dilated image, $I_{sproutRepaired} = I_{dilateSprouts}$ & $I_{whiteFroreground}$
- Add the repaired sprout image to the original foreground 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.

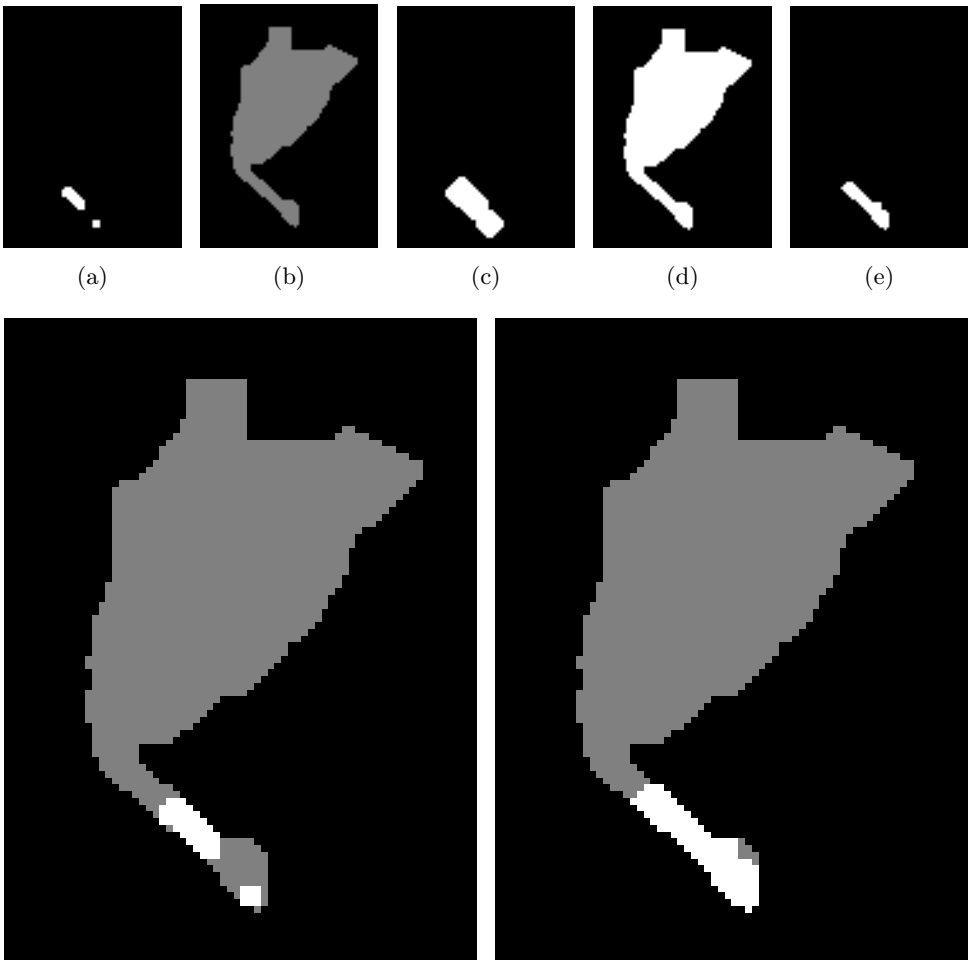
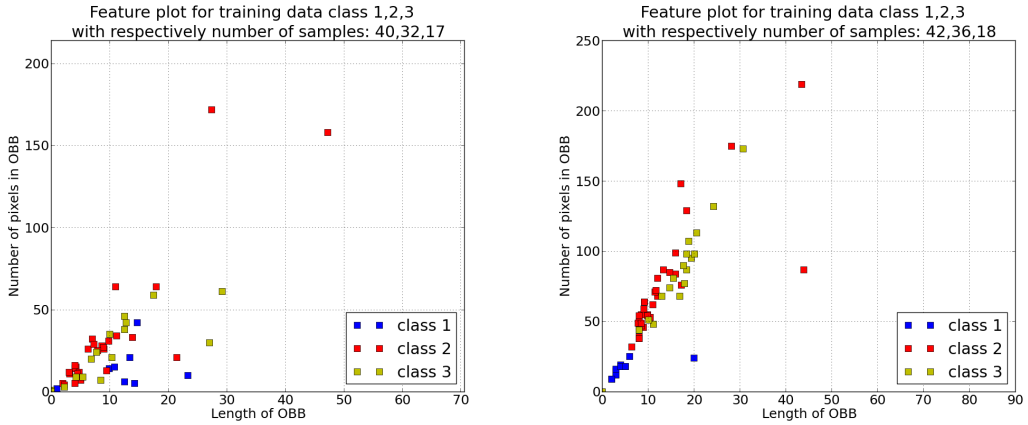


Figure 5.20: (a) Sprout before. (b) Foreground image. (c) Sprout dilated. (d) Added (b) twice. (e) Bitwise AND operation between (c) and (d). (f) Result before, with (a) and (b) added. Result after, with (e) and (b) added.

The result of optimizing the processing component gave more truthfully features, which is shown in figure 5.20. Comparing image (f) and (g) it is clear that image (g) tells more accurate how the spout is structured. This effects the feature plot which is shown in figure 5.21. Image (a) shows how the preprocessing output, before the implementation, gave bad feature extraction and hence the features were clustered randomly together. Image (b) shows the result in how the features is more spread out.

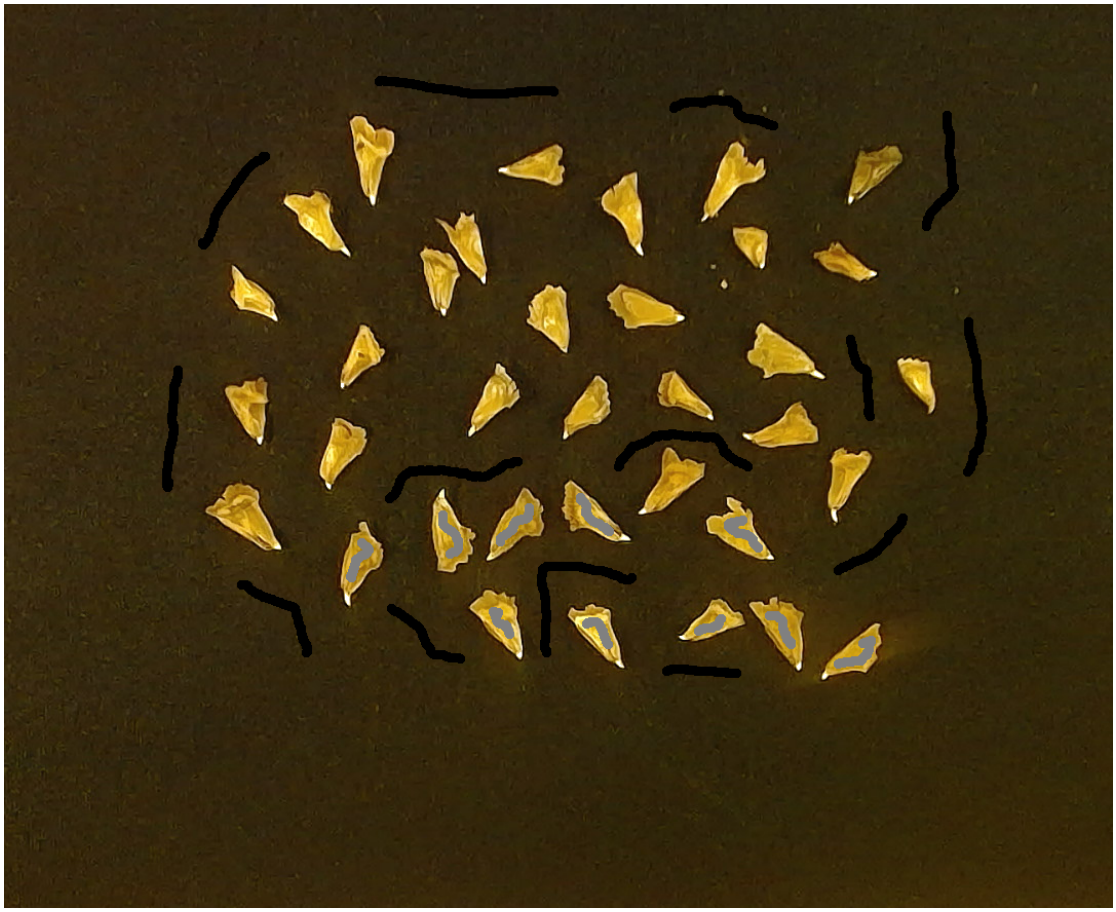


(a) (b)

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

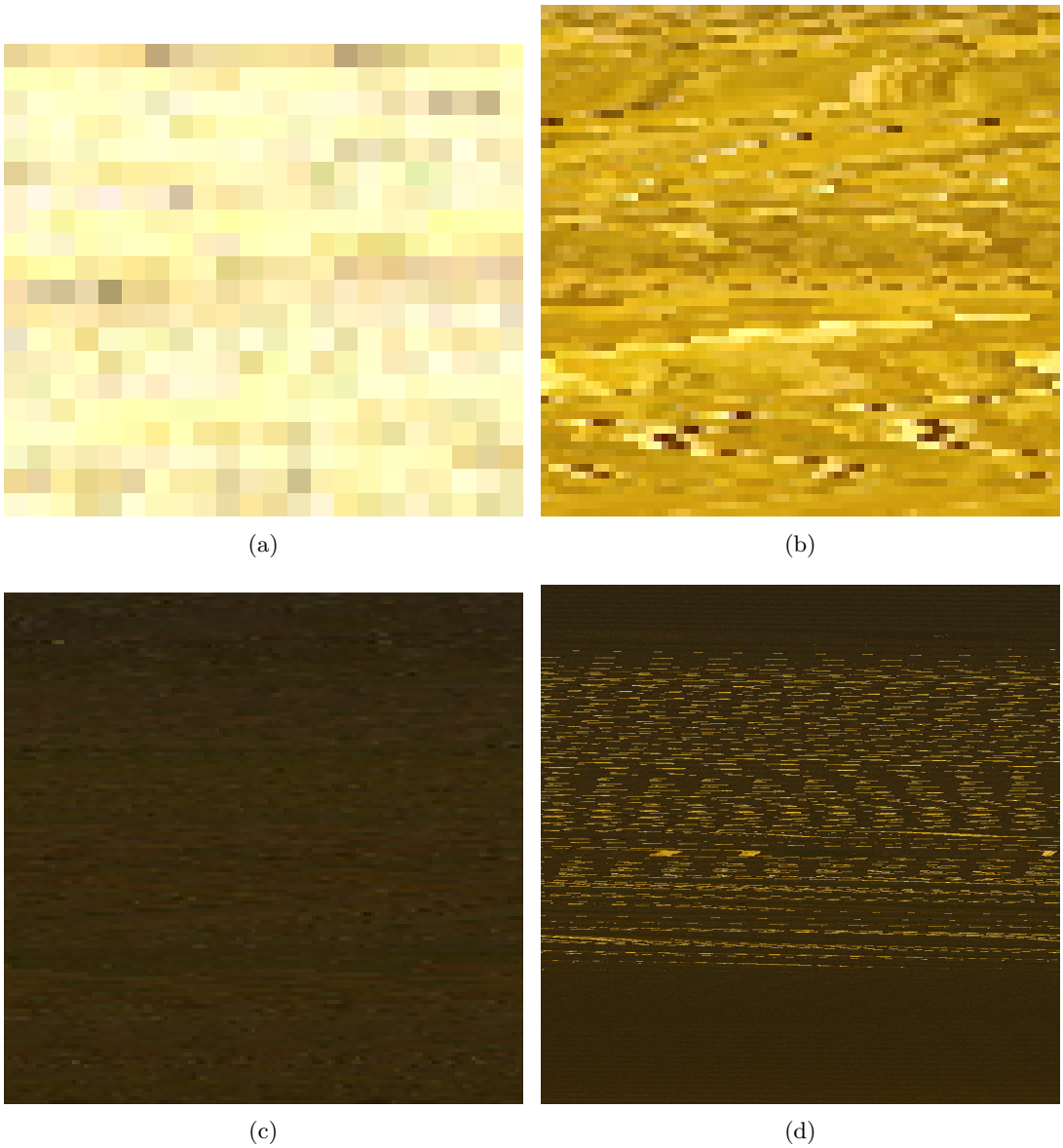
5.4.6 Semi-supervised learning

In this section an approach of using semi-supervised learning is explored. The idea was to use an alternative methods for generation the foreground, seed and sprout images. Semi-supervised learning is an AI methods that use some part of the data as labelled data and the rest as unlabelled data. This is often used where it is expensive or troublesome to use supervised learning. Perhaps the amount of data is too big to have all labelled data. This is especially interesting in order to generate a foreground, seed and sprout image, since each data is a pixel. Therefore it would be troublesome to label each pixel in a image as background, sprout or seed pixel. To start out, some labelled data is needed. This is generated in this case by drawing raise the intensity value on some of the sprout pixel to 255, 128 on some seed pixels and 0 on some background pixels. The result is shown in figure 5.22. All the rest of the pixel in the images are not manipulated and therefore is unlabelled data.
Then the methods was to extract all the pixels in the original image, which has a RGB value of (255, 255, 255) in the mask image. The extracted pixels were stacked in a sprout container. The same process was applied for the (128 128, 128) and (0,0,0) RGB values. The result was three container images with pure selected sprout pixel, seed pixels and background pixels respectively. The container images of the sprout, seed and background pixels is shown in image (a), (b) and (c) respectively in figure 5.23. Image (d) is the container image for the unlabelled data. The container images contains 440, 4692, 16640 and 928332 pixels for image (a), (b), (c) and (d) in figure 5.23 respectively. In order to see if there is any difference, the pixels where plotted in a RGB 3D plot. This is shown in figure 5.24. Based on the 3D plots, it would interesting to classify each pixels in the images using the labelled data, either with semi-supervised or supervised learning. However this was not implemented further in the project, but left as an idea for future work.



(a)

Figure 5.22: (a) Drawing white on sprout pixels, drawing gray on seed pixels and drawing black on background pixels.



(a) (b)

(c) (d)

Figure 5.23: Container images of the labelled data. (a) Sprout pixels. (b) Seed pixels. (c) Background pixel. (d) Unlabelled data pixels.

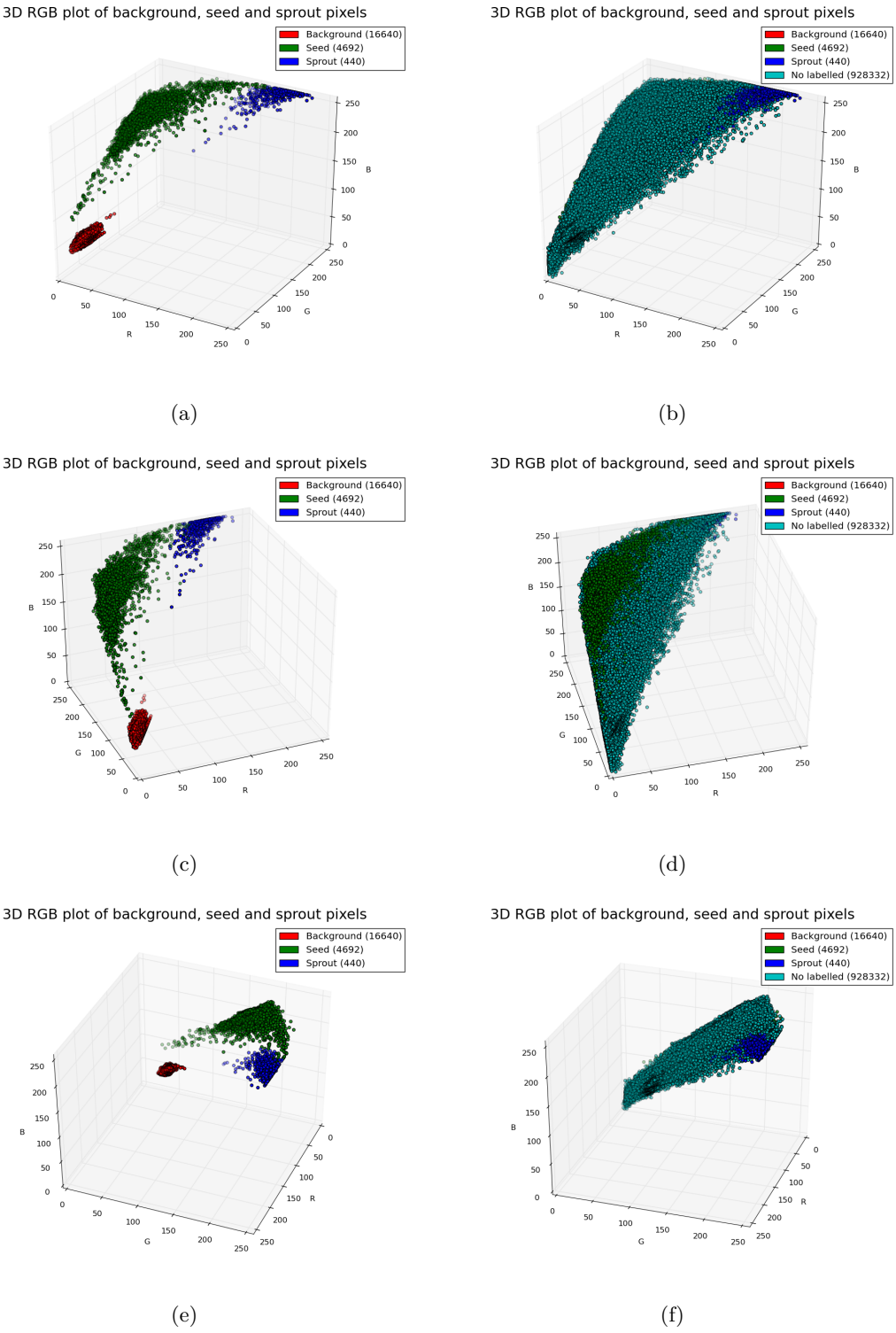


Figure 5.24: Container images of the labelled data. (a) Sprout pixels. (b) Seed pixels. (c) Background pixel. (d) Unlabelled data pixels.

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.

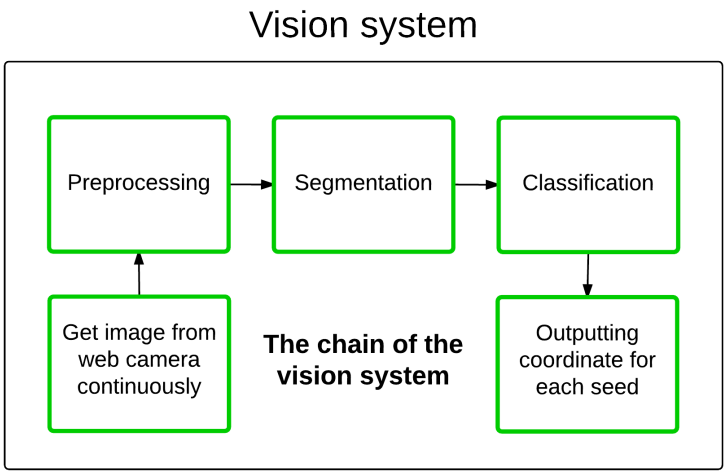


Figure 6.1: The chain of the project for the vision system

6.1 Get image from web camera

Argument for choosing the web camera, Logitech C930e is described in 5.4.1. 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 *foreground* 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.

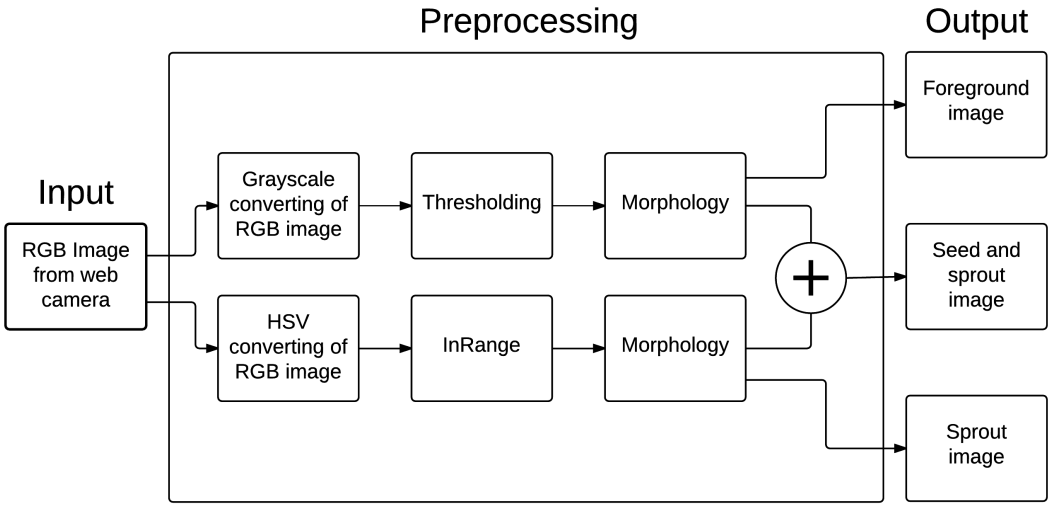


Figure 6.2: Preprocessing flow in order to generate the foreground, sprout and the combined seed and sprout image.

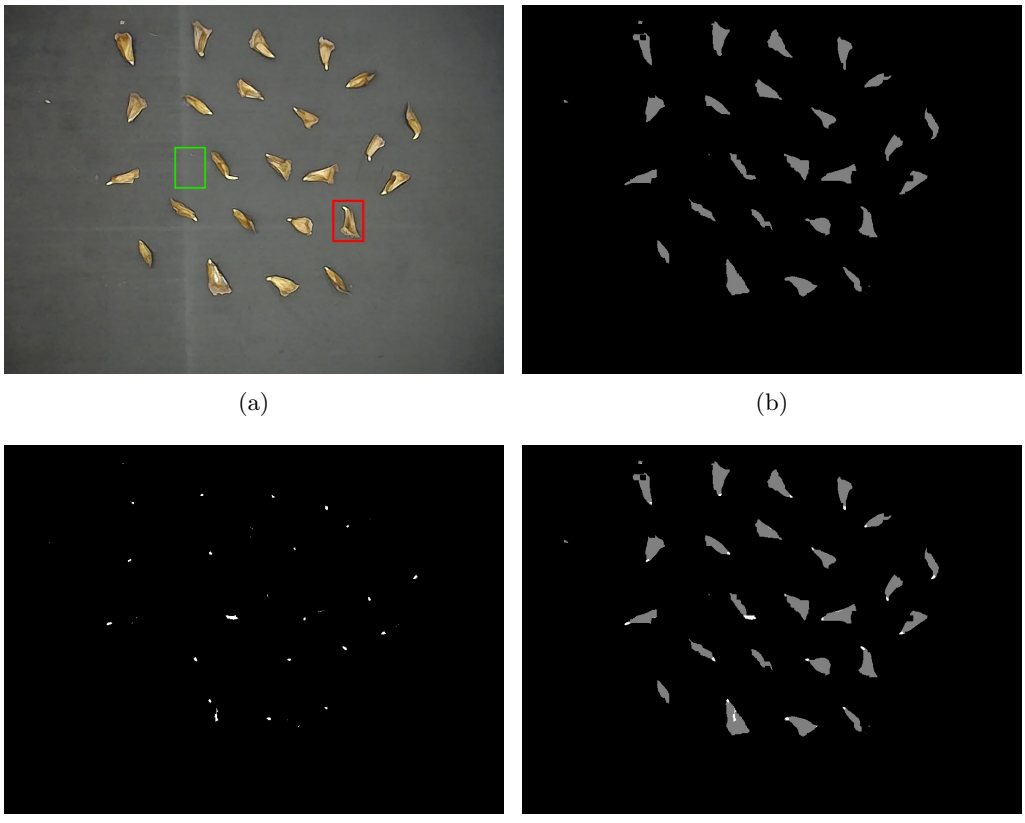
The foreground image is produced by converting the input image to grayscale, threshold it and finally apply the morphology process *closing* in order to repair the seed structures after the threshold. The conveyor belt, i.e. the background is relatively darker than the seeds and sprouts, which make a simple threshold sufficient. The threshold value was chosen to be 128. The result was acceptable, but morphology was needed to repair the structures. This value was change later in the project, using the OTSU optimal threshold value, which is described in section 5.4. A binary sprout image is produced by converting the RGB image to a *hue, saturation and value* (HSV) image and after use the inRange function from the OpenCV library.The inRange function checks if pixels in the HSV map lies within a defined interval. If true a white pixel is return, else a black pixel is returned. The interval is defined by a minimum and maximum value for the HSV parameters. The full hue range between 0-180 is used, the saturation range is 0-118 and value range is 155-255. These values is hardcoded and gave acceptable results. A future work is to use semi supervised learning to generate the sprout image. This is further described in section 10.3. Morphology is useed to repair the sprout structures. Finally the *Seed and sprout* image is produced by adding the sprout image with the ground image. The *Closing* morphology process is applied in order to close the gabs within object. This is performed by first dilate the image which expand the foreground 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 6.3, image (a) and (b) shows an example of a foreground image before and after the morphology respectively.
An example of a different images is shown in figure 6.4. A RGB input image is shown in figure image (a). 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 on the following page:

(a) (b)

Figure 6.3: (a) Foreground before morphology. (b) Foreground after morphology.

- Foreground image, image (b) 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, image (c) is a binary image, where the sprout pixels has intensity value of 255. The rest of the pixels are black.
- Seed and sprout image, image (d) is the combination of the Foreground image, image (b) and the sprout image (c). 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.



(a) (b)

(c) (d)

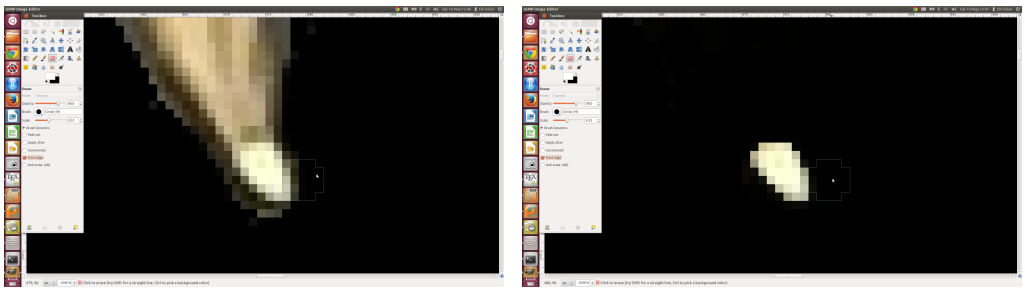
Figure 6.4: (a) Input RGB image. (b) Foreground image. (c) Sprout image. (d) Seed and sprout image.

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 carried out. 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 6.4(a). The test included the following images:

- ROI of seed with sprout, figure 6.6
- ROI of background, figure 6.7:
- Seed image, where non-seed pixels was set to 0, figure 6.8.
- Sprout image, where non-sprout pixels was set to 0, figure 6.9.

Figure 6.8 and figure 6.9 was generated by manual pixel selection in GIMP. The process is shown in figure 6.5. A 3D plot in figure 6.10 and figure 6.11 shows the color map of RGB and HSV respectively. The result heavily depends on the accuracy in the manual pixel selection. From the 3D plots non of the color mapping approaches can be claimed to perform better than the other. From literature [42], 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 [30]. Therefore the HSV method is the chosen approach in the preprocessing part.



(a) (b)

Figure 6.5: (a) Input RGB image with pixels represent seed and sprout. (b) Filtered RGB image with manual pixel selection using GIMP, with pixels represent sprout only.

Figure 6.6: Cropped image

Figure 6.7: Cropped background image

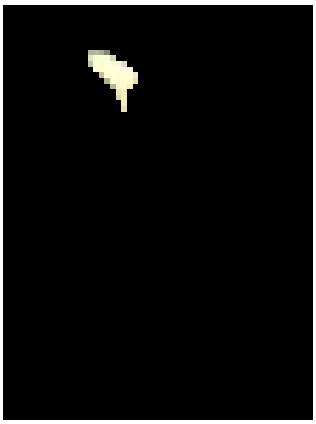


Figure 6.8: Sprout pixels only

Figure 6.9: Seed pixels only

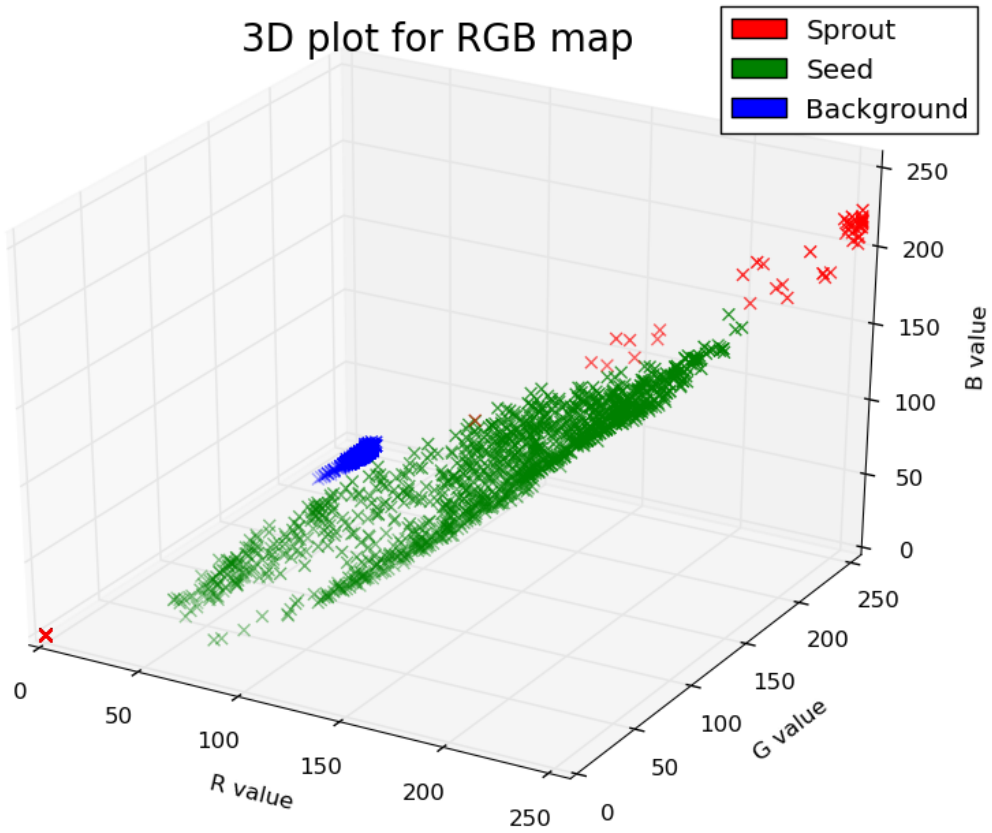


Figure 6.10: Preprocessing flow in order to generate the foreground, sprout and the combined seed and sprout image

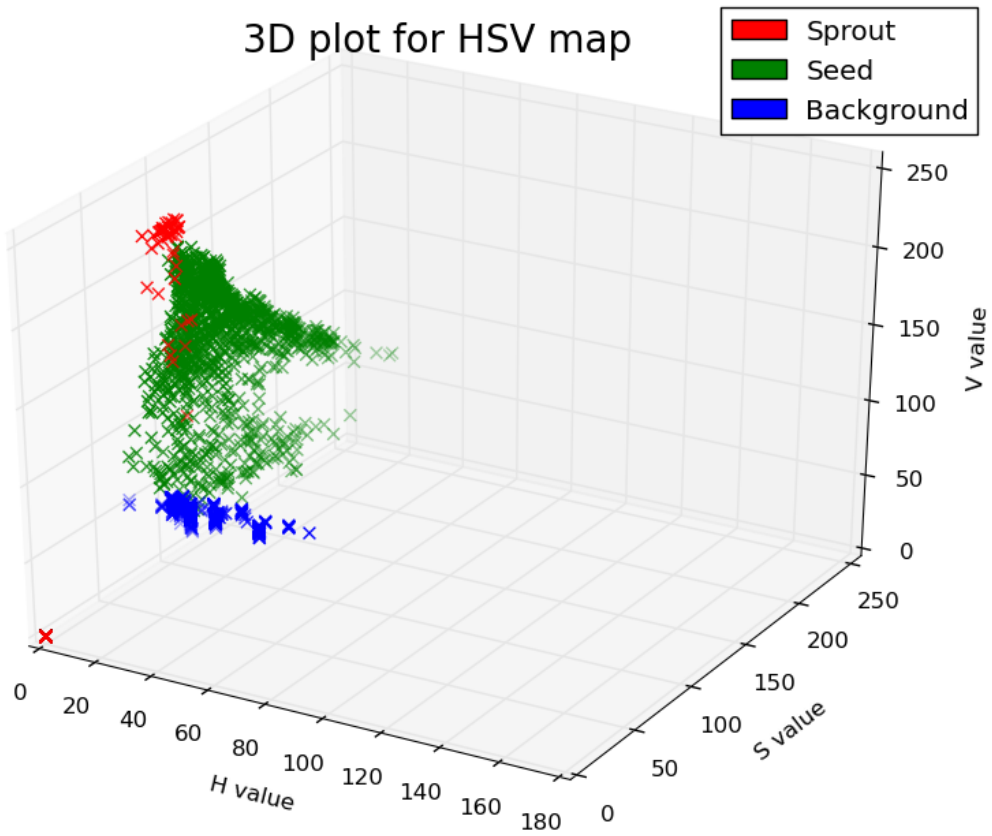


Figure 6.11: Preprocessing flow in order to generate the foreground, sprout and the combined seed and sprout image

6.3 Segmentation

The three output images *Foreground 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 6.12.

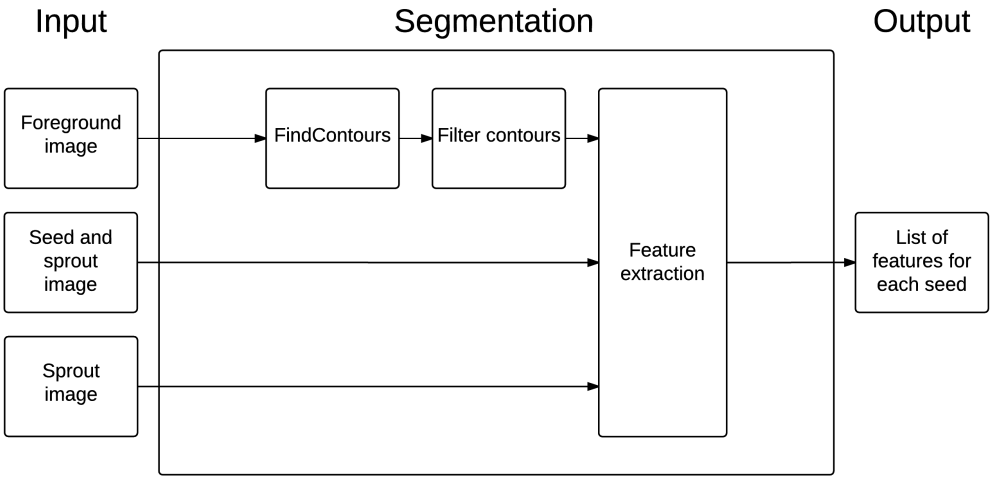


Figure 6.12: Segmentation flow in order to generate the list of features for each seed in the RGB image.

6.3.1 Filter contours

The foreground image is processed using the function FindContours from the OpenCV library. The function performs segmentation of a binary image and returns a list of all the contours found in the binary image, including leftover noise blobs, which were not removed by the morphology in the preprocessing component. A contour is a list of (u,v) coordinates of edge pixels in the image for any given shape that differs from the background. To filter out the noise contours, which could be e.g. an artefact or a piece of any material from the conveyor belt, a contour area interval with upper and lower threshold values is implemented. This is done by using the findContourArea function from the OpenCV library. In order to define the minimum and maximum threshold values for the contour areas, a testing image was analysed. This testing image has been produced and edited in Gimp (GNU Image Manipulator Program) in order to have a mix of seeds with different type of sprouts, e.g long, small or no sprouts. The test image is shown in figure 6.13. The cyan rings with green boundaries is drawn on top of the image to illustrated which contours that is outside the defined contour area interval. A blue and a red ring indicate the largest and smallest contour area respectively.
A histogram of the contour area for all contours in the testing image is shown in figure 6.14. From 0 to 157 on the x-axis, five noise contours is expected. The area of these is 2.0, 18.5, 0.0, 127.5, 29.5. From the reference [7] 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. The maximum contour, which is illustrated with a blue ring in figure 6.13 has an area of 1574 pixels. A minimum of 200 pixels seems reasonable for cutting off the noise contours in the images. The upper threshold is set to 2000 pixels. The minimum contour area within the interval is 332 pixels, which is illustrated with a red ring in figure 6.13.
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,

Figure 6.13: Test image with a mix of 44 seeds.

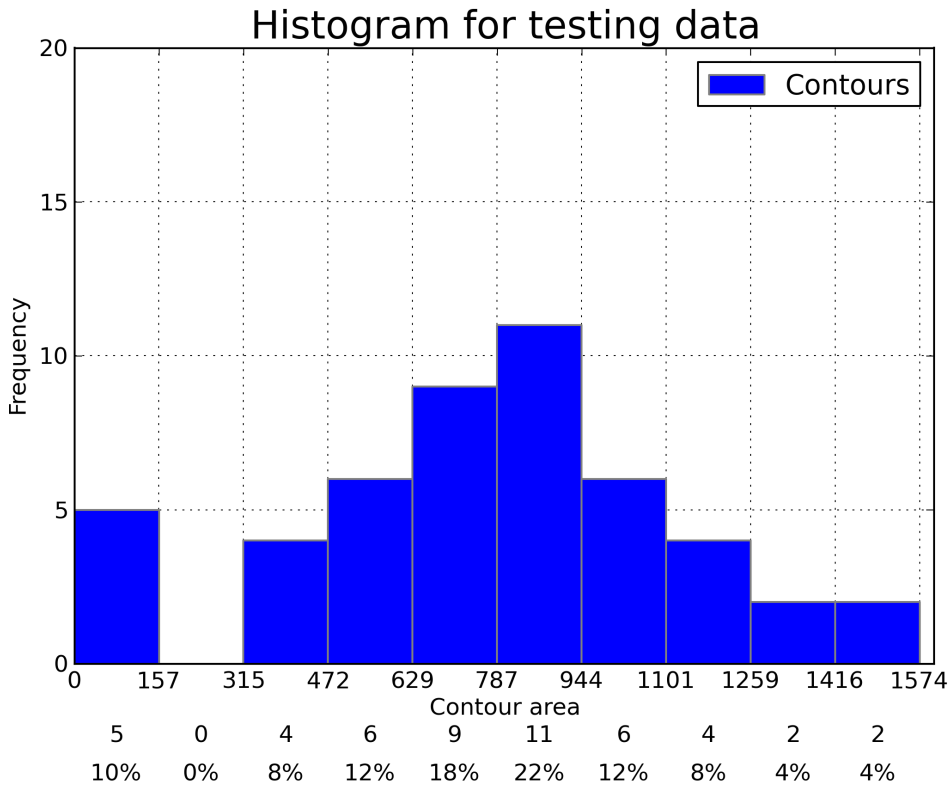
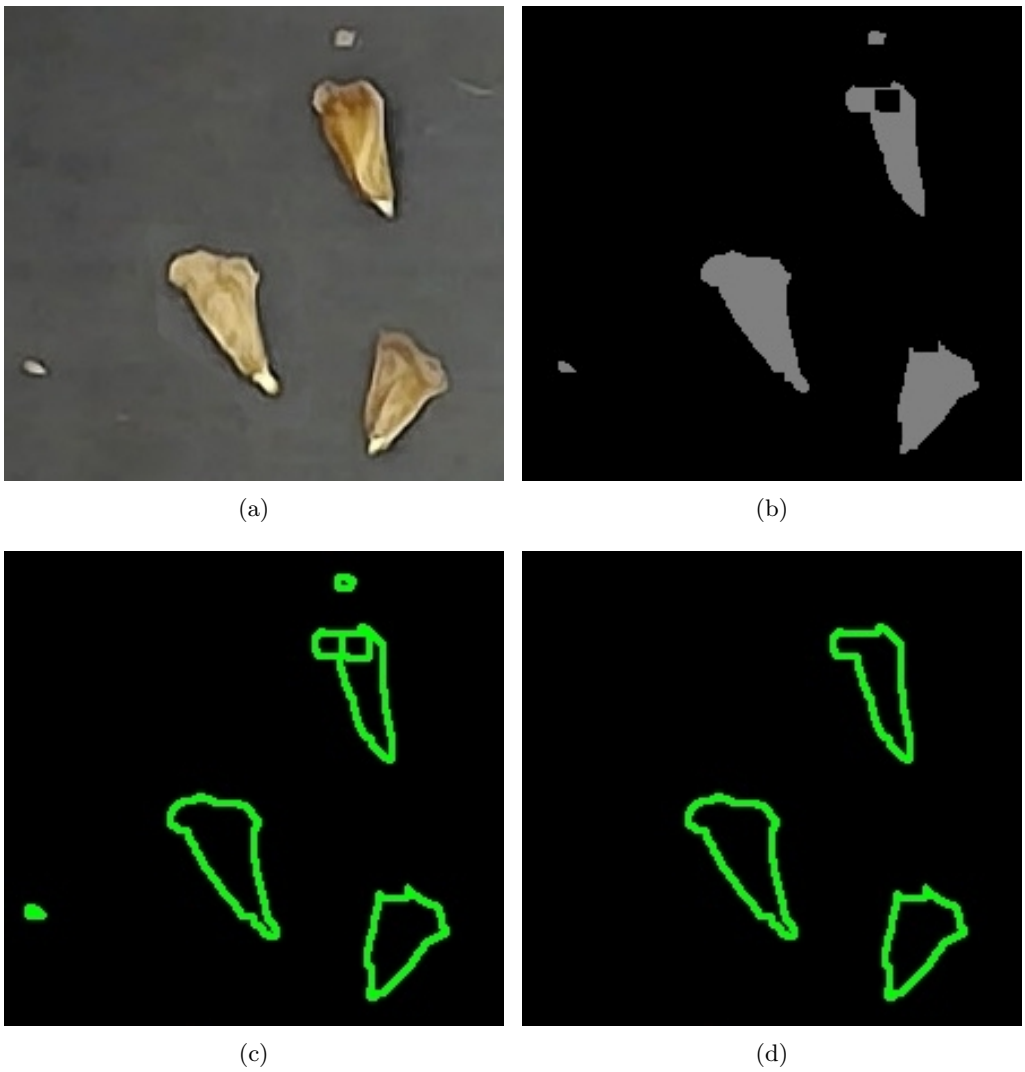


Figure 6.14: Histogram of contour area in the test image with 44 seeds and 5 noise blobs

a ROI was cropped out from the input images in figure 5.12(c). The test included the following images:

- ROI of input image, figure 6.15(a)
- ROI of foreground image after the morphology process, figure 6.15(b)
- ROI of drawn contours before filtering, figure 6.15(c)
- ROI of drawn contours after filtering, 6.15(d)

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 is the camera mounted closer to the conveyor belt, this maximum threshold needs to be adjusted, since the contour areas will increase. If two seeds is touching or overlapping each other, the area will likely be greater than the upper threshold limit, and the two seed will be ignored by the vision system. This topic is further discussed in section 10.2.



(a) (b)

(c) (d)

Figure 6.15: 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.

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 assumed to be a sprout pixel and if the pixel is gray, it is assumed to be a seed pixel. After the list of seed and sprout is generated for each seed an oriented bounding box is fitted around the sprout pixels. The number of white pixels, length, width and center of mass coordinate is extracted using the minRectArea from OpenCV. The ratio width/length is calculated, and together with hue values a list of features is generated out from each image. The primary feature is the length of the bounding box. The secondary feature is the number of white pixels, in order to verify the bounding box. 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 6.16(a).
  - The length of the sprout is longer than 3 mm, figure 6.16(b).
  - The sprout is curved or twisted, figure 6.16(c).
  - The color of the sprout is yellow or brownish, figure 6.16(d).
- An object is categorized as good if:
  - The length of the sprout is between 1 to 3 mm, figure 6.16(e)
  - The color of the sprout is white, figure 6.16(f)

If the condition which categorize a seed as good is present while a condition which categorize the seed as bad, the seed is categorized as bad. E.g. if the sprout of an object is white, but longer than 3 mm, then the object is bad.
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 6.17(a), a ROI has been cropped out for better visualization. In figure 6.17(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 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

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

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

Figure 6.16: Showing different conditions of the sprouts

(a) (b)

Figure 6.17: Image (a) Cropped out object from RGB input image. Image (b) A red OBB drawn around the sprout pixels

of the function *minRectArea*. However calculating the center of mass is available through the use of moments [38]. To see the difference between the two methods, a comparison is shown in figure 6.18. 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, though the green dots seems more accurate. It all comes down to the type of grasping, which in this project will be performed by a pneumatic suction end-effector. At the moment the methods of using moments is used, since this was first implemented in the proof of concept described in subsection 5.3.1, before using the *minRectArea* function. Argument for changing method with COM from *minRectArea* would be to save computational time. This is reserved for future work.

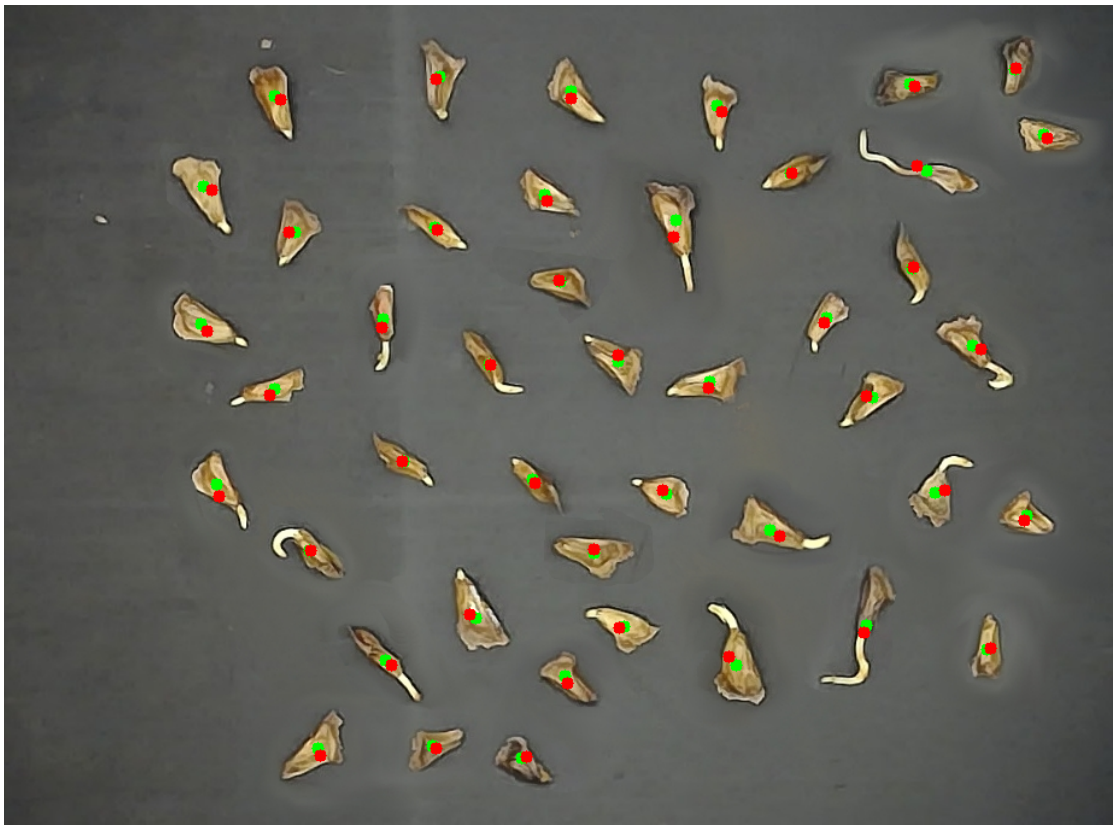


Figure 6.18: Red dots indicate COM using *minRectArea*. Green dots indicate COM using moments

6.3.3 Clustering

As described in section 6.3.2 the feature extraction is based on the OBB that span the sprout pixels for each contour. The data from the feature extraction is later used in the classification module in section 6.4. In order to maximize the classification rate, it is important to minimize the false negatives and false positives. The false positives is most important to minimize, since the impact of planting a bad seed in the ground is worse than discarding a good seed from the planting process.

To measure the false negatives and false positives two tests were carried out, which is described in as follows:

False negatives test

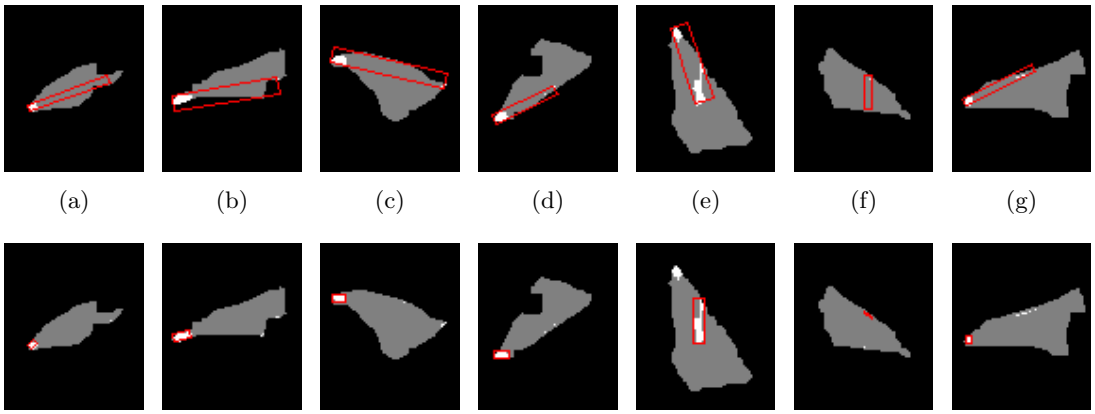
The first test contains a RGB image with good labelled seeds. These are shown image (a) in figure 6.19. The dimensions of the image is 920 x 680 pixels. Doing the test, image (a) was preprocessed. The result is shown in image (b), where red bounding boxes is drawn to indicate the OBBs of the sprouts.
By observing the dimensions of the bounding boxes, sometimes an extracted bounding box is spanning white pixel, that do not belong to real sprouts in the image. The result of the first test shows 24 seeds with 7 false negatives. The type I error rate is $\frac{7}{24} = 29.16\%$. The seven type I error examples is cropped out and is shown from image (a) to (g) in figure 6.20.

In order to deal with the false negative bounding boxes a K-means cluster algorithm was implemented in order to better find the cluster of white pixel that belongs to true sprout pixels. The output of the K-means cluster algorithm is two lists. One list with "true" and one list with "false" sprout pixels. The system dont know the ground truth.

(a) (b)

Figure 6.19: (a) RGB image with good seeds. (b) Result of image (a) that went through the preprocessing and segmentation component.

Therefore it is assumed that the majority of the white pixels comes from the true sprout area. A size check on the output lists is performed and the list with most white pixels is selected to be the list with the "true" sprout pixels. The result of the K-means cluster algorithm is from image (h) to (n) in figure 6.20



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

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

Figure 6.20: (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

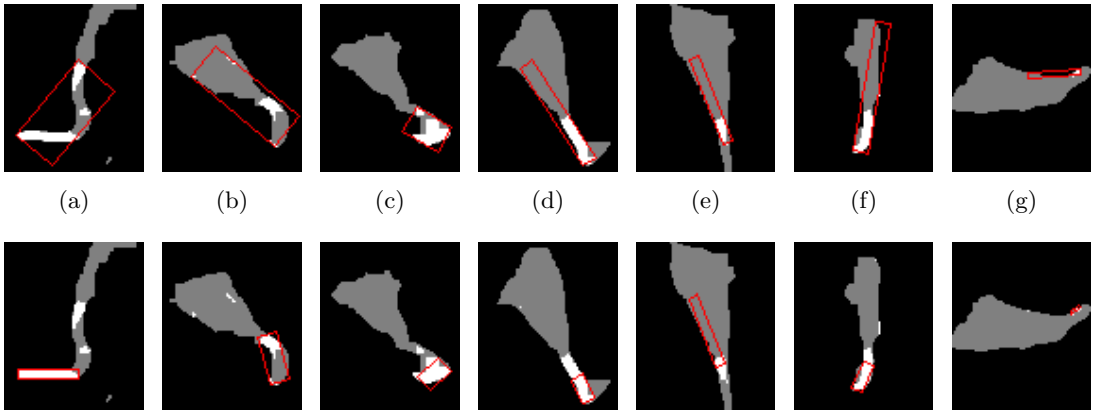
Overall implementing the K-means cluster algorithm helped to reduce false negatives, which is shown in image (h) to (n) in figure 6.20. The type I error has been reduced to down to $\frac{2}{24} = 8.33\%$. However the assumption of having two lists after using the K-means cluster algorithm and use the list with most pixels is not always the truth. Image 6.20(l), which is still a false negatives, shows the result of assuming that the biggest cluster contains true sprout pixels. Image 6.20(m) is also false negatives, but the amount of white sprout pixels were limited and hence the result which is shown in figure 6.20(f). If the seed do not have any sprout, no bounding box is fitted and hence all the features for the given seeds is zero, expect the center of mass.

False positives test

A second test was carried out in order to evaluate the effectiveness of the K-means cluster algorithm on bad labelled data, where the sprouts are too long. The result is shown in figure 6.22. This shows that the cluster algorithm depends on the result of the

(a) (b)

Figure 6.21: (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 6.22: (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

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. Repairing the sprouts is an optimization task and described in subsection 5.4.5. Scenario where two sprouts are close or touching each each other is described in subsection 5.4.5 and further in section 10.1 and section 10.2.

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

]

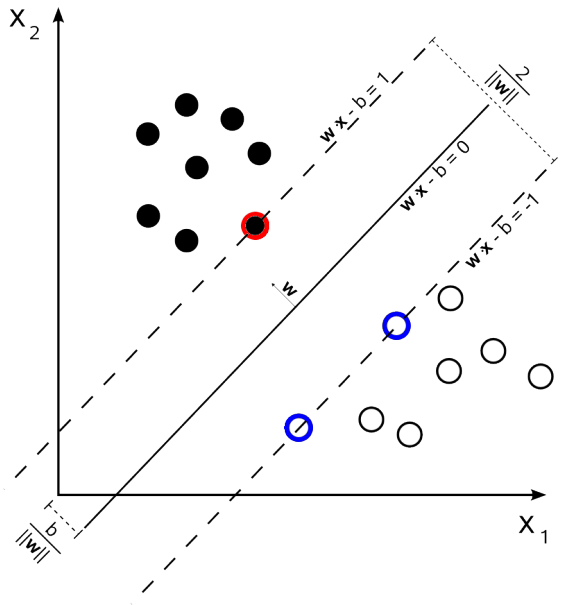
6.4 Classification

In the beginning of the project, the Perceptron classifier was implemented from scratch in order to learn how a classifier works. After gaining experience with classifiers, a more sophisticated classifier was needed. This was chosen to be a Support Vector Machine (SVM) classifier. The arguments are described in subsection 5.4.3. The classifier was integrated by using the off-the-shelf sklearn library from scikit-learn [40]. The SVM builds on the principle of Perceptron, but the math behind is more complex [22]. The fundamental idea behind SVM is to minimize the equation 6.1 under the constrains of equation 6.2 in order to maximize the margin for the separation line between two classes.

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_n \tag{6.1}$$

$$y_i(\mathbf{w}^T\Phi(x_i) + b \geq 1 - \xi_n \tag{6.2}$$

The x_i, y_i is the feature input, $\mathbf{w}$ is the weight vector, b is the classifier offset, $\Phi(x_i)$ is the kernel functions, C is the regularization parameter and the ξ is the slack variable. The main difference between the Perceptron and SVM is that the the Perceptron finds the first possible solution that exists, which is not margin optimized. The SVM finds the optimal margin, if the parameters is optimal, by maximize the margin. Additionally the SVM can be implemented as a soft-margin classifier by having the $C\sum_{i=1}^{N}\xi_n$ therm which defines the allowance for violations in the margin. The larger the C parameters the more influence will the margin violation have and the classifier tends to overfit. For small values of C, the violations in the margin has less influence and the classification regions will be more smooth, since the allowance of violations is higher [12].
In figure 6.23 from [11] shows a feature plot in 2D. The separating hyperplane in this case is a separating line which is drawn between the white and black samples. The margin between support vectors for the SVM is defined to be maximized when the therm $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ is minimized. The support vectors are coloured red and blue for the black and white data points respectively.



(a)

Figure 6.23: (a) Two classes are separated using a SVM. The margin is maximized regards to the support vectors, which is illustrated with red and blue color for the black and white datapoints.

6.4.1 Kernels

Image (a), (b) and (c) from figure 5.18 in subsection 5.4.4 were preprocessed and feature extracted, which generates two feature vectors, v_x and v_y. The selected features were original the length and number of pixels in the oriented bounding box. The features were normalized in order to have similar range between 0 and 1. From the Sci-kit-learn library different kernels were available [40]. The choice were between linear, polynomial, sigmoid, radial basis function (RBF) and customized kernels. In [22] a SVM with RBF kernel was used for classification of seedling samples. The linear and RBF kernel were therefore investigated. Using the linear kernel, the result is shown in figure 6.25, where image (a) shows the training data and the learned classification regions. The result in figure image (a) shows that the different clusters were overlapping, which makes the classification more difficult. Image (b) shows the classified testing data. Image (c) shows the final RGB input, where each seed is classified and marked with a color that represent the classification region. The classification rate is $1 - \frac{18}{51} = 64.71\%$ with the regulation parameter $C = 100$. This parameter was chosen empirically. 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.

The RBF kernel was tried out as well in order to see if this could increase the classification rate. From the online video lecture of CalTech [9] the RBF is defined in equation 6.3, where the basis is shown in equation 6.4. The radial therm $\|x - x_i\|^2$ is the squared Euclidean distance between two data points, hence the name for the kernel.

$$y(x) = \sum_{i=1}^{N} w_i K(x, x') \tag{6.3}$$

$$K(x, x') = \exp(-\gamma \|x - x_i\|^2) \tag{6.4}$$

From [8] the RBF kernel can be expressed as a Gaussian distribution with equation 6.4 using the γ variable, which is defined in equation 6.5.

$$K(x, x') = \exp(-\frac{\|x - x_i\|^2}{2\sigma^2}) \text{ where } \gamma = \frac{1}{2\sigma^2} \tag{6.5}$$

The idea behind the RBF kernel is from each center of each data point a Gaussian distribution explains the influence of the given data point to the neighbour points in the data set. The influence drops exponential the further away neighbour points are. The classification region is therefore based on the sum of all Gaussian distributions in the training data. The γ parameter, which is multiplied with the radial therm, determine how much influence far away point will have compared to nearby data points, which change the classification regions. From equation 6.5 the γ is inverse of the standard deviations of a Gaussian distribution. Therefore small value of γ gives larger distributions and larger value makes the Gaussian slimmer.

6.4.2 Result of SVM

Testing different γ parameters has been tested with older data, which is shown in appendix G. This shows that with a higher γ value the more over fitting occurs. Testing with the RBF, the parameters was chosen to be $C = 100$ and $\gamma = 0.8$ after evaluating the test, which is described in appendix H. Using the RBF kernel, the result is shown in figure 6.26. The classification rate is $1 - \frac{18}{51} = 64.71\%$.
A grid search approach would propperly be more suitable in order to find the optimal parameters [3]. However it was decided to test the other features, which is described in section 6.3.2 in order to see if the features cluster means where more significant difference. Each feature was indexed as followed:

- Feature 1: Length of bounding box.
- Feature 2: Width of bounding box.
- Feature 3: $\frac{width}{length}$ of bounding box.
- Feature 4: Number of sprout pixels within the bounding box.

The rest of the features described in 6.3.2, i.e. mean and standard deviation of the hue value for the sprout pixels where not included in the pairwise test. Reason for this, which is described in subsection 5.4.2, is that using a webcamera do not produce trustful hue features. Therefore feature 1 to 4 was pair-wise tested. In the following documentation, a featureX vs featureY test is noted as "XvsY" test. I.e the following test was performed: 1vs2, 1vs3, 1vs4, 2vs3, 2vs4 and finally 3vs4. The result of the pair-wise test, shown in the appendix I gave features that were clustered together, expect for the 3vs4 test. The new features were tested with a linear and RBF kernel again, which the result is shown in figure 6.27 and figure 6.28 respectively. The new features were more separable and hence produced less misclassification. To summarize the following test was carried out: 1vs4 with linear kernel, 1vs4 with RBF kernel, 3vs4 with linear kernel and 3vs4 with RBF kernel. These test is shown in as table (a), (b), (c) and (d) in figure 6.24 respectively.

		Predicted class	
		Positive	Negative
Actual class	Positive	13	4
	Negative	14	20

(a)

		Predicted class	
		Positive	Negative
Actual class	Positive	12	5
	Negative	13	21

(b)

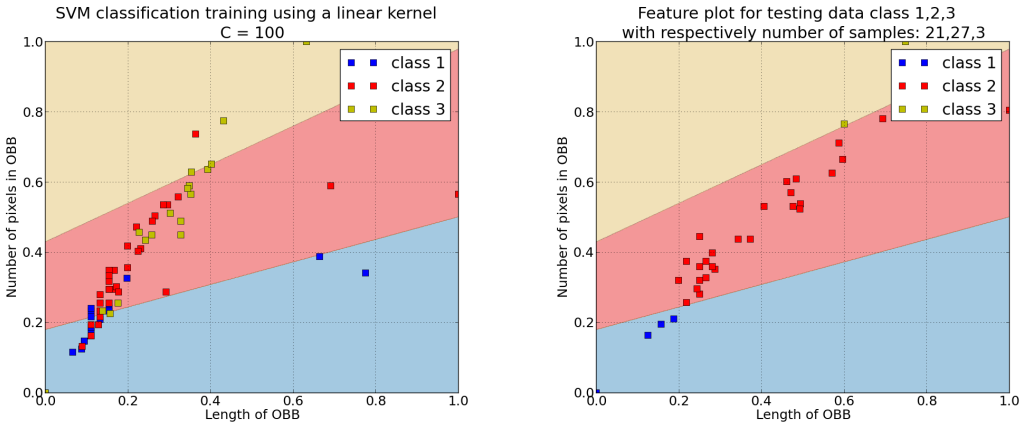
		Predicted class	
		Positive	Negative
Actual class	Positive	12	5
	Negative	1	33

(c)

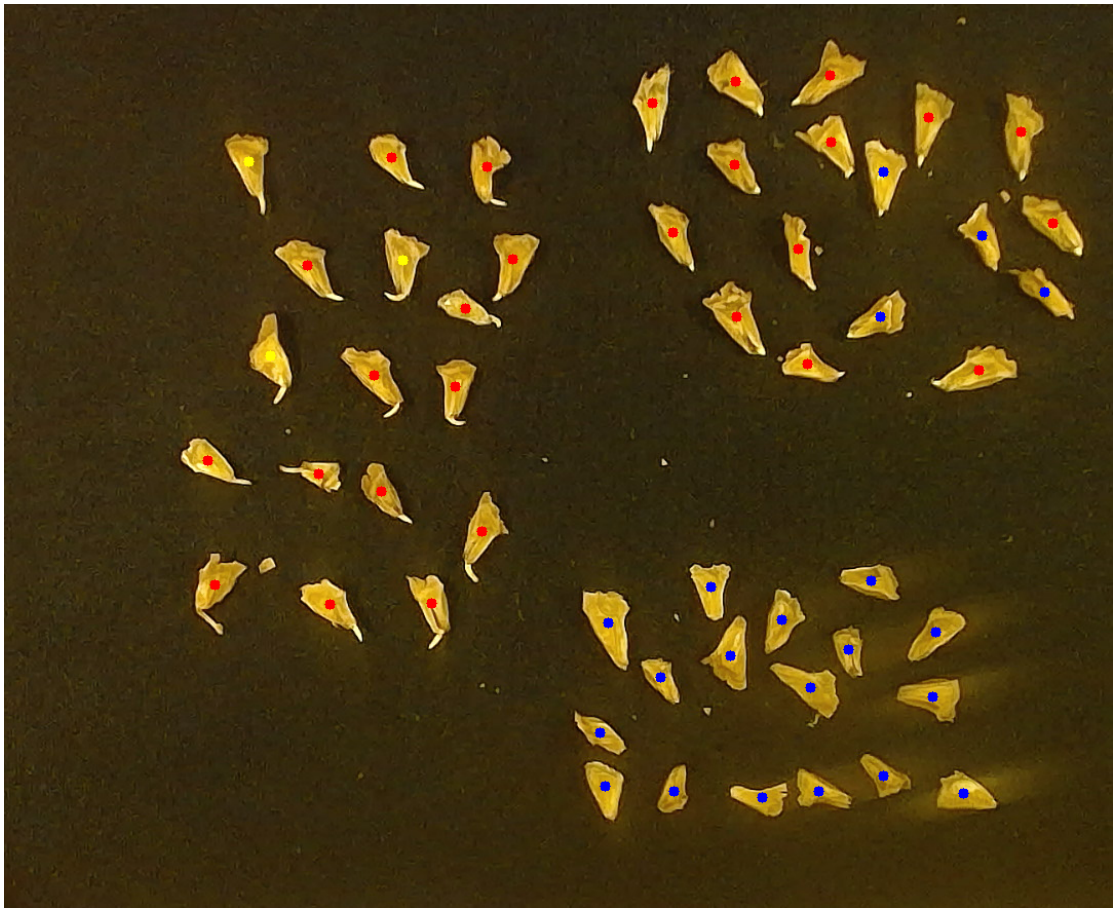
		Predicted class	
		Positive	Negative
Actual class	Positive	13	4
	Negative	3	31

(d)

Figure 6.24: Confusion matrices for following feature tests. (a) 1vs4 with linear kernel. (b) 1vs4 with RBF kernel. (c) 3vs4 with linear kernel. (d) 3vs4 with RBF kernel.

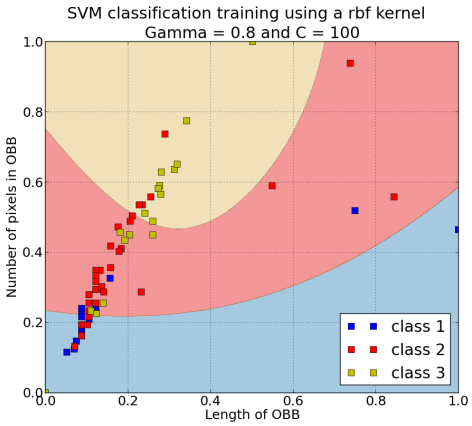


(a) (b)

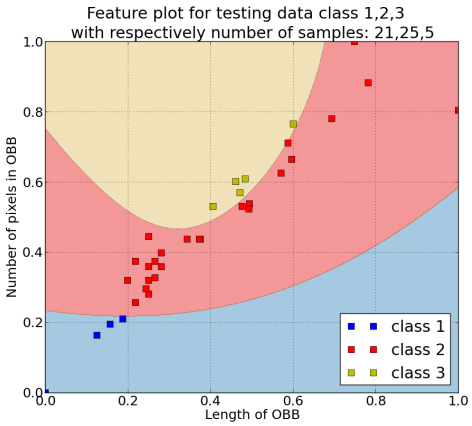


(c)

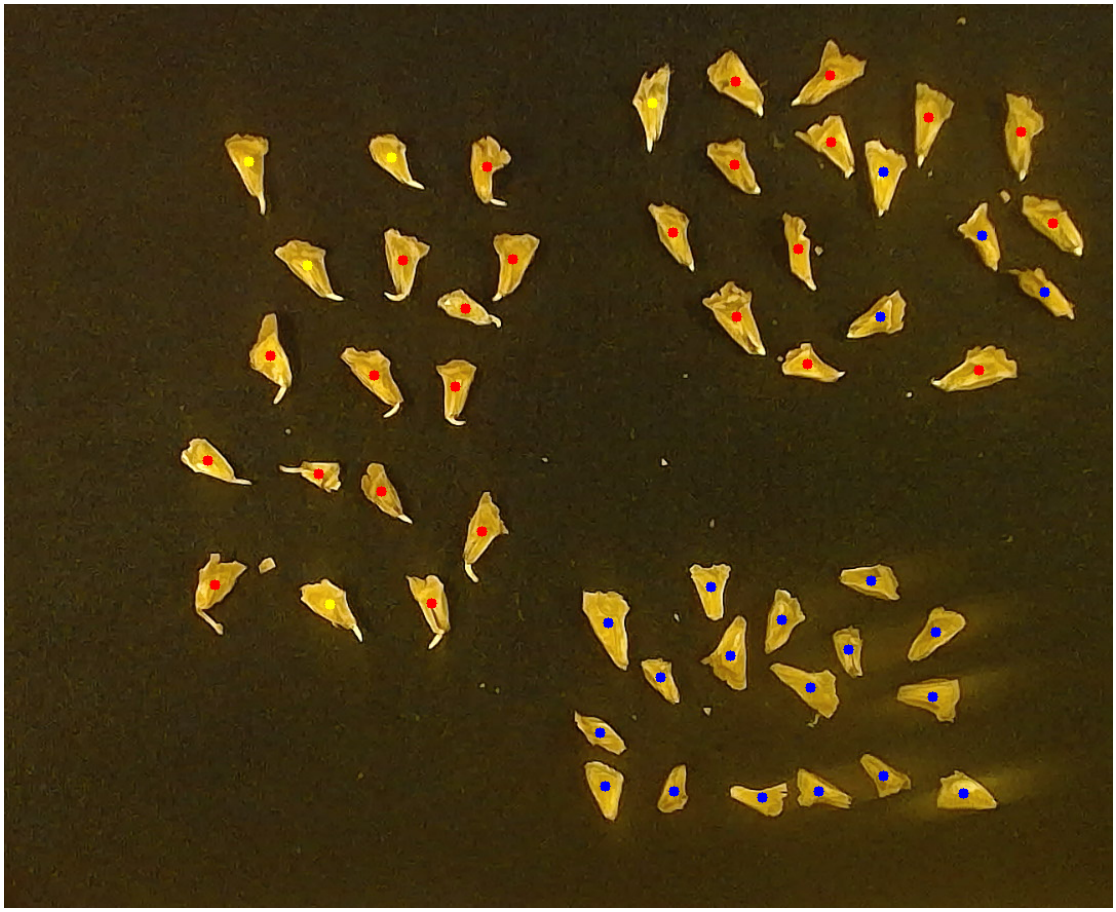
Figure 6.25: (a) Training data with learned classification regions. (b) Testing data classified due to classification regions. (c) RGB image with classified seed where each seed is marked at their center of mass with a respectively class color. Blue is class1, red is class2 and yellow is class3.



(a) $\gamma = 0.8, C = 10$

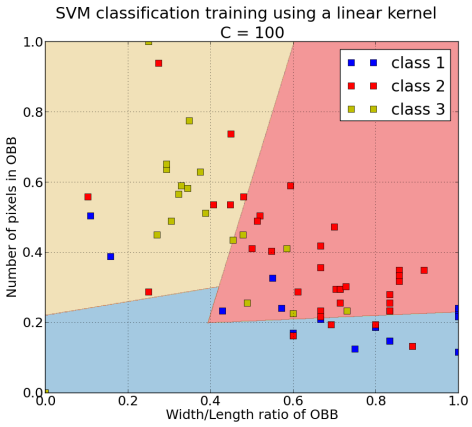


(b)

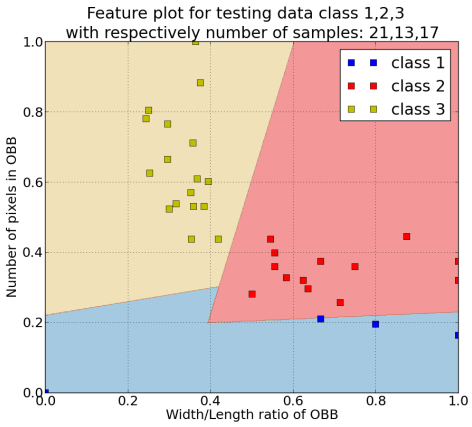


(c)

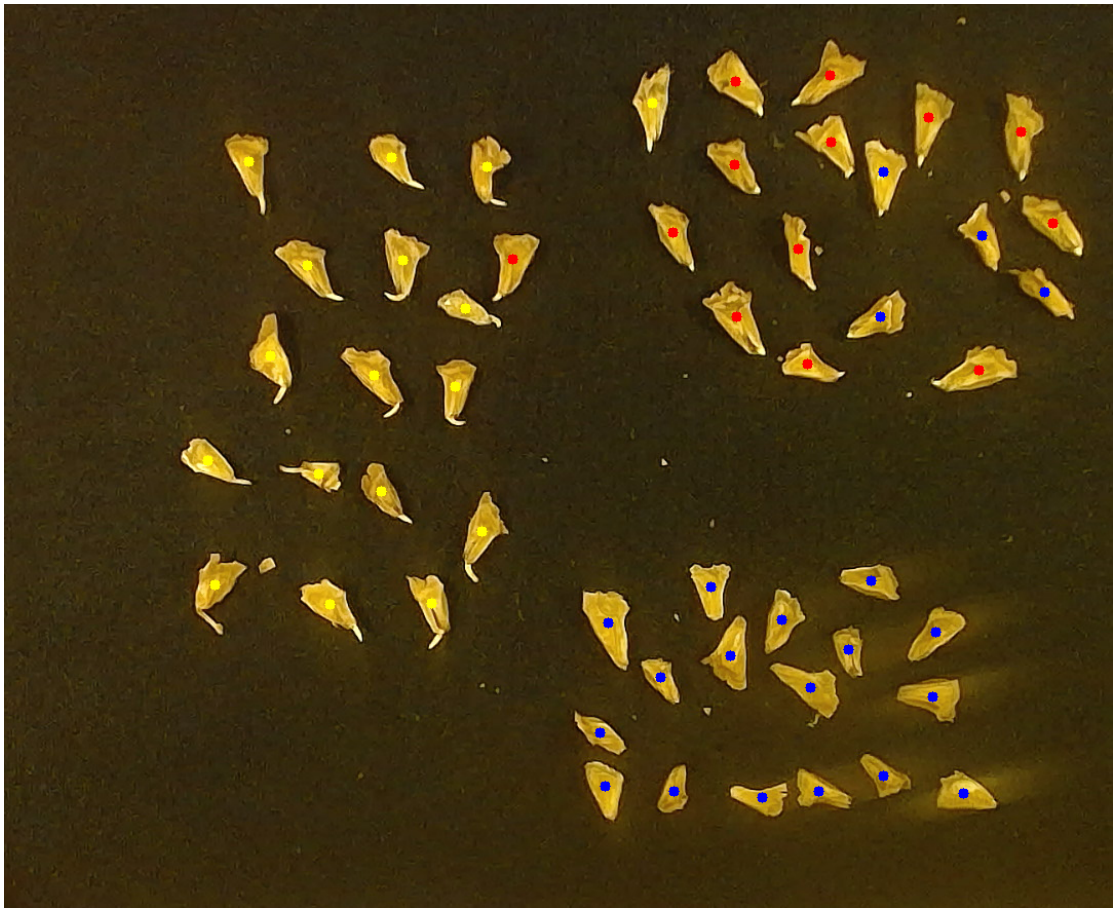
Figure 6.26: (a) Training data with learned classification regions. (b) Testing data classified due to classification regions. (c) RGB image with classified seed where each seed is marked at their center of mass with a respectively class color. Blue is class1, red is class2 and yellow is class3.



(a) $\gamma = 0.8, C = 10$

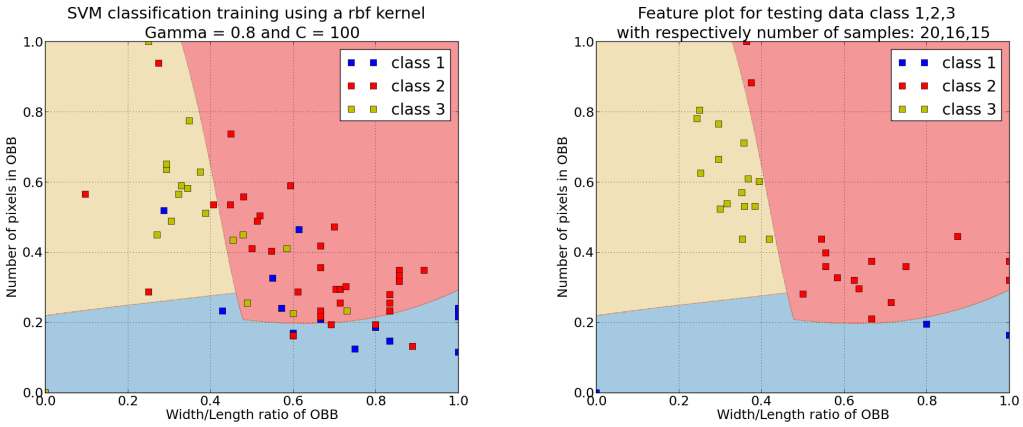


(b)

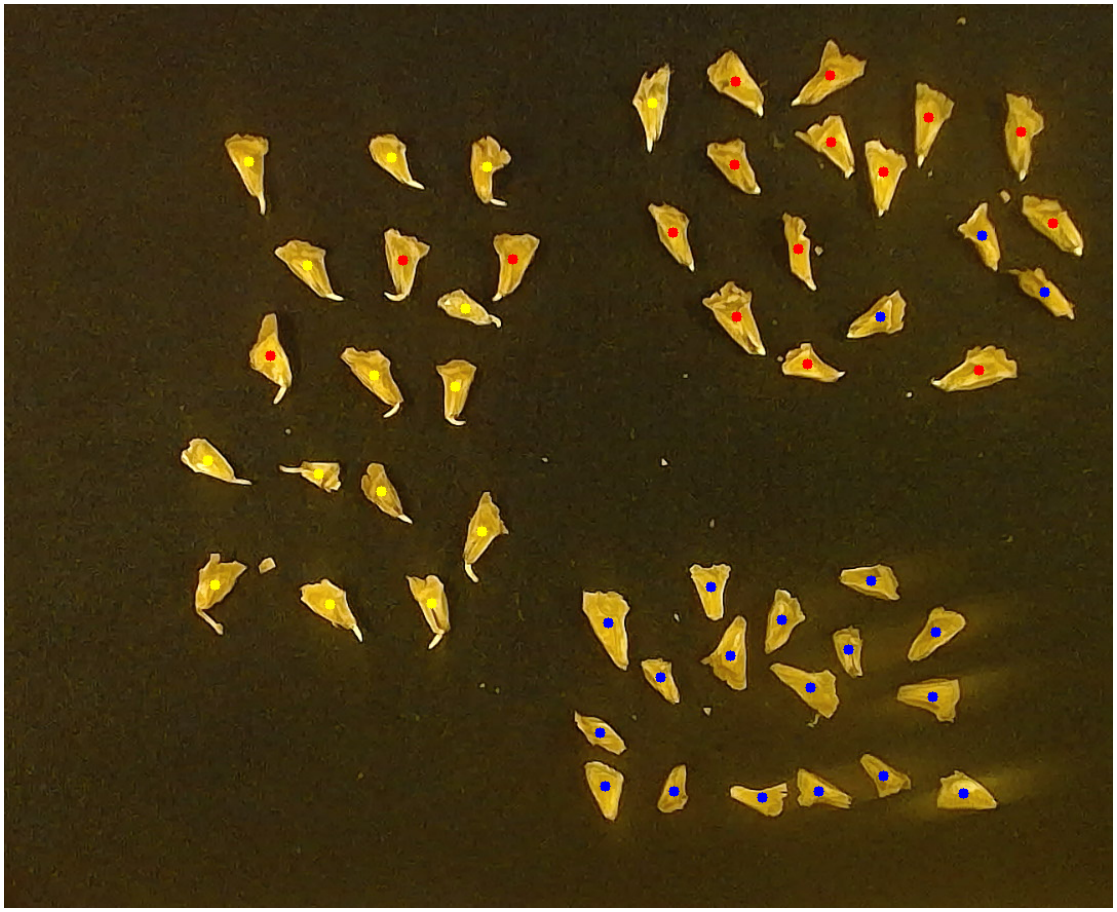


(c)

Figure 6.27: (a) Training data with learned classification regions. (b) Testing data classified due to classification regions. (c) RGB image with classified seed where each seed is marked at their center of mass with a respectively class color. Blue is class1, red is class2 and yellow is class3.



(a) $\gamma = 0.8, C = 10$ (b)



(c)

Figure 6.28: (a) Training data with learned classification regions. (b) Testing data classified due to classification regions. (c) RGB image with classified seed where each seed is marked at their center of mass with a respectively class color. Blue is class1, red is class2 and yellow is class3.

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:

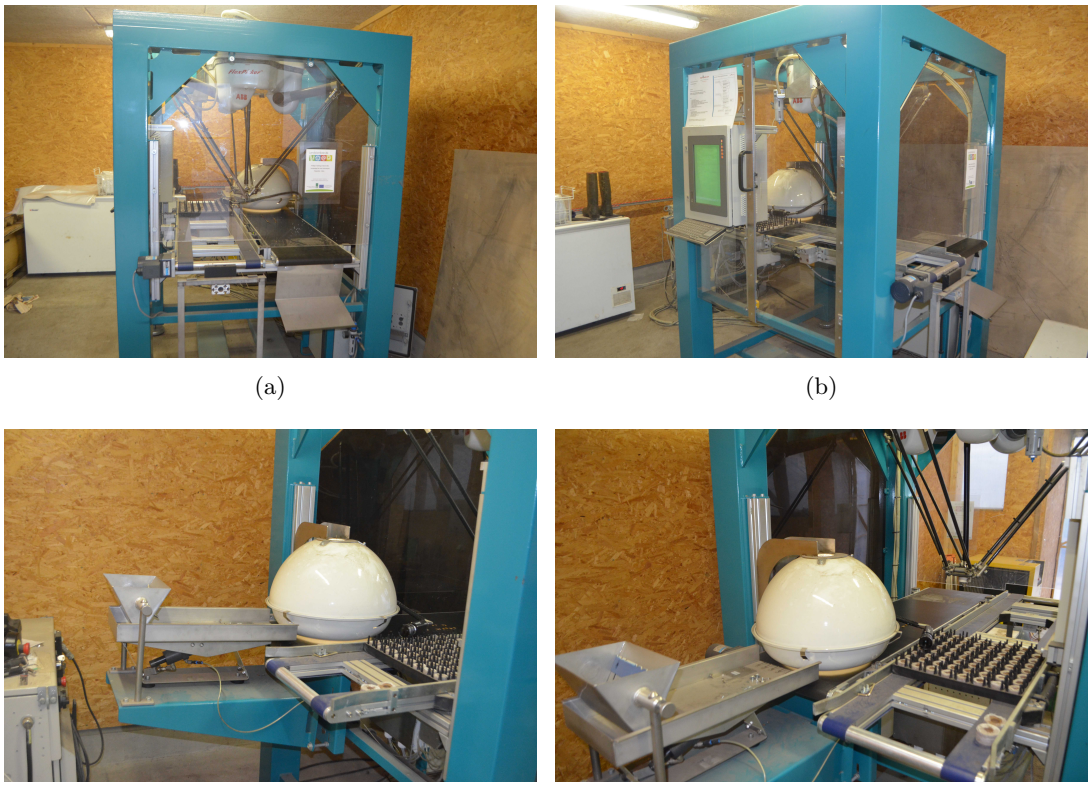
$$x = \frac{\left(u - \frac{columns}{2}\right) \cdot z}{f_x} \tag{6.6}$$

$$y = \frac{\left(v - \frac{rows}{2}\right) \cdot z}{f_y} \tag{6.7}$$

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 6.6 and 6.7 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) [10]. The *Chain of the robotic system* is described in section 5.2 in figure 5.1.

8 Discussion

Working with organic materials is more complex than initially thought. Using tree seeds which can change characteristic from day to day, sets high demands for project planning. However the project methods *Chain of Project*, described in section 5.2 has been a good structure to keep the project on track.

Besides from fulfilling the learning goals, described in section 3.3 the goal of the project was to have a vision system, which can detect and classify tree seeds, by using the principle of supervised learning. As described the vision system contains different component, where each one is discussed as follows:

Get image from web camera

This component worked out of the box, using the OpenCV library. However this component will not be able to handle other interfaces than USB. Therefore if the project changes camera to e.g. Ethernet communication, the interface must be changed.

Preprocessing component

The preprocessing component was optimized in order to repair the sprout features. The fore ground image is generated using OTSU's threshold algoritm and the sprout image is generated using the HSV methods. The weak link here is the HSV methods, which suffers from the hardcoded HSV parameters. The implementation of the sprout repairing methods using bitwise AND operation worked well. The dilation iteration can be discussed and perhaps implemented adaptively. Alternative methods like semi-supervised learning would be an interesting perspective to implement in order to produce the foreground, seed and sprout images. In overall the preprocessing component worked well, but suffers from the low quality of the input RGB images from the web camera.

Segmentation component

The segmentation component was optimized with a K-means clustering algorithm in order to catch the ground truth white pixels. However an optimization of the preprocessing component helped the clustering algorithm. The feature extraction, based on the bounding boxes of sprouts worked ok. However the feature extraction depends heavily on the segmentation and preprocessing results. It could be interesting to test other features.

Classification component

The Perceptron classifier was implemented to learn how classifiers and supervised learning works. This has given a great knowledge and perspectives. The Perceptron was replaced with the Support Vector machine which worked well and gave high classification rates. However in figure 6.25 and figure 6.26 the support vector machine give results that can be discussed. The separating line between the red and yellow area is higher than expected. One reason could be choice of features, which was shown later had plays an important role. It can also be discussed why the SVM with a linear kernel performs slightly better than the RBF kernel, even that the RBF is more sophisticated. The choice of parameters was based on emperically approach. Using a grid search might gives other parameters and improve the result for a SVM with a RBF kernel. Another idea is the number of samples, which is relatively low.
In figure 6.25,6.26, 6.27 and 6.28, image (b) shows the testing data that have been classified. The ground truth can be seen in image (c). However a better interpretation of image (b) would be to have the ground truth of the testing data in image (b). This was notices late in the project and therefore not changed.

Outputting 3D coordinates

This component was using the pinhole model to transform from 2D (u,v) space to 3D (x,y,z) space. However coordinate in 3D was relative to the camera frame and needs to be calibrated to a robotic frame, which can only be done when the setup is known.

The new setup

The wooden box in Robolab was far form ideal regarding shape and light condition. Many days of work was spend doing optimization of the vision system since the setup changed. It would have been more effective, if the white metal sphere, seen in figure 7.1 from ImproSeed was available in RoboLab. In that way the essential part of the vision system would be available 24/7. Time would be saved and could be spend on the robotic interface. In order to work with robots, a lot of time must be invested. The geographical distance between Odense and Sunds did not contribute to faster interfacing between the vision and robotic system. An other approach would be to have a full working weeks at the company in Sunds.

Data

Training data of three classes has been fed to the system in order to let the system learn. The parameters for the Support Vector Machine has been changed in order to fit best the training data. Having a validation data in order to verify the parameters would be preferred. If the validation data with the parameter is successful then the system would have been ready to be tested with the testing data. These could have been generated easily by placing the seeds differently in the field of view. In other words the system has learned from training data, but not verified. The validation data would be generated by using cross validation.

Number of classes

At first place it was intuitive to have three classes in order to divide the seeds into three categorize, like sprouts are too short, sprouts are OK length and finally sprouts are too long. However this is still a binary classification, since two of the classes represent the same condition a seed can have, i.e bad or good condition. If the sprout of a seed is either too short or too long, it is still in the bad category. Even with a higher dimensional feature space, the classification is still binary.

9 Conclusion

A project methods, named *Chain of Project*, has been used to structure the project into components such as *Preprocessing, Segmentation and Classification*. This involves implementing two proof of concepts. The first proof of concept is a *Black Pepper Detection System*, which lays the basic functionality for the preprocessing and segmentation component. A second proof of concept focus in the principle of supervised learning where a Perception classifier is implemented. The result of the two proof of concepts is the foundation for the final vision system, where the preprocessing and classification component has been further optimized. The preprocessing component is optimized such that sprouts are repaired and the classification component is optimized such that the Perceptron is exchanged with a support vector machine (SVM). The system is taught by the principle of supervised learning. Three classes of labelled training data has been used with a SVM in order to classify unlabelled testing data. Each seed in the cameras field of view has been classified and each seeds center of mass has been extracted. The center of mass coordinate is further transformed to a 3D (x,y,z) relative to the image frame using the pinhole model. The final result in figure 6.27 and 6.28 shows how the seeds in the cameras field of view has been classified using a SVM classifier with a linear and radial bases function kernel respectively.

The final result shows that using a SVM with a linear kernel, with the regularization parameter $C = 100$, gives the highest classification rate of $1 - \frac{6}{51} = 88.24\%$. The false negatives error rate is $1 - \frac{5}{51} = 9.80\%$ and the false positive error rate is $1 - \frac{1}{51} = 1.96\%$. Different features were tested. The best result was gained by using the $\frac{width}{length}$ ratio and number of pixels within the oriented bounding box of the sprouts.

A classification rate of 88.24% is acceptable, when referring to the forest planting efficiency of $60 - 75\%$. However the classification rate depends on how ground truth is defined. Through this project the ground truth was defined by human visual inspection. The hue feature was not extracted correctly due to low quality of the web camera and hence not taking into account.

10 Future work

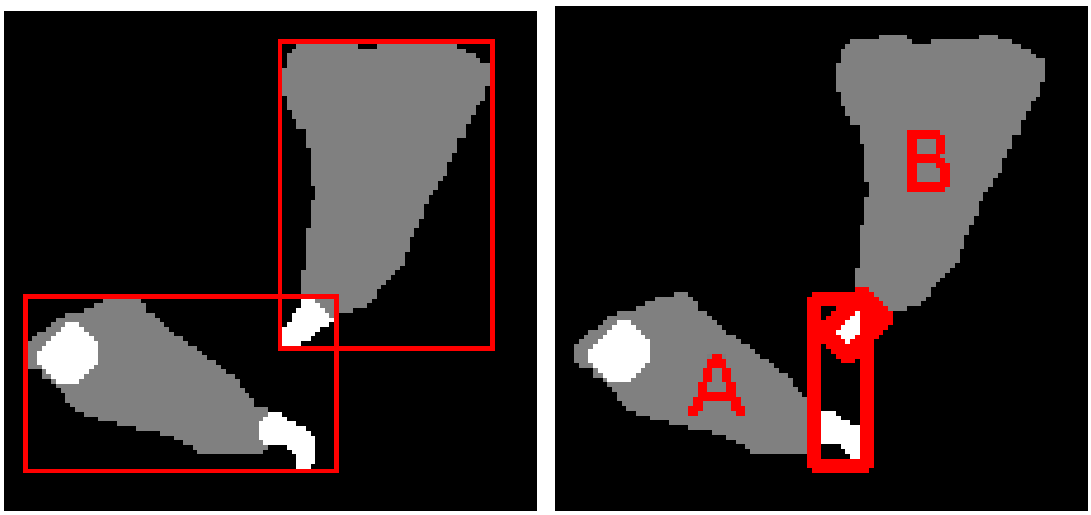
In this chapter different task for future work is explained. The overall future work is obviously to have a robust vision system, which is interfaced to a robotic system through ROS. However the future work described in this chapter is focus on the vision system only.

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 foreground 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 a 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 foreground 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 an 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 foreground image and the seed and sprout image, different approached could be interesting. By the use of semi-supervised learning the foreground image could be created by having training data of background pixels and foreground pixels. This should be extended to find sprout pixels as well.

10.4 Other features

A obviously future work would include optimizing the segmentation component with different features. Features which are invariant to scale, rotation and translation is always good features, compared to features which are not invariant. An example of this would be to use the Hu moments [4]. However a pair-wise test using the seven Hu-moments did not give any particular seperable featureplot. The result is shown in appendix K. Exploring new features is an interesting and important future work. Additionally classification in 3D or higher dimension with the SVM or perhaps the Random Forest algorithm would be interesting.

10.5 Other tree seed cultivars

Only one tree cultivar was used in the project. Having a system that is able to learn other cultivars characteristic and be able to classify more than one cultivar is future work.

10.6 Save computation time

In subsection 6.3.2 figure 6.18 two methods of extracting the center of mass (COM) coordinate is present. Computation time could be saved, since the COM is already available as attribute when calling the *minAreaRect* and hence moment calculations could be omitted.

10.7 Estimating what part of pipeline to work on next

There is a lot to do for future work. From the video lecture from Coursera.org [13], an analytically approach of estimating the most important component to work on is presented. The idea is that the overall system percentage is measured and each individual component in the pipeline is evaluated to see how much further optimization will increase the overall percentage of the system. This will minimize waste time in the future work process. This project was discovered few days before hand-in and therefore unfortunately was not used in the project.

11 Bibliography

[1] Basler scout gige vision area scan cameras, .
URL http://sine.ni.com/nips/cds/view/p/lang/da/nid/210344.
[Online; accessed 2015-05-24.

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

[3] Scikit-learn - grid search, .
URL http://scikit-learn.org/stable/modules/grid_search.html.
[Online; accessed 2015-05-27.

[4] Hu moments - opencv, .
URL http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html.
[Online; accessed 2015-05-23.

[5] Opencv - open source computer vision libray, .
URL http://opencv.org/.
[Online; accessed 2015-05-27.

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

[7] Area of single pixel - opencv, .
URL http://answers.opencv.org/question/58/area-of-a-single-pixel-object-in-opencv/.
[Online; accessed 2015-05-23.

[8] Rbf kernel - wikipedia, .
URL http://en.wikipedia.org/wiki/Radial_basis_function_kernel.
[Online; accessed 2015-05-27.

[9] Rbf function, lecture 16 - caltech by professor yaser abu-mostafa, .
URL https://www.youtube.com/watch?v=O8CfrnOPtLc&feature=youtu.be&t=104.
[Online; accessed 2015-05-27.

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

[11] Support vector machine - wikipedia, .
URL http://en.wikipedia.org/wiki/Support_vector_machine.
[Online; accessed 2015-05-27.

[12] Scikit-learn - svm parameters, .
URL http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html#example-svm-plot-rbf-parameters-py.
[Online; accessed 2015-05-27.

[13] Part of pipeline to work on next - courseara, .
URL https://class.coursera.org/ml-005/lecture/113.
[Online; accessed 2015-05-27.

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

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

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

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

[18] 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].

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

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

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

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

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

[24] 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.

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

[26] OpenCv dev team.
Opencv - support vector machines.

http://docs.opencv.org/modules/ml/doc/support_vector_machines.html, available 4-12-2014.

[27] 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.

[28] 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.

[29] 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.

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

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

[32] 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].

[33] 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.

[34] 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.

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

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

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

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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].

[44] 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].

[45] 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 an uniform light distribution. The camera is interfaced to the nearby PC by an USB connection. This is shown in figure A.2.

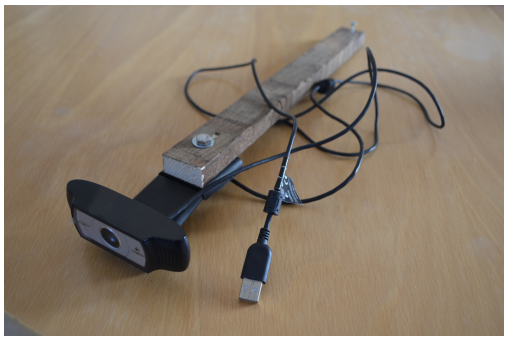


Figure A.1: Webcamera extended with wooden stick

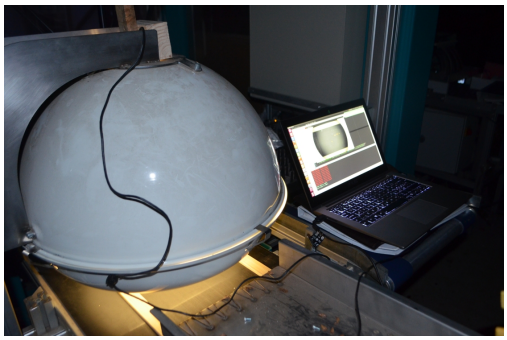


Figure A.2: Webcamera mounted inside sphere and connected to PC

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.

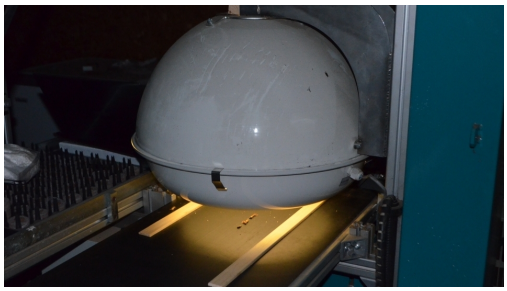


Figure A.3: Wooden sticks used to align distance

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 an 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

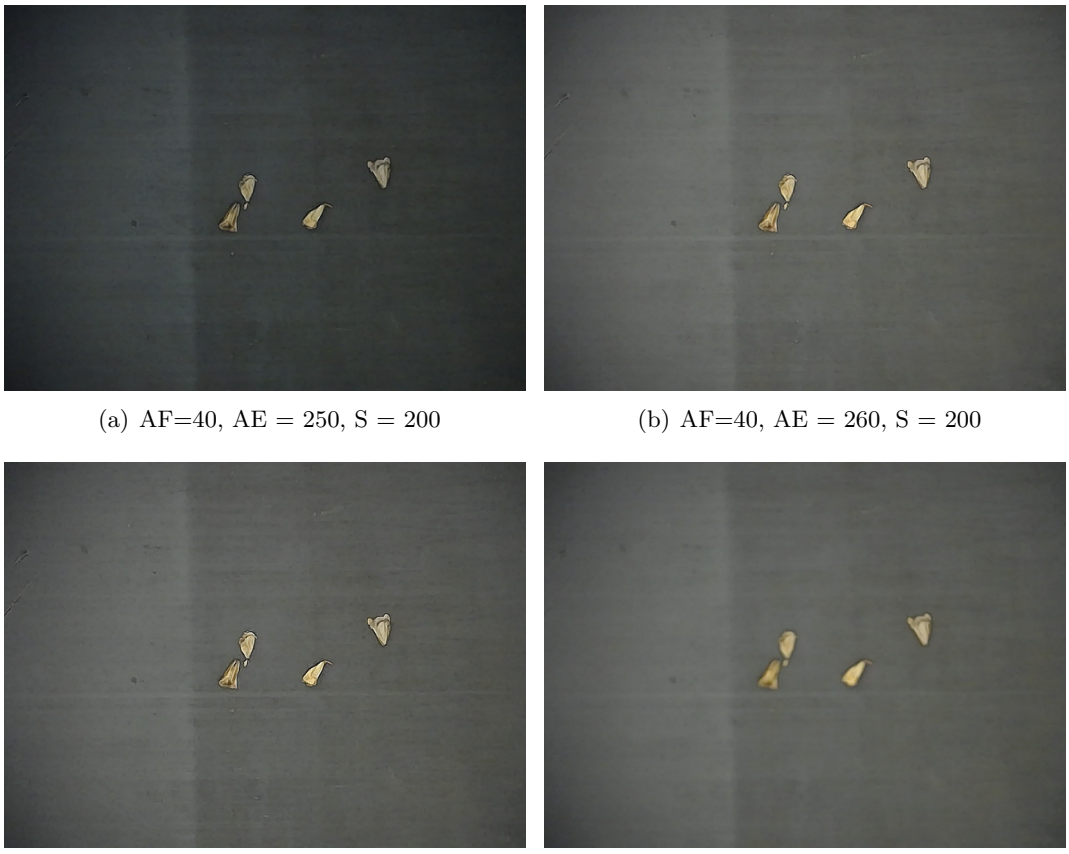
The reason for disable the autofocus is to avoid any uncontrolled behaviour of the web camera while the system runs. Since the distance from the seed on the conveyor belt and to the camera lens do not change over time, the argument for having a fixed zoom exist. If the autofocus is not disabled, the 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 5.4.1 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

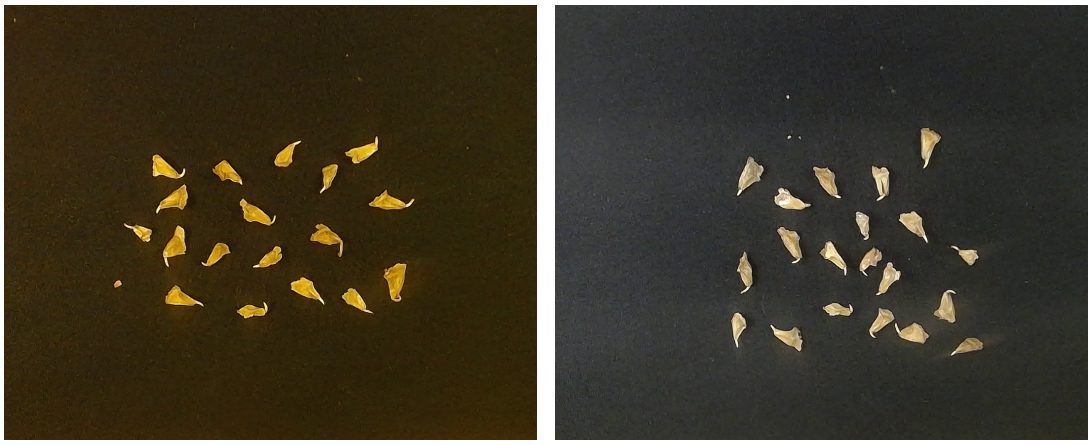
(d) AF=50, AE = 260, S = 200

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 an 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.

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

Listing 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 [17].

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. Evt et Basler 2MP farve kamera med Ethernet tilkobling [1].

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 [2].

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

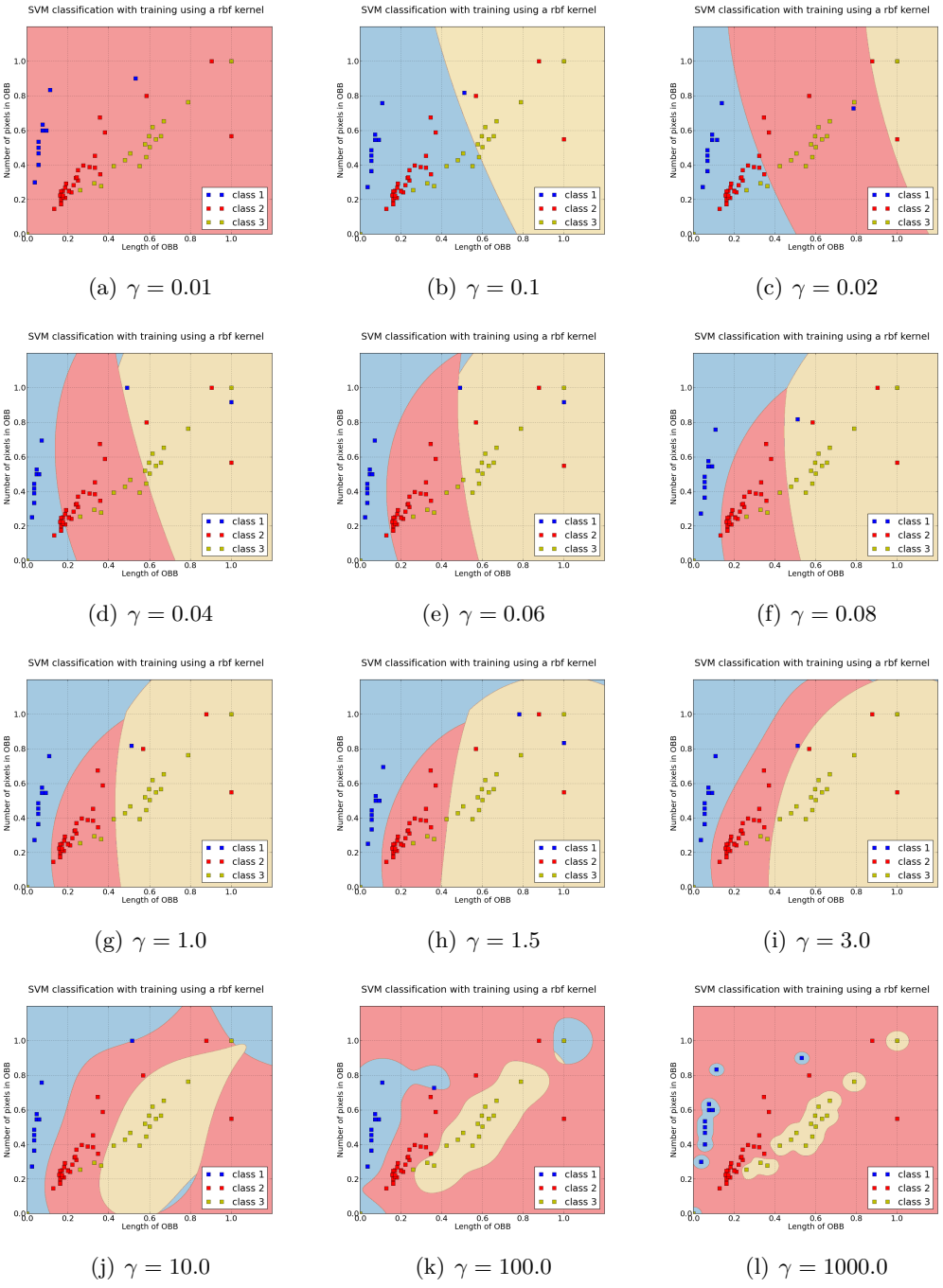
g. Udarbejde interface til robotten ved brug af ROS systemet.
   - Fastlæg hvilket interface der ligger til ABB Flexpicker robotten.
   - Opbyg ROS node, der publisher et 3D punkt og teste om dette kan modtages i robotsystemets buffer, samt at robotten flytter sig til det ønskede 3D punkt.

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) $\gamma = 0.08$

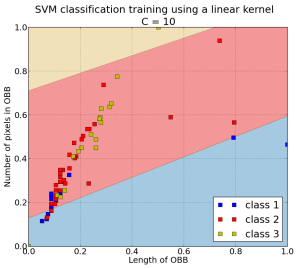
(g) $\gamma = 1.0$ (h) $\gamma = 1.5$ (i) $\gamma = 3.0$

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

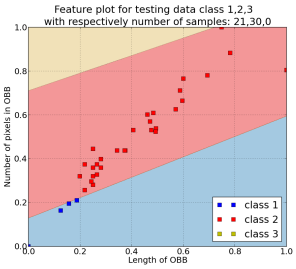
Figure G.1: Images where the γ goes from 0.01 (a) to 1000 (m).

H SVM result with normalize data

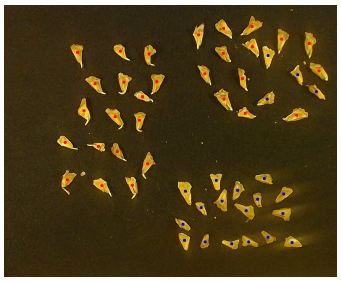
On the following page, several test is performed testing the parameters γ and C. The result is shown in figure I.1. The images is aligned row-wise. Each row contains three images. E.g row 1 row goes from image (a), (b), (c), row 2 goes from image (d), (e), (f) and so forth. The image in the first column is a feature plot with training data of class1, class2 and class3 and the classification regions learned from the training data. The second column is a feature plot with testing data, where the testing data is classified due to which region the feature is localized. The third and last column shows the result of the classification seen in the RGB image. Each seed is marked classified by drawing the color, respective to the regions, at the center of mass of the seed.
The following pages in this appendix is structured as followed:
Image (a) to (l) is using a RBF kernel with different parameters of γ and C. Image (m) to image (o) is with a linear kernel.



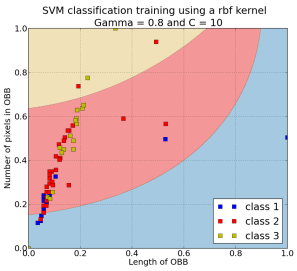
(a) Linear kernel, C = 10



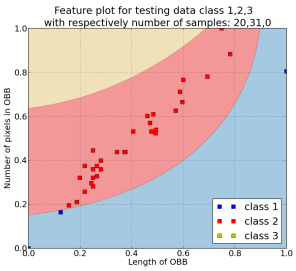
(b)



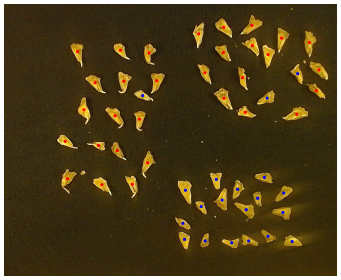
(c)



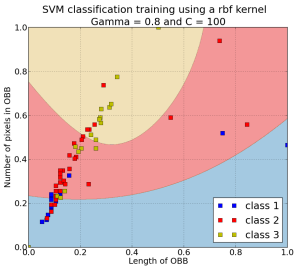
(d) $\gamma = 0.8, C = 10$



(e)

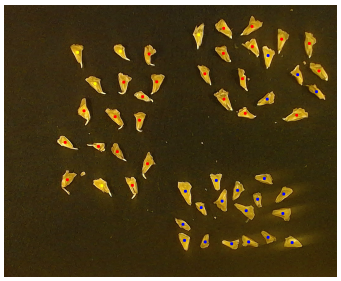


(f)

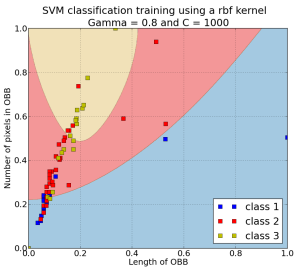


(g) $\gamma = 0.8, C = 100$

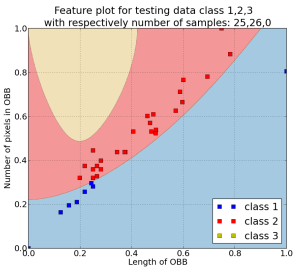
(h)



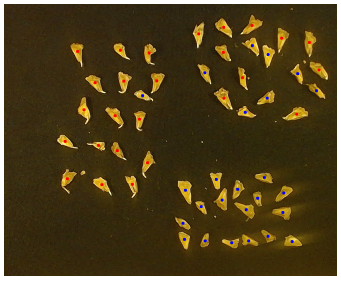
(i)



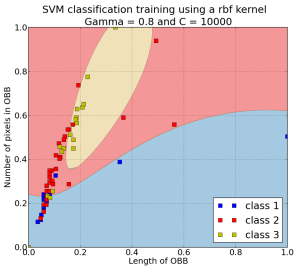
(j) $\gamma = 0.8, C = 1000$



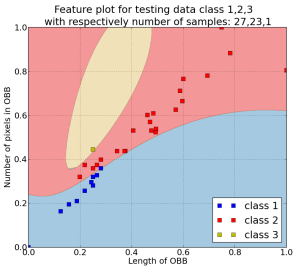
(k)



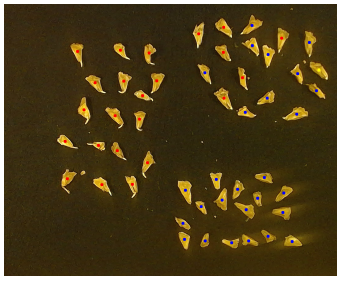
(l)



(m) $\gamma = 0.8, C = 10000$



(n)

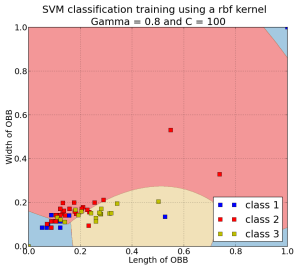


(o)

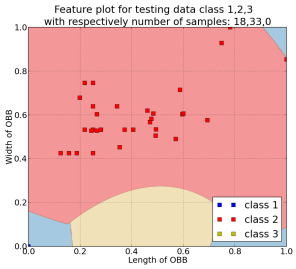
Figure H.1: Images where the γ goes from 0.01 (a) to 1000 (m).

I SVM result with normalize data with other features

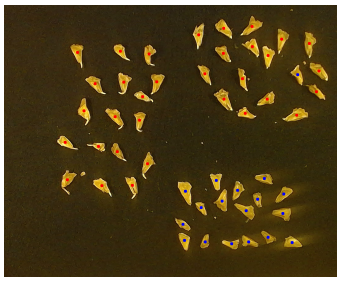
In this appendix several test is performed testing different combinations of features in 2D, using the parameters $\gamma = 0.8$ and $C = 10$.
The result is shown in figure I.1. The images is aligned row-wise. Each row contains three images. E.g row 1 row goes from image (a), (b), (c), row 2 goes from image (d), (e), (f) and so forth. The image in the first column is a feature plot with training data of class1, class2 and class3 and the classification regions learned from the training data. The second column is a feature plot with testing data, where the testing data is classified due to which region the feature is localized. The third and last column shows the result of the classification seen in the RGB image. Each seed is marked classified by drawing the color, respective to the regions, at the center of mass of the seed.



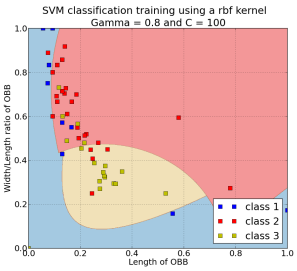
(a) Linear kernel



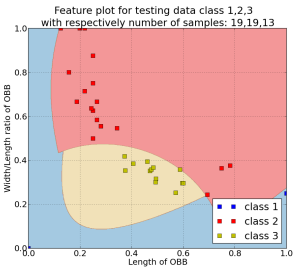
(b)



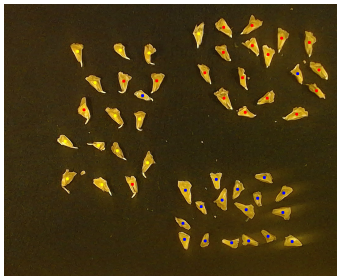
(c)



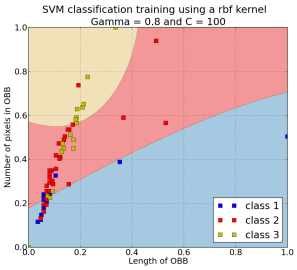
(d) $\gamma = 0.8, C = 10$



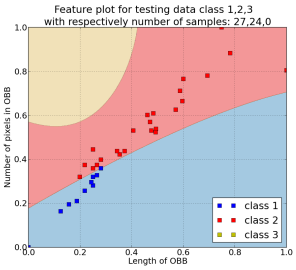
(e)



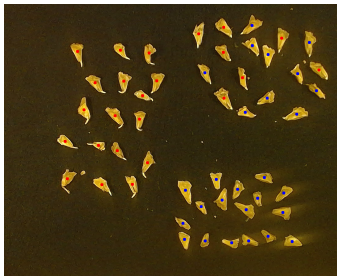
(f)



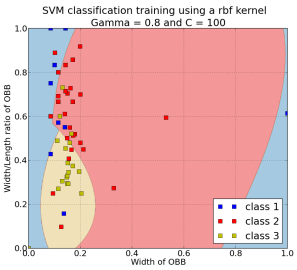
(g) $\gamma = 0.8, C = 100$



(h)

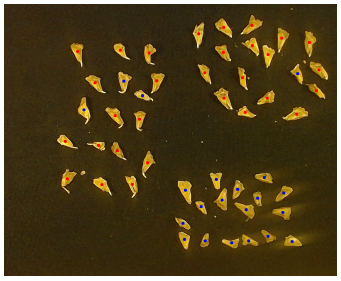


(i)

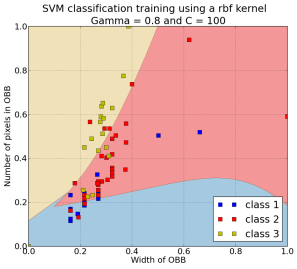


(j) $\gamma = 0.8, C = 1000$

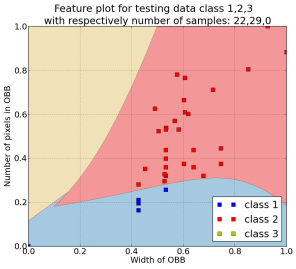
(k)



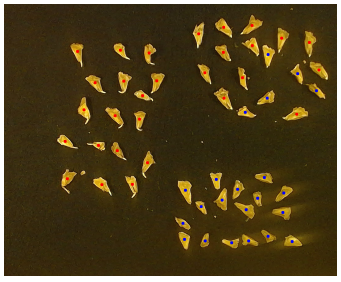
(l)



(m) $\gamma = 0.8, C = 10000$



(n)



(o)

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

J Pseudo code for K-means cluster algorithm

Input: List of all white "sprout" pixels $p_i(u, v)$
Output: Two clustered lists: $cluster_1$, $cluster_2$ 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, $currentCenter_1$ and $currentCenter_2$

---

while *TRUE* **do**
  for *i'th pixel* p_i *in input list* **do**
    $d1 := \sqrt{currentCenter_1^2 - p_i^2}$
    $d2 := \sqrt{currentCenter_2^2 - p_i^2}$
    if $d1 < d2$ **then**
      Append p_i to $cluster_1$
    else
      Append p_i to $cluster_2$
    end
  end
  $newCenter_1 := \frac{1}{n} \sum_{i=0}^{n} cluster_1$
  $newCenter_2 := \frac{1}{n} \sum_{i=0}^{n} cluster_2$
  if $newCenter_1 = currentCenter_1$ ***AND*** $newCenter_2 = currentCenter_2$ **then**
    break
  else
    $currentCenter_1 = newCenter_1$
    $currentCenter_2 = newCenter_2$
  end
end

Algorithm 2: K means algorithm for K = 2

K Pair-wise HU-moments test

This appendix show the result of a pair-wise Hu-moment test. The seven Hu-moments is index from 0 to 6. In figure K.1 the image goes from (a) to (l). The first image (a) is a pairwise test between Hu moment 0 vs Hu moment 1. The test is noted as XvsY, i.e. for image (a) the test is 0vs1. The next image (b) is 0vs2, image (c) is 0vs3 and so forth all the way up to 2vs3 in image (l). In figure K.2 the test continues from 2vs4 all the way up to 5vs6.

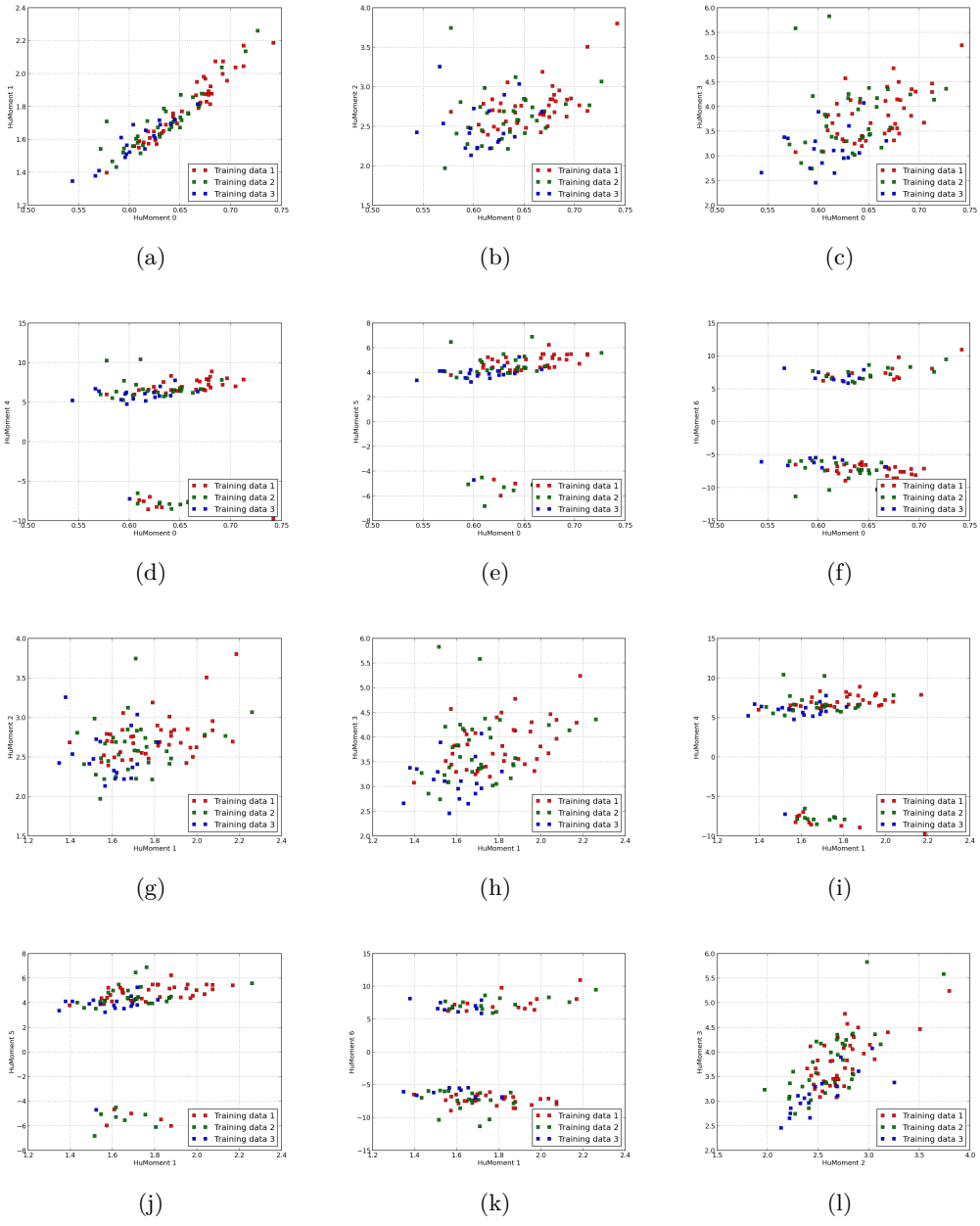
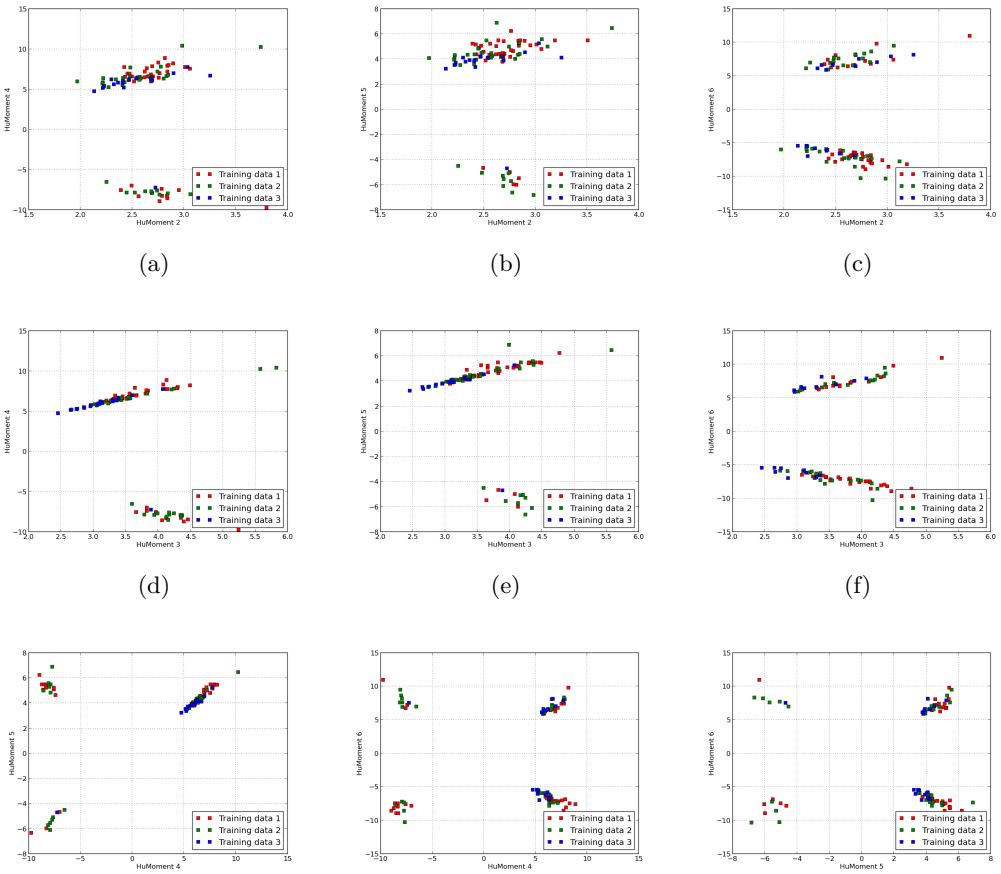


Figure K.1: Pair-wise Hu-moment test



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure K.2: Pair-wise Hu-moment test