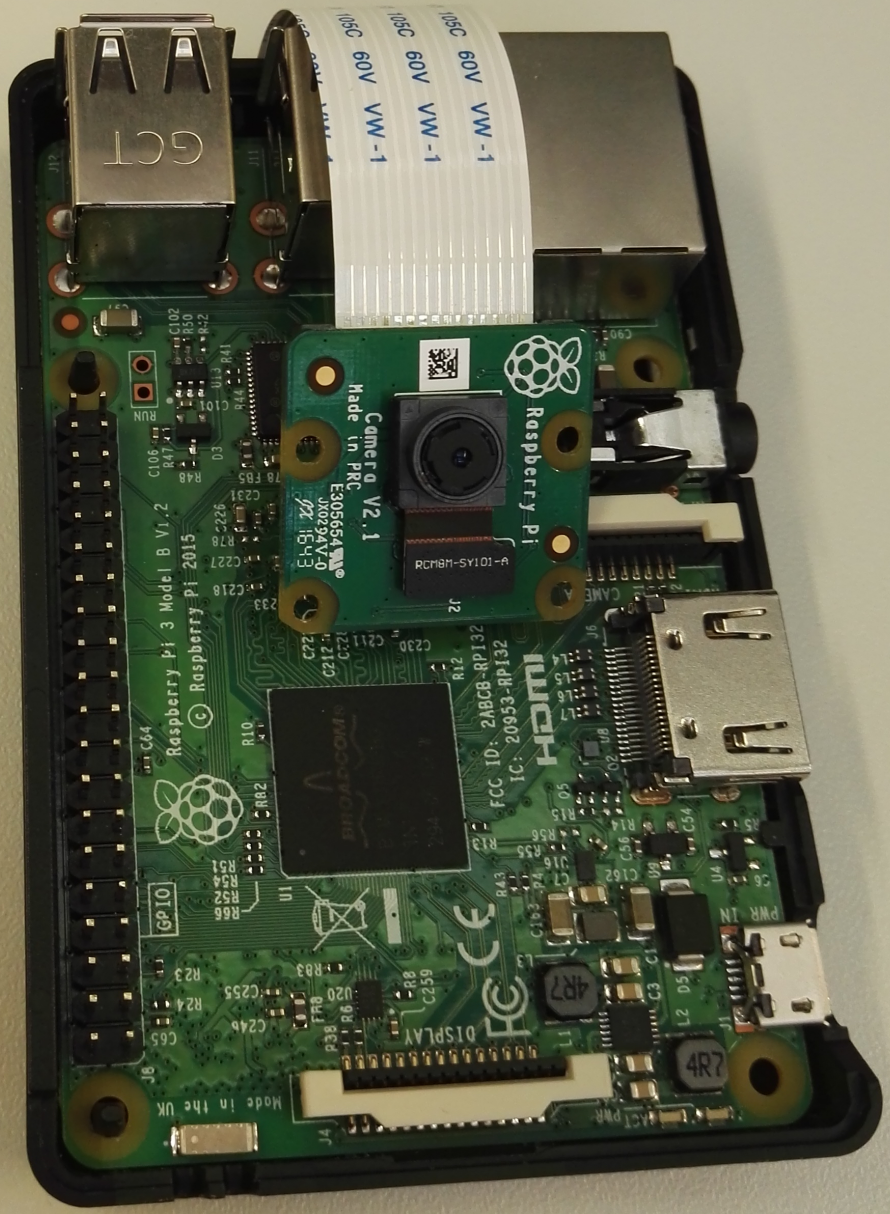
Smart sensors and communication using IoT in supermarkets

Shelf monitor system

M.D. Berkers & E. Hagenaars

Technische Universiteit Delft



TU Delft Delft University of Technology

kpn

SMART SENSORS AND COMMUNICATION USING IOT IN SUPERMARKETS

SHELF MONITOR SYSTEM

by

M.D. Berkers & E. Hagenaars

in partial fulfillment of the requirements for the degree of

Bachelor of Sciences
in Electrical Engineering

at the Delft University of Technology,

Students:	M.D. Berkers	4223438
	E. Hagenaars	4272404
Project duration:	March 24, 2017 - July 7, 2017	
Supervisors:	J. Hoekstra (TU Delft)	
	P. Marcelis (KPN)	
	B. Frens (KPN)	

TU Delft Delft University of Technology

ABSTRACT

This thesis tries to find a solution for the problem of managing and monitoring the banana shelf in a supermarket using IoT. The research focuses on using a wireless sensor that detects some features of the banana shelf while being non-intrusive. The three main features that are examined of the shelf are the quality and quantity of the bananas and the quality of the shelf. First a research was conducted to find the best sensor to use for these measurements. The chosen sensor is a color image sensor, the platform for the IoT device is a Raspberry Pi. Using the python programming language in combination with the openCV library image processing was used to detect the features.

The image is first smoothed using a Gaussian filter, afterwards the foreground is segmented. The different segmentation methods are researched and adaptive thresholding is used. To determine the quantity of the bananas and quality of the shelf the stickers on the bananas are detected. This detection is implemented using different filtering methods ranging from spectral filtering to color thresholding. With the segmented foreground the quality of the bananas is assessed using a color histogram. This information is then sent to a communication module that is connected to a IoT dashboard for user interpretation.

With the proposed design the status of the shelf including the percentage of the shelf filled, the quality of the bananas on the shelf and the neatness of the shelf are available for a supermarket manager to better organize his supermarket. This sensor makes it possible to better organize the banana shelf and act preemptive instead of reactive.

Contents

1 INTRODUCTION

This chapter contains the introduction of the thesis. First a background of the assignment of the thesis will be given in Section 1.1. From the assignment a problem will be defined in Section 1.2 and contains the main problem and goal for the thesis. The last Section 1.3 will contain a synopsis of the thesis. To explore the technical aspects of the assignment state-of-the-art analysis is performed in Chapter 3.

1.1. BACKGROUND

According to a study by Gartner Inc [1], the number of devices connected to the Internet will increase by 30% over the next year. Internet of Things (IoT) devices are now still largely used by consumers, but business applications are also growing. KPN New Business has a group working on IoT devices and presented us with a business case. A client of KPN identified some problems within a supermarket, KPN wants to produce a proof of concept to demonstrate the usefulness of IoT devices.

As part of a non-disclosure agreement no further information about the client can be discussed in the thesis.

KPN asked us to research possible solutions for the following problems:

- Detecting obstructions in aisles and in front of emergency exits.
- Monitoring shelf and fruit quality of the banana shelf.
- Communications between sensors and the Internet in a supermarket environment.

The idea was to develop two sensors and connect these using wireless communication. Two subgroups work on developing a sensor and the last subgroup works on the communication between the sensors and provide a main hub to the Internet. Our thesis is about the monitoring of the shelf and fruit quality of the banana shelf.

1.2. PROBLEM DEFINITION

The problem of monitoring the banana shelf can be split up in three parts.

- Monitoring the quality of the bananas so there are always ripe and no rotten bananas on the shelf.
- Determining the quantity of the bananas to ensure there are always enough bananas for customers to buy.
- Checking if the shelf is properly filled, looking at the orientation of the bananas.

In the supermarket it currently takes too long to detect when the fruit shelves have to be refilled. This creates the problem that when customers want to buy bananas they are disappointed because they are unavailable. It also frequently happens that the bananas that are available, are not ripe or almost rotten. Having a shelf filled with green and brown bananas leaves a bad impression to the customers about the fruit and vegetables section of the supermarket in a whole.
The reason bananas are the main focus in our thesis is because of a research conducted by the supermarket. They found evidence that the opinion of the customer about the fruit and vegetables section strongly depends on the banana shelf. If the banana shelf is not filled properly and the fruit is of bad quality, the opinion about the whole section is influenced. To increase customer satisfaction they want to make sure the shelf is properly filled and the fruit available is of high quality.
Our solution will be a wireless sensor capable of monitoring the banana shelf on quality and quantity and send this information at a regular interval to a communication module.

1.3. THESIS SYNOPSIS

The goal of this thesis is to research the possibilities of a sensor to monitor different types of shelves. We chose the banana shelf because this shelf is the most important one in a supermarket. The project is divided in 3 parts, first research the possibilities and choose a sensor. After we chose the image sensor the design of the different types of methods to detect the features of the shelf was done. When we had developed some early prototypes we used the sensor in combination with the board in combination with test images to test our prototypes and make changes where needed.

2
PROGRAM OF REQUIREMENTS

2.1. INTRODUCTION

The overall assignment is to create a system of smart sensors in a supermarket which are connected to an Internet of Things (IoT) platform. The IoT platform will provide managers an overview of the store and the current problems or defects detected by these smart sensors. This thesis will focus on one of the smart sensors: a banana shelf observation device. Observing the shelf, the device will monitor three main features of the shelf. These are the quality and quantity of the fruit and the quality of the shelf. To achieve this, the system will detect characteristics that give information about one of the main features.
An important feature of the quality of the shelf is product placement. If fruits are placed as they supposed to be, the same patterns and objects will be shown. For example, Chiquita bananas have blue stickers on the front, these blue stickers will be shown if placed properly, they will have the branches on top and will be positioned vertically.
A feature to define the quality of the fruit is odor or color distribution. When fruits ripen, these features will change. For example, bananas turn from green (unripe) to yellow (ripe) to brown (overripe).
To determine the quantity of fruits, the surface percentage of these specific colors are important. If the background has a different color, this is easily solved. But with a similar colored background, other methods need to be used.
An important property of IoT devices is user friendliness. This means that the device should be easily implemented with low maintenance. All of these features will be considered in the program of requirements.

2.2. FUNCTIONAL REQUIREMENTS

This thesis will focus more on the sensing of the features that define the quality/quantity of the fruits and the quality of the shelf. The functionality will therefore be:

(1.1) The system should sense the color distribution of the bananas alone while ignoring the background.

(1.2) The system should calculate the surface percentage of matching colors compared to the background of the bananas.

(1.3) The system should detect and count at least 95% of the stickers visible on the bananas.

2.3. ECOLOGICAL EMBEDDING IN THE ENVIRONMENT

The environment of the device will be a supermarket. A focus is customer satisfaction and employee efficiency. With this in mind, the following requirements for the environment can be made:

(2.1) The device must not obstruct the customers.

(2.2) The device should be non-intrusive and no modifications are to be made to the shelf.

2.4. SYSTEM REQUIREMENTS

The system requirements can be divided into three sections. First the usage of the device where the requirements for the users is defined. Secondly, the production and development of the device. And finally, the discarding of the device.

2.4.1. UTILIZATION

The utilization requirements are created in the perspective of the user, the manager or an employee. It therefore has to be assumed that the user does not have extensive knowledge of electronic devices.

(3.1.1) Resetting should be a single button operation.

(3.1.2) The device will update the feature values every five minutes.

(3.1.3) Between updating actions, the device should be in low-power or hibernating mode.

(3.1.4) The computational time should be held to a minimum and should not exceed the update rate.

(3.1.5) Foreground extraction should at least extract 90% of the actual foreground and extract at most 10% of the background as an error.

(3.1.6) 95% of the stickers should be detected. The false detection of a sticker should not exceed 10% of the total stickers.

2.4.2. PRODUCTION AND PUTTING INTO USE

The production of the device will be done by an external partner. This partner formulated some requirements for the production and initialization of the product:

(3.2.1) Software should be open source and have no additional cost to the device.

(3.2.2) Hardware components should be widely available and the cost should be reduced to a minimum.

(3.2.3) Initialization of the system should be easy with minimal input.

(3.2.4) The installation of the device should be easy.

2.4.3. DISCARDING

Discarding the whole device is not recommendable. In order to prevent this, multiple independent components should create the full system. With this information, the following requirement can be formulated:

(3.3.1) Defect components should be replaceable without completely installing a new module.

2.5. DEVELOPMENT OF MANUFACTURING METHODOLOGIES

For the development, the methodologies are free of choice as long as they can be explained why the methodology was used. The platform and programming language are not preassigned.

2.6. LIQUIDATION METHODOLOGIES

When the product is not in use anymore, it should be discarded the same as all other normal electronic devices. The product should almost have the same sort of components as the average computer or mobile device.

3
TECHNICAL REVIEW

This section contains a technical review that performs a state-of-the-art analysis of the present research in this field and possible theory that can be used in designing the system. First a choice is made to determine which type of sensor is best used for the problem. After the sensor is decided, multiple considerations on the use of the sensor to make the best decision in what way the sensor should be implemented, we search for more theory. No specific research was found concerning an IoT device for the detection and classification of bananas on shelves. However, there is a lot of research done on object detection of fruit, including bananas, for automated fruit picking systems. Separately, there are some papers that discuss the quality measurement of bananas and other fruit specifically. No system was found that combines both systems to detect and measure these features in a static environment.

3.1. TYPES OF SENSORS

3.1.1. IMAGE SENSORS

In previous studies [2][3][4][5][6][7][8], image sensors were used for detecting and locating fruits with various image processing algorithms or AI algorithms. In this section, multiple types of image sensors are discussed.

BLACK AND WHITE SENSORS

B/W sensors were widely used for fruit detection until 2000 when the last research was done [4]. Since color sensors have replaced B/W sensors and have improved the results [2], this type of sensor will not be of use to us. A positive property of B/W images is the dimensions. The B/W image sensors provide a 2 dimensional array while a RGB image sensor provides a 3 dimensional array with a Red, Green and Blue component. The computational power needed for operations with B/W images is much less.

COLOR SENSORS

Color sensors capture more information. The color of the fruit is important for the detection. For bananas, the color yellow is dominant and will be most important. The stickers on the bananas are a bright blue, which is an uncommon color in fruits or vegetables. This could also be a good identifier. Color sensors where used in specific research of fruit detection and localization [2].

SPECTRAL AND THERMAL SENSORS

Spectral imaging detect objects based on their reflectance at different wavelengths. Detecting fruit can then even be achieved if the background color is similar to the fruit color. A con of using a spectral sensor is the budget and space. Spectral sensors are expensive, this price increases when the size of the sensor decreases. The use of thermal sensors are useful if the background is similar to the fruit. Fruit absorbs and radiates more heat than leaves or other compounds. These changes in heat radiation are so small that it is hard to monitor. Near Infrared (NIR) sensors where used in combination with color sensor for the detection of various fruit [3]. The book "Hyperspectral Imaging Technology in Food and Agriculture" uses the hyperspectral sensor for their research [5].

3.1.2. ELECTRICAL-NOSE

Detecting the ripeness of a fruit is mostly done by looking at its color or smelling for odors. An Electrical-nose to detect these odors can be very accurate in determining the ripeness of fruit. A big research in ripeness detecting is done with a Ethylene Detector [9]. This detection of ethylene is very complex and the equipment is expensive. To reduce costs, a gas sensor array structure is proposed [10] containing the following gas sensors shown in Table 3.1.

In combination with a trained neural network [3] the results should be acceptable at monitoring the shelf life process.

Name	Main Application
MQ-3	Alcohol
MQ-5	LPG, Natural gas, Coal gas
MQ-9	CO and combustible gas
MQ-131	Ozone
MQ-135	Air quality control
MQ-136	Sulfur dioxide

Table 3.1: List of gas sensors.

3.1.3. CAPACITIVE SENSOR

A capacitive sensor has been used for detecting and measuring ethylene in fruit ripening process [9]. The different concentrations of ethylene in a closed chamber can influence the capacitance measured by a capacitive sensor. Another study has used a parallel plate capacitor for the classification of the ripening process [11]. A big flaw in this design is that the parallel plates need to be on either side of a single banana with maximum distance of 10 cm.

3.1.4. PRESSURE SENSOR

By using pressure sensors that support the shelf, the total mass of the bananas can be measured and thus the fill rate of the shelf. This is easy implemented. Register the empty weight of the shelf and the filled weight of the shelf. The difference is the total weight of the bananas. A signal could be send when that weight drops beneath a certain threshold.

3.1.5. INFRARED SENSORS

By using infrared sensors for detecting when the bananas are blocking the sensors beam, it can be estimated if the shelf is empty along its line. The more sensors are blocked, the more bananas lie on the shelf.

3.1.6. CONCLUSION ON TYPE OF SENSOR

The chosen solution for the problem is a color image sensor. After this choice more theory can be considered for the object detection of fruit quality detection. The objects and bananas have to be detected and the background has to be removed. After only the bananas remain in the image can we look at the quality.

3.2. COLOR FORMATS

There are several types of color formats that can be used using a color image sensor. The input format when using an image sensor is mostly RGB or BGR. Each format has its pros and cons for image processing. In the next few sections we discuss three formats: Grayscale, RGB and HSV.

3.2.1. GRAYSCALE

Grayscale images are created from RGB with the standard transformations given by the IEC standard [12]. Grayscale have the positive that all data is encoded into one matrix, this does mean information is lost during the transformation. Most classical image processing techniques were developed for grayscale. This format is useful in detecting the luminosity of object dark is almost black and the more illuminated an object the light the shade of gray.

3.2.2. RGB

RGB is the color format that is most regularly used and most images are saved in this format. RGB represents an image in the form of it's Red intensity, green intensity and blue intensity. Little image processing is done on the RGB spectrum as it is usually converted to grayscale.

3.2.3. HSV

An important color format for image processing is HSV, which stands for Hue, Saturation and Value. The Hue is the 'angle' of a color. When increasing the angle, the color changes. Most colors with the same Hue are referenced as the same color, for example dark-blue and light blue. The Saturation is the colorfulness

of a color in respect to its brightness. The Value can also be called lightness or brightness. When the Value approaches zero, the color becomes more darker until the color is black. Figure 3.1 visualizes this distribution.

The benefit of the Hue property is the color definition. With the Hue alone, a color can be identified. For example, the color yellow is around 60°. This can easily be filtered this way. The advantages of the Saturation and Value are the lighting, black or white colors have distinguished values for the S and V. When taking an image in a darker room, the Hue should not be affected in theory.

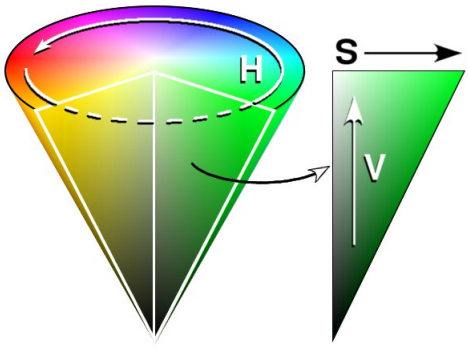


Figure 3.1: Visualization of the HSV color format.

3.3. OBJECT DETECTION

The object detection is to detect stickers on the bananas when they are on the shelf. There are several methods to detect the stickers [13]. They are blue which is easily seen on the yellow bananas that allows multiple color and brightness methods to be used.

3.3.1. CLASSIFICATION ALGORITHMS AND MACHINE LEARNING

Classification Algorithms and Machine Learning are big research topics. Instead of creating heavy algorithms, a simple algorithm which has been trained to find certain patterns are used to classify objects in a given input. These inputs can be from sensors. After training, the algorithm uses only simple calculations to transform the input to a desired classification. The training however is difficult and almost never results in a global solution for big problems. For some Classification Algorithms, more than thousands of training samples have to be provided to the machine and computation power is big during this training. In the next part, the classification algorithm Neural Networks will be discussed.

Detection with Neural Networks has been done before for different problems, including the detection of fruits in images [3] the use of a NN with an Electronic-nose [10],detecting peaches using a color image [7] and the ripening classification on bananas with histograms [11]. Since trained NNs only use simple equations with weights and sigmoids (which can be converted to step functions later on), the computational power can be very low and easy to implement on a device. Neural Networks are widely used in classification since small errors do not matter for the output. A difficult problem would be training the network, which can be very intensive.

3.3.2. FEATURE DETECTION [8]

Feature detection is the method of first extracting features from the object you want to detect. A feature is an interesting part of an image, this can be an edge, corner, ridge or blob. By extracting the image's interesting features they can form a classifier. The classifier can be trained if multiple reference images are used to get a more accurate results. The input image is split up in small regions and within this region features are detected. If the features of the classifier match the detected features there is a a chance of an object. This chance depends on the amount of training data used and the matching algorithm [14].

3.3.3. TEMPLATE MATCHING

Template matching is the practice of having a reference image and matching the input against this template. The most common way of template matching is the use of cross-correlation [15]. This method is in the spatial

domain, there is also the method of transforming the template with the Fast Fourier Transform (FFT) to the spectral domain and preforming the cross-correlation of the spectral input image and the spectral image of the template [16]. Template matching works well if the orientation and shape of the template is the same as it is seen on the image. When the shape and orientation of the template differs it will not produce accurate results.
Instead of matching against a template or detecting features there is also the possibility of filtering away the rest of the image. Because the object that we want to detect is well defined in color and brightness we can try filtering on change in color and work with thresholding to filter away the noise.

3.3.4. EDGE DETECTION

Filtering methods for detecting regions of interests (RoI) is done by edge detection. Object boundaries are a powerful tool for detection, the edges stay visible under different lighting and changes in color. There are a lot of different methods for object boundary detection using edges [17]. The resolution of said edge detection methods are robust and with additional background filtering can provide accurate detection. The problem is when objects overlap, when there is no clear space between objects the edges can overlap and only one will be detected instead of two.

3.3.5. SPECTRAL FILTERING [13]

For spectral filtering the image has to first be transformed to the spectral domain with the Fast Fourier Transform (FFT). In the spectral domain filters of the pass and stop type are easily implemented. After creating a mask for the filter a Hadamard product is preformed and the resulting matrix is the filtered spectral image. This image can then be transformed back to the spatial domain to get the filtered image.

3.3.6. COLOR THRESHOLDING

With the properties of the HSV format, filtering becomes useful. Filtering with color components is called thresholding. With the Hue, specific colors can be filtered by creating an upper and a lower threshold. The same can be done for the RGB values. Since the stickers are blue, this color could be used in a blue color threshold. The problem for only looking at the Hue for threshold is that if the Saturation or Value are close to their maximum values the result will not be the color Hue says it is, but rather the color seen is black or white. Therefore we need to look away from the maximum and minimum of both the Saturation and Value.

3.3.7. QUATERNION FILTERING

Images can be analyzed using quaternions, quaternions can express images following the model in Equation 3.1 where b, c, d are the RGB values [18].

$$\begin{gathered} i^2 = j^2 = k^2 = ijk = -1 \\ a + bi + cj + dk \end{gathered} \tag{3.1}$$

where a, b, c and d are real number and i, j and k are fundamental quaternion units.
If we display a color image as a simple point with three dimensions, the RGB dimensions. The Quaternion Fourier Transform (QFT) [19] can create a spectral density image that contains all color information. Because Fourier transforms are computationally heavy a more efficient implementation like the Fast Fourier Transform can be used for 2-D spaces. There has been research of implementing a Quaternion Fast Fourier Transform that will use the same principle for 3-D spaces [20].

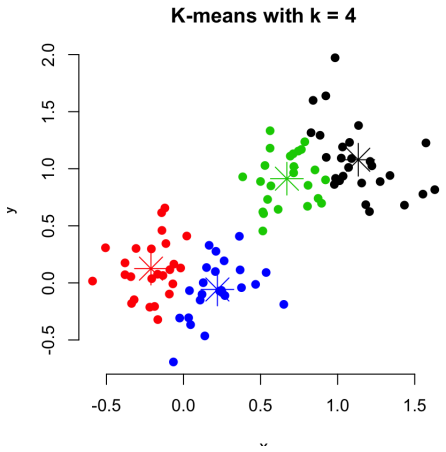


Figure 3.2: Example of K-means Clustering.

3.3.8. CLUSTER DETECTION

If using filter methods to create images only containing stickers, some type of cluster detection can be used to detect cluster and correctly locate and count the stickers. A lot of research has been done on the subject of cluster detection [21][22][23][24]. An input matrix for a cluster detection unit could resembles a binary NxM matrix with either ones or zeros. This matrix may have some noise. The goal is to filter the noise and to detect and locate the centroids of these clusters. Cluster detection can be done numerous ways:

- K-means
- Hierarchical Clustering

When the clusters can be seen as blobs, other methods like Breadth First Search (BFS) or Depth First Search (DFS) algorithms are possible as well.

K-means clustering is a way to classify and find clusters. K-means clustering is an unsupervised classification algorithm. From a given set of points in N dimensions, the algorithm searches for the centroids. With given coordinates of white points and the number of clusters, the algorithm accurately finds the centers of all clusters. In Figure 3.2, four clusters are shown with its centroid. Each iteration, the centroids of the clusters are updated until the differential distance of these centroids converges.

The downside of this algorithm, is that the number of cluster must be known. There are several ways to determine the number of clusters but it is found to be fairly difficult. Another problem is that every point is used and cannot be ignored. This means that noise cause problems in correctly determining the centroid.

Hierarchical clustering is an iterating algorithm where two clusters are merged each iteration until a criteria breaks the iteration. The initialization creates N number of clusters where N is the number of points. A map of every distance between the cluster is created. To link or merge two clusters, numerous linkage criteria can be used. If there would be no breaking criteria, the number of clusters would converge to one cluster. Choosing a stopping criteria can be difficult. A good stopping criteria would be a maximum distance between the two closest clusters at that time. Since the clusters are numerous pixels apart, this parameter should chosen easily. The downside of hierarchical clustering is the computation power, the complexity is $O(2^n)$ which is too difficult for large images with over 4 million pixels.

The BFS and DFS algorithms are not clustering algorithms but can be used under certain circumstances [24]. When all noise is filtered and the clusters are dense enough, a BFS or DFS algorithm can be very efficient for clustering. The difference of these algorithms are the path they walk trough the neighboring points.

3.4. FOREGROUND SEGMENTATION

The subject of removing the background leaving only objects on interest in the image is well researched. There are three main choices for segmentation: Background subtraction [25], Thresholding [26] and Edge based segmentation [27].

3.4.1. BACKGROUND SUBTRACTION

Background subtraction is used for detecting moving objects in frames from static image sensors. The fundamental approach is using the differences in the reference image or the "background image" to detect change and thus an object. This method is often used for videos feeds where moving objects are detected. To keep a good background model it has to be updated regularly to adapt to changes in luminance and background.

3.4.2. THRESHOLDING

Thresholding for image segmentation uses the difference in background and foreground. The difference between the two depends on the color format and input used. Several thresholding methods are available, to deal with changes in foreground and background there are adaptive threshold algorithms. One of these algorithms is the OTSU method [28]. This threshold selection methods creates a histogram of the grayscale values. The threshold is then chosen between the two highest peaks to get the best difference between main properties present.

3.4.3. EDGE BASED SEGMENTATION

By detecting the contours of objects it should be possible to find all the boundaries of the objects. If all the boundaries are found the next step is differentiating between the inside and outside of the object. To make this classification several parameters of the objects have to be determined. With such a classifier, the contours can be filled when the object matches the classifier or otherwise discarded.

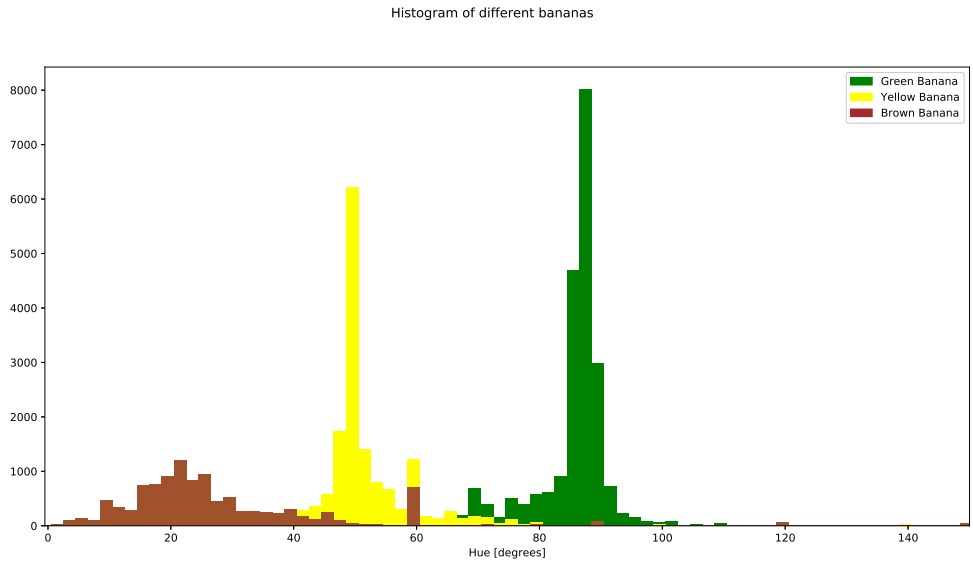


Figure 3.3: Histogram of a brown, yellow and green banana.

3.5. QUALITY OF FRUIT WITH IMAGE SENSING

A couple of papers link the color of fruits and vegetables to their ripeness [29], [30] and [11]. One method for looking at all the bananas at once is analyzing the color of the whole frame and determine the color distribution of the total image [31]. Other methods include red green ratios of the image [30].

3.5.1. HISTOGRAMMING

A histogram is a block graph which shows the occurrence distribution of certain values in a data stream. For images, this data stream is the image array and the values origins from a channel of each pixel, for example the Red value in a RGB image. These histograms can tell numerous things about images, for example object tracking [31]. If the Hue would be chosen for the histogram, the occurrences for every color would be known. The hue histogram of a brown, green and yellow banana can be seen in Figure 3.3.

3.5.2. HUE PEAK DETECTION

As seen in Figure 3.3, peaks can easily been found for certain banana examples where the colors are centered at one Hue. The examples given by the figure are examples of a green, yellow and brown banana. If we would use images which have mixed colors, these peaks would become less big. Brown color has a low saturation and is therefore spread more evenly throughout the Hue field. The brown color in the example also has a small peak in the far yellow area. This could create problems.

Peak finding algorithms are easy and efficient. To find the global peak, the maximum value should be found. When using images with mixed colors, multiple peak values become relevant. Peaks are defined by the following definition where x(n) is a value on an array x at point n:

$$x(n-1) < x(n) > x(n+1) \tag{3.2}$$

With noise, this definition creates numerous false peaks. It is therefor wise to implement a threshold value which either thresholds the minimum value of the peak itself or the minimum differential value between x(n) and x(n-1) or x(n+1).

3.5.3. HUE AREA DIVIDING

Saad [11] proposed to divide the Hue in three or more discrete parts. For example three parts consisting of brown, yellow and green. These boundary values are assigned depending on the corresponding histogram values. Simple algorithms could be applied to these values, but more interesting would be the use of neural networks. Since the input is discrete and the number of inputs is decreased significantly, making neural networks easier to implement. Neural Networks are good classification machines that respond well to different

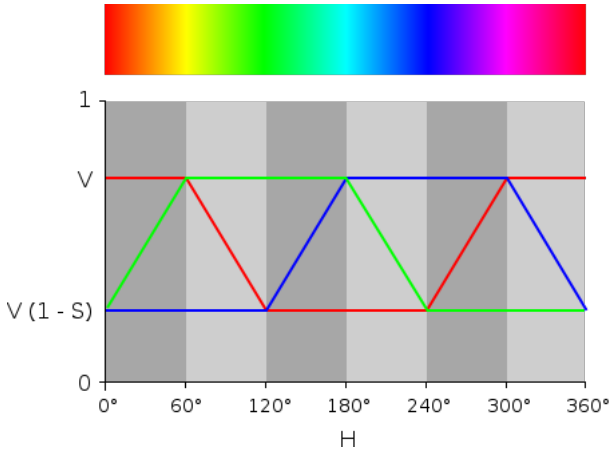


Figure 3.4: RGB values in perspective of the Hue.

inputs with noise or errors.

3.5.4. RED-GREEN RATIO

Another approach is computing the R/G ratio of the image. Where R is the value of all the red pixels added and G idem for the green pixels. This approach was proposed by Intaravanne [30] to determine the ripeness of bananas. Figure 3.4 shows the relation of the RGB values and the Hue. Since only the Hue values until around 150°are relevant, blue becomes irrelevant. The R/G value for green will be small while the R/G value for brown should be much bigger. The R/G value switches from lower than 1 to bigger than one at perfect yellow.

A possible problem for this method is background colors. If the bananas have a green background, the R/G value would be bigger than expected. This is why a foreground filter should be implemented.

4
DESIGN

4.1. DESIGN SYSTEM

The most important part of the design is the type of sensor that will be used to measure the three main features. In Section 3, the different kind of sensors are listed.

Electronic-nose
The Electronic-nose is discussed in Section 3.1.2. This device is accurate at determining the ripeness of the fruit and could only be used for this purpose. The problems with this sensor is the expensive price and invasive property since the fruit should be enclosed in a chamber.

RGB image sensor
The RGB image sensor is discussed in Section 3.1.1. The benefits of this device is the cheap price, the high accuracy, the flexible position and the multi-functionality. A problem with a camera is the power usage.

Spectral and thermal image sensor
The spectral image sensor with sub-option the thermal image sensor is discussed in Section 3.1.1. Since fruit is an organic material, it can easily be detected with the spectral and thermal image sensors. With this information, the same image processing as for the RGB image sensor could be used except for the histogramming since colors are not measured. Unfortunately, these sensors are expensive and complex and don't provide an overall solution.

Pressure sensors
The pressure sensors are discussed in Section 3.1.4. These sensors can measure the weight disctribution of the shelf. With this information, the fruit quantity and shelf quality can be determined with accuracy. This sensor however is invasive to the shelf.

Infrared sensor
An array of infrared sensors is discussed in Section 3.1.5. When placing these sensors on the side of the shelf, obstructions can be detected. These obstructions can then determine the quantity of the fruit and the quality of the shelf. This implementation is invasive and could be inaccurate.

4.1.1. DECISION ARGUMENTATION

To determine which sensor needs to be chosen, an overview is given in Table 4.1.

Sensor	Accuracy	Cost	Installation	Power	Multifunctional
Electric Nose	high	high	invasive	unknown	1
RGB sensor	high	low	non-invasive	medium	3
Spectral sensor	high	high	non-invasive	medium	2
Weight Scale	high	low	invasive	low	2
Infrared	low	low	invasive	low	2

Table 4.1: Overview of the sensors

From this table, all the sensors that have the invasive property are rejected. High power sensors are rejected as well, since the sensor should be able to run on batteries. This leaves two sensors: The RGB sensor and the spectral sensor. The RGB sensor provides the solution for all of the features and is the cheapest sensor and will therefore be used.

4.1.2. DESIGN BLOCK

With the RGB image sensor, every feature can be monitored. To determine the quality of the fruit, a foreground extraction method should be implemented since only the foreground should be evaluated. This image containing only the bananas is then passed to an unit which determines the quality based on the color of the fruit. With this foreground segment, the quantity of the fruits can also be determined using the total foreground surface compared to the background. To determine the quality of the shelf, the position of the fruit is important. When placed correctly, the stickers on bananas are visible. Since these stickers are an important feature of the shelf quality, they should be detected and counted. The block design for the RGB sensor is shown in Figure 4.1.

The input will be a NxMx3 matrix representing a RGB image. This image will be created by the image sensor. The output will be the number of clusters found, the foreground percentage and the green, yellow and brown percentage in the image.

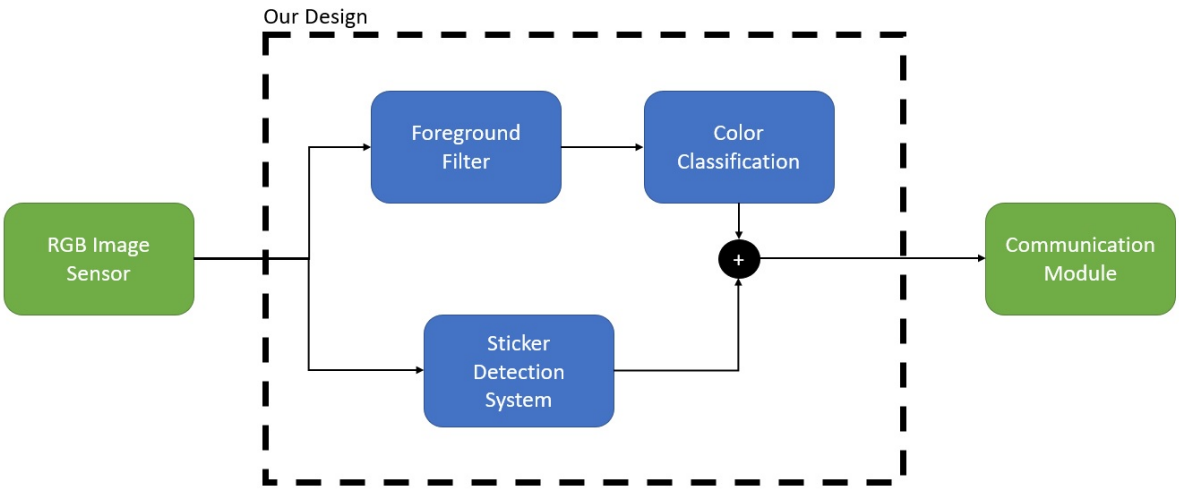


Figure 4.1: Block Diagram of the design.

4.2. FILTERING

The goal is to detect the labels on bananas on the shelf of a supermarket. We use filtering for the detection of the stickers on bananas. These stickers give information about the placement and orientation, this is used to determine the quality of the shelf.

4.2.1. WHAT IS BEING FILTERED

All of the bananas are supposed to have a sticker from the manufacturer, the stickers in our case are blue. The image taken by the camera is represented in several color formats each with their own characteristics. To determine what color best to use for filtering we compared the generated images 5.2. The RGB arrays displayed as grayscale do not offer a clear distinction between the stickers and the surrounding area. Converting the original image to grayscale has more gradient than RGB, but definite differences in color are not detectable because all colors are merged together. The HSV color format has some interesting properties, all color is encoded in the Hue, if we filter for changes in this domain we can see changes in color. The Saturation domain of HSV determines the saturation of the color. The saturation of the label is different than the saturation of the of surrounding area (the surface of the bananas).

To achieve the best contrast between the background and the stickers the Hue and Saturation from HSV are used.

4.2.2. AVAILABLE FILTERING METHODS

In Chapter 3 several filtering methods are introduced for filtering images. There are several methods available for the detection of change in color, each with it's own properties and effectiveness. The most important feature is the robustness, the solution has to provide results in a lot of different sceneries. The working environment is variable but we can assume some similarities. The background of the shelf is usually black, this is good for working with the saturation, because there is little change in the background. We have a sample photo of what an average shelf looks like in Figure 5.1. The photo is not shot from the desired angle but everything that we want to look at is in view.

4.2.3. COMPARE AVAILABLE METHODS

There are methods that can use RGB images and provide accurate information about the color image with a single transformation. In the technical review in Section 3.3.7 look at converting color images to quaternions. This method might use more of the information but the math behind the Fourier Transform in the quaternion domain is advanced. If this was the only subject of our bachelor thesis we might be able to do it, but for the application it will be too difficult. The computational power needed to perform a 2D Fast Fourier Transform (FFT) on a high resolution image (2000x2000) is significant and would require 4000 1D Transformations [32]. The main advantage of using quaternions to filter is that color information is preserved. This makes it possible to inverse the transformation and get a color image as the result. difficulty is that the needed amount of operations needed to calculate the FFT of quaternions and the difficulty of the interpretation of the results.
The next methods all use a single domain for their calculations/transformations. We look at grayscale, hue and saturation as the domains to work with. For images that are 2D there are multiple options to filter, we can transform the image to a spectral image with the Discrete Fourier Transform (DFT) or a different implementation the FFT. The other option is to use a 2D filter.
For images that are 2D there are multiple options to filter, we can transform the image to a spectral image with the Discrete Fourier Transform (DFT) or a different implementation the FFT. The other option is to use a 2D filter, the 2D filter is faster than the DFT. The FFT is a efficient implementation of the DFT.

The DFT has a difficulty of N^2 operations where N is the number of points. The FFT needs $N\log_2 N$ operations, the larger N is the more efficient the it becomes. The 2D versions of the DFT and FFT perform the operations for each row, and then each column, thus having the original operations multiplied by $N+N$ for a square matrix. If we compute the DFT and FFT of a 2000x2000 image (N=2000), it needs $N+N = 2000+2000 =$ 4000 1D DFT or FFT operations. The number of operations needed for computing the DFT is seen in Equation 4.1 operations. the FFT of the same image uses significantly less operations as seen in Equation 4.2. So the FFT needs 0.55% of the operations the DFT uses.

$$N^2 = 2000^2 = 4 \cdot 10^6 4000 \times 4 \cdot 10^6 = 16 \cdot 10^9 \tag{4.1}$$

$$Nlog_2 N = 2000 log_2 2000 = 220004000 \times 22000 = 8.8 \cdot 10^7 \tag{4.2}$$

A 2D filter shifts a kernel, a specific type of square matrix over the image and calculated the sum of the square matrix multiplied with the image. The amount of operations needed to compute the total image strongly depends on the size of the kernel. In a 2000x2000 image there are $4 \cdot 10^6$ data points, we need one matrix multiplication for each pixel. For a 3x3 kernel the number of operations needed is given in Equation 4.3. The difficulty of a spacial filter is given by the R^2 where R is the width or height of a square kernel. If the kernel becomes larger it will be more efficient to calculate thefiltered result with a fourier transform of the kernel.

$$2000 \times 2000 = 4 \cdot 10^6 4 \cdot 10^6 * 3^2 = 3.6 \cdot 10^7 \tag{4.3}$$

There is a difference in result of the different filter methods. In the spectral domain it will be easy to apply an ideal filter that passes or blocks certain frequencies. With the spatial filter it is hard to do the same, in the spatial domain we more often look at edge detection so change of pixels that are close to each other instead of the total image at the same time. This produces different results but can both be used for our purpose.
The goal of the filter is to detect the stickers that are put on fruit that resides on the shelf. This is not the only demand for the system, we want to make the system work on a battery so power consumption is also important. Power consumption is strongly related to execution time of the program. The shorter the active time of the program the less power is used. There is a trade-off between resolution and speed that needs to be considered for the final implementation. The goal is good enough detection in the shortest time. The two methods that have the potential to quality are the FFT filtering and the spatial filter.

The last option only later came into focus when working with the HSV color format. When the foreground is already segmented from the background only the bananas with the stickers are left in the picture. The only blue left in the image then are the stickers. If we can filter away all colors except for blue then the only thing left should be the stickers.

We will implemented three methods and make a choice for the final implementation based on the test results.

4.2.4. DESIGN OF CHOSEN METHODS

Each filter method requires it's own type of input and filter region.

For the spatial filter there are several kernels that were designed for edge detection and removing noise [33]. We have to look at what the best input is for the filter to get the best detection result for the stickers. Because we want to look at edges or changes in the color domain it seems logical to analyze the HSV color format. In this format the color is in one array and the saturation in one array. With spatial places where the object is will have very small or very big values compared to the surroundings, to get the best resolution between sticker and the rest we can use a threshold value that determines if it belong to a sticker or whether it belongs to the background. This gives two possible outputs, '1' what is a sticker and '0' what is not a sticker. The output should then be a bitwise array of the same resolution of the original image.
For the frequency domain filtering there is also the choice between inputs what produces the best results after filtering. The same argument can be made for this method to use the HSV color format because all the relevant changes can be examined with more resolution and within one array. To analyze the frequency domain we first have to convert the input array to the frequency domain. In the frequency domain we create a filter, this filter has to be the same size as the converted input to preform a Hadamard matrix product. After this product we have the filtered image in the frequency domain, to retrieve the information in the spatial domain to extract the locations of the object we have to preform the inverse transform.
If we filter the image for color the desired range needs to be determined. Blue has a color angle that could range from 180°to 300°. Because of the note in HSV section in the technical review also the saturation and value need to be assessed. Because they should not be near their maximum or minimum.

4.3. CLUSTER DETECTION

The cluster detection receives a binary image containing noisy clusters at the locations of the stickers. In order to neglect this noise, a smoothing filter should be implemented to create blobs at the locations of the stickers.

4.3.1. BLOB FILTERING

2D correlation

2D correlations are widely used in image filtering. The filter moves over each pixel and adds the multiplications of the filter 'pixel' overlapping the image pixel. This creates a smooth image without noise if thresholding is applied in the end. A flaw of the system could be the time when the size of the image or filter increases.

Integral imaging

A different method for filtering noise in the image is to convert the image to a integral image. This performs an integration on the image from the top left corner to the bottom right. By examining the edges at the four corners of a rectangle the mean value in the rectangle can be determined. If a cluster is present in the rectangle the mean value will be large, if there is no rectangle the mean value is small. Calculating the integral is a easy operation and does not require a lot of time. The building of a new matrix with the edge values is inefficient as four additions and four multiplication are needed per pixel.

Both options should be suitable for the task. Since it is unpredictable which method will be faster, both will be used for testing the system.

4.3.2. BLOB DETECTION

The blob detection unit will receive a binary image from the sticker filter. This image will contain 'blobs', clusters with directs neighbors.

Three cluster detection algorithm were discussed in Section 3. One simple blob detection method was discovered during implementation and will be explained briefly.

K-Means clustering

The first option for cluster detection is K-means clustering which is discussed in Section 3.3.8. K-means clustering is a fast and easy method to detect clusters and its centroid. A big downside of this method is the unknown amount of clusters, this number is a variable. Since we do not know the number of stickers as well, additional algorithms should be implemented to find this number which increases the complexity.

Hierarchical clustering

Cluster detection can be done with Hierarchical Clustering which is discussed in Section 3.3.8. The problem of the unknown number of clusters is terminated with this implementation if the stopping criteria for merging clusters is chosen properly. However, the method is complex and the stopping criteria for merging is sensitive. The robustness of the system will be doubtful.

BFS/DFS algorithm

The BFS or DFS algorithm (Section 3.3.8) can be used since the clusters are defined as blobs. The complexity of these algorithms is small and would be easy to implement.

Simple blob detection

During implementation, the simple blob detection algorithm was found. The speed is by far the best of all the functions, since the image is divided in different small images and then evaluated for blobs by counting the values in that section. A simple blob detection function could be the most fitting algorithm. In addition, the filter options are extensive, where it is possible to filter on blob size, convexity, thresholds, circularity, inertia ratio and minimum/maximum distances.

When evaluating every blob finding method, the BFS and simple blob detector will be chosen for testing. The K-means clustering is lacking, since the number of clusters are not known. The algorithms to determine the number of clusters are not accurate and time consuming if the clustering algorithm would be ran multiple

times using various estimations. The Hierarchical Clustering method wil fail since the stopping criteria for the merging stage is too sensitive for different images. It would also become time consuming if the image would become larger or would have more '1' pixels. Where the K-means and Hierarchical Clustering lack the execution time for large images, the BFS and simple blob detection would suffice.

4.4. IMAGE SEGMENTATION

Image segmentation is used to remove the background of the image and only display the objects of interest. This result is then used to create a color histogram to make a estimation of the quality of the product on the shelf. It is important to remove the background and keep all important information. The background or the color of the shelf is in most cases black, so we take this as a reference for filtering.

For foreground segmentation all the available inputs can be used. Most methods discussed in the theory use grayscale images for image segmentation. Because the objects on the foreground have the predefined color from brown to green, color can be used as a criteria for segmentation. The background that should be removed is black, the problem with this surface is that it reflects light. The amount of light reflected depends on the angle of the camera. On the corners of the shelf a lot of light is reflected in the direction of the camera. Because of this reflection the background on these spots can look very light instead of black. This makes it difficult to only use grayscale for foreground segmentation. The other information that is available is the image in HSV or RGB format. The fruit to detect has a specific color, when the Hue from HSV is used for segmentation the corresponding Hue value can determine the threshold.
There are several methods for image segmenting but one of the most generally used is thresholding [26]. Because the background is usually very uniformly colored the foreground can be detected using a threshold. The problem with this detection method is that there needs to be a clear difference in shade between foreground and background.

The other option is background subtraction, there are several options for background subtraction [25]. Our solution should be able to deal with the changes in lighting, the other requirement is low power, this implies something computationally light. The methods reviewd by Piccardi [25] all use grayscale images. This works because often the main differentiator between background and foreground is luminosity of the objects. In our situation artificial lighting from the ceiling causes a lot of the objects to be in the shadow of others. This makes it hard to differentiate between foreground objects and the background. For background subtraction you need information about the background, to make the system as non-intrusive as possible. It will be hard to get a clear picture of the background, without interrupting usual business.

The best method for the situation is thresholding because of the clear difference in color in combination with difference in lighting of the background and foreground. If only one of the aspects is used there will not be more accuracy than with background subtraction. But if we can combine thresholding in the hue and saturation domain it should be possible to deal with the lighting problem. Thresholding does not need calibration with a background and can work in all supermarkets that fit the model.

4.5. COLOR CLASSIFICATION

In order to determine the quality of the fruit and the quantity of the fruit, color computations need to be made.

In Section 3.5, three possible options for the quality and quantity measurements have been discussed.

4.5.1. QUALITY MEASUREMENTS

Histogramming

Two of the methods make use of histogramming in the Hue space of the HSV image (Section 3.5.1). One of the method involves peak finding in this Hue histogram. The peak will often tell what the dominant color is, and this could indicate what quality the fruit has. However this method is sensitive for peak errors or even distributions of colors. The other method involves dividing the histogram in three parts: A green, yellow and brown area. From these areas the counts are added and the ratio of the area with the total count is calculated. This method tells us more about the overall distribution, which will be more accurate since all the bananas are taken at once. It however does require more computational power.

Red-green Ratio

The other method describes the proposal of calculating the Red/Green ratio [30] discussed in Section 3.5.4. Here, all the red and green pixels are counted and then divided by each other. It has been shown that the

difference can been seen as a high value for brown bananas and low value for green bananas. This method could be inaccurate when using multiple bananas in one picture.

Since the Hue could describe the Red/Green ratio as well, it would be more convenient to use this histogram. The R/G ratio only uses two colors, while the Hue uses every color possible which would result in greater accuracy. On top of that, a histogram provides more information than a R/G ratio and will be more easy to evaluate. It is therefore that the histogramming with the three color areas will be used for testing.

4.5.2. QUANTITY MEASUREMENTS

It would be possible to measure the quantity with histogramming. The background of the image was thresholded to black. This black resembles a Hue of 0°. If all the counts, except the Hue at 0°counts would be added, the total counts for the foreground would be known. If these counts where divided by the total number of counts, including the background at Hue is 0°, the percentage of foreground would be known. For this reason, the histogramming implementation would suffice for testing.

5
IMPLEMENTATION

The platform chosen to implement the methods and test the system will be Raspberry Pi in the Python programming language. The Raspberry Pi delivers enough calculating power for image processing tools while being relatively cheap in price. The Raspberry Pi also has dedicated hardware, for example the Pi Camera. Python is an easy programming language with widely available libraries which will help making the implementation easier. Some libraries that could come in useful:

- PiCamera [34], this is a library which can easily take images or record videos with the Pi Camera with simple functions.
- NumPy [35], an extensive library dedicated to array calculations and other algorithms which involve arrays.
- OpenCV [36], a library specialized in image processing calculations.

The code in Python which are relevant to this implementation is documented in Appendix D.

5.1. INPUTS AND IMAGE SENSOR

Retrieving an image from the sensor is the first step in the system. The image sensor is an 8 MP (megapixel) color sensor. Because the distance from the camera to the shelf is not constant the region of interest (RoI) is not the same for each setup. To make sure we get the maximum resolution for the RoI the maximum resolution of the camera is used. Before this image is given to the rest of the system it will need to be cut to only contain the banana shelf. To calibrate the cutting of the image to the RoI this has to be done manually for now. There have been no tests where the system was placed in the final location.

The Pi Camera provides a 3280 by 2464 pixel resolution with horizontal FOV of 62.2°and vertical FOV of 48.8°. This will grant an effective area of 1.2 meters in length per height unit and a 0.9 meters in width per height unit. For example, if a shelf of 3 by 2 meters would have to be monitored, the minimal height of the camerawould be 2.5 meters. This height could be altered with additional lenses.

Since the shelf is illuminated by big fluorescent tubes, the images could be too bright for the camera. An important property of the PiCamera involving brightness is the (International Organization for Standardization) ISO value. On the PiCamera this value can range from 0 to 1600 where 0 is very dark while 1600 is bright. In Figure 5.1, the difference in ISO is shown for the range of 50 to 150. During testing, multiple images with different ISO values will be used to test effectiveness per ISO value.

Figure 5.1: The difference in brightness for numerous ISO values.

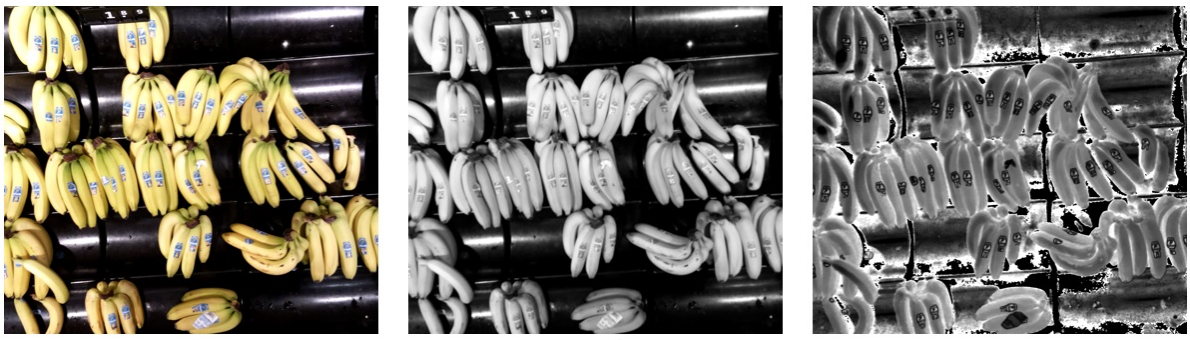


Figure 5.2: The input images of the thresholds with from left to right: The original in RGB, grayscale and saturation.

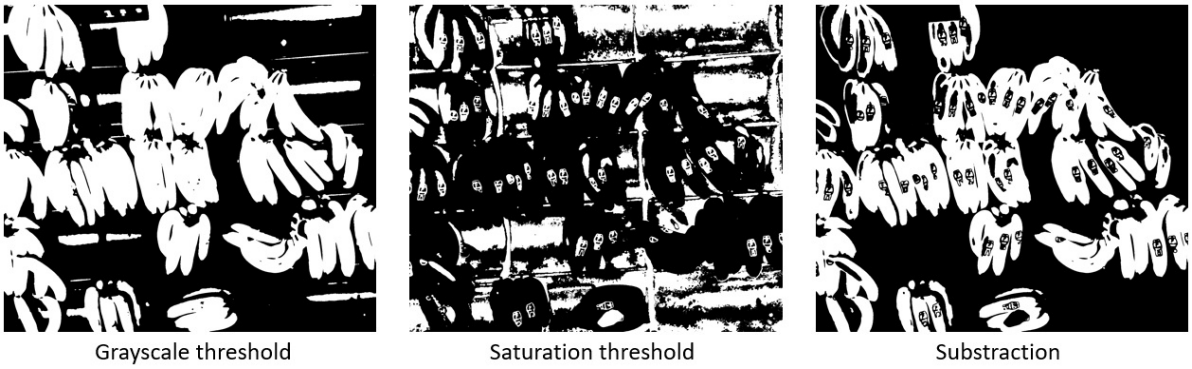


Figure 5.3: The resulting images after thresholding with white being past the threshold and black under. From left to right there is the grayscale threshold result, saturation threshold result and the subtraction of the results saturation subtracted from grayscale.

5.2. FOREGROUND SEGMENTATION

Foreground segmentation is applied to the original image. The used color formats for the image are from HSV the Hue and Saturation. The RGB image is also converted to grayscale to use in thresholding.

The domains used for thresholding are grayscale and the saturation of the original image. before thresholding is applied first the input is preprocessed. To remove some of the noise in the image and smooth surfaces a Gaussian filter is used. To get enough blurring to remove the noise around the edges of the objects a filter size of ksize = 11 is used. The Gaussian kernel is created using the openCV function getGaussianKernel [36] get the coefficients, this function calculates these following the Equation 5.1.

$$G_{\mathrm{i}} = \alpha \cdot e^{-\mathrm{i}-\frac{\mathrm{ksize}-1}{2}^2}{2\sigma^2} \tag{5.1}$$

Where i = 0, ..., ksize -1 and α is the scale factor chosen that $\sum_{i=0}^{\mathrm{ksize}-1} G_i = 1$. The value for σ, the Gaussian standard deviation is automatically calculated for ksize with $\sigma = 0.3 * ((\mathrm{ksize}-1) * 0.5 - 1) + 0.8$. This kernel is then correlated with the original RGB image to give a smoothed version. For thresholding we use the grayscale and color converted HSV versions of the smoothed image. The original, grayscale and saturation of a test image are given in Figure 5.2

To find and determine the correct threshold values of the foreground, we first take a look at the picture in grayscale. As seen in figure the bananas are relatively light compared to the background. To determine the correct threshold value we use the method named after Otsu explained in Section 3. A typical threshold value for the test images is in the neighborhood of 105. The resulting image of thresholding with 105 can be seen in Figure 5.3.

As seen in the resulting image there is still some background of the image included that is bright because of the reflection from the lights. In the saturation image in Figure 5.2 the background at the spots is not bright and thus we can threshold this image. The value used for this threshold aims to extract the dark spots of the saturation where the grayscale is bright. With a threshold of around 65 the banana's are not detected but all the required background is included. The result of this threshold is displayed in Figure 5.3.

Figure 5.4: The resulting foreground in RGB.

To extract the foreground using both images we subtract the saturation threshold from the grayscale threshold and get an estimate of the foreground seen in Figure 5.3. There is still some noise in the resulting image and to remove it a morphological transformation is applied, we first erode the image. This decreases the size at the edges, and afterwards we dilate the image. Dilating extends the image at the edges. Small points or edges are thus removed. The resulting image is used as a mask for the original image to create a color view of the foreground. The result is given in Figure 5.4

5.3. OBJECT DETECTION OF STICKERS

The detection accuracy of stickers on bananas varies with changes in the input images. Because of the changes in environment and each detection method having its pros and cons for dealing with those changes. Three methods have been implemented to test for the final setup:

- A spectral filter in the Fourier domain that filters the saturation from HSV.
- A small kernel 2D filter that spatial filter with a Laplacian kernel.
- Thresholding combined with color matching in the HSV domain.

All methods have the same goal to detect the stickers and provide a binary map to the cluster detection module. Because the stickers are always present on the foreground and never on the background we use the already segmented image for detecting the stickers. This is needed because the background has a lot of differences in lighting because of reflection. The foreground segmentation is adapted a bit to make sure the labels are present in the mask. This is done by morphologically dilating the edges of the mask. A bit of background around the foreground is included in the mask but this is not a problem for the sticker detection.

5.3.1. FOURIER DOMAIN FILTERING.

The input used for the spectral filtering for the stickers is the saturation. Because most of the elements we want to filter away have low frequency components we use a bandstop filter for in the low frequency region. Instead of watching peak values we look at the lowest values, this is why the filter removes the objects to detect. The origin where frequency is zero is still important for information so we filter from 2 to 3 with a circular ideal filter. The equation used to create the ideal circular filter is given in Equation 2.

After filtering both the saturation the results is made binary with a threshold to get the regions of the stickers. The threshold value used is around 20 on the scale from 0 to 255. Because the stickers will always be on the bananas and never on the background we remove any regions outside the foreground. For this we use the foreground segmentation mask. We preform a bitwise and with the mask of the foreground and the result from the filter. The accuracy of the detection of stickers depends on the changes in saturation. If the image is overexposed then filtering for saturation produces less results.

5.3.2. 2D LAPLACIAN FILTER

With a 2D Lapacian style filter it produces a type of high pass filter for the saturation of HSV. To make sure the value during the filter does not overflow the values in the kernel are divided by the sum of the elements. We

compute the correlation of the kernel and the saturation with the openCV filter2D function [36]. After this we use a threshold to get a binary image of the regions of interest. Because of the same reasons as with spectral filtering, that no stickers can be found outside the foreground the same process is applied to the result of the Laplacian filter. We subtract the background mask that is enlarged from the result to only get the stickers. There is still some false positives in the result even after the background subtraction. These small errors can be detected in the cluster detection and are acceptable.

5.3.3. COLOR MATCHING HSV

The stickers that we want to detect always are always blue, if no foreign objects are present they should be the only blue elements in the image. This makes it possible to locate the stickers by filtering the color, the easiest way to filter color is using the HSV domain. The region for blue in the HSV domain is around 180° to 300° this is on a scale from 0° to 360°. The openCV library region is from 0° to 180°. Thus the region of blue for openCV is from 90° to 150°.

The stickers are always present in the foreground, instead of looking at the original image background we can use the segmented foreground iamge.

The result of every filter is seen in Figure 6.2. It would be possible to combine the results or choose the one with the best performance with respect to processing time and accuracy. Some noise in the in the result is acceptable because in the following module the detected clusters are classified based on their area removing small blobs.

5.4. CLUSTER DETECTION

5.4.1. PRE FILTER

To exterminate small misplaced clusters caused by items with the same features as the stickers, a filter will always be needed. In Section 4.3, both filters will be tested. In python, the OpenCV [36] library offers numerous dedicated functions for these methods:

- cv2.integral(image)
- cv2.filter2D(image, -1, filter)

INTEGRAL IMAGE

The integral image is calculated with the openCV function Integral. This creates a map with large integers that are the sum of the pixels in the image. The sum of the pixels in the window is given by Equation 5.2.

$$Sum = A + D - B - CMean = \frac{Sum}{size * size} \tag{5.2}$$

where A, B, C and D are the vertices of of the window. To get the mean of the sum of the pixels the sum is divided by the total number of pixels in the window. This is the same as the square of the size of the window.

2D FILTER

In OpenCV's documentation, the complete function is defined by:

```
1        dst = cv2.filter2D(src, ddepth, kernel, dst, anchor, delta, borderType)
```

Only three parameters are of interest for the implementation. The src is for the input image, the ddepth is static at -1 and will keep the resolution of the input image, which will be 8 bit. And lastly the kernel, which is the filter.

The function implements a correlation with default values:

$$dst[x,y] = \sum_{x_i=0}^{cols} \sum_{y_i=0}^{rows} kernel(x_i, y_i) * src(x + x_i + 1, y + y_i + 1) \tag{5.3}$$

Where cols and rows are the number of columns and rows of the kernel. The $kernel$ is the filter matrix, x is the x coordinate of the image and y is the y coordinate of the image.

The kernel will be a NxN matrix of ones where N is the filter size. This filter size will be variable during testing. This ones filter will count every value within the size parameter to the pixel position. When dealing

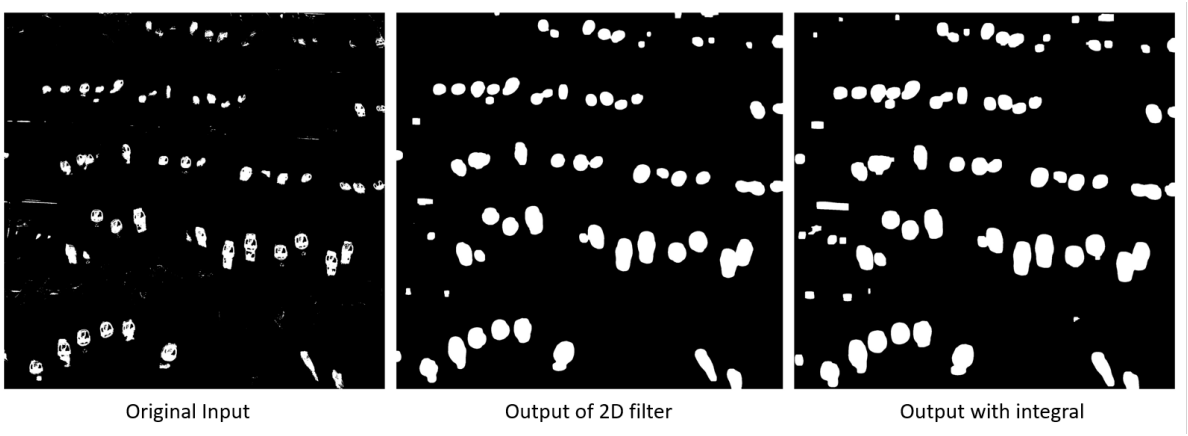


Figure 5.5: Input and output of the two methods used during implementation design. Left is the input binary image. The middle image is the output of the 2D filter with filter size of 39 and threshold of 50.000. The right image is the output of the integral image method with filter size 39 and threshold 20.

with a large filter, these values can become unpredictably high. A solution to this problem is normalizing the output. This can be done by dividing the ones matrix with the squared value of the filter size.

After this filter operation, the image should be converted to binary. Since we do not want small cluster, the filtered image needs the following thresholding computations:

```
1        image_f(image_f<threshold) = 0 #this negates the small clusters.
2        image_f(image_f>0) = 1 #this finalizes the binary image
```

5.4.2. CLUSTER DETECTION

Two types of cluster detection will be tested. The simple blob finder and the BFS algorithm. The simple blob finder has a python implementation while the BFS algorithm will have to be programmed without dedicated functions.

BFS ALGORITHM

The BFS is initially designed for connected structures with one tree of connections. Since the image could be seen as more trees, a solution for this problem should be implemented. For every cluster a unique value could be assigned starting with 2 when dealing with a binary image where 0 and 1 are already used. The algorithm will check every pixel for an '1'. When an '1' is found, a new tree is started. At this moment the BFS algorithm is initialized. For this first point, the value is changed to the next unique value. At this point surrounding pixels are evaluated for an '1'. This position is then added to the queue. When evaluation of this point is done the point is deleted from the queue and the next point in the queue is evaluated. This iterates until the queue is empty which means the cluster is found completely and assigned the unique value. After this, the unique value is incremented by one. When every pixel is evaluated for an '1', the algorithm is done and no '1s' should be present in the new array.

The number of clusters can then be simply calculated by finding the maximum value and subtracting '1'. The function $numpy.where(array, n-1)$ could be used to find all the points of cluster N where n is the unique value of the cluster. Numpy is a simple array library for python. To find the x centroid position of this cluster, all the x coordinates can be added and divided by the number of points in the cluster. The same operation can be done for the y centroid position.

Iterating through all clusters provides an array with all the positions of the centroids of the clusters.

SIMPLE BLOB FINDER

The dedicated function of OpenCV for the simple blob finder uses the following function in the corresponding syntax:

1. $parameters = cv2.SimpleBlobDetector\_Params()$. This creates an object for all the parameters that can be used to filter the blobs.

2. $detector = cv2.SimpleBlobDetector\_create(parameters)$. This creates the detector object with correct parameters.

3. $keypoints = detector.detect(image)$. This creates a keypoints object, including the coordinates, the diameter along with other unimportant information about the found clusters.

The variable names of the coordinates and size are defined as pt and $size$.

Parameters The parameter object has numerous options for filtering:

1. Color: blobColor [0 255]

2. Size: minArea [0, ∞] and maxArea [0, ∞]

3. Shape, which has multiple suboptions:

   (a) Circularity: minCirculatity and maxCircularity [0, 1]. Where a circle has the value one and a square the value 0.785.

   (b) Convexity: minConvexity and maxConvexity [0, 1]

   (c) Inertia Ratio: minInertiaRatio and maxInertiaRatio [0,1]

The input image of the simple blob finder should be a grayscale array from 0 to 255, the binary image from 0 to 1 is therefore compatible.

5.5. HISTOGRAMMING

As an input, the extracted foreground image will be provided. For this histogram the Hue will be of most importance. The histogramming functions will be provided by the library of NumPy which is specialized in array computations.

In order to use the $histogram$ function, the Hue image will have to be transformed into a single row array. For this the $Array.flatten()$function of NumPy will be used.

The two input arguments of relevance for succecfully create a histogram array will be the flattened array and the possible range in that array. The hue will be in range from 0 to 179. This is a standard in the OpenCV library.

The two output arguments that will be created are an array with the counted occurrences per value ($counts$) and an array containing the range of the histogram ($bins$). An example of the output is seen in Figure 3.3.

With this information, four features need to be calculated:

1. The percentage of the foreground counts in relation to the total image

2. The percentage of brown counts in relation to all the counts of the foreground

3. The percentage of yellow counts in relation to all the counts of the foreground

4. The percentage of green counts in relation to all the counts of the foreground

To determine the counts of the foreground are the counts of yellow, brown and green combined. The background counts are all stored in Hue = 0 since the RGB value [0,0,0] is converted to a Hue of 0. The range of the three colors are defined as [0,15] for brown, [16,25] for yellow and [26, 60] for green.

The total counts of these ranges are added and a percentage is calculated, which will describe more about the ripeness of all bananas as a collective.

6
TESTING AND RESULTS

Testing the system of the fruit monitoring system needs to apply to different test criteria, these are discussed in the first section. When all the criteria are known, the test setup needs to be designed. This could be with the implementation of a GUI or a multi-stage test design. This is discussed in the second section. The third section will show the test results and observations made during these tests. At last, these test results and observations will be discussed.

6.1. TEST CRITERIA

The test criteria need to be representative to the real situation. This real situation is located in a supermarket. Some of the important situations possible in supermarkets are: an empty shelf, bad product placement, variety of fruit quality and objects blocking the view. From these situation, test criteria can be defined:

1. A realistic range of fruit quality. For bananas this is from green to brown spots.
2. An empty shelf as test sample.
3. The test samples should be test on different heights and angles.
4. The images should be locally saved for further evaluation and testing.
5. The test samples should include blockades of the sensor.
6. The intermediate signals should be saved

6.2. TEST SETUP

The test setup will be done in 2 stages. One stage to acquire the test samples at a real supermarket. These test images need to be taken multiple times in order to obtain multiple scenarios. For every time of the day, the morning of the opening, during lunch hours, diner time and closing time, some test images need to be taken. To create a representative image, the real Pi Camera will be used to create these images.

Since the Raspberry Pi does not have an integrated interface, a new script and interface need to be made in order to create these images. This setup could be very basic with only a button and a LED. The button has the use of actually taking the image, and the LED will be turned on when the system is ready to take an image.

The second stage is actually testing the functions of the system. Here, the test images of the first stage will be used as input for the total system. In order to evaluate every system step, the intermediate images need to be saved. An useful tool would be a simple GUI which easily inputs an given image and shows the intermediate images in a grid with the option to save these images.

6.3. RESULTS

The test results will be shown per operation. This will allow for a better evaluation of the total process. In total, there are 2 main operations: the sticker detection and the foreground histogramming.

6.3.1. STICKER FILTER

Three methods to filter stickers are implemented, the method that produces the best results is seen in Figure 6.2. This method uses color thresholding in the HSV domain to detect the blue labels on the foreground of the image. Noise is first removed by using a Gaussian smoothing filter. There is blue detected in the black of the background but because we only look at the blue on the bananas these results are filtered away. All the stickers from the original images in Figure 6.1 are visible on in the filtering result. The other two methods are

(a) ISO 50 (b) ISO 150

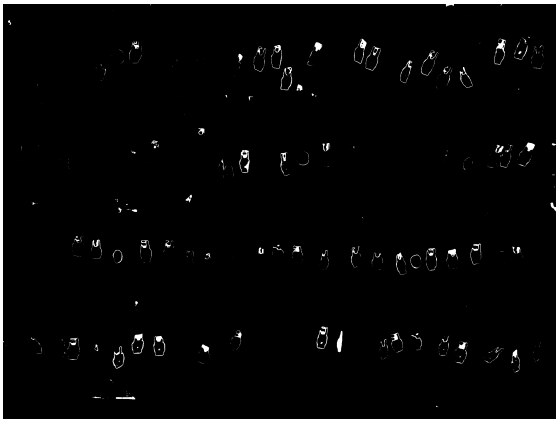
Figure 6.1: Original test image with ISO of 50 and 150.

(a) ISO 50 (b) ISO 150

Figure 6.2: Sticker filter results with ISO of 50 and 150.

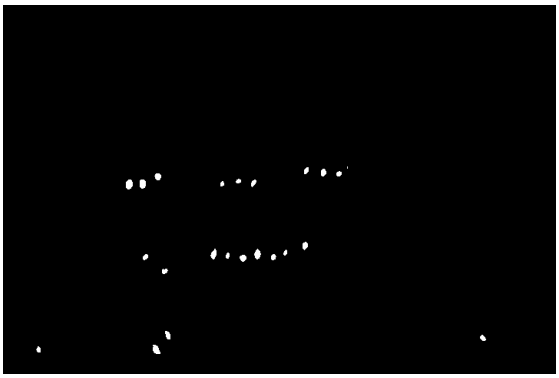
visible in Figure 6.3, these are the 2D laplacian and the spectral filter method of the original image seen in Figure A. All of the methods do detect the stickers. The results of these two methods do differ when the ISO is different. The filtering methods produce the best results with an ISO of 100, this ISO is seen in the Figure 6.3. The results of the 2D Laplacian filter are better than the spectral filter. One idea could be to combine the results of multiple filtering methods to have an additional check and calculate a final result. The result of both the 2D filter and the spectral filter is still less than the color threshold method. The reason for this is because these two methods focus on change, this change is only detectable at the edges of the objects. This is why only the outlines of the stickers are visible. This causes problems to detect clusters because the outline will not create just one cluster if it is fully detected. The color threshold method doesn not have this inaccuracy because we focus on the color blue, this color fills the whole object and creates a good cluster.

(a) 2D Laplacian filtered and thresholded

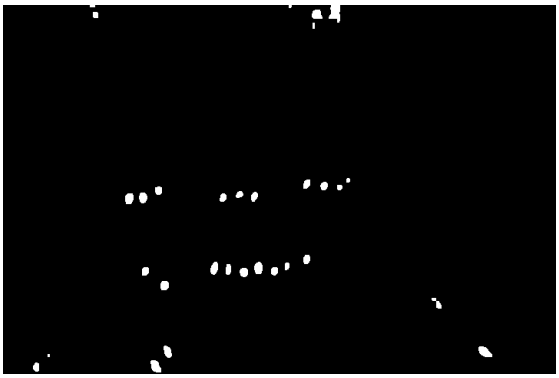


(b) Spectral filtered bandstop and thresholded

Figure 6.3: Sticker filter results with ISO 100 and left the 2D Laplacian filter and right the Spectral filter.



(a) ISO 50



(b) ISO 150

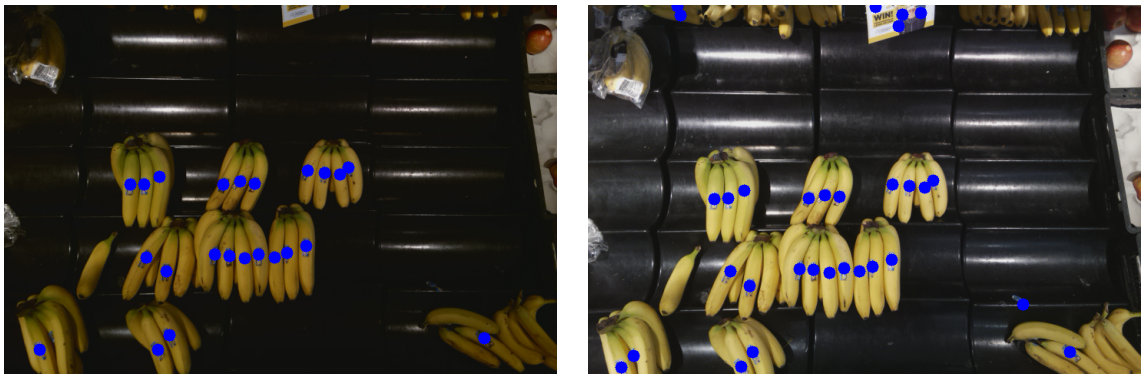
Figure 6.4: Blob filter results with ISO of 50 and 150.

6.3.2. STICKER DETECTION

In Figure 6.4, the blob filter results are shown with the different ISO values. When running the algorithm with big cluster sizes, most of the clusters got merged. Since the sticker filter used a Gaussian smoothing filter , most of the noise is already removed. Figures 6.4 and 6.2 are very similar, the only function of the blob filter is creating larger blobs exterminating two clusters that represent one sticker. As expected, the simple blob finder performed substantially better than the BFS algorithm. The BFS algorithm performed worse when giving more clusters in an image where the performance of the blob finder stayed the same. The advantage of the simple blob finder over the BFS algorithm was the numerous option to filter the blobs with ease. With a simple parameter, the blobs could be filtered with one line of code where the BFS had to be implemented into the code.

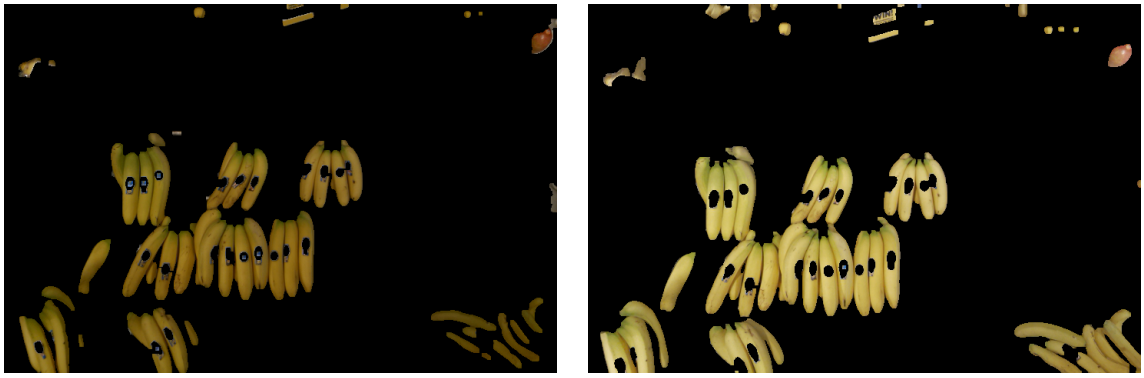
6.3.3. FOREGROUND SEGMENTATION

In Figure 6.6 the result of the foreground segmentation is displayed for two different values of the ISO. The segmentation requires the bananas to be properly illuminated for the best segmentation. With an ISO of 100 the segmentation produces the best results. It captures most of the foreground while still filtering away enough of the background. In Figure 6.7 another representative foreground segment is seen. Created from the original in Figure A.1. As seen all of the bananas are present in the foreground, the only problem with this result is that some of the area that does contain bananas is not present in the foreground mask. This creates the problem that if the conclusion is made that the shelf is 60% filled that it might in reality be only 50%. This has to be taken into account when drawing conclusion from the data. There is another method for foreground segmentation that is only viable when the color of the background is completely black. The segmented image seen in Figure A.1 uses the same method used for filtering the stickers. It looks at the HSV



(a) ISO 50 (b) ISO 150

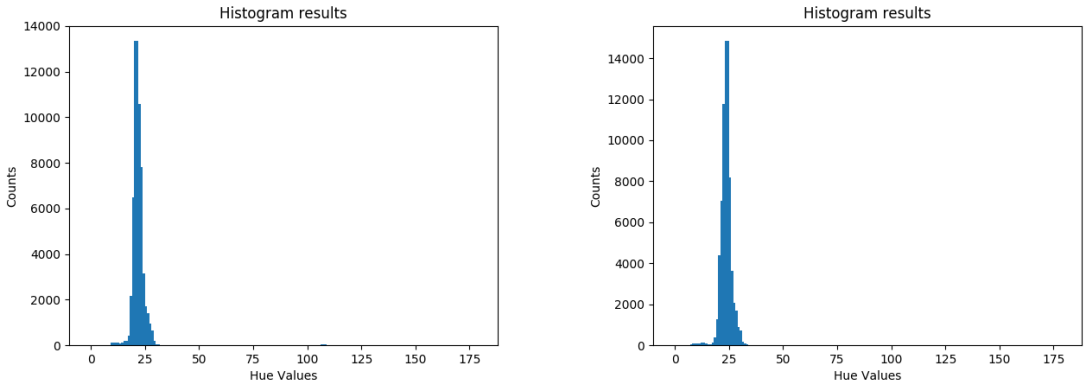
Figure 6.5: Sticker detection results with ISO of 50 and 150.



(a) ISO 50 (b) ISO 150

Figure 6.6: Foreground extraction results with ISO of 50 and 150.

Figure 6.7: Foreground segmented image from Figure A.1 using all three domains of HSV with the threshold values [5, 30, 30] and [50, 225, 225]. Everything outside of this region is removed.



(a) first (b) second

Figure 6.8: Histogram results with ISO of 50 and 150.

domain and filters everything out of it's region. The chosen region for the foreground is [5, 30, 30] and [50, 225, 225] for H, S and V respectively. The region from 5-50 contains green, yellow and brown. Because of the small of offsets in S and V the black and white background parts are removed. The only noise left in the image is on the top part, there the background is not completely black.

6.3.4. FOREGROUND HISTOGRAM

Figure 6.8 shows the results of the histogramming of the two foreground segmentation. As expected, the differences on the histograms with different ISOs are minimal since the luminance is not altering the Hue. On the image, the bananas are mostly yellow. The small hill at the 10-12 area is caused by the apple being extracted in the foreground. The other tests with a black background resulted in a similar result. However, the images with a green background had more counts in the green Hue area since when extracting the foreground with an green background, parts of these background gets extracted.

6.4. ADDITIONAL COMPLETE MEASUREMENTS

In Appendix B and C some complete measurements have been preformed and their output is shown. The first measurement is where part of the background is green. Because green is one of the colors we want to detect, the foreground segmentation has trouble filtering this bright color. The brightness and saturation of the green background is not filtered away as seen in the image multiplied with the foreground mask. This causes the percentage of the foreground to be inaccurate and the color histogram has additional values in the green region. The sticker filter and detection is not inhibited by the presence of the green background. All the

stickers are detected some additional “stickers“ are detected that are not actual stickers. Because some of of these cannot be filtered away but will always be detected the amount of clusters needs to be calibrated when placed to ignore these. The second measurement is a image of the shelf where the background is completely green. This has some of the same problems as the previous measurement where the foreground segmentation does not accurately create a mask over the bananas. The solution provided expects a black background for the banana shelf and this would be a requirement for the system to function the best.

6.5. DISCUSSION

The accuracy of the implemented methods depends strongly on the picture that is used. When the background of the banana shelf is not black, then the foreground segmentation does not work like intended. The consequence is that the total foreground calculated is not representative for the total shelf covered by bananas. The difference between the correct foreground and the calculated foreground is not is always within 10% of the actual value. This makes it possible to still make an estimated guess about the actual foreground. The goal is not 100% accuracy but rather to signal the manager of the supermarket about the status of the shelf. We think that this can still be achieved even if some of the background is removed.

There is one big problem with the chosen solution. The foreground segmentation used determines by looking at peak values in the histogram. When the banana shelf is completely empty this adaptive thresholding will generate a very wrong picture because the chosen threshold does not represent the bananas. Because of this we also included the counting of stickers on the bananas. When the system does detect foreground but no stickers are detected, surely something is wrong because there should always be stickers in sight when there are bananas left. There is a correlation between the total foreground and the number of detected stickers.

The next feature that we want to detect is the quality of the bananas. The histogram created from the segmented image should only contain the color information of the bananas. The color is then divided into three parts. The problem with this is that when there is only one banana that is rotten on the shelf, the histogram with the amount of brown is not detectable. But the manager of the supermarket will probably want to remove this one bad banana from the shelf. The way to deal with this problem does not lie with the sensor but rather the data interpretation. Rotten bananas will never be put on the shelf by the shelf filler, so instead of analyzing the amount of brown, the focus should be on the change from yellow to brown. For this long term analysis is needed and we instead of only giving the percentage should also pass along the total amount of brown.

7

CONCLUSION AND RECOMMENDATIONS

In conclusion, our design extracts all the features we wanted to detect and produces results that could be helpful for a supermarket manager. The prototype made is not yet an IoT device in the sense that it uses a battery and could be placed anywhere, but the potential is there. The system can determine how filled the banana shelf is, what the quality of the bananas on the shelf is, and if it is properly filled. There are a few conditions that are needed for the system to work, the background of the shelf has to be black to get the best results. When the background of the shelf is not black, the system does work but the accuracy the determining how filled the shelf is decreases. When these conditions are met, the accuracy of the of the sticker detector when calibrated is greater than 95%. The foreground segmentation when the background is completely black also meets our requirement of more than 90%.

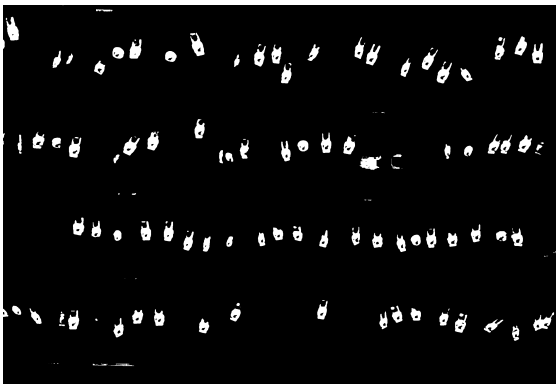
The applications of the design are needed in the supermarket of the future where algorithms can tell employees when and where what needs to be restocked. The second reason is that because the data is being stored in a database it can be analyzed to react preemptive instead of reactive. The next step in this research is the application of machine learning, can we use a trained system to detect defects in the fruit. That would make it possible to spot rotten fruit with precision so the manager can take action. We think the system can be expanded to different sorts of fruit and vegetables. The quality of the fruit or vegetables can not always be determined by only looking at color but the amount is always detectable.

Appendices

A

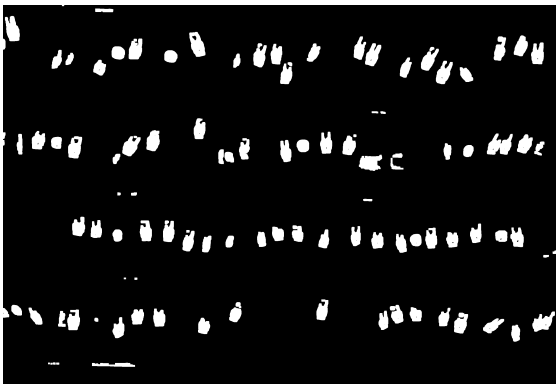
PRIMARY TEST SAMPLE

(a) Original image

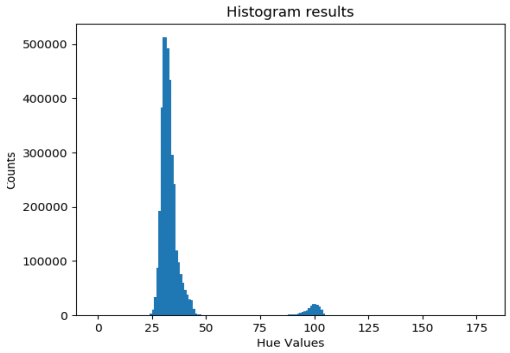


(b) Sticker filter

(c) Foreground segmentation



(d) Blob filter



(e) Histogram

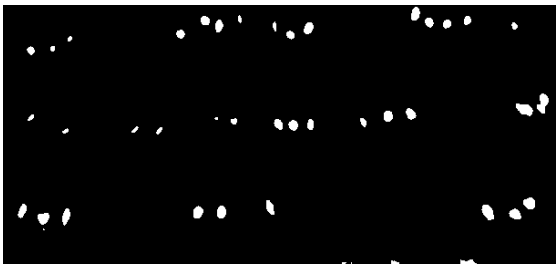
(f) Cluster detection

Figure A.1: Results of a test with a high resolution.

B

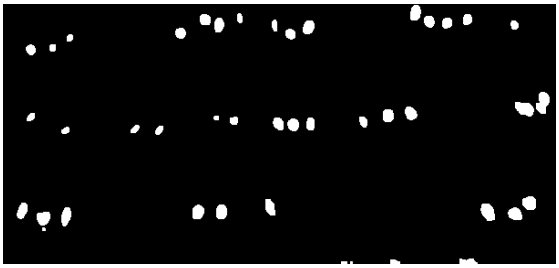
ADDITIONAL TEST 1

(a) Original image

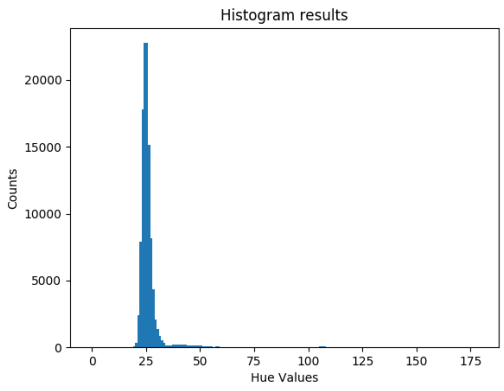


(b) Sticker filter

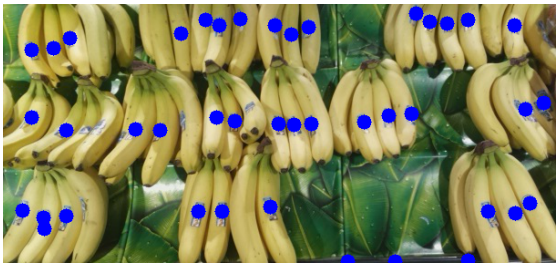
(c) Foreground segmentation



(d) Blob filter



(e) Histogram

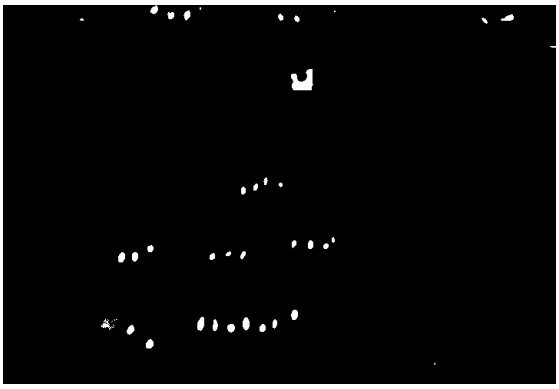


(f) Cluster detection

Figure B.1: Results of test 1.

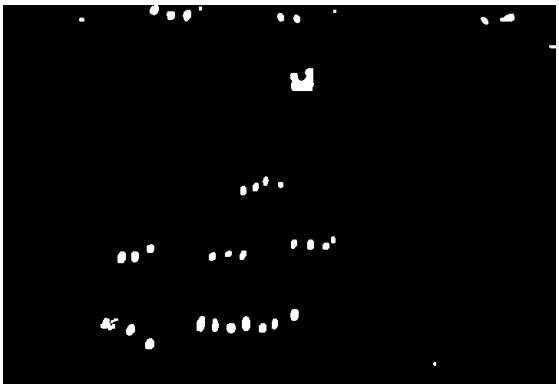
C
ADDITIONAL TEST 2

(a) Original image

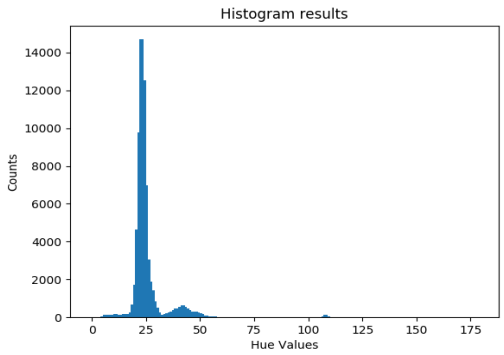


(b) Sticker filter

(c) Foreground segmentation



(d) Blob filter



(e) Histogram

(f) Cluster detection

Figure C.1: Results of test 2.

D
PYTHON CODE

D.1. STICKER FILTER

```
'''
Mandatory Sticker filter
IN:     hsv         NxMx3 array representing an image in HSV format
        gray        NxM array representing the gray format
OUT:    mask        NxM array of the mask
        hsv_fil     NxMx3 array of hsv with opening
        opening     NxM array of the opening
'''
def sticker_filter1(hsv, gray):
    # Threshold input
    ret2, thresh2 = cv2.threshold(hsv[:,:,1], 65, 255, cv2.THRESH_BINARY_INV)
    ret5, thresh5 = cv2.threshold(gray,65 ,255, cv2.THRESH_BINARY)

    # substract thresh
    sub = cv2.subtract(thresh5, thresh2)

    # close substraction
    opening = cv2.morphologyEx(sub, cv2.MORPH_CLOSE, kernel, iterations = 5)

    # create toher output paramters
    lower = np.array([90, 30, 30], dtype=np.uint8)
    upper = np.array([150,225,225], dtype=np.uint8)
    hsv_fil = cv2.bitwise_and(hsv,hsv, mask=opening)
    mask = cv2.inRange(hsv_fil, lower, upper)

    return mask, hsv_fil, opening
```

```
'''
Second filter with 2D filter method
IN:     S       NxM array representing the Saturation of the image
        opening NxM array of the opening calculated in cluster_filter1
OUT:    final   NxM array of the foreground mask
'''
def clust_filter2d(S, opening):
    # filter with Saturation
    kernel = np.array([[0,-1,0],[-1,9,-1],[0,-1,0]], dtype='f')
    kernel = kernel/np.sum(kernel)
    fil_S = cv2.filter2D(S,cv2.CV_8U, kernel)

    # simple threshold
    fil_S = fil_S<50

    # create inverse mask
    mask_inv = cv2.bitwise_not(opening)
    kernel = np.ones((7,7))
    mask_inv = cv2.dilate(mask_inv, kernel, iterations=10)

    # substract inverse mask from fil_s
    final = cv2.subtract(fil_S.astype(np.uint8)*255, mask_inv)

    return final
```

```
'''
Create filter for clust_dig_filter
'''
def create_filter(r, typ, sizex, sizey):
    a,b = sizex/2, sizey/2
    if (typ == 0):
        z = np.ones((sizex,sizey))
        x,y = np.ogrid[-a:sizex-a, -b:sizey-b]
        mask = x*x + y*y <= r*r
        z[mask] = 0
    elif (typ == 1):
        z = np.zeros((sizex,sizey))
        x,y = np.ogrid[-a:sizex-a, -b:sizey-b]
        mask = x*x + y*y <= r*r
        z[mask] = 1
    elif (typ == 2):
        z = np.zeros((sizex,sizey))
        x,y = np.ogrid[-a:sizex-a, -b:sizey-b]
        mask = x*x + y*y <= r*r
        z[mask] = -1

    return z

'''
Second filter with digital filter method
IN:     S                   NxM array of the Saturation
        mask                NxM array of the opening in previous function
OUT:    dig_filter_test2    NxM array of new mask of foreground
'''
def clust_dig_filter(S, mask):
    # calculate dft of S
    img_fft_S = cv2.dft(np.float32(S),
                        flags = cv2.DFT_COMPLEX_OUTPUT | cv2.DFT_SCALE)

    # create filter
    sizex, sizey = S.shape[:2]
    x = create_filter(3, 0, sizex, sizey)
    z = create_filter(2, 1, sizex, sizey)
    h = np.zeros((sizex,sizey,2))
    h[:,:,0] = x + z
    h[:,:,1] = x + z

    # operations with filter
    img_fil_fft_S = img_fft_S * np.fft.ifftshift(h)
    img_inv_S = cv2.idft(img_fil_fft_S, flags=cv2.DFT_COMPLEX_OUTPUT)
    output_array_S = cv2.magnitude(img_inv_S[:,:,0], img_inv_S[:,:,1]) < 20

    # create mask
    dig_filter_test2 = cv2.bitwise_and(output_array_S.astype(np.uint8)*255,
                                       output_array_S.astype(np.uint8)*255,
                                       mask=mask)

    return dig_filter_test2
```

D.2. CLUSTER DETECTION

```
'''
Call bw_func to start
'''

"""
Function for BWS algorithm, change all cluster points to it value
"""
def BFS(x, it, i_i, j_i):
    Queue = [(i_i,j_i)] # create QueueX and QueueY

    # while size of Queue is not zero
    while (len(Queue) != 0):
        point = Queue[0]    # point evaluated is in Queue[0]
        x[point[0], point[1]] = it  # value should be it

        # check surroundings
        for di in range(-1,2):
            for dj in range(-1,2):
                # if surrounding is 1
                if (x[point[0] + di, point[1] + dj] == 1):
                    tPoint = (point[0] + di, point[1] + dj)
                    # if not in queue already --> add
                    if(tPoint not in Queue):
                        Queue.append(tPoint)
        Queue.pop(0)    # delete point in queue after evaluation
    return x

"""
Function to create a BW array of the binary array
IN      x               NxM binarized array
OUT     bw              NxM array with min = 2 and max = number of clusters + 1
"""
def bwfunc(x):
    # padding for simple iterations
    x = np.lib.pad(x, 1, padzero)
    it = 2 # start iteration
    s = np.shape(x)

    # iteration through every point
    for i in range(1,s[0]-1):
        for j in range(1,s[1]-1):
            # if is one -> new cluster found
            if(x[i,j] == 1):
                x = BFS(x, it, i, j)
                it += 1
    return x
```

```
'''
Pre Filter to created blobs instead of clusters
IN:     img         The NxM binary image
        size        Filter size
        thres       Threshold to filter small clusters out
OUT:    fil         The binary output
'''
def pre_filter(img, size, thres):
    kernel = np.ones((size,size), np.float32)/(size*size)
    fil = cv2.filter2D(img.astype(np.float32), -1, kernel)
    fil = fil > thres
    return fil.astype(np.uint8)

'''
Blob detector which finds the blobs
IN:     bit_map     The NxM binary image
        mincluster  Minimum Area of the blobs
OUT:    x           Array of X coordinates
        y           Array of Y coordinates
'''
def blobdetector(bit_map, mincluster):
    # Parameters of the blobdetector
    params = cv2.SimpleBlobDetector_Params()
    params.blobColor = 255
    params.filterByColor = True
    params.filterByCircularity = False
    params.filterByConvexity = False
    params.filterByInertia = False
    params.filterByArea = True
    params.minArea = mincluster
    params.maxArea = 100000

    # create detector and find keypoints
    detector = cv2.SimpleBlobDetector_create(params)
    keypoints = detector.detect(bit_map*255)
    x = np.array([])
    y = np.array([])
    for i in keypoints:
        x = np.append(x, i.pt[0])
        y = np.append(y, i.pt[1])

    return x, y
```

D.3. FOREGROUND SEGMENTATION

```
'''
Function to extract foreground
IN:     img             NxMx3 array representing the imagen in RGB [0 - 255]
OUT:    h_foreground    NxM array of the foreground Hue [0 - 179]
        opening         NxM array of the foreground mask [0 | 255]
'''
def foreground_segmentation(img):
    # Smoothen the image with gaussian filter
    kernel = cv2.getGaussianKernel(11, 0)
    Gaussian = kernel * kernel.T
    img_smooth = cv2.filter2D(img, cv2.CV_8U, Gaussian)

    # convert to gray and HSV format
    img_gray = cv2.cvtColor(img_smooth, cv2.COLOR_RGB2GRAY);
    img_hsv = cv2.cvtColor(img_smooth, cv2.COLOR_RGB2HSV);

    # threshold the smooth gray image and the smooth saturation
    ret2, thresh2 = cv2.threshold(img_hsv[:,:,1], 65, 255, cv2.THRESH_BINARY_INV)
    ret3, thresh3 = cv2.threshold(img_gray,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

    # substract the thres values found
    back_rm = cv2.subtract(thresh3, thresh2)

    # create opening and mask
    kernel = np.ones((5,5), np.uint8)
    opening = cv2.morphologyEx(back_rm, cv2.MORPH_OPEN, kernel, iterations=1)
    mask = opening/255

    # return the Hue array with mask and opening
    return img_hsv[:,:,0]*mask, opening
```

D.4. HISTOGRAMMING

```
"""
Function to call with a HSV Foreground image
IN:          h_foreground        H of HSV image of the foreground
OUT:         per_green           Percentage of green pixels
             per_yellow          Percentage of yellow pixels
             per_brown           Percentage of brown pixels
CONSTANTS:   brown_range         Range of brown Hue  [01 - 15]
             yellow_range        Range of yellow Hue [16 - 30]
             brown_range         Range of green Hue  [31 - 60]
"""
def color_histogram(h_foreground):
    # Constants for the color ranges
    brown_range = [1, 15]
    yellow_range = [16, 25]
    green_range = [26, 60]

    # Histogramming
    data = h_foreground.flatten()
    counts, bins = np.histogram(data, range(179+1))

    # Counting for every category
    counts_brown = np.sum(counts[brown_range[0]:brown_range[1]])
    counts_yellow = np.sum(counts[yellow_range[0]:yellow_range[1]])
    counts_green = np.sum(counts[green_range[0]:green_range[1]])
    counts_colors = counts_brown + counts_yellow + counts_green
    counts_black = counts[0]
    counts_total = np.sum(counts)

    # Percentages
    p_b = float(counts_brown) / float(counts_colors) * 100.0
    p_y = float(counts_yellow) / float(counts_colors) * 100.0
    p_g = float(counts_green) / float(counts_colors) * 100.0
    p_f = float(counts_total) / float(counts_total) * 100.0

    return p_b, p_y, p_g, p_f, counts, bins
```

BIBLIOGRAPHY

[1] Gartner Inc., *Gartner says 8.4 billion connected "things" will be in use in 2017, up 31 percent from 2016,* (2017).

[2] A. Gongal, S. Amatya, M. Karkee, Q. Zhang, and K. Lewis, *Sensors and systems for fruit detection and localization: A review,* Computers and Electronics in Agriculture **116**, 8 (2015).

[3] I. Sa, Z. Ge, F. Dayoub, B. Upcroft, T. Perez, and C. McCool, *Deepfruits: A fruit detection system using deep neural networks,* Sensors **16** (2016), 10.3390/s16081222.

[4] Y. Edan, D. Rogozin, T. Flash, and G. E. Miles, *Robotic melon harvesting,* IEEE Transactions on Robotics and Automation **16**, 831 (2000).

[5] R. L. Bosoon Park, *Hyperspectral Imaging Technology in Food and Agriculture*, 1st ed., 10, Vol. 1 (Springer-Verlag New York, The address, 2015) an optional note.

[6] J. P. Wachs, H. I. Stern, T. Burks, and V. Alchanatis, *Low and high-level visual feature-based apple detection from multi-modal images,* Precision Agriculture **11**, 717 (2010).

[7] F. Kurtulmus, W. S. Lee, and A. Vardar, *Immature peach detection in colour images acquired in natural illumination conditions using statistical classifiers and neural network,* Precision Agriculture **15**, 57 (2014).

[8] H. N. Patel, *Fruit detection using improved multiple features based algorithm,* (2011).

[9] S. Janssen, K. Schmitt, M. Blanke, M. L. Bauersfeld, J. Wollenstein, and W. Lang, *Ethylene detection in fruit supply chains,* Philosophical Transactions of the Royal Society of London Series A **372**, 20130311 (2014).

[10] A. Sanaeifar, S. S. Mohtasebi, M. Ghasemi-Varnamkhasti, and M. Siadat, *Application of an electronic nose system coupled with artificial neural network for classification of banana samples during shelf-life process,* in *2014 International Conference on Control, Decision and Information Technologies (CoDIT)* (2014) pp. 753–757.

[11] H. Saad, A. P. Ismail, N. Othman, M. H. Jusoh, N. fadzlina Naim, and N. A. Ahmad, *Recognizing the ripeness of bananas using artificial neural network based on histogram approach,* in *2009 IEEE International Conference on Signal and Image Processing Applications* (2009) pp. 536–541.

[12] IEC 61966-2-1:1999, *Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB*, IEC (1999).

[13] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision* (Thomson-Engineering, 2007).

[14] P. Viola and M. Jones, *Rapid object detection using a boosted cascade of simple features,* in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 1 (2001) pp. I–511–I–518 vol.1.

[15] J. N. Sarvaiya, S. Patnaik, and S. Bombaywala, *Image registration by template matching using normalized cross-correlation,* in *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies* (2009) pp. 819–822.

[16] A. Hii, C. E. Hann, J. Chase, and E. V. Houten, *Fast normalized cross correlation for motion tracking using basis functions,* Computer Methods and Programs in Biomedicine (2006).

[17] E. Rachmawati, M. L. Khodra, and I. Supriana, *Edge based approach in object boundary detection on multiclass fruit images,* in *2016 4th International Conference on Information and Communication Technology (ICoICT)* (2016) pp. 1–6.

[18] T. A. Ell and S. J. Sangwine, *Hypercomplex fourier transforms of color images,* IEEE Transactions on Image Processing **16**, 22 (2007).

[19] Z. Lu, Y. Xu, X. Yang, L. Song, and L. Traversoni, *2d quaternion fourier transform: The spectrum properties and its application in color image registration,* in *2007 IEEE International Conference on Multimedia and Expo* (2007) pp. 1715–1718.

[20] S.-C. Pei, J.-J. Ding, and J.-H. Chang, *Efficient implementation of quaternion fourier transform, convolution, and correlation by 2-d complex fft,* IEEE Transactions on Signal Processing **49**, 2783 (2001).

[21] W. Zhu, Q.-L. Fu, and J.-Q. Bai, *The real-time object detection algorithm based on orbp and cascade svm,* in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (2016) pp. 1023–1027.

[22] H. C. Karaimer, I. Cinaroglu, and Y. Bastanlar, *Combining shape-based and gradient-based classifiers for vehicle classification,* in *2015 IEEE 18th International Conference on Intelligent Transportation Systems* (2015) pp. 800–805.

[23] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, *An efficient k-means clustering algorithm: analysis and implementation,* IEEE Transactions on Pattern Analysis and Machine Intelligence **24**, 881 (2002).

[24] M. A. Kaur, M. P. Sharma, and M. Verma, *A appraisal paper on breadth-first search, depth-first search and red black tree,* International Journal of Scientific and Research Publications (IJSRP) **4** (2014).

[25] M. Piccardi, *Background subtraction techniques: a review,* in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, Vol. 4 (2004) pp. 3099–3104 vol.4.

[26] V. Sivakumar and V. Murugesh, *A brief study of image segmentation using thresholding technique on a noisy image,* in *International Conference on Information Communication and Embedded Systems (ICICES2014)* (2014) pp. 1–6.

[27] J. Fan, D. K. Y. Yau, A. K. Elmagarmid, and W. G. Aref, *Automatic image segmentation by integrating color-edge extraction and seeded region growing,* IEEE Transactions on Image Processing **10**, 1454 (2001).

[28] N. Otsu, *A threshold selection method from gray-level histograms,* IEEE Transactions on Systems, Man, and Cybernetics **9**, 62 (1979).

[29] S. R. Rupanagudi, B. S. Ranjani, P. Nagaraj, and V. G. Bhat, *A cost effective tomato maturity grading system using image processing for farmers,* in *2014 International Conference on Contemporary Computing and Informatics (IC3I)* (2014) pp. 7–12.

[30] Y. Intaravanne, S. Sumriddetchkajorn, and J. Nukeaw, *Ripeness level indication of bananas with visible and fluorescent spectral images,* in *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology* (2012) pp. 1–4.

[31] M. Verma and B. Raman, *Object tracking using joint histogram of color and local rhombus pattern,* in *2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)* (2015) pp. 77–82.

[32] B. Guoan and Z. Yonghong, *Transforms and Fast Algorithms for Signal Analysis and Representations* (Birkhäuser Basel, 2004).

[33] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, *Design of an image edge detection filter using the sobel operator,* IEEE Journal of Solid-State Circuits **23**, 358 (1988).

[34] PiCamera, *Picamera docs,* (2017).

[35] SciPy, *Numpy user guide,* (2017).

[36] OpenCV, *Open source computer vision 3.1,* (2015).