Industrial Project by Vilmorin

Digital Image Fundamentals

Partners:	Evelyn Paiz
	Nadile Nunes
Instructors:	Hubert Konik
	Carlos Arango

1 Introduction

The processing and understanding of images has currently become one of the most significant fields of computer sciences [1]. Because of that, a great number of people working with images has rapidly increased. This means that the demand for image processing tools also grows [2]. Modern digital technology has made it possible to manipulate multi-dimensional signals with systems that range from simple digital circuits to advanced parallel computers. The goal of this manipulation can be divided into three categories [5]:

• Image Processing	image in → image out
• Image Analysis	image in → measurements out
• Image Understanding	image in → high-level description out

Image processing technology has provided innovations and benefits in the fields of medicine, security, manufacturing, science and agriculture [1]. One company that works and uses this technologies, is Vilmorin. It is the 4th worldwide seed company and creator of varieties of seeds. It's need is oriented in those methods to efficiently segment, recognize and classify vegetables and fruits. Colour, shape and size are available parameters that could be used. More complex characteristics could be also added, as texture been one of those characteristics. The objective of this traineeships was to study and test some existent applications on a given fruit (melon) and perform the most efficient one in order to get a good tool to characterize melons for experts at Vilmorin company [4].

The focus of this project was mainly on the fundamental concepts of image processing. Also a few introductory remarks about image analysis was made. It is not easy to develop, actualize or give the training on image processing applications by employing classical programming languages and techniques. Thus, function libraries addressing to image processing algorithms have been annexed to these programming languages and turned into available tools. MATLAB has been one of the most preferred application domains in the development of image processing applications [1]. Therefore, MATLAB was the tool used in the present project. It's superior aspect, when compared to other applications, of a wide capacity of mathematical operations made it a good programing language to model images as matrixes and worked with them.

2 Objectives

- Test the knowledge acquired in the course.
- Test the skills and creativity to solve an image processing problem in the industry.

3 Problem definition

The problem at hand was to analyze and classify a group of images with melons in it. The data was a set of pictures of melons such as the ones at figure 1.

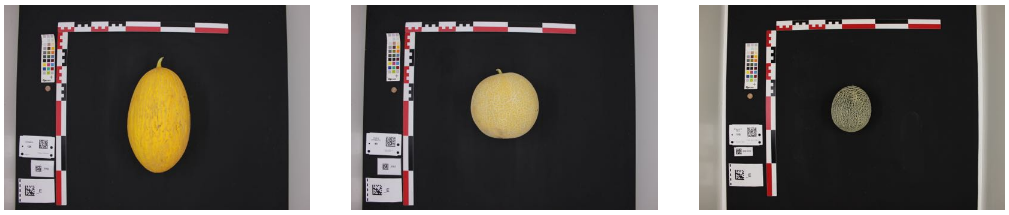


Figure 1: Sample of data set worked

4 Methodology

The following process was done:

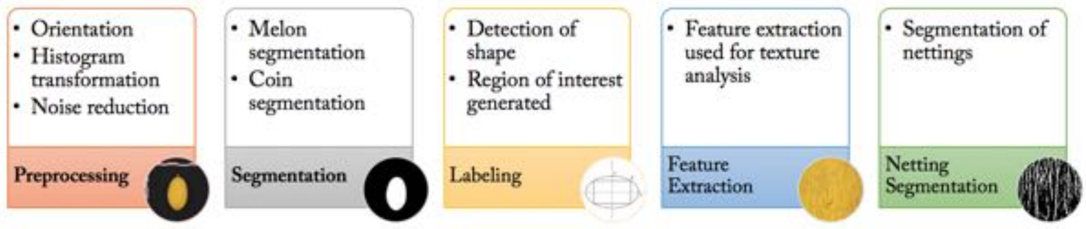


Figure 2: Graphical description of the methodology used

4.1 Preprocessing

For the preprocessing section, first it was identified the correct orientation for each image. Followed by this, it was developed a preprocessing step that could improve the segmentation and feature extraction steps, that followed. The following methods were programed and tested (the ones with the best result for this application were selected):

1. Histogram transformation:
   - Dynamic expansion $\Leftarrow$ *selected*
   - Histogram equalization
   - Adaptive histogram equalization
2. Noise reduction:
   - Averaging filtering
   - Gaussian filtering $\Leftarrow$ *selected*
   - Median filtering
   - Sharpening of the image

4.2 Segmentation

Next, the process of segmentation was performed. This was done doing a change of color space, from *RGB* to *HSV*, and defining the correspondent color thresholds. Two segmentations were achieved, the segmentation of the melon area from the background and the segmentation of the 1 cent coin. To the segmentation of the melon, the stem was taken away.

4.3 Labeling

Two processes were done at the labeling level. First, from the melon segmentation, it was detected whether the melon was spherical or circular (analyzing it's eccentricity). It was returned the diameter or the distance of the major and minor axis accordingly in centimeters (using the coin diameter as reference from the segmentation of the coin).

Next, it was found the greatest rectangle that you can fit inside the segmented melon (definition 4.1) and selected it as the region of interest (area for cropping).

Definition 4.1 — Greatest rectangle fit in an ellipse. Suppose that the upper righthand corner of the rectangle is at the point (x,y). Then you know that the area of the rectangle is $4xy$ and you know that:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1 \tag{1}$$

Thinking of the area as a function of x, we have

$$\frac{dA}{dx}=4x\frac{dy}{dx}+4y \tag{2}$$

Differentiating (1) with respect to x, we have

$$\frac{2x}{a^2}+\frac{2y}{b^2}\frac{dy}{dx}=0 \tag{3}$$

so

$$\frac{dy}{dx}=-\frac{b^2x}{a^2y} \tag{4}$$

and

$$\frac{dA}{dx} = 4y - \frac{4b^2x^2}{a^2y} \quad (5)$$

Setting this to 0 and simplifying, we have $y^2 = \frac{b^2x^2}{a^2}$. From (1) we know that

$$y^2 = b^2 - \frac{b^2x^2}{a^2} \quad (6)$$

Thus, $y^2 = b^2 - y^2$, $2y^2 = b^2$, and $\frac{y^2}{b^2} = \frac{1}{2}$. Clearly, then, $\frac{x^2}{a^2} = \frac{1}{2}$ as well, and the area is maximized when

$$x = \frac{a}{\sqrt{2}} = \frac{a\sqrt{2}}{2} \quad (7)$$

$$y = \frac{b}{\sqrt{2}} = \frac{b\sqrt{2}}{2} \quad (8)$$

4.4 Feature Extraction

Texture analysis refers to the characterization of regions in an image by their texture content. It attempts to quantify intuitive qualities described by terms such as rough, smooth, silky, or bumpy as a function of the spatial variation in pixel intensities. Then, it was selected 3 different feature extraction methods used for texture analysis and they were used in the cropped area of the melons:

1. **Range**: in areas with smooth texture, the range of values in the neighborhood around a pixel will be a small value; in areas of rough texture, the range will be larger. This was used because the melons presented rough areas in the netting parts and smooth areas in the rest.
2. **Standard deviation**: calculating the standard deviation of the pixels in a neighborhood indicated the degree of variability of the pixel values in the region.
3. **Entropy**: a statistical measure of randomness that could be used to characterize the texture of the input image.

This methods were used because it was possible to obtain from them a co-occurrence matrix that was used then for the next section. Also, these statistics could characterize the texture of the images because they provide information about the local variability of the intensity values of pixels in each image.

4.5 Netting Segmentation

As an extra, it as proposed and implemented a method for segmenting the netting on the melons. This was achieved using texture segmentation to identify regions based on their texture. The goal was to segment two kinds textures in the image using texture filters. To achieve this, the three features extracted from the previous section were tested and the best was selected (entropy).

5 Results

At the end of the process it was possible to obtain a final program that ran and completed all the steps from figure 2. The following results were obtained from each step:

5.1 Preprocessing

The preprocessing was divided into three process:

Correct orientation

At first moment it was decided to use the orientation property of the largest rule, located at the left part, to define the orientation of the image. The problem with this approach was the fact that the correction by the orientation worked only in some cases. Then, as a second option it was used the coin position on the image to identify the current orientation and fix it. For this approach, it was used an image of the coin to find it in the melon image. Once it was found, it was verified the position to make the correction. This method was effective, but also slow and the new database contained images that didn't had the coin on them. This made impossible the use this approach.

As the third and definitive option, first it was checked if the image was horizontal or vertical (using the comparison of the sizes) and if it was vertical, the image was rotated 90 degrees. After that, it was checked if the image is upside down, dividing the image in two sides (left and right) and measuring the area in each. If the right side had more area, means that image was upside down, so it was rotated 180 degrees. Figure 3 shows an example of this method.

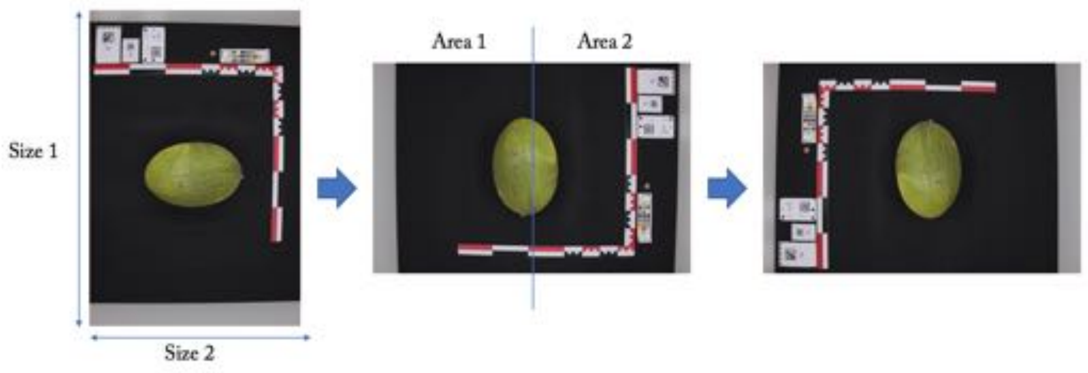
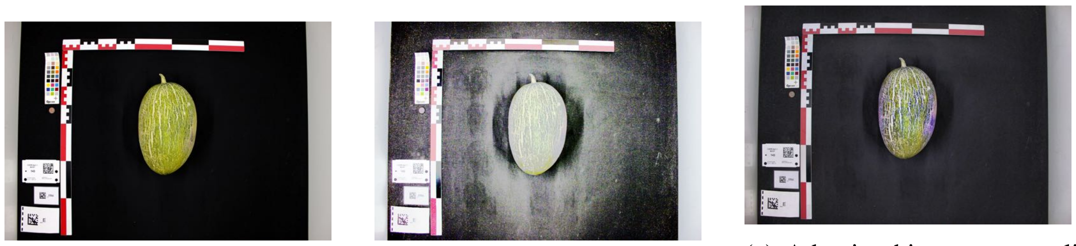


Figure 3: Example of orientation correction method on a melon image

Histogram transformation

For the histogram transformation (no matter what the method was), the transformation was applied on each channel of *RGB* and the result was merge on a final color image. The three histogram transformation methods were tested and the results showed that for this application, the dynamic expansion (imadjust) method was the best choice considering the steps that were to follow. Figure 4 shows an example of the three methods.

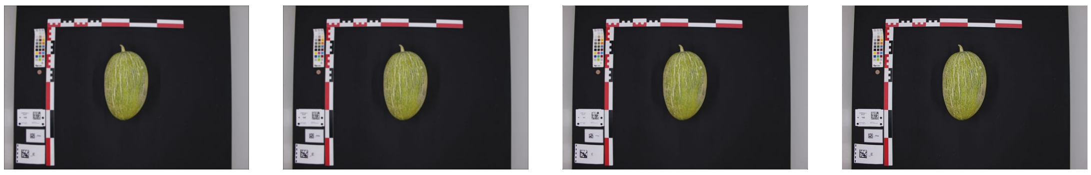


(a) Dynamic expansion (b) Histogram equalization (c) Adaptive histogram equalization

Figure 4: Example of histogram transformation methods on a melon image

Noise reduction

Next, almost all the noise reduction methods supported a color image, so the methods were applied directly on the image. The only one that was applied on each channel was the median filter. As the previous section, all the noise reduction methods were tested and it was the gaussian filtering the one that gave the best results. Figure 5 shows an example of the use of all the noise reduction methods.

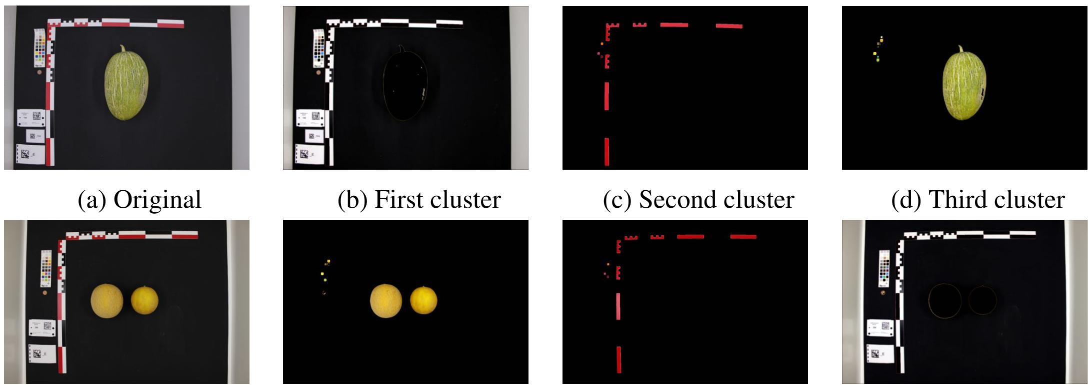


(a) Averaging filtering (b) Gaussian filtering (c) Median filtering (d) Sharpening

Figure 5: Example of noise reduction methods on a melon image

5.2 Segmentation

For the segmentation, the same process was done for the melon and the coin. The first approach was a $k-mean$ color segmentation. It was tested, but the results were not always correct and returned in order. The main difficulty (as figure 6 shows)was to know the order in which the melon would come in the cluster and to define a correct number of k clusters, because sometimes it was necessary 3 and sometimes 4. Also, it was noted that the distribution of the colors on the images were not behaving completely like a cluster distribution.

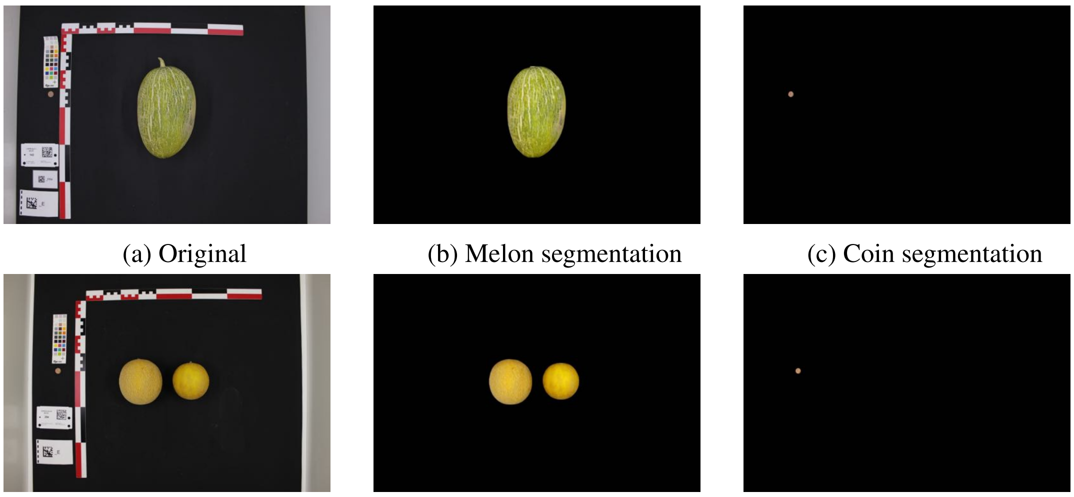


(a) Original (b) First cluster (c) Second cluster (d) Third cluster

(e) Original (f) First cluster (g) Second cluster (h) Third cluster

Figure 6: Example of discarded color segmentation method ($k-means$) on two melon images

The final approach was actually a color threshold on the HSV color space. The parameters that defined the melon segmentation was the H and the S values. For the coin segmentation, all the values were used. Various thresholds were tested, but the values chosen were obtained using the application *color threshold* from MATLAB. Here, the RGB, HSV and $L^*a^*b^*$ color spaces were evaluated, and the one with the best results was the HSV color space. Finally, on the melon segmentation, the stem was removed using a opening approach. Figure 7 shows examples of the final segmentation results.



(a) Original (b) Melon segmentation (c) Coin segmentation

(d) Original (e) Melon segmentation (f) Coin segmentation

Figure 7: Example of final method of segmentation (*HSV* color threshold) on two melon images

5.3 Labeling

The labeling was divided into two parts:

Shape definition

To verify the shape of the melon, the eccentricity was used. It was noted that the value for the eccentricity was between 0 and 1 (an ellipse whose eccentricity was 0 was actually a circle, while an ellipse whose eccentricity was 1 was a line segment). Some melons had a shape similar to a circle but were not a perfect one; therefore it was needed a different value to choose between circular and elliptical. It was tested some images to analyze the eccentricity and at the end it was decided to use the number 0.35 as a threshold for the shape.

To convert the size of the axis from pixels to centimeters, first it was used an image from a coin, and it's measurements were taken and applied on the conversion. The problem with this was the uncertainty of the scale. For the second solution it was used the segmentation made to capture only the coin from the original image. With this, it was possible to take the measurement of the coin in the same scale as the melon. If the image to be labeled didn't have a coin inside, it was used a segmented coin from a previous image in the conversion. All the results were written at the end on a text file, as the following example from the melons of the previous section (figure 7) shows:

```
Image melon1.JPG
Eccentricity: 0.763262
Ellipse
MajorAxisLength: 367.831839 pixels and 25.628527 cm
MinorAxisLength: 237.652234 pixels and 16.558318 cm

Image melon2.JPG
Melon 1
Eccentricity: 0.328478
Circle
Diameter: 174.367919 pixels and 12.334140 cm

Melon 2
```

```
Eccentricity: 0.180179
Circle
Diameter: 146.489436 pixels and 10.362119 cm
```

Region of interest

To crop the image, it was used a boolean value to identify if the melon was a circle or ellipse. If it was a circle, the area to crop will be a square, so only one value for the square side needs to be calculated. On the opposite way, if it was an ellipse, the area to crop will be a rectangle so it was need to calculate the height and width (using the definition 4.1).

For the case of the circle melon, the crop function worked well, which means that the region was all covered by the melon. But, if the shape was an ellipse, some problems presented when the melon had one part more narrow than the other. In this case some small dark area may appear on the corner of the region cropped.

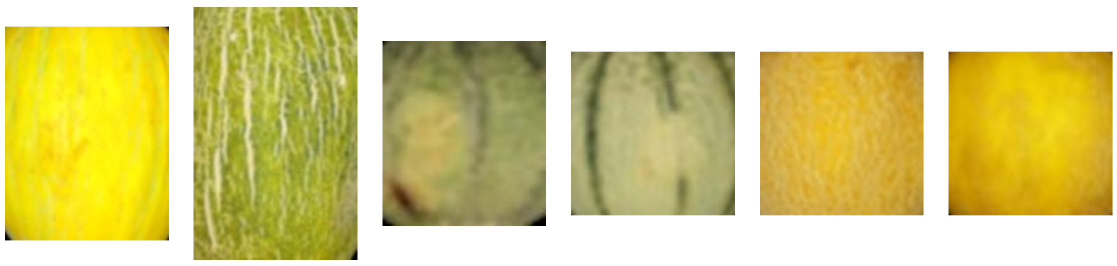
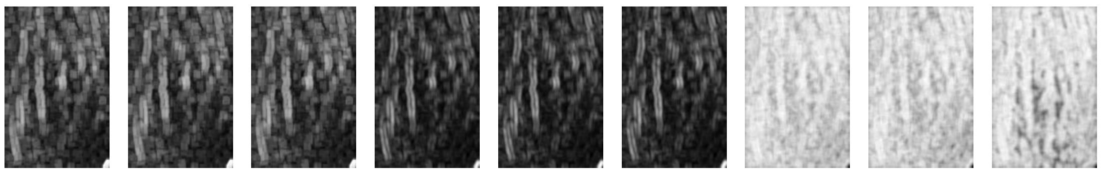


Figure 8: Example of crops from region of interest for various melon images

5.4 Feature Extraction

Using the region of interest obtained on the previous section, the texture analysis was done. As figure 9 shows, the corresponding statics selected (range, standard deviation and entropy) were computed on a co-occurrence matrix and the results were plotted for each channel of the image. Here the *RGB*, *HSV* and $L^*a^*b^*$ color spaces were tested and it was decided that the *RGB* had better results. This results were evaluated and it was decided that the entropy contained the best information to be used on the next section of netting segmentation.



(a) Range (b) Standard deviation (c) Entropy

Figure 9: Example of texture analysis statics on a melon image

5.5 Netting Segmentation

For this last part, it was used also the images obtained from the *ROI* section and the co-concurrence matrix obtained after the feature extraction (entropy selection). First, it was converted each channel of the co-concurrence matrix into a gray-level image and threshold them. Combining the three threshold images was created a mask and a outline of the boundary between the two textures. This mask was applied on the *ROI* image and as final result, it was possible to get an image with the boundary between the netting and the rest of the melon. The results were good enough for images that didn't had some color patches on the melons, but for the ones it had, the texture was considered as the color patches instead of the nettings as figure shows 10.

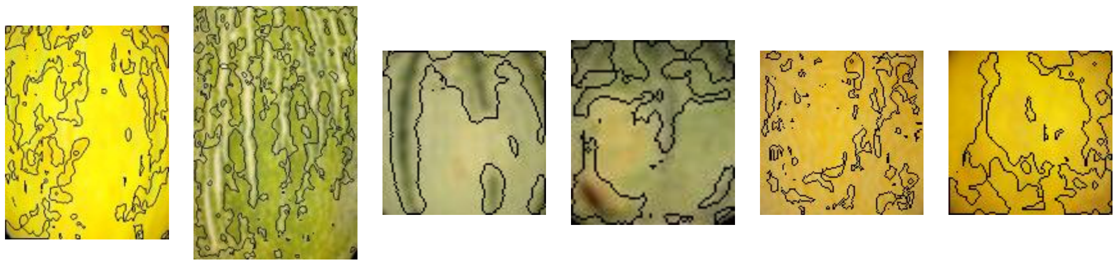


Figure 10: Example of netting segmentation for various melon images

6 Programming functions

R The project was developed in a centered document called **main.m**, which analyzed and classified the group of images with the melons in it, from Vilmorin company. The process was done following the functions:

6.1 Data Loading

Function 6.1 — get_all_files.m. Gets all files under a specific directory.
⇒ **input - *inputPath*:** The input path for the data to be loaded.
⇒ **input - *fileExtension*:** The type of file to search into the directory.
⇐ **output - *fileList*:** The list of the files that are contained in the directory.

Function 6.2 — load_data.m. Loads the data from an input directory into a vector.
⇒ **input - *inputPath*:** the input path for the data to be loaded.
⇐ **output - *dataList*:** a vector with the data read from the directory.

6.2 Preprocessing

Function 6.3 — correct_orientation.m. Corrects the orientation of an image to be horizontal.
⇒ **input - *I*:** the image to be corrected.
⇒ **input - *sizeI*:** the final size percentage for the output image (goes form the range of 0 to 1).
⇐ **output - *I*:** image corrected.

Function 6.4 — hist_transf.m. Histogram transformation in an image with a specific method type.
⇒ **input - *type*:** the type of histogram transformation to perform:
- *imadjust:* adjust image intensity values or colormap.
- *histeq:* enhance contrast using histogram equalization.
- *adapthisteq:* enhances the contrast of images by transforming the values in the intensity image I.

⇒ **input - *I*:** the image to correct.
⇐ **output - *I_transformed*:** image corrected.

Function 6.5 — reduce_noise.m. Reduces the noise in an image with a specific method type.
⇒ **input - *type*:** the type of noise reduction to perform:
- *averaging:* returns an averaging filter.
- *gaussian:* returns a rotationally symmetric Gaussian lowpass filter.
- *median:* returns a median filtering.
- *sharpening:* returns an enhanced version of the image.

⇒ **input - *I*:** the image to correct.
⇒ **input - *size*:** the size of the reduction of noise.
⇐ **output - *I_reduced*:** image corrected.

6.3 Segmentation

Function 6.6 — segment.m. Segmentation of image based on thresholding for the HSV color space.
⇒ **input - *I*:** the image to be segmented.
⇒ **input - *hueThresholdLow*:** the lower limit of the hue threshold.
⇒ **input - *hueThresholdHigh*:** the higher limit of the hue threshold.
⇒ **input - *saturationThresholdLow*:** the lower limit of the saturation threshold.
⇒ **input - *saturationThresholdHigh*:** the higher limit of the saturation threshold.
⇒ **input - *valueThresholdLow*:** the lower limit of the value threshold.
⇒ **input - *valueThresholdHigh*:** the higher limit of the value threshold.
⇐ **output - *mask*:** final segmented image (binary).
⇐ **output - *smallestAcceptableArea*:** the area of the biggest object that exist in the final mask.

6.4 Labeling

Function 6.7 — def_shape.m. Detect whether the melon is spherical or circular using the coin diameter as reference.
⇒ **input - *I*:** the image to be analyzed (as a binary).
⇒ **input - *Ref*:** the reference image to compare I (as a binary).
⇒ **input - *size*:** the size used by the reference.
⇒ **input - *pid*:** file id to save the results.
⇐ **output - *circle*:** if the melon(s) is/are a circle or not.
⇐ **output - *majorAxis*:** array with the value(s) of the major axis of the melon(s). If the melon have a circular shape, the major will be zero.
⇐ **output - *minorAxis*:** array with the value(s) of the minor axis of the melon(s).
⇐ **output - *centroid*:** array with the values of the centroid of the melon(s).

Function 6.8 — crop_image.m. Uses the data acquired during the labeling to crop the segmented melon(s) to future analysis.
⇒ **input - *I*:** the image to be analyzed (as a binary).
⇒ **input - *circle*:** array with boolean values indicating if the shape of the melon is circular or not.
⇒ **input - *majorAxis*:** array with the value(s) of the major axis of the melon(s). If the melon have a circular shape, the major will be zero.
⇒ **input - *minorAxis*:** array with the value(s) of the minor axis of the melon(s).
⇒ **input - *centroid*:** array with the values of the centroid of the melon(s).
⇐ **output - *cropedMelon*:** one or more images, depending of how many melons are inside the image.

6.5 Feature Extraction

Function 6.9 — texture_analysis.m. Corrects the orientation of an image to be horizontal.

⇒ **input - *type*:** the type of texture analysis to perform:
  - *entropyfilt:* calculates the local entropy of a grayscale image. Entropy is a statistical measure of randomness.
  - *stdfilt:* calculates the local standard deviation of an image.
  - *rangefilt:* calculates the local range of an image.

⇒ **input - *I*:** the image to analyze.

⇒ **input - *nhood*:** is a multidimensional array of zeros and ones where the nonzero elements specify the neighborhood for the range filtering operation.

⇐ **output - *J*:** gray-level co-occurrence matrix.

6.6 Netting Segmentation

Function 6.10 — texture_segmentation.m. Use texture segmentation to identify regions based on their texture.

⇒ **input - *I*:** image to be segmented.

⇒ **input - *J*:** gray-level co-occurrence matrix.

⇐ **output - *outlineTexture*:** final outlined texture image.

7 Bibliography

[1] Eker, Bulent, and Eker, Aysegul Akdogan, *MATLAB-Based Applications for Image Processing and Image Quality Assessment*, University of Kragujevac, 2013.

[2] Krasula, Lukas, Klima Milos. Rogard, Eric, and Jeanblanc Edouard, *MATLAB-based Applications for Image Processing and Image Quality Assessment, Part I: Software Description*, Radioengineering, vol. 20, No. 4., pp 1009-1015, 2011.

[3] Krasula, Lukas, Klima Milos. Rogard, Eric, and Jeanblanc Edouard, *MATLAB-based Applications for Image Processing and Image Quality Assessment, Part II: Experimental Results*, Radioengineering, vol. 21, No. 1., pp 154-161, 2012.

[4] Plantin, B., *Vegetable classification by coloured texture recognition*, Master Thesis, Gjøvik University College, Saint-Etienne Unversity, 2016.

[5] Young, Gerbrands, and Van Vliet, *Fundamentals of Image Processing*, Delft University of Technology, 2009.