Image Processing and Computer Vision

Fruit Inspection

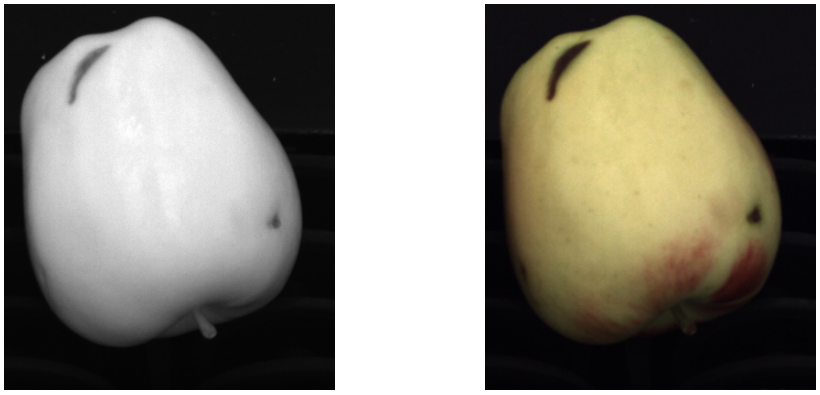
Facundo Nicolas Maidana (facundo.maidana@studio.unibo.it)
Riccardo Spolaor (riccardo.spolaor@studio.unibo.it)

Contents

1 Introduction

The use of modern *Image Processing* and *Computer Vision* techniques is widely spread in the fruit supply chain as a way of guaranteeing quality and homogeneity of the final product to the end-users, whilst reducing both production costs and times.

The present report, describes the implementation of a project proposed by *UNITEC S.p.a.* [1, 2], which aims at developing an intelligent system capable of detecting imperfections and defects in fruits. Each fruit is acquired through a *NIR (Near Infra-Red)* and colour camera with little parallax effect, as seen in Figure 1.



(a) Near Infra-Red image of the fruit. (b) Colour image of the fruit.

Figure 1: Comparison of the image of a fruit caught with a Near Infra-Red and a colour camera.

The project includes three distinct tasks. In each of them a group of different colour images of fruits, along with their respective NIR version, is presented. The images must be analysed and altered through Image Processing techniques in order to gain information about the presence of imperfections on the fruits. The following requirements are expected to be achieved:

- **First Task: Fruit Segmentation and Defect Detection.** Three images of apples are presented. The goal is to segment them from the background and locate imperfections caused by holes on their skin;
- **Second Task: Russet Detection.** Two images of apples with unwanted reddish-brown parts (referred as russets) are observed. The aim is to identify these areas in the most precise way as possible;
- **Final Challenge: Kiwi Inspection.** It is required to analyse five images of kiwis in order to locate their defects. Extra effort is required to pre-process them because of a great amount of noisy artefacts in the background of some of the given images.

The technology stack used to develop the solution is composed of *Python* [3] as programming language and *OpenCV* [4] as a Computer Vision framework. The source code of the implementation that is illustrated in the report is available in the following *Github* repository [5], where the solution is presented in a series of notebooks and the system can be simulated through command line scripts.

Although this is an academic project, the working pipeline has been developed while considering an industrial setting where the inspection of the fruits has to be carried out by an *online* system. A set of different methods have been tested in terms of quality, speed and robustness. The proposed architecture, although not thoroughly tested because of a lack of instances, is expected to function even with images of fruits different than the ones presented. Moreover, a discussion about the portability of the system in different industrial environments is conducted for each task.

2 First Task: Fruit Segmentation and Defect Detection

The first task aims to develop a system that detects imperfections on the surface of fruits. The sample images are observed in Figure 2.

(a) First NIR image. (b) Second NIR image. (c) Third NIR image.

(d) First colour image. (e) Second colour image. (f) Third colour image.

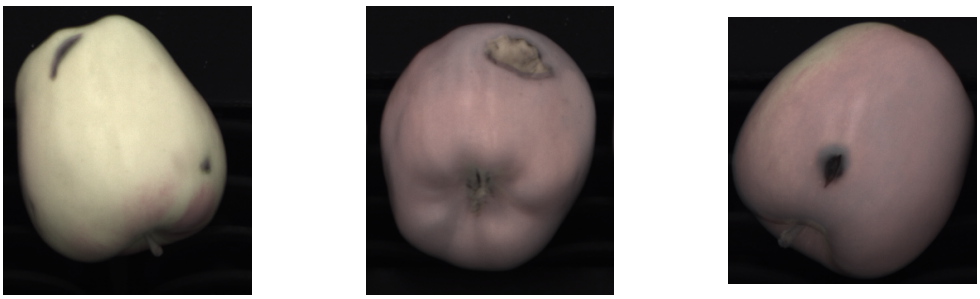
Figure 2: Source images of the first task.

In order to localize their defects two steps have to be carried out:

- Each fruit should be outlined by means of a binary mask that separates the foreground (the fruit) from the background;
- A search to identify the defects on the fruit must be carried out on the outlined portion of the image.

2.1 Image Analysis

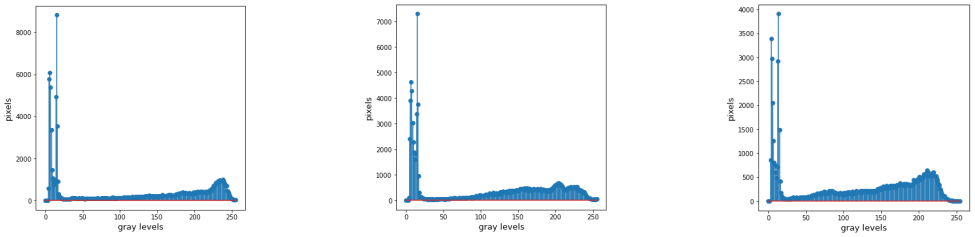
Firstly, the parallax between the images is observed in Figure 3. The NIR and colour images overlap almost perfectly, therefore the NIR version can be used to obtain the binary segmentation masks of the fruits which can be applied even on their colour version.



(a) First image. (b) Second image. (c) Third image.

Figure 3: Parallax between the colour and NIR images.

In Figure 4, the *gray-level histograms* of the three NIR images are computed and analysed. By observing the results, a greater concentration of pixels around the interval $[0, 50]$ can be identified, which should correspond to the background region. Moreover, other small peaks are present around higher values of the gray levels, which should belong to the foreground corresponding to the fruits themselves.



(a) First image. (b) Second image. (c) Third image.

Figure 4: Gray-level histograms of the NIR images.

The distribution of background and foreground pixels is slightly different among the three images and the histograms can not be clearly identified as *bimodal*. A threshold that binarizes the images cannot be trivially chosen, especially in the third one, where a confusing separation among background and foreground pixels is observed.

The NIR images are not *inherently binary*. The lack of bimodality in the histograms can be traced back to the presence of defects and textures on the fruits. Moreover, the presence of shadowy areas created by lighting variations can also have a major impact.

2.2 Pre-processing

Sources of noise may have an impacting factor on the results of the thresholding algorithms. In order to guarantee better performances during the binarization tasks a pre-processing operation has been applied to the NIR images to remove unwanted noisy signals and smooth them.

More specifically, the following filters have been considered and tested:

- **Median blur:** A *Median blur* non-linear filter using a kernel of dimension 3×3
- **Bilateral filter:** A *Bilateral filter* using kernel of dimension 7×7 and spatial standard deviation (σ_s) and intensity standard deviation (σ_r) equal to 75.

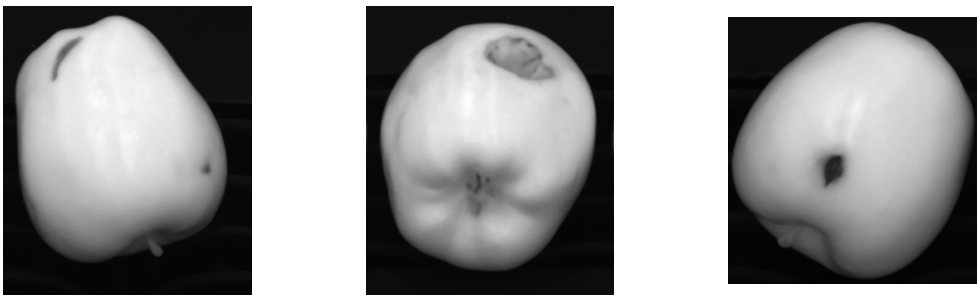
These filters are able to preserve the edges of the images during their de-noising operations. Edge preservation is important in order to obtain a binary mask that has a good precision in outlining the fruits.

Moreover, median filters are effective in removing impulse noisy signals, which can be traced back to the small brighter artefacts in the background of the images.

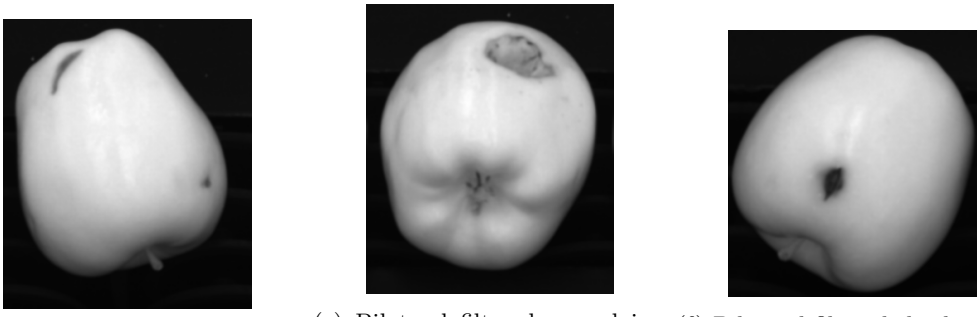
On the other hand, Bilateral filters can remove Gaussian-like noise, a result which cannot be obtained with Median filters. Although, the elimination of Gaussian-like noise, naturally presented in the images, is not deemed as necessary for the segmentation process, since small variations of intensity may not be considered impacting in the results.

The outcomes in Figure 5 are in line with the observations above, as the images are de-noised while the edges are preserved and the artifacts in the background are partially deleted.

The two filters have been tested on the set of images for a total of $3,000$ times. The Median filter turned out to be faster with an average cost of ≈ 0.0008 seconds per-image rather than the ≈ 0.0013 seconds obtained by the Bilateral Filter. Hence, the former is preferred to the latter and considered for the task.



(a) Median filtered first image. (b) Median filtered second image. (c) Median filtered third image.



(d) Bilateral filtered first image. (e) Bilateral filtered second image. (f) Bilateral filtered third image.

Figure 5: Comparison of the results of the Median and Bilateral filters on the images.

2.3 Segmentation by Binarization

In order to compute the binary masks that segments the fruits from the background *segmentation through binary thresholding* is applied over the NIR images pre-processed by *Median Blur*.

Four different methods are taken into consideration and compared in terms of time performances, robustness and quality of the output:

- **Manual Binarization by Intensity Thresholding**
- **Otsu's Algorithm**
- **A tweaked version of Otsu's Algorithm**
- **Adaptive Thresholding**

The pipeline to obtain the mask of a fruit is shown in Figure 6. The steps of the process are the following:

1. **Masking:** An initial mask of the fruit is obtained by thresholding the NIR image through a specific method in order to segment the background out of the original image.
2. **Flood-filling:** Possible holes present inside the mask of the fruit are then removed through a *flood-fill* approach.
    - The mask is padded by one-pixel-unit on each dimension with intensities equal to the ones of the background (0). This procedure guarantees that if a fruit overflows the image borders, the flood-filling operation is still applied correctly;
    - The mask is flood-filled with the same colour as the one of the foreground region (255). This way solely the holes are visible with intensity 0;
    - The padding is removed and the colours of the mask inverted. Thus, the holes end up having an intensity of 255, which is the same colour as the one of the foreground region in the mask computed at point 1;

- A *bit-wise addition* between the mask computed at the previous point and the one obtained at point 1 is performed, thus resulting in a mask where the region of the fruit (the foreground) has intensity 255 and no holes.

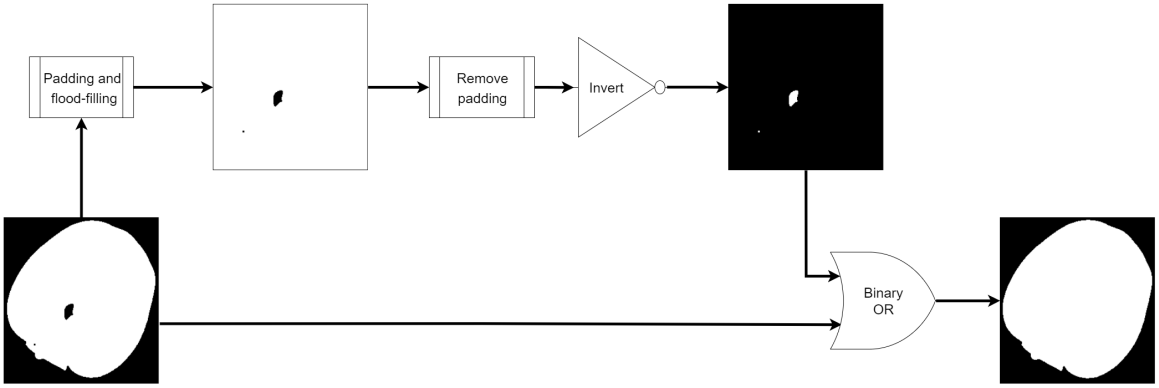
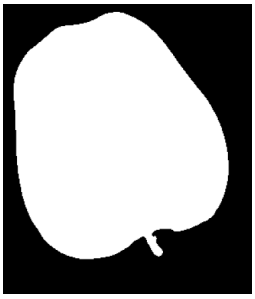


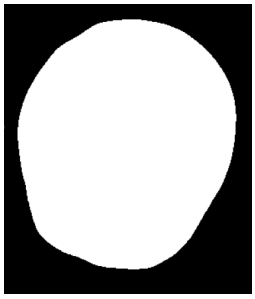
Figure 6: Illustration of the masking process.

Adaptive Thresholding uses a slightly different approach in the first part of the pipeline as explained in section 2.3.4.

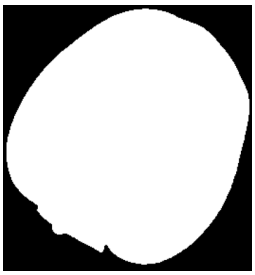
2.3.1 Manual Intensity Thresholding



(a) First image segmentation mask.

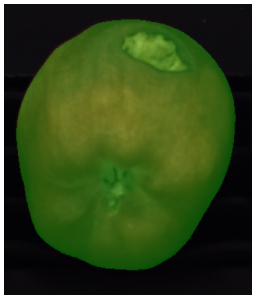


(b) Second image segmentation mask.

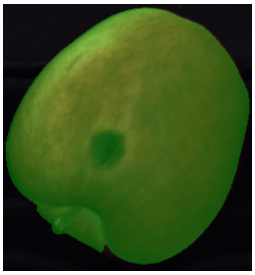


(c) Third image segmentation mask.

(d) First image segmentation outline.



(e) Second image segmentation outline.



(f) Third image segmentation outline.

Figure 7: Segmentation results obtained by Manual Intensity Binarization with threshold = 50.

The first approach is the simplest, a global threshold t is manually provided in order to segment the image I by transforming each of its pixels (i, j), such that:

$$\forall i, j,\ I(i,j) = \begin{cases} 0, & \text{if } I(i,j) \leq t \\ 255, & \text{otherwise} \end{cases}$$

Where 0 corresponds to background pixels and 255 to foreground pixels. To determine an appropriate threshold value the *gray-level histograms* of the NIR images have to be analysed.

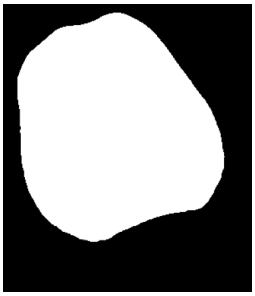
As previously explained, finding such threshold for the given images is not trivial considering that the gray-level histograms are not *bimodal*. Various experiments have reported that a good value of the threshold that guarantees a good binarization is found in the range $[20 - 50]$. The actual value chosen for t is 50.

According to Figure 7, the results of this method are satisfactory since a precise outline of each fruit is obtained. Moreover, the time taken to compute the segmentation is just ≈ 0.0003 seconds per image.

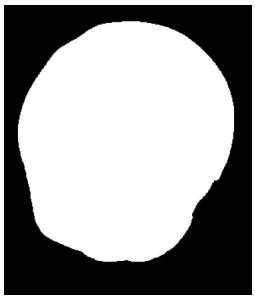
Using a global threshold that is the same for each image is possible in this kind of system, since the assumption of the presence of uniform and constant illumination is valid. However, the system is a bit too rigid and finding a global threshold for each image would be more robust to small possible changes of intensity.

2.3.2 Otsu's Algorithm

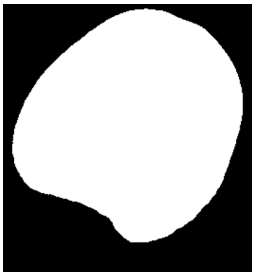
As an alternative approach, *Otsu's Algorithm* is tested. It computes for each image a different global threshold t that divides two *maximally homogeneous regions*.



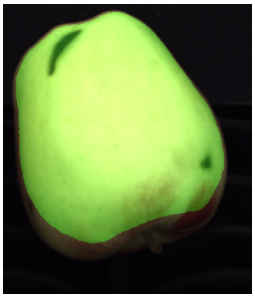
(a) First image segmentation mask ($t = 109$).



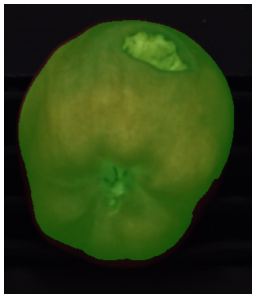
(b) Second image segmentation mask ($t = 98$).



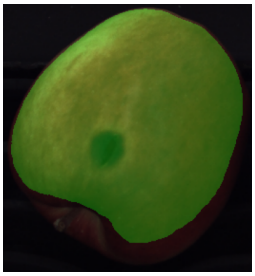
(c) Third image segmentation mask ($t = 104$).



(d) First image segmentation outline.



(e) Second image segmentation outline.



(f) Third image segmentation outline.

Figure 8: Segmentation results obtained by Otsu's Algorithm.

The method analyses an image gray-level histogram, detecting the regions in a way such that their *within-group variance* (σ_W^2) is minimal. In other words, the sum of the weighted variance of the two maximally homogeneous regions divided by t is minimized. The two variances are weighted according to the number of pixels in each region.

The algorithm uses an equivalent and faster approach by maximizing the so called *between-group variance* (σ_B^2) of the two regions. This value indicates how well they are separated. The segmentation of an image I is then performed according to the obtained threshold t as in the previous case.

Due to the nature of the method, that computes a cost function between two regions, *Otsu's Algorithm* should be robust to small changes of intensity in the images and should produce appropriate results for bimodal images.

As it can be seen in Figure 8, the segmentation results are particularly bad as darker regions of the fruits are classified as "background". Pixels of these areas are closer in intensity to the ones of the background rather than to the ones that make up the fruit, thus a higher threshold t than the best one is obtained in order to maximize σ_B^2.

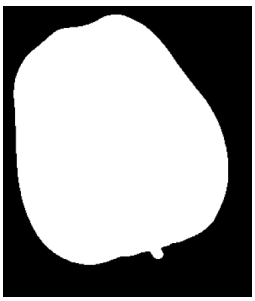
Regarding the given instances, the quality of the segmentation could be considered sufficient, since the defects are all present between the bounds of the obtained binary masks and they could be all eventually detected in successive steps. However, considering that the aim of the project is to build a system that would be used in an industrial setting, the results cannot be recognized as valid. In newly introduced instances defects could still be present in the darker regions of the fruits and *Otsu's Algorithm* could not be able to provide a binary segmentation that includes them. Hence, these defects would be unnoticed.

On a better note, the average time taken to compute the segmentation of an image is comparable to the one of *Manual Intensity Thresholding*, with ≈ 0.0004 seconds per image.

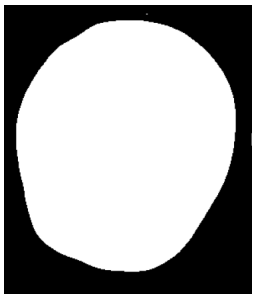
2.3.3 Tweaked Otsu's Algorithm

As previously observed, *Otsu's Algorithm* positive aspects reside on:

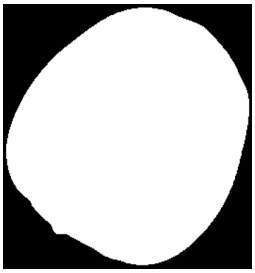
- **Speed** of the thresholding process;
- **Robustness** at calculating a global threshold t which is different for each image, thus providing still accurate results even if small lighting variations occur.



(a) First image segmentation mask ($t = 32$).

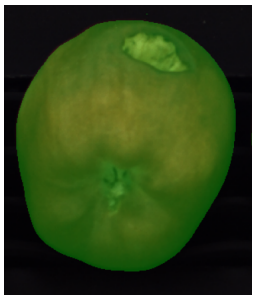


(b) Second image segmentation mask ($t = 29$).

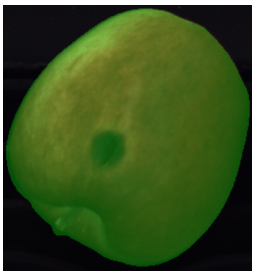


(c) Third image segmentation mask ($t = 31$).

(d) First image segmentation outline.



(e) Second image segmentation outline.



(f) Third image segmentation outline.

Figure 9: Segmentation results obtained by Tweaked Otsu's Algorithm.

Nonetheless, on the given instances, *Otsu's Algorithm* was not able to provide a global threshold to outline the fruits correctly. In order to exploit its benefits while improving its performance a new simple procedure is proposed which will be called *Tweaked Otsu's Algorithm*.

Given an image I, it obtains a global threshold t using *Otsu's Algorithm* and then it multiplies the obtained global threshold t by a *tweak factor* in order to decrease (or increase) its value obtaining a new threshold t'.

$$t = \texttt{otsu}(I)$$

$$t' = t \cdot \text{tweak factor}$$

Tuning the tweaking factor on demand, a better segmentation of the fruits can be obtained according to t', while maintaining the robustness to different light intensities by computing the threshold on each image and not as a global value.

For the task, a *tweak factor* of value 0.3 has been chosen. Figure 9 shows that results obtained through this approach are optimal as the whole fruit is segmented from the background in its entirety in each image.

Regarding the performances, the time taken to segment each image is comparable to the one of *Otsu's Algorithm* and *Manual Intensity Thresholding*, namely ≈ 0.0005 seconds.

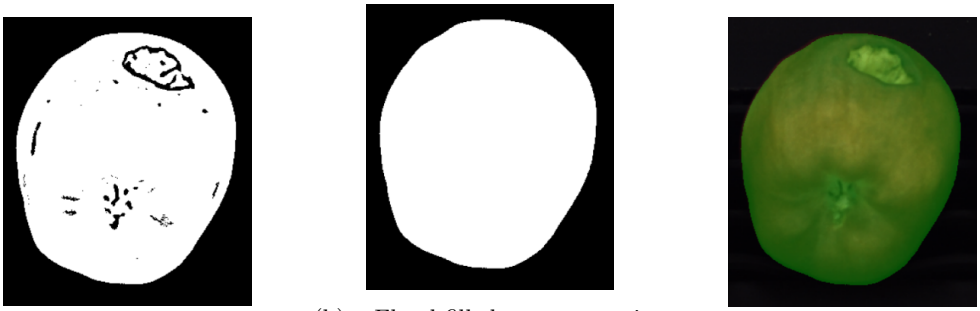
2.3.4 Local Adaptive Thresholding

As a final approach, the *Local Adaptive Thresholding* method was tested. Differently from previous algorithms, *Local Adaptive Thresholding* does not compute a global value to segment the image. Instead, a local threshold is calculated for each pixel of the image, taking as reference a window of pixels around it, called *neighbourhood*. This technique deals efficiently with non-uniformely illuminated scenes.

The threshold of each pixel p in an image I, $t(p)$, is computed by a fast operator that chooses as a threshold the mean of the neighbourhood of p ($\mu(N_p)$). In order to achieve a better segmentation result, especially when pixels lack background and foreground pixels in their neighbourhood, a constant C can be subtracted from the result in order to improve it.

$$\forall p \in I, t(p) = \mu(N_p) - C$$

For the given problem a value of C equal to 5 has been chosen, along with a neighbourhood of size 11×11 for each pixel.



(a) Binary segmentation mask. (b) Flood-filled segmentation mask. (c) Outlined fruit.

Figure 10: Example of the segmentation results obtained by Local Adaptive Thresholding on the second image.

Figure 10 shows an example of the masking process, where *flood-filling* is applied to remove the holes from the white mask.

Given the nature of the method which computes a threshold for each pixel, it is expected to be slower than the previously analysed global thresholding methods. In fact it manages to segment an image in around 0.0009 seconds, taking higher time than all the previous ones. A further reason of this overhead

is even found in the fact that two *flood-fill* operations have to be carried out. The first is applied to fill the background, which is initially mainly white due to the computation of the threshold at a local level (As seen in Figure 11). The second is the classic *flood-filling* applied to remove the holes from the white mask.

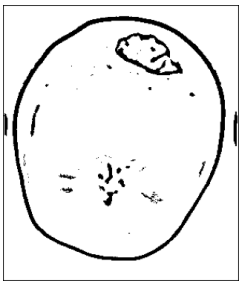
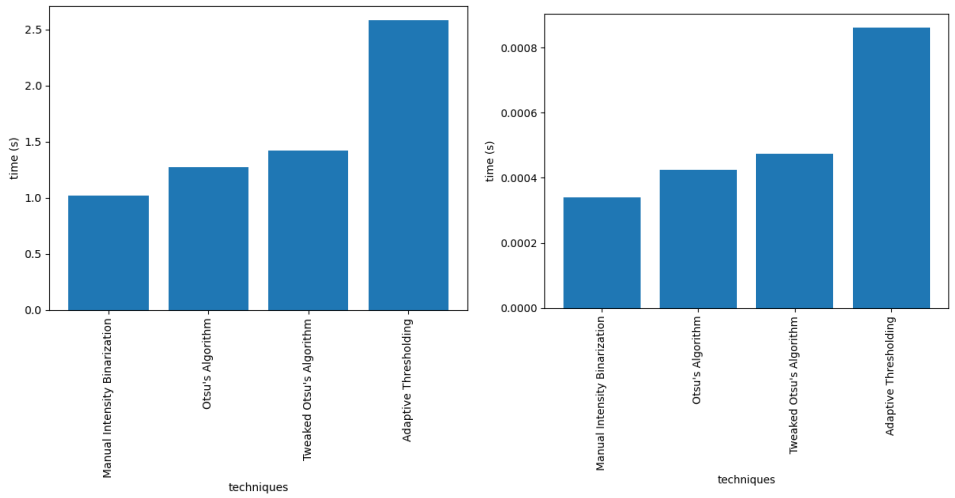


Figure 11: Example of the result of the application of Adaptive Thresholding before applying Flood-Filling.

However, the method manages to precisely segment the fruits even if small light changes are present around their lower part. Nonetheless, applying a local thresholding method in a system where the assumption of uniform illumination is present, such as the one of this project, is excessive. As previously observed, the *Tweaked Otsu's Algorithm* and the *Manual Intensity binarization* methods are able to obtain equivalently good results in less time.

2.3.5 Comparisons of the Algorithms



(a) Total time.

(b) Average time per instance.

Figure 12: Comparison of the execution time of the Thresholding algorithms on $3,000$ instances.

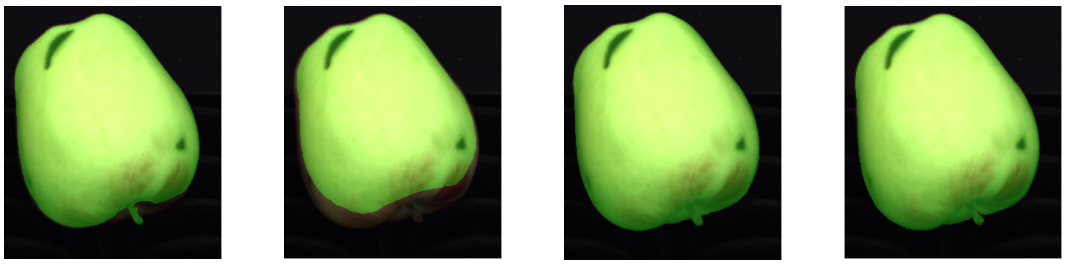
Table 1 sums up the characteristics of the tested Binarization Thresholding algorithms in terms of quality, speed and robustness.

Figure 12 illustrates the total and average time taken to apply the segmentation of the three images repeated for a total of $3,000$ times. *Manual Intensity Binarization* is the fastest method, directly followed by *Otsu's Algorithm* and *Tweaked Otsu's Algorithm* which speeds are still great, but not as much as the first method. On the other hand *Local Adaptive Thresholding* is the slowest between all methods as it takes approximately at least double the time to segment an image with respect to the others.

	Manual Intensity Binarization	Otsu's Algorithm	Tweaked Otsu's Algorithm	Local Adaptive Thresholding
Quality	High	Low	High	High
Speed	Fastest	Fast	Fast	Slow
Individual per-image	No	Yes	Yes	Yes

Table 1: Comparison of the Binarization Thresholding Algorithms.

In terms of the quality of segmentation, high quality results are obtained by all methods except for *Otsu's Algorithm* which fails to outline the fruits in a precise manner. Figure 13 recaps the segmentation results of the various methods applied on the first fruit.



(a) Manual Intensity Binarization.

(b) Otsu's Algorithm.

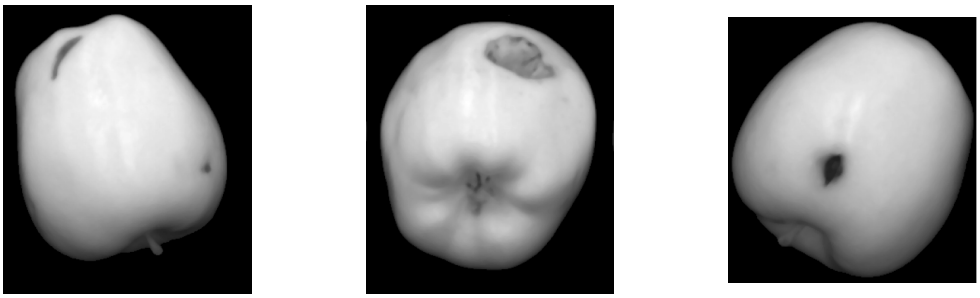
(c) Tweaked Otsu's Algorithm.

(d) Local Adaptive Thresholding.

Figure 13: Comparison of the quality of the segmentation of the Thresholding algorithms on the first fruit.

Regarding the robustness of the methods, mostly all of them can deal with changes of illumination in the scene and define Binarization Thresholds (one global, or one per pixel as in *Local Adaptive Thresholding*) specific for each instance. The sole exception is with *Manual Intensity Binarization* as it considers a threshold which is equal for all the images, and is thus not robust.

Tweaked Otsu's Algorithm is hence considered the best method and it is finally selected for the segmentation task. The masks obtained through its computations are finally applied over the pre-processed Near Infra-Red images as seen in Figure 14.



(a) First image.

(b) Second image.

(c) Third image.

Figure 14: Segmentation results.

2.4 Defects Identification

After the segmentation of the fruits from the background is performed the second part of the task is addressed. It consists in identifying the defects on the masked fruits.

Judging by the pictures of the fruits, the defects present strong edges, therefore an edge extraction algorithm would be ideal to identify them.

This second part of the task is divided in the following steps for each image:

- Obtaining an edge mask through *Canny's edge extraction method*;
- Filling the edges of the defects, obtaining a blob for each edge;
- Cleaning-up the edge mask from noisy artefacts.

2.4.1 Edge Extraction

In order to extract the edges, *Canny's method* is applied to the filtered and masked Near Infra-Red images.

The method proposed by Canny consists in the following steps:

- The image is filtered through a *Gaussian filter*, leveraging on the separability of the Gaussian function to speed-up the computations. He showed that for 1D vectors (where edges are considered noisy steps) the filtering operations lead to the best detection of edges. The same concept applies for 2D matrices, namely images.
- Then, a *Sobel kernel* filtering operation in both horizontal and vertical directions is performed in order to to get the first derivatives of each pixel in both directions (G_x, G_y). From these values the *Edge Gradient* and the *Gradient directions* are obtained.

$$\text{Edge Gradient}(G) = \sqrt{G_x^2 + G_y^2}$$

$$\text{Gradient Direction}(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

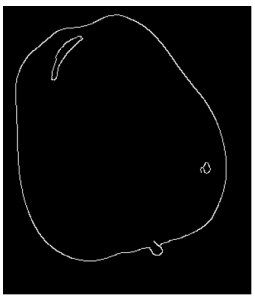
- *Non-Maxima Suppression* is applied. For every pixel it is checked whether it is a local maximum in its neighborhood in the direction of gradient. The pixel is kept as an edge point if that occurs, otherwise it is discarded.
- *Hysteresis* is finally executed to furtherly filter actual edge points. It makes use of two thresholds: a higher one t_h and a lower one t_l. Pixels with intensity gradient above or equal to t_h are considered edge points, while pixels with intensity gradient below t_l are discarded. Pixels who lie between these two thresholds are classified as edges if and only if they are connected to "sure-edge" pixels.

Note: *Canny's method* implementation in *OpenCV* does not perform the initial *Gaussian Smoothing*, therefore the operation was manually applied with a value of $\sigma = 1$ and a kernel size computed by a rule-of-thumb:

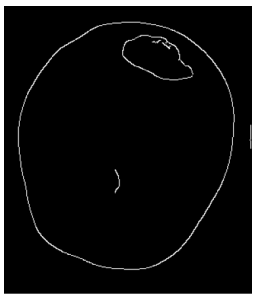
$$\text{kernel size} = (2 \cdot k + 1 \times 2 \cdot k + 1), \text{ with } k = \lceil 3 \cdot \sigma \rceil$$

Regarding the *hysteresis* thresholds, the best results have been obtained with $t_h = 130$ and $t_l = 60$.
To remove the edges along the border of the fruit the binary masks that segment the fruit from the background have been eroded by a rectangular *structuring element* of size 15×15 and then applied over the edge masks. Thanks to this operation, solely the edges of the defects, along with some noisy artifacts, are preserved.

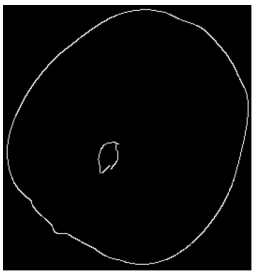
Figure 15 illustrates the edge extraction process.



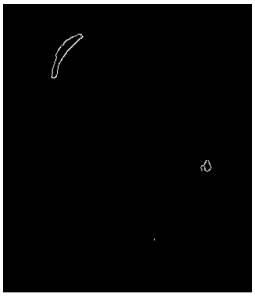
(a) First image edge mask.



(b) Second image edge mask.

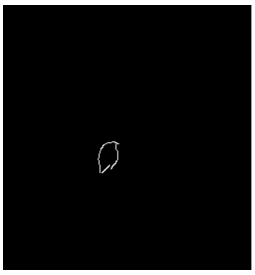


(c) Third image edge mask.

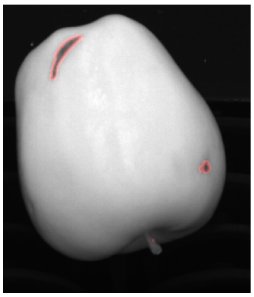


(d) First image defects edge mask.

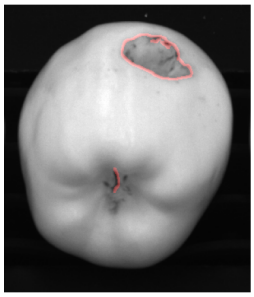
(e) Second image defects edge mask.



(f) Third image defects edge mask.



(g) First image outlined edges.



(h) Second image outlined edges.

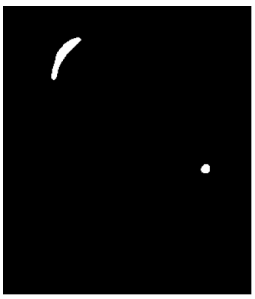
(i) Third image outlined edges.

Figure 15: Edge extraction process.

2.4.2 Defects Filling

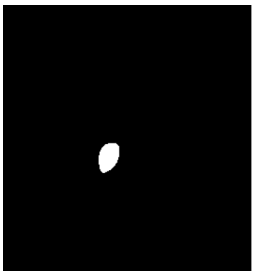
In order to better identify the area of the defects a successive operation is performed. The edges of the defects in the edge mask are filled through a *Closing* operation with a circular *structuring element* of size 50×50 and a blob is obtained in correspondence of each defect area even if some noisy artefacts remain.

To remove them a *Median Blur* of kernel 7×7 is performed. The idea behind this operation is that small artefacts can be considered as some kind of *impulse noise*. Hence, a Median filter is able to remove them while preserving the structure of the blobs in correspondence of the defects. The cleaned-up edge mask and the defects highlighted by them can be seen in Figure 16.

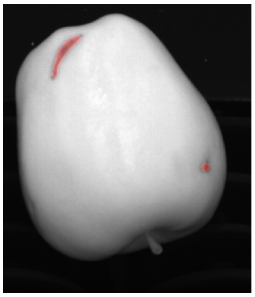


(a) First image blobs mask.

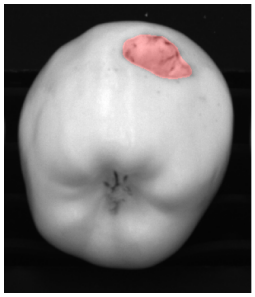
(b) Second image blobs mask.



(c) Third image blobs mask.



(d) First image blobs highlight.



(e) Second image blobs highlight.

(f) Third image blobs highlight.

Figure 16: Blobs of the defects.

2.5 Results

An algorithm performing the steps described above is built and it can be executed via command line or by the method `detect_defects`.

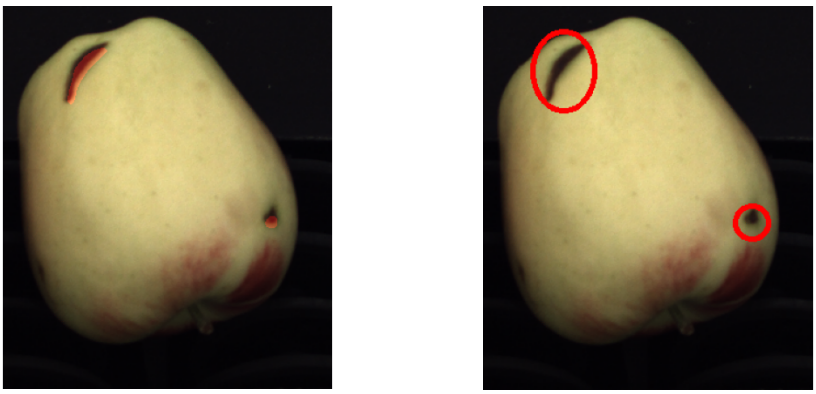
The algorithm:

1. Filters the NIR image by *Median Blur* with a kernel 5×5;
2. Applies *Tweaked Otsu's Algorithm* to obtain the segmented fruit with a given *tweak factor*;
3. Uses *Canny's Algorithm* with given thresholds t_l and t_h by firstly blurring the masked image through *Gaussian Blur* with a given σ;
4. It obtains the defect blobs by the edge mask and removes the noisy artifacts;
5. Shows the blobs of the defects, counts their occurrences and highlights their areas through ellipses.

Once again, it can be observed in Figure 17 that the results on the given images are pretty satisfactory using the previously identified parameters:

- *tweak factor* $= 0.3$;
- $t_l = 60$;
- $t_h = 130$;
- $\sigma = 1$.

Finally, the time for detecting the defects is computed by applying the `detect_defects` function on each image $1,000$ times, for a total of $3,000$ executions. The algorithm is fast, as it can detect the defects of an image with a mean time of ≈ 0.03 seconds.



(a) First image defects.

(b) First image defect areas.

(c) Second image defects.

(d) Second image defect areas.

(e) Third image defects.

(f) Third image defect areas.

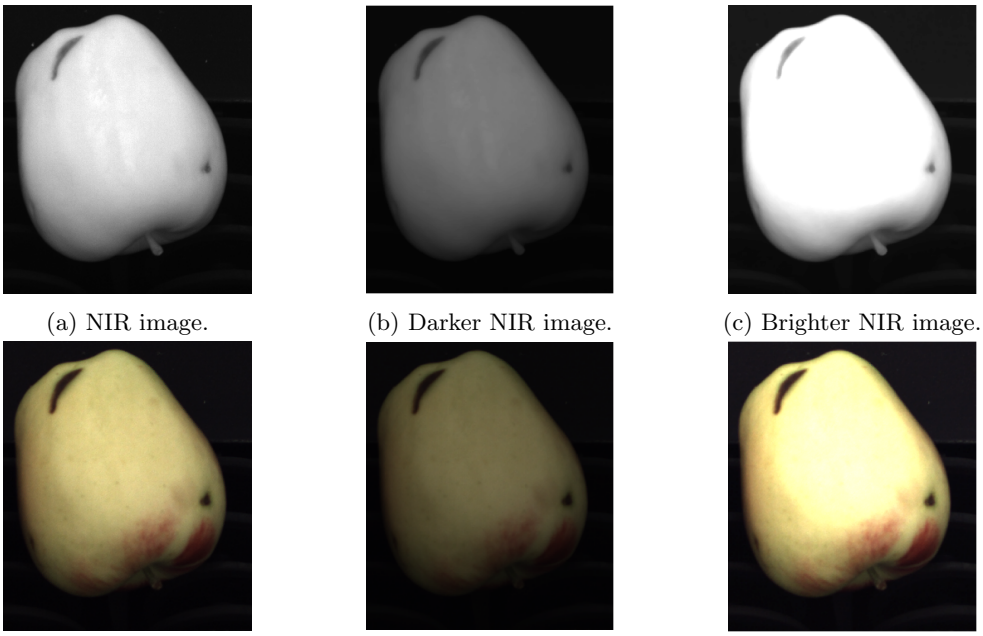
Figure 17: Results of the defects detection system implemented by the `detect_defects` algorithm.

2.5.1 Portability

The system is designed to work in an environment with a uniform and constant illumination, which should apply even for images different than the given ones. Nonetheless, a robust segmentation method is provided thus the system should adapt easily to environments with different illumination.

As a result, it is easily portable to other industrial settings by simply modifying some of its parameters. In order to prove so, a version of the colour images and the NIR images with a darker and brighter contrast intensity are computed from their initial versions and shown in Figure 18.

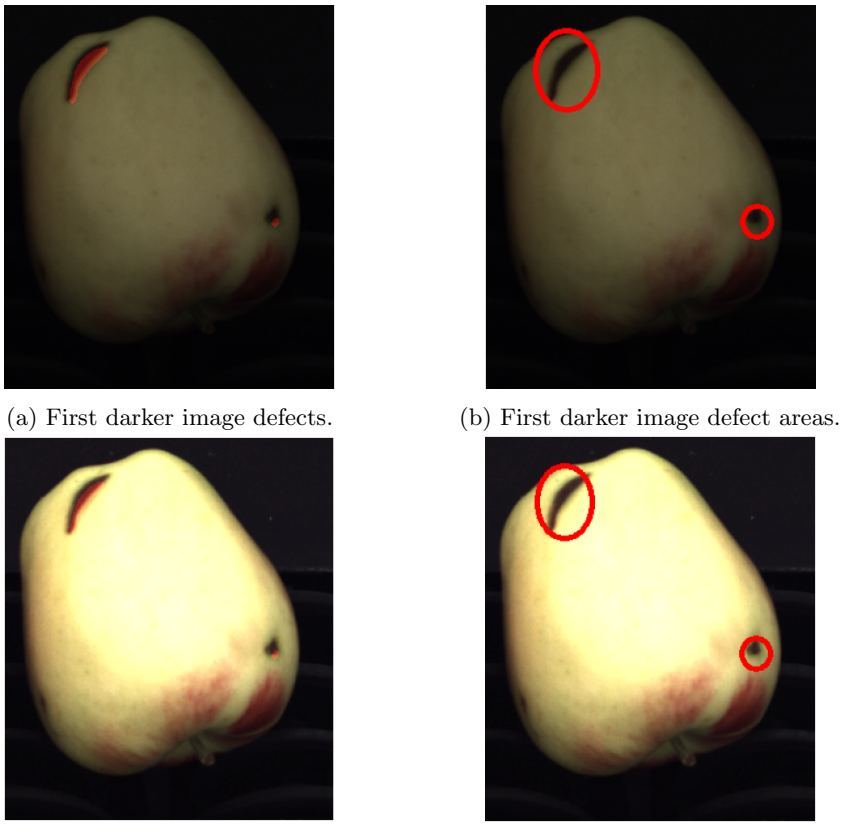
With the same tweak factor and slightly different Canny thresholds ($t_l = 30$ and $t_h = 60$) the defect detection works on the darker images. Similarly, for thresholds ($t_l = 80$ and $t_h = 180$) the system works on the brighter images. An example of the results for the first image are shown in Figure 19.



(a) NIR image. (b) Darker NIR image. (c) Brighter NIR image.

(d) Colour image. (e) Darker colour image. (f) Brighter colour image.

Figure 18: Example of the darker and brighter computed versions of the first image.



(a) First darker image defects. (b) First darker image defect areas.

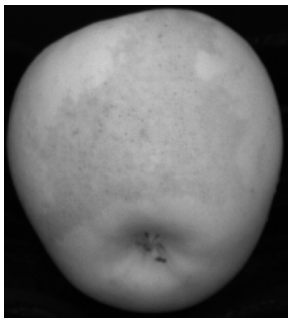
(c) First brighter image defects. (d) First brighter image defect areas.

Figure 19: Results of the defects detection system implemented by the `detect_defects` algorithm on the modified versions of the first image.

3 Second Task: Russet Detection

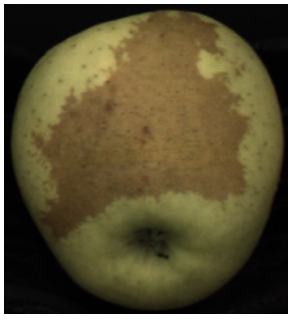
The second task concerns the development of a system aimed at locating unwanted reddish-brown areas, called russets, in the fruits with no false positives (if possible). The fruit's images, presented in Figure 20, are acquired using both a *Near Infra-Red* and a colour camera.

(a) First NIR image.



(b) Second NIR image.

(c) First colour image.

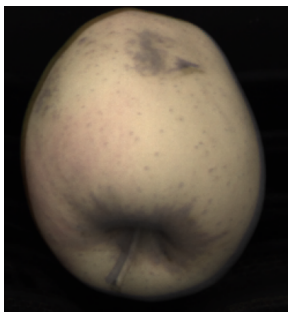


(d) Second colour image.

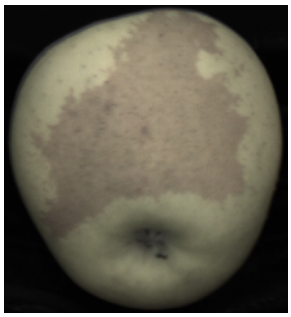
Figure 20: Source images of the second task.

3.1 Image Analysis

In Figure 21 the parallax between the source images is analysed. Unlike in the previous task, the two cameras have a greater angle of orientation and the parallax appears to be higher and inconsistent between the colour and the Near infra-red images.



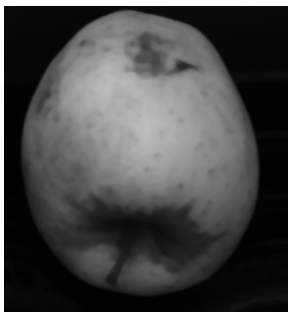
(a) First image.



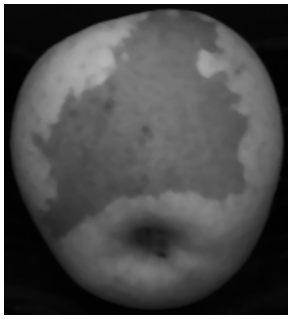
(b) Second image.

Figure 21: Parallax between the colour and NIR images.

Hence, in order to mask the background from the fruits the grayscale version of the colour images (Figure 22) is obtained in spite of using the NIR ones.



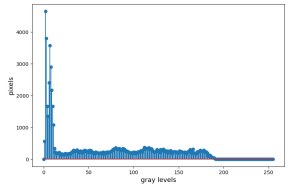
(a) First image.



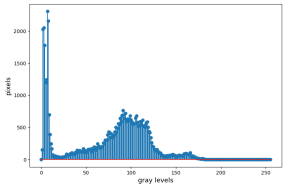
(b) Second image.

Figure 22: Grayscale version of the colour images.

In Figure 23 the *gray-level histograms* of the two grayscale images are illustrated. The histogram of the first image is not *bimodal*, although the one of the second shows two distinct distributions of pixels.



(a) First image.



(b) Second image.

Figure 23: Gray-level histograms of the pre-processed grayscale images.

3.1.1 Pre-processing and Segmenting

As explained in the previous task, a good segmentation of the fruits implies a pre-processing operation. To achieve that, the gray-scale version of the images are filtered by a *Median filter* of *kernel* 5×5.

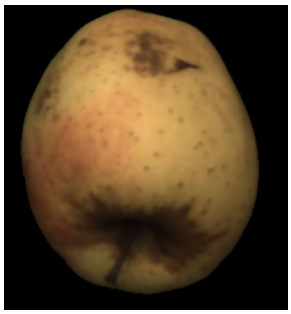
The pre-processed grayscale images are used to obtain the binary masks of the fruits that separate them from the background. *Tweaked Otsu's Algorithm* is used as a segmentation technique, in a similar fashion to the previous task, using a *tweak factor* of 0.4. This approach has been used because of its speed and robustness, assuming again an uniform illumination of the environment. The small holes inside the white part of the masks are then removed through a *flood filling* approach.

On the other hand, the colour images are pre-processed through a *Bilateral filter* using kernel of dimension 7×7 and spatial standard deviation (σ_s) and intensity standard deviation (σ_r) equal to 75. The smoothing process is expected to help make similar colour regions more uniform, less noisy and easier to identify.

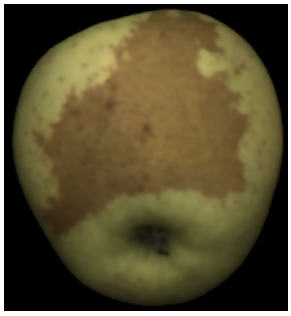
The usage of a *Bilateral filter* instead of a *Median filter* to process the colour version of the images is related to the fact that the removal of the *Gaussian Noise*, naturally presented in the images, may have a great impact in enhancing the performances of the detection of the russets.

Bilateral filters are edge-preserving, therefore they are expected to avoid blurring the contours of the russets and lead to a better and more precise detection of them.

Finally, the masks obtained by the binary segmentation of the gray-scale images are applied over the pre-processed colour version of the images and the results are shown in Figure 24. The pre-processed colour images are used to detect the russet as it will be explained in the following sections.



(a) First image.



(b) Second image.

Figure 24: Binary masks applied over the colour images.

3.1.2 Analysis of Colour Spaces

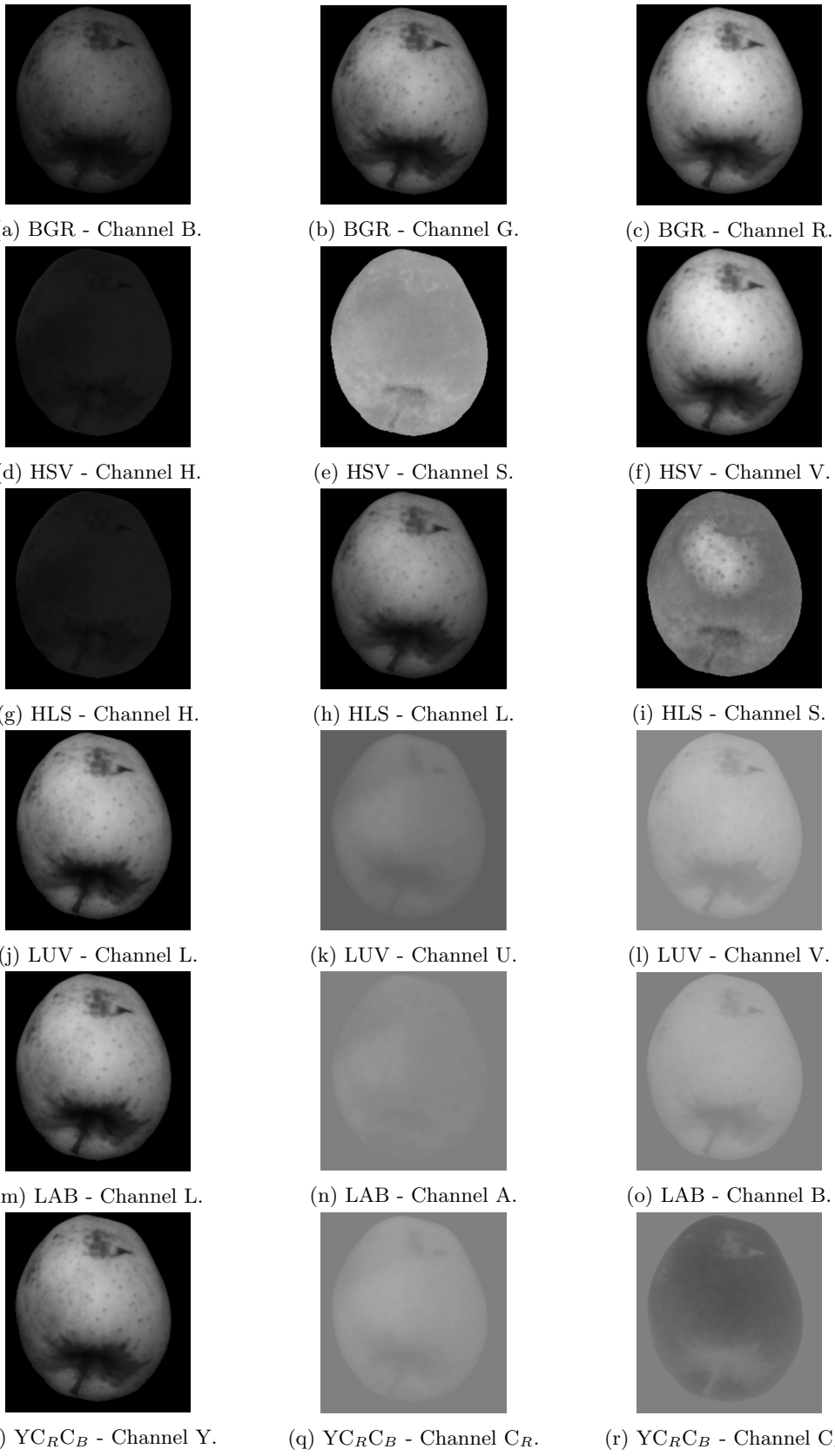
Colour spaces are representations of colours in different dimensions according to a specific organization. Different colour space representations of the images have been considered and the channels which constitute each one of them have been analysed.

The observed colour spaces are:

- **BGR**: the original colour space of the images, composed of a Blue (B), Green (G) and Red (R) light channels;
- **HSV**: a colour space consisting of a channels representing the hue of the image (H), the saturation of the image (S) and the brightness of the image (L);
- **HLS**: a colour space consisting of a channel representing the hue of the image (H), the lightness of the image (L) and the saturation of the image (S);
- **LUV**: A transformation of the *CIE XYZ* colour space, which attempts perceptual uniformity and which is extensively used in computer graphics (composed of the channels L, U and V);
- **LAB**: A transformation of the *CIE XYZ* colour space, which is composed of a channel about perceptual lightness (L), and two other channels related to the four unique colours of human vision: red, green, blue and yellow (A and B). Due to the nature of this colour space, which describes the difference in hue of the colour perceived by human vision in its last two channels, good results in detecting the russets are expected;
- **YC$_R$C$_B$**: a colour space consisting of a channel about the brightness of the image (Y), one about the red differences in the image (C$_R$) and one about the blue differences in the image (C$_B$);

From a preliminary visual analysis of the channels of each colour space, as seen in Figure 25 and 26, the following considerations are expressed:

- In the *BGR* colour space, the most interesting channels are the G and the R one, since too poor contrast of intensity is present in the B channel.
- In the *HSV* colour space, the first and second channel (H and S) contain the most important information. In particular it seems they are able to visually distinguish between the russet and the healthy part of the fruit in the second image while not so good distinctions can be observed in the first image.
- The same applies for the *HLS* colour space, except the channels with the greatest amount of information are the first and third (H and S).
- In the *LUV* colour space the channels with most information about distinction of intensities are the last two (U and V). In particular, the V channel is the one that better visually segments the healthy part and the russet part of both fruits.



(a) BGR - Channel B. (b) BGR - Channel G. (c) BGR - Channel R.

(d) HSV - Channel H. (e) HSV - Channel S. (f) HSV - Channel V.

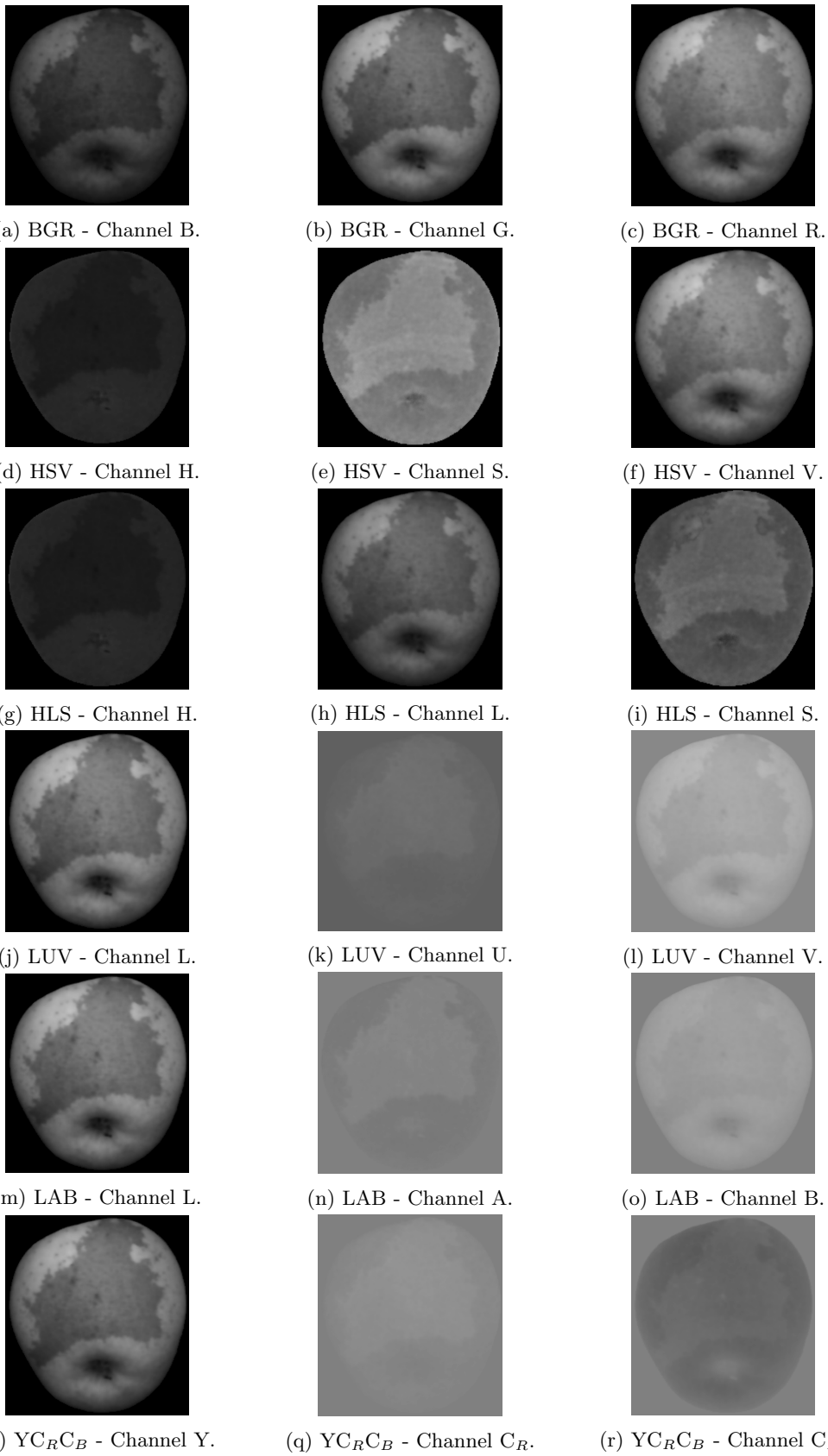
(g) HLS - Channel H. (h) HLS - Channel L. (i) HLS - Channel S.

(j) LUV - Channel L. (k) LUV - Channel U. (l) LUV - Channel V.

(m) LAB - Channel L. (n) LAB - Channel A. (o) LAB - Channel B.

(p) YC_RC_B - Channel Y. (q) YC_RC_B - Channel C_R. (r) YC_RC_B - Channel C_B.

Figure 25: Analysis of the channels of the colour spaces in the first image.



(a) BGR - Channel B. (b) BGR - Channel G. (c) BGR - Channel R.

(d) HSV - Channel H. (e) HSV - Channel S. (f) HSV - Channel V.

(g) HLS - Channel H. (h) HLS - Channel L. (i) HLS - Channel S.

(j) LUV - Channel L. (k) LUV - Channel U. (l) LUV - Channel V.

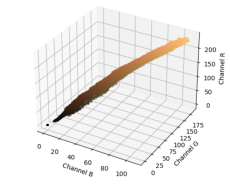
(m) LAB - Channel L. (n) LAB - Channel A. (o) LAB - Channel B.

(p) YC_RC_B - Channel Y. (q) YC_RC_B - Channel C_R. (r) YC_RC_B - Channel C_B.

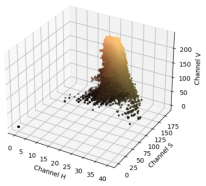
Figure 26: Analysis of the channels of the colour spaces in the second image.

- In the *LAB* colour space, similarly to *LUV*, the last two channels (A and B) contain most information to segment the russet of the fruit. In particular, it looks like the A channel provides a better distinction in the second image, while the B channel is better at visually distinguish the russet of the first image.
- Finally, in YC_RC_B the most interesting information resides in the last two channels (C_R and C_B) and the last looks like the best at identifying a difference between the russet and the healthy parts of both fruits.

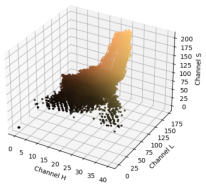
In Figure 27 and 28, a three-dimensional plot of the pixels colour distribution in each colour space is illustrated for the two images.



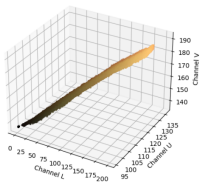
(a) BGR colour space.



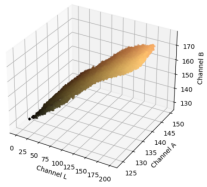
(b) HSV colour space.



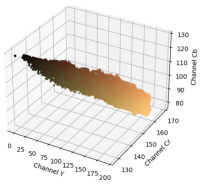
(c) HLS colour space.



(d) LUV colour space.

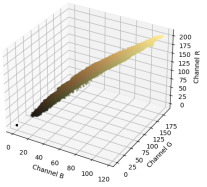


(e) LAB colour space.

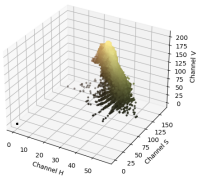


(f) YC_RC_B colour space.

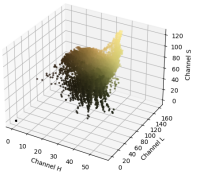
Figure 27: Colour distribution of pixels of the first image in different colour spaces.



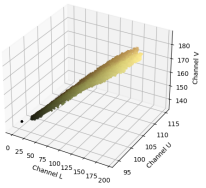
(a) BGR colour space.



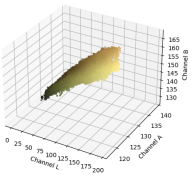
(b) HSV colour space.



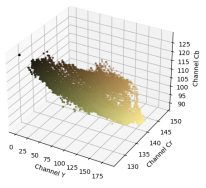
(c) HLS colour space.



(d) LUV colour space.



(e) LAB colour space.

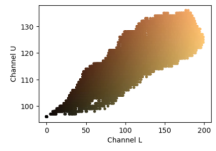


(f) YC_RC_B colour space.

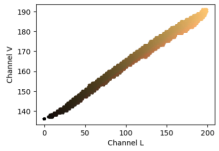
Figure 28: Colour distribution of pixels of the second image in different colour spaces.

The results of this analysis highlight that the BGR colour space is the least interesting one, as different colours are linearly distributed and difficult to distinguish. All the other distributions generally present interesting patterns. In some of them more or less distinct clouds of points of the same colour can be identified.

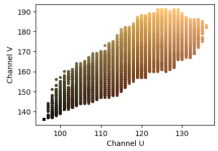
A more interesting and clear analysis of the distribution of the pixels in the different colour spaces is then performed while considering all the possible combinations of pairs of channels.



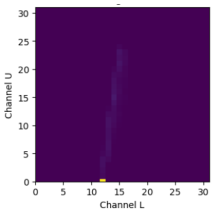
(a) LUV - channels L and U.



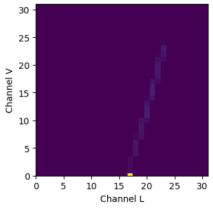
(b) LUV - channels L and V.



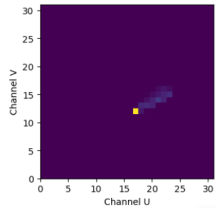
(c) LUV - channels U and V.



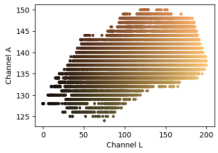
(d) LUV - channels L and U histogram.



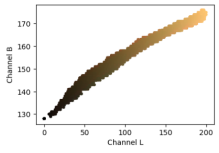
(e) LUV - channels L and V histogram.



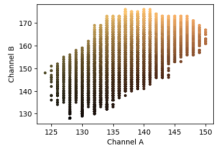
(f) LUV - channels U and V histogram.



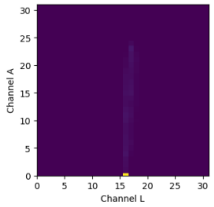
(g) LAB - channels L and A.



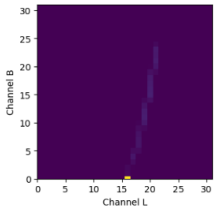
(h) LAB - channels L and B.



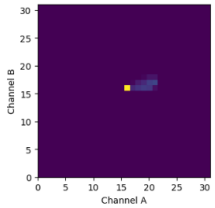
(i) LAB - channels A and B.



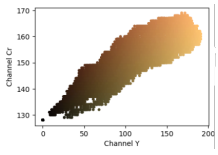
(j) LAB - channels L and A histogram.



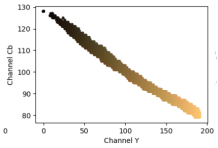
(k) LAB - channels L and B histogram.



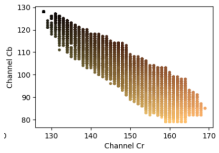
(l) LAB - channels A and B histogram.



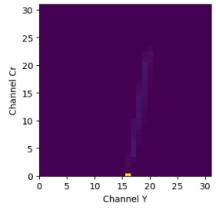
(m) YC_RC_B - channels Y and C_R.



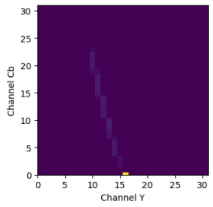
(n) YC_RC_B - channels Y and C_B.



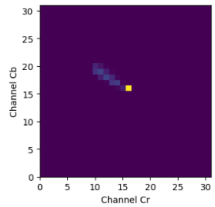
(o) YC_RC_B - channels C_R and C_B.



(p) YC_RC_B - channels Y and C_R histogram.

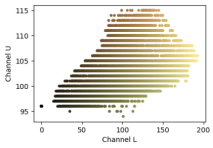


(q) YC_RC_B - channels Y and C_B histogram.

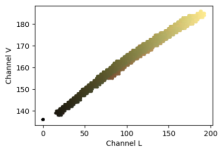


(r) YC_RC_B - channels C_R and C_B histogram.

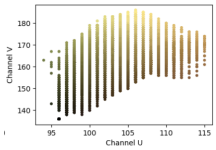
Figure 29: Colour distribution of pixels of the first image in different colour spaces considering pairs of channels.



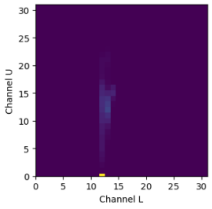
(a) LUV - channels L and U.



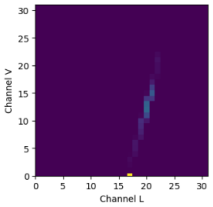
(b) LUV - channels L and V.



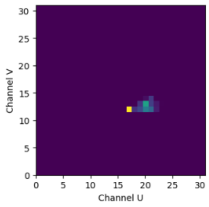
(c) LUV - channels U and V.



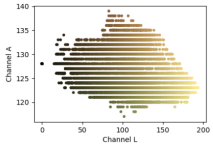
(d) LUV - channels L and U histogram.



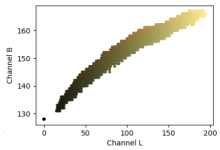
(e) LUV - channels L and V histogram.



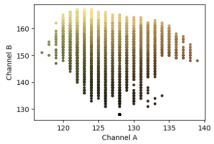
(f) LUV - channels U and V histogram.



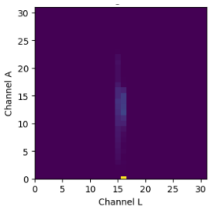
(g) LAB - channels L and A.



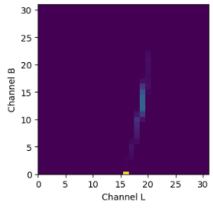
(h) LAB - channels L and B.



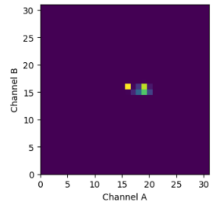
(i) LAB - channels A and B.



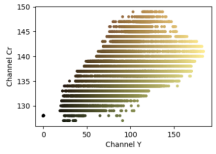
(j) LAB - channels L and A histogram.



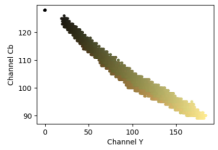
(k) LAB - channels L and B histogram.



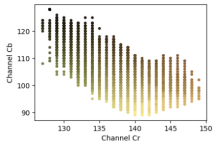
(l) LAB - channels A and B histogram.



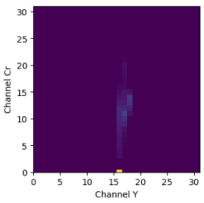
(m) YC_RC_B - channels Y and C_R.



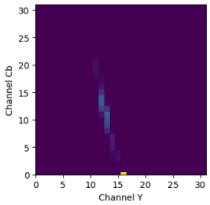
(n) YC_RC_B - channels Y and C_B.



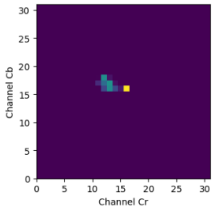
(o) YC_RC_B - channels C_R and C_B.



(p) YC_RC_B - channels Y and C_R histogram.



(q) YC_RC_B - channels Y and C_B histogram.



(r) YC_RC_B - channels C_R and C_B histogram.

Figure 30: Colour distribution of pixels of the second image in different colour spaces considering pairs of channels.

More specifically, for each image, a colour distribution of the pixels is observed alongside a 2D histogram which is particularly useful to identify denser clusters of points. The results express that:

- No interesting patterns or clusters can be identified in the BGR colour space considering different pairs of channels. Therefore, it is not useful to segment the russet region of the fruits from their healthy component.
- The H and S channels of the HSV colour space are able to identify some denser regions of pixels of different colours in the second fruit, while no regions are clearly identifiable in the first fruit.
- A similar result as the previous one is obtained in the HLS colour space, regarding the channels H and S.
- Distinct denser regions of colour are observable in the LUV colour space regarding the channels U and V especially in the second fruit.
- The LAB colour space separates regions of colours in both fruits when the A and B channels are considered. Once again, the clearer distinctions are seen in the second image. Nonetheless, LAB provides the best results among all the colour spaces in identifying different clusters of colors in the first image. This is particularly clear by looking at its 2D histogram.
- Finally, in the YC_RC_B colour space similar results to the one of LUV are obtained with the channels C_R and C_B.

From this analysis it can be hypotesized that good results in segmenting the russet region from the healthy part of the fruit with a colour-based approach can be obtained with the **LUV**, **LAB** and $\mathbf{YC_RC_B}$ colour spaces as seen in Figures 29 and 30.

In particular, it looks like that **LAB** could yield the best results. Other colour spaces (BGR, HSV and HLS) should instead be ignored.

3.2 Russet Extraction by Clustering

After the preliminary analysis, the actual colour-based segmentation of the russet region from the rest of the fruit is performed, while considering the previously analysed colour spaces.

In particular, three different colour segmentation techniques are tested:

- Colour quantization with *K-means*
- Colour quantization with a *Gaussian Mixture* clustering approach;
- Colour segmentation by considering the *Mahalanobis distance* from a reference colour.

3.2.1 K-Means Quantization

The first experimented approach consists in the automatic quantization of different colour regions by using the *K-means* clustering method. It consists in finding a given number of clusters (n) by:

1. Assigning n initial clusters centers among the data points;
2. Until convergence occurs:
    - Assign each data point to the nearest cluster center;
    - Set all the cluster centers to the mean of the points of each cluster.

The reasons behind the selection of this approach reside in the following facts:

- It is an automatic clustering method which does not require to know any information on the colour of the russet or the healthy parts of the fruit in order to segment them. Therefore, even if the colours in the images are fairly different, as long as the russet regions and the healthy parts of the fruit have distinct colours, they should be divided in separate cluster.

- Denser clusters of points are observed in the previously analysed 2D histograms of the images. According to the 2D distributions of colours they also seem to sometimes be described by circular regions, which K-Means is good at identifying.

In particular, K-Means is computed on the images while imposing to find 3 different clusters of points, consisting in: the background, the russet regions and the healthy part of the fruit.

The algorithm is firstly run on relevant colour spaces while all the channels are provided, obtaining bad results for both images. In particular, it occurs that the darker russet is sometimes classified as background because of its closer similarity to that part of the image or that areas of the russet are mixed with areas of the healthy fruit.

(a) First image - LUV colour space.

(b) Second image - LUV colour space.

(c) First image - LAB colour space.

(d) Second image - LAB colour space.

(e) First image - YC_RC_B colour space.

(f) Second image - YC_RC_B colour space.

Figure 31: K-Means quantization results on the LUV, LAB and YC_RC_B colour spaces.

The clustering is repeated while considering just the channels that contain the most important pieces of information for each colour space. Figure 31 shows the results of this operation on the LUV, LAB and YC_RC_B colour spaces.

Areas of the first fruit appear not to be well divided, while the second fruit is almost perfectly divided in the three desired clusters when LAB is considered.

Some noisy artefacts are present in the division obtained by the three colour spaces. Moreover, the

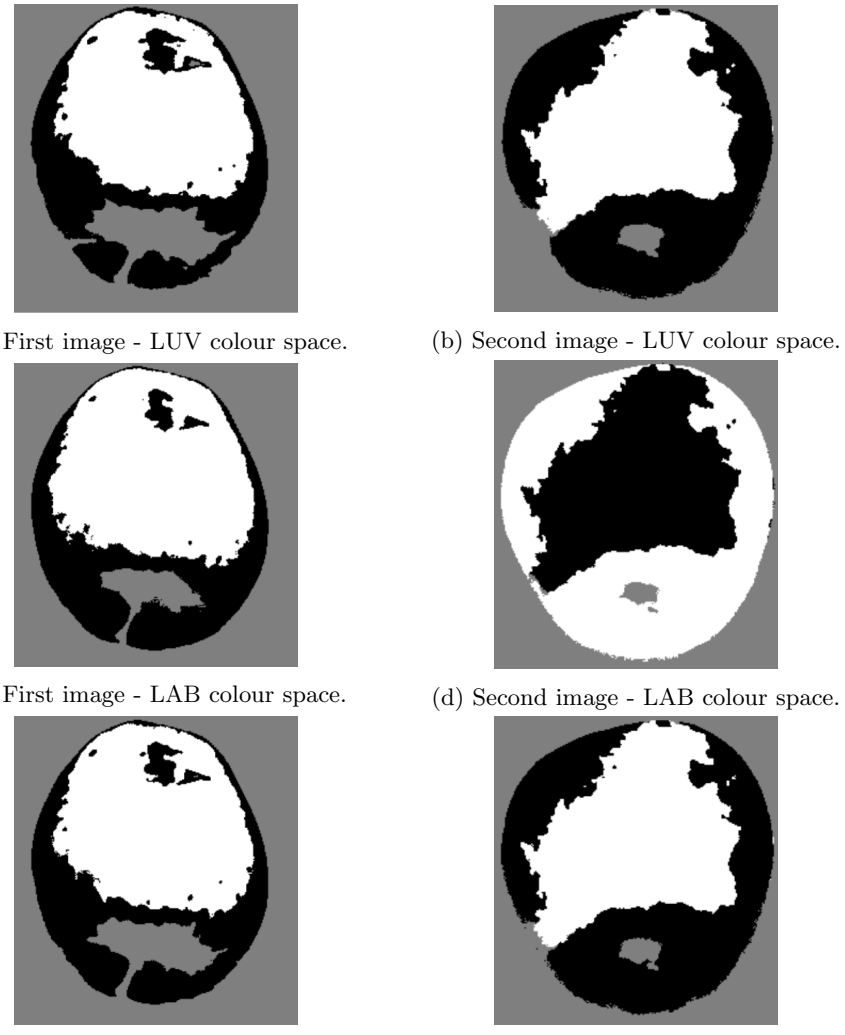
quality of the results is in line with the analysis of the colour spaces carried out in section 3.1.2.

This segmentation method could be considered suitable if the fruits that the system had to inspect were all of the type of the second image, as the russet region in this fruit has a colour which is fairly different that the one of the healthy part.

Unfortunately, the algorithm fails at clustering the first fruit because it appears to have a portion of the healthy part which is too close to one of the russet regions from a colour perspective resulting in them being mixed together, while the remaining healthy part of the fruit is collected in another cluster.

In conclusion, a different approach has to be considered in order to obtain reliable results for each kind of fruit.

3.2.2 Gaussian Mixture Quantization



(a) First image - LUV colour space.

(b) Second image - LUV colour space.

(c) First image - LAB colour space.

(d) Second image - LAB colour space.

(e) First image - YC_RC_B colour space.

(f) Second image - YC_RC_B colour space.

Figure 32: Gaussian Mixture quantization results on the LUV, LAB and YC_RC_B colour spaces.

The second approach is quantization by *Gaussian Mixture*. It follows a similar procedure to the one of *K-Means*. Given n components to find:

1. Initially, n starting guesses are chosen for the location and shape of the Gaussian distributions;
2. Until convergence occurs:

- For each point, the probability weight of being part of each cluster is found;
- For each cluster, an update on its location, normalization and shape based on all data points according to their probability weights is carried out.

Similarly to *K-Means*, the reason behind the choice of this method of clustering is that it is automatic. Furthermore, denser clusters of points are observed in the previously analysed 2D histograms of the images and these could possibly correspond to different Gaussian distributions.

As before, Gaussian Mixture is computed on the images while imposing to find 3 different Gaussian components: the background; the russet regions and the healthy part of the fruits.

The clustering method has been applied to relevant colour spaces while considering the most important channels of each of them.

Figure 32 shows again that the best results are obtained for the LAB colour space regarding the second fruit. In general, *Gaussian Mixture* yields good performances on the same fruit among all relevant colour spaces. The outcomes regarding the first fruit are still not satisfactory.

Since similar results as the K-Mean approach are obtained, this method is considered unsuitable for the task, as the russets of the first fruit cannot be identified.

3.3 Russet Extraction by Colour-Based Segmentation

The third approach uses the same method as the one proposed in the assignment: colour segmentation by *Mahalanobis distance* from a specific reference colour.

In particular, the characterization of the reference colour can be estimated by obtaining the *mean* (μ) and the *inverse covariance matrix* (Σ^{-1}) of a series of training samples that are selected from a *Region of Interest (ROI)* of the same colour as the reference one.

The training samples have been chosen as squared patches of the *ROI* and not as single pixels for robustness reasons. The resulting estimated *mean* and *covariance matrix* of the reference colour are computed as the sum of the mean and covariance matrices obtained by each sample divided by the number of samples.

Given a training sample S of a ROI with $|S|$ pixels in a given colour space with n channels $\{ch_1, ..., ch_n\}$, the mean μ_S of the reference colour according to the sample is computed as:

$$\mu_S = \begin{bmatrix} \mu_S(ch_1) \\ \vdots \\ \mu_S(ch_n) \end{bmatrix} = \frac{1}{|S|} \sum_{pixel} \sum_{p \in S} p$$

The inverse covariance matrix Σ_S of the same sample S in the given colour space is obtained as:

$$\Sigma_S = \begin{bmatrix} \sigma^2_{(ch_1,ch_1)} & \cdots & \sigma^2_{(ch_1,ch_n)} \\ \vdots & \ddots & \vdots \\ \sigma^2_{(ch_n,ch_1)} & \cdots & \sigma^2_{(ch_n,ch_n)} \end{bmatrix}$$

Where:

$$\sigma^2_{(ch_i,ch_j)} = \frac{1}{|S|} \sum_{pixel} \sum_{p \in S} (p_{ch_i} - \mu(ch_i)) \cdot (p_{ch_j} - \mu(ch_j))$$

Once estimated the final *mean* μ and *covariance matrix* Σ of the reference colour, the *Mahalanobis distance* (d_M) of a pixel p (seen as a vector of intensities for the n channels) from the reference colour can be computed as:

$$d_M(p) = \sqrt{(p - \mu)^T \Sigma^{-1} (p - \mu)}$$

Finally, the distance of each pixels from the reference colour is obtained and collected in a resulting image I. Just the pixels presenting a distance less or equal than a certain threshold t can be classified as part of the *ROI*, while the others can be discarded. The resulting binary mask is computed as follows:

$$\forall \text{ pixel } p \in I, \begin{cases} p \in ROI, & \text{if } p \leq t \\ p \notin ROI, & \text{otherwise} \end{cases}$$

It is important to notice that the calculation of Σ and μ are performed *offline*, hence the *russet detection system* would not need to compute the reference colour for every provided instance. For this reason, the time needed to perform this operation is not taken into account.

3.3.1 Mahalanobis Distance from the Russets

First of all, the method is applied while computing a reference colour that considers all the russet regions. In order to extract this colour, the *Regions of Interest* corresponding to the various russets have to be extracted.

The results of the *Gaussian Mixture* can be exploited to obtain the russet regions while avoiding a manual identification. Although, they are not correctly separated from other parts of the fruits, they can be simply identified through some image transformations such as *erosions* or changes of intensity of regions of pixels.

An example of the process of the extraction of a russet regions is presented in Figure 33, while the extracted russets themselves can be seen in Figure 34.

(a) Gaussian Mixture results in the LUV space of the first image. (b) Mask of the russet region. (c) Highlighted russet region in the fruit. (d) Extracted russet region.

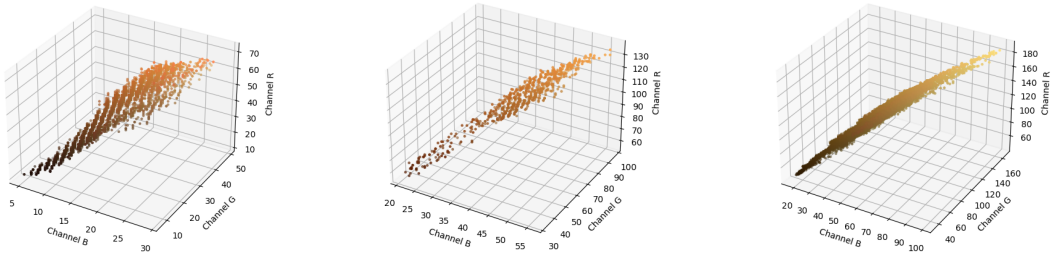
Figure 33: Process of the extraction of a russet region for sampling.

(a) First russet of the first image. (b) Second russet of the first image. (c) Russet of the second image.

Figure 34: Extracted russet regions.

The *BGR* colour distribution of the obtained russet regions is then illustrated in Figure 35. It is evident how the three areas are particularly different in terms of intensities, therefore computing a

reference colour that considers all of them for segmentation may yield bad results. Nonetheless, the process is still tested.



(a) Colour distribution of the first russet.

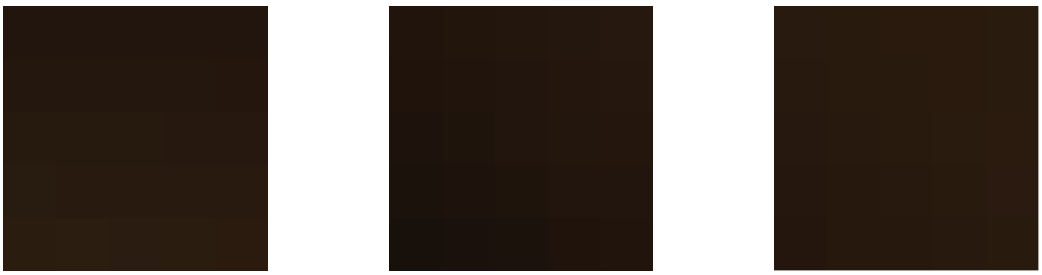
(b) Colour distribution of the second russet.

(c) Colour distribution of the third russet.

Figure 35: Colour distribution of pixels of the russet regions.

As a second step, a series of sample patches is extracted from the three russet *ROIs*. In order to obtain them the method `extract_patches_2d` of the *scikit-learn* library[6] has been used. All patches of dimension 5×5 are obtained from the images of each russet. Then, the ones containing background pixels are discarded. Finally, 200 patches are randomly chosen among the remaining ones.

An example of a sample patch from each ROI is shown in Figure 36.



(a) Sample of the first russet.

(b) Sample of the second russet.

(c) Sample of the third russet.

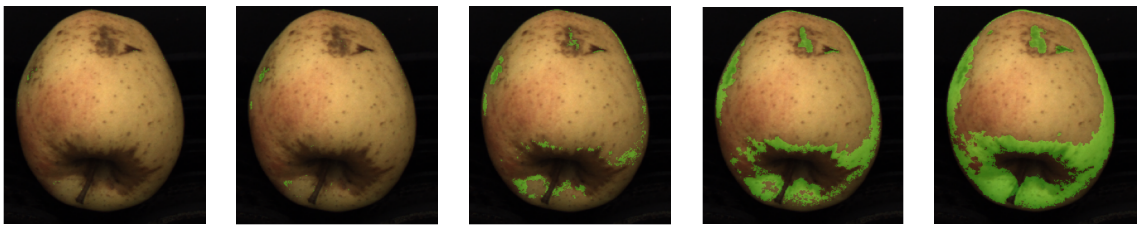
Figure 36: Example of samples from each russet.

Finally, the colour segmentation process occours. The mean μ and the inverse covariance matrix Σ^{-1} of the reference colour of the sample patches are obtained for each colour space considering all the 3 channels. Then the *Mahalanobis distance* is computed for each image and solely the pixels at a maximum distance according to threshold t are considered as pixels corresponding to russet regions.

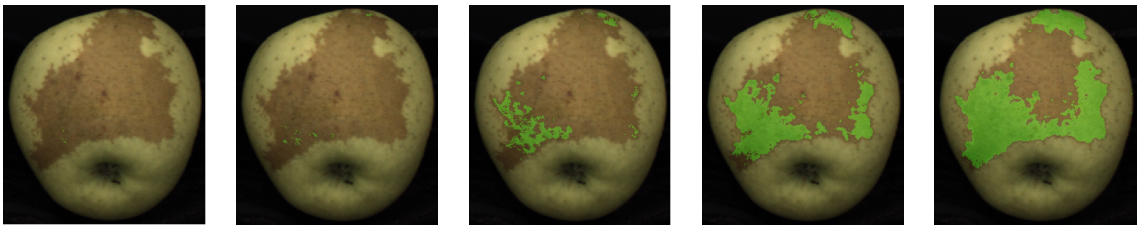
The process is repeated considering different thresholds t ($2, 3, 5, 7$ and 9). Similar results are obtained for each colour space:

- The pixels of the darker russet at the bottom of the first fruit are never recognized for any value of t.
- On the other hand, most pixels of the brighter small russet on the top of the fruits are correctly classified for thresholds 7 and 9. Unfortunately, a huge number of false positives in correspondence to the healthy part in the bottom of the fruit and around it are also obtained considering these thresholds.
- Regarding the second fruit, pixels of the russet start to be identified for t equal to 5 and above. No false positives are obtained, however, the number of false negatives is pretty high, as most pixels of the russet are not classified as such.

An example of the result of applying the segmentation in the *LAB* colour space with the different thresholds is shown in Figure 37.



(a) First figure (threshold 2). (b) First figure (threshold 3). (c) First figure (threshold 5). (d) First figure (threshold 7). (e) First figure (threshold 9).

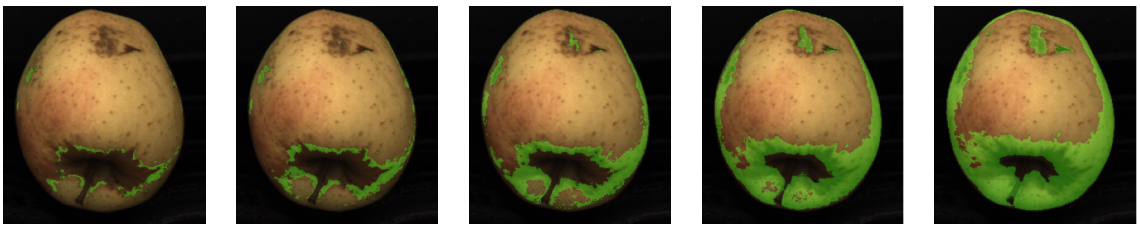


(f) Second figure (threshold 2). (g) Second figure (threshold 3). (h) Second figure (threshold 5). (i) Second figure (threshold 7). (j) Second figure (threshold 9).

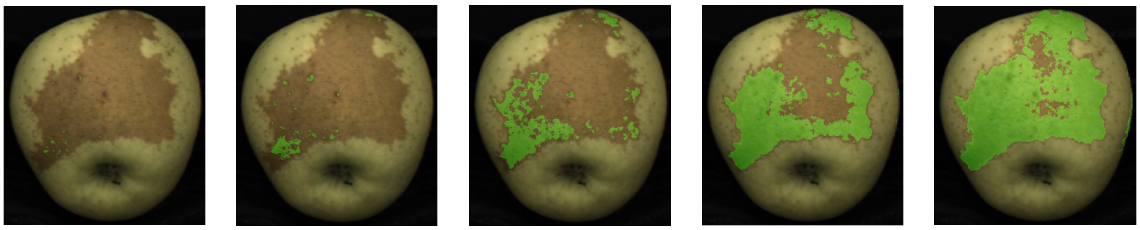
Figure 37: Segmentation by colour distance from the russet regions in the LAB colour space, considering different thresholds.

The process is repeated for each colour space and for different thresholds t $(2, 3, 5, 7$ and $9)$. This time, just the most important channels of each colour space are considered to compute the reference colour and the *Mahalanobis distance*. This time varied results are obtained among the different colour spaces. The best, although not satisfactory, ones are yielded by the LAB an YC_RC_B colour spaces.

As seen in Figure 38, most of the russet of the second fruit is identified with no false positives, while parts of both russets are highlighted in the first even though a high amount of false positives persist.



(a) First figure (threshold 2). (b) First figure (threshold 3). (c) First figure (threshold 5). (d) First figure (threshold 7). (e) First figure (threshold 9).



(f) Second figure (threshold 2). (g) Second figure (threshold 3). (h) Second figure (threshold 5). (i) Second figure (threshold 7). (j) Second figure (threshold 9).

Figure 38: Segmentation by colour distance from the russet regions in the LAB colour space using channels A and B and considering different thresholds.

It is evident that sampling a reference colour based on all the russets of the two fruits is not a good approach to identify them, as their distribution of colours is too different.

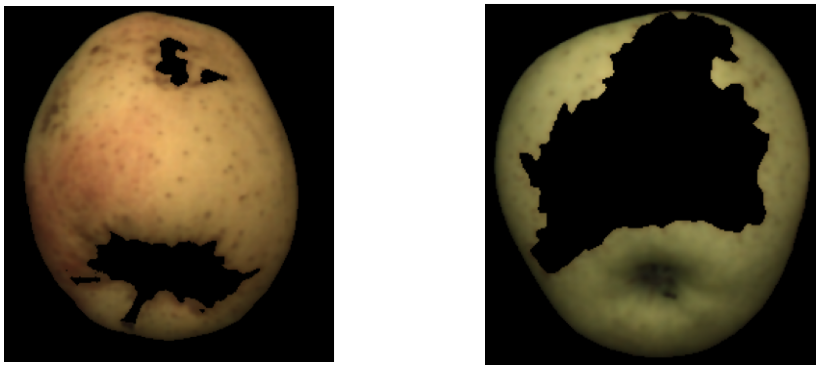
Furthermore, the experiments made it clear that the LAB colour space is the most suitable to identify similar coloured regions, therefore it is the only one considered in the next tests, along with its main channels (A and B).

3.3.2 Mahalanobis Distance from the Healthy Region

As an alternative approach, the segmentation is applied by considering a reference colour computed on all the healthy parts of the fruits.

This method relies on the fact that classifying pixels that correspond to the healthy part of the fruit is the inverse task of classifying pixels of the russet regions. By complementing the set of healthy fruit pixels and removing the background, then the pixels of the russet regions are obtained.

As illustrated in Figure 39, the healthy russet regions are obtained using a similar method as in section 3.3.1.

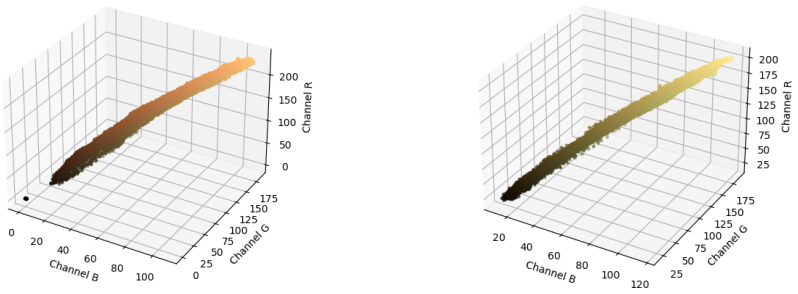


(a) Healthy region of the first image.

(b) Healthy region of the second image.

Figure 39: Extracted healthy fruit regions.

Once again, the BGR 3D colour distribution of the two healthy parts of the fruits are plotted in Figure 40. It looks like there is a greater similarity in the distribution of pixels than the ones of the russet regions.



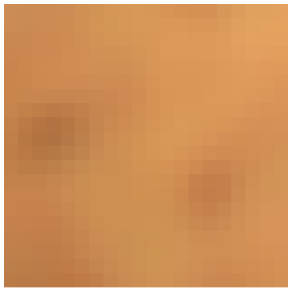
(a) Colour distribution of the healthy part of the first fruit.

(b) Colour distribution of the healthy part of the second fruit.

Figure 40: Colour distribution of pixels of the healthy part regions.

Symmetrically to the previous case, a series of sample patches is extracted from the two healthy fruit *ROIs*. In this case $1,000$ 20×20 patches per region are obtained and examples for each region are shown in Figure 41.

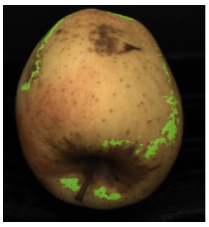
(a) Sample of the healthy region of the first fruit.



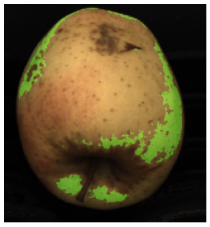
(b) Sample of the healthy region of the second fruit.

Figure 41: Example of samples from each healthy region.

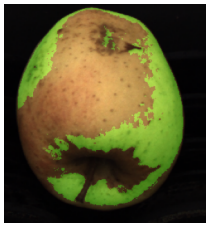
The segmentation process is repeated considering different thresholds t ($2, 3, 5, 7$ and 9) and the LAB colour space with channels A and B. The healthy part of the first fruit cannot be entirely identified with any value of t. Furthermore a lot of false positives corresponding to the russet regions are obtained for thresholds 7 and 9 as observed in Figure 42.



(a) First figure (threshold 2).

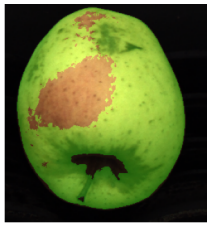


(b) First figure (threshold 3).

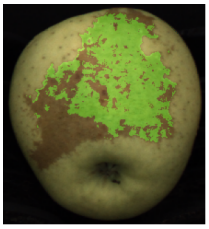


(c) First figure (threshold 5).

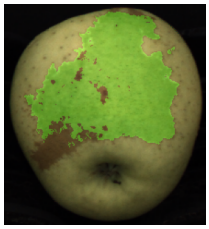
(d) First figure (threshold 7).



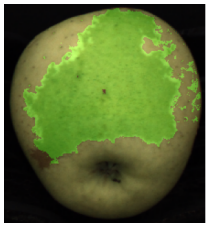
(e) First figure (threshold 9).



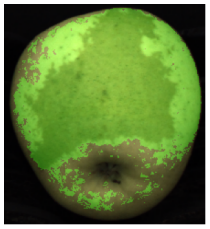
(f) Second figure (threshold 2).



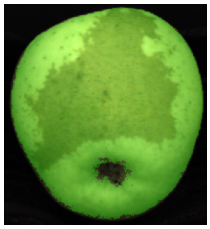
(g) Second figure (threshold 3).



(h) Second figure (threshold 5).



(i) Second figure (threshold 7).



(j) Second figure (threshold 9).

Figure 42: Segmentation by colour distance from the healthy region in the LAB colour space using channels A and B and considering different thresholds.

Interestingly, the reference colour seems to be more similar to the russet of the second fruit than its healthy region. This is pointed out by the fact that mostly pixels of the russet are classified as being closer to the reference colour for low values of the thresholds. By increasing t by 7 or 9, all the fruit is classified as being healthy, russet region included.

The obtained segmentations are not reliable at all and the results make it clear that the two fruits are composed of palettes of colour that are too distinct from each other. Using a single reference colour to segment the russets from the healthy parts of the fruits is not possible.

3.3.3 Russet Detection Based on Fruit Classification

As noticed from previous experiments, it is evident how the colour scheme of the fruits is too different. The same can be said for the russet regions. The final implemented segmentation approach relies on all these observations.

In order to correctly spot the russets of a fruit it is firstly necessary to classify whether the fruit is of the first type (f_1, or the first image) or of the second one (f_2, or the second image). Once the fruit has been classified, a segmentation of the russets is computed.

In detail the process consists in:

1. Obtaining a reference colour of the healthy regions of both fruit described by their means and inverse covariance matrices:
   - c_{h1}, described by μ_{h1} and Σ_{h1}^{-1} for the first fruit;
   - c_{h2}, described by μ_{h2} and Σ_{h2}^{-1} for the second fruit.
2. Obtaining a reference colour of the two russets of the first fruit described by their means and inverse covariance matrices:
   - c_{r1}, described by μ_{r1} and Σ_{r1}^{-1} for the first russet;
   - c_{r2}, described by μ_{r2} and Σ_{r2}^{-1} for the second russet.
3. Obtaining a reference colour of the russet of the second fruit described by its mean and inverse covariance matrix:
   - c_{r3}, described by μ_{r3} and Σ_{r3}^{-1}.
4. Classifying whether the given instance is of the class of the first or second fruit:
   - The instance is segmented in two binary masks according to the first reference colour of the healthy fruit c_{h1} and with the second reference colour of the healthy fruit c_{h2} using a common threshold t;
   - If the number of pixels in the mask obtained by the segmentation with respect to c_{h1} is greater than the number of pixels in the mask obtained by the segmentation with respect to c_{h2} the fruit is classified as a fruit of the first type (f_1). Otherwise the fruit is classified as a fruit of the second type (f_2).
5. Segmenting the russets from the fruit:
   - If the fruit is classified as f_1, the russets are segmented by keeping just the pixels at a *Mahalanobis distance* less than a certain threshold with respect to c_{h1} and c_{h2};
   - If the fruit is classified as f_2, the russets are segmented by keeping just the pixels at a *Mahalanobis distance* less than a certain threshold with respect to c_{h3};

Note: *This process assumes that just two varieties of fruits are present in the system: the one of the first image and the one of the second image. A further assumption is that just the kind of russets of a specific fruit can be present in other instances of the same fruit (e.g.: A russet with the same colour as the one in the second fruit cannot appear in fruits of the same class as the ones of the first type).*

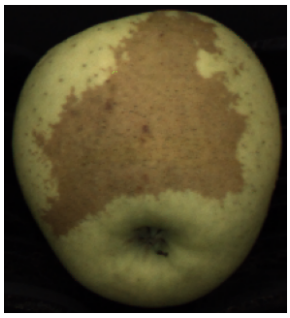
A series of sample patches is extracted from different *ROIs* in order to compute the *mean* and *inverse covariance matrices* of c_{h1}; c_{h2}; c_{r1}; c_{r2} and c_{r3}. In particular:

- $1,000$ patches of size 20×20 are obtained from ROIs related to c_{h1};
- $1,000$ patches of size 20×20 are obtained from ROIs related to c_{h2};
- 500 patches of size 5×5 are obtained from ROIs related to c_{r1};
- 500 patches of size 5×5 are obtained from ROIs related to c_{r2};
- $1,000$ patches of size 20×20 are obtained from ROIs related to c_{r3}.

The reason why less samples of smaller sizes are obtained for c_{r1} and c_{r2} is that the relative ROIs are particularly small compared to the others.

Regarding the *Mahalanobis distance* computed according to reference colour c_{h1}, it is observed in Figure 43 that a threshold value of 3 is enough to correctly distinguish the two healthy parts of the fruits and classify them. A large amount of pixels are obtained and correctly classified as the healthy part of the first fruit in the first image, while almost no pixels at all are obtained in the second image.

(a) Pixels assigned to reference colour c_{h1} in the first fruit.

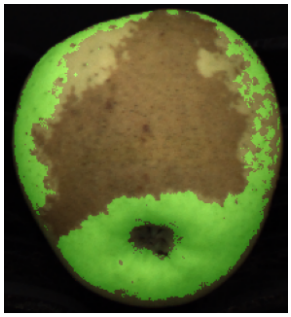


(b) Pixels assigned to reference colour c_{h1} in the second fruit.

Figure 43: Classification of the fruits showing the pixels with Mahalanobis distance less or equal than 3 from the healthy part of the first fruit (reference colour c_{h1}).

Symmetrically, the *Mahalanobis distance* calculated according to reference colour c_{h2} shows in Figure 44 that once again for a threshold with value 3 the two healthy parts of the fruits are distinguished and they can be classified.

(a) Pixels assigned to reference colour c_{h2} in the first fruit.



(b) Pixels assigned to reference colour c_{h2} in the second fruit.

Figure 44: Classification of the fruits showing the pixels with Mahalanobis distance less or equal than 3 from the healthy part of the second fruit (reference colour c_{h2}).

It looks like a threshold of 3 is enough to classify a fruit as being of the first or second kind (f_1 or f_2) according to its healthy region. If a greater amount of pixels at *Mahalanobis distance* less or equal than 3 with respect to c_{h1} are found than the ones at a same distance with respect to c_{h2}, then the fruit is classified as f_1. Otherwise it is classified as f_2.

Now that the fruits can be discriminated, it is easy to segment the russet regions as seen in Figure 45. With a threshold of value 7, all pixels of the darker russet region at the bottom of the first fruit (with reference colour c_{r1}) are obtained with a few false positives at the border of the fruit. The presence of these false positives is due to the fact that pixels at the border of the fruit are in a particularly shadowy area.

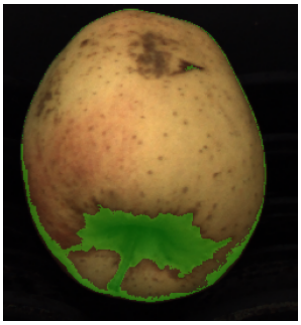


Figure 45: First russet extraction from the first fruit by using Mahalanobis distance with threshold 7 from c_{r1}.

Regarding the small russet region at the top of the first fruit (with reference colour c_{r2}), most of its pixels can be obtained with a threshold equal to 1.5, with just a few false positives near the darker russet region at the bottom as seen in Figure 46.

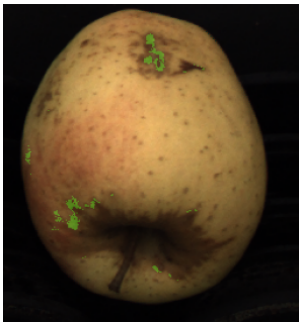


Figure 46: Second russet extraction from the first fruit by using Mahalanobis distance with threshold 1.5 from c_{r2}.

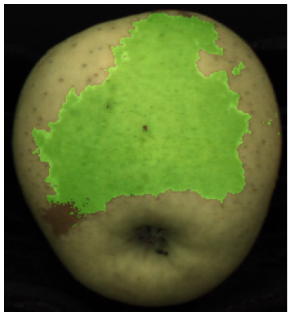
The two detected russet regions are joined together in a singular russet mask. By eroding the mask that separates the fruit from the background computed in section 3.1.1 and performing a bitwise *AND* operation between it and the previously computed russet mask, the false positives at the border of the fruit are removed.

(a) Raw russet mask for the first fruit. (b) Cleaned-up russet mask for the first fruit.

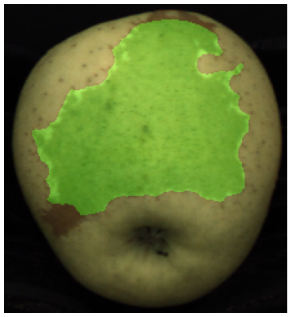
Figure 47: Russet masking process in the first fruit.

Finally, the russet mask is cleaned-up by using a *Median Blur* filter of dimension 5×5 over it, in order to remove small noisy points that can be considered similar to impulse noise. A final *closing* operation is applied to join close small possible gaps in the russet mask. The results are illustrated in Figure 47.

Regarding the russet of the second fruit (with reference colour c_{r3}), with a threshold of value 7, almost all of its pixels are obtained with a few false positives. The same kind of cleaning operations as the russet mask of the first fruit are performed and a cleaned-up mask is returned. Figure 48 illustrates these operations.



(a) Raw russet mask for the second fruit.



(b) Cleaned-up russet mask for the second fruit.

Figure 48: Russet masking process in the second fruit.

3.4 Results

An algorithm performing the russet detection step with the *colour segmentation by fruit classification and russet distance detection* method is built and it can be executed via command line or by the method `detect_russet`.

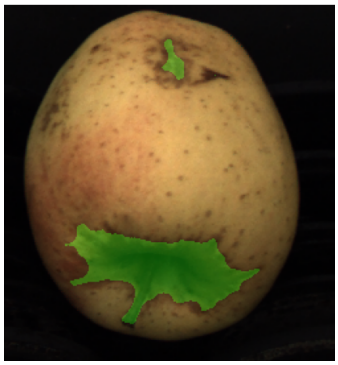
The algorithm:

1. Converts the colour image to grayscale and filters it by *Median Blur* with a kernel 5×5;
2. Filters the colour image by a *Bilateral filter* using kernel of dimension (7×7) and spatial standard deviation (σ_s) and intensity standard deviation (σ_r) equal to 75;
3. Applies *Tweaked Otsu's Algorithm* on the grayscale image to obtain the segmented mask of the fruit with a given *tweak factor*;
4. Applies the mask over the filtered colour image;
5. Turns the masked colour image to the LAB colour space;
6. Obtains the fruit class by a given list of reference colours of the healthy parts of the fruits of the system (described by their mean and inverse covariance matrix);
7. Gets a russet mask by applying colour segmentation with respect to the reference colour of the possible russets of the class of the detected fruit;
8. Cleans-up the russet masks;
9. Shows the russet regions, counts their occurrences and highlights their areas through ellipses.

Once again, it can be observed in Figure 49 that the result on the given images are pretty satisfactory using a *tweak factor* $= 0.4$ and the previously identified thresholds for colour segmentation.

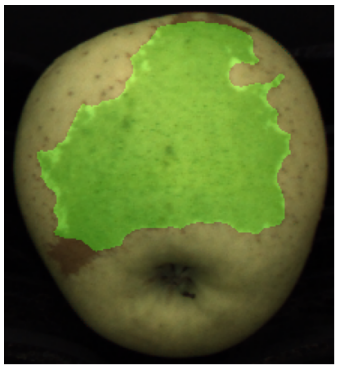
The *mean* and *inverse covariance matrices* of the reference colours of the healthy regions of the fruits and the russet regions are pre-computed *offline* and loaded from the memory.

Finally, the time for detecting the russet is computed by applying the `detect_defects` function on each image $1,000$ times, for a total of $2,000$ executions. The algorithm is fast, as it can detect the defects of an image with a mean time of ≈ 0.01 seconds.



(a) First image russets.

(b) First image russets areas.



(c) Second image russets.

(d) Second image russets areas.

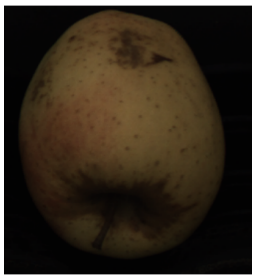
Figure 49: Results of the russet detection system implemented by the `detect_russet` algorithm.

3.4.1 Portability

Unlike the *defect detection system* presented in the previous notebook, the *russet detection system* is not designed to be automatically ported to other industrial settings by simply changing some of its parameters.

Small changes of illumination can in fact affect the colour distribution of the images. Therefore the means and inverse covariance matrices of the reference colours of the *ROIs* must be manually recomputed with respect to the given environment.

(a) Colour image.



(b) Darker colour image.

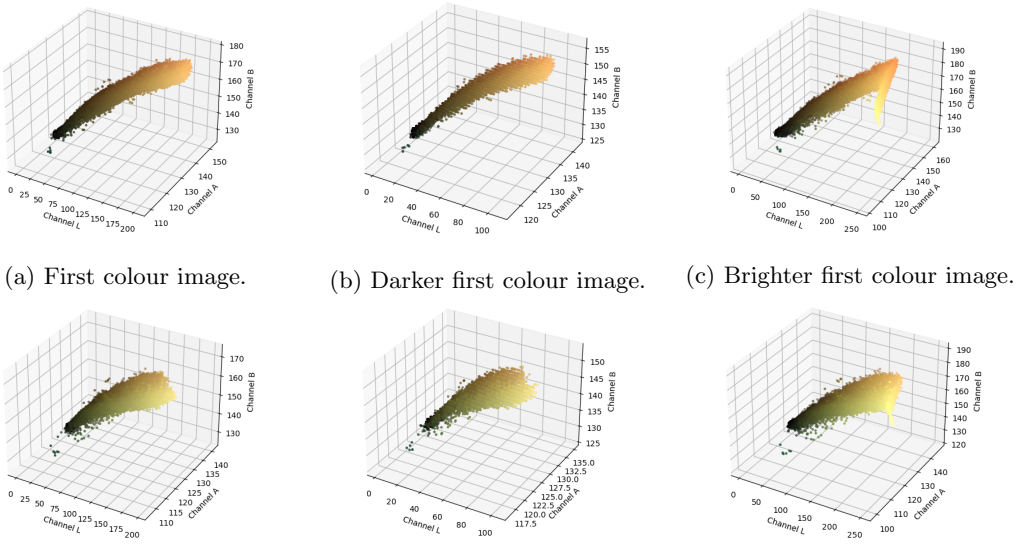
(c) Brighter colour image.

Figure 50: Example of the darker and brighter computed versions of the first image.

In order to prove so, a version of the colour images with a darker and brighter contrast intensity are computed from the initial images and an example is shown in Figure 50.

A comparison of the colour space of each image along with their darker and lighter counterparts are illustrated in Figure 51. It can be observed that the colour distributions are indeed different, especially

for what concerns the brighter version of the images, where an abnormal spike is present.



(a) First colour image. (b) Darker first colour image. (c) Brighter first colour image.

(d) Second colour image. (e) Darker second colour image. (f) Brighter second colour image.

Figure 51: Colour distribution of pixels of the fruits in the BGR colour space with darker and brighter light intensities.

3.4.2 Adding a New Fruit Class

In order for the system to robustly detect russets, it is designed to work on a specific set of predetermined species of fruits. Nonetheless, adapting the system by introducing a new kind of fruit with its own reference colours of its healthy part and its russet regions is fairly easy.

To prove so, a new kind of fruit with a new kind of russet region is introduced and shown in Figure 52.

Figure 52: Extra fruit added in the system.

In order to correctly classify the russet of fruits of this new type it is sufficient to:

1. Compute the reference colour of its healthy region;
2. Compute the reference colour of its russet region;
3. Consider this new fruit during the fruit classification step.

The results of the detection of the russet for this new kind of fruit are expressed in Figure 53.

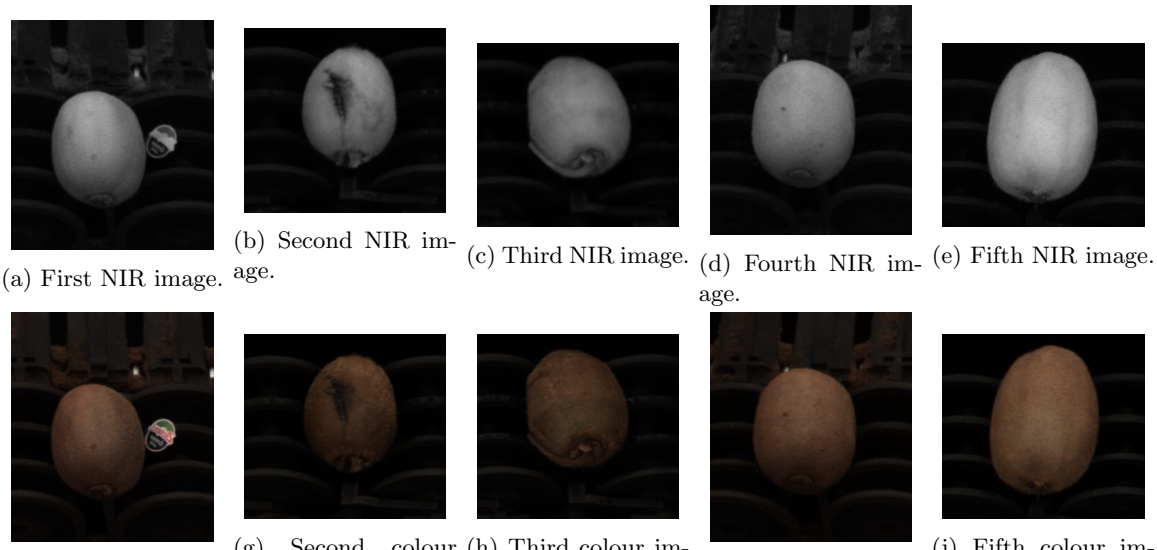
(a) Extra image russets. (b) Extra image russets areas.

Figure 53: Results of the russet detection system implemented by the `detect_russet` algorithm on the newly introduced fruit.

4 Final Challenge: Kiwi Inspection

The final challenge considers the colour and Near Infra-Red images of five different kiwis as seen in Figure 54.

It is required to localize eventual defects on the fruits, while applying special care in removing the dirt on the conveyor as well as the sticker in the first image.



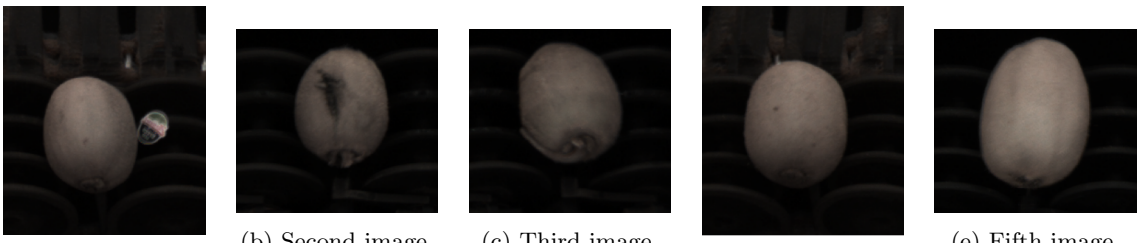
(a) First NIR image. (b) Second NIR image. (c) Third NIR image. (d) Fourth NIR image. (e) Fifth NIR image.

(f) First colour image. (g) Second colour image. (h) Third colour image. (i) Fourth colour image. (j) Fifth colour image.

Figure 54: Source images of the final challenge.

4.1 Image Analysis

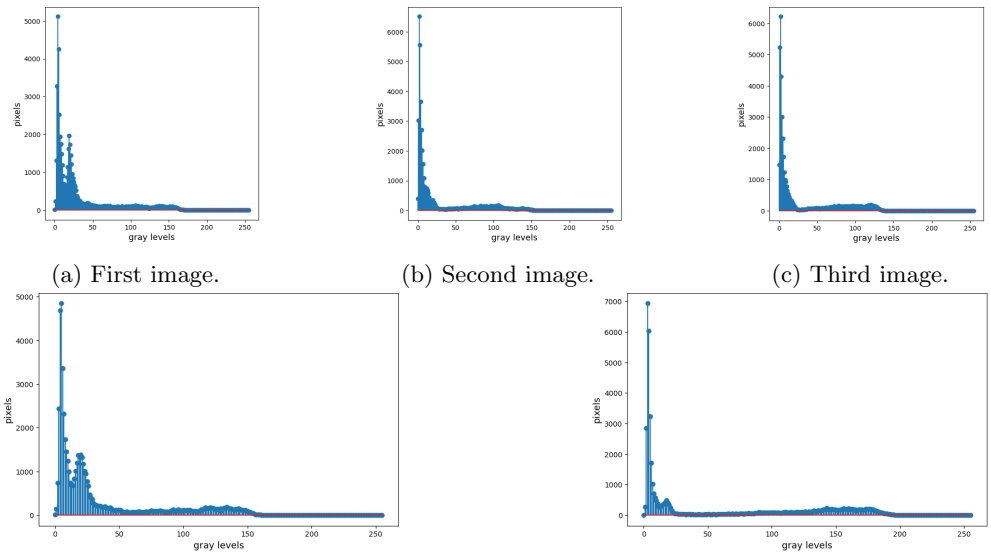
As in the previous tasks, as a first image analysis step the parallax between the colour and NIR images is observed. Looking at Figure 55 they seem to overlap almost perfectly, therefore the NIR version of the images can be used to obtain the binary masks that segment the fruits from the background, which can be also applied over their colour versions.



(a) First image. (b) Second image. (c) Third image. (d) Fourth image. (e) Fifth image.

Figure 55: Parallax between the colour and NIR images.

The *gray-level histograms* of the NIR images are then computed and illustrated in Figure 56.



(a) First image. (b) Second image. (c) Third image.

(d) Fourth image. (e) Fifth image.

Figure 56: Gray-level histograms of the NIR images.

The plots make it clear that the histograms are not *bimodal*. However, in the range of levels $[0, 50]$ a concentration of pixels is present, which would most probably account for background pixels. In the first and fourth images the concentration of pixels in this range does not define a clear peak, probably due to the presence of the noisy conveyors dirt in the background.

4.2 Pre-processing

As mentioned in section 2.2 of the first task, in order to guarantee better performances in the binarization procedure a pre-processing operation has to be applied to the NIR images. It is important to notice that the images of this task do not present an uniform background due to the presence of particularly noisy artefacts corresponding to the conveyor belts dirt in the background.

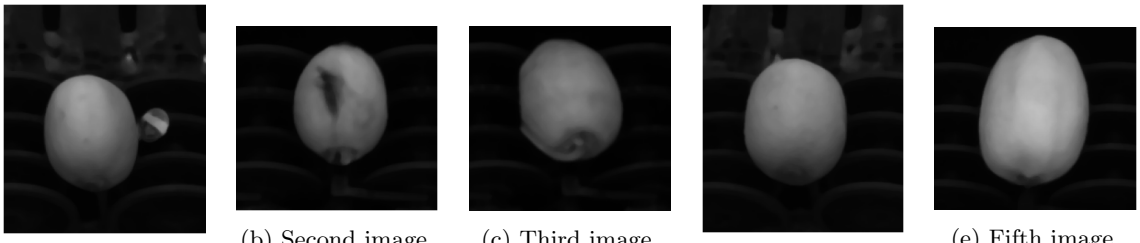
In particular, two different kind of pre-processing techniques have been taken into account and compared in terms of quality and speed:

- *Median Blur filtering*;
- *Non-Local Means filtering*.

4.2.1 Median Blur Filtering

Firstly, a *Median blur* non-linear filter using a kernel of dimension 7×7 has been applied. As specified in the previous tasks, this filter guarantees that the noisy details of the images are blurred and the edges are preserved.

The results are in line with the observation above, as the images are denoised while the edges are preserved and the artifacts on the background are partially removed as seen in Figure 57.

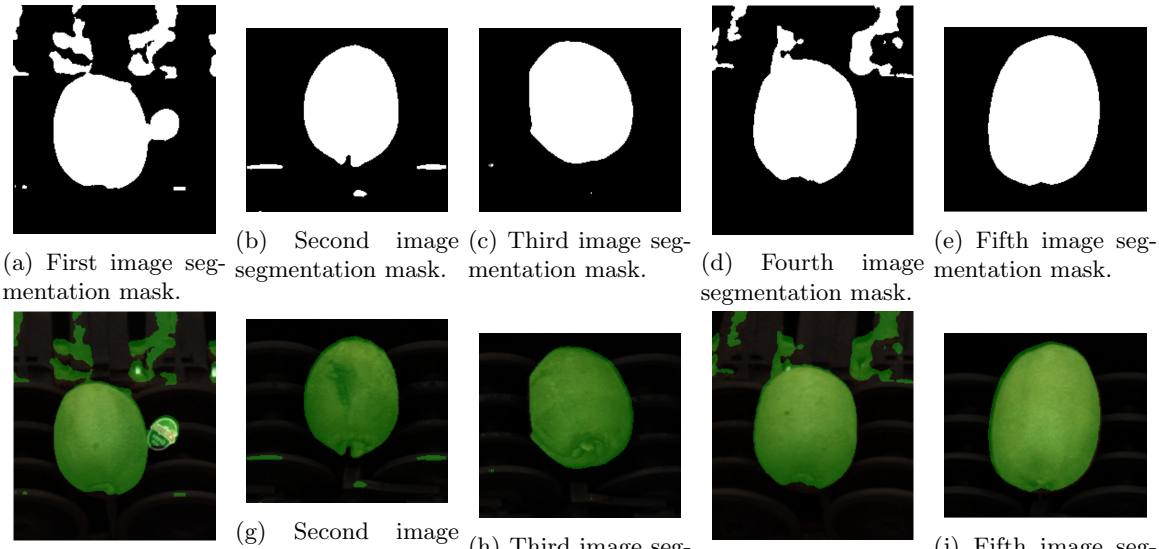


(a) First image. (b) Second image. (c) Third image. (d) Fourth image. (e) Fifth image.

Figure 57: NIR images filtered by Median Blur.

The same technique described in section 2.3.3 has been considered in order to segment the fruits from the background, namely *Tweaked Otsu's Algorithm* followed by *flood-filling*. For the given problem, a *tweak factor* of value 0.4 has been chosen.

As observed in figure 58, the results obtained through this approach are satisfactory as in each image the whole fruit is fully segmented from the background in its entirety. However, in the first and fourth images the segmentation is not fully obtained. In fact, a large amount of noisy artifacts of the dirt in the background are included in the *foreground region* and some of these are attached to the fruits themselves.



(a) First image segmentation mask. (b) Second image segmentation mask. (c) Third image segmentation mask. (d) Fourth image segmentation mask. (e) Fifth image segmentation mask.

(f) First image segmentation outline. (g) Second image segmentation outline. (h) Third image segmentation outline. (i) Fourth image segmentation outline. (j) Fifth image segmentation outline.

Figure 58: Tweaked Otsu's results on the NIR images filtered by Median Blur.

4.2.2 Non-Local Means Filtering

As the previous pre-processing method (*Median blur*) presents some problems in segmenting the fruits completely from the background, a *Non-local means* denoising filter is considered and applied to the

images.

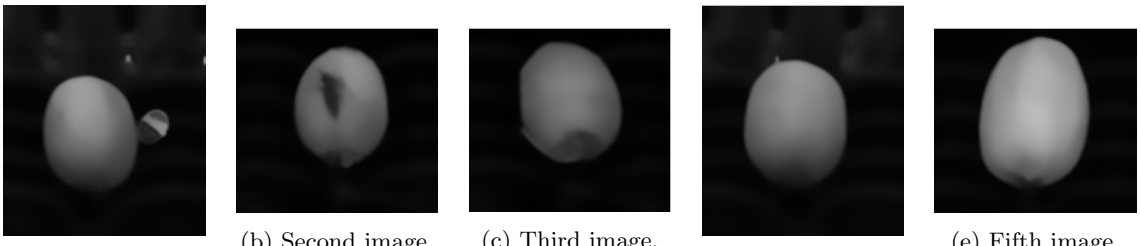
Non-local means filtering is an advanced non-linear filtering method that preserves the edges, while removing noise from the original image. It processes each pixel in the images while considering all the other pixels, which are weighted by similarity with respect to it according to their neighbour region.

Given an image I, the output of a pixel p, $O(p)$ is given by:

$$O(p) = \sum_{\text{pixel } q \in I} w(p,q) \cdot I(q)$$

Where $w(p,q)$ is the weight of pixel p with respect to q according to their similarity in their neighbour region.

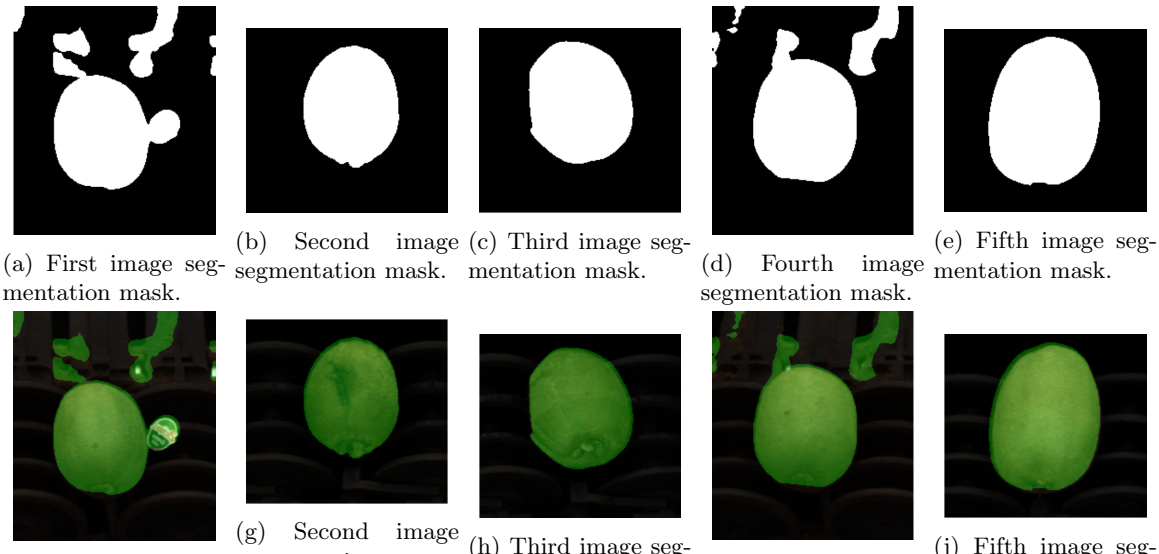
The usage of the filter with a high strength (parameter `h` of the function `fastNlMeansDenoising` of *OpenCV*) blurs a great amount of background noise, in spite of also removing details in the image as seen in Figure 59.



(a) First image. (b) Second image. (c) Third image. (d) Fourth image. (e) Fifth image.

Figure 59: NIR images filtered by a Non-Local Means filter.

The fruits are segmented once again by *Tweaked Otsu's Algorithm* with a *tweak factor* of 0.4.



(a) First image segmentation mask. (b) Second image segmentation mask. (c) Third image segmentation mask. (d) Fourth image segmentation mask. (e) Fifth image segmentation mask.

(f) First image segmentation outline. (g) Second image segmentation outline. (h) Third image segmentation outline. (i) Fourth image segmentation outline. (j) Fifth image segmentation outline.

Figure 60: Tweaked Otsu's results on the NIR images filtered by a Non-Local Means filter.

Judging by Figure 60, the segmentation results obtained through this pre-processing technique are better as for each image the whole fruit is almost fully separated from the background and the conveyor dirt is generally less marked. Unfortunately, patches of background noise are still connected to

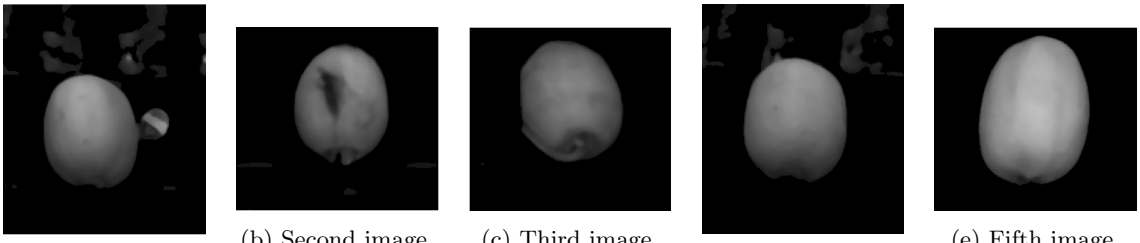
the fruit region in the first and fourth images.

However, the difference in time taken to pre-process the images by the two methods is quietly large. *Non-local Means filter* is almost 100 times slower than *Median Blur* as it takes ≈ 0.1 seconds to process an image with respect of the ≈ 0.001 seconds that are spent by the latter.

Due to its severe slowness *Non-local Means filter* cannot be deployed in an *online* system as the one considered in the task. Furthermore, it does not manage to completely remove the conveyors background noise, so it is discarded in favour of a *Median Blur* pre-processing.

4.3 Segmentation

The masks obtained through binarization with a *Median Blur* pre-processing and *Tweaked Otsu's Algorithm* are applied over the pre-processed NIR images and the results are observed in Figure 61.



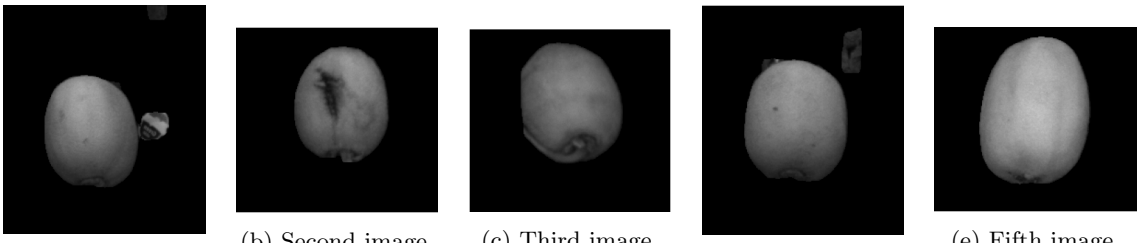
(a) First image. (b) Second image. (c) Third image. (d) Fourth image. (e) Fifth image.

Figure 61: Segmented pre-processed NIR images.

As previously stated, the obtained segmentation still present some artefacts in the foreground region of the masks that should be detached from the fruits and removed.

4.3.1 Artefacts Elimination

The background artefacts connected to the fruit all present an elongated shape that protrudes from their top region vertically. An *opening* on the segmentation masks using a structuring element that consider this characteristics would be enough to disconnect them from the fruit.



(a) First image. (b) Second image. (c) Third image. (d) Fourth image. (e) Fifth image.

Figure 62: Results of the opening operation to detach the noisy elements from the fruits.

However, new instances provided to the system could possibly present conveyor dirt artefacts that protrude horizontally from the sides of the fruit and for robustness reasons two different structuring elements are considered for two different *opening* operations on the masks:

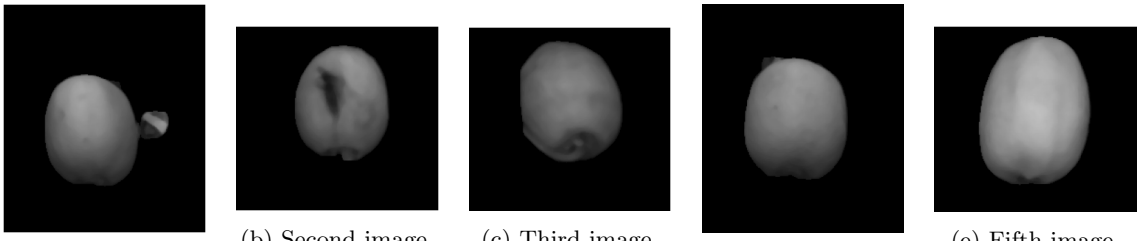
1. An ellipse with shape 20×5, used to remove possible background noise protruding horizontally from the fruit;

2. An ellipse with shape 5×20, used to remove possible background noise protruding vertically from the fruit.

This process guarantees that the background elements are detached from the fruit in the newly obtained masks as seen in Figure 62.

Afterwards, the detached noise is completely removed from the foreground by computing the contours of the blobs in the binary segmentation mask and by filling the pixels inside the largest one, that corresponds to the contour of the fruit itself.

The results are illustrated in Figure 63. It can be observed that all the fruits are correctly outlined without the presence of noisy elements with the exception of the first image, which still presents an unwanted sticker.



(a) First image. (b) Second image. (c) Third image. (d) Fourth image. (e) Fifth image.

Figure 63: Clean-up of the detached noisy defects from the segmented images.

4.3.2 Sticker Elimination

In order to remove the sticker in the first image, a segmentation of the fruit according to the *Mahalanobis distance* from a reference colour of the skin of the kiwis is performed in a similar fashion to the second task in section 3.3. The reason behind such choice is related to the fact that the sticker intuitively presents colours which are largely different than the ones of the kiwis. Therefore, it would have a large *Mahalanobis distance* from the fruit itself and could be easily identified and removed with the proposed method.

Transformation and intensity invariant *object detection* techniques were originally considered in order to match a template of the sticker in the images, but lately discarded as the object detection is based on a single sticker template. If different stickers are introduced in the system they would not be found by the algorithm.

On the other hand, a colour segmentation approach would manage to separate even different stickers than the one provided, as the *Mahalanobis distance* is computed with respect to a reference colour of the kiwis and not a reference colour of a particular sticker.

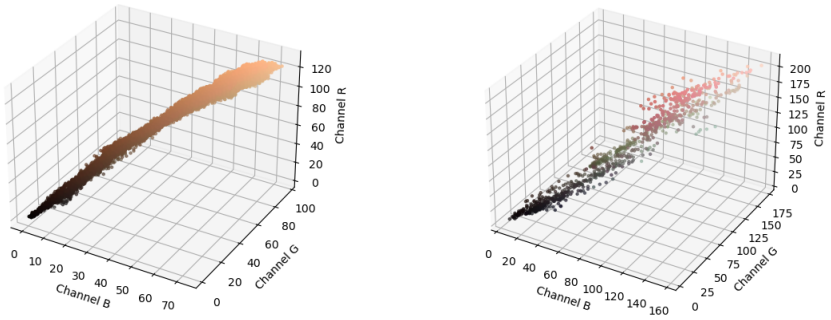
(a) First image. (b) Second image. (c) Third image. (d) Fourth image. (e) Fifth image.

Figure 64: Colour images processed by a Bilateral filter and masked from the background.

The colour images are initially pre-processed through a *Bilateral filter* using kernel of dimension 7×7 and spatial standard deviation (σ_s) and intensity standard deviation (σ_r) equal to 75 for the same reasons specified in the second task section 3.1.1. They are then masked from the background through

the masks obtained in the previous section. The results are observed in figure 64.

A three-dimensional plot of the colour distribution in the *BGR* colour space of the pixels of the kiwis and the sticker is presented in Figure 65. It confirm the intuition that the two colour distributions are pretty different.



(a) Kiwi colour distribution.

(b) Sticker colour distribution.

Figure 65: Colour distribution of pixels in the BGR colour space of the kiwi and the sticker.

In order to compute a colour reference of the kiwi, a series of samples are extracted from the fruits regions in the third and last colour images, as they present the best separation of the fruits from the background. In particular, 200 samples of size 20×20 are extracted from both kiwis. An example of a sample patch from each kiwi is shown in Figure 66.

(a) Sample of the kiwi in the third image.

(b) Sample of the kiwi in the fifth image.

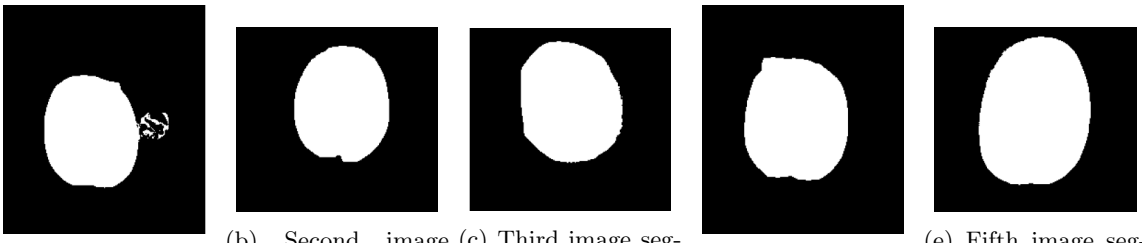
Figure 66: Example of samples of the kiwis.

The actual colour segmentation by considering the *Mahalanobis distance* from the reference colour is finally performed. The *LAB* colour space with channels A and B is used as it yielded the best results in the previous task.

The colour reference is estimated through the *mean* (μ) and the *inverse covariance matrix* (Σ^{-1}) computed on the provided samples as in the previous task. It is important to remark that that the calculation of Σ^{-1} and μ are performed *offline*, hence the time needed to carry out this operation is not taken into account.

Once the distance of each pixels from the reference colour is computed and collected in a resulting image, just the ones with a distance less or equal than a certain threshold t can be classified as part of the kiwi, while the others are discarded. As seen in Figure 67, with $t = 10$, all of the pixels of the fruits are obtained for all images with a few false positives that correspond to the sticker region in the first image. A binary mask segmenting the fruit from the background and the sticker is thus obtained

for each image.



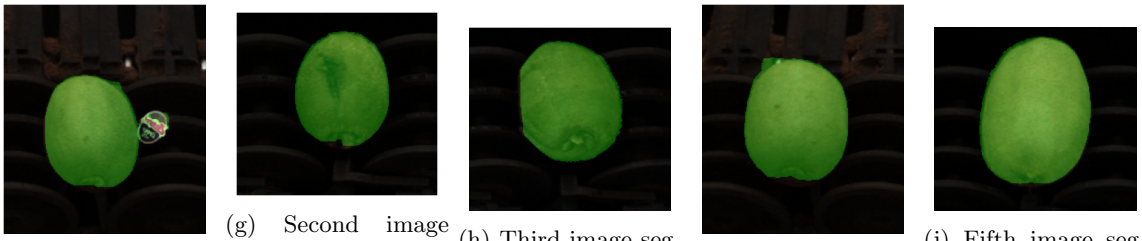
(a) First image segmentation mask.

(b) Second image segmentation mask.

(c) Third image segmentation mask.

(d) Fourth image segmentation mask.

(e) Fifth image segmentation mask.



(f) First image segmentation outline.

(g) Second image segmentation outline.

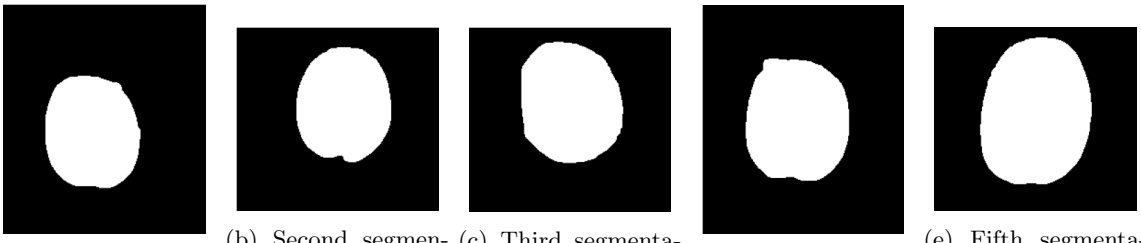
(h) Third image segmentation outline.

(i) Fourth image segmentation outline.

(j) Fifth image segmentation outline.

Figure 67: Segmentation by colour distance from the kiwi in the LAB colour space using channels A and B and considering threshold $t = 10$.

An *opening* is performed in order to clean the masks from the remaining sticker artefacts. the results are observed in Figure 68.



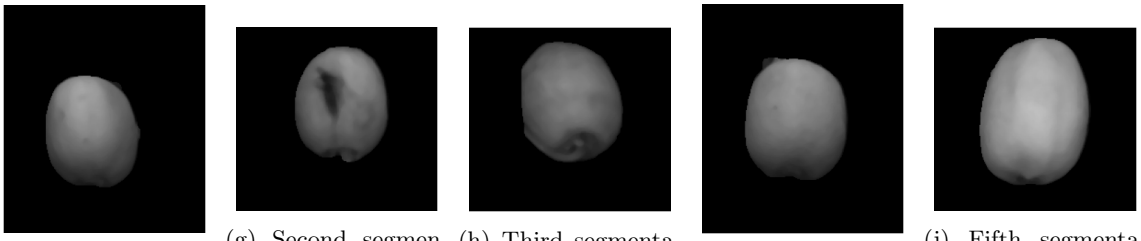
(a) First segmentation mask.

(b) Second segmentation mask.

(c) Third segmentation mask.

(d) Fourth segmentation mask.

(e) Fifth segmentation mask.



(f) First segmentation result.

(g) Second segmentation result.

(h) Third segmentation result.

(i) Fourth segmentation result.

(j) Fifth segmentation result.

Figure 68: Final segmentation results.

4.4 Defects Identification

In order to extract the edges, *Canny's method* is applied to the filtered and masked Near Infra-Red images as in the first task section 2.4.

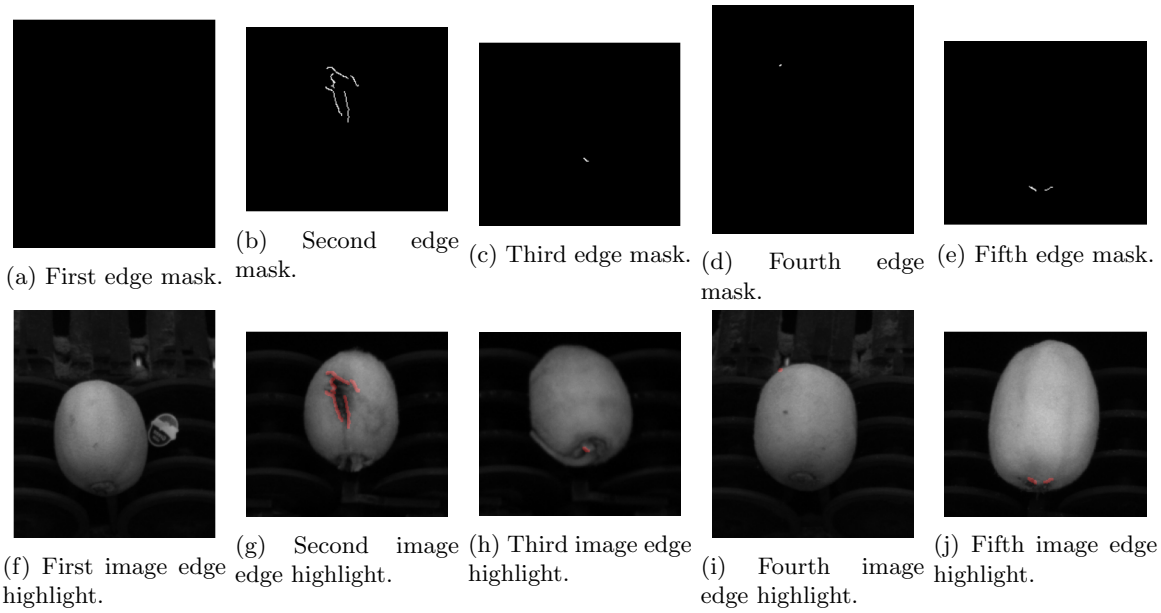
An initial *Gaussian Smoothing* is, once again, manually applied with a value of $\sigma = 1$ and a kernel size computed by a rule-of-thumb:

$$k = \lceil 3 \cdot \sigma \rceil$$

$$\text{kernel size} = (2 \cdot k + 1 \times 2 \cdot k + 1)$$

Regarding the *hysteresis* thresholds, the best results have been obtained with $t_h = 85$ and $t_l = 50$.

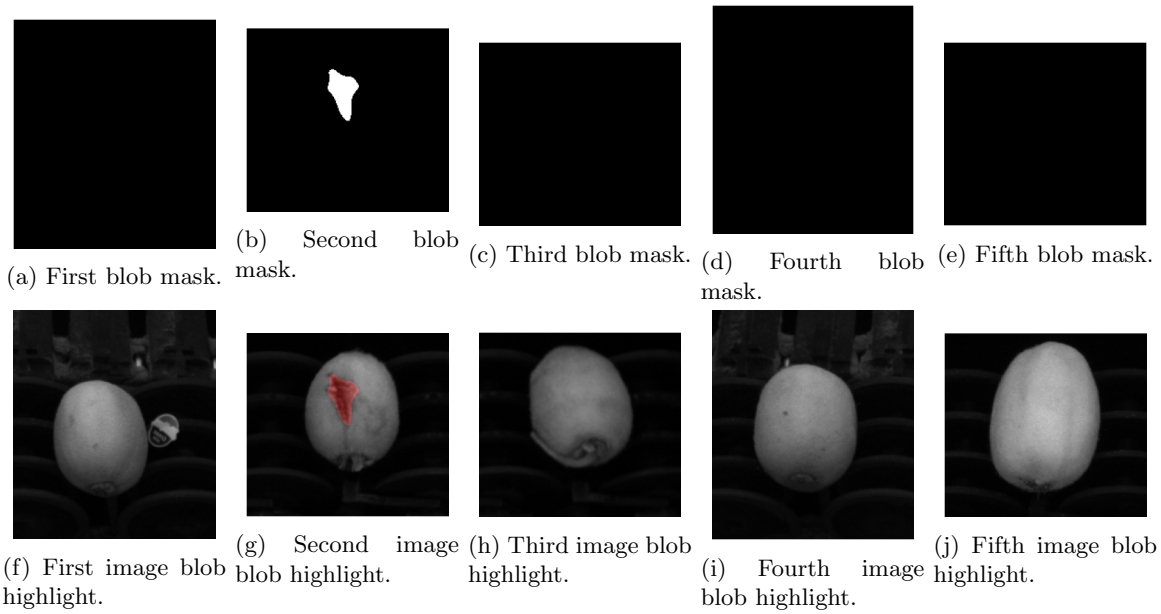
To remove the edges along the border of the fruit the binary masks that segment the fruits from the background are eroded by a rectangular *structuring element* of size 15×15 and then applied over the edge mask as in the first task. Thanks to this operation, solely the edges of the defects, along with some noisy artifacts, are preserved. Figure 69 illustrates the obtained edge masks for each fruit.



(a) First edge mask. (b) Second edge mask. (c) Third edge mask. (d) Fourth edge mask. (e) Fifth edge mask.

(f) First image edge highlight. (g) Second image edge highlight. (h) Third image edge highlight. (i) Fourth image edge highlight. (j) Fifth image edge highlight.

Figure 69: Edge extraction.



(a) First blob mask. (b) Second blob mask. (c) Third blob mask. (d) Fourth blob mask. (e) Fifth blob mask.

(f) First image blob highlight. (g) Second image blob highlight. (h) Third image blob highlight. (i) Fourth image blob highlight. (j) Fifth image blob highlight.

Figure 70: Blob of the defects.

In order to fill the edges of the defects in the edge mask and obtain a blob in correspondence of each defect, a *Closing* operation with a circular structuring element of size 50×50 is applied. Finally, to remove the noisy remaining artefacts a *Median Blur* of kernel 7×7 is performed. The motivation behind this operation can once again be found in 2.4. The cleaned-up edge masks can be seen in Figure 70. As expected, defects are solely found in the second image.

4.5 Results

An algorithm performing the steps described above is built and it can be executed via command line or by the method `detect_defects`.

The algorithm:

1. Filters the NIR image by *Median Blur* with a kernel 7×7;
2. Filters the colour image by a *Bilateral filter* using kernel of dimension (7×7) and spatial standard deviation (σ_s) and intensity standard deviation (σ_r) equal to 75;
3. Applies *Tweaked Otsu's Algorithm* to obtain the segmented fruit;
4. Performs two openings through ellipses to clean up the masks from dirt in the background and removes the remaining artefacts by returning the mask of the largest blob (the fruit itself);
5. Applies the mask over the filtered colour image;
6. Turns the masked colour image to the LAB colour space, considering just the A and B channels;
7. Gets a fruit mask by applying colour segmentation with respect to the colour reference of the fruit colour to remove possible stickers;
8. Uses *Canny's Algorithm* by firstly blurring the masked image through *Gaussian Blur*;
9. It obtains the defect blobs by the edge mask and removes the noisy artifacts;
10. Shows the blobs of the defects, counts their occurrences and highlights their position through ellipses.

(a) Second image defects.

(b) Second image defects areas.

Figure 71: Results of the defect detection system implemented by the `detect_defects` algorithm on the second image.

Once again, it can be observed in Figure 71 that the defects in the second image are clearly located using the previously identified parameters:

- $tweak\ factor = 0.4$;
- $t_l = 50$;
- $t_h = 85$;
- $\sigma = 1$.

The *mean* and *inverse covariance matrices* of the kiwi reference colours are pre-computed *offline* and loaded from the memory.

Finally, the time for detecting the defects is computed by applying the `detect_defects` function on each image $1,000$ times, for a total of $5,000$ executions. The algorithm is fast, as it can detect the defects of an image with a mean time of ≈ 0.02 seconds.

4.5.1 Portability

As for the *russet detection system* designed in the second task, this newly built *defect detection system* cannot be ported to other industrial settings by simply changing some of its parameters.

Small changes of illumination can in fact affect the colour distribution of the images and the mean and inverse covariance matrix of the kiwi reference colour have to be manually recomputed with respect to the given environment.

5 Conclusion

The systems proposed for the resolution of the given tasks all manage to satisfy the requirements on the designated test images. In fact, optimally results are reached in the identification of the imperfections on each fruit. Moreover, the proposed architecture has been conceived as a robust solution for a production system. The particular attention put in building a strong segmenting algorithm along with a careful defect identification process would presumably guarantee that the system is able to produce efficient results even for unseen instances of fruits.

In addition, the *defect detection system* built for the first task has been proved to be easily portable to other industrial settings (e.g.: locations with different illumination conditions) by simply tuning its parameters. On the other hand, the *russet detection system* of the second task and the *defect detection system* of the final challenge require the offline computation of specific data, namely the mean and inverse covariance matrices of the various colour references. Thus their portability is not as automatic as the algorithm built for the first task.

Besides, each procedure is exploitable in an *online* production environment as the time needed to perform the inspection of a single fruit is at average in the order of hundredths of seconds.

Unfortunately, the quality of the system on new instances can be solely presumed. In order to verify this statement, a greater amount of test images should be considered. Future work could possibly be related to the extension of the system to the inspection of different fruits. Finally, if a greater amount of test images were provided, another possible implementation could consist in the test of *Machine Learning* methods to solve the tasks and the results could be compared with the current system.

References

[1] UNITEC S.p.a. *UNITEC Group official website*. URL: `https://www.unitec-group.com/`.

[2] UNITEC S.p.a. *Unisorting official website*. URL: `https://www.unisorting.com/`.

[3] Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995. URL: `https://www.python.org/`.

[4] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000). URL: `https://opencv.org/`.

[5] Facundo Nicolas Maidana and Riccardo Spolaor. *Fruit Inspection*. `https://github.com/RiccardoSpolaor/Fruit-Inspection/`. 2022.

[6] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. URL: `https://scikit-learn.org/stable/`.