UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Computer Vision M

Fruit Inspection

Students: Simone Soprani

Marco Barry

Filippo Manuzzi

Academic Year 2022/2023

Index

Introduction

This project aim is to develop software for the automatic detection of defects and imperfections of fruits in a production line. Ideally, the results would allow to identify and select faulty pieces of fruit in a real-time industrial application.

The line is equipped with two cameras providing a colored image and a near infrared (NIR) image of the fruit that is being analyzed respectively.

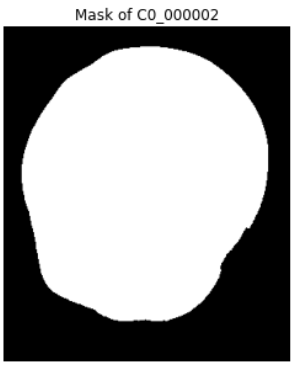
Task 1: Fruit segmentation and defect detection

The first task consists of the detection of external defects of apples.

Initially we proceeded with a segmentation of the images in order to remove the background but keeping intact the apple. To do this we first apply a gaussian blur to the NIR images so that the noise is removed. We apply the smoothing by a 3x3 kernel, a good compromise in order to keep the edges of the apples sharp enough.

Once the image is denoised we segment the image, achieving a clear distinction between background and foreground. To achieve this, we resorted to the Otsu method capable of performing an automatic thresholding of the images.

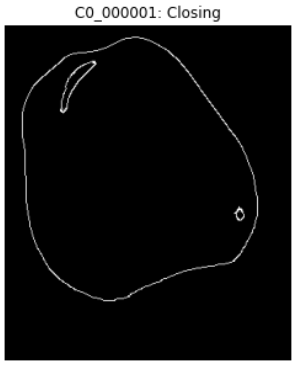
We can clearly see that the apples present holes, to obtain a mask of the apples we need to fill in these holes. To do this we make use of the flood-fill approach.

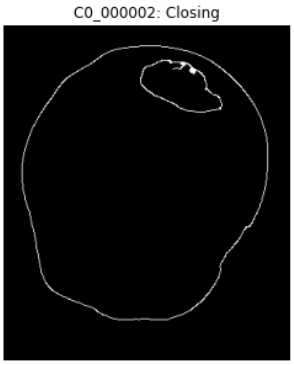


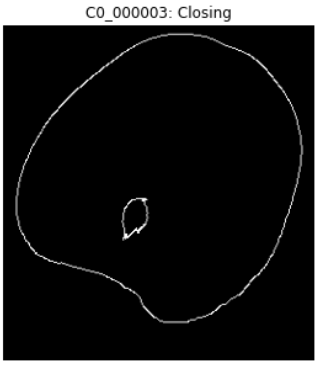
Summing the filled holes blobs with the segmented images presenting holes we obtain a mask of the apple that can be used to isolate the foreground, eliminating the background from the image.

Once the NIR image without background is obtained we can apply a Canny edge detector after a bilateral filter. The filter applied is the best for edge preserving smoothing, and it is tuned to denoise the image with a kernel of 11 pixels and a value of sigma of 35.

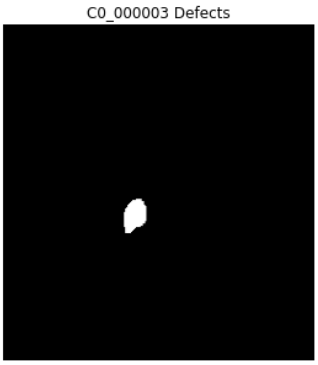
We can then tune the edge detector as a compromise between the training set of images, in order to obtain good enough edges of both the defects and the outer edges of the apple. To have this we tuned the edge detector with a lower threshold of 50 and a higher threshold of 130. To consolidate the edges, we then perform a closing operation with a 5x5 kernel, achieving closed borders of both defects and edges of the apple.





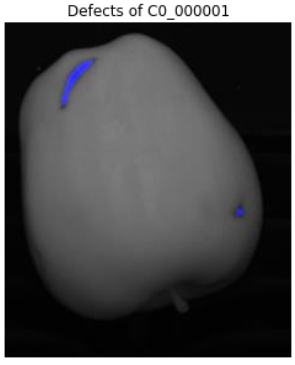


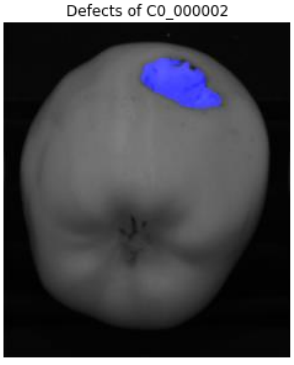
Now to isolate the defects we fill the un-defected parts of the apple with a flood-fill approach and then we subtract the obtained image to the fully filled mask of the apple to highlight the defects, and we perform an opening with a 3x3 kernel to the obtained image to remove the parts of the apple not related to the defects.

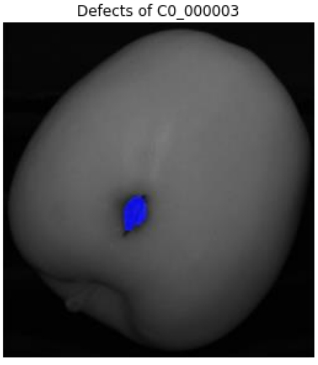


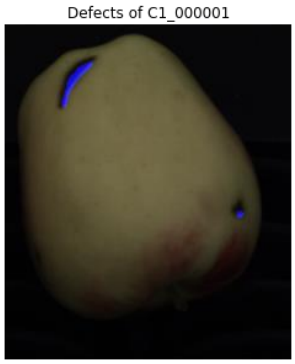
Once we have obtained the defects, we can count the number of blobs in each image and tell the user if the apple is compliant or if it has defects and the number of them. To do this the *findContours* function has been employed.

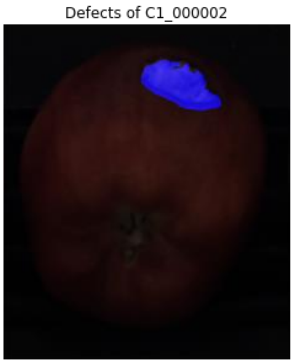
Finally, we can show the user the defects; in order to do this, we change the color of the defects, and we superimpose them to the original NIR and colored images.

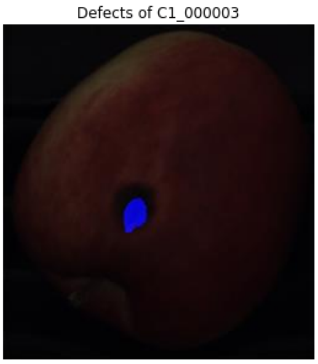












The result is satisfactory as the program can detect all defects of the training set without false positives and negatives.

Task 2: Russet detection

We can observe that this task is centred around colour identification and heterogenous distances, since the information that an area of the apple is a russet comes from a change in hue of the surface (in other terms, the infrared and, in part, the grey-scale images do not pick up on this type of imperfection).

The way this task has been implemented and thought out is the following. The code of the second task can be logically divided into six sections:

1. Importing of libraries and opening of sample images
2. Identification of the Area of Interest and creation of a mask
3. Use of inRange Operator to select a range of colours used for Russet detection (Gross Colour Segmentation)
4. Calculation of Mean and Covariance Matrix
5. Mahalanobis distance to identify Russets
6. Polish, count and print of results

1. Import of Libraries and opening images

Most straightforward segment of code, the following libraries are imported

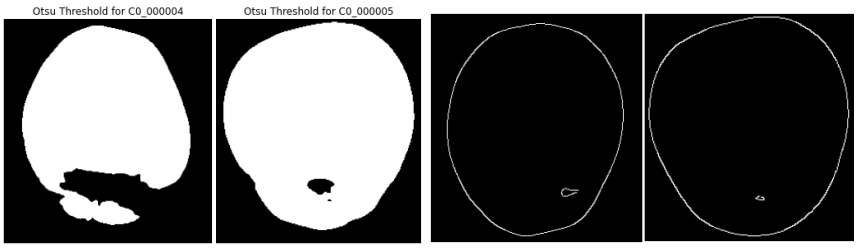
```
# Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt
from scipy.spatial import distance
```

And the four Task 2 sample images of apples are opened and saved in arrays (coloured, grayscale, and Infrared)

2. Identification of the Area of Interest and creation of a mask

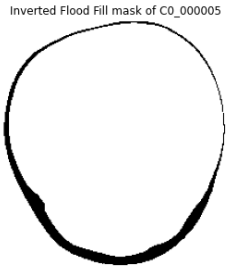
Much like in task one, it is necessary to execute operations exclusively on parts of the image that contain the apple. If this condition is not met, this would completely offset calculations of means and variances, ruining any chance to detect russets correctly.

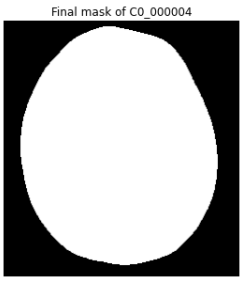
So firstly, we apply Gaussian blur, followed by an automatic Otsu Thresholding of the image. This produces results which are not closed and that have holes, so to correct and complete the mask, Canny Edge detection is applied to the bilaterally filtered images (in a way to preserve edges and clean up any imperfection).

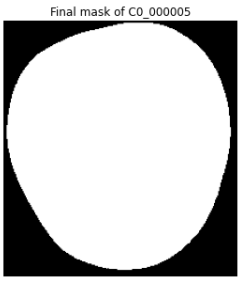


Now, the holes inside the mask are filled via floodfill of the inverted mask. For apple number 2 the mask is now complete, but apple number 1 is not yet complete as shadows have created an open hole that does not get filled by the FloodFill operation. Therefore, we apply another FloodFill with the edges as the mask,

thus creating a contour of the mask to which apply invert and then use OR operand with the unfinished mask resulting in a satisfactory mask.







3. Gross Colour Segmentation

To start the identification of the russet we first assume on the possible colours any russet can take. Using a Colour Selector Program (https://github.com/alkasm/colorfilters), it was possible to find a good range of values in an appropriate colour space (the best results are obtained by using HSV colour space, although HSL, Luv, Lab and RGB were also tested) where the brown of the russets of the training set is mostly selected, and the rest of the apple is filtered out.

Once we have the brown russet pixels, we then subsequently use erode operation to deselect any outlier pixels. The logic being that a Russet is a wide surface of irregular colour on the apple, outlier pixels are selected because they may be within range, but if they are not closely clustered together, they are unwanted as they are not part of a russet. Thus, by eroding, the outliers will be eliminated and only 'real' russet colour will be considered (in other words, small russets or outlying pixels are eliminated and considered part of Russet selection)

We now have a new mask of Russet brown pixels. We can use this to calculate average russet colour and average non-russet colour.

```
img = (image.copy()).astype('uint8')
hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
# define range
lower_brown = np.array([0,0,0])
upper_brown = np.array([20,152,150])
# Threshold the HSV image to get only brown colors
mask = cv2.inRange(hsv, lower_brown, upper_brown)
kernel = np.ones((5,5), np.uint8)
nu_mask = cv2.erode(mask, kernel)
# Bitwise-AND mask and original image
res = cv2.bitwise_and(img,img, mask= nu_mask)
RANGE.append(res)
other = img - cv2.bitwise_and(img,img, mask= mask)
```

4. Calculation of Mean and Covariance Matrix

The mean is calculated for both the in-range pixels and the out-of-range pixels, where in-range pixels are the pixels that were selected by the previous in-range and erosion operations; covariance matrix is calculated for the apple.

The difficulty is to select the correct pixels and make them usable by the functions numpy.mean and numpy.cov. To do this we considered the coloured in range and out of range images, cycled through each pixel and if it wasn't black, i.e., all values are different to 0, insert them in either a SLT pixel array.

The reason we put the values in a "wide" matrix is to use the inbuilt mean and cov numpy functions. The mean values and the inverse of the covariance matrix will be used later to determine whether a pixel is part of the russet or not.

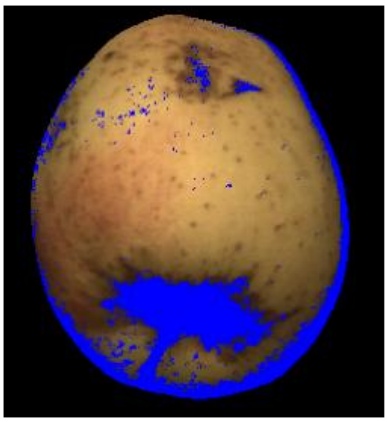
```
MEAN RANGE

APPLE 1 -> Colored pixels: 547 . Black pixels: 66953
Mean: [46.89031079 35.68738574 21.23583181]

APPLE 2 -> Colored pixels: 4966 . Black pixels: 47834
Mean: [123.6804269   98.25211438  56.07974225]


------------------------------------------------------------


MEAN NOT RANGE

APPLE 1 -> Colored pixels: 34435 . Black pixels: 33065
Mean: [136.27861188  99.97215043  55.09266734]

APPLE 2 -> Colored pixels: 24041 . Black pixels: 28759
Mean: [102.57135726  95.68499646  53.24795142]


------------------------------------------------------------


COVARIANCE MATRICES
...
 [[ 0.01018576 -0.00933941 -0.00256526]
 [-0.00933941  0.0576473  -0.07810716]
 [-0.00256526 -0.07810716  0.13535266]]
```

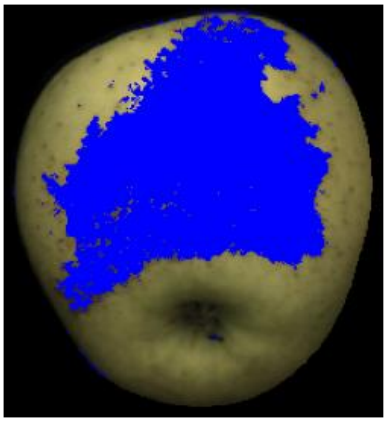
```python
for image in NOT_RANGE:

    SLT = []
    BKG = []
    for a in range(image.shape[0]):
        for b in range(image.shape[1]):
            pix = image[a,b,:]
            if (pix[0] != 0 and pix[1] != 0 and pix[2] != 0):
                SLT.append(pix)
            else:
                BKG.append(pix)
    print("APPLE",j+1,"-> Colored pixels:",len(SLT),". Black pixels:", len(BKG))

    SLT = np.array(SLT).astype('uint8')
    mean = np.mean(SLT, axis=0)
    MEAN_NR.append(mean)
    print("Mean:",mean,"\n")
```

5. Mahalanobis distance

Now all necessary values are known and calculation of Mahalanobis distance is possible. The strategy used is to calculate Mahalanobis distance of each pixel of the apple from both the in-range mean and the out-of-range mean and mark the pixels that are closer to the in-range mean. This allows to avoid arbitrariness in selecting a cut-off value for colour distance from russet colour.

We use Mahalanobis distance as it is a more generalised distance which considers the variance and covariance of the dimensions, in our case the 3-dimensional colour space.



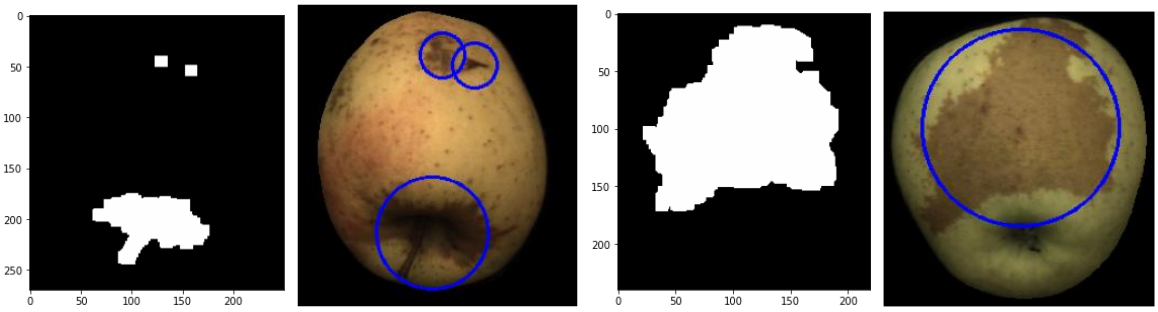


6. Final Operations

The marked pixels are mostly correct, but a large problem persists: the shadow on the edges of the apples is considered a russet. To solve this and try and keep the algorithm as general as possible, the solution used is to ignore the edges of the apple (19-pixel wide edge of apple is ignored).

This is done by dilating the edge image found previously with Canny Edge Detection and using it to eliminate the border from the full image mask. This creates a mask of only the central part of the apple, where russet detection is possible and correct.

The last operations are then to count russet sections a, by counting the blobs, and circle the russets on the coloured image of the apple.

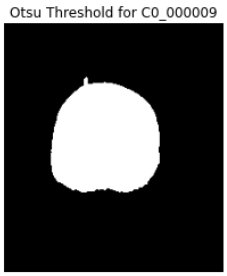


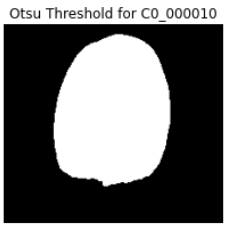
Final Challenge: Kiwi inspection

The final challenge consists of spotting possible defects of kiwis.

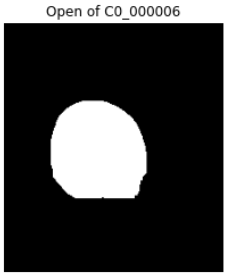
In order to isolate the kiwis from the background, we remove the background in fashion similar to the one of the first two tasks. We firstly apply a gaussian blur with a 3x3 kernel to denoise the image, retaining only the important information.

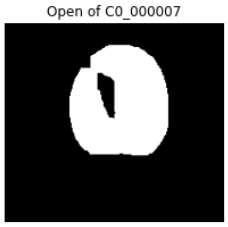
Once the image is denoised we proceed with binarization aimed at creating a mask to later superimpose on the original image, in such a way to eliminate the useless background information. We then apply the Otsu algorithm, able to find an optimal threshold for the images.

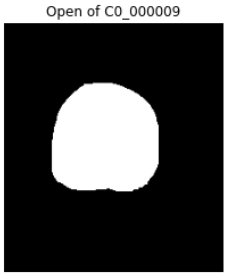


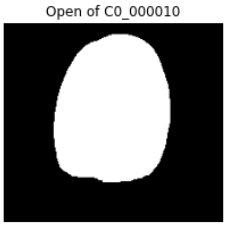


The result is not yet satisfactory as portion of the background are incorrectly classified as foreground; to work around this problem we perform an opening with a 17x17 kernel, big enough to remove the parts of the background such as the label but not so big as to lose too much information.

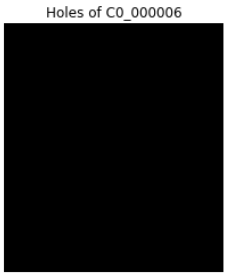


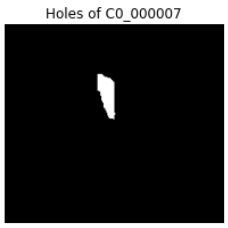


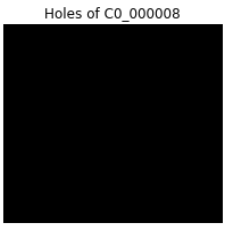


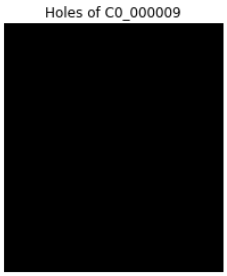


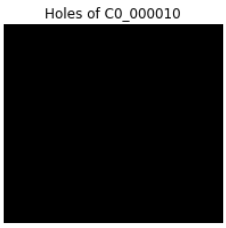
Having the segmented images we can identify the defects right away through a modified flood-fill approach, in which we identify the holes in the blobs.











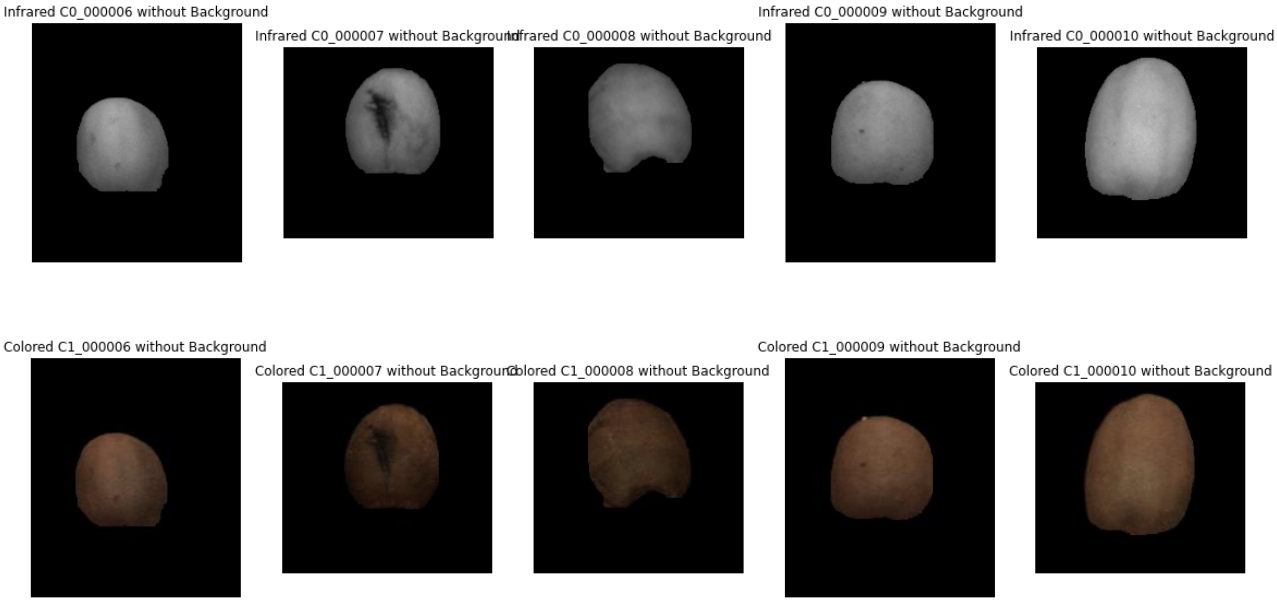
Having now images that are empty if the defect is not present, we can proceed to count the number of blobs; the blobs remaining in these images will in fact be the defects of the kiwi.

Lastly, we isolate the kiwis from the background; to do this we do an Otsu thresholding of the logical OR of the previous result of Otsu segmentation, presenting the defects, and the isolated defects, in such a way to

obtain full kiwi masks.

The mask following this operation still presents a portion of the background, to obviate this problem we perform an opening of kernel 17x17.

We can finally superimpose the mask to both NIR image and colored image.



To present the user to a visual notification of the defects of the kiwi we highlight the defects mask obtained previously and we superimpose it to the images above.