CMP3108M Image Processing Assessment 1 Report

A logo of a university

Description automatically generated

Mathews Joy

[25186202@students.lincoln.ac.uk](mailto:25186202@students.lincoln.ac.uk)

JOY20779936

University of Lincoln

School of Computer Science

**Task1 – Pre-processing**

**Task 1.1 – 1.2 Load input image & Convert input image to greyscale**

For this task we can simply load our desired image from our image folder utilizing the “imread()” function by passing in the full image name. The “rgb2gray()” function was used, where we pass in the image we previously read in, into this function. This function converts the original image to greyscale by eliminating “the hue and saturation information while retaining the luminance” (MathWorks, 2023a).

**Task 1.3 Rescale image using bilinear interpolation**

Interpolation is “process of using known data to estimate unknown values at other locations [essentially] an image method to increase [or decrease in our case] the number of pixels in a digital image” (Dr Duan, 2023). Bilinear interpolation produces smoother results then methods like nearest neighbour interpolation and is quicker then bicubic interpolation. Bilinear interpolation is achieved using the “imresize()” function, passing in the greyscale image from task 1.2 as the input, setting the 2nd parameter of the function to “0.5” to indicate we want to reduce image size by half, and the 3rd parameter we set too “bilinear” to indicate that we want to use the bilinear interpolation method to carry out the resizing.

**Task 1.4 Produce histogram for the resized image** **before enhancing**

The “histogram()” function was used to generate a histogram of the image from task 1.3. Histograms are a great way to be able to visualise the image in terms of bins as rectangular bars, the height of the bar indicating the number of elements (number of pixels at each brightness level) in the bin, helping show the underlying data distribution of the image.

**Task 1.5 Enhance the image**

2 popular image enhancement methods are contrast stretching and histogram equalization. Contrast stretching aims to increase picture contrast by extending the range of intensity values included in the image to cover a desired range of values. Histogram equalisation modifies the intensity values of all the pixels in the image such that the histogram is flattened. Both methods were trialled upon our scenario. However, the output of histogram equalization using the “imhisteq()” function, was harsh and over exaggerated the image darkness and contrast. Instead, contrast stretching via the “imadjust()” function was preferred as it give a more subtle enhancement but helped bring the details of the image out more.

**Task 1.6 Histogram after enhancement**

Like step 1.4 the “histogram()” method was used but using the output image from task 1.5 as the input. We observe between the histogram before and after enhancement (*figure 1 in task 1.8 section*), there is little difference between the 2 graphs. This is because contrast stretching maps the minimum and maximum intensity values in an image to the minimum and maximum values in the desired range, respectively. The overall shape of the histogram remains the same after contrast stretching. On the other hand, histogram equalization modifies the intensity values of all the pixels in the image such that the histogram is flattened (Luke, J, 2016).

**Task 1.7 Image Binarization**

Binarization is the process of converting greyscale / coloured images into a binary image (black and white pixels). It can be useful for picking out key objects. The “imbinarize()” function was used, it takes several parameters, including the input grayscale image, the method of thresholding, the polarity of the foreground, and the sensitivity. For the threshold value we set it to “adapative”, this method computes a threshold for each pixel based on the local image contrast. Alternatively, we can use the histogram from task 1.6 to determine a threshold value, which for our example we could use a value of 120 (divided by 255), as this is the point on the graph where the peak starts the rise. The polarity of the foreground was set to “dark”, this means that the foreground is darker then the background. Finally, the sensitivity was set to 0.5, as this value allowed the screws and washers to be seen the most. Furthermore, for the threshold value methods such as the “Otsu method” can be used, “In the simplest form, the algorithm returns a single intensity threshold that separate pixels into two classes, foreground and background” (Otsu’s method, 2023). However, Otsu method is preferred on bimodal histogram distributions.

A screenshot of a computer screen

Description automatically generated**Task 1.8 Display the resultant images for task 1**

Figure 1. Graph of the re-sized image, histograms before and after enhancement, enhanced image and binarized image.

**Task 2 – Edge Detection**

Edge detection is a technique used to find places in the image where the intensity changes quickly. Before applying edge detection, it is common to perform noise reduction in the image, as noisy images prompt variability in the local contrast along an edge. Two noise reductions methods where experimented with; median filtering using the “medfilt2()” function and also gaussian filtering using the “imgaussfilt()”. Median filtering was found to provide better results for edge detection in our scenario, as when applying gaussian filtering it would smooth the image too much making the edges harder to detect. However, the gaussian filtering function has a “sigma” parameter (level of blurring) that can be adjusted, but as median filtering worked out the box and showed promising results, it was chosen to stick with this. After noise reduction, 3 edge detection algorithms using the “edge()” function was used to find the best edge detection result. The algorithms being Canny, Sobel, and Prewitt. The results can be seen in figure 2.

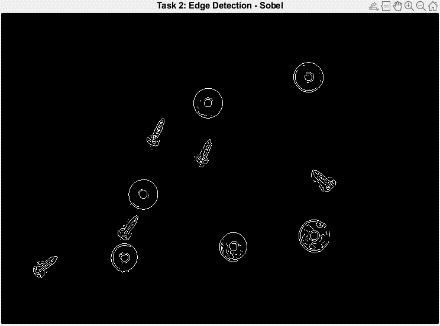
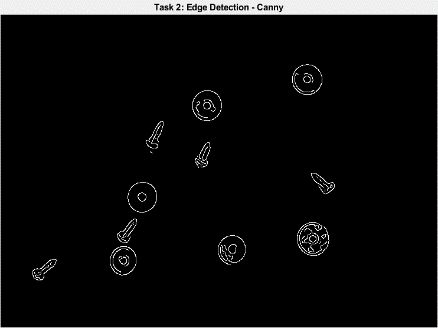
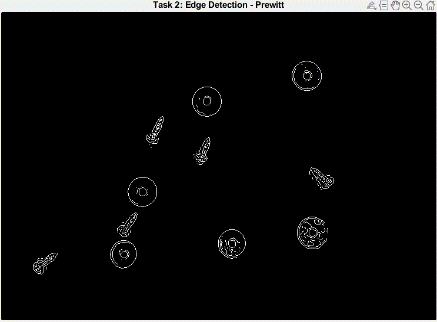


Figure 2. Image results using different edge detection algorithms (left to right, sobel, canny, prewitt).

Canny gave the best result, as all the edges it detected are smooth and most accurate. The Prewitt and Sobel methods gave similar results but were missing some edge values and showed more noise in the resultant image compared to canny. This was expected as “canny uses hysteresis to thresholding to detect edges allowing for more accurate detection” (Tsankashvili, N, 2020). On the contrary, canny is more computationally expensive, so if computational resource was a key factor canny would not be optimal.

**Task 3 – Simple Segmentation**

A screenshot of a computer

Description automatically generatedTo segment the image, we first must ensure all edges of the screws and washers are connected to be able to fill them so we can segment the objects. We can use the morphology method of closing which performs dilation then erosion of the image, essentially connecting any broken edges. The “imclose()” method can achieve this, and we also define and pass in a structuring element to the closing function. The “strel()” function was used to create a structuring element the shape of a disk of radius 3, as it fits the general shape of the screws and washers. To then segment the objects we can use the “imfill()” function and passing “holes” as a parameter option, that will then fill in the holes/regions in a binary image. We then get the screws and washers filled in as white and the background as black, we can then easily distinguish the objects. Finally, to remove any small objects (generally caused by noise) that are not screws/washers we can use the “bwareopen()” method with a threshold value of 20. Giving the result shown in figure 3.

Figure 3. Segmented image result

**Task 4 – Object Recognition**

To recognise the specific objects and label code them, we must have a certain feature we can use to differentiate the 2 possible object categories. It was decided to use each segmented objects aspect ratio to identify if it is a screw or washer, the aspect ratio being ratio of the width to the height of an object. Aspect ratio was chosen as the comparative feature as our sample images are all taken from the same bird’s eye view angle, and are similar zoom levels, so the size and shapes of each screw and washer are on the same level, making them easy to differentiate via aspect ratio. We first use the “regionprops()” function that allows the extraction of specific properties for each object in a segmented image, in particular we want the 'MajorAxisLength' and 'MinorAxisLength' properties. The aspect ratio is then calculated used the major and minor axis length properties and is used to assign a colour to each object in the image. [If the aspect ratio of an object is greater than 2, it is assigned the colour red to indicate a screw, otherwise, it is assigned the colour orange](https://www.mathworks.com/help/images/ref/regionprops.html) to indicate a washer. The colours are stored in the “cmap” variable which is used as the colourmap, [that is used to assign colours to each object in the image based on its aspect ratio](https://www.mathworks.com/help/matlab/ref/colormap.html)(MathWorks, 2023b).

**Task5 – Robust Method**

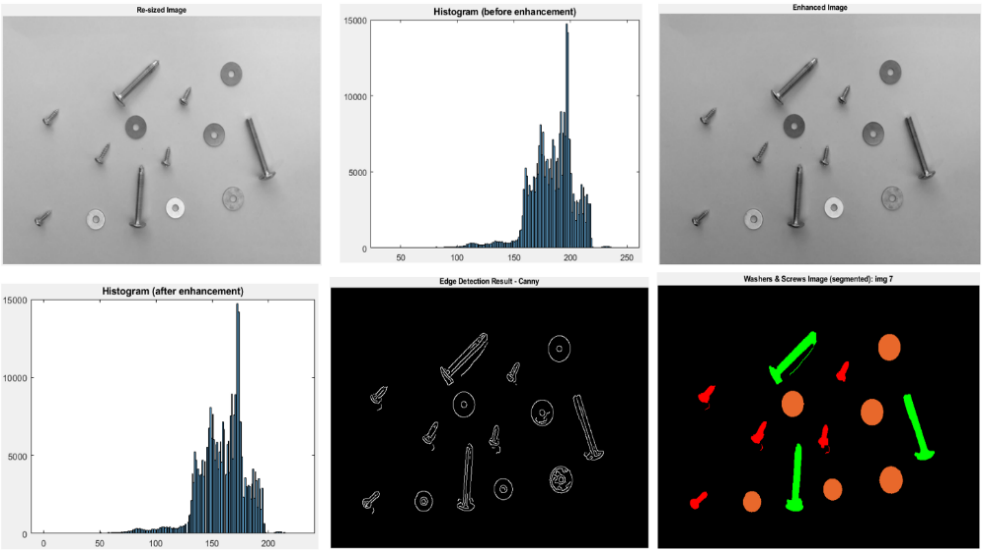
For task 5 to achieve a robust method, we must identify the key points of code from tasks 1 – 4, that are hard coded to our first sample image. Once the hard coded elements are adapted, the code from task 1-4 can be implemented into a function and we can loop through all 10 samples images, apply our custom function, and return the results. From tasks 1-5 there are no major hard coded elements, however for task 4 it works only for given screw and washers but cannot differentiate large screws. We opted to add an additional “elseif” statement, which checks if the aspect ratio of an object is greater then 1.8 but less then 4 it is classified as a small screw, however if the ratio is greater then 4 it is a large screw. We then take the code from task 1-4 with the modified task 4 and create a function named “screw\_washer\_detection“, that takes a single argument that is the input image you wish to segment, and returns 2 objects “labeled\_image” (segmented image) and “cmap” (colourmap of the segmented image). A loop is then created from 1 – 10 (for all 10 images in our dataset), that extracts the image name and attempts to read in each image in a “try,catch” block, catching resulting errors in the catch block such as a invalid file name or file was not found error (MathWorks, 2023c). If the image was successfully read, we can pass it through the custom function we defined that will go through the task 1-4 code and segment the objects and return a successfully segmented binary image, along with its colourmap to distinguish between washer, small and long screw. Example of the full process of a random image (IMG\_07) is displayed in figure 4.

Figure 4. Full robust process output results at each stage.

**Task 6 - Performance Evaluation**

This task involved calculating the dice score (Sørensen-Dice similarity coefficient) between an image and its ground truth. The “dice()” method is used to compare our segmented images from task 5 to its ground truth, a dice score of 1 indicating a perfect overlap and 0 showing no overlap. The formula for dice score being: **dice(A,B) = 2 \* | intersection(A,B) | / ( | A | + | B | )**, where | A | and | B | are the number of elements of each set (MathWorks, 2023d). Also the precision and recall values are calculated for each segmented image and ground truth using “bfscore()”, precision is “the fraction of detections that are true positives rather than false positives” and recall is “the fraction of true positives that are detected rather than missed” (MathWorks, 2023e). Formula for precision and recall being: **Precision = True Positive / True Positive + False Positive**and**Recall = True Positive / True Positive + False Negative**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IMG\_01,**  0.945,  1,  1 | **IMG\_02,**  0.928,  0.969,  0.996 | **IMG\_03,**  0.941,  1,  1 | **IMG\_04,**  0.910,  0.957,  0.989 | **IMG\_05,**  0.898,  0.943,  0.995 |
| **IMG\_06,**  0.870,  0.942,  0.988 | **IMG\_07,**  0.910,  0.928,  0.999 | **IMG\_08,**  0.938,  0.982,  1 | **IMG\_09,**  0.917,  0.951,  0.999 | **IMG\_10,**  0.940,  0.995,  1 |

Below are the evaluation metric scores (figure 5), and the mean/standard deviations (figure 6).

|  |  |  |
| --- | --- | --- |
|  | **Mean** | **SD** |
| **Dice** | 0.920 | 0.023 |
| **Precision** | 0.967 | 0.026 |
| **Recall** | 0.996 | 0.004 |

Figure 5. Table of metrics [image, dice, precision, recall]. Figure 6. Table of mean & SD.

Overall, we can observe we achieved a mean dice score of 0.920 across all 10 images meaning a high degree of similarity between the segmented image and ground truth, with a standard deviation of 0.023, showing how spread out the predictions are. The mean precision (0.967) and mean recall (0.996), show the segmented results from our solution, is of good quality and generally accurately represents our ground truth. In image samples 05 and 06 we have a below 0.9 dice score, if we look at the ground truth for these images, they present more challenging scenarios where the screws are heavily touching each other. To improve this, we can implement the “Otsu’s method” for the binarized images and hyperparameter tune the method in MATLAB across the 10-sample image to perform automatic thresholding.

**References**

Dr Duan, Wenting, Lecture 3: Image Enhancement, lecture slides, Image Processing, CMP3108M, University of Lincoln, delivered 10th October 2023

Luke, J. (2016) Difference between contrast stretching and histogram equalization, Stack Overflow. Available at: https://stackoverflow.com/questions/41118808/difference-between-contrast-stretching-and-histogram-equalization (Accessed: 10 December 2023).

MathWorks (2023a) Convert RGB image or colormap to grayscale, Convert RGB image or colormap to grayscale - MATLAB rgb2gray - MathWorks United Kingdom. Available at: https://uk.mathworks.com/help/matlab/ref/rgb2gray.html (Accessed: 01 December 2023).

MathWorks (2023b) View and set current colormap - MATLAB colormap - MathWorks United Kingdom. Available at: https://uk.mathworks.com/help/matlab/ref/colormap.html (Accessed: 13 December 2023).

MathWorks (2023c) Execute statements and catch resulting errors - MATLAB try catch - MathWorks United Kingdom. Available at: https://uk.mathworks.com/help/matlab/ref/try.html (Accessed: 15 December 2023).

MathWorks (2023d) Sørensen-Dice similarity coefficient for image segmentation - MATLAB dice - MathWorks United Kingdom. Available at: https://uk.mathworks.com/help/images/ref/dice.html (Accessed: 16 December 2023).

MathWorks (2023e) Contour matching score for image segmentation - MATLAB bfscore - MathWorks United Kingdom. Available at: https://uk.mathworks.com/help/images/ref/bfscore.html (Accessed: 16 December 2023).

Otsu’s method (2023) Wikipedia. Available at: https://en.wikipedia.org/wiki/Otsu’s\_method (Accessed: 11 December 2023).

Tsankashvili, N. (2020) Comparing edge detection methods, Medium. Available at: https://medium.com/@nikatsanka/comparing-edge-detection-methods-638a2919476e (Accessed: 11 December 2023).

**Appendix**

**A screenshot of a computer program

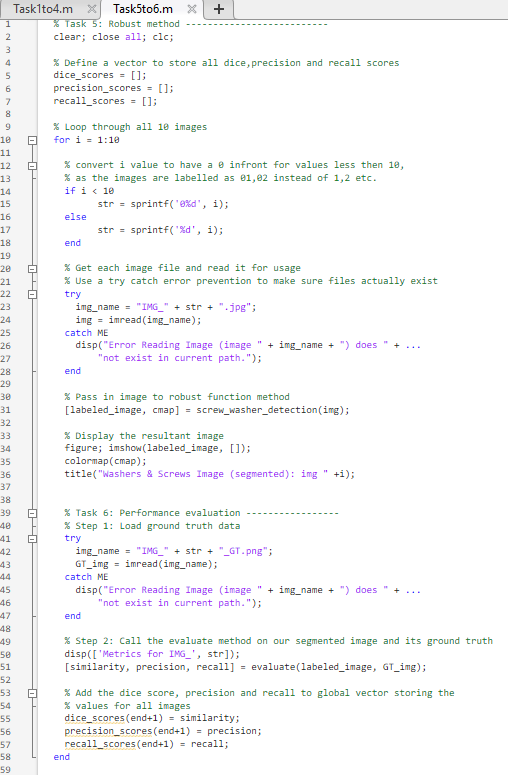
Description automatically generatedTask1to4.m**

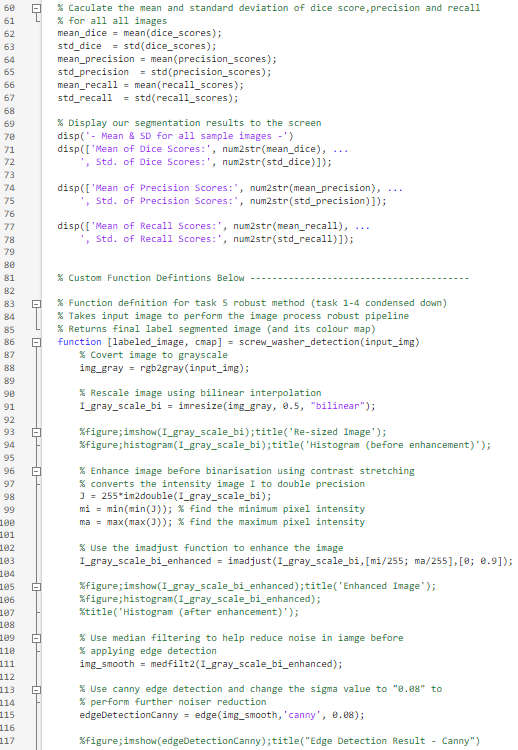
**A screenshot of a computer program

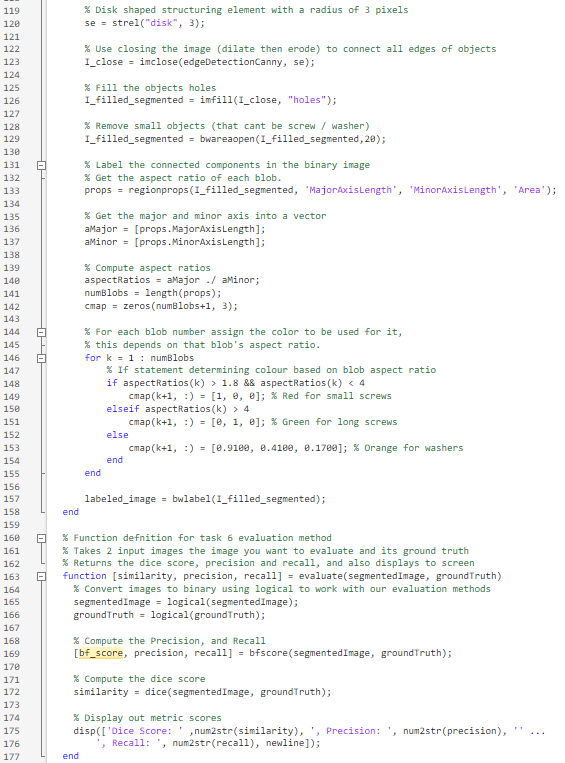
Description automatically generated**

**A screenshot of a computer code

Description automatically generated**

**Task5to6.m**

****

****