UNIVERSITY OF
LINCOLN

Computer Vision

CMP9135M | Assessment Item 1

Peter Hart | 12421031

Task 1: Image Segmentation

The first task presented in this assignment involved two input images 'Topimage0000.bmp' and 'Topimage0013.bmp', these images were greyscale food packaging images which were captured with a black background. Ultimately, the objective of this task was threefold; to produce an accuracy binary mask representation of the food packaging tray object, to segment and crop the food packaging tray and to segment the packaging label.

The images were first acquired by presenting the user with a prompt interface, to which they are provided they opportunity to navigate and select any number of food packaging input images which will be subsequently processed. To this end, each input image is then saved into a 2048 x 2950 matrix, which effectively stored the pixel intensity and spatial data pertaining to each grayscale image. However, further observation indicated that the black background pixels contained within the image featured a small amount of noise referred to as Gaussian noise, this is a type of statistical noise whereby the probability distribution of the noise intensity values is founded upon a normal or Gaussian distribution. This Gaussian noise was likely caused by the camera sensor during image acquisition, and therefore caused intermittent pixel intensity variations to be distributed throughout the input images. Therefore, a small amount of image pre-processing was required, whereby a small Gaussian spatial filter was applied throughout the image prior to further image processing techniques being applied to the input images.

As previously mentioned, a fundamental objective which is desired to be accomplished within this task is an accurate segmentation depiction for the food tray object which exists within both input images, such that a complete binary mask can be observed. To this end, an observation into the histogram of both input images revealed that the pixel intensity distribution was limited, thus the range of the pixel intensities that were featured within the image was low. For example, the original histogram for one of the input greyscale images featured a large cluster of pixels with a low intensity, whereas the frequency of higher intensity pixels could be observed as being underpopulated. As such, the cluster of low intensity pixels lead to an overall darker image being depicted, this consequently impaired the image contrast and in turn limited the segmentation accuracy potential.

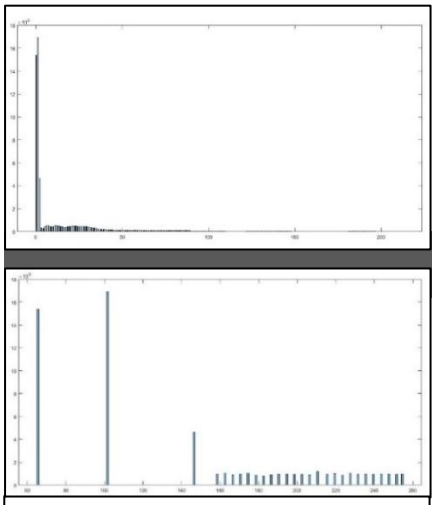

Figure 1. Histogram comparison: before and after histogram equalisation is applied respectively.

Thus, this issue was mitigated through the application of a histogram equalisation image enhancement technique. Effectively, histogram equalisation manipulates the original image histogram such that the original pixel intensity distribution was mapped to a uniform distribution, whereby the intensity values are spread throughout the entire intensity range such that each possible intensity value was equiprobable for being featured within the image. To surmise, this increase in the range of pixel intensity values allowed for increased illumination within the image and in turn increased the amount of contrast, allowing more discernible features to be observed and extracted from the image.

With the enhanced input images, an image segmentation technique can subsequently be applied for the extraction of the food tray object that reside within both input images, this was ultimately accomplished through the application of a thresholding technique. As elaborated by Shapiro and Stockman (2001), the objective of the thresholding image segmentation technique was to use the input greyscale images and create a new binary image, whereby the background and any detected foreground objects can be distinctly observed. Ultimately, the image was segmented through a thresholding technique by iterating through all the pixel intensity values within the input images, comparing them to a threshold pixel intensity value T. To this end, a new image matrix was created at the same dimensional size as the original input images, where all pixel intensity values above this threshold were considered as part of the food packaging object and assigned a binary intensity value of 1, whereas the pixel intensity values which fell below this threshold value were discarded as irrelevant background pixels and assigned a binary intensity value of 0. At the outcome of this, a new binary image depicting a binary segmentation mask was created, whereby all the high intensities described the food tray objects and the low intensities described irrelevant pixels.

Otsu thresholding was utilised for the acquisition of an appropriate threshold value, this largely operates by iterating through all possible threshold intensity values and estimating a measure of intensity spread, such as the mean pixel intensity value, that becomes apparent on either side of the threshold. As explained by Gonzalez et al (2013), this will subsequently be incorporated for estimating the "between-class" variance, where a higher variance is indicative of a higher threshold segmentation performance. This procedure would continue to be applied throughout all possible intensity threshold values "to maximise the separability of the resultant classes in grey levels" (Otsu, 1979).

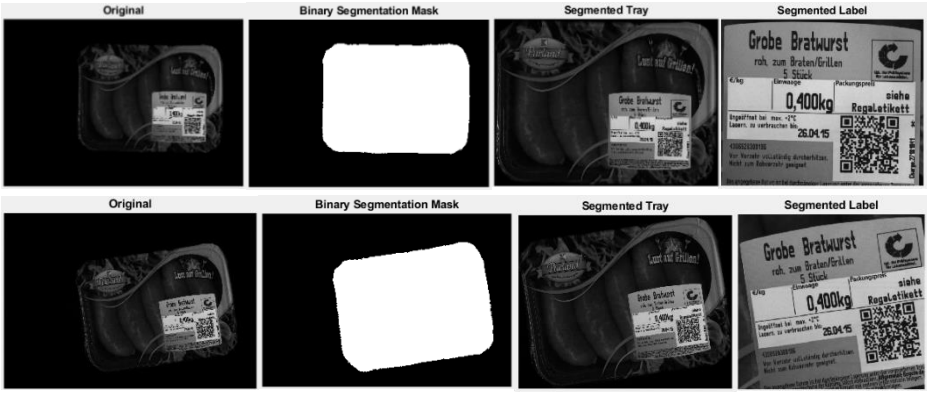

Figure 2. Final segmented outputs using Otsu's thresholding.

While a binary mask has successfully been established, an issue of paramount importance which arose is the sudden increase in noisy pixels which can be observed to be distributed throughout the output binary mask, whereby random scatters of high intensity pixels can be observed amongst the background pixels. While the previously discussed histogram equalisation technique was effective for increasing the amount of contrast in the image, this technique also procured a side effect of also increasing the amount of contrast that resided within some of the undetected noisy background pixels. Morphological operations were applied to the output binary mask, this refers to a tool for extracting image components that are deemed to be useful in regards to the visual representation and description of a region shape where morphological operations such as erosion or dilation can be used to process an image in regards to a logical union or intersection operation on an image (Gonzalez et al., 2013).

A morphological erosion operation was initially applied to minimise the frequency of unwanted object pixels that are distributed throughout the foreground of the image by forcing all the object pixels featured within the tray mask to undergo a "shrinking procedure" (Farhan et al, 2016), though a side effect of this procedure caused an unnecessary loss of data due to an amount of object pixels being removed from the target tray object within the mask, thereby losing segmentation accuracy. Thus, this is subsequently followed by a morphological dilation, effectively applying a "swelling procedure" (Farhan et al, 2016) with the remaining object pixels. This order of operations, namely erosion followed by dilation, can be referred to as the morphological opening of an image (Parker, 1997), and was effective for removing any noisy foreground pixels while also restoring any pixels lost with the food tray object.

With an effective binary mask generated for the food tray object within each input image, the binary mask could be referenced when attempting to crop and extract the tray from both input images. As such, with the noisy pixels removed, the tray foreground object was measured regarding the object positional proximity relative to the image spatial domain. Using the MATLAB regionprops function, the four corner coordinates of the object could be detected and subsequently utilised for estimating a bounding box around the entire object, the data of which was then referenced when cropping and extracting the entire tray object from the original image without image enhancements.

However, one final objective which remained was the extraction of the smaller label area which resided on the bottom right corner of the food packaging tray. This was accomplished through a similar process, whereby a new mask was established based on the original pixel intensities of the input image through an Otsu thresholding segmentation technique, therefore the new threshold value would instead by estimated based on the previously created cropped image using the previous food packaging tray mask. As such, a morphological opening operation was applied to the output binary mask to minimise noisy foreground pixels while maintaining the integrity of the relevant foreground objects. In turn, a new matrix was created which depicts the pixel locations of each individual foreground object within the binary mask, such that the pixel intensity values of each connected component are assigned a unique integer value to retain the spatial information.

Figure 3. Example of binary mask output upon applying morphological opening for the label.

As a result, the area of each unique object within the binary mask could be computed, whereby the largest area should depict the label itself, therefore the object with the largest area can be extracted while the other objects are merely discarded. Thus, upon a single object remaining on the mask, this object can in turn be extracted by detecting the four corners of the object, whereby these four coordinates could once again be used to extract the object with a bounding box which in turn could be referenced when attempting to crop the final masked output.

Task 2: Feature Calculation

The second task of this assignment revolved around the extraction of key features from a given input image; ImgPIA.jpeg and applying analysis both in the frequency and spatial domains of the image.

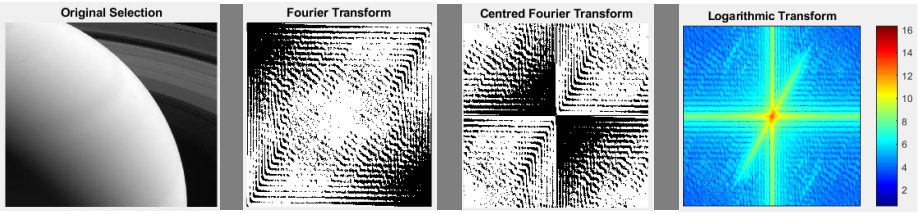

Figure 4. Fourier transforms computed based on a randomly selected region of the input image.

A spectral approach toward textural analysis follows the extraction of image features by observing the properties of an image within the Fourier domain, otherwise referred to as the frequency domain, where each point within the Fourier domain is representative of a unique frequency which coexists within the spatial domain of the image. As such, a fast Fourier transform (FFT) algorithm was utilised to compute the discrete Fourier transform (DFT) for converting the input image into a two-dimensional Fourier transform equivalent using the inbuilt fft2 function within MATLAB. As such, the Fourier transformed image can be interpreted as the further away an image point may be from the origin, the higher the frequency value will be.

This Fourier transform output was then manipulated further by re-centring the zero-frequency component using the inbuilt fftshift function in MATLAB, where the four quadrants of the original Fourier are rearranged such that the centre of the Fourier transform is localised to the middle of the image, such that the origin is set to the polar coordinate position equivalent of (1,1). The resulting arrangement of the Fourier transform can subsequently allow symmetry amongst the various frequencies to be assumed. However, the range of the intensity values within the Fourier transform image can be observed to be too large, thus some of the intensity values appear black. As such, a subsequent logarithmic transform was applied to the centred Fourier transform, the outcome of which presents a new image which depicts the full frequency intensity range.

Similarly, the frequencies which fall within a specific direction could be extracted by applying a circular mask at different radius intervals, where the origin of the circular mask was set to the centre of the image. As such, a larger radius in turn infers a larger circle which can be used to evaluate a larger distance of frequencies from the origin of the Fourier transformed image. As such, a binary mask was computed which depicted a circular shape generated

at a given radius, the logarithmic Fourier transformed image was then multiplied with the circular mask where any frequencies which could be observed as having a greater distance away from the mask were set to 0. This in turn was applied to a total of four different distance examples, namely where r was equal to a radius of 25, 50, 75 and 100. Similarly, the distance feature extractions were estimated by performing a summation of the frequency intensities contained within the extracted region to present a frequency magnitude estimation, where the frequency magnitude was assessed at different distances of the logarithmically Fourier transformed image.

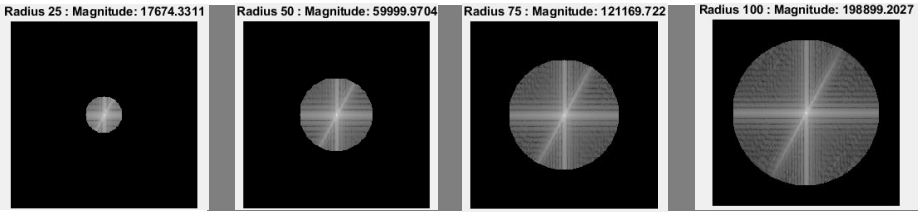

Figure 5. Example outputs when extracting frequency features from a range of distances.

As such, features could be extracted through frequency filtering with the logarithmic transformed image. A binary mask was created in the form of an angular mask, whereby a function was created to iterate through the entirety of the grid of frequencies depicted within the logarithmic Fourier transformed image and evaluates polar coordinate location of each frequency against the Fourier transform centre origin using the inbuilt MATLAB function atan2. For example, the frequency located at coordinate (18,18) presents an angle of -2.356 radians, this coordinate in turn was evaluated as not being located within the currently set angle so therefore it would not be included in the angular mask. Furthermore, this was repeated throughout the entire Fourier transform with a set angle interval of 45˚ or approximately 7.854 radians to simulate angular regions of 45˚ being extracted from the image. Thus, the angular features were collated by multiplying the logarithmic transformed image against the estimated angular mask, where any frequencies which failed to fit within this angle of pixels were set to 0. This operation was applied for a total of four different angular sections from a randomly selected region of the original input image, where the frequency magnitude of the extracted angular regions was computed and compared.

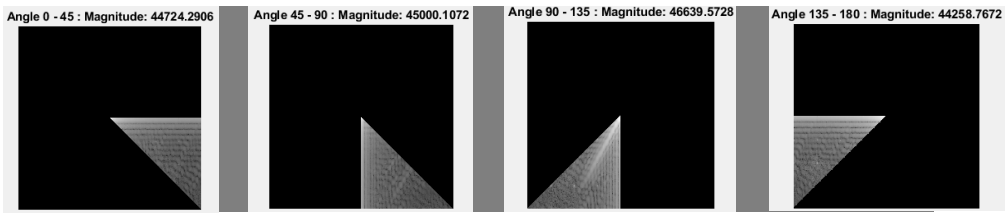

Figure 6. Example outputs when extracting frequency features from a range of different angles.

To surmise, the magnitude of frequencies can be observed to increase as the radius value is increased, which in turn infers that an increased count of frequencies is incorporated as a larger circular mask is incorporated. Similarly, it can be argued that the magnitude of pixels magnitude of frequencies between an angle of 90˚ and 135˚ are notably higher than when compared with the alternative regions, this can also be observed as a line of the main peaks within the Fourier transform can be observed within the pixel representation of the Fourier transform, effectively a white diagonal line of pixel intensity values can be observed, therefore showing that a line of peak frequencies can be located within this angle of frequency values which as a result would produce a higher magnitude in comparison to other angles.

Table 1. 1st order grey level features extracted from the image histogram.

Mean	Variance	Skewness	Kurtosis
32.458	2.091e+15	2.150	7.993

However, the texture of an image region can be determined based on the grey level distribution throughout the entirety of the full image of pixels, the features of which can be used to analyse the properties of an image through the spatial domain of the image. First-order statistics can be derived from a 1st order histogram of the input image, where the grey level distribution can be observed through various computed moments; namely mean, variance, skewness and kurtosis.

To surmise, the mean describes average intensity value throughout the numerous grey level intensity values whereas the second central moment variance compares each intensity value with the previously calculated mean intensity. This in turn showed that the average frequency intensity value can be argued as being dark with an intensity value of 32.458, though the variance of which was indicative that the grey levels are largely spread out away from the mean intensity and therefore showing that a wide range of grey levels are distributed throughout the input image. Additionally, the skewness statistic evaluates the amount of symmetry that exists with the intensity probability distribution relative to the previously estimated mean and the kurtosis assesses the flatness of the intensity probability distribution. To this end, the positive skewness value of 2.150 and a positive kurtosis value of 7.993 indicated that the grey level distributions localised within the histogram are skewed right with a "heavy-tailed" distribution.

Table 2. Features extracted from co-occurrence matrix based on the input image.

	256-bit	**32-bit**	**16-bit**	**8-bit**
Contrast	5.123	0.109	0.038	0.018
Correlation	0.999	0.998	0.998	0.994
Homogeneity	0.885	0.970	0.985	0.991
Entropy	4.479	2.231	1.592	1.139
Energy	0.087	0.269	0.395	0.502
Sum of Average	66.933	9.139	5.290	3.355

As elaborated by Aggarwal and Agrawal (2012), while the 1st order statistics revealed features relating to the grey level probability distributions, they provided no insight regarding position proximity correlations that could exist between the grey levels within the input image. As such, six additional features were extracted through the computation of a grey-level co-occurrence matrix (GLCM), where instead the quantification of features such as the coarseness, smoothness and similar texture features. Co-occurrence matrices were computed with offset values of 0˚ and a distance of 1 pixel. The contrast feature is indicative of the local level variations within a region of the input image while entropy assessed the amount of randomness that existed throughout the grey-level distribution. To this end, this showed that a greater amount of textural sharpness and grey level complexity within the texture of the input image existed when a higher bit-depth was used. Furthermore, the homogeneity feature is a measure which assesses the closeness of the grey level distribution and the energy feature measures the uniformity of the grey level distribution, which as a result demonstrated that the textural grey level features could be observed as utilising similar grey levels throughout all bit-depth variations but incorporated a higher amount of uniformity as a lower bit depth was used. On the other hand, correlation is a feature which measures the amount of consistency of the image texture that exists between pixels in two different directions, however this showed that the textural consistency remained constant despite applying different bit-depths of the input image.

Task 3: Object Tracking

The third and final task in this assignment encompassed trajectory estimation based on a dataset which was generated by a generic video detector, where the coordinate position of an object was detected and saved between each frame of a video. Specifically, the dataset provided included real coordinate files of an object; 'x.csv' and 'y.csv', and the noisy coordinates captured by the video detector; 'a.csv' and 'b.csv'. Upon observing the initial plot for the noisy coordinates relative to the real coordinates, it can be observed that the trajectory derived from the noisy coordinates is unreliable and irrespective to the true positions of the object being tracking by the generic video detector. As such, a Kalman filter can be applied, where a new estimated trajectory can be calculated based on the noisy coordinates recorded to produce an overall more accurate and reliable projection solution. The Kalman filter features a total of two imperative steps; prediction and correctional update. Fundamentally, this filter uses the data stored regarding the previous state and uses it to infer knowledge about a future state. As such, starting with an assumed initial state estimate and an initial state error covariance matrix, a new state vector and error covariance matrix are first predicted given the state at the previous time step. However, the subsequent update step ultimately estimates and determines the current state of the system provided the position measurements that was supplied at that time step.

Within the update step, an estimated innovation covariance matrix was calculated using the information from the previous time step, whereby this matrix would entail insight regarding the variability of the coordinate measurements. For example, a higher variability can be indicative that the observed measurements change frequently, therefore the amount of confidence that could be portrayed with these measurements would be low. In contrast, a lower variability can be indicative that the measurements are precise and reliable, therefore the amount of confidence would be higher. Similarly, the estimated state covariance matrix provides an assessment on how much a given state is estimated to change. Overall, the covariance matrices can be utilised for a Kalman gain computation to provide an assessment on the amount of change which should effectively be brought upon the current estimate, depending on how much confidence can be observed with the current estimate. Furthermore, a higher amount of manipulation of the current estimate can be expected when the variability is higher with either of the covariance matrices, therefore the Kalman gain value would in turn be higher.

As such, the update step would assign new covariance and state estimations of the coordinate measurements in the current observation vector, given the information known from the previous time step. As a result, this cycle of operations would be repeated and applied when operating on the subsequent observation vector, using previously estimated state and state covariance matrices as reference when predicting and updating the estimate in the next time step until the entirety of the input observation vector has been processed by the Kalman filter. In conclusion, the resulting output from the Kalman filter should present the new estimated velocity vectors based on the input noisy velocity vectors, where the estimated velocities should present an improved and more reliable representation of the object velocity tracking.

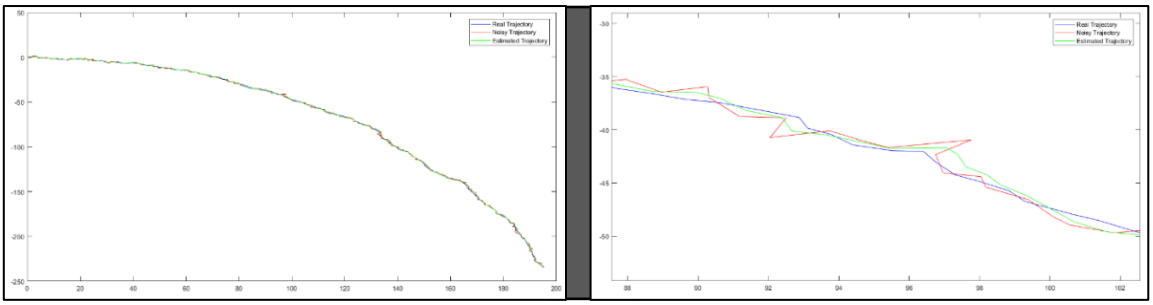

Figure 7. Example of Kalman filter trajectory estimated, full view and a zoomed view respectively.

Overall, the captured noisy trajectory of the object being tracked can be observed as being unstable in segments, deviating away intermittently in comparison to the real trajectory. However, as previously elaborated, the application of the Kalman filter procures a new estimated trajectory which improves upon the original noisy trajectory, providing a smoother trajectory plot which is arguably more representative of the real trajectory of the object being tracked.

Table 3. Error metrics estimated for measuring the performance of the Kalman filter.

MAE	Standard Deviation	RMSE
0.620	0.318	0.574

The performance of the estimated trajectory projected through the Kalman filter was assessed further by computing several different error metrics by comparing the estimated coordinates with the original noisy coordinates. Mean absolute error (MAE) and the root mean squared error (RMSE) measured the average magnitude of all absolute errors whereas the standard deviation of the absolute errors measured the overall spread of the estimated error values. Overall, the MAE and RMSE produced an error value of approximately 0.620 and 0.574 respectively, this low error therefore can be argued to have demonstrated an overall positive performance of the Kalman filter and in turn showing an accurate tracking performance when tracking the object trajectory over time. However, the low estimated standard deviation of the absolute error with a value of 0.318 also demonstrated that the estimated error values accumulate near the estimated mean absolute error, which in turn showed the robustness and reliability of the object tracking estimations that are projected by the Kalman filter.

Reference List

Aggarwal, N., Agrawal, R.K. (2012) *First and Second Order Statistics Features for Classification of Magnetic Resonance Brain Images*. Journal of Signal and Information Processing, 3(2) 146-153. Available from: http://file.scirp.org/Html/2-3400180_19553.htm [accessed: 22-04-2018].

Bellotto, N., Hu, H. (2009) *Simultaneous People Tracking and Recognition with a Service Robot: A novel approach using sensor fusion and Bayesian estimation*. Germany: VDM Verlag.

Farhan, M.A., Swarup, K.S. (2016) *Mathematical morphology-based islanding detection for distributed generation.* IET Generation, Transmission and Distribution, 10 (2) 518-526. Available from: https://ieeexplore.ieee.org/document/7407695/ [accessed: 27-04-2018].

Gonzalez, R.C., Woods, R.E., Eddins, S.L. (2013) *Digital Image Processing using MATLAB*, 2nd edition. Haryana: McGraw Hill Education (India).

Guan, Y., Jiang, W.L., Zhang, B., Chen, Y., Huang, X., Zhang, J., Liu, S., He, J., Zhou, Z., Ge, Y. (2017) *Value of whole-lesion apparent diffusion coefficient (ADC) first-order statistics and texture features in clinical staging of cervical cancers*. Elsevier Clinical Radiology, 72(11) 951-958. Available from: https://www-sciencedirect-com.proxy.library.lincoln.ac.uk/science/article/pii/S0009926017303458?_rdoc=1&_fmt=high&_origin=gateway&_docanchor=&md5=b8429449ccfc9c30159a5f9aeaa92ffb&ccp=y [accessed: 26-04-2018].

Haralick, R.M., Shanmugam,K., Dinstein, I. (1973) *Textural Features for Image Classification*. IEEE Transactions on Systems, Man, and Cybernetics, SMC-3(6) 610-621. Available from: https://ieeexplore.ieee.org/document/4309314/ [accessed: 20-04-2018].

Otsu, N. (1979) *A Threshold Selection Method from Gray-Level Histograms*. IEEE Transactions on Systems, Man, and Cybernetics, 9 (1) 62-66. Available from: http://web-ext.uaizu.ac.jp/course/bmclass/documents/otsu1979.pdf [accessed: 25-04-2018].

Shapiro, L.G., Stockman, G.C. (2002) *Computer Vision*. United States of America: Prentice Hall.

Parker, J.R. *Algorithms for Image Processing and Computer Vision.* (1997) United States of America: John Wiley and Sons.

Uppuluri, A. (2008) *GLCM Texture Features*. Available from: https://uk.mathworks.com/matlabcentral/fileexchange/22187-glcm-texture-features [accessed: 20-04-2018].

Appendix A. Task 1 MATLAB Script

```
% CMP9135M - Computer Vision - Assessment Item 1 - 12421031 - Peter Hart

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 1: Image Segmentation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all; clear; clc; % Reset environment

currentDir = pwd;

% Dialog box for file selection (filter = .jpg,.png)
[fileNames, pathName, filterIndex] = uigetfile({'*.jpg;*.png;*.bmp;','All
Image Files';'*.*','All Files'},'Select Input Images for Superpixel
segmentation','MultiSelect', 'on');

% Check if only one file is selected
if ~iscell(fileNames)
    fileNames = {fileNames}; % If only one file is selected, ensure the
file name is cell and not character
end

for fileid=1:length(fileNames)
    selectedFile = strcat(pathName,char(fileNames(fileid)));
    im1 = imread(selectedFile);
    im = imgaussfilt(im1);
    figure;
    histogram(im);
    % Get tray segmentation
    im_histeq = histeq(im);
    figure;
    histogram(im_histeq);
    level = graythresh(im_histeq);
    BW = imbinarize(im_histeq,level);
    se = strel('square',8);
    BW_open = imopen(BW,se);

    rprops = regionprops(BW_open,'BoundingBox'); %Establish a bounding box
    bbox = rprops.BoundingBox; %surround superpixel with bounding box

    imTrayCropped = imcrop(im, bbox); %crop the original image based on
this mask

    % Get label segmentation

    im2 = imTrayCropped;

    level = graythresh(im2);
    BW2 = imbinarize(im2,level);
    se1 = strel('disk',19);
    se2 = strel('disk',28);
    BW2_open = imerode(BW2,se1);
    BW2_open = imdilate(BW2_open,se2);

    L = bwlabel(BW2_open);
    highestArea = 0;
    M = zeros(size(BW2_open));
    label = zeros(size(M));
```

```
    for i=1:max(max(L))
        M = zeros(size(BW2_open));
        M = L == i;

        rprops = regionprops(M,'BoundingBox'); %Establish a bounding box
        bbox = rprops.BoundingBox; %surround superpixel with bounding box

        width = bbox(3); height = bbox(4);
        area = width * height;
        if area > highestArea
            highestArea = area;
            label = M;
            labelbbox = bbox;
        end
    end

    imLabelCropped = imcrop(imTrayCropped, labelbbox); %crop the original
image based on this mask


    %output results
    figure;
    subplot(2,2,1); imshow(im); title('Original');
    subplot(2,2,2); imshow(BW_open); title('Binary Segmentation Mask');
    subplot(2,2,3); imshow(imTrayCropped); title('Segmented Tray');
    subplot(2,2,4); imshow(imLabelCropped); title('Segmented Label');
end
% end of script
```

Appendix B. Task 2 MATLAB Script

```
% CMP9135M - Computer Vision - Assessment Item 1 - 12421031 - Peter Hart

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 2: Feature Calculation %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

close all; clear; clc; % Reset environment

% Load image
currentDir = pwd; fileDir = '\dataset\ImgPIA.jpeg';
selectedFile = strcat(currentDir,fileDir);

% Image Preprocessing
input_im_256 = imread(selectedFile);
input_im_256 = rgb2gray(input_im_256);

figure;imshow(input_im_256);
rect = getrect;
im = imcrop(input_im_256,rect);
close;

% Begin operations
figure;
subplot(2,3,1);
imshow(im,[]);
title('Original Selection');

% Compute Fourier Transform
F = fft2(im,256,256);
subplot(2,3,2); imshow(F); title('Fourier Transform');

% Center FFT
F = fftshift(F);
subplot(2,3,3); imshow(F); title('Centred Fourier Transform');

% Measure the minimum and maximum value of the transform amplitude
subplot(2,3,4); imshow(abs(F),[0 100]); colormap(jet); colorbar; title('Min
and Max of the Transform Amplitude');

% Logarithm amplitude
logF = log(1+abs(F));
subplot(2,3,5); imagesc(logF); colormap(jet); colorbar; title('Logarithmic
Transform');

% Phases
angularF = angle(F);
subplot(2,3,6); imshow(angularF,[-pi,pi]); colormap(jet); colorbar;
title('Angular Phases');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Task 2 : Part 1

figure;
n = 1; r = 100; angleInterval = 45;
for i=0:angleInterval:135
```

```
    FMasked = logF;
    angularMask = getAngleMask(logF,i,i+angleInterval);
    FMasked(~angularMask) = 0;
    total = sum(sum(FMasked));
    subplot(2,2,n); imshow(FMasked,[]); title(strcat("Angle ", num2str(i),
"? - ", num2str(i+angleInterval), "? : Magnitude: ", num2str(total))); axis
equal; grid on;
    n = n + 1;
end

figure;
n = 1; distanceInterval = 25;
for i=25:distanceInterval:100
    FMasked = logF;
    radiusMask = getRadiusMask(logF,i);
    FMasked(~radiusMask) = 0;
    total = sum(sum(FMasked));
    subplot(2,2,n); imshow(FMasked,[]); title(strcat("Radius ",
num2str(i)," : Magnitude: ", num2str(total))); axis equal; grid on;
    n = n + 1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Task 2 : Part 2

[pixelCounts, GLs] = imhist(input_im_256);

% Get the number of pixels in the histogram.
numberOfPixels = sum(pixelCounts);

% Get the mean gray lavel.
meanGL = sum(GLs .* pixelCounts) / numberOfPixels;
% Get the variance, which is the second central moment.
varianceGL = sum((GLs - meanGL) .^ 2 .* pixelCounts) / (numberOfPixels-1);
% Get the standard deviation.
stdDev = sqrt(varianceGL);
% Get the skew.
skew = sum((GLs - meanGL) .^ 3 .* pixelCounts) / ((numberOfPixels - 1) *
stdDev^3);
% Get the kurtosis.
kurtosis = sum((GLs - meanGL) .^ 4 .* pixelCounts) / ((numberOfPixels - 1)
* stdDev^4);

GLCM256 = graycomatrix(input_im_256, 'offset', [0 1],'NumLevels', 256,
'Symmetric', true);
GLCM32 = graycomatrix(input_im_256, 'offset', [0 1],'NumLevels', 32,
'Symmetric', true);
GLCM16 = graycomatrix(input_im_256, 'offset', [0 1],'NumLevels', 16,
'Symmetric', true);
GLCM8 = graycomatrix(input_im_256, 'offset', [0 1],'NumLevels', 8,
'Symmetric', true);

stats_256 = GLCM_Features1(GLCM256,0);
stats_32 = GLCM_Features1(GLCM32,0);
stats_16 = GLCM_Features1(GLCM16,0);
stats_8 = GLCM_Features1(GLCM8,0);

function radiusMask = getRadiusMask(F,radius)
    radiusMask = false(size(F));
```

```
    totalRows = size(F,1); totalCols = size(F,2);
    centreX = round(totalRows / 2); centreY = round(totalCols/ 2);
    % Create circle based on input radius
    [imageSizeX, imageSizeY] = meshgrid(1:totalRows, 1:totalCols);
    radiusMask =  (imageSizeX - centreX).^2 + (imageSizeY- centreY).^2 <=
radius.^2;
end

function angularMask = getAngleMask(F,angMin,angMax)
    angularMask = false(size(F));
    totalRows = size(F,1); totalCols = size(F,2);
    centreX = round(totalCols/2); centreY = round(totalRows/2);
    angMin = deg2rad(angMin); angMax = deg2rad(angMax);
    for i=1:totalRows
        for j=1:totalCols
            ang = atan2(i-centreY,j-centreX);
            if ang > angMin && ang < angMax
                angularMask(i,j) = true;
            end
        end
    end
end

% end of script
```

Appendix C. GLCM Features MATLAB Function Utilised for task 2.

The following GLCM function was designed by Uppuluri (2008).

```
function [out] = GLCM_Features1(glcmin,pairs)
%
% GLCM_Features1 helps to calculate the features from the different GLCMs
% that are input to the function. The GLCMs are stored in a i x j x n
% matrix, where n is the number of GLCMs calculated usually due to the
% different orientation and displacements used in the algorithm. Usually
% the values i and j are equal to 'NumLevels' parameter of the GLCM
% computing function graycomatrix(). Note that matlab quantization values
% belong to the set {1,..., NumLevels} and not from {0,...,(NumLevels-1)}
% as provided in some references
% http://www.mathworks.com/access/helpdesk/help/toolbox/images/graycomatrix
% .html
%
% Although there is a function graycoprops() in Matlab Image Processing
% Toolbox that computes four parameters Contrast, Correlation, Energy,
% and Homogeneity. The paper by Haralick suggests a few more parameters
% that are also computed here. The code is not fully vectorized and hence
% is not an efficient implementation but it is easy to add new features
% based on the GLCM using this code. Takes care of 3 dimensional glcms
% (multiple glcms in a single 3D array)
%
% If you find that the values obtained are different from what you expect
% or if you think there is a different formula that needs to be used
% from the ones used in this code please let me know.
% A few questions which I have are listed in the link
% http://www.mathworks.com/matlabcentral/newsreader/view_thread/239608
%
% I plan to submit a vectorized version of the code later and provide
% updates based on replies to the above link and this initial code.
%
% Features computed
% Autocorrelation: [2]                        (out.autoc)
% Contrast: matlab/[1,2]                      (out.contr)
% Correlation: matlab                         (out.corrm)
% Correlation: [1,2]                          (out.corrp)
% Cluster Prominence: [2]                     (out.cprom)
% Cluster Shade: [2]                          (out.cshad)
% Dissimilarity: [2]                          (out.dissi)
% Energy: matlab / [1,2]                      (out.energ)
% Entropy: [2]                                (out.entro)
% Homogeneity: matlab                         (out.homom)
% Homogeneity: [2]                            (out.homop)
% Maximum probability: [2]                    (out.maxpr)
% Sum of sqaures: Variance [1]                (out.sosvh)
% Sum average [1]                             (out.savgh)
% Sum variance [1]                            (out.svarh)
% Sum entropy [1]                             (out.senth)
% Difference variance [1]                     (out.dvarh)
% Difference entropy [1]                      (out.denth)
% Information measure of correlation1 [1]     (out.inf1h)
% Informaiton measure of correlation2 [1]     (out.inf2h)
% Inverse difference (INV) is homom [3]       (out.homom)
% Inverse difference normalized (INN) [3]     (out.indnc)
% Inverse difference moment normalized [3]    (out.idmnc)
%
% The maximal correlation coefficient was not calculated due to
```

```
% computational instability
% http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
%
% Formulae from MATLAB site (some look different from
% the paper by Haralick but are equivalent and give same results)
% Example formulae:
% Contrast = sum_i(sum_j(  (i-j)^2 * p(i,j) ) ) (same in matlab/paper)
% Correlation = sum_i( sum_j( (i - u_i)(j - u_j)p(i,j)/(s_i.s_j) ) ) (m)
% Correlation = sum_i( sum_j( ((ij)p(i,j) - u_x.u_y) / (s_x.s_y) ) ) (p[2])
% Energy = sum_i( sum_j( p(i,j)^2 ) )           (same in matlab/paper)
% Homogeneity = sum_i( sum_j( p(i,j) / (1 + |i-j|) ) ) (as in matlab)
% Homogeneity = sum_i( sum_j( p(i,j) / (1 + (i-j)^2) ) ) (as in paper)
%
% Where:
% u_i = u_x = sum_i( sum_j( i.p(i,j) ) ) (in paper [2])
% u_j = u_y = sum_i( sum_j( j.p(i,j) ) ) (in paper [2])
% s_i = s_x = sum_i( sum_j( (i - u_x)^2.p(i,j) ) ) (in paper [2])
% s_j = s_y = sum_i( sum_j( (j - u_y)^2.p(i,j) ) ) (in paper [2])
%
%
% Normalize the glcm:
% Compute the sum of all the values in each glcm in the array and divide
% each element by it sum
%
% Haralick uses 'Symmetric' = true in computing the glcm
% There is no Symmetric flag in the Matlab version I use hence
% I add the diagonally opposite pairs to obtain the Haralick glcm
% Here it is assumed that the diagonally opposite orientations are paired
% one after the other in the matrix
% If the above assumption is true with respect to the input glcm then
% setting the flag 'pairs' to 1 will compute the final glcms that would
result
% by setting 'Symmetric' to true. If your glcm is computed using the
% Matlab version with 'Symmetric' flag you can set the flag 'pairs' to 0
%
% References:
% 1. R. M. Haralick, K. Shanmugam, and I. Dinstein, Textural Features of
% Image Classification, IEEE Transactions on Systems, Man and Cybernetics,
% vol. SMC-3, no. 6, Nov. 1973
% 2. L. Soh and C. Tsatsoulis, Texture Analysis of SAR Sea Ice Imagery
% Using Gray Level Co-Occurrence Matrices, IEEE Transactions on Geoscience
% and Remote Sensing, vol. 37, no. 2, March 1999.
% 3. D A. Clausi, An analysis of co-occurrence texture statistics as a
% function of grey level quantization, Can. J. Remote Sensing, vol. 28, no.
% 1, pp. 45-62, 2002
% 4. http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
%
%
% Example:
%
% Usage is similar to graycoprops() but needs extra parameter 'pairs' apart
% from the GLCM as input
% I = imread('circuit.tif');
% GLCM2 = graycomatrix(I,'Offset',[2 0;0 2]);
% stats = GLCM_features1(GLCM2,0)
% The output is a structure containing all the parameters for the different
% GLCMs
%
% [Avinash Uppuluri: avinash_uv@yahoo.com: Last modified: 11/20/08]

% If 'pairs' not entered: set pairs to 0
```

```
if ((nargin > 2) || (nargin == 0))
   error('Too many or too few input arguments. Enter GLCM and pairs.');
elseif ( (nargin == 2) )
    if ((size(glcmin,1) <= 1) || (size(glcmin,2) <= 1))
       error('The GLCM should be a 2-D or 3-D matrix.');
    elseif ( size(glcmin,1) ~= size(glcmin,2) )
        error('Each GLCM should be square with NumLevels rows and NumLevels
cols');
    end
elseif (nargin == 1) % only GLCM is entered
    pairs = 0; % default is numbers and input 1 for percentage
    if ((size(glcmin,1) <= 1) || (size(glcmin,2) <= 1))
       error('The GLCM should be a 2-D or 3-D matrix.');
    elseif ( size(glcmin,1) ~= size(glcmin,2) )
       error('Each GLCM should be square with NumLevels rows and NumLevels
cols');
    end
end


format long e
if (pairs == 1)
    newn = 1;
    for nglcm = 1:2:size(glcmin,3)
        glcm(:,:,newn)  = glcmin(:,:,nglcm) + glcmin(:,:,nglcm+1);
        newn = newn + 1;
    end
elseif (pairs == 0)
    glcm = glcmin;
end

size_glcm_1 = size(glcm,1);
size_glcm_2 = size(glcm,2);
size_glcm_3 = size(glcm,3);

% checked
out.autoc = zeros(1,size_glcm_3); % Autocorrelation: [2]
out.contr = zeros(1,size_glcm_3); % Contrast: matlab/[1,2]
out.corrm = zeros(1,size_glcm_3); % Correlation: matlab
out.corrp = zeros(1,size_glcm_3); % Correlation: [1,2]
out.cprom = zeros(1,size_glcm_3); % Cluster Prominence: [2]
out.cshad = zeros(1,size_glcm_3); % Cluster Shade: [2]
out.dissi = zeros(1,size_glcm_3); % Dissimilarity: [2]
out.energ = zeros(1,size_glcm_3); % Energy: matlab / [1,2]
out.entro = zeros(1,size_glcm_3); % Entropy: [2]
out.homom = zeros(1,size_glcm_3); % Homogeneity: matlab
out.homop = zeros(1,size_glcm_3); % Homogeneity: [2]
out.maxpr = zeros(1,size_glcm_3); % Maximum probability: [2]

out.sosvh = zeros(1,size_glcm_3); % Sum of sqaures: Variance [1]
out.savgh = zeros(1,size_glcm_3); % Sum average [1]
out.svarh = zeros(1,size_glcm_3); % Sum variance [1]
out.senth = zeros(1,size_glcm_3); % Sum entropy [1]
out.dvarh = zeros(1,size_glcm_3); % Difference variance [4]
%out.dvarh2 = zeros(1,size_glcm_3); % Difference variance [1]
out.denth = zeros(1,size_glcm_3); % Difference entropy [1]
out.inf1h = zeros(1,size_glcm_3); % Information measure of correlation1 [1]
out.inf2h = zeros(1,size_glcm_3); % Informaiton measure of correlation2 [1]
%out.mxcch = zeros(1,size_glcm_3);% maximal correlation coefficient [1]
%out.invdc = zeros(1,size_glcm_3);% Inverse difference (INV) is homom [3]
```

```
out.indnc = zeros(1,size_glcm_3); % Inverse difference normalized (INN) [3]
out.idmnc = zeros(1,size_glcm_3); % Inverse difference moment normalized
[3]

% correlation with alternate definition of u and s
%out.corrm2 = zeros(1,size_glcm_3); % Correlation: matlab
%out.corrp2 = zeros(1,size_glcm_3); % Correlation: [1,2]

glcm_sum  = zeros(size_glcm_3,1);
glcm_mean = zeros(size_glcm_3,1);
glcm_var  = zeros(size_glcm_3,1);

% http://www.fp.ucalgary.ca/mhallbey/glcm_mean.htm confuses the range of
% i and j used in calculating the means and standard deviations.
% As of now I am not sure if the range of i and j should be [1:Ng] or
% [0:Ng-1]. I am working on obtaining the values of mean and std that get
% the values of correlation that are provided by matlab.
u_x = zeros(size_glcm_3,1);
u_y = zeros(size_glcm_3,1);
s_x = zeros(size_glcm_3,1);
s_y = zeros(size_glcm_3,1);

% % alternate values of u and s
% u_x2 = zeros(size_glcm_3,1);
% u_y2 = zeros(size_glcm_3,1);
% s_x2 = zeros(size_glcm_3,1);
% s_y2 = zeros(size_glcm_3,1);

% checked p_x p_y p_xplusy p_xminusy
p_x = zeros(size_glcm_1,size_glcm_3); % Ng x #glcms[1]
p_y = zeros(size_glcm_2,size_glcm_3); % Ng x #glcms[1]
p_xplusy = zeros((size_glcm_1*2 - 1),size_glcm_3); %[1]
p_xminusy = zeros((size_glcm_1),size_glcm_3); %[1]
% checked hxy hxy1 hxy2 hx hy
hxy  = zeros(size_glcm_3,1);
hxy1 = zeros(size_glcm_3,1);
hx   = zeros(size_glcm_3,1);
hy   = zeros(size_glcm_3,1);
hxy2 = zeros(size_glcm_3,1);

%Q    = zeros(size(glcm));

for k = 1:size_glcm_3 % number glcms

    glcm_sum(k) = sum(sum(glcm(:,:,k)));
    glcm(:,:,k) = glcm(:,:,k)./glcm_sum(k); % Normalize each glcm
    glcm_mean(k) = mean2(glcm(:,:,k)); % compute mean after norm
    glcm_var(k)  = (std2(glcm(:,:,k)))^2;

    for i = 1:size_glcm_1

        for j = 1:size_glcm_2

            out.contr(k) = out.contr(k) + (abs(i - j))^2.*glcm(i,j,k);
            out.dissi(k) = out.dissi(k) + (abs(i - j)*glcm(i,j,k));
            out.energ(k) = out.energ(k) + (glcm(i,j,k).^2);
            out.entro(k) = out.entro(k) - (glcm(i,j,k)*log(glcm(i,j,k) +
eps));
            out.homom(k) = out.homom(k) + (glcm(i,j,k)/( 1 + abs(i-j) ));
```

```
            out.homop(k) = out.homop(k) + (glcm(i,j,k)/( 1 + (i - j)^2));
            % [1] explains sum of squares variance with a mean value;
            % the exact definition for mean has not been provided in
            % the reference: I use the mean of the entire normalized glcm
            out.sosvh(k) = out.sosvh(k) + glcm(i,j,k)*((i -
glcm_mean(k))^2);

            %out.invdc(k) = out.homom(k);
            out.indnc(k) = out.indnc(k) + (glcm(i,j,k)/( 1 + (abs(i-
j)/size_glcm_1) ));
            out.idmnc(k) = out.idmnc(k) + (glcm(i,j,k)/( 1 + ((i -
j)/size_glcm_1)^2));
            u_x(k)          = u_x(k) + (i)*glcm(i,j,k); % changed 10/26/08
            u_y(k)          = u_y(k) + (j)*glcm(i,j,k); % changed 10/26/08
            % code requires that Nx = Ny
            % the values of the grey levels range from 1 to (Ng)
        end

    end
    out.maxpr(k) = max(max(glcm(:,:,k)));
end
% glcms have been normalized:
% The contrast has been computed for each glcm in the 3D matrix
% (tested) gives similar results to the matlab function

for k = 1:size_glcm_3

    for i = 1:size_glcm_1

        for j = 1:size_glcm_2
            p_x(i,k) = p_x(i,k) + glcm(i,j,k);
            p_y(i,k) = p_y(i,k) + glcm(j,i,k); % taking i for j and j for i
            if (ismember((i + j),[2:2*size_glcm_1]))
                p_xplusy((i+j)-1,k) = p_xplusy((i+j)-1,k) + glcm(i,j,k);
            end
            if (ismember(abs(i-j),[0:(size_glcm_1-1)]))
                p_xminusy((abs(i-j))+1,k) = p_xminusy((abs(i-j))+1,k) +...
                    glcm(i,j,k);
            end
        end
    end

%     % consider u_x and u_y and s_x and s_y as means and standard
deviations
%     % of p_x and p_y
%     u_x2(k) = mean(p_x(:,k));
%     u_y2(k) = mean(p_y(:,k));
%     s_x2(k) = std(p_x(:,k));
%     s_y2(k) = std(p_y(:,k));

end

% marginal probabilities are now available [1]
% p_xminusy has +1 in index for matlab (no 0 index)
% computing sum average, sum variance and sum entropy:
for k = 1:(size_glcm_3)

    for i = 1:(2*(size_glcm_1)-1)
        out.savgh(k) = out.savgh(k) + (i+1)*p_xplusy(i,k);
```

```
        % the summation for savgh is for i from 2 to 2*Ng hence (i+1)
        out.senth(k) = out.senth(k) - (p_xplusy(i,k)*log(p_xplusy(i,k) +
eps));
    end

end
% compute sum variance with the help of sum entropy
for k = 1:(size_glcm_3)

    for i = 1:(2*(size_glcm_1)-1)
        out.svarh(k) = out.svarh(k) + (((i+1) -
out.senth(k))^2)*p_xplusy(i,k);
        % the summation for savgh is for i from 2 to 2*Ng hence (i+1)
    end

end
% compute difference variance, difference entropy,
for k = 1:size_glcm_3
% out.dvarh2(k) = var(p_xminusy(:,k));
% but using the formula in
% http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
% we have for dvarh
    for i = 0:(size_glcm_1-1)
        out.denth(k) = out.denth(k) -
(p_xminusy(i+1,k)*log(p_xminusy(i+1,k) + eps));
        out.dvarh(k) = out.dvarh(k) + (i^2)*p_xminusy(i+1,k);
    end
end

% compute information measure of correlation(1,2) [1]
for k = 1:size_glcm_3
    hxy(k) = out.entro(k);
    for i = 1:size_glcm_1

        for j = 1:size_glcm_2
            hxy1(k) = hxy1(k) - (glcm(i,j,k)*log(p_x(i,k)*p_y(j,k) + eps));
            hxy2(k) = hxy2(k) - (p_x(i,k)*p_y(j,k)*log(p_x(i,k)*p_y(j,k) +
eps));
%             for Qind = 1:(size_glcm_1)
%                 Q(i,j,k) = Q(i,j,k) +...
%                     ( glcm(i,Qind,k)*glcm(j,Qind,k) /
(p_x(i,k)*p_y(Qind,k)) );
%             end
        end
        hx(k) = hx(k) - (p_x(i,k)*log(p_x(i,k) + eps));
        hy(k) = hy(k) - (p_y(i,k)*log(p_y(i,k) + eps));
    end
    out.inf1h(k) = ( hxy(k) - hxy1(k) ) / ( max([hx(k),hy(k)]) );
    out.inf2h(k) = ( 1 - exp( -2*( hxy2(k) - hxy(k) ) ) )^0.5;
%     eig_Q(k,:)   = eig(Q(:,:,k));
%     sort_eig(k,:)= sort(eig_Q(k,:),'descend');
%     out.mxcch(k) = sort_eig(k,2)^0.5;
% The maximal correlation coefficient was not calculated due to
% computational instability
% http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
end

corm = zeros(size_glcm_3,1);
corp = zeros(size_glcm_3,1);
% using http://www.fp.ucalgary.ca/mhallbey/glcm_variance.htm for s_x s_y
```

```
for k = 1:size_glcm_3
    for i = 1:size_glcm_1
        for j = 1:size_glcm_2
            s_x(k)  = s_x(k)  + (((i) - u_x(k))^2)*glcm(i,j,k);
            s_y(k)  = s_y(k)  + (((j) - u_y(k))^2)*glcm(i,j,k);
            corp(k) = corp(k) + ((i)*(j)*glcm(i,j,k));
            corm(k) = corm(k) + (((i) - u_x(k))*((j) -
u_y(k))*glcm(i,j,k));
            out.cprom(k) = out.cprom(k) + (((i + j - u_x(k) -
u_y(k))^4)*...
                glcm(i,j,k));
            out.cshad(k) = out.cshad(k) + (((i + j - u_x(k) -
u_y(k))^3)*...
                glcm(i,j,k));
        end
    end
    % using http://www.fp.ucalgary.ca/mhallbey/glcm_variance.htm for s_x
    % s_y : This solves the difference in value of correlation and might be
    % the right value of standard deviations required
    % According to this website there is a typo in [2] which provides
    % values of variance instead of the standard deviation hence a square
    % root is required as done below:
    s_x(k) = s_x(k) ^ 0.5;
    s_y(k) = s_y(k) ^ 0.5;
    out.autoc(k) = corp(k);
    out.corrp(k) = (corp(k) - u_x(k)*u_y(k))/(s_x(k)*s_y(k));
    out.corrm(k) = corm(k) / (s_x(k)*s_y(k));
%     % alternate values of u and s
%     out.corrp2(k) = (corp(k) - u_x2(k)*u_y2(k))/(s_x2(k)*s_y2(k));
%     out.corrm2(k) = corm(k) / (s_x2(k)*s_y2(k));
end
% Here the formula in the paper out.corrp and the formula in matlab
% out.corrm are equivalent as confirmed by the similar results obtained

% % The papers have a slightly different formular for Contrast
% % I have tested here to find this formula in the papers provides the
% % same results as the formula provided by the matlab function for
% % Contrast (Hence this part has been commented)
% out.contrp = zeros(size_glcm_3,1);
% contp = 0;
% Ng = size_glcm_1;
% for k = 1:size_glcm_3
%     for n = 0:(Ng-1)
%         for i = 1:Ng
%             for j = 1:Ng
%                 if (abs(i-j) == n)
%                     contp = contp + glcm(i,j,k);
%                 end
%             end
%         end
%         out.contrp(k) = out.contrp(k) + n^2*contp;
%         contp = 0;
%     end
%
% end

%       GLCM Features (Soh, 1999; Haralick, 1973; Clausi 2002)
%           f1. Uniformity / Energy / Angular Second Moment (done)
%           f2. Entropy (done)
%           f3. Dissimilarity (done)
%           f4. Contrast / Inertia (done)
```

```
%            f5. Inverse difference
%            f6. correlation
%            f7. Homogeneity / Inverse difference moment
%            f8. Autocorrelation
%            f9. Cluster Shade
%           f10. Cluster Prominence
%           f11. Maximum probability
%           f12. Sum of Squares
%           f13. Sum Average
%           f14. Sum Variance
%           f15. Sum Entropy
%           f16. Difference variance
%           f17. Difference entropy
%           f18. Information measures of correlation (1)
%           f19. Information measures of correlation (2)
%           f20. Maximal correlation coefficient
%           f21. Inverse difference normalized (INN)
%           f22. Inverse difference moment normalized (IDN)
```

Appendix D. Task 3 MATLAB Script.

```
% CMP9135M - Computer Vision - Assessment Item 1 - 12421031 - Peter Hart

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Task 3: Object Tracking %
%%%%%%%%%%%%%%%%%%%%%%%%%%

close all; clear; clc; % Reset environment

currentDir = pwd; % Get the current directory
aDir = strcat(currentDir,'\dataset\a.csv'); % Save input file locations
bDir = strcat(currentDir,'\dataset\b.csv');
xDir = strcat(currentDir,'\dataset\x.csv');
yDir = strcat(currentDir,'\dataset\y.csv');

x = csvread(aDir);y = csvread(bDir); % Load noisy coordinates
xr = csvread(xDir);yr = csvread(yDir); % Load real coordinates

dt = 0.1;
z = [x;y];
totalFrames = length(x);

[px py] = kalmanTracking(z);

e = (x - px).^2 + (y - py).^2;
mean_e = mean(e);
std_e = std(e); %std(e) / sqrt(length(e));
rms = rms(e);%sqrt(mean_e).^2;

disp(strcat("Mean Error: ", num2str(mean_e)));
disp(strcat("Standard Error: ", num2str(std_e)));
disp(strcat("RMS Error: ", num2str(rms)));

% Plot the Trajectory
plot(xr,yr,'b-');
hold on;
plot(x,y,'r-');
plot(px,py,'g-');
legend('Real Trajectory','Noisy Trajectory','Estimated Trajectory');
hold off;


function [px py] = kalmanTracking(z)
    % Track a target with a Kalman filter
    % z: observation vector
    % Return the estimated state position coordinates (px,py)

    dt = 0.1; N = length(z); %  time interval; number of samples

    R = [0.3 0; % Observation noise
         0 0.3];
    H = [1 0 0 0; % Cartesian observation model
         0 0 1 0];
    F = [1 dt 0 0; % CV motion model
         0 1 0 0;
         0 0 1 dt;
         0 0 0 1];
    Q = [0.1 0 0 0; % Motion noise
```

```
          0 0.2 0 0;
          0 0 0.1 0;
          0 0 0 0.2];

    P = Q; % Initial state covariance
    x = [0 0 0 0]'; % Initial state
    s = zeros(4,N); % Current state

    for i = 1 : N
        [xp Pp] = kalmanPredict(x, P, F, Q);
        [x P] = kalmanUpdate(xp, Pp, H, R, z(:,i));
        s(:,i) = x; % Save current state
    end
    px = s(1,:);
    py = s(3,:); % Contain the velocities on x and y respectively
end

function [xp, Pp] = kalmanPredict(x, P, F, Q)
    % Prediction step of Kalman filter.
    % x: state vector
    % P: covariance matrix of x
    % F: matrix of motion model
    % Q: matrix of motion noise
    % Return predicted state vector xp and covariance Pp
    xp = F * x; % predict state
    Pp = F * P * F' + Q; % predict state covariance
end

function [xe Pe] = kalmanUpdate(x, P, H, R, z)
% Update step of Kalman filter.
    % x: state vector
    % P: covariance matrix of x
    % H: matrix of observation model
    % R: matrix of observation noise
    % z: observation vector
    % Return estimated state vector xe and covariance Pe
    S = H * P * H' + R; % innovation covariance
    K = P * H' * inv(S); % Kalman gain
    zp = H * x; % predicted observation
    xe = x + K * (z - zp); % estimated state
    Pe = P - K * S * K'; % estimated covariance
end

% end of script
```