4061/7060 IMAGE SENSORS & PROCESSING ASSIGNMENT 4 MAY 2019

(Assignment Value: 13%)

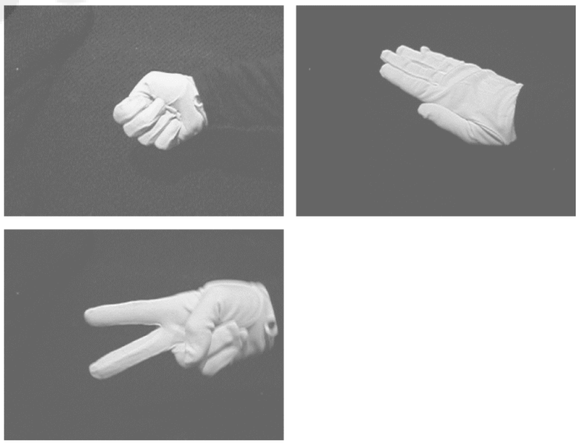
Overview

This assignment is intended to give you some hands-on experience with thresholding, morphological cleaning, feature estimation and classification as covered in lectures. In this assignment work you will be required to modify several functions which implement thresholding and clean up stages followed by shape feature estimation.

NOTE: Unlike previous assignments you can use **any** of the functions in the MATLAB image processing toolbox except ***graythresh*** to complete this assignment. Please stick to the basic MATLAB function set where possible and do not use functions from the machine vision and similar toolboxes.

Background

A client wants your company to develop an automated version of the children's game ROCK / PAPER / SCISSORS which uses a webcam to take images of the player's gloved hand. The component of the system you are working on is supposed to automatically classify the hand gesture from the webcam image. You have been given three sets of example images of rock paper and scissors gestures to classify. Examples of these images are given below.



Above: examples of 'rock' 'paper' and 'scissors' from the sample data.

In order to solve this problem, you will need to firstly threshold and filter out the region containing the hand gesture and then implement a feature extraction step stage that extracts shape estimates from the data. These will then be fed into a KNN classifier which you have been given.

Assignment Submission

Assignments are to be submitted via MyUni as a ZIP archive file containing the **four** key MATLAB functions and a Word or PDF document summarising your results with comments on the performance of the processing steps employed. Marks will be deducted for late submission.

Do NOT (I repeat DO NOT) include the images from the dataset in your submission. I will deduct marks if you do!

Source Materials

Source code, test functions and example imagery for this assignment are located on the MyUni website (https://www.myuni.adelaide.edu.au).

This code includes the following functions:

- **hand_threshold.m** - A function to be written by you to estimate a suitable isoththresh threshold for a given greyscale hand image. *Currently this file generates a random value.*
- **extract_hand.m** – A function to be written by you to extract the hand gesture region from the greyscale image. This function should use **hand_threshold.m** to provide an initial threshold value. *Currently this function merely uses the random threshold supplied by hand_threshold.m*
- *all_hand_regions.m* – a function that steps through the set of images and applies the function hand_extract.m. Running this command with the parameter 'full' will generate values for all 4 data sets.
- **hand_features.m** – A function to be written by you to extract shape features from a binary region containing a hand gesture as produced by extract_hand.m. *Currently this function produces a set of 5 random values.*
- *all_hand_features.m* – a function that runs through the images extracting the region of interest and then calls feature_extract.m to estimate the shape features you have developed. The result is a 2D array of feature vectors and a list of class labels which can be fed into all_hand_classify.m (see below). Running this command with the parameter 'full' will generate values for all 4 data sets.
- *knn_classify.m* – implements a very simple KNN classifier.
- *all_hand_classify.m* – a function that splits the supplied feature and class data in two and runs knn_classify.m to evaluate the ability of the features to classify the image data.
- *hu.m* – another feature method which you can try out for features for classifying the hand images based on the Hu moments.
- **hand_signature.m** – a function function to be written by you to estimate the boundary signature of a binary hand image obtained from *extract_hand.m*
- *signature_test.m* – a test function for hand_signature.m
- *signature_classfy.m* – a function which runs through the entire set of images and attempts to classify the boundary signature obtained from *hand_signature.m* as either rock paper or scissors.

The image data is contained in 3 sub-directories of around 150 images each which should give you plenty of examples to work with.

Exercise 4A – (3%) – Hand Region Extraction

STEP 1 – Modify the supplied function **t = hand_threshold(I)** such that it implements the simple isothresh thresholder. The thresholder should use an initial estimate of the overall average image intensity and apply the isothresh calculation for at least 10 iterations.

Recall that the isothresh calculation is defined as:

$$t_{k+1} = \frac{\bar{I}(I > t_k) + \bar{I}(I <= t_k)}{2}$$

Where $\bar{I}(I > t_k)$ and $\bar{I}(I <= t_k)$ represent the average intensity for values above and below the threshold value.

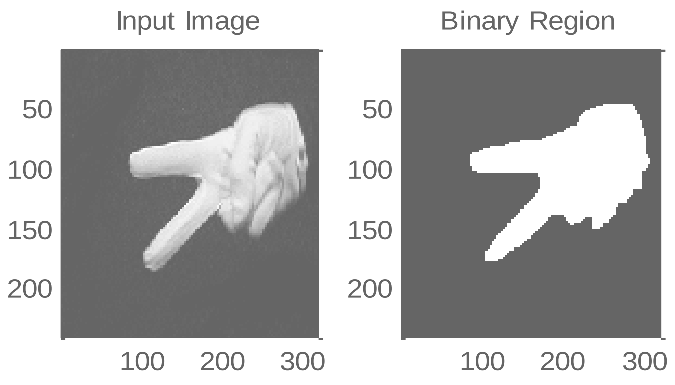
STEP 2 - Modify the supplied function ***B = extract_region(I)*** such that it thresholds the supplied image ***I*** based on hand_threshold() and returns a "cleaned up" binary image ***B*** representing the hand.

Thresholding should be based on the hand_threshold() estimate but may need to be rescaled (up or down) to deal with shadows in palm of the hand. Do NOT use **greythresh** from the image processing toolbox to do this.

After thresholding, the function should also remove artefacts such as point noise and merge fragmented regions together to form a good representation of the hand gesture. You will need to use morphological filters to achieve this. One problem you may also face is in dealing with shadows in the palm of the hand and fragmenting of the hand into more than one region. Refer the materials from week 7 for clues on how to minimise this. You may use combinations of functions such as **imerode, imdilate, imopen, imclose** and **imfill** etc from the image processing toolbox to assist here.

The resulting binary image should contain one connected region representing the hand to make it easy for your feature extraction processing. You may use **bwlabel** to assist with this step.

An example of this processing is given below (note: your own solution may not look exactly like this):



Illustrate your processing results using examples from the gesture images and explain how you came up with these processing steps in your assignment report. Make special note of any situations where the thresholding may be problematic. Also detail how you performed the cleanup steps in your solution.

You can use the supplied test script ***all_hand_regions*** to check your thresholding on the first 3 sets of test images.

Comment: *The quality of the thresholding and cleanup stage will have an impact on any features you estimate from the regions and hence the quality of the final classification performance. You may need to revisit this processing step once you have the classifier working.*

Hint: *Applying a small scale offset to the threshold estimated by* ***isothresh*** *(eg. "thresh_used = k * thresh_est;" may help to reduce problems with shadows seen in the palm of the hand for the "scissors" images.*

Exercise 4B – (4%) – Feature Classification

STEP 1- Using ideas covered in the feature estimation lectures write a series of functions to extract feature measurements from either the binary or grey-scale images returned by the ***hand_extract*** function.

Here the aim is to find a set of feature measures which vary between the "rock", "paper" and "scissors" images and could be used for classification. You can choose any reasonable feature measures you like, however this collection of routines must include two subroutines to estimate the ratio of boundary length to area, and roundness measures described in Lecture 10 (note: you can use regionprops() to help you obtain these values).

Given these individual feature estimates modify the wrapper function ***F=hand_features(B)*** which takes the binary images from ***hand_extract.m*** and returns a 1xN row vector of feature measurements. Here the row vector is a concatenation of the individual feature measures from your subroutines. For example, concatenating the ratio of boundary length to area, and circularity measures will produce a row vector of two values. Remember, ideally, your measures should be largely invariant to changes in position, rotation and scale of the object. If they are not then they will vary noticeably from image to image and make classification more difficult.

Suggestions:

- **regionprops** – this inbuilt MATLAB function can return numerous measures of a binary region such as area and perimeter. These can be used to construct several of the shape measures examined in lectures (see Week 10 notes for definitions).
- **hu** – a supplied function for calculating the Hu moments of a region (this implementation also includes an 8th moment sometimes used in Gait analysis).
- **fitellipse** – a simple function that fits an ellipse to the outline of an object.

It is suggested you experiment with simple shape measures derived from ***regionprops*** first as these are likely to provide the best classification features for this problem.

The function ***[F,C] = all_hand_features*** can be used to test your processing and creates a 2D array ***F*** of the feature estimates from the images in the first 3 datasets. The remaining value ***C*** is a list of the types for each vector (eg. class 1,2,3 etc).

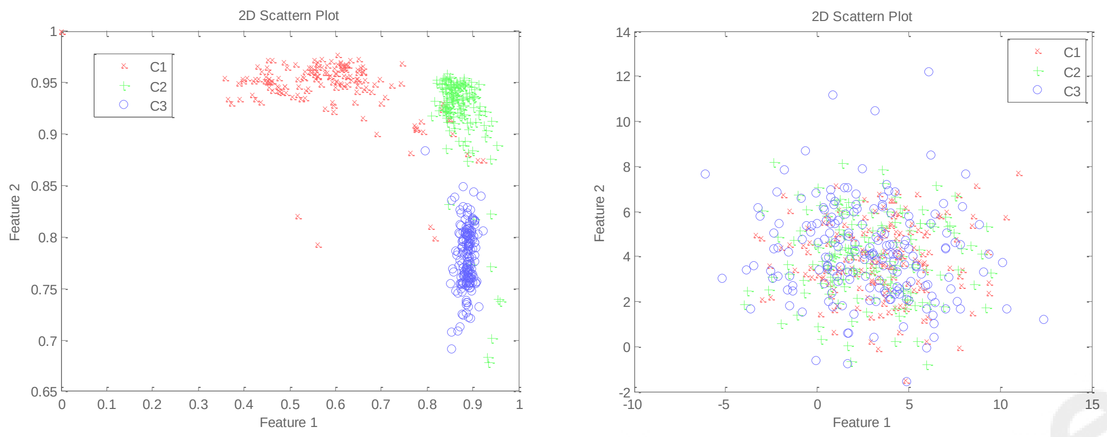
Illustrate your feature measure experiments using examples from the gesture images and write this up in the assignment report.

To try and determine which features might be useful in classification a class feature plot function **cplot.m** has also been included with the code. This function allows you to plot different combinations of features from your estimates as clusters of different coloured markers. As shown in lectures, features which are good at classifying shapes will produce separated clusters whilst poor filters will result in clusters which are overlapping.

Examples:

cplot(F(:,[2 4],C)	display a 2D scatterplot of the 2nd and 4th features
cplot(F(:,1:3),C)	display a 3D scatterplot of the first 3 features in F

Here F and C were created using the **all_hand_features.m** function



ABOVE: examples of very good (left) and very bad (right) clustering of feature data

Comments: *You can also try some of the other binary shape measures mentioned in week 8. There is no single 'magic feature' which will solve this classification problem. You may have to use combinations of different features to get good results. Simply using all the features, you can come up with (both good and bad) will not guarantee good classification either, so be selective. Use 2D and 3D scatter-plots of combinations of the features to guide you.*

STEP 2- Use the supplied function ***[F,C] = all_hand_features*** to construct an array of feature measurements ***F*** and class numbers ***C*** for the datasets of hand gesture imagery (classes 1,2, and 3 ***C*** are 'rock' 'paper' and 'scissors'). Plot various combinations of the feature measures for the "rock" "paper" and "scissors" data and then try to find a combination of features which clusters the feature data according to the type of image. This may require you to add or remove features steps in your ***hand_feature*** function.

Use the supplied function ***[C_predicted,C_true] = all_hand_classify(F,C,k)*** to perform a KNN classification analysis of your extracted features. This function splits the data in two and tests one half of the data using the other half (ie 50:50 test and training). The *k* term is the number of nearest neighbours used (eg. 3,5,7 etc). ***C_predicted*** contains the KNN class estimates and ***C_true*** the true class labels. For this work, only use data from the first 3 sets of gestures. Test with different values of *k*.

Create a summary table showing the number of correct and incorrect matches for each class based on the outputs of ***all_hand_classify***. Using the above tools, try to improve your classifiers performance. Use both the class results and plots of the raw features to guide your choice of features to use. In order to get better performance, you may need to use different feature types, or fine tune your region extraction processing. You will need to experiment with different combinations to get a solution which gives reasonable performance.

Write up your results including plots of any features which appear to separate out the 3 gestures. It is recommended that you make mention of any features you might have tried even if they didn't work.

Comments: *Don't expect to get perfect 100% correct classification. If you can get over 85% correct matching then you are doing reasonable well. Also you will need to write your own test code to plot the various features so you can see how well they cluster (hint try cplot()).*

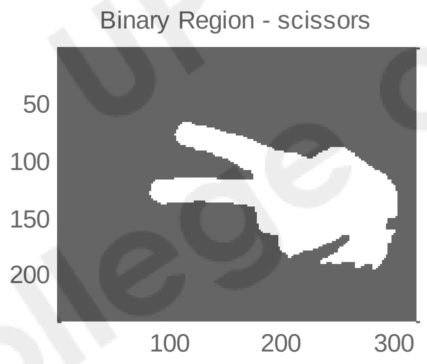
Exercise 4C - (4%) – Signature Classification

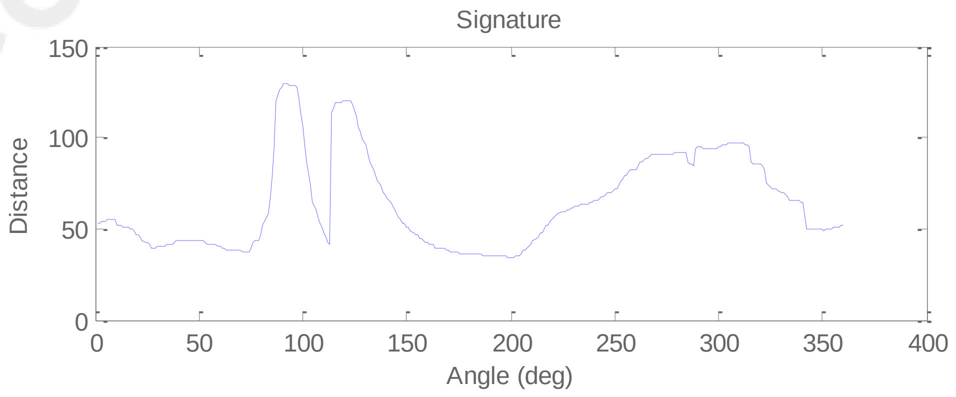
An alternative to the shape-based feature classifier approach shown earlier is to use a boundary descriptor such as the signature method presented in week 10. Boundary signature classification involves converting each object boundary into a 1D set of values representing the boundary contour as a function of distance and angle.

In this final part of the assignment you have been provided with a simple signature classifier **signature_classify.m** which compares three reference signatures created from the outline of the extracted hands to the signatures of each image in the 'rock' 'paper' 'scissors' databases. Here each signature to be matched is compared to the reference signatures using the sum of absolute differences. To cope with rotation this test is conducted for all possible circular shifts of the signature. Scale is also handled by normalising each signature relative to its mean value.

Your task is to write the missing function ***S = hand_signature(B);*** which estimates the signature S of the extracted hand shape B created by **hand_extract(I).** Here the signature is a representation of the maximum distance between the centroid and the pixels on the boundary for angles from 0 to 359 degrees.

An example of this is shown below for one of the 'scissor' images in the database:





Above: Example hand shape and its associated boundary signature.

The <u>suggested</u> algorithm for calculating the shape signature is shown on the next page.

Suggested Algorithm:

1. *Find the centroid of the hand and the coordinates of pixels on the hand boundary (this can be readily obtained using regionprops)*
2. *Subtract the centroid position (cx,cy) from the boundary pixel coordinates to give a list of points centred on (0,0).*
3. *For each rotation X from 0 to 359 degrees:*
   a. *Rotate the coordinates by angle X (eg. see expression below)*

      $x' = x\cos(angle) + y\sin(angle)$

      $y' = -x\sin(angle) + y\cos(angle)$

   b. *Find the edge points within approximately 2 pixels of the y-axis.*
   c. *Set the signature for angle X to be the maximum y value of the identified points. If the value is less than zero, then set it to zero.*

This algorithm essentially computes the distance from the reference point to the farthest edge of the object along the y axis and then rotates the object slightly and repeats the process etc…

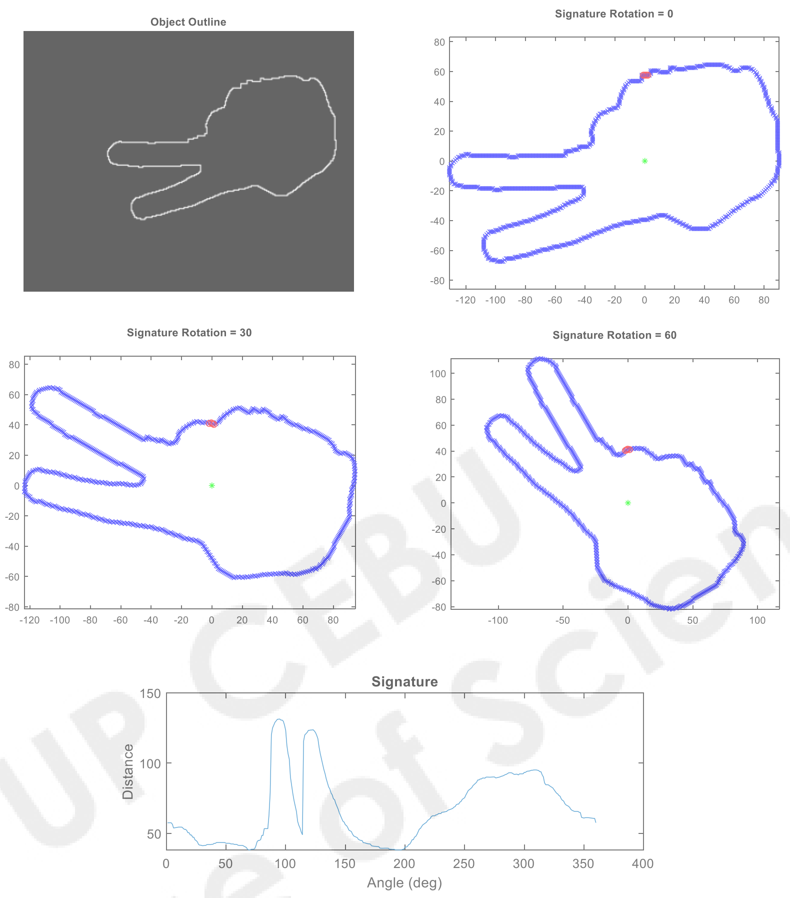
IMPORTANT: The resulting signature S should be a 1x360 element vector of numbers representing distances from the centroid to the boundary for each angle from 0 to 359 degrees.

STEP 1 – implement the above algorithm to construct a shape signature. This function can be tested using the script **signature_test.m.**

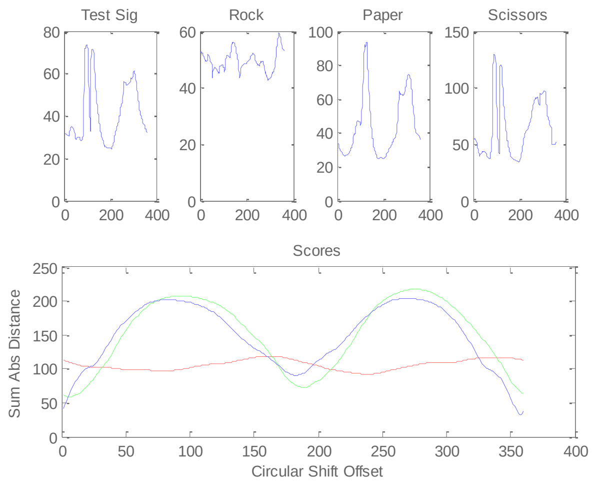
STEP 2 - Run the example classifier **signature_classify.m** and check the results. This script will construct 3 reference signatures and then go through each image in the database. A plot window will show the match scores at each iteration. If successful, you should be able to obtain a final match score above 80%. However, you may also need to further fine tune your *hand_extract.m* function.

As with parts 4A and 4B make sure you write up your results.

See illustrations on the next page.



Above: *an illustration of generating the shape signature by rotating the boundary pixels of the object through 0 to 359 degrees. The pixels in shown in red are the ones along the y-axis used to estimate the radial distance for each rotation angle (as per step 3b).*

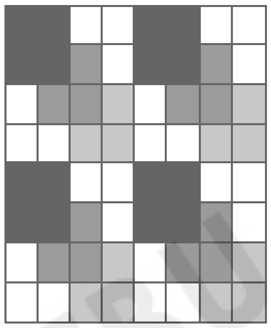


Above: *an example result plot for a 'scissor' image showing the current signature and the three reference signatures. The distance for each 1 degree shift of the signatures is shown at the bottom*

Exercise 4D - (2%) – Written Questions

Question 1 (1%)

The following represents a 4-colour texture to be analysed:



<u>By hand</u>, calculate the co-occurrence matrices for this texture for the offsets:

a) 2 pixels down

b) 1 pixel down and to the right

You may assume the above texture repeats at the boundaries. Assume black=0, and white=3 in the above illustration.

Include a description of how you derived your estimates.

Question 2 (1%)

The "thickness" of an object was also suggested in class as another type of metric for telling the differences between geometrically similar shapes such as

O and **O** or L and **L**

Based on the material covered in lectures, suggest a way (ie. an algorithm) by which this might be calculated. Make sure you clearly explain your reasoning. Use diagrams as necessary.

(hint: there are several possibilities, but if you're stuck think of using a morphological filtering technique as a basis)