Introduction to Vision and Robotics: Vision Assignment Report

Ink Pansuwan (s1409577) and Isabella Chan (s1330027)

Introduction

In general, our approach in implementing a coin counter splits into 2 main parts which include detection and classification. In detection, we focus on segmenting the images to enhance the differences between foreground and background. We later detect all of the objects in all images to compute their properties. These will then be our training data which is to be used in training our classification model. In terms of classification, we focus on using multivariate Gaussian model to classify unseen images and the performance will be shown as a confusion matrix, calculated value and percentage correct.

The first section of the report is going to give a **brief outline** of the different stages of our coin counter. The second section will go into **more details about each stage**, including explanation and implementation of our ideas, as well as alternative approaches. The results for each stage are displayed after the explanation. Section 3 will show the **structure of our code** and give a brief description of the scripts and methods. See section 4 for the **classification results and anaylsis**. In the last section, the performance of our program will be **discuss along with limitation, problems and possible improvements.** The MATLAB code of our program will be included in the last section of this report as an appendix.

Methods

Outline of Vision Techniques

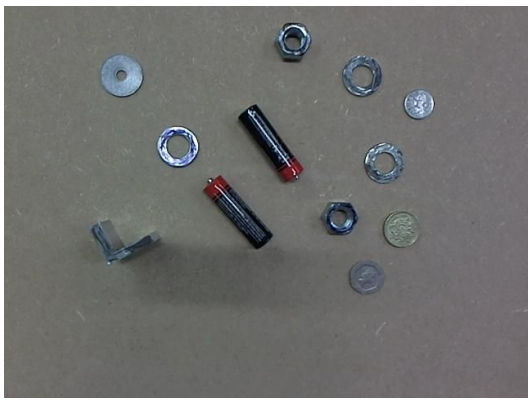
A brief outline of the different stages of our approach to the coin counter problem. Please next section for detailed explanation and result for each stage.

1. **Detection**
   - 1.1. Remove illumination by normalising all images
   - 1.2. Creates a background image by median filtering
   - 1.3. Extracts foreground images in rgb and binary forms for all images by OR thresholding all colour channels
2. **Classification**
   - **2.1. Training**
     - 2.1.1. Extract feature vectors for each detected objects and label each valid object (+ Noise Removal)
     - 2.1.2. Build a Gaussian model and obtain the mean, covariance matrix and apriori
   - **2.2. Testing**
     - 2.2.1. Repeat steps from Detection stage
     - 2.2.2. Disregard invalid objects by putting a threshold on the filled area of the object
     - 2.2.3. With the model built in Training stage, compute class probabilities and assign the class with the greatest value
     - 2.2.4. Hand label all objects' true classes to measure the accuracy of our classification

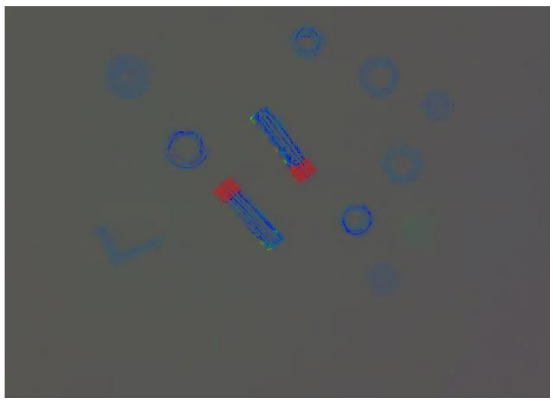
Further Explanation and Results

1.1 Remove illumination by normalising all images

Normalise RBG values of the original images to remove varying illuminations and shadows. Illumination and shadows could act as noise in our images. This step is crucial as skipping this will create extra patches of non-objects and missing parts of the target objects. Even though this step helps, some noises are still shown in the resulting images.



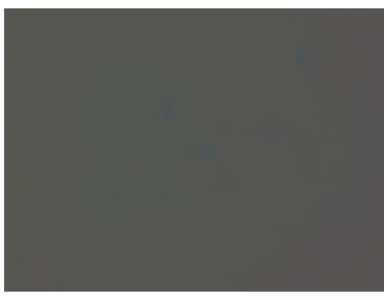
Original image



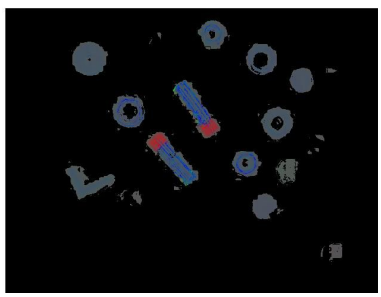
RBG normalized image

1.2 Creates a background image by median filtering

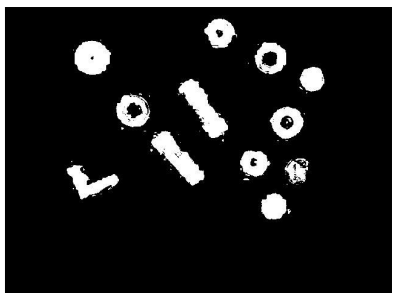
We later did **median filtering** to obtain background and then **background subtraction** to obtain segmented images. Median filtering looks at the r, g and b pixel values for each pixel in every image, and extract the median value to be the background. The concept is that since the camera is stationary in this problem, most of the observations for a given pixel should be of the background. We then convert these to **binary images** as they are simpler to work with and further experiment with image enhacements. The following shows the background, the foreground image in rgb colour, and the binary foreground image.



Background



rgb foreground



Binary foreground

We experimented with **mode and mean filtering**, but the results are not as good as median filtering. The following images show the result of both experimentation.

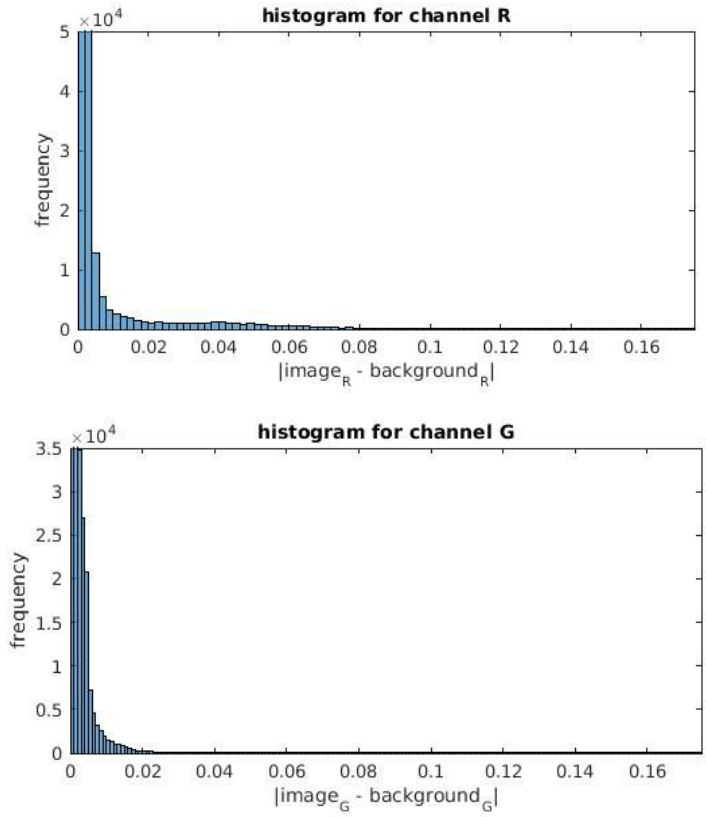
Mode filtering **Mean filtering**

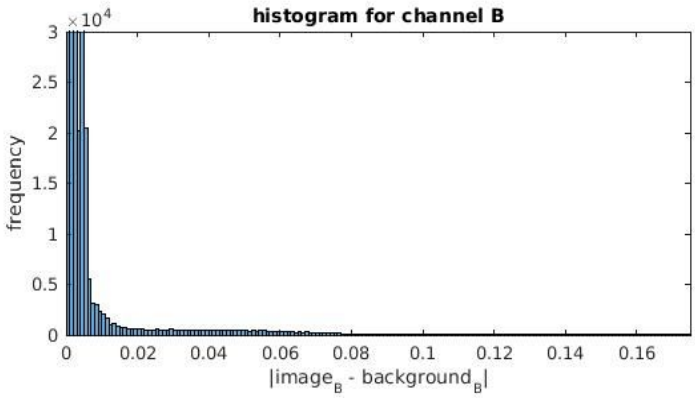
1.3 Extracts foreground images in rgb and binary form for all images by ‘OR thresholding’ all colour channels

To isolate objects, we do thresholding. For each colour channel(r, g, b) we did image subtraction between the image and the background image:

```
diff_R/G/B = modulus(image_R/G/B - background_R/G/B)
```

If the colour pixel in the image is very similar to that of the background image, the absolute difference will be very small. We later plot a histogram of the absolute differences for each colour channel and an obvious drop in frequency for each colour channel is shown:





:

For one image, the threshold can be determined by testing out different values within the range of the drop and pick the ones that outputs the highest quality of foreground for all images.

By eye-balling the histograms for all the provided images, for each colour channel of every images, the best threshold value is identified:

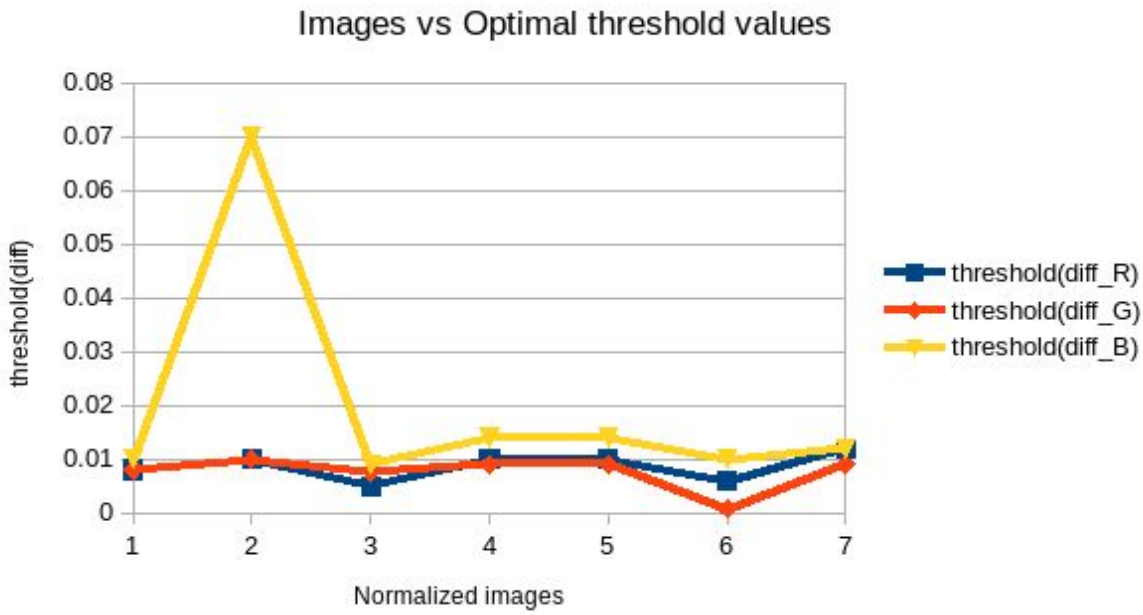


Figure1: Threshold values for each colour channel for all 7 training images. These threshold values are being used during the extraction of objects in order to identify the objects from their background.

We later calculated the following as they should be the best representatives:
Average :[0.0087,0.0086,0.0199]
Mode : [0.01,0.009,0.014 and 0.01]
Median: [0.01,0.009,0.012]

Analysis of the 3 threshold values:

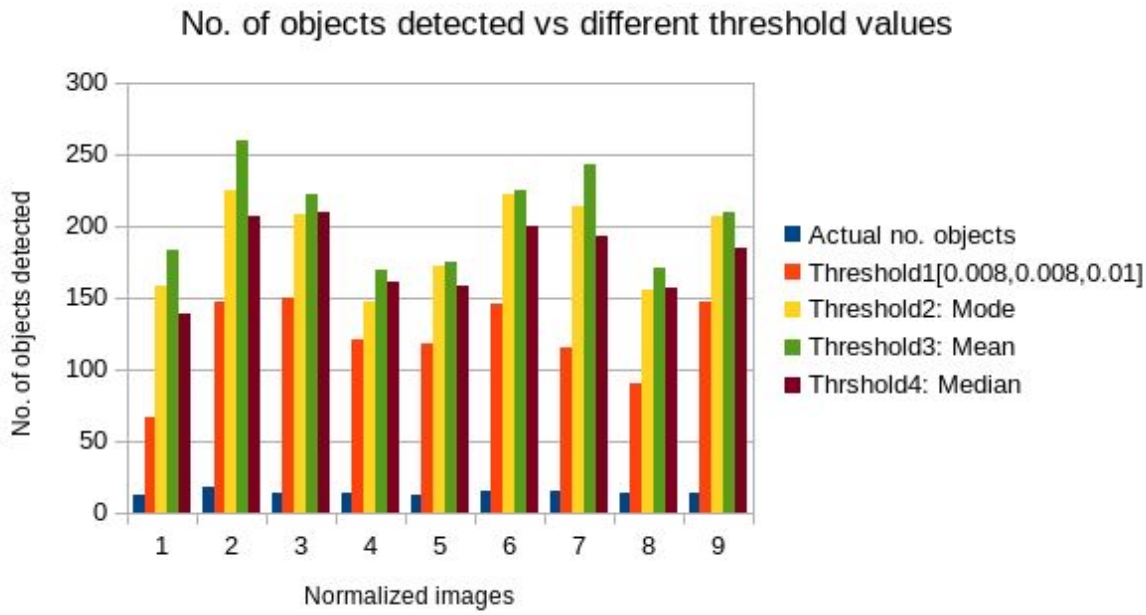
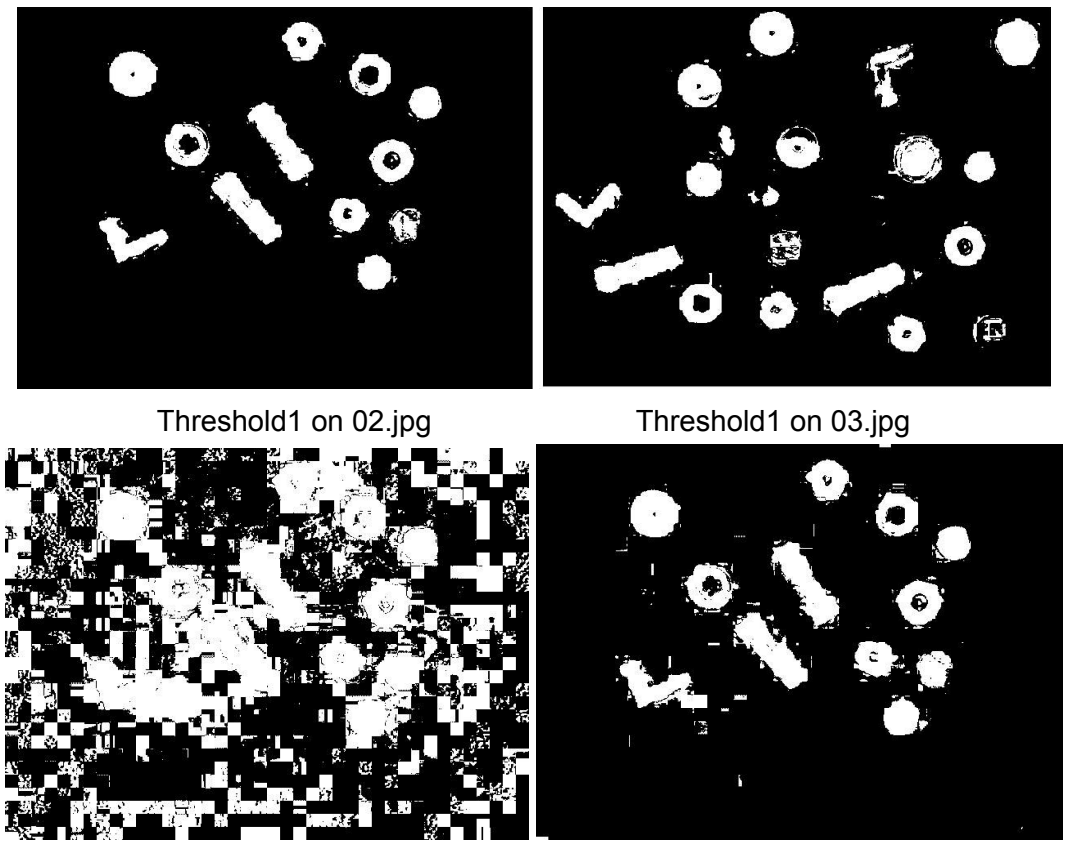
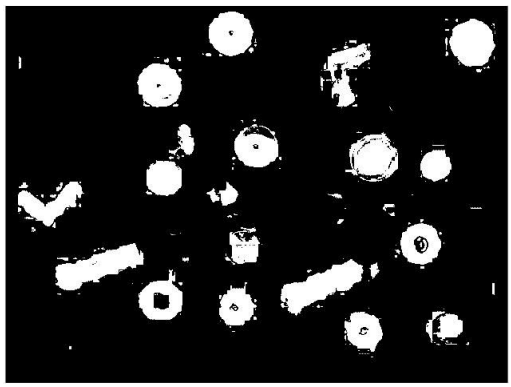


Figure 2: The number of detected objects for each set of threshold value and actual number of objects are used to identify the threshold values that would work best for all images.



Threshold2: Mode Threshold3: Mean



Threshold4: Median

Threshold1: Works well on image1 but on others, too much missing information eg. 1 pound coin.
Threshold2: Too much noise.
Threshold3: Most object areas retained, some noise.
Threshold4: Less object areas retained ans less noise than threshold3.

We will disregard the noise when we train our classifier therefore we are concerned with getting the fullest area of objects. We therefore want to experiment further with threshold 3 but also threshold4 to allow comparisons.

Having run data using these two thresholds, the results show that there is not big difference between them. Nevertheless, we think that if we have more testing images the mean should give us better classification accuracy because after removing the noise, mean threshold gives you fuller objects and our classifer seems to be doing very well at recognizing noise as noise. Our choice on the threshold will be justified at the result section.

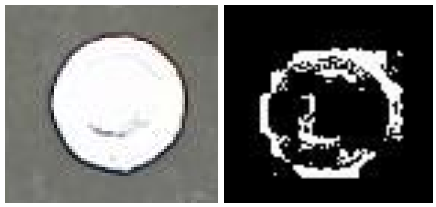
The mean of all the values for each colour channel is used as the threshold for that colour channel. Thus, we can extract the objects using this condition:
`thr(diff_R) or thr(diff_G) or thr(diff_B)`
where absolute differences with values bigger than the threshold is considered as an object. Two separate images are created - a coloured, normalised foreground image and a binary foreground image.

Notes on Noise Removal: Noise will be removed in classification stage. We currently keep noise as the detection is not always perfect therefore some object gets split into many separate parts. Removing the noise now may remose those parts therefore they have to be kept to get the fullest area. We use function bwconncomp with a connectivity of 8 has been used to find connected components:

Examples of correct detections, missing detections and invalid detections: At this stage, every noise is counted as an object. Ignoring the noise, we are actually able to retrieve most objects.

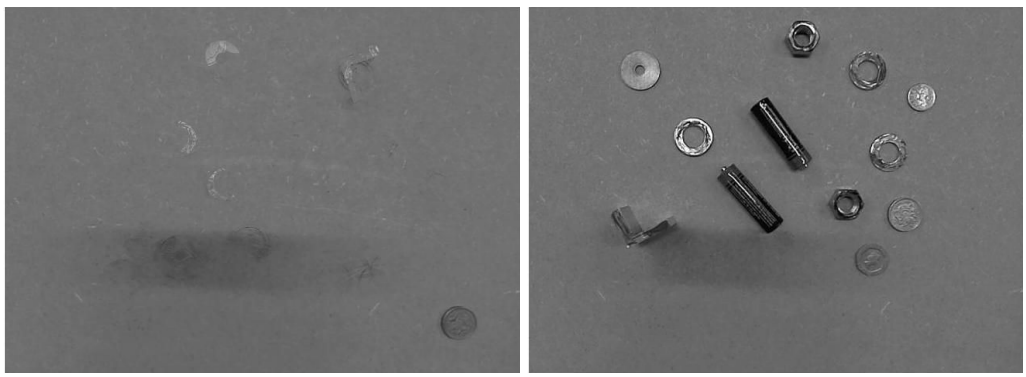
- **Correct detection:** 2 batteries are successfully identified due to the distinction between the pixel values of the battery and the background. There is also miminum effect from the reflection of light on their surface.

- **Missed detection:** The 20p is detected as mutlitple objects for each cluster of white pixels. Much information is lost during the normalisation stage where illumination is removed.

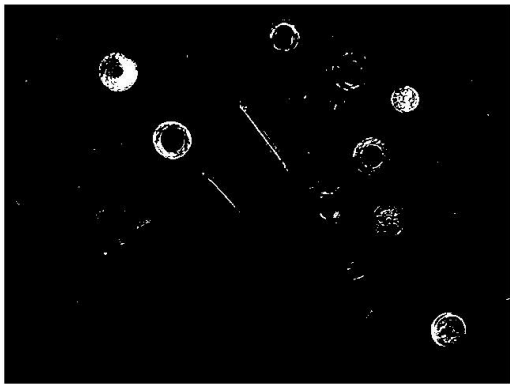
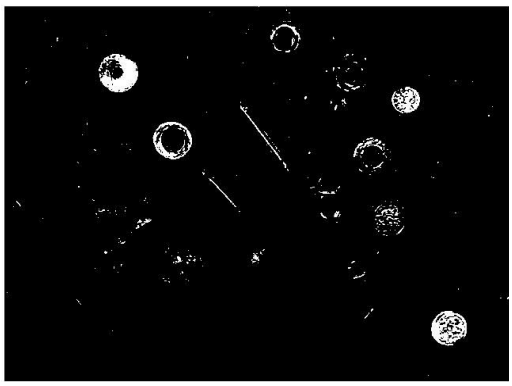


- **Invalid Detections:** This blob in the centre of the binary foreground image does not belong to any coin. This is the noise leftover on our background after background subtraction.

Alternative method experimented for obtaining background and foreground: **work on grayscale images without normalization**



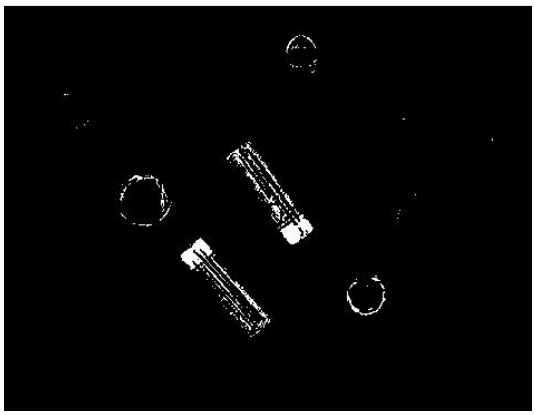
Background obtained Original image-grayscale

After background subtraction & thresholding using bgremove function

This approach does not work as well as normalising the image first, too much noise and too little object information are left.

We also experimented with findthresh to **obtain the threshold for extracting the foreground**:




bgremove & findthresh background subtraction & findthresh

These do not work as well as eye-balling histograms to get the thresholds.

2.1.1 Extract feature vectors for each detected objects and label each valid object (+ Noise Reduction)

Noise Reduction: Prior to features extraction, we experiment with built-in functions to obtain even fuller objects and even less noise in our images:

Number of detected objects vs Noise Removal Function									
Image number	Actual no. of objects	bwmorph with T3	bwmorph with T4	cleanup with T3	cleanup with T4	medfilt2 with T3	medfilt2 with T4	imopen with T4	imopen with T4
02.jpg	12	183	139	67	62	77	70	20	15

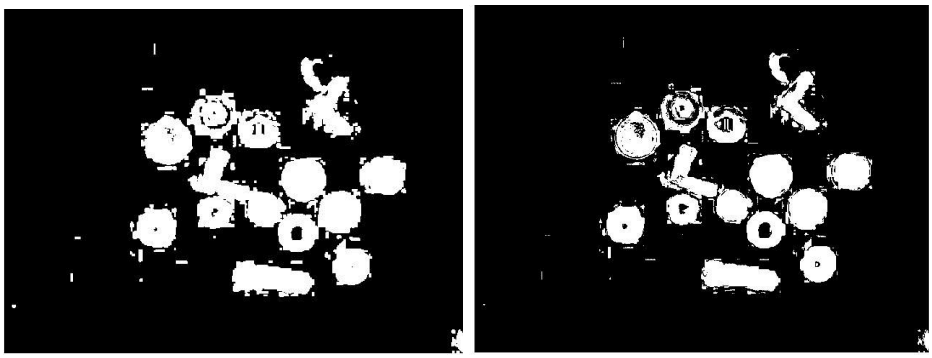
03.jpg	18	259	206	89	84	140	118	27	26
04.jpg	13	222	209	85	88	132	113	27	31
05.jpg	13	169	161	75	61	88	77	30	32
06.jpg	12	175	158	65	60	101	91	22	21
07.jpg	15	224	200	82	80	148	116	35	33
08.jpg	15	243	193	84	85	124	102	31	24
09.jpg	13	170	157	71	63	84	54	24	22
10.jpg	14	209	184	61	67	111	82	33	29

Table 3: T3 and T4 indicates threshold3 (mean) and threshold4 (median). Both thresholds show similar performance with noise removal in all functions, suggesting that the noise removal function performance is independent of the threshold value.

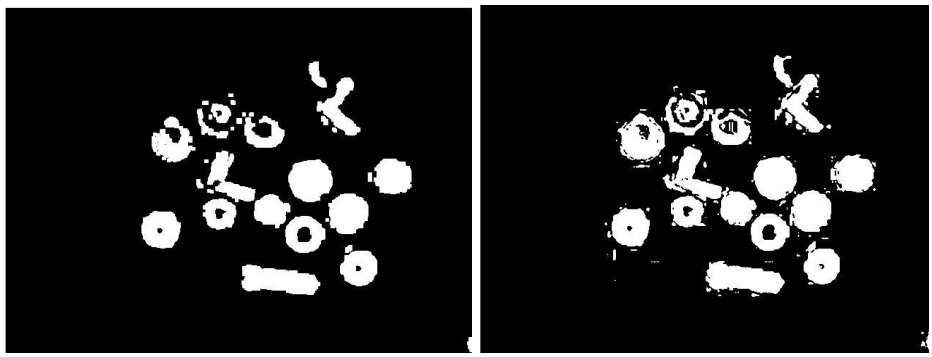
Results and analysis for each noise removal function:

- **bwmorph**(image,'erode',0): highest number of detected objects.
- **cleanup**(image,0.5,0.5,0): indeed removes lots of noise but objects become really close together and can be classified as the same object.
- **medfilt2**(image): Less invalid detections, not too clusterred together, reasonable object area still left.
- **imopen with strel('disk',3,8)**: Much less invalid detection than others but lots of area missing out.

It seems that enhancing the images at this point could create confusion such as connecting two close objects together as one object



clean up with T3 bwmorph with T3



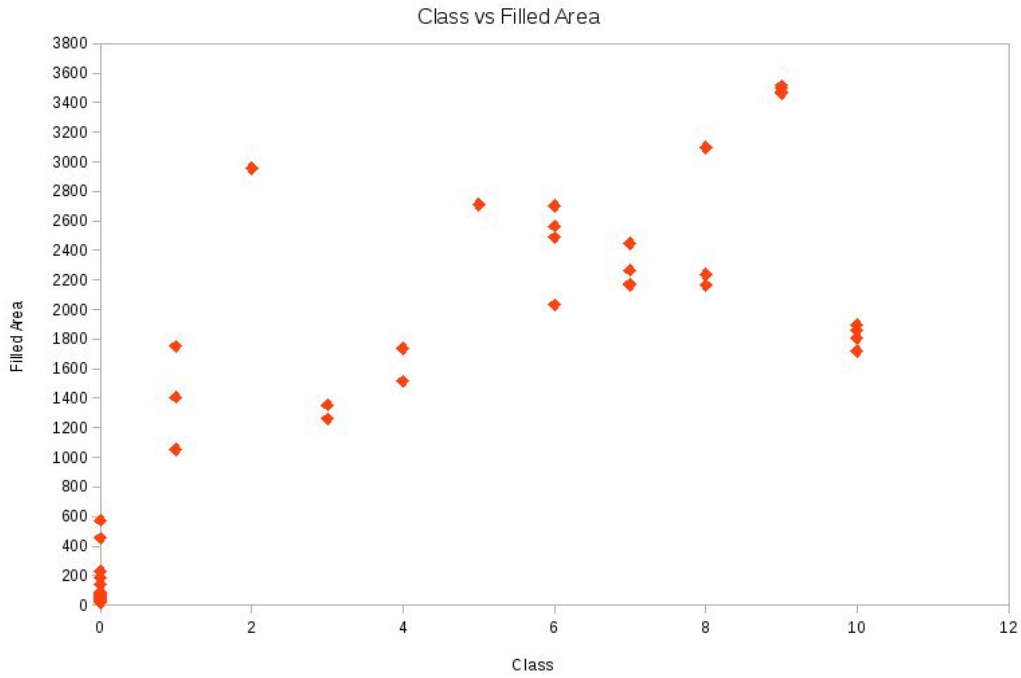
imopen with T3 medfilt2 with T3

Extract feature vectors for each detected objects and label each valid object: Using bwconncomp for extraction as it amends some incorrect detections such as when a one pound coin has been detected as two objects. Since the pixels are close enough to each other, the function would extract the two components as one object. For example, here is a picture of the one pound coin detected.

Without setting the connectivity or setting a connectivity too low, this coin will be labelled as two or more objects. With bwconncomp, most of the white bits will be held together as one object.

Once the objects have been extracted, the feature vectors for them can be computed. The feature vector is a 5-D matrix containing compactness and moments. We picked 5-D because we are using 'buildmodel' function to build our model which requires (n+1) instances for 'n' properties and only 6 instances of the 50p can properly be detected.

More Noise Reduction: Further noise removal will be carried out at the testing stage according to a threshold. This threshold is determined at the training stage. For each labelled object, the function extract_properties returns a matrix containing all feature vectors for each object and the filled area for each object. Our idea is that, if the area is smaller than a certain value, it is an invalid object. This certain value will be somewhere just below the filled area of the smallest valid object detected. The true class labels for a couple images (which holds at least one instance of each class) and the objects' filled areas can give us an idea of the range for this value. See Plot 1 for the variation of class and filled areas.



Plot 1: For all detected invalid objects or noise, we assigned a class 0 so that we can see the difference in filled area between valid and invalid objects. Upper bound for the size of noise is roughly at around 600 units.

The threshold is picked to be 500 because that seems to be the area where there is a gap between invalid and valid objects. This threshold value will be used at testing stage.

2.1.2 Build a Gaussian model and obtain the mean, covariance matrix and apriori

Training a Multivariate Gaussian model: For each labelled object that is not noise, the true class label is assigned manually and its feature vector will be stored in a matrix. We have 10 classes in total for ten different valid objects.

Class 1 = £1
Class 2 = £2
Class 3 = 50p
Class 4 = 20p
Class 5 = 5p
Class 6 = washer with a small hole (75p)
Class 7 = washer with a large hole (25p)
Class 8 = angle bracket (2p)
Class 9 = AAA battery (no value)
Class 10 = nut (no value)

The training images we used are i2, i3, i4, i7, i8, i9 and i10.jpg. When the user sees a noise object, they should input 0 which should disregard that object and move on to the next one. A Gaussian model is built using the feature vectors and their class labels, which gives us the parameters mean, covariance and prior probabilities.

Note on training image selection: We are aware that the assignment sheet suggests us to use 50:50 of ‘simple images’ for training and testing. However, since we have 5 properties, using only 5 images does not give sufficient instances of each object class. Still, we could have reduced the number of properties to 3 but this will negatively impact our classification results too much. In terms of the potential problem with over-feeding the data, it is very unlikely as we barely have enough instances to create the model from using only 7 training images. However, the drawback of this is that we are left with 2 images for testing which will not be sufficient to fully analyse the classifier’s performance.

2.2.1 Repeat steps from Detection stage

For all testing images, repeat the steps from the detection stage to normalise, extract background and foreground images.

2.2.2 Disregard invalid objects by putting a threshold on the filled area of the object

Nosie removal is carried out at this stage. On extracting the feature properties of the objects, the filled area for each object will be extracted too. If the filled area is too small, the object will be considered as noise and put into a garbage class, Class 0. The threshold has been determined to be 500 (see 1.3).

2.2.3 With the model built in Training stage, compute class probabilities and assign the class with the greatest value

Using the Multivariate Gaussian model, for each detected valid object we compute the probability for each class given that object, and assign the class label of that object with the class that has the highest probability.

2.2.4 Hand label all objects’ true classes to measure the accuracy of our classification

Trueclasses of each detected object is labelled, in order to compute the accuracy of our code.

Implementation of Vision Techniques and Structure of Code

*Functions denoted by * are built-in functions or university provided codes or modified university provided code*

Script 1: preprocessImages.m		
Lines	**Function(s) used**	**Description of code/function**
4-27		Load all rgb images and convert them to type M x N x 3 doubles.
29-34	extract_rgb normIm	Normalise all images.
37	extract_bg	Extract a background image.
39-50	extract_fg	Extract rgb and binary foreground images for all images.

Sciprt 2: mytrainingcode.m		
Lines	**Function(s) used**	**Description of code/function**
12-14		Declare variables: - Dim: number of features - maxclass: total number of classes - N: total number of training images
19-42	extract_properties mygetproperties*	Extract feature vectors and filled areas for all detected objects. User labels each detected object their true class: - 0 = noise (disregarded) - 1 = £1 - 2 = £2 - 3 = 50p - 4 = 20p - 5 = 5p - 6 = Washer with small hole (75p) - 7 = Washer with large hole (25p) - 8 = Angle bracket (2p) - 9 = Battery (no value) - 10 = Nut (no value) True class labels stored in trueclasses and feature matrix stored in vec.
45	buildmodel*	Build a Gaussian model. The outputs are the model parameters.

48-60		Save all relevant variables

Script 3: mytestingcode.m		
Lines	**Function(s) used**	**Description of code/function**
11-25		define necessary parameters
26-46	extract_properties	User labels the true class labels for each detected objects. True class labels stored in test_true_classes
48-67	extract_properties classify* multivariate*	Extracts feature vector for each detected objects. Determines whether the object is a noise or valid object. Noise belongs to class 0. Classification output stored in class_vec.
69-90	regionprops* text*	Places labels on classified objects for presentation
96-123		Calculates the predicted value and the actual value of each image.
127-139	confusionmat*	Compute and display confusion matrix and accuracy.
		Save relevant data into mytestingdata.mat

Result

Determining Threshold Values and Noise Removal Function:

In order to determine the threshold values in 1.3 and the noise removal function in 2.1.1, the program has been run with different parameters to access the performances of different approaches. There are three scenarios - threshold 3 (mean), threholds 4 (median) and threshold 3 with medfilt 2 - and threshold 3 with no noise removal function is picked as our best approach towards the problem, since it provides the best classification result. To justify this, we are going to discuss the limitations of each approach:

Mean threshold:

- A battery is always classified as angle bracket.
- 2 pound coin can be confused with a nut.
- 1 pound coin and 5p can be confused with a small hole washer.
- Big washer can be confused as a 2 pound coin.

Median threshold:

- A battery is always classified as angle bracket.
- 2 pounds coin is always classfied as either 1 pound or a nut.
- 20p coin is always classified as either small or big hole washer.
- 1 pound coin can be angle bracket.
- 5p can be a small washer.
-

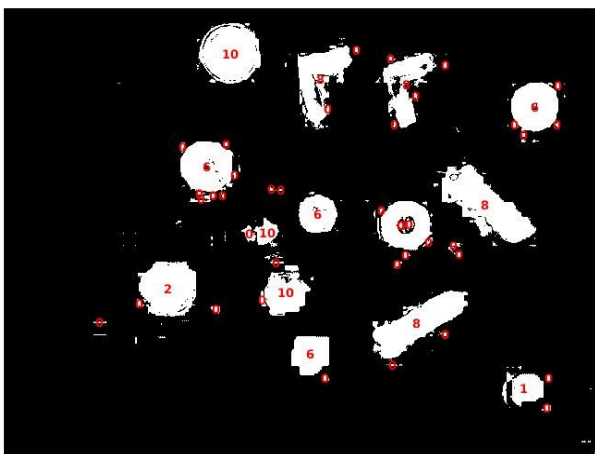
Mean threshold with medfilt2:

- A battery is always classified as angle bracket.
- 2 pounds coin is always classified as a small hole washer or a nut.
- Big hole washer is classifed as 2 pounds coin.
- 20p coin can be classified as a small hole washer.
- 1 pound coin can be classifed as 2 pounds coin.

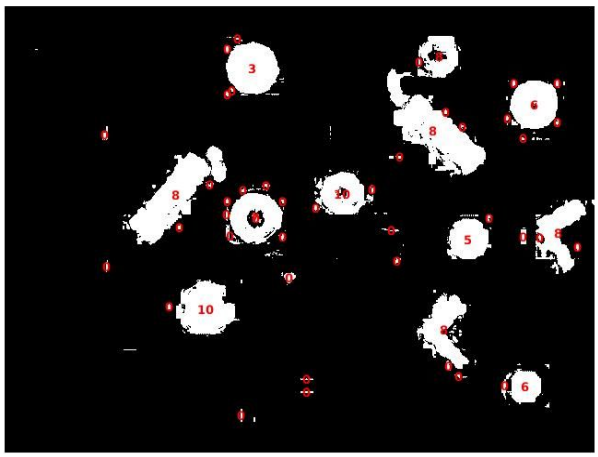
However, for median threshold, **1 pound coin** is classified as an angle bracket. This is very strange as in theory, they should have different compactness and moments as they are very different in shape and size. This may be due to some errors in labelling true classes as it is unique to median threshold case.

- **Mean threshold** provides the least confusion in the classification as the other two have more of an object being ‘always classified’ as something else. Therefore, with the 2 testing images we have, we think that mean thresholding to extract the foreground for our program has the most potential.

Classification Results



08.jpg



09.jpg

08.jpg:
Confusion matrix:

		Predicted classes									
		0	**1**	**2**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Actual classes:	**0**	38	0	0	0	0	0	0	0	0	1
	1	0	1	0	0	0	0	0	0	0	1
	2	0	0	1	0	0	0	0	0	0	1
	4	0	0	0	0	0	1	0	0	0	0
	5	0	0	0	0	0	1	0	0	0	0
	6	0	0	0	0	0	2	0	0	0	0
	7	0	0	0	0	0	0	1	0	0	0
	8	0	0	0	0	0	0	0	2	0	0
	9	0	0	0	0	0	0	0	2	0	0
	10	0	0	0	0	0	0	0	0	0	0

Note: no class 3 object has been detected.

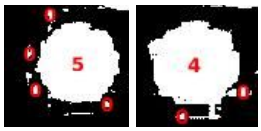
Valid detections: 14, True = 7, False = 7.
Invalid detections: 38, True = 38, False =0.
Calculated value: 6.3300
Actual value: 8.0400

09.jpg
Confusion matrix:

		Predicted classes										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Actual classes:	**0**	41	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	1	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	1
	3	0	0	0	0	0	0	0	0	0	0	0
	4	0	0	0	0	0	1	0	0	1	0	0
	5	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	0	1	0	0	1	0	0	0	0
	7	0	0	0	0	0	0	0	1	0	0	0
	8	0	0	0	0	0	0	0	0	2	0	0
	9	0	0	0	0	0	0	0	0	2	0	0
	10	0	0	0	0	0	0	0	0	0	0	1

Valid detections: 12, True = 5, False = 7
Invalid detections: 41, True = 41, False = 0.
Calculated value: 2.4000
Actual value: 5.1900

Since we did not have enough instances of 50p in simple images, we tested on hard images. (19.jpg and 20.jpg)



50p is falsely classifed as 5p coin and 20p coin in these images. Again, this is a bad classification as they greatly differ in size. However, it is more reasonable for 50p to be classified as 20p as they do resemble in shape more than 5p and 50p. We think the classification is bad because of insufficient properties and training data.

Discussion

Overall results and proposed explanations

- **AAA battery** is always classfied as an angle bracket. This may be because they most similar to each other as they are not circular in shape. However, angle bracket is always classfied correctly, this may be because the instances of angle bracket we put into training have less variations than the battery instances. Since battery is bigger than angle bracket, it therefore has more chance of its surrounding areas being detected as part of it.
- **2 pound coin** being classified as 1 pound, a nut, washer: our proposed explanation is that the gold outer layer of 2 pounds is often incomplete or these smaller objects are detected as a part of them. Therefore, their areas might overlap.
- **Washers**: big-hole washer and 2 pound coins have similar area and compactness. This also applies to small-hole washer and 20p coin.
- However, for median threshold, **1 pound coin** is classified as an angle bracket. This is very strange as in theory, they should have different compactness and moments as they are very different in shape and size. This may be due to some errors in labelling true classes as it only happens in the median threshold case.

Problems and improvements with explanations

Detection:

1. **bwconncomp:** maybe we can try mean-shift clustering? blob analysis?

Classification:

2. **Insufficient training and testing images:** therefore more images are needed to gather more information for our model therefore classification can be more accurate. Also more testing images are required for better analysis and we would like to meet the 50:50 ratio as suggested as that will allow a fair performance testing.
3. **Limited number of properties and insufficient instances for each class:** we need more instances of each class to allow us to meet the condition of buildmodel function therefore more number of properties can be added. We currently have 5 properties which are compactness and moments. We would like to add extra ones such as area, filled area, perimeter and centroid. This will enable us to be as specific about each class as possible therefore distinction an be made easier. However, we are aware of over-feeding the class which will be try to keep that under control.
4. **Unequal number of instances of each class:** this means a certain class will be a better representative of its own class better than others. Therefore, this class is likely to be classified as something else and other similar classes may be more likely to be classified as belong to this class. Hence it introduces bias to the model.
5. **Noise being detected as part of an object:** This alters the values of the properties calculated. For example, if the area changes, the compactness may vary accordingly. Therefore, more variations are introduced.
6. **Large amount of noise detected:** Although we set conditions and manually remove them to help improve the performance of our classifier. It is still a manual and tedious

work. Therefore, it will not be practical if, for example, we are given 100 images to work with. Hence, better noise removal technique is required.

7. **Multivariate gaussian model:** Not a problem per say but we can try to experiment with other approaches to compae the performance such as template matching.
8. **Foreground extraction method:** we thought of another approach to determination of threshold values that may be effective but unable to test due to time constraint. If we plot the distribution of pixel values, we should obtain a Gaussian curve, and the mean would be at the range of pixel values which corresponds to the background, Thus we can take the first (and/or second) standard deviation as the pixel values for the background, and thus removing those values could give us the foreground image.

Credits

We worked together on the coding and the report and we think it is fair the split the credits 50:50. In general, Isabella focus mainly on the backbone of the code and Ink focuses on experimenting with different methods. However, we interchange roles from time to time.

(3955 words)

Code

extract_bg.m

```
function [ image_bg ] = extract_bg( image_vec, show )
%Returns a background image from a list of images
% image_vec is a cell containing all images
% if show = 1, display background image

% get the number of images
len = length(image_vec);

% creates 3 arrays to store r/g/b values for all images
vec_r = cell(1,len);
vec_g = cell(1,len);
vec_b = cell(1,len);

for i = 1:len
        [vec_r{i},vec_g{i},vec_b{i}] = extract_rgb(image_vec{i});
end

% obtain the size for each colour channels
[row,col] = size(vec_r{1});

result_r = vec_r{1};
result_g = vec_g{1};
result_b = vec_b{1};

% do median filtering to obtain the value for each colour channel
% on each pixel
for x = 1:row
        for y = 1:col
        R = zeros(1,len);
        G = zeros(1,len);
        B = zeros(1,len);
        for i = 1:len
        R(i) = vec_r{i}(x,y);
        G(i) = vec_g{i}(x,y);
        B(i) = vec_b{i}(x,y);
        end
        med_r = median(R);
```

```
        med_g = median(G);
        med_b = median(B);

        result_r(x,y) = med_r;
        result_g(x,y) = med_g;
        result_b(x,y) = med_b;
        end
end

% combines the 3 colour channels together
image_bg = cat(3, result_r, result_g, result_b);

% display image
if ~(show == 0)
        figure,
        imshow(uint8(255*image_bg));
        title('background image')
end

end
```

extract_fg.m

```
function [ image_fg, image_fg_bin ] = extract_fg( image_rgb_norm, image_bg, show
)
% extract foreground from an image
% image_rgb_norm: normalised rgb image
% image_bg: background image
% returns the rgb foreground image and binarised foreground image
%   show = 1 to display image

        % r,g,b channels for normalised rgb image
        im_r = image_rgb_norm(:,:,1);
        im_g = image_rgb_norm(:,:,2);
        im_b = image_rgb_norm(:,:,3);

        % r,g,b channels for background image
        bg_r = image_bg(:,:,1);
        bg_g = image_bg(:,:,2);
        bg_b = image_bg(:,:,3);

        % absolute differences between colour channels
```

```
        diff_r = abs(im_r-bg_r);
        diff_g = abs(im_g-bg_g);
        diff_b = abs(im_b-bg_b);

%       % shows 3 histograms for each channel to determine threshold
%       figure
%       subplot(3,1,1)
%       histogram(diff_r)
%       title('histogram for channel R')
%
%       subplot(3,1,2)
%       histogram(diff_g)
%       title('histogram for channel G')
%
%       subplot(3,1,3)
%       histogram(diff_b)
%       title('histogram for channel B')

        % using mean
        thresh_r = 0.0087;
        thresh_g = 0.0086;
        thresh_b = 0.0199;

%       % using mode
%       thresh_r = 0.001;
%       thresh_g = 0.009;
%       thresh_b = 0.01;

        % using median


        % initialises rgb foreground image and binary foreground image
        [row,col,d] = size(image_rgb_norm);
        image_fg = image_rgb_norm;
        image_fg_bin = image_rgb_norm(:,:,1);

        for x = 1:row
        for y = 1:col
        if (diff_r(x,y) >= thresh_r || diff_g(x,y) >= thresh_g || diff_b(x,y) >= thresh_b)
                image_fg(x,y,1) = image_rgb_norm(x,y,1);
                image_fg(x,y,2) = image_rgb_norm(x,y,2);
                image_fg(x,y,3) = image_rgb_norm(x,y,3);
```

```
            image_fg_bin(x,y) = 255;
        else
            image_fg(x,y,:) = 0;
            image_fg_bin(x,y) = 0;
        end
        end
        end

        if ~(show == 0)
        figure
        imshow(uint8(255*image_fg));
        title('rgb foreground image')
        figure
        imshow(uint8(255*image_fg_bin));
        title('binary foreground image')
        end
end
```

extract_properties.m

```
function [detected, properties, filledAreas, labelled] =
extract_properties(image_binary)
% Extract the properties as a 5-dimensional feature vector for each detected object
% Input is a binary image
% Output "detected" is the total number of detected objects (valid or
% invalid) in the image. Output "properties" is a d x 5 matrix for d
% detected objects in the image. Output filledAreas is a d x 1 vector for
% the filled area of each detected object. Output labelled is the binary
% image which has been labelled all detections.

        % enhance the image
        %se = strel('disk',3,8);
        %refined = imopen(image_binary,se);
        refined = bwmorph(image_binary,'erode',1);

        BW = image_binary;
        CC=bwconncomp(BW,8);
        L=labelmatrix(CC);

        props=regionprops(CC,'area');
        idx=([props.Area]>15);
        BW2 = ismember(L,find(idx));
```

```
        CC2 = bwconncomp(BW2,8);
        labelled = labelmatrix(CC2);
        props = regionprops(labelled,'FilledArea');
        filledAreas = [props.FilledArea];

        detected = length(unique(labelled))-1;
        for j = 1:detected
        properties(j,:) = mygetproperties(labelled==j);
        end
end
```

extract_rgb.m

```
function [R,G,B] = extract_rgb( Image_rgb )
% Extract the normalised R, G and B channels for a given image file

        Image_red = Image_rgb(:,:,1);
        Image_green = Image_rgb(:,:,2);
        Image_blue = Image_rgb(:,:,3);

        [row,col] = size(Image_red);

        for x = 1:row
        for y = 1:col
        Red = Image_red(x,y);
        Green = Image_green(x,y);
        Blue = Image_blue(x,y);

        if (Red == 0 && Green == 0 && Blue == 0)
                Red = 1;
                Green = 1;
                Blue = 1;
        end

        rgb_sum = Red + Green + Blue;

        NormalisedRed = Red/rgb_sum;
        NormalisedGreen = Green/rgb_sum;
        NormalisedBlue = Blue/rgb_sum;

        Image_red(x,y) = NormalisedRed;
        Image_green(x,y) = NormalisedGreen;
```

```
        Image_blue(x,y) = NormalisedBlue;
        end
        end

        R = Image_red;
        G = Image_green;
        B = Image_blue;

end
```

mygetproperties.m

```
% gets property vector for a binary shape in an image
%   modified version of the edinburgh university getproperties file
%   modified the number of features where dimension = 5
function vec = mygetproperties(Image)

        [H,W] = size(Image);
        area = bwarea(Image);
        perim = bwarea(bwperim(Image,4));

        % compactness
        compactness = perim*perim/(4*pi*area);

        % rescale properties so all have size proportional
        % to image size
%       vec = [4*sqrt(area), perim, H*compactness];

% get scale-normalized complex central moments
c11 = complexmoment(Image,1,1) / (area^2);
c20 = complexmoment(Image,2,0) / (area^2);
c30 = complexmoment(Image,3,0) / (area^2.5);
c21 = complexmoment(Image,2,1) / (area^2.5);
c12 = complexmoment(Image,1,2) / (area^2.5);
%c=[c11,c20,c30,c21,c12]

  % get invariants, scaled to [-1,1] range
ci1 = real(c11);
ci2 = real(1000*c21*c12);
tmp = c20*c12*c12;
ci3 = 10000*real(tmp);
ci4 = 10000*imag(tmp);
```

```
tmp = c30*c12*c12*c12;
ci5 = 1000000*real(tmp);
ci6 = 1000000*imag(tmp);

%ci=[ci1,ci2,ci3,ci4,ci5,ci6]

%       vec = [compactness,i1,i2,i3,i4,i5,i6,i7];
        vec = [compactness,ci1,ci2,ci3,ci4];
%       vec = [compactness,ci1,ci2];        %only use 3 as only have 4 samples
```

mytestingcode.m

```
% my testing code

% Extracts n x 5 feature property matrix for n test images
% Extracts properties for every detected object in each image
% Uses the filled area of an object to determine whether it is a valid
% object or invalid object (such as noise)
% Classifies each valid detection with classify method (from university)
% class_vec stores all classification output
% test_true_classes stores all user labelled true classes

%load('mytrainingdata.mat')

%to put in true classes for testing images
M =2; % ALTER HERE - no. of test image

%test_true_classes is the correct class we manually put in.
%class_vec is the classes our classifer come up with
test_true_classes = cell(M,1);
class_vec = cell(M,1);
%for labelling output with class number:
labelled = cell(1,M); %new
binary_image = cell(1,M); %new

startIdx = 8;
finIdx = 9;

% give true class labels
index =1;
figure,
for i = startIdx:finIdx
```

```
        subplot(2,1,1);
        imshow(uint8(image_vec{i}));
        current_test_fg_bin = image_vec_fg_bin{i};
        [d,p,a,l] = extract_properties(current_test_fg_bin);
        curr_test_classes = zeros(1,d);

        for j = 1:d
        subplot(2,1,2);
        imshow(l==j)
        curr_test_classes(1,j) = input('Insert (actual) class no.: ');
        end
        test_true_classes{index} = curr_test_classes;
        labelled{index} = l;
        binary_image{index} = image_vec_fg_bin{i};
        index = index+1;
end
close()

% Now run classification

index = 1;
% extract property vectors and classes from each bin_image
for i = startIdx:finIdx
        current_test_fg_bin = image_vec_fg_bin{i};
        [d,p,a,l] = extract_properties(current_test_fg_bin);

        curr_class_vec = zeros(1,d);
        for j = 1:d
        if(a(j) >= 200) % add maximum!!!
        class = classify(p(j,:),maxclasses,Means,Invcors,Dim,Aprioris)
        curr_class_vec(1,j) = class;
        else
        curr_class_vec(1,j) = 0;
        end
        end
        class_vec{index} = curr_class_vec;
        index = index + 1;
end

%for labelling different classes.
for x = 1:M
  l = labelled{x}; %containing labelled images
```

```
  curr = class_vec{x}; %current class vec
  s = regionprops(l, 'Centroid'); %get the centroid out
  h = figure();
  ishold
  hold on;
  imshow(binary_image{x});
  t = strcat('image ',num2str(x));
  title(t);

  for k = 1:numel(s) %number of detected objects
        a = curr(1,k); %select each class in the class vector
        c = s(k).Centroid; %select centroid
        text(c(1), c(2), sprintf('%d', a), ...
        'HorizontalAlignment', 'center', ...
        'VerticalAlignment', 'middle','Color','red','FontWeight','bold');
        end
        hold off;
        filename = strcat('results',num2str(x),'.jpg');
        saveas(h,filename)
end

%to store values
calculated_value = cell(1,M);
actual_value = cell(1,M);

%calculating the value:
 for i = 1:M
        actual = test_true_classes{i};
        curr = class_vec{i};

        v1 = length(find(curr==1));
        v2 = length(find(curr==2))*2;
        v3 = length(find(curr==3))*0.5;
        v4 = length(find(curr==4))*0.2;
        v5 = length(find(curr==5))*0.05;
        v6 = length(find(curr==6))*0.75;
        v7 = length(find(curr==7))*0.25;
        v8 = length(find(curr==8))*0.02;

        vv1 = length(find(actual==1));
        vv2 = length(find(actual==2))*2;
        vv3 = length(find(actual==3))*0.5;
```

```
        vv4 = length(find(actual==4))*0.2;
        vv5 = length(find(actual==5))*0.05;
        vv6 = length(find(actual==6))*0.75;
        vv7 = length(find(actual==7))*0.25;
        vv8 = length(find(actual==8))*0.02;

        calculated_value{i} = v1+v2+v3+v4+v5+v6+v7+v8;
        actual_value{i} = vv1+vv2+vv3+vv4+vv5+vv6+vv7+vv8;
end
 calculated_value
 actual_value


% to display confusion matrix
for a = 1:M
        [confusion_mat,order] = confusionmat(test_true_classes{a},class_vec{a})
        [row,column] = size(confusion_mat); % in case the size changes BUT row and
column should match anyway.
        valid = confusion_mat;
        invalid = valid(:,1);
        valid(:,1)=0;
        valid_detection = sum(sum(valid))
        valid_true = sum(sum(diag(valid)))
        valid_false = valid_detection-valid_true
        invalid_detection = sum(invalid)
        invalid_true = invalid(1,1)
        invalid_false = invalid_detection-invalid_true
end

save('mytestingdata');
```

mytrainingcode.m

```
% my training code

% For each image, two figures are shown with the top being the original rgb
% image and the bottom being the binary labelled image. One object is shown
% each time, and user can give the true class label to each object. If the
% object is noise, input 0 and that object will be disregarded.

% Obtains n by d feature vector matrix for all n valid objects and d dimensions
% Obtains n by 1 true class vectors
```

```
% Builds a classification model

% load('preprocessImages.mat');

Dim = 5;
maxclasses = 10;
N = 7;

%%extract feature vectors and classes from each bin_image
%alllabels = cell(7,1);
%allareas = cell(7,1);
index = 1;
figure,
for i = 1:N
        subplot(2,1,1);
        imshow(uint8(image_vec{i}));
        current_fg_bin = image_vec_fg_bin{i};
        [d,p,a,l] = extract_properties(current_fg_bin);
        for j = 1:d
        subplot(2,1,2);
        imshow(l==j);
        label = input('Insert class no.: ');
        %imagelabels(j) = label;
        if (label > 0)
        vec(index,:) = p(j,:);
        trueclasses(index) = label;
        index = index + 1;
        end
        end
        %alllabels{i} = imagelabels;
        %allareas{i} = a;
end
close()

index = index - 1; % the total number of valid detected items

% build statistical model
[Means,Invcors,Aprioris] = buildmodel(Dim,vec,index,maxclasses,trueclasses);

% % save all data into a .mat file
% modelfilename = 'mytrainingdata';
```

```
% eval(['save ',modelfilename,' Dim vec trueclasses maxclasses Means Invcors
Aprioris'])

% save all data at this stage
save('mytrainingdata');
```

normIm.m

```
function [ image_rgb_norm ] = normIm( r,g,b, show )
%Returns the normalised RGB image
%   r = red channel; g = green channel; b = blue channel
%   if show = 0, do not display figure

        image_rgb_norm = cat(3,r,g,b);
        if ~(show==0)
        figure()
        imshow(uint8(255*image_rgb_norm))
        end
end
```

preprocessImages.m

```
% script #1
% This script preprocesses all images provided for thevision assignment.

% read all images (simple + hard)
i2 = double(imread('02.jpg')); % simple
i3 = double(imread('03.jpg'));
i4 = double(imread('04.jpg'));
i5 = double(imread('05.jpg'));
i6 = double(imread('06.jpg'));
i7 = double(imread('07.jpg'));
i8 = double(imread('08.jpg'));
i9 = double(imread('09.jpg'));
i10 = double(imread('10.jpg'));
i17 = double(imread('17.jpg')); % hard
i18 = double(imread('18.jpg'));
i19 = double(imread('19.jpg'));
i20 = double(imread('20.jpg'));
i21 = double(imread('21.jpg'));

% store all images in a cell called image_vec
% the first 7 elements are training images
```

```
% the 8th to 9th elements are testing images
% the rest are the hard images, stored here for extracting a background
image_vec = {i2, i3, i4, i7, i8, i9, i10, i5, i6, i17, i18, i19, i20, i21};
len = length(image_vec);


% normalise all images and store them in a cell called image_vec_norm
image_vec_norm = cell(1, len);
for i = 1:len
        [r, g, b] = extract_rgb(image_vec{i});
        image_vec_norm{i} = normIm(r,g,b,0);
end

% creates a background image called image_bg
image_bg = extract_bg(image_vec_norm,1);
title('background image');

% store all foreground images in a cell called image_vec_fg
image_vec_fg = cell(1,len);
% store all binary foreground images in a cell called image_vec_fg_bin
image_vec_fg_bin = cell(1,len);

for i = 1:len
        [image_vec_fg{i}, image_vec_fg_bin{i}] = extract_fg(image_vec_norm{i},
image_bg, 0);
        % save binary foreground images into jpeg files
        filename = strcat('bin_fg',num2str(i),'.jpg');
        imwrite(uint8(255*image_vec_fg_bin{i}),filename);
end

% save all variables into a .mat file
save('data');
```