University of Edinburgh

Introduction to Vision and Robotics

Assessed Practical 1: Coin Counter

Authors:
s1317642 Kendeas Theofanous
s1452672 Demetra Charalambous

October 24, 2016

Introduction

The goal of this practical was to develop a Matlab program that takes an image as an input and detects coins and various other objects represented on the image. The program acts like a coin counter. There is a fixed set of 10 object classes that the program should recognise. Each object class has a specific value as listed below:

- one and two pound pieces
- 50, 20 and 5 pence pieces
- washer with a small hole (75p)
- washer with a large hole (25p)
- angle bracket (2p)
- AAA battery (no value)
- and nut (no value)

For the purpose of this assignement, we were given 14 test images. All the images were captured orthographically viewing a scene with the different objects and a static background. Our algorithm detects the objects in the image, segments the objects from the background and classifies the different objects into their distinct classes.

Our main approach was to first create a background using the images provided. To distinguish the objects from the background, we removed the constructed background from each image and converted the image to binary. The white colour represented the different objects and the black colour represented the background. As each image had more than one object, we had to isolate each object of the image. By doing that we were able to create the training set for our program. For the classification process, we followed the same procedure for each test image as described above to get all the objects of the test image. To classify the different objects we calculated the properties of each object in the training set as well as the properties of each object of the test image. By comparing these values, a class is assigned to every object. Depending on the predicted class, the total value of each test image is then calculated.

Methods

In this section, all the methods that were used to run the program are described here. We have also included trial methods that were eventually not used due to unsatisfactory results. The methods are listed below:

main is the main operation of our training and classifying. It contains all the function calls of our algorithm.

read_images is a simple function we implemented to read the different images used in our program. Apart from the directory and the file pattern used to find and read the images, the function has also a parameter **isbinary** which denotes whether the images in the given directory are binary. If they are, a 3-dimensional array is returned containing the read images. This is used to read the binary object images of the training set. Otherwise, a 4-dimensional array is returned containing the RGB images desired.

The **construct_bg** function constructs the background from a set of images given as parameter. All images contain foreground objects and since the camera and illumination are largely stationary, we synthesise a background image by median filtering the values of all pixels in each of the images. We assume that foreground objects will be observed in a few of the images at a given pixel but that most observation will be of the background.

The binary images are created using the function **subtract_bg**. In the lab assignments, each image contains only one object and therefore a threshold value was easy to find by plotting the histogram of the binary image, smoothing it and then automatically generating the threshold. We had a different approach. Subtracting the background from a set of images gives the difference images. The fact that our images contain multiple objects lead us to decide a threshold value after tests and observations. Therefore, the difference images are set to be equal to its values greater than the selected threshold. The channel images are or'ed together to get the binary images. Small regions are detected due to image noise.

We tried further processing of the images to reduce that, using commands like **strel('disk', 1)** and **imclose(binary_image, s)** to close gaps and get clear blobs and **bwmorph 'erode'** and **'dilate'** to remove the image noise. The clarity of the objects in the images after the processing was poor, so we discarded these changes on the binary images. The results of these trials are shown in the 'Results' section.

To train the classifier, the function **get_objects** identifies all the separate objects of a binary training image by getting the largest object of the image, using **getlargest** and subtracting it from the image successively. The training set was created manually by taking six images out of nine of the simple set. We decided to use four training images for each of the ten object classes. The **getproperties** function, provided to us, is used to create an array of vectors holding the properties of the objects. When using get properties vec = [compactness, ci1, ci2] as provided in the function, the evals are exceptionally small, close to

zero, thus the system cannot assign classes. When using get properties vec = [4*sqrt(area), perim, H*compactness], all objects are misclassified. After trial and error we chose to use vec= [complexmoment / 1000, radii, filledArea] as it was more accurate.

We use probabilistic object recognition using the multivariate Gaussian Distribution model. Then, the **buildmodel** function is used to give us the Means of every class, the Invcors i.e. the Inverse covariance matrices for the feature properties and the Apriori of the classes. The Dimension parameter is set to 3 because the Vectors parameter is 3-dimensional. We provide 40 training objects and the Classes parameter is an array containing the true classes of each of the objects. This completes the training of the classifier.

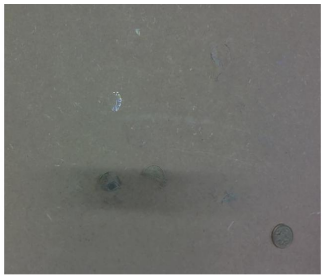
Testing the classifier begins by identifying the individual objects given a test image with multiple objects in it using the function **get_objects**. Their properties are then found using the **getproperties** function. Classification is done using the function **classify**, given the above parameters.

The **multivariate** function inside the classify function, calculates the probability of the object belonging to each class. The classifier then evaluates the probabilities and classifies the objects according to the highest probability calculated.

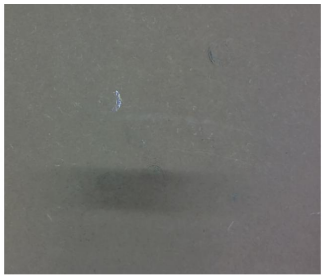
To calculate the total value of each image, we use the function **calc_value** and according to the predicted classes of every object we obtain the total value of the image.

Results

In our first attempt to construct the background we used only the nine simple images provided. As you can see in Figure 1a, the "one pound" object was at the same position in most of the images and that made it impossible to synthesise the background without it. Therefore, it was ideal to use all 14 images provided to us because the more frames we have for each pixel the more accurate the result will be after median filtering.



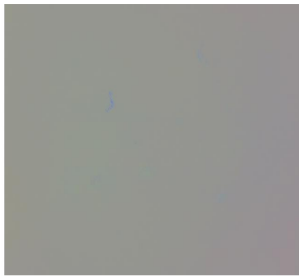
(a) Figure 1a



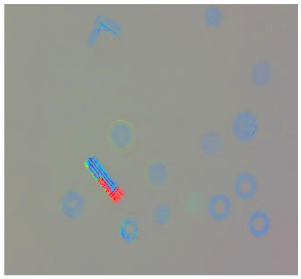
(b) Figure 1b

Figure 1: Background Construction

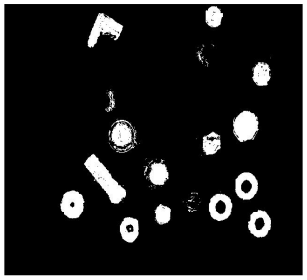
As suggested by the coursework description we tried to normalise our RGB image by splitting it into its red, green and blue channels and each RGB value of each pixel was divided by the sum of its individual RGB values. An example image is shown in Figure 2c and we decided not to use it because of its bad quality.



(a) Normalised Background



(b) Normalised Image

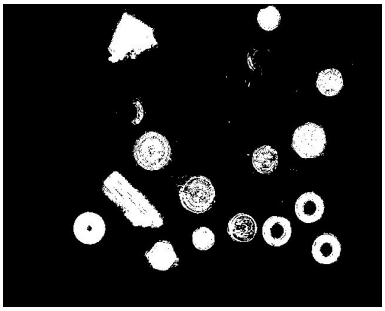


(c) Normalised Binary image

Figure 2: Normalisation of RGB images

Subtraction of the background of the actual image in Figure 3a results to the binary image in Figure 3b, which we used in our training set. The threshold value we used was 0.06.

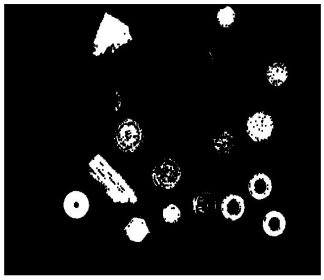
(a) Actual Image



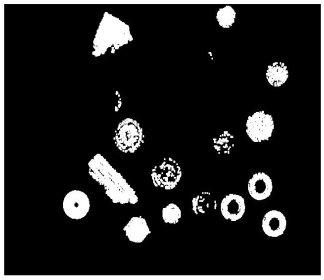
(b) Binary Image

Figure 3: Background Subtraction and Conversion to Binary

As described in the Methods section, we tried to reduce image noise from the images using the bwmorph function, though the result was disappointing. In the following figure you can see the methods we used with their result.



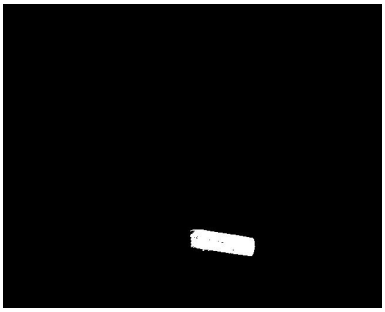
(a) 'erode' 1



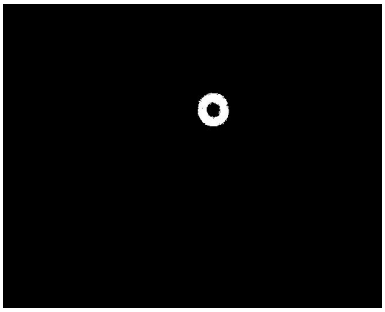
(b) 'erode' and 'dilate' 1

Figure 4: Noise Reduction

In Figure 5a you can see one of the battery objects we used as training and in Figure 5b a washer with a large hole.



(a) Battery



(b) Washer with large hole

Figure 5: Objects of Training Set

After getting the properties of the forty objects in the training set, we built a model and here are the Means and Inverse Covariance matrices. The A prioris are uniform 0.100 for all the classes. These matrices are used in the classification process.

$$Means = 1.0e+03 * \begin{bmatrix} 0.0424 & 4.5847 & 4.2077 \\ 0.0376 & 2.3779 & 3.1953 \\ 0.0309 & 1.3268 & 2.9752 \\ 0.0245 & 0.4667 & 1.8188 \\ 0.0294 & 1.1443 & 2.7205 \\ 0.0228 & 0.3784 & 1.5800 \\ 0.0200 & 0.2483 & 1.2612 \\ 0.0272 & 0.8313 & 2.3000 \\ 0.0287 & 0.7804 & 2.2508 \\ 0.0243 & 0.5342 & 1.8317 \end{bmatrix}$$

$$Invcors(:,:,1) = 1.0e+05 * \begin{bmatrix} 0.0056 & -0.0000 & 0.0000 \\ 0.0001 & -0.0000 & 0.0000 \\ 0.0174 & -0.0000 & -0.0001 \\ 0.0233 & 0.0002 & -0.0002 \\ 0.0277 & -0.0002 & 0.0000 \\ 0.0002 & 0.0000 & -0.0000 \\ 1.0246 & 0.0086 & -0.0117 \\ 0.0443 & -0.0000 & -0.0002 \\ 0.0023 & 0.0000 & -0.0000 \\ 0.0188 & -0.0004 & 0.0001 \end{bmatrix}$$

$$Invcors(:,:,2) = \begin{bmatrix} -1.6272 & 0.0048 & -0.0018 \\ -0.0242 & 0.0001 & -0.0001 \\ -0.8448 & 0.0008 & 0.0032 \\ 18.5837 & 0.1493 & -0.1607 \\ -19.6366 & 0.1784 & -0.0348 \\ 0.5022 & 0.1658 & -0.0970 \\ 857.8832 & 8.0175 & -10.1043 \\ -1.9400 & 0.0065 & 0.0063 \\ 2.6706 & 0.0617 & -0.0561 \\ -41.6379 & 2.2274 & -0.9820 \end{bmatrix}$$

$$Invcors(:,:,3) = 1.0e+03 * \begin{bmatrix} 0.0006 & -0.0000 & 0.0000 \\ 0.0000 & -0.0000 & 0.0000 \\ -0.0073 & 0.0000 & 0.0000 \\ -0.0203 & -0.0002 & 0.0002 \\ 0.0003 & -0.0000 & 0.0000 \\ -0.0004 & -0.0001 & 0.0001 \\ -1.1663 & -0.0101 & 0.0134 \\ -0.0234 & 0.0000 & 0.0001 \\ -0.0030 & -0.0001 & 0.0001 \\ 0.0110 & -0.0010 & 0.0005 \end{bmatrix}$$

When testing our classifier, we used the images shown in Figure 6.

(a) Test Image 1 (b) Test Image 2 (c) Test Image 3

Figure 6: Classification

After extracting the objects of the test binary image, we calculated the properties of each object and compared them with the properties vector of the training set. The classes are numbered from 1 to 10 for "Battery", "Angle Bracket", "One-Pound", "Two-Pound", "50p", "20p", "5p", "small-hole" washer, "large-hole" washer and "nut" respectively. Using classification on the above the results obtained are the following:

Test Image 1 Results:

$$RealClasses = \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 8 & 8 & 10 & 9 & \mathbf{4} & 6 & 6 \end{bmatrix}$$
$$PredictedClasses = \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 8 & 8 & 10 & 9 & \mathbf{6} & 6 & 6 \end{bmatrix}$$

Calculated Value = 4.39 GBP

Test Image 2 Results:

$$RealClasses = \begin{bmatrix} 2 & 5 & 8 & 2 & \mathbf{10} & 6 & 9 & 6 & 10 & 3 & \mathbf{4} & 6 & \mathbf{4} \end{bmatrix}$$
$$PredictedClasses = \begin{bmatrix} 2 & 5 & 8 & 2 & \mathbf{8} & 6 & 9 & 6 & 10 & 3 & \mathbf{6} & 6 & \mathbf{0*} \end{bmatrix}$$

Calculated Value = 5.09 GBP

Test Image 3 Results:

$$RealClasses = \begin{bmatrix} 1 & 1 & 2 & 8 & \mathbf{10} & 9 & \mathbf{10} & 9 & 9 & 7 & 6 & \mathbf{4} \end{bmatrix}$$
$$PredictedClasses = \begin{bmatrix} 1 & 1 & 2 & 8 & \mathbf{6} & 9 & \mathbf{4} & 9 & 9 & 7 & 6 & \mathbf{3} \end{bmatrix}$$

Calculated Value = 4.97 GBP

*Where zero (0) is the class assigned to objects that have very small probabilities for every class. The program prints an "Error message" and explains to the user what class zero stands for.

Discussion

As you can observe, we overall faced a problem on recognising the objects in Class 10("nut" object) and in Class 4 ("one-pound" object). Class 10 is usually misclassified with Class 8 ("small hole" washer), Class 6 ("20p") and Class 4 ("one pound").

It was expected to have a problem on distinguishing nut objects from 20p objects due to their similar shape and size when converting to binary. Their property vectors (obtained from the training set using four objects of each class) are listed below and we can see that their values are similar:

$$20p = \begin{bmatrix} 0.0231 & 0.4289 & 1.6630 \\ 0.0230 & 0.4208 & 1.6590 \\ 0.0228 & 0.3303 & 1.4980 \\ 0.0222 & 0.3336 & 1.5000 \end{bmatrix}$$

$$Nut = \begin{bmatrix} 0.0247 & 0.5669 & 1.8890 \\ 0.0252 & 0.6150 & 1.9760 \\ 0.0242 & 0.5291 & 1.8210 \\ 0.0230 & 0.4256 & 1.6410 \end{bmatrix}$$

However, we were surpriced to see the nut classified as a small hole washer, as their property values were not that similar.

Nut properties(taken from Image 2):

$$Nut = \begin{bmatrix} 0.0250 & 0.5873 & 1.9120 \end{bmatrix}$$

Washer with a small hole properties(taken from training set):

$$washersmallhole = \begin{bmatrix} 0.0274 & 0.8493 & 2.3210 \\ 0.0261 & 0.7008 & 2.1010 \\ 0.0283 & 0.9775 & 2.4980 \\ 0.0271 & 0.7976 & 2.2800 \end{bmatrix}$$

We have also encountered a huge problem with classifying the one pound (Class 4). One pound object is never classified correctly. We observed that in the images given, one pound objects are usually in the shadow and rarely change place. These problems made it difficult for us to create the one pound training set because of the bad quality of all the one pound objects.

One pound properties(taken from training set):

$$onepound = \begin{bmatrix} 0.0245 & 0.4838 & 1.8450 \\ 0.0238 & 0.4819 & 1.7580 \\ 0.0255 & 0.4043 & 1.8860 \\ 0.0240 & 0.4970 & 1.7860 \end{bmatrix}$$

The one pound objects (1 and 2) were classified as Class 6 (20p). Considering the property values of Class 6 in the previous page, we observe that their values (especially for the one pound 2 object) are similar. For this situation, the reason that we think this happens is that their area is almost the same. If we simply take one pound coin and place a 20p coin above it, it almost covers its area. As we are using the "filledArea" property, it was expected to missclassify objects with similar areas.

One pound properties(taken from Image 1):

$$onepound1 = \begin{bmatrix} 0.0233 & 0.3430 & 1.5610 \end{bmatrix}$$

One pound properties(taken from Image 2):

$$onepound2 = \begin{bmatrix} 0.0240 & 0.5036 & 1.8070 \end{bmatrix}$$

The one pound object 3 was classified as Class 3 (two pounds). This might happenned due to their shape similarity and due to bad quality of the one pound objects.

One pound properties(taken from Image 3):

$$onepound3 = \begin{bmatrix} 0.0218 & 0.1858 & 1.0700 \end{bmatrix}$$

The one pound object below is classified as Class 0. As it is clear, its properties are quite smaller than the normal pound values. This is because the object in the binary image had also black pixels inside (apart from the normal white pixels) and using the getlargest method, only the white pixels were used to represent the object. This made the one pound object a lot smaller than it actually is and deformed it.

One pound properties(taken from Image 2):

$$onepound = \begin{bmatrix} 0.0222 & 0.3367 & 1.5160 \end{bmatrix}$$

Overall, apart from the one pound objects and nut objects that are missclassified, the program works well for all the other objects. The positive fact is that it does not take too much time to run and it is very efficient. We have also tried creating the training set with self-collected data (taking pictures of each object individually), but when testing it we realised -because of the different lighting conditions and camera positions- the program would perform poorly. Thus, we have decided to use the objects of the images provided even though it sometimes leads to unsuccessful classifications.

Work Distribution

We used pair programming for the purpose of this assignment. The report load was equally divided as well. We both agree the final mark should be distributed 50:50.

Appendix

main

```
% From the directory extract the images files needed
% to construct the background
% Directory name is changed manually
directory = '/Users/Desktop/IVR_cwk1/files/';
filePattern = '0*.jpg';
% Read the images and store them in a 4-dimensional array
show = 0;
isbinary = 0;
images = read_images(directory, filePattern, isbinary, show);
% Construct the background using the given set of images
show = 0;
background = construct_bg(images, show);
% Convert background to double and save
bg = double(background) / 255;
imwrite(bg, ['bg' '.jpg']);
% Select the images to be used as training set
filePattern_train = 'p*.jpg';
% Read the images and store them in a 4-dimensional array
show = 0;
isbinary = 0;
train_images = read_images(directory, filePattern_train, isbinary,
    show);
% Get the number of images in the training set
num = size(train_images, 4);
for n = 1 : num
    norm_tr_images(:,:,:,n) = double(train_images(:,:,:,n))/255;
end
% Substract the background from each image and convert it to binary
thres = 0.06;
show = 0;
save = 0;
tr_binary_images = subtract_bg(norm_tr_images, bg, thres, show, save
    );
% Here training objects were selected manually
% By using the function get_objects
% Read the objects of the training set
show = 0;
isbinary = 1;
filePattern_obj = 'obj*.jpg';
tr_obj = read_images(directory, filePattern_obj, isbinary, show);
% Get the properties of each image in the training set
for i = 1 : size(tr_obj, 3)
    Vecs(i,:) = getproperties(tr_obj(:,:,i));
end
show = 0;
if (show > 0)
    disp(Vecs);
end
% Given the classes of each image in the training set
% and calculate means, inverse covariance and aprioris
Dim = 3;
N = 40;
Numclass = 10;
Classes = [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5,
    5, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8, 9, 9, 9, 9, 10, 10, 10,
    10];
[Means,Invcors,Aprioris] = buildmodel(Dim,Vecs,N,Numclass,Classes);
show = 1;
if (show > 0)
```

```
    display(Means);
    display(Invcors);
    display(Aprioris);
end
% ******************Testing*****************************
% Read the test image
show = 0;
isbinary = 0;
filePattern_test = 'tst1.jpg';
tst_img = read_images(directory, filePattern_test, isbinary, show);
% Convert the test image to double
norm_tst_img = double(tst_img) / 255;
% Substract the background from test image and convert it to binary
thres= 0.06;
show = 0;
save = 0;
tst_binary_image = subtract_bg(norm_tst_img, bg, thres, show, save);
% Extract the objects that they will be used for testing
num_obj = 12;
show = 0;
save = 0;
tst_obj = get_objects(tst_binary_image, num_obj, show, save);
% Get the properties of each object image
classes = zeros(1,size(tst_obj,3));
for i = 1 : size(tst_obj, 3)
    test_vecs(i,:) = getproperties(tst_obj(:,:,i));
    [class, evals] = classify(test_vecs(i,:),Numclass,Means,Invcors,
    Dim,Aprioris);
    if ((evals < (1.0e-200)))
        classes(1,i) = 0;
        disp(['************Error in recognising object: ' num2str(i)
    ]);
        disp('************Class zero assigned!');
        disp(' ');
        figure
        imshow(tst_obj(:,:,i));
    else
        classes(1,i) = class;
    end
end
disp(classes);

% Calculate total value of the objects in the test image
result = calc_value(classes);
disp(result);
```

read_images

```
function images = read_images( directory, filePattern, binary, show
    )
    % Function to read the images for further processing
    % Retrieve the directory and file pattern of the images
    filePattern2 = fullfile(directory, filePattern);
    jpgFiles = dir(filePattern2);
    % Initialize variables
    dim = length(jpgFiles);
    if (binary == 0)
    % If the picture is not binary
    % Create 4-dimensional array
        images = zeros(480, 640, 3, dim, 'uint8');
        % Read the images and show them if asked
        for d = 1 : dim
            image = imread(jpgFiles(d).name);
```

```
            images(:,:,:,d) = image;
            if (show > 0)
                figure(d)
                imshow(images(:,:,:,d));
            end
        end
    else
    % If the picture is binary
    % Create 3-dimensional array
        images = zeros(480, 640, dim, 'uint8');
        for d = 1 : dim
            image = imread(jpgFiles(d).name);
            images(:,:,d) = image;
            if (show > 0)
                figure(d)
                imshow(images(:,:,d));
            end
        end
    end
end
```

construct_bg

```
function  background  = construct_bg(images, show)
%Function that extracts the background of a set of images
    % Get the background as the median over all the chosen images
    background = median(images, 4);
    if (show > 0)
        figure
        imshow(background);
    end
end
```

subtract_bg

```
function binary_images = subtract_bg( images, background,
    thres_value, show, write )
% Function that removes the background image from each given image
% Returns the binary image
%   Store the length of the images
    num = size(images, 4);
%   Initialise and normalise the 4-dimensional array that will be
    returned
    binary_images = zeros(480, 640, num);
    for n = 1 : num
        % Subtract background from the original image
        no_background = abs(imsubtract(background, images(:,:,:,n)))
    ;
        % Get the binary image for each channel
        % Given a specific thresshold value after tests and
    observations
        thresholded_image = no_background > thres_value;
        % Or the images per channel to get the binary image
        binary_image = thresholded_image(:,:,1) | thresholded_image
    (:,:,2) | thresholded_image(:,:,3);
        % Show the binary images if asked
        if (show > 0)
            figure
            imshow(binary_image);
        end
        % Save the binary images if asked
        if (write > 0)
```

```
            imwrite(binary_image, ['binary_image' int2str(n) '.jpg']);
        end
        % Store the binary image into a new 4-dimensional array
        binary_images(:,:,n) = binary_image;
    end
end
```

get_objects

```
function objects = get_objects( img, num_obj, show, save )
%Function that extracts objects from an image
% *For the training set: All the objects were saved and chosen manually
    for i = 1 : num_obj
        large = double(getlargest(img,0));
        img = abs(imsubtract(large, img));
        objects(:,:,i) = large;
    end
    % Loop through the objects if asked to be shown
    if (show > 0)
        for n = 1 : num_obj
            figure
            imshow(objects(:,:,n));
        end
    end
    % Save image objects if asked
    if (save > 0)
        for n = 1 : num_obj
            im = objects(:,:,n);
            filename = sprintf('obj%d.jpg', n);
            imwrite(im, filename, 'jpg');
        end
    end
end
```

getproperties

```

function vec = getproperties(Image)
% gets property vector for a binary shape in an image
     Image = getlargest(Image,0);
     stats= regionprops(Image, 'FilledArea', 'MajorAxisLength', 'MinorAxisLength');
     FilledA = cat(1,stats.FilledArea);
     diameters = mean([stats.MajorAxisLength stats.MinorAxisLength],2);
     radii = diameters / 2;
     c = complexmoment(Image,1,1)/1000;
     %only use 3 as only have 4 samples
     vec = [radii, c, FilledA];
end
```

classify

```
% classifies a test feature vector v into one of N classes
% given the class means (Means) and inverse of covariance matrices
% (Invcors) and aprori probabilities (Aprioris)
function [class, evaluations] = classify(v,N,Means,Invcors,Dim,Aprioris)
        evals = zeros(N,1);
```

```
        IC = zeros(Dim,Dim);
        for i = 1 : N
            % We need to reshape since Invcors(i,:,:) gives 1
    xDimxDim matrix
            IC = reshape(Invcors(i,:,:),Dim,Dim);
            evals(i) = multivariate(v,Means(i,:),IC,Aprioris(i));
        end
        evaluations = evals';
        bestclasses = find(evals == max(evals));
        class = bestclasses(1);
end
```

calc_value

```
function result = calc_value( classes )
% Function that calculates the value of the objects of the image
% by finding the value of each object according to its class
    % Initialise variables
    num = size(classes, 2);
    result = 0;
    for i = 1 : num
        if (classes(1,i) == 2)
        % class angle bracket(2p)
            result = result + 0.02;
        elseif (classes(1,i) == 3)
        % class two pounds
            result = result + 2;
        elseif (classes(1,i) == 4)
        % class one pound
            result = result + 1;
        elseif (classes(1,i) == 5)
        % class 50p
            result = result + 0.50;
        elseif (classes(1,i) == 6)
        % class 20p
            result = result + 0.20;
        elseif (classes(1,i) == 7)
        % class 5p
            result = result + 0.05;
        elseif (classes(1,i) == 8)
        % class washer with a small hole(75p)
            result = result + 0.75;
        elseif (classes(1,i) == 9)
        % class washer with a large hole(25p)
            result = result + 0.25;
        % classes battery, nut and 'error'(Class 0) are omitted
        % as their value is zero
        end
    end

end
```