AV Assignment 2 Report

Sean Wilson and Daniel Wren
School of Informatics, The University of Edinburgh
s0831408@sms.ed.ac.uk, s0789266@sms.ed.ac.uk

Abstract

Extract a moving object from a motion sequence can be achieved by removing the stationary background. Using the extracted moving object in each frame, a motion history image (MHI) can be created for the object in the image. By calculating scale, rotation and moment invariant features of the MHI the motion sequence can be classified with by a trained Gaussian classifier. With the features obtained and used during the classification, the percentage of correct classifications was 46% (11 out of 24 correct classifications) with rock and scissors being confused 63% of the time. These results could be improved by making the features invariant to which hand is used (flipping). The results could further be improved by locating the number of fingers or a more successful set of features.

Introduction

The program discussed below classifies 24 short motion sequences of a hand playing 'rock, paper, scissors'. The program aims to correctly classify each motion sequence as one of the three actions of this game. There are several steps involved in preparing the motion sequence for classification, these steps are discussed below in detail. After the motion sequence is analysed a feature vector is returned, which describes the motion sequence in a scale, moment and rotation invariant manner. This vector can then be used to classify the motion sequence.

Classification of a motion sequences is performed by a Gaussian distribution model. Using an 8-fold validation method, the variance and mean for each action's feature vector are calculated. This information can then be used by a Bayesian classifier to determine the best fit for a motion sequence. Each motion sequence is processed in turn and classified by calculating the distance between it's feature vector and the feature vectors of the three classes.

The program used in this report was written in Matlab and utilises Matlab's matrix manipulation tools to do mathematical calculations on the images.

The subsequent sections of this report explain the algorithms used to process the motion information and classify as an action. The program is analysed on classifying 24 motion sequences of a hand performing either of the game actions from different angles. The program is analysed on it's classification accuracy.

Algorithms

Background Image

The background of the image is that of a brown textured carpet. As the camera can move slightly from frame to frame and the lighting conditions may change between frames, an average of two captured images will be used for background removal. Two RGB images of the back ground are imported to two dimensional matrices, the values of each pixel in each image are summed and divided by two to get the average RGB value for that pixel. To test whether there is any improvement of using an averaged background image the classification will be tested with using each of the two background images separately and the average, the results will be compared in the discussion section.

Motion Extraction

There are two methods which could be considered here. The first method is to subtract the previous image from the new image in the sequence. The resulting image matrix shows the difference between the two frames.

The second is to extract the background image from each frame in the motion sequence. This removes any stationary pixels in the image. The remaining pixels with a high value represent the change between the two images.

The first method would lead to a smaller resulting matrix and would show little change between frames. There would however be a clear difference between rock and paper action sequences. For example, there would be a distinct change from the initial closed fist position to the extension of the fingers for scissors, or indeed paper.

The second method would give a clear image of the hand's location in each image. When placed in the MHI the whole hands motion will reactivate each pixel it occupies. This would not happen in the first method due to the skin colour being similar, and so being cancelled as background. This would be an undesirable effect and as such method two will be used.

It is worth mentioning that due to the sleeve of subjects arm being a dark value, this is last in the background removal (see figure 1). Although this is part of the motion, it is beneficial for classification as it helps to isolate the hand in the image.

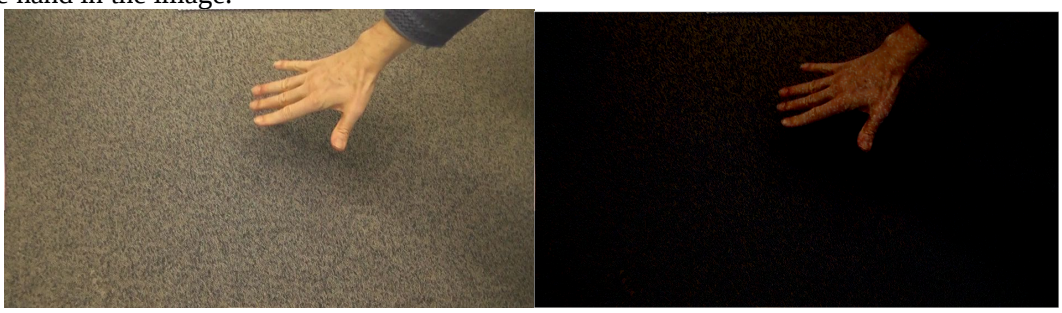


Figure 1: Left, raw image and right, image after averaged background removal

Image Clean-up

To calculate the bounding box the image must be cleaned up to a binary image in which the background pixels obtain the value 0 and the moving object pixels become 1. To decide the threshold between a pixel colour becoming a 0 or a 1, a function called 'doThresh.m' is called, which displays the threshold for the image with a hand. It was decided after observing the histogram (see appendix A) from several clear pictures with a non-blurry hand and through trialling several values, that the colour value of 40 was the optimal threshold.

Once the image is binary, the stray and dangling pixels from the background can be removed to give a smoother image using the 'cleanup.m' function. This function dilates and erodes 3 times to smooth the image. This uses Matlab's 'imerode' to zero any isolated pixels with the value of 1.

Bounding Box Calculation

The bounding box of the hand in the cleaned binary image can be calculated by finding the remaining areas of joined high value pixels in the image matrix. This can be done using Matlab's 'regionprops' function. The largest region of joined high valued pixels should be the moving object, unless there is no moving object in the image. In these cases, the largest bounding box is ignored if under a set dimension. If above this threshold, the top left pixel location, height and width of the bounding box are returned. As can be seen if figure 2, the bounding box is then displayed on the binary image. As the motion history image will need to be cropped to only include the area of interest, it is beneficial to record the largest and smallest x and y values of bounding boxes to be later used when cropping.

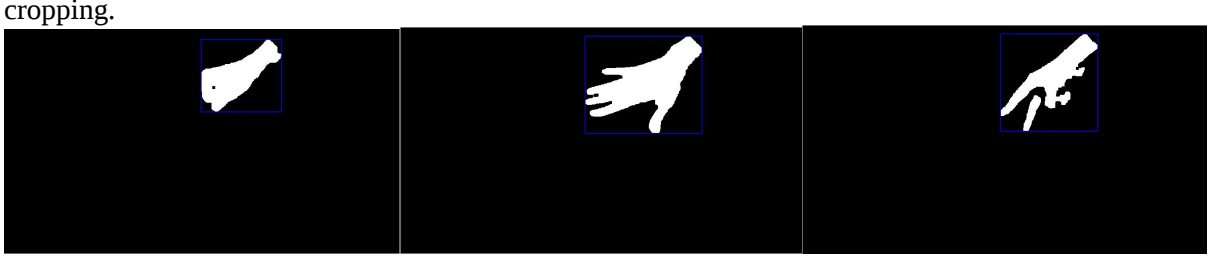


Figure 2: Cleaned binary image with bounding boxes

Motion History Image

The motion history image (MHI) displays the history of the motion in a single image. The motion is represented by a gradient in the grey scale of the objects path. The stronger the white, the more recent the motion. To compute the MHI matrix, each binary image processed is converted into a matrix of zeros and values of 255. This matrix is stored and updated for each image. As a new image is processed, the history matrix is updated by subtracting 10 from each pixel that is not zero. All the pixels in the history matrix with a value of 255 in the new matrix become 255 in the history, indicating the most recent movement captured.

After the last image is added, the matrix is converted to grey scale (0 - 1) using ‘mat2gray’. The resulting matrix is cropped using the bounding box calculated by the largest and smallest x and y values recorded from the individual bounding boxes of each image. The resulting image can be seen in figure 3.

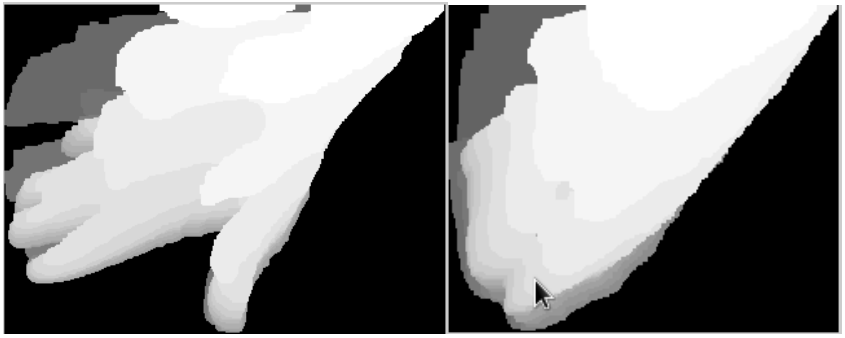


Figure 3: MHI example for 1-1 (paper) and 1-4 (rock)

Moment Invariant Descriptor

To classify the motion captured, it is vital to obtain features that can describe the motion history image. Moment invariant features can be captured that describe the MHI with six values that are invariant to scale, location and, rotation. The compactness of the MHI can also be calculated which can be used to distinguish between the three actions. Compactness is calculated by:

$$\frac{1}{4\pi}\frac{perimeter^2}{area}$$

The area is calculated by summing all the pixels that are not zero in the MHI. The perimeter is calculated using Matlab’s ‘perim’ function which finds the first active pixel (value in the matrix not zero) and checks its neighbouring pixels for the next active pixel, giving a total count of pixel around the circumference of the image.

It is expected that compactness will be the most distinguishing feature of the three actions as, for example rock will be more compact than scissors, which in turn will be more compact than paper due to the extended fingers. The moment invariants were calculated using the formula in figure 3, where *r* and *c* are the rows and columns of the MHI matrix, *r* and *c* hat is the centroid of the image and *i* is the complex (imaginary). *u* and *v* are dependent on which features are being described (see appendix B for details).

$$c_{uv} = \sum_r \sum_c ((r-\hat{r}) + i(c-\hat{c}))^u ((r-\hat{r}) - i(c-\hat{c}))^v mhi(r,c)$$

Each c_{uv}represents one of the six moment invariant features used to describe the MHI. A seven part feature vector for each image is recorded. As there must be less features used than training examples, only 6 of the seven features will be used. As such, the final moment invariant will be dropped in place for the inclusion of compactness. To test whether there is an improvement of using compactness over the sixth moment invariant feature, classification will be run using each in turn and compared on which produces the least variance.

Classifier

To ensure the best fit possible 8-fold cross validation will be implemented. In total there are eight motion sequences of each of the three actions. To train each class, seven of the sequences for each action will be used as training data and will be used in a Bayesian classifier to classify the three remaining test examples. This will be repeated, changing the three test examples each time until all 24 motion sequences are classified.

For each fold, the feature vectors of the training data are averaged, returning the average feature vector for each action. This is used as the mean for that class (the three classes are rock, paper and scissors). The covariance matrix of the training data is also calculated. The mean and covariance matrix give a Gaussian distribution of the three classes which can be used to classify the test set. The similarity between each test motion is then calculated for the three classes using the mean (the average feature vector) and the error distribution (the covariance matrix). The test motion is then classified as the class it most resembles. After all motions are classified by the training data a confusion matrix and classification percentage is printed out (as seen in figure 5).

Results

Table 1 shows the classification results obtained depending on which features were used in the 6 dimensional feature vector where v represents the value in that index of the feature vector.

v = 1	**v = 2**	**v = 3**	**v = 4**	**v = 5**	**v = 6**	**Result**
compactness	c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i5}	46%
compactness	c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i6}	38%
compactness	c_{i1}	c_{i2}	c_{i3}	c_{i5}	c_{i6}	42%
compactness	c_{i1}	c_{i2}	c_{i4}	c_{i5}	c_{i6}	33%
compactness	c_{i1}	c_{i3}	c_{i4}	c_{i5}	c_{i6}	33%
compactness	c_{i2}	c_{i3}	c_{i4}	c_{i5}	c_{i6}	42%
c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i5}	c_{i6}	42%

Table 1: Classification from different feature sets

Class	Rock	Paper	Scissors
Rock	3	2	3
Paper	3	5	0
Scissors	5	0	3

Table2: Confusion matrix for 46% results

Classifications were attempted with less features (4 dimensional and 5 dimensional feature vectors) but results ranged from 25-30% and were therefore deemed inferior. From this table it is clear that [compactness, c_{i1}, c_{i2}, c_{i3}, c_{i4}, c_{i5}] is the most successful set of features with the highest classification result of 46% and a confusion matrix is shown in table 2.

This table represents the feature vectors extracted according to the aforementioned features:

vector	**compactness**	c_{i1}	c_{i2}	c_{i3}	c_{i4}	c_{i5}
1-4 (rock)	3.64	-0.04	0.34	-0.65	0.22	-0.87
1-5 (rock)	4.40	-0.06	-1.84	-0.31	3.25	-9.35
1-6 (rock)	7.73	-0.16	-14.06	19.14	15.26	169.26
2-3 (rock)	7.56	0.01	0.59	-1.60	-0.10	-5.40
2-6 (rock)	4.85	0.01	-0.11	-0.07	0.01	-0.04
3-1 (rock)	2.04	-0.06	-0.58	-0.19	-1.15	-1.60
3-2 (rock)	2.78	0.08	-1.02	0.60	1.25	-0.77
3-3 (rock)	2.66	-0.08	-1.29	1.00	0.94	0.60
1-1 (paper)	2.18	-0.01	0.63	-1.63	-0.14	-5.37
1-2 (paper)	2.41	-0.01	0.21	-1.59	0.15	-5.31
1-3 (paper)	2.38	-0.06	-2.27	-0.60	3.22	-8.08
2-2 (paper)	5.61	0.02	0.15	0.02	0.01	0.02
2-5 (paper)	7.24	0.03	0.41	0.13	0.04	0.11
3-7 (paper)	4.44	-0.04	-0.20	0.06	0.25	-0.41

3-8 (paper)	4.71	-0.04	-0.50	0.09	0.48	-1.01
3-9 (paper)	3.63	-0.04	-0.20	0.17	0.31	-0.53
1-7 (scissors)	4.70	-0.01	0.60	-1.30	-0.20	-8.30
1-8 (scissors)	4.40	-0.10	-1.00	0.60	0.80	-1.2
1-9 (scissors)	7.50	-0.30	-9.33	271.11	160.20	8000.30
2-1 (scissors)	5.40	0.01	0.50	0.10	0.50	-0.50
2-4 (scissors)	5.60	0.10	0.20	-0.10	0.60	-0.70
3-4 (scissors)	2.40	-0.01	0.80	-0.50	0.30	0.50
3-5 (scissors)	2.80	-0.01	0.01	0.01	0.01	-0.01
3-6 (scissors)	3.70	-0.01	-0.30	0.20	0.01	0.10

Discussion

Using an average background image instead of a single background image improved the results of the image clean-up function. This is because there were minute changes in the camera's location between frames in the motion capture, which gave a slight shift in the background matrix each time. The averaged background minimised the effect this inflicted on the background removal between images. An extension to this paper could include getting the difference between each subsequent frame instead of using the background and classifying the MHI of this instead, as previously mentioned, this is not expected to give an increase in accuracy.

Image clean-up for scissor frames were less effective than that for rock and paper. This was due to the amount of shadow around the closed fingers. This effected the area of the image, which in turn effected the compactness values.

The bounding boxes for each image worked successfully because with each image, the hand was detected as the largest region of high values. The dilate and erode code managed to remove any stray pixels that could effect the bounding box, the lower bound also removed the chance of stray pixels being bounded when no hand was detected in the image.

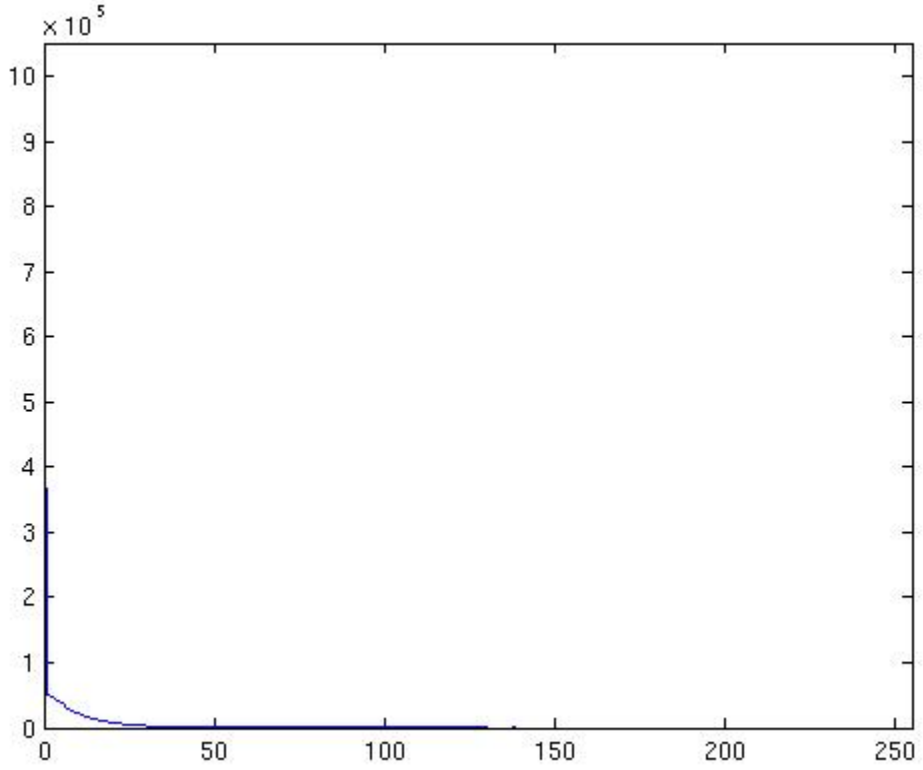
The MHI for rock and paper were relatively clean and distinguishable, but the same cannot be said for scissors. As can be seen in the confusion matrix, scissors actions were interpreted as rock actions 63% of the time (5/8 times) and as scissors the other 33% - this gives us a 33% success rate for scissors classification. Furthermore, 1-9 (a scissor action) obtained particularly bad results for some features. On the other hand, classification of the paper class was the most successful with a 63% success rate. Rock actions had a fairly even classification but only a 33% success rate.

In terms of features used, [compactness, $\mathbf{C}_{i1}$, $\mathbf{C}_{i2}$, $\mathbf{C}_{i3}$, $\mathbf{C}_{i4}$, $\mathbf{C}_{i5}$] made up our feature vectors as these were the most successful with the given data of the dozen different sets that were tested.

To further improve the accuracy of this program, the moment invariants should be made invariant to flipped images. This could increase the accuracy as the motion can come from any direction, performed by the left or right hand.

Appendix

Appendix A



This historgram is after background removal. As can be seen there is a high concentration of dark pixels (the background) and a scatter of lighter pixels (the hand). The cut off was chosen at a value of 40, this is where the background colour seems to end and the hand begins. This cut off does remove the shadowed parts of the hand.

Appendix B

Scale invariant moments (A represents area):

s11 = c11 /(A2)
s20 = c20 /(A2)
s21 = c21 /(A2.5)
s12 = c12 /(A2.5)
s30 = c30 /(A2.5)

Rotation invariant moments:

ci1 = real(s11)
ci2 = real(1000 ∗ s21 ∗ s12)
ci3 = 10000 ∗ real(s20 ∗ s12 ∗ s12)
ci4 = 10000 ∗ imag(s20 ∗ s12 ∗ s12)
ci5 = 1000000 ∗ real(s30 ∗ s12 ∗ s12 ∗ s12)
ci6 = 1000000 ∗ imag(s30 ∗ s12 ∗ s12 ∗ s12)

Code

main.m

```
%--Average over the two background images provided
avBackground = averageImages('background1', 'background2', 0);

%--Get a list of all folders in train
rootDir = cd;
trainDir = strcat(rootDir, '/train/');

featureVectors = getFeatureVectors(trainDir, avBackground);

classes = [1;1;1;2;2;2;3;3;3;1;2;3;1;2;3;3;3;3;2;2;2;1;1;1];

classification(featureVectors, classes);

disp('Classified');
```

---

getFeatureVectors.m

```
%--This is passed either train, paper, rock or scissors
function fVectors = getFeatureVectors(folderDir,background)

folderList = dir(folderDir);
fVectors = zeros(length(folderList)-2,6);

%---------For each sequence------------
for i = 3 : (length(folderList))
   nextDir = strcat(folderDir,folderList(i).name);
   fileList = dir(nextDir);
   historyMatrix = mat2gray(zeros(540, 960));
   xMin = 960;
   yMin = 540;
   xMax = 0;
   yMax = 0;
   bBoxInfo = [xMin, yMin, xMax, yMax];
   cuv = 0;

   %---------For each picture---------
   for k = 3 : (length(fileList))
          fileName = fileList(k).name;
          disp(fileName);

          %--Get difference
          filePath = strcat('train/', folderList(i).name, '/');
          disp(filePath);
          hand = getDifference(background, filePath, fileName, 0);

          %--Turn to bw
          doHist(hand, 0);
          whiteHand = doThresh(hand, 40, 0);
          binaryImage = cleanup(whiteHand, 3, 3, 0);

          %--Get bounding boxes
          box = boundingBox(binaryImage, 0);
```

```
        if box ~= 0
                if box(1) < xMin
                        xMin = box(1);
                end
                if box(2) < yMin
                        yMin = box(2);
                end
                if (box(3) + box(1)) > xMax
                        xMax = box(1) + box(3);
                end
                if (box(4) + box(2)) > yMax
                        yMax = box(2) + box(4);
                end
        end

        bBoxInfo = [xMin, yMin, xMax, yMax];

        %--MHI--don't do until hand seen
        if (xMin < xMax)
                historyMatrix = updateHistory(bBoxInfo, binaryImage, historyMatrix, 0);
        end
end

%--Draw big box
%drawBigBox(bBoxInfo,historyMatrix,0);

%--Change to gray scale and crop
cropSize = [bBoxInfo(1), bBoxInfo(2), (bBoxInfo(3) - bBoxInfo(1)), (bBoxInfo(4) - bBoxInfo(2))];
historyMatrix = mat2gray(historyMatrix);
historyMatrix = imcrop(historyMatrix, cropSize);
figure(2);
imshow(historyMatrix);

%---------Get feature vector-------

%--Calculate area
area = sum(sum(bwlabel(historyMatrix)));
perim = bwarea(bwperim(historyMatrix,8));

%--Scale and translation invariant
s11 = (translationInvariant(1,1,area,historyMatrix)) / area^2;
s20 = (translationInvariant(2,0,area,historyMatrix)) / area^2;
s21 = (translationInvariant(2,1,area,historyMatrix)) / area^2.5;
s12 = (translationInvariant(1,2,area,historyMatrix)) / area^2.5;
s30 = (translationInvariant(3,0,area,historyMatrix)) / area^2.5;

%--Rotation, scale and translation invariant
ci1 = real(s11);
ci2 = real(1000 * s21 * s12);
ci3 = 10000 * real(s20 * s12 * s12);
ci4 = 10000 * imag(s20 * s12 * s12);
ci5 = 1000000 * real(s30 * s12 * s12 * s12);
ci6 = 1000000 * imag(s30 * s12 * s12 * s12);

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

```
   fVector = [compactness,ci1,ci2,ci3,ci4,ci5]
   fVectors(i-2,:) = fVector;
end
```

---

getDifference.m

```
function difImage = getDifference(background, filePath, fileName, fig)

        cd(filePath);
        nextImage = importdata(fileName,'jpg');
        cd ../..;

        difImage = abs(nextImage - background);

        if fig > 0
                figure(fig);
                imshow(difImage);
        end
end
```

---

doThresh.m

```
function threshImage = doThresh(image,threshold,show)
        num = 1;                % used to set the number of times image is sized
        [m,n,num] = size(image);

        for row = 1 : m         % = 640 (width)
        for col = 1 : n    % = 480 (height)
        if image(row,col) < threshold
        threshImage(row,col) = 0;
        else
        threshImage(row,col) = 1;
        end
        end
        end
        if show > 0
        figure(show)
        imshow(threshImage)
        end
end
```

---

boundingBox.m

```
function box = boundingBox(binaryImage, show)

        pHeight = 540;
        pWidth = 960;

        regions = regionprops(binaryImage, 'all');
        numRegions = size(regions, 1);

        %--Display figure if not zero
        if show > 0
```

```
    figure(show);
    imshow(binaryImage);
        end

        %--Get the largest bounding box
        largestArea = 0;
        boxIndex = 0;
        for k = 1 : numRegions
                if regions(k).Area > largestArea
                        largestArea = regions(k).Area;
                        boxIndex = k;
                end
        end

        if (boxIndex ~= 0) && (regions(boxIndex).Area < 100)
                boxIndex = 0;
        end

        %--If bounding box found..
        if boxIndex > 0
                box = regions(boxIndex).BoundingBox;
                xMin = box(1);
                yMin = box(2);
                xMax = xMin + box(3);
                yMax = yMin + box(4);

                hold on
                line(xMin, yMin);
                line(xMax, yMax);

                x = [xMin xMax xMax xMin xMin];
                y = [yMin yMin yMax yMax yMin];
                if show > 0
                        plot(x, y, 'LineWidth', 2);
                end
                hold off

        %--If no box found..
        else
                box = 0;
                display('No hand detected');
        end
end
```

---

drawBigBox.m

```
function bigBox = drawBigbox(bBoxInfo,historyImage,fig)

        if fig > 0
                figure(fig);
                imshow(mat2gray(historyImage));
                hold on;

                line(bBoxInfo(1), bBoxInfo(2));
                line(bBoxInfo(3), bBoxInfo(4));
```

```
                x = [bBoxInfo(1) bBoxInfo(3) bBoxInfo(3) bBoxInfo(1) bBoxInfo(1)];
                y = [bBoxInfo(2) bBoxInfo(2) bBoxInfo(4) bBoxInfo(4) bBoxInfo(2)];
                plot(x, y, 'LineWidth', 2);
                hold off;
        end
end
```

---

findMoments.m

```
function cuv = translationInvariant(u,v,mhi)

        rHat = 0;
        cHat = 0;
        cOfM = [rHat, cHat];
        area = bwarea(bwlabel(mhi));

        for row = 1 : size(mhi, 1)
                for col = 1 : size(mhi, 2)
                        rHat = rHat + (row * mhi(row,col)) / area;
                        cHat = cHat + (col * mhi(row,col)) / area;
                end
        end

        for row = 1 : size(mhi, 1)
                for col = 1 : size(mhi, 2)
                        cuv = ((complex((row - rHat), (col - cHat)))^u) * ((complex((row - rHat), (col - cHat))^v)
* mhi(row,col);
                end
        end
end
```

---

classification.m

```
function result = classification(Vecs,Classes)

        Dim = 6;

        counter = 0;

        trainingData = zeros(21,Dim);
        trainingClasses = [1;1;1;1;1;1;1;2;2;2;2;2;2;2;3;3;3;3;3;3;3]

        confusionMatrix = zeros(3,3);

        rockSamples = find(Classes == 1);
        rockVectors = Vecs(rockSamples,:)

        paperSamples = find(Classes == 2);
        paperVectors = Vecs(paperSamples,:)

        scissorsSamples = find(Classes == 3);
        scissorsVectors = Vecs(scissorsSamples,:)

        for i = 1 : 8

                tempRockVectors = rockVectors;
```

```
            tempPaperVectors = paperVectors;
            tempScissorsVectors = scissorsVectors;


            tempRockVectors(i,:) = [];
            tempPaperVectors(i,:) = [];
            tempScissorsVectors(i,:) = [];


            trainingData(1:7,:) = tempRockVectors;
            trainingData(8:14,:) = tempPaperVectors;
            trainingData(15:21,:) = tempScissorsVectors;

            [Means,Invcors,Aprioris] = buildmodel(Dim,trainingData,21,3,trainingClasses);

            disp('i =');
            disp(i);
            rockResult = classify(rockVectors(i,:),3,Means,Invcors,Dim,Aprioris);
            paperResult = classify(paperVectors(i,:),3,Means,Invcors,Dim,Aprioris);
            scissorsResult = classify(scissorsVectors(i,:),3,Means,Invcors,Dim,Aprioris);

            disp(rockResult);
            disp(paperResult);
            disp(scissorsResult);

            if (rockResult == 1)
                  counter = counter + 1;
                  confusionMatrix(1,1) = confusionMatrix(1,1) + 1;
            elseif (rockResult == 2)
                  confusionMatrix(1,2) = confusionMatrix(1,2) + 1;
            else
                  confusionMatrix(1,3) = confusionMatrix(1,3) + 1;
            end

            if (paperResult == 2)
                  counter = counter + 1;
                  confusionMatrix(2,2) = confusionMatrix(2,2) + 1;
            elseif (paperResult == 1)
                  confusionMatrix(2,1) = confusionMatrix(2,1) + 1;
            else
                  confusionMatrix(2,3) = confusionMatrix(2,3) + 1;
            end

            if (scissorsResult == 3)
                  counter = counter + 1;
                  confusionMatrix(3,3) = confusionMatrix(3,3) + 1;
            elseif (scissorsResult == 1)
                  confusionMatrix(3,1) = confusionMatrix(3,1) + 1;
            else
                  confusionMatrix(3,2) = confusionMatrix(3,2) + 1;
            end
      end

      result = (counter / 24) * 100
      confusionMatrix
end
```