Advanced Vision 2011/2012
Practical 2

Stephen McGruer (s0840449) and Wenqi Yao (s0838969)

1 Introduction

In this assignment, we were tasked with extracting hand movements from sequences of images, and classifying each sequence into one of the three moves of the game 'Rock, Paper, Scissors'. We built a system that is able to reliably extract a hand from a scene in the presence of noise, is robust in handling translation, rotation and flipping of movements, and is able to classify the extracted features with high accuracy. In cross-validation testing on our data set, our system has a >80% classification rate.

2 Algorithms

2.1 Extracting the Hand

In order to extract the hand from an image frame, we first subtracted off the background image. We computed the 'average' background as a weighted sum of the two backgrounds provided - $\alpha(background1) + (1-\alpha)(background2)$. By trialling a range of values, we found that the second image more accurately represented the background, and the best accuracy was given by $\alpha = 0.2$. When doing the background subtraction we also isolated the red component of the images, as hands show more clearly in this reduced spectrum. The subtracted image was then converted to greyscale.

To extract the hand from the greyscale image, the saturation was increased by using `imadjust`. While this function increases noise, it helps the capture of the scissors movements by making the second finger clearer, and increases the quality of the rock movement slightly. The saturated image was then thresholded to a black and white image, which was dilated and eroded a number of times in order to remove noise and to try and connect any fingers to the hand.

Once the image had been cleaned, we used `bwlabel` and `regionprops` to extract the largest connected component in the image, which should be the hand.

We had initially attempted to use Harris corner detection to extract the hand in each frame, but the specular background consistently interfered with the calculation of derivatives. The dark specks have a large colour difference from the lighter background. This causes the colour gradient between pixels to be very high, and hence the specks are classified as 'important' features.

After processing, many of the middle images in our sequence (i.e. when the full 'rock', 'paper' or 'scissors' movement is carried out) are clearly extracted. However, if sufficient noise survives the initial erode, the individual points merge into a single noise component, which can easily outsize the hand and thus be selected by our algorithm. Additionally, we

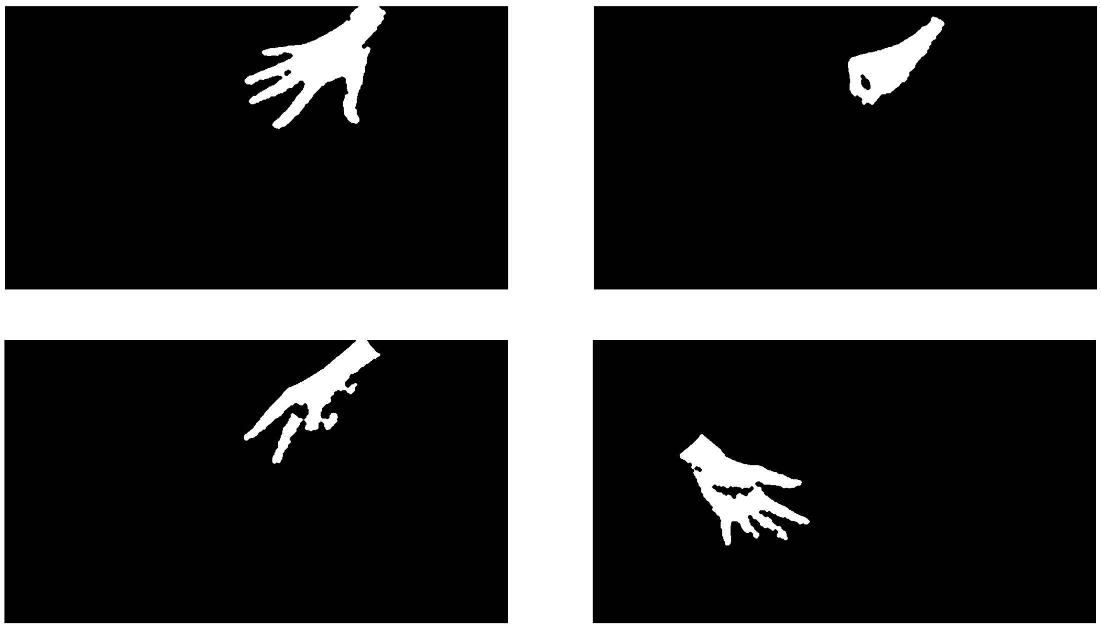
Figure 1: Four example images after the first stage of background subtraction and hand extraction. For many of the images, we manage to extract clear images that would be easily classified by a human viewer.

still struggle to extract the second finger during many of the scissors sequences - this issue is less serious than it seems as it still leaves a distinctive gesture for us to recognise.

Some examples of extracted hands can be found in Figure 1.

2.2 Computing the Bounding Box

For each sequence, the bounding box was determined via the following steps:

1. For each image i, the maximum and minimum x values (max_x^i, min_x^i), maximum and minimum y values (max_y^i, min_y^i), and the mean x and y values (avg_x^i, avg_y^i) were found.
2. The overall mean x value of the sequence was then found (avg_x).
3. The sequence R was computed by removing all images j for which $|avg_x^j - avg_x| < \tau$, for a predefined threshold τ. This helped prevent 'noise' frames from skewing the results.
4. The maximum and minimum x and y values for the entire sequence were set as the maximum and minimum of the reduced set R $(max_x = argmax\{max_x^i\}$, etc.)
5. To compensate for some images being 'left-handed' and others being 'right-handed', the mean x values of the entering and exiting frames of R were examined. If the values were in the left half of the image, the variable $flipped$ was set.

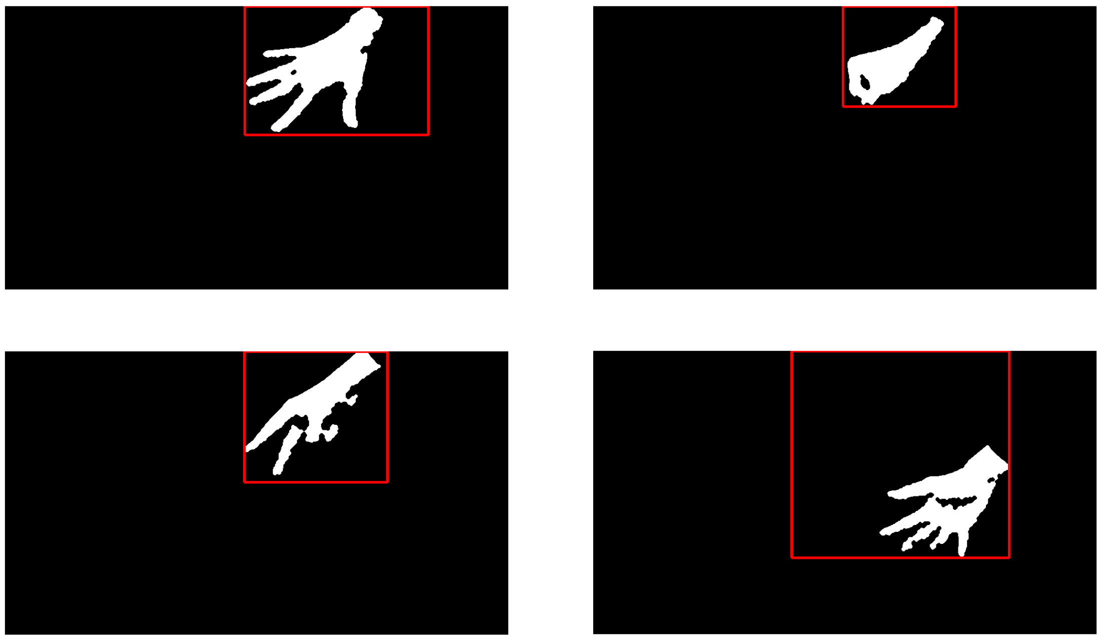

Figure 2: Four example images after the bounding box calculation. Note that the last image has been flipped to be right-handed.

6. If *flipped* was set, steps 1 through 4 were repeated with the images in the sequence flipped horizontally.

Our bounding box calculation is robust in the presence of noise, and sets all of our training images to be right-handed. The result of applying our bounding box algorithm to the previous examples can be found in Figure 2.

2.3 Computing the Motion History Image

Our implementation of the motion history image algorithm followed that of Davis and Bobick [1]. For a sequence of t images, the value of a pixel (x, y) in the motion history image H_τ was determined by the function:

$$H_\tau(x,y,t) = \begin{cases} \tau, & \text{if } D(x,y,t) = 1, \\ max(0, H_\tau(x,y,t-1) - 1), & \text{otherwise} \end{cases}$$

where $D(x, y, z)$ was the value of the pixel (x, y) in the t-th image of the sequence.

This definition means that areas of more recent movement appear brighter in the output image. Some examples of our generated motion history images can be found in Figure 3.

2.4 Computing the Features

As the hand gestures may appear at any scale and orientation within the sequence, we computed seven invariant features. These were the image *compactness* and the six rotation

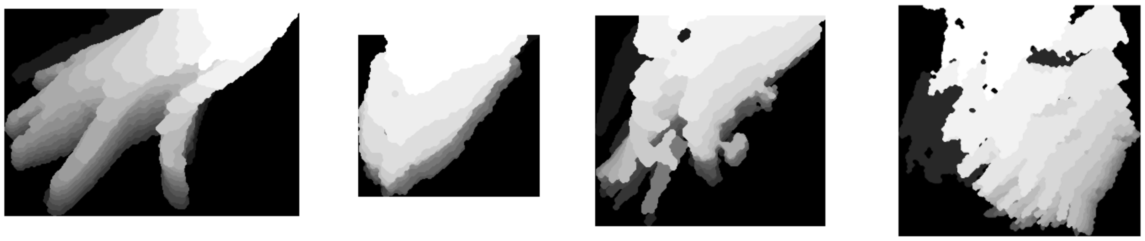
Figure 3: The Motion History Image for four example images.

invariant moments (ci_n) given in [2], altered for use with history moment images. We also experimented with using Hu invariants[3], but found that they yielded poorer results.

Plotting two of the features (as seen in Figure 4) returned by our calculations show that most of the input images are grouped according to class, but that there are a number of instances where the classes are not separated. This indicates that our results are not performing as well as they could, either due to poor moment calculations or due to poor motion history images. One possible avenue for exploration is to use features that are also skew invariant, as this would remove the need for us to try and flip images. The full set of feature values on our sequence data can be seen in Table 2.

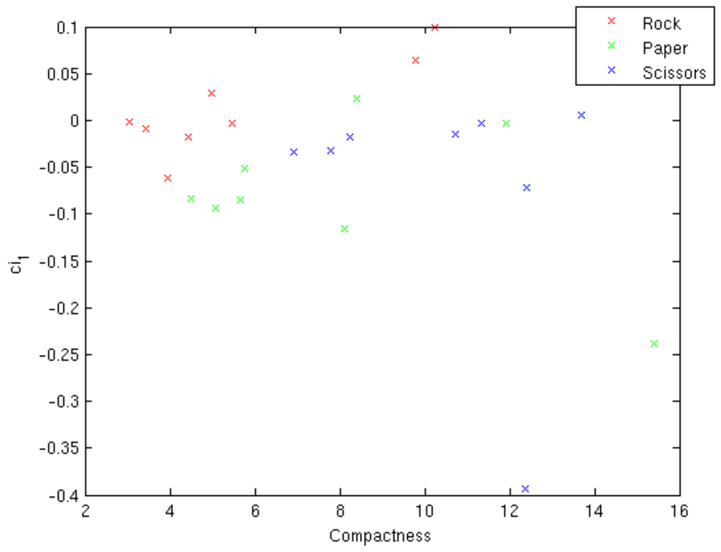

Figure 4: A plot of compactness and ci_1 for each MHI, coloured by class. The Rock motions are almost linearly separable, but the Paper and Scissors movements are harder to separate.

2.5 Classification

To test our system, we applied 8-fold cross-validation. In each round of testing,the 8 sets of samples from each class were split into 7 sets used for training the classifier, and 1 set for testing.

As there were only 7 training sets available for each iteration of testing, we selected the first 6 features from the vector to train the classifier. Note that when applying our system to a new set of inputs we will be able to use all 7 features, as we will have 8 training samples available for each class.

Classification was done using a Multivariate Gaussian classifier, with a mean and covariance calculated for each class of training data. The probability of each test sequence belonging to a certain class was then calculated, and the class with maximum probability selected. We also tried using a Naive Bayes based classifier, but found that it had poor performance, possibly due to the lack of independence between the features.

3 Results

Running the classifier on the validation set as described above gave us an accuracy of 83.3%, correctly classifying 20/24 of the images. The confusion matrix can be seen in Table 1.

We were able to determine likely causes for three of our four failing tests. In the two cases where a Paper or Scissor movement was confused for a Rock movement, the hand curled into a fist as it left the image. This likely led to our motion history image more closely resembling the other Rock sequences than the Paper or Scissors ones. In the sequence where we confuse a Paper motion for a Scissors one, the hand appears at an extreme angle, such that the bottom three fingers are in shadow and difficult to extract - this leaves only the top finger, and thus it is mistakenly identified as a Scissors movement. In the final case, where a Rock movement is mistaken for a Paper one, we were unable to determine the exact cause.

The first two cases indicate that it may be beneficial to attempt to ignore the majority of the hand, and concentrate only on the fingers, which might give clearer differences between each class (no fingers for Rock, five for Paper, and two for Scissors). The third case is caused by an already identified problem - extracting fingers in shadows - and requires improvements to our hand extraction algorithm.

		Prediction		
		Rock	Paper	Scissors
Actual	Rock	7	1	0
	Paper	1	6	1
	Scissors	1	0	7

Table 1: The confusion matrix for our validation testing.

4 Conclusion

We have produced a classifier that reliably extracts hand data from a sequence of images, and is able to classify the extracted data as one of three possible movements from the ‘Rock, Paper, Scissors’ game with reasonable accuracy. However, there are still areas for improvement. Examples include improving the initial hand extraction to reduce noise and more reliably extract single fingers, using skew-invariant moments instead of our naive and potentially error-prone ‘flipping’ method, and finding other features that can provide a clearer separation between classes.

Sample	**Compactness**	**ci_1**	**ci_2**	**ci_3**	**ci_4**	**ci_5**	**Class**
1.1	5.7486	-0.0518	0.2721	0.4202	-0.5678	-0.1505	Paper
1.2	8.1233	-0.1154	0.3713	-0.6698	-0.4953	0.1191	Paper
1.3	15.3900	-0.2389	-0.3910	1.1841	0.2574	0.1287	Paper
1.4	3.4394	-0.0095	0.3638	-1.3826	-0.3458	-1.0599	Rock
1.5	3.0631	-0.0019	0.4901	-1.2049	-0.4111	-1.5616	Rock
1.6	3.9476	-0.0620	-0.5582	-2.6181	2.1706	-5.3516	Rock
1.7	10.7246	-0.0146	-1.1437	-3.7701	1.8573	-9.3412	Scissor
1.8	12.3977	-0.0715	-0.2329	-1.4081	1.2622	-1.8987	Scissor
1.9	12.3832	-0.3934	-1.4838	8.4581	-3.9402	-0.4026	Scissor
2.1	11.3398	-0.0028	0.5097	-0.0355	-0.0915	0.2465	Scissor
2.2	8.4144	0.0232	0.1884	0.0165	0.0704	0.0024	Paper
2.3	9.7797	0.0649	0.2116	0.1308	0.0596	0.0428	Rock
2.4	13.7071	0.0053	0.2565	0.2279	-0.1089	-0.1094	Scissor
2.5	11.9272	-0.0029	-0.0829	0.0154	-0.0058	-0.0069	Paper
2.6	10.2403	0.0997	-0.0193	-0.0202	0.0430	-0.0015	Rock
3.1	4.9891	0.0297	-0.0000	0.0000	0.0000	-0.0000	Rock
3.2	5.4630	-0.0033	0.3400	-0.0368	-0.0734	0.1028	Rock
3.3	4.4218	-0.0181	0.4565	-0.3057	0.0593	-0.6613	Rock
3.4	8.2558	-0.0175	-0.3979	-0.7480	0.4779	-1.5390	Scissor
3.5	6.9117	-0.0343	-0.2309	-0.3903	0.3987	-0.4808	Scissor
3.6	7.7937	-0.0321	0.3952	-0.7507	0.2581	-1.7101	Scissor
3.7	5.6740	-0.0851	0.4283	0.1079	-0.8817	-0.2967	Paper
3.8	4.5124	-0.0833	0.5478	0.0140	-0.9296	-0.0623	Paper
3.9	5.0929	-0.0942	0.9989	-1.0572	-0.2074	0.9316	Paper

Table 2: The feature values for the 24 samples.

References

[1] A. Bobick and J. Davis, "*The Representation and Recognition of Action Using Temporal Templates*", IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(3):257-267.

[2] Introduction to Vision and Robotics, Lecture 5.
`http://www.inf.ed.ac.uk/teaching/courses/ivr/lectures/ivr5.pdf`

[3] M-K. Hu, "*Visual Pattern Recognition by Moment Invariants*", IRE Transactions on Information Theory, IT-8:pp. 179-187, 1962.

A Code Listing

A.1 run_validation.m

```
% This script performs an evaluation of our system, by extracting
% features from the training data and then performing an 8-fold cross
% validation evaluation on the features.

%% Setup

% Debug mode switch.
display_mode = 0;

% Average the backgrounds
bg1 = imread('../backgrounds/background1.jpg');
bg2 = imread('../backgrounds/background2.jpg');
alpha = 0.2;
average_bg = (alpha * bg1) + (1 - alpha) * bg2;

% The training directories.
dirs = dir(fullfile('..', 'train', '*-*'));

% The sequence classes: rock = 1, paper = 2, scissors = 3.
classes = [ 2 2 2 1 1 1 3 3 3 3 2 1 3 2 1 1 1 1 3 3 3 2 2 2 ];

%% Feature Extraction

disp('Beginning feature extraction.');

features = extract_features('train' , dirs, average_bg, display_mode);

disp('Completed feature extraction');

%% Feature processing

disp('Pre-processing the features for the classifier.');

% How many features to use in the classifier.
NUM_FEATURES = 6;

reduced_features = features(1:24, 1:NUM_FEATURES);

% Tie the training data to their classes, and sort them in order of class.
reduced_features = [reduced_features, classes'];
reduced_features = sortrows(reduced_features, NUM_FEATURES + 1);

disp('Features processed.');
```

```
%% Testing on the validation set

disp('Beginning testing.');

% The number of tests to run.
NUM_TESTS = 8;

confusion_matrix = zeros(3, 3);

% Testing using 8-fold cross validation: for each test, three of the
% training sequences are extracted and used to test a classifier trained
% on the remaining twenty-one sequences.
for i = 1 : NUM_TESTS
    % Select one sequence each from the the rock, paper and scissor
    % classes, to use as validation data.
    test_rows = [i, NUM_TESTS + i, (2 * NUM_TESTS) + i];

    % Remove the validation data from the training data.
    feature_train = reduced_features;
    feature_train(test_rows, :) = [];

    % Train a classifier.
    [means, covs] = train_classifier(feature_train, 3);

    validation_data = reduced_features(test_rows, 1:NUM_FEATURES);

    % Test the classifier on the validation data.
    [confidence, output_classes] = test_classifier(validation_data, ...
        means, covs);

    % Update the confusion matrices. The output classes should be
    % [1 2 3] for each iteration.
    for j = 1 : 3
        confusion_matrix(j, output_classes(j)) = ...
            confusion_matrix(j, output_classes(j)) + 1;

        if output_classes(j) == j
            disp(['RIGHT: Confidence ' num2str(confidence(j))]);
        else
            disp(['WRONG: Confidence ' num2str(confidence(j))]);
        end
    end
end

disp('Finished testing.');
disp('Results:');
confusion_matrix %#ok<NOPTS>
disp('Accuracy:')
disp(strcat(num2str(sum(diag(confusion_matrix))), ' / 24'));
```

A.2 extract_features.m

```
function [ features ] = extract_features( prefix, dirs, background, display )
%EXTRACT_FEATURES Extracts the features from a set of image sequences.
%   Given a prefix path and a set of 1+ directories at that location,
%   returns a n-by-7 matrix, where each row i is a set of features for the
%   image sequence in the ith directory.
%
%   The features returned are the compactness and the 6 rotation invariant
%   moments. See www.inf.ed.ac.uk/teaching/courses/ivr/lectures/ivr5.pdf.
%
%   If the display parameter is set to 1, shows the individual hand frames,
%   bounded-box sequences, and motion history images for every sequence.

features = zeros(length(dirs), 7);
for i = 1 : size(dirs,1)
    files = dir(fullfile('..', prefix, dirs(i).name, '*.jpg'));
    num_files = size(files, 1);

    % Load each image in and attempt to extract the hand.
    frames = cell(1, num_files);
    for j = 1 : num_files
        tmp = imread(fullfile('..', 'train', dirs(i).name, files(j).name));

        frames{j} = extract_hand(tmp, background);

        % In debug mode, show the individual extracted-hand frames.
        if display
            imshow(frames{j});
            pause;
        end
    end

    % Find the average bounding box of the sequence, and if the sequence
    % needs to be flipped.
    [xmin xmax ymin ymax should_flip] = get_bounding_box(frames);

    % In debug mode, show a video of the sequence with the bounding box.
    if display
        for j = 1 : length(frames)
            image = frames{j};
            if should_flip
```

```
                image = flipdim(im, 2);
            end

            imshow(image);
            hold on;
            rectangle('Position', [xmin ymin xmax-xmin ymax-ymin], ...
                'LineWidth', 4, 'EdgeColor', 'r');
            hold off;
            pause(0.1);
        end
        pause
    end

    % Crop the sequence to the bounding box.
    cropped_frames = {};
    k = 1;
    for j = 1 : length(frames)
        image = frames{j};

        % All sequences are normalised (via mirroring) to be right-handed.
        if should_flip
            image = flipdim(image,2);
        end

        % Crop each image to the bounding box, and remove empty images.
        found = find(image == 1, 1);
        if (~isempty(found))
            newim = image(ymin:ymax, xmin:xmax);
            cropped_frames{k} = newim; %#ok<AGROW>

            k = k + 1;
        end
    end

    mhi = compute_motion_history(cropped_frames);

    % In debug mode, show the motion history image.
    if display
        imshow(mhi);
        pause;
    end

    features(i,:) = compute_mhi_features(mhi);
end

end
```

A.3 extract_hand.m

```
function [ result ] = extract_hand(image, bg)
%EXTRACT_HAND Extracts the hand from an image.
%   Does bg sub against an optional bg (loads default otherwise),
%   thresholds, etc. Returns a matrix the same size as the image but
%   1/0 for features.

% Matching on just the red colour spectrum produces better results.
image(:, :, 2) = 0;
image(:, :, 3) = 0;

% Apply background subtraction.
difference = rgb2gray(image - bg);

% Saturate the image (improves the capture of the scissors movement
% and some rock movements.)
difference = imadjust(difference);

% Threshold to a binary image.
threshold = graythresh(difference);
difference = im2bw(difference, threshold);

% Now erode/refill the image to clean it.
struct_elem = strel('octagon', 3);
difference = imerode(difference, struct_elem);
difference = imdilate(difference, struct_elem);
difference = imdilate(difference, struct_elem);
difference = imerode(difference, struct_elem);

% Locate the biggest item.
label = bwlabel(difference, 4);
properties = regionprops(label, 'Area'); %#ok<MRPBW>
biggest_area = max([properties.Area]);
index = find([properties.Area] == biggest_area);

result = ismember(label, index);

end
```

A.4 get_bounding_box.m

```
function [ xmin, xmax, ymin, ymax, flipped ] = get_bounding_box( sequence )
%GET_BOUNDING_BOX Extracts the average bounding box for a set of images.
%   Returns the bounding box that covers the average detected objects in
%   a sequence of images. Assumes all images in a sequence are the same
%   size. Also determines if an image is left-handed and thus needs to be
%   flipped.

sequence_length = length(sequence);

image_width = size(sequence{1}, 2);

image_middle = image_width / 2;

% 1/5th of the number of images.
x = int8(sequence_length / 5);

flipped = 0;

[ymax, ymin, xmax, xmin, avgxs] = get_bbox(flipped);

% If the entering and exiting x values are to the left of the middle,
% the image likely needs to be flipped.
len = length(avgxs);
first_last = [avgxs(1:x); avgxs(len - x:len)];
flipped = sum(first_last < image_middle) == length(first_last);
if (flipped)
    [ymax, ymin, xmax, xmin] = get_bbox(flipped);
end

    function [ymax, ymin, xmax, xmin, avgxs] = get_bbox(flipped)
        % Finds the maximum and minimum x and y values for every image,
        % and the set of average x values as well.
        sequence_info = zeros(sequence_length, 6);
        for i = 1 : length(sequence)
            image = sequence{i};
            if (flipped),
                image = flipdim(image,2);
            end
            [row, col, ~] = find(image == 1);
            if (length(row) > 1),
                [ymax, ymin, xmax, xmin, avgy, avgx] = ...
                    findmaxmins(row, col);
                sequence_info(i,:) = [ ymax, ymin, xmax, xmin, avgy, avgx ];
            end
        end

        % Ignore any rows whose centre point is too far away from the
        % average centre point. This is done to ignore the effects of
        % 'noise' frames.
        means = mean(sequence_info);
        avgx = means(6);
        thresh = 200;
        sequence_info(find(abs(sequence_info(:,6) - avgx) > thresh),:) = [];

        % Find the overall maximum and minimum x and y values.
        maxes = max(sequence_info);
        ymax = maxes(1);
        xmax = maxes(3);

        mins = min(sequence_info);
        ymin = mins(2);
        xmin = mins(4);

        avgxs = sequence_info(:,6);

    end

    function [ymax, ymin, xmax, xmin, avgy, avgx] = findmaxmins(row, col)
        % Finds the maximum, minimums, and averages for a single
        % image.
        ymax = max(row);
        ymin = min(row);
        xmax = max(col);
        xmin = min(col);
        avgy = mean(row);
        avgx = mean(col);
    end
end
```

A.5 compute_motion_history.m

```
function [ output_image ] = compute_motion_history( sequence, tau )
%COMPUTE_MOTION_HISTORY Computes the MHI for a sequence of images.
%   Assumes all images are the same size.
%
% The motion history image is computed as:
%
%   H_tau(x, y, t) = tau if D(x,y,t) = 1, else max(0, H_tau(x,y,t-1) - 1)
%
% D(x,y,t) is the binary image that shows motion - here we assume that
% each frame counts as showing motion.

% By default, tau is the total number of frames.
if nargin < 2
    tau = length(sequence);
end

output_image = zeros(size(sequence{1}));
```

```
% For each pixel in the image, tau is the frame number where the
% pixel last appeared as a 1.
pixels_in_frame = find(sequence{tau} == 1);
indices = [pixels_in_frame , ones(size(pixels_in_frame)) * tau ];
for frame = tau - 1 : -1 : 1,
    pixels_in_frame = setdiff(find(sequence{frame} == 1), indices(:,1));
    new_set_indices = ...
        [pixels_in_frame, ones(size(pixels_in_frame)) * frame ];
    indices = [indices ; new_set_indices ]; %#ok<AGROW>
end

output_image(indices(:,1)) = indices(:,2);
output_image = mat2gray(output_image);

end
```

A.6 compute_mhi_features.m

```
function mid = compute_mhi_features ( image )
%COMPUTE_MHI_FEATURES Computes the features for an MHI.
%   The features returned are the compactness and the 6 rotation invariant
%   moments described in IVR (lecture 5.)

area = sum(sum(image));
perimeter = bwarea(bwperim(image, 8));
compactness = perimeter^2 / (4 * pi * area);

% The centre of mass calculation is weighted with regards to the
% MHI value at each pixel.
mean_img = [0, 0];
for i = 1 : size(image, 1),
    for j = 1 : size(image, 2),
        mean_img = [ mean_img(1) + (i * image(i, j)) , ...
                     mean_img(2) + (j * image(i, j)) ];
    end
end
mean_img = [mean_img(1) / area, mean_img(2) / area];

% Calculate the complex central moments, adapted for a MHI.

% Compute (r - r_m), for all values of r.
rcs = repmat(1:size(image, 1), size(image, 2), 1)';
rcs = rcs - mean_img(1);

% Compute i(c - c_m), for all values of c.
sqrts = sqrt(-1) * ((1:size(image, 2)) - mean_img(2));
cs = repmat(sqrts, size(image, 1), 1);
```

```
% Compute (r - r_m) +- i(c - c_m), for all values of r and c.
% For some reason, using only (r - r_m) + i(c - c_m) makes the
% classifier a lot more accurate.
rcs_p = rcs + cs;
rcs_n = rcs + cs;

% Calculate the scale invariant moments.
s11 = find_c(1,1) / area^2;
s20 = find_c(2,0) / area^2;
s21 = find_c(2,1) / area^2.5;
s12 = find_c(1,2) / area^2.5;
s30 = find_c(3,0) / area^2.5;

% Calculate the rotation invariant moments.
ci = zeros(6, 1);
ci(1) = real(s11);
ci(2) = 1000 * real(s21 * s12);
ci(3) = 10000 * real(s20 * s12 * s12);
ci(4) = 10000 * imag(s20 * s12 * s12);
ci(5) = 1000000 * real(s30 * s12 * s12 * s12);
ci(6) = 1000000 * imag(s30 * s12 * s12 * s12);

mid = [compactness; ci];

    function c0 = find_c(u,v)
        rc_u = rcs_p.^u;
        rc_v = rcs_n.^v;
        rc_uv = rc_u .* rc_v;
        rc_uv = rc_uv .* image;
        c0 = sum(sum(rc_uv));
    end
end
```

A.7 train_classifier.m

```
function [means, covariances] = train_classifier(features, num_classes)
%TRAIN_CLASSIFIER Trains a multivariate gaussian classifier.
%  Computes the means and covariances for a set of features

means = cell(1, num_classes);
covariances = cell(1, num_classes);

% The dimensionality is size - 1, as the last column of the features
% is the class.
dim = size(features, 2) - 1;
```

```
num_examples = size(features, 1) / num_classes;

% For each class, compute the means of the features for that
% class.
for i = 1 : num_classes,
    start_point = (num_examples * (i - 1)) + 1;
    class_features = ...
        features(start_point:start_point + (num_examples - 1), 1:dim);

    means{i} = mean(class_features);
    covariances{i} = cov(class_features);
end

end
```

A.8 test_classifier.m

```
function [confidence, classes] = test_classifier(features, means, covs)
%TEST_CLASSIFIER Classifies a set of features via a multivariate gaussian.
%  The gaussian is parameterised by a set of means and covariances, with
%  one parameter set per class.

num_tests = size(features, 1);

num_classes = size(means, 2);
num_features = size(features, 2);

classes = zeros(1, num_tests);
confidence = zeros(1, num_tests);
for j = 1 : num_tests
        probs = zeros(1,3);
        feature = features(j, 1:num_features);

        % For each class, calculate the probability based on a
        % multivariate gaussian.
        for i = 1 : num_classes
            diff = feature - means{i};
            cov = covs{i};
            prob = 1 / sqrt(det(2 * pi * cov));
            mantissa = -0.5 * (diff * (cov \ diff'));
            prob = prob * exp(mantissa);
            probs(i) = prob;
        end

        % Select the class with the maximum probability.
        [p, class] = max(probs);
        classes(j) = class;
        confidence(j) = p / sum(probs);
end

end
```