Vikram Voleti

09EE3501

Summer 2012 Internship Report

Department of Electrical Engineering

Indian Institute of Technology Kharagpur

TABLE OF CONTENTS

1. Introduction

During my Summer Internship, I worked under Prof. Aurobinda Routray, IIT Kharagpur, on the project "Fatigue Recognition in Human Drivers." My contribution to the project has been to write MATLAB codes for the algorithms used to implement Hidden Markov Models. I used various papers as references for these algorithms, the most important paper I followed being "A Tutorial on Hidden Markov Models" by Lawrence R. Rabiner.

The list of codes I have written is:

- Hidden Markov Model
  - Forward Algorithm
  - Backward Algorithm
  - Baum-Welch Algorithm
  - Viterbi Algorithm
- Finger-Tip Gesture Recognition [Triangle, Square or Diamond]

All the codes written are original, and have been optimised to the best of my abilities. As far as possible, mathematical operations have been converted to matrix operations to speed things up; only the most general MATLAB functions such as `sum`, `for` loops, have been used while functions like `mean`, and more complicated ones have been avoided to decrease running time; the least number of variables has been declared to save space; a reference list explaining all the variables used in the code, including their dimensions and a description, has been mentioned in each code to increase user-friendliness; an example code has also been added to show how to use the code; comments have been added for easier understanding.

In order to demonstrate the correctness of the codes, I created a MATLAB program called "Finger-Tip Gesture Recogniser" that could recognise gestures drawn by the tip of a finger. This code tracks the movement of a finger-tip, and at the end displays the name of the shape drawn: a Triangle, a Square, or a Diamond. Hidden Markov Models were used to convert the observations, which were the finger-tip coordinates, to the hidden states, which were Triangle, Square and Diamond.

A detailed description of the codes is given below. The motive was to create original, optimised, and user-friendly codes for Hidden Markov Models in MATLAB. These codes shall be utilised in carrying forward research in the project by other individuals.

2. Hidden Markov Model

a. Introduction

A Markov State Sequence is a sequence of states where the probability of the occurrence of any state as the next state does not depend on any of the previous states, other than the current state. These states give a certain output.

A Hidden Markov Model gives a model of a state sequence where it is not known which state is occurring, i.e. the states are hidden. The only thing that is known is the set of observations. The various variables used in describing an HMM are:

N = total number of states,
M = total number of possible observations,
T = total number of observations made (for each observation set),
EG = total number of examples/observation sets recorded.

A Hidden Markov Model is described by three parameters:

1. Pi: the initial probability matrix. Pi contains information about which state occurs at the beginning, i.e. at the start of observation.
2. a: the transition probability matrix. a contains information about what state will occur next, given the current state. Since the total number of states is N, a is an NxN matrix. a(i,j) is equal to the probability of the next state being S_j, given the present state is S_i.
3. b: the emission matrix. b contains information about the probability of the occurrence of a certain observation, given the state the system is in. Since there are N possible states and M possible observations, b is an NxM matrix. b(i,j) is equal to the probability of the occurrence of observation Oj, given the system is in state S_i.

In addition to these, there are two very important quantities that are calculated when dealing with Hidden Markov Models: Alpha and Betaa (in order to avoid confusion with the MATLAB in-built "Beta"). These are described below.

Hence, the different matrices used are:

Pi : Nx1 matrix : Vector of initial probabilities of states,
a : NxN matrix : Probability of transition from state S_i to state S_j,
b : NxM matrix : Probability of observing V_k for state S_i,
Ob : Mx1 matrix : Vector of all possible observations,
O : EGxT matrix : Matrix of EG no. of 1xT dimensional observation sets,
Alpha : NxT matrix; Alpha(i,t) = Probability of observing partial observation sequence from start to time t, i.e. $O_1, O_2, ..., O_t$, and being in state S_i, at time t,
Betaa : NxM matrix; Betaa(i, find(Ob==O(eg,t),1)) = Probability of partial observation sequence from t+1 to end, i.e. $O_{t+1}, ..., O_T$, given the state at time t was S_i.

Alpha and Betaa are used in the following algorithms to find out the parameters of an HMM, or to determine the state sequence from the observation set.

b. Forward Algorithm

The Forward Algorithm calculates the quantity `Alpha`, given the inputs `Pi`, `a`, `b`, `Ob` and `O`. `Alpha(i,t)` is the probability of observing partial observation sequence from start to time t, i.e., $O_1, O_2, ..., O_t$, and being in state S_i, at time t. The function call is:

```
[Alpha, c, P] = ForwardAlgo(Pi, a, b, Ob, O)
```

where `c` is a `Tx1` matrix, `c(t)` = Probability of the partial observation sequence till time `t`, and `P` is an `Nx1` matrix, `P(n)` = Probability of being in state S_n at the end time `T`.

```
function [Alpha, c, P] =
ForwardAlgo(Pi, a, b, Ob, O)

%% Forward Algorithm

%% Setting up matrices and variables

% Pi: Nx1 matrix : Vector of initial
probabilities of states
% a: NxN matrix : Prob. of transition
from state Si to state Sj
% b: NxM matrix : Prob. of observing Vk
for state Si
% Ob : Mx1 matrix : Vector of all
possible observations
% O : EGxT matrix : Matrix of EG no. of
1xT dimensional observation sets

N = size(a,1);
M = size(Ob,1);
T = size(O,2);
EG = size(O,1);

c = zeros(T,1); cNew = c;
Alpha = zeros(N,T); AlphaNew = Alpha;

%% Adding up the normalised Alpha
values for each example found using the
Forward Algo

for eg = 1:EG %for loop to run through
all the examples

%% Initialization

% Alpha: NxT matrix
% Alpha(i,t) = Probability of the
partial Obs. seq. O1,O2,...,Ot, and
state Si at t
% Alpha(i,1) = Pi(i) * b(i,find(Ob ==
O(eg,1),1))

AlphaNew(:,1) = Pi .* b(:,
Ob==O(eg,1));

%Normalising AlphaNew and adding to the
Alpha matrix:
cNew(1) = sum(AlphaNew(:,1));
Alpha(:,1) = Alpha(:,1) +
AlphaNew(:,1)/cNew(1);

%% Induction

% Alpha(j,t) = ( sum w.r.t. i
(Alpha(i,t-1)*a(i,j)) ) * b(j,
find(Ob==O(eg,t),1))

for t = 2:T
    for j = 1:N
        AlphaNew(j,t) =
sum(AlphaNew(:,t-1).*a(:,j)) * b(j,
Ob==O(eg,t));
    end
    cNew(t) = sum(AlphaNew(:,t));

    %Normalising AlphaNew(:,t) and
adding it to Alpha(:,t)
      Alpha(:,t) = Alpha(:,t) +
AlphaNew(:,t)/cNew(t);
end

c = c + cNew;

%% Termination

% P(O|parameters) = sum w.r.t. i of:
Alpha(i,T)

%P = sum(Alpha(:,T));
P = Alpha(:,T);

end %end of for loop running through
all the examples

%% Taking avg of all the Alpha's by
diving by number of examples

Alpha = Alpha/EG;

end
```

c. Backward Algorithm

The Backward Algorithm calculates the quantity `Betaa` (in order to avoid confusion with the MATLAB built-in "Beta"), given the inputs `Pi`, `a`, `b`, `Ob` and `O`. `Betaa(i, find(Ob==O(eg,t),1))` is the probability of observing the partial observation sequence from time `t+1` to `end`, i.e. $O_{t+1}, \ldots, O_T$, given the state at time `t` was S_i . The function call is:

```
Betaa = BackwardAlgo(Pi, a, b, Ob, O)
```

```
function [Betaa] = BackwardAlgo(Pi, a, b, Ob, O)

%% Backward Algorithm

%% Setting up matrices and variables
% a: NxN matrix = Prob. of transition from state Si to state Sj
% b: NxM matrix = Prob. of observing Vk for state Si
% Ob : Mx1 matrix : Vector of all possible observations
% O : EGxT matrix : Matrix of EG no. of 1xT-dimensional observation sets

N = size(a,1);
M = size(Ob,1);
T = size(O,2);
EG = size(O,1);

Betaa = zeros(N,T); BetaNew = Betaa;

%% Adding up the normalised Beta values for each example, found using the Backward Algo

for eg = 1:EG %for loop to run through all the examples


%% Initialization

% Beta: NxT matrix
% Beta(i,find(Ob==O(eg,t),1)) = Probability of partial obs. seq. from t+1 to end, given state Si at t
% Beta(i,T) = 1

%{
for i=1:N
    Betaa(i,T) = 1;
end
%}

BetaNew(:,T) = 1;
Betaa(:,T) = Betaa(:,T) + BetaNew(:,T);


%% Induction

% Beta(i,t) = ( sum w.r.t. j a(i,j)*b(j,find(O(t+1)))*Beta(j,t+1) )

for t = (T-1):-1:1
    BetaNew(:,t) = a*( b(:,Ob==O(eg,t+1)).*BetaNew(:,t+1) );
    %Normalising BetaNew(:,t) and adding it to Beta(:,t)
    c(t) = sum(BetaNew(:,t));
     Betaa(:,t) = Betaa(:,t) + BetaNew(:,t)/c(t);
end


end %end of for loop running through all the examples

%% Finding average of Beta by diving by number of examples

Betaa = Betaa/EG;

end %function end
```

d. Baum-Welch Algorithm

The Baum-Welch Algorithm is used to train the Hidden Markov Model using training data, i.e. to determine the values of the parameters of the Hidden Markov Model, viz. `Pi` : the initial probability matrix, `a` : the transition probability matrix, and `b` : the emission matrix. The training data is in the form of the observation sets `O`.

Initially, the Baum-Welch algorithm requires certain (generally random) initial values for these matrices, and the values of `Alpha` and `Betaa` determined using these initial values. In each iteration of the Baum-Welch algorithm, it calculates new values for `Pi`, `a` and `b`, `Alpha` and `Betaa`. It iterates its algorithm for a total of `maxIters` times to get the best possible values that concur with the training data `O`. As the algorithm iterates, `iters` is the variable used to store the current iteration number. `PiNew`, `aNew`, `bNew`, `AlphaNew` and `BetaaNew` are the new values of `Pi`, `a`, `b`, `Alpha and Betaa`, respectively, calculated in the next iteration. `oldLogProb` is a variable used to store the log-likelihood which describes how correct the current values for `Pi`, `a` and `b` are. This shall be used to compare with `logProb`, the same quantity calculated with the new values `PiNew`, `aNew` and `bNew`, to check if the algorithm has converged. Finally, lP is the matrix used to store all the values of `logProb` calculated in each iteration.

As an example, the initialisation of variables is:

```
Pi = rand(N,1); %random initialisation
Pi = Pi/sum(Pi); %ensuring all values sum to 1
a = rand(N,N); %random initialisation
a = a./repmat(sum(a,2), [1 N]); %ensuring every row sums to 1
b = rand(N,M); %random initialisation
b = b./repmat(sum(b,2), [1 M]); %ensuring every row sums to 1


iters = 0;
maxIters = 100;
oldLogProb = -Inf;
lP = zeros(0,1);
Alpha = ForwardAlgo(Pi, a, b, Ob, O);
Betaa = BackwardAlgo(Pi, a, b, Ob, O);
```

The function call is:

```
[PiNew, aNew, bNew, AlphaNew, BetaNew, logProb, lP] =
BaumWelsh(Pi, a, b, Ob, O, Alpha, Betaa, iters, maxIters,
oldLogProb, lP);
```

```
function [PiNew, aNew, bNew, AlphaNew, BetaNew, logProb, lP] =  BaumWelsh(Pi, a, b, Ob, O, Alpha, Betaa, iters, maxIters, oldLogProb, lP)

%% Reestimation of parameters using Baum-Welch method, or the EM method

% Pi : Nx1 matrix : Vector of initial probabilities of states
% a : NxN matrix : Prob. of transition from state Si to state Sj
% b : NxM matrix : Prob. of observing Vk for state Si
% Ob : Mx1 matrix : Vector of all possible observations
% O : EGxT matrix : Matrix of EG no. of 1xT dimensional observation sets
% Alpha : NxT matrix; Alpha(i,t) = Probability of the partial Obs. seq. O1,O2,...,Ot, and state Si at t
% Betaa : NxM mmatrix; Betaa(i,find(Ob==O(eg,t),1)) = Prob. of partial obs. seq. Ot+1,...,OT, given state Si at t, in example eg

N = size(a,1);
M = size(Ob,1);
T = size(O,2);
EG = size(O,1);

% 1)Loop though all the examples, add up all the normalised Pi, a and b found,
% 2)and finally take their average (divide them by number of examples)

PiNew = zeros(N,1);
aNew = zeros(N,N); aN = zeros(N,N);
bNew = zeros(N,M); bN = zeros(N,N);

%% 1)Loop though all the examples, add up all the normalised Pi, a and b found
for eg = 1:EG

%% Xi: NxNxT matrix
% Xi(i,j,t) = Probability of being in state Si at t and Sj at t+1
% denominator = sum thru i: sum thru j : Alpha(i,t) * a(i,j) * b(j,find(O(eg,t+1))) * Betaa(j,t+1)
% Xi(i,j,t) = ( Alpha(i,t) * a(i,j) * b(j,find(O(eg,t+1))) * Betaa(j,t+1) )/denominator
Xi = zeros(N,N,T);

for t = 1:(T-1)
    den = sum( Alpha(:,t) .* ( a * ( b(:,Ob==O(eg,t+1)).*Betaa(:,t+1)) ) );
    Xi(:,:,t) = a.*((b(:,Ob==O(eg,t+1)).*Betaa(:,t+1))*Alpha(:,t)')'/den;
end


%% Gamma: NxT matrix
% Gamma(i,t) = Probability of being in state Si at t
% Gamma(i,t) = sum thru j: xi(i,j,t)
Gamma = reshape(sum(Xi,2), [size(Xi,1) size(Xi,3)]);
Gamma(:,T) = Alpha(:,T).*Betaa(:,T)/sum(Alpha(:,T).*Betaa(:,T));


%% Parameters

% Pi: Nx1 matrix
% Pi(i) = Expected number of times in state Si at t = gamma(i,1)
sumGamma = sum(Gamma(:,1)); %for normalisation
%Normalising Gamma(:,1) and adding it to PiNew:
PiNew = PiNew + Gamma(:,1)/sumGamma;

% a: NxN matrix
% a(i,j) = Prob. of transition from state Si to state Sj
% a(i,j) = (Expected no. of transitions from Si to Sj)/(Expected no. of transitions from Si)
%        = (sum w.r.t. t thru 1:(T-1) of:  Xi(i,j,t))/(sum w.r.t. t thru 1:(T-1) of: Gamma(i,t))
numr = sum(Xi(:,:,1:(T-1)),3);
sumG = (sum(Gamma(:,1:(T-1)),2)); sumG = repmat(sumG, 1, N);
aN = numr./sumG;
%Normalising aN and adding it to aNew:
aNewSum = sum(aN,2); aNewSum = repmat(aNewSum, 1, N);
```

```
aNew = aNew + aN./aNewSum;

% b: NxM matrix
% b(i,m) = Prob. of observing Vm for
state Si
% b(i,m) = (Expected no. of times in
state Si and obs. Vm)/(Expected no. of
time in state Si)
%        = (sum w.r.t. t of: Gamma(i,t)
s.t. Ot = Vm)/(sum w.r.t t of:
Gamma(i,t))
for m = 1:M
      bN(:,m) =
(Gamma*(O(eg,:)==Ob(m))')./sum(Gamma,2)
;
end
%Normalising bN and adding it to bNew:
bNewSum = sum(bN,2); bNewSum =
repmat(bNewSum, 1, M);
bNew = bNew + bN./bNewSum;



%%
end % end of for loop going throught
the examples

PiNew = PiNew/EG;
aNew = aNew/EG;
bNew = bNew/EG;

%% Finding new Alpha and Betaa

[AlphaNew, c] = ForwardAlgo(PiNew,
aNew, bNew, Ob, O);

BetaNew = BackwardAlgo(PiNew, aNew,
bNew, Ob, O);



%% LogProb

iters = iters + 1;

% Log likelihood = sum thru t
(log(sum(alpha(:,t)))
logProb = sum(log(c));
lP = [lP; logProb];

fprintf('iter# %d  oldLogProb %f
logProb %f\n', iters, oldLogProb,
logProb);

if iters < maxIters
    diff = abs(logProb - oldLogProb);
    avg = (abs(logProb) +
abs(oldLogProb) + eps)/2;
    if diff/avg < 1e-4
        disp('converged');
        return;
    else
        oldLogProb = logProb;
        [PiNew aNew bNew AlphaNew
BetaNew logProb lP] = BaumWelsh(PiNew,
aNew, bNew, Ob, O, AlphaNew, BetaNew,
iters, maxIters, oldLogProb, lP);
    end
end



end %function end
```

e. Viterbi Algorithm

To determine the most probable state sequence `Q`, given `Pi`: the Initial Probability matrix, `a`: Transition matrix, `b`: Emission matrix, `Ob`: matrix of Possible Observations, and `O`: matrix of observations (one observation set). The function call is:

```
Q = ViterbiAlgo(Pi, a, b, Ob, O)
```

```
function [Q] = ViterbiAlgo(Pi, a, b,
Ob, O)

% Viterbi Algorithm
%   To determine the most probable
state sequence Q,
%   given Pi: the Initial Probability
matrix, a: Transition matrix,
%   b: Emission matrix, Ob: matrix of
Possible Observations, and
%   O: matrix of observations (one
observation set)

%% Setting up matrices and variables
% Pi: Nx1 matrix : Vector of initial
probabilities of states
% a: NxN matrix : Prob. of transition
from state Si to state Sj
% b: NxM matrix : Prob. of observing Vk
for state Si
% Ob : Mx1 matrix : Vector of all
possible observations
% O : Txr matrix : Matrix of T r-
dimensional observations (only 1
observation set)

N = size(a,1);
M = size(Ob,1);
T = size(O,1);
r = size(O,2);

% delta: NxT matrix : delta(i,t) =
Highest prob. along a path for first t
obs. and ending in Si
Delta = zeros(N,T);

% psi: NxT matrix : argmax(delta)
Psi = Delta;

% Q: Tx1 matrix: vector of the most
probable state sequence
Q = zeros(T,1);
bprod = ones(N,T);
for t = 1:T
    for ri = 1:r
        bprod(:,t) = bprod(:,t) .*
b(:,Ob(:,ri)==O(t,ri),ri);
    end
end

%% Initialization
%Delta(:,1) = Pi .* b(:, Ob==O(1));
%Delta(:,1) = Pi .* b_1(:,
find(Ob==O(1,1))).*b_2(:,
find(Ob==O(1,2)));
Delta(:,1) = Pi .* bprod(:,1);
Psi(:,1) = 0;

%% Recursion

% delta(j,t) = max w.r.t. i  delta(i,t-
1) * a(i,j) * b(j,find(O(t)))
% psi(j,t) = argmax w.r.t. i
delta(i,t-1) * a(i,j) * b(j,find(O(t)))

for t=2:T
    for j=1:N
        [Delta(j,t) Psi(j,t)] = max(
Delta(:,t-1) .* a(:,j) * bprod(j,t) );
    end
end


%% Termination
[~, Q(T)] = max(Delta(:,T));

%% Path back-tracking

for t=(T-1):-1:1
    Q(t) = Psi(Q(t+1),t+1);
end

end %function end
```

3. Finger-Tip Gesture Recognition [Triangle, Square, Diamond]

Using the MATLAB codes for Hidden Markov Models, I have developed a Gesture Recognition code which tracks the tip of a finger and recognises the gesture made by it as either a `Triangle,` or a `Square,` or a `Diamond` (or that it doesn't know). A detailed explanation of each step is explained in the code itself.

a. Finger-Tip Recognition

The program takes video input from the webcam of the console it is working in. The tip of a finger is identified by a clever algorithm involving skin-pixel segmentation, and a priority search in the skin area.

Details: An area is defined where the gesture can be made, and within that area all the pixels that have their RGB values as those of skin are classified as skin pixels. The RGB values of each pixel in the frame captured by the camera are converted into HSI (Hue-Saturation-Intensity) values. Generally, Hue values lie in the range 0° to 360°, Saturation values lay within 0 and 100, and Intensity between 0 and 1. These values for skin colours are limited to 0° to 60° Hue, and 10 to 40 Saturation. In this way, a binary mask is created which shows skin pixels from non-skin pixels. In this mask, the first pixel from the top is identified as the finger-tip.

b. Finger-Tip Tracking

The coordinates of the finger-tip in each frame are appended to a matrix, hence keeping track of the motion of the finger-tip. So in addition to displaying every frame captured on the screen as an image, I have marked all the previous coordinates of finger-tip image in red, and the current coordinates in yellow. Thus, the movement of the finger-tip is also displayed on screen.

c. Gesture Recognition

Once all the coordinates of the finger-tip have been stored, the gesture drawn is identified by using Hidden Markov Models. The input is the matrix of all the coordinates, and the output is the hidden state the coordinates belong to: Triangle, Square or Diamond.

Details: Firstly, the observations are converted to difference of current and previous coordinates. Thus, instead of the actual coordinates, our observations are the change in coordinates. We are checking whether the values of the x-coordinate and y-coordinate have increased or decreased. I have used two cascaded HMM's for gesture recognition.

The first HMM recognises the sequence of directional lines made by the finger-tip. The input is the sequence of changes in finger-tip coordinates, and the output for each observation is one of the eight possible directions the finger-tip can move. Thus the possible hidden states are `Right`, `Down`, `Left`, `Up`, `Down-Right`, `Down-Left`, `Up-Left` and `Up-Right.`

The second HMM recognises the shape made based on the sequence of directional lines obtained from the previous HMM, the possible shapes being `Triangle`, `Square` and `Diamond`. The input to this HMM is the output of the first HMM, and the output of this HMM is shape of the gesture recognised.

```
function [] =
FingerTipGestureRecognition()
%% MATLAB Code for Finger-Tip Gesture
Recognition using two Hidden Markov
Models

% Recognises the gestures Triangle,
Square, and Diamond,
% performed by the tip of a finger.
%   -> Please ensure the background
does not have skin-coloured portions.
%   -> First, a preview of the video to
be recorded is shown for about 2
%   seconds, along with a new blank
window kept open.
%   -> Position your finger only in the
left half of the preview video.
%   -> Once the words 'GO' appear on
the blank window, start your gesture.
%   -> The window will now show a
mirror image of the preview video,
along with
%   tracking of the fingertip.
%       Make note of the limits of the
window, and the half-line shown for
convenience.
%       Only perform the gesture in the
half your finger is in.
%   -> Perform the gesture for about 5
seconds.
%   -> The recognised gesture name
shall be displayed on the screen.
%   -> The command window can also be
checked to know the recognised gesture.

clear; close all;

% Matrix to store coordinates of
finger-tip (will be used subsequently)
fingercoord = zeros(0,2);

% Defining a kernel for blurring images
(will be used subsequently)
kernel = ones(5,5)/25;

% Setting up video
vid = videoinput('winvideo',1);
set(vid, 'ReturnedColorspace', 'RGB');

% Finding the size of the window frame
vidsize = get(vid, 'VideoResolution');
width = vidsize(1);
height = vidsize(2);

% Finding size of screen
screensize = get(0,'ScreenSize');

% Setting up a preview of the video to
be recorded
figure('name', 'preview', 'Position',
[1 screensize(4)/2 vidsize(1)
vidsize(2)]);
im = image(zeros(height, width, 3));
line([.5*width .5*width], [1 height],
'color', 'g', 'LineWidth', 2);
preview(vid, im);

% Setting up a the video to be recorded
as gesture
figure('name', 'RGBimage');
pause(2.5);
text(.3,.5,'GO', 'BackgroundColor' ,
'white', 'Color', 'black', 'FontName',
'Impact', 'FontSize', 100);
pause(.001);
tts('go');
pause(.3);

count = 0;
tic;

while(count<27)

% Getting image as double
RGBimage = im2double(getsnapshot(vid));
% Taking mirror-image
RGBimage =
RGBimage(:,size(RGBimage,2):-1:1,:);

%Convention of directions x and y in
image is:
%Inc_x:DOWN, Inc_y:RIGHT

% Blurring the image
blur = RGBimage;
blurred = conv2(RGBimage(:,:,1),
kernel);
blur(:,:,1) =
blurred(3:(size(blurred,1)-
2),3:(size(blurred,2)-2));
blurred = conv2(RGBimage(:,:,2),
kernel);
blur(:,:,2) =
blurred(3:(size(blurred,1)-
2),3:(size(blurred,2)-2));
blurred = conv2(RGBimage(:,:,3),
kernel);
blur(:,:,3) =
blurred(3:(size(blurred,1)-
2),3:(size(blurred,2)-2));
%figure('name', 'smoothed'),
imshow(blur);

% Determining h s i values for each
pixel
HSIimage = rgb2hsi(blur);
```

```
% Determining mask for skin-coloured
regions:
% 0 <= Hue(0:360) <= 60, 10 <=
Saturation(0:100) <= 40
mask = zeros(size(RGBimage(:,:,1)));
mask((HSIimage(:,:,1)>=0)&(HSIimage(:,:
,1)<=60)&...

(HSIimage(:,:,2)>=10)&(HSIimage(:,:,2)<
=40)) = 1;
blurred = conv2(mask, kernel);
mask = blurred(3:(size(blurred,1)-
2),3:(size(blurred,2)-2));
mask(mask<.5) = 0;
mask(mask>=.5) = 1;
%figure('name', 'mask'), imshow(mask);

% Determining mask for region of
gesturing as the right half of the
image
mask(:,1:.5*size(mask,2)) = 0;

% Finding the tip of the finger within
the mask created as the first white
pixel found from top
% in convention Inc_x:DOWN, Inc_y:RIGHT
[y x] = find(mask'>0,1);

% If the finger-tip is detected, and it
lies within 3 pixels of the
% boundary of the right half of the
image
if ~isempty(x) && y>.5*width+3 &&
y<width-3 && x>3 && x<height-3

    % Appending matrix 'fingercoord'
with new coordinates of first white
pixel
    fingercoord = [fingercoord; x y];

    % Showing image with path of first
white pixel
    fingerpath = blur;
    for i = 1:size(fingercoord,1)
        fingerpath((fingercoord(i,1)-
1):(fingercoord(i,1)+1),(fingercoord(i,
2)-1):(fingercoord(i,2)+1),1) =
255/255;
        fingerpath((fingercoord(i,1)-
1):(fingercoord(i,1)+1),(fingercoord(i,
2)-1):(fingercoord(i,2)+1),2) = 63/255;
        fingerpath((fingercoord(i,1)-
1):(fingercoord(i,1)+1),(fingercoord(i,
2)-1):(fingercoord(i,2)+1),3) = 52/255;
    end

    % Showing region of white pixel
found above as a 6x6 red region
    fingerpath((x-2):(x+2),(y-
2):(y+2),1) = 1;
    fingerpath((x-2):(x+2),(y-
2):(y+2),2) = 1;

    imshow(fingerpath);

    % Showing boundary of right half of
image
    % Convention of directions x and y
in image processing:
    % Inc_x:RIGHT, Inc_y:UP
    line([.5*width .5*width], [1
height], 'color', 'g', 'LineWidth', 2);
    line([width width], [1 height],
'color', 'g', 'LineWidth', 2);
    line([.5*width width], [1 1],
'color', 'g', 'LineWidth', 2);
    line([.5*width width], [height
height], 'color', 'g', 'LineWidth', 2);
    count = count + 1;
end

toc;
pause(.100);

end

close all;
closepreview(vid);

% Displaying the whole path of the
first white pixel found
fingerpath = blur;
for i = 1:size(fingercoord,1)

fingerpath(fingercoord(i,1):(fingercoor
d(i,1)+1),fingercoord(i,2):(fingercoord
(i,2)+1),1) = 1;

fingerpath(fingercoord(i,1):(fingercoor
d(i,1)+1),fingercoord(i,2):(fingercoord
(i,2)+1),2) = 0;

fingerpath(fingercoord(i,1):(fingercoor
d(i,1)+1),fingercoord(i,2):(fingercoord
(i,2)+1),3) = 0;
end
% Also, displaying the last fingertip
if ~isempty(x) && y>.5*width+3 &&
y<width-3 && x>3 && x<height-3
    fingerpath((x-2):(x+2),(y-
2):(y+2),1) = 1;
    fingerpath((x-2):(x+2),(y-
2):(y+2),2) = 1;
    fingerpath((x-3):(x+3),(y-
3):(y+3),3) = 0;
end;
figure('name', 'fingerpath'),
imshow(fingerpath);
```

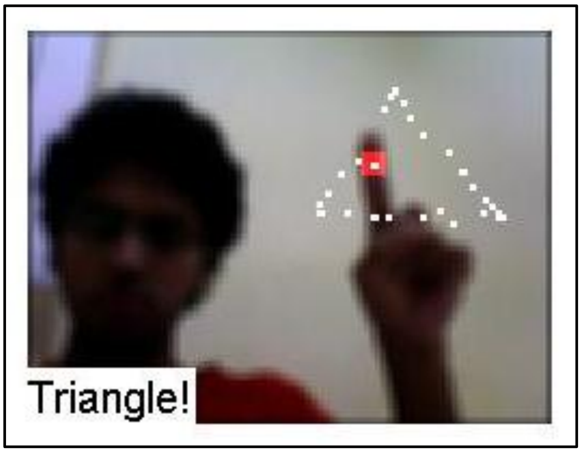
```
%% HMM 1: To determine the state among Right, Down, Left, Up, DR, DL, UL, UR

% Possible States: Right, Down, Left, Up, DR, DL, UL, UR
% Matrix of the observations O
% Changing from convention of image: Inc_x:DOWN, Inc_y:RIGHT
% to convention of graphs: Inc_x:RIGHT, Inc_y:UP
O = [fingercoord(:,2) height-fingercoord(:,1)];

% Finding difference of current coordinates from previous ones to get an observation sequence
% that describes constancy or increase or decrease in x and y coordinates
Obs = sign(O - [O(1,:); O(1:(size(O,1)-1),:)]);

% Deleting the first 4 and the last 2 observations, assuming them to be not part of the gesture
Obs = reshape(Obs(5:(size(Obs,1)-2),:),size(Obs,1)-6,size(Obs,2));

% Possible Observations: [NoChangeIn_x NoChangeIn_y; Dec_x Dec_y; Inc_x Inc_y]
Ob = [ 0 0; -1 -1; 1 1];

% Initial Probability matrix
Pi = [.4; .03; .03; .04; .4; .03; .03; .04];

% Transition matrix
a = [.6 .25 .01 .01 0.1 .01 .01 .01; .01 0.6 .25 .01 0.1 .01 .01 .01; .01 .01 .65 .15 .01 .01 .01 .15; .01 .01 .01 .93 .01 .01 .01 .01;...
    .01 .01 .15 .01 .65 .15 .01 .01; .01 .05 .05 .01 .01 0.6 .25 .01; .01 .01 .055 .055 .01 .01 0.6 .25; .01 .01 .01 .01 .01 .01 .01 .93];

% Emission matrix
b(:,:,1) = [.1 0 .9; .34 .33 .33; .1 .9 0; .34 .33 .33; .1 0 .9; .1 .9 0; .1 .9 0; .1 0 .9]; % for x coordinates
b(:,:,2) = [.34 .33 .33; .1 .9 0; .34 .33 .33; .1 0 .9; .1 .9 0; .1 .9 0; .1 0 .9; .1 0 .9]; % for y coordinates

% Finding the sequence of states for the different observations
% using the Viterbi algorithm
% V = ViterbiAlgo(InitialProbMatrix, TransitionMatrix, EmissionMatrix,...
%   PossibleOutputs, Observations)
V = ViterbiAlgo(Pi, a, b, Ob, Obs);
```

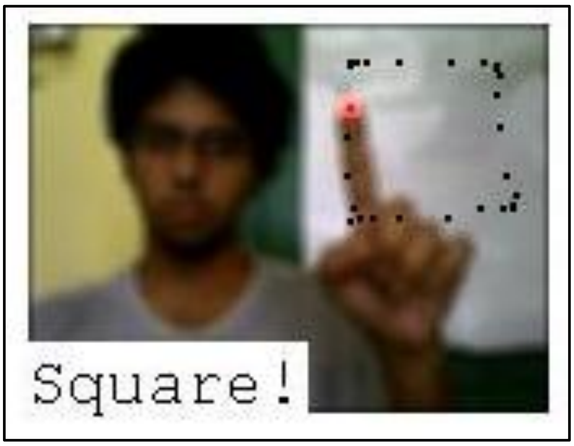
```
%% HMM 2: To determine the state among Triangle, Square, Diamond

% Possible States: Triangle, Square, Diamond
% Observation sequence: Sequence of states found in previous HMM
% Possible Observations: [R; D; L; U; DR; DL; UL; UR]
OB = [1; 2; 3; 4; 5; 6; 7; 8];

% Initial Probability matrix
PI = [.34; .33; .33];

% Transition matrix
A = [.98 .01 .01; .01 .98 .01; .01 .01 .98];

% Emission matrix
B = [.09 .04 .25 .04 .25 .04 .04 .25; 0.22 0.22 0.22 0.22 .03 .03 .03 .03; .03 .03 .03 .03 0.22 0.22 0.22 0.22];

% Determiining the gesture by finding the state in which the system was,
% using the Viterbi algorithm
%Q = ViterbiAlgo(prior, transmat, obsmat, OB, V);
Q = ViterbiAlgo(PI, A, B, OB, V);

if size(unique(Q))==1
    switch unique(Q)
        case 1
            disp('Triangle!');
text(1,.7*width,'Triangle!', 'BackgroundColor' , 'white', 'Color', 'black', 'FontSize', 15);
        case 2
            disp('Square!');
            text(1,.7*width,'Square!', 'BackgroundColor' , 'white', 'Color', 'black', 'FontSize', 15);
        case 3
            disp('Diamond!');
            text(1,.7*width,'Diamond!', 'BackgroundColor' , 'white', 'Color', 'black', 'FontSize', 15);
    end
else
    disp('Are you kidding me..?');
    text(1,.7*width,'Are you kidding me..?', 'BackgroundColor' , 'white', 'Color', 'black', 'FontName', 'Impact', 'FontSize', 15);
end

end %function end
```

d. Use of HMM

For each of the two HMM's, the input is stored in the variable `O`, the set of possible observations `Ob` and the values of `Pi`, `a` and `b` already been declared in the code. The values of `Alpha` and `Betaa` are determined using the `Forward` and `Backward Algorithms`. All of these values are then fed into `Viterbi Algorithm` to find out the hidden states.

This code was made only to illustrate the correctness of the HMM algorithms. There are simpler ways to accomplish the same task, the first being to use only one HMM, but his would require a lot of training data. The two sets of values of `Pi`, `a` and `b` used in this code were statistically determined by observing the chance of occurrence of the respective states in the three shapes drawn. Therefore, there could be errors in recognition, but the program serves its purpose of demonstrating that the HMM codes work.





4. Conclusion

- **Hidden Markov Models:**

  I have successfully implemented Hidden Markov Models in MATLAB. This required me to write codes to implement four codes:
  - Forward Algorithm
  - Backward Algorithm
  - Baum-Welch Algorithm
  - Viterbi Algorithm

  The codes I wrote were completely original, efficient, optimised and user-friendly.

- **"Finger-Tip Gesture Recogniser"**

  This program was developed in order to demonstrate the codes for Hidden Markov Models work. This program could recognise the shape drawn by a human finger-tip using Hidden Markov Models. The main work flow of this program is:
  - Capture image from the video feed.
  - Perform skin-pixel segmentation, and make a priority search within the skin-pixel area to find the tip of a human finger.
  - Store the coordinates of the finger-tip in a matrix so as to record its path.
  - Repeat this process until the gesture is completed, i.e. the finger-tip reaches a point within a fixed radius of the starting point.
  - Taking the coordinates of the finger-tip as "observations" for a Hidden Markov Model, using the codes on HMM's find out the shape of the gesture as the "hidden state."

  The training of the Hidden Markov Model involved setting the values of the initial probability matrix, transition probability matrix and the emission matrix. The actual training would have taken a lot of time and effort, so I approximated the values of those matrices to the limiting case where infinite number of training data could be given. This small trick worked very well and gave wonderful results.

5. References

[1] Lawrence R. Rabiner, "A Tutorial on Hidden Markov Models," February 1989

[2] Mark Stamp, "A Revealing Introduction to Hidden Markov Models," February 2012

[3] Sebastien Marcel, Olivier Bernier, Jean-Emmanuel Viallet and Daniel Collobert, "Hand Gesture Recognition using Input-Output Hidden Markov Models"

[4] B. H. Juang; L. R. Rabiner, "Hidden Markov Models for Speech Recognition," *Technometrics,* Vol. 33, No. 3 (August 1991), pp 251-272

[5] Guy Leonard Kouemou, "History and Theoretical basics of Hidden Markov Models"