Experiments and results

In the experiment, we first trained our GMM-HMM model using first half of the dataset (T/2=100) of each person's activity recordings. After training each activity separately, we obtained 13 probability transition matrices A, and observation transition probability matrices B. Then, we randomly selected data from second half of the dataset to test our recognition accuracy. Specifically, to choose samples randomly, we first randomly selected a person from the four people. Next, we chose the starting point of the sample using a random number, and length of testing sequence is a quarter of dataset (T/4=50), so that the selection operation is equivalent to adopt a sliding window with 50% overlapping. The number of random samples for training is 200. Finally, for each testing sample, we respectively calculated likelihood of this observation with estimated parameters from each activity. The likelihood was obtained using forward algorithm. The activity class with the largest likelihood is supposed to be the true activity.

Table I shows the results of the recognition accuracy just using features from the first skeleton. The number of pose state we used was 3, and joint state number is 5. The average overall accuracy is 91% if all testing samples are randomly selected from four persons. To evaluate individual recognition performance, we further conducted experiments where samples were just from one single person. We find testing samples from person 3 achieves the highest accuracy (94.2%) while test on person 4 gives the worst performance (82.4%). The result that recognition accuracy varies amongst individuals indicates a possible relationship between different people and recognition performance. If we could include more people in our training step, probably we could get a better result since more individuals' diversities are considered.

Table I Recognition accuracy with respect to individuals (1st joint data)

Activity	Test on person 1	Test on person 2	Test on person 3	Test on person 4	Test on randomly
Still	100	100	100	100	100
Talking on the phone	100	100	100	100	78
Writing on white board	62	100	100	100	100
Drinking water	100	100	100	100	100
Rinsing mouth with water	100	100	100	100	100
Brushing teeth	38.5	100	100	53	58
Wearing contact lense	100	36.5	100	13.5	100
Talking on the couch	100	35.5	92.5	100	81.5
Relaxing on the couch	100	23.5	100	0	66.5
Cooking (chopping)	65.5	100	100	100	100

Cooking (stirring)	100	100	96.5	100	100
Opening pill container	100	100	35.5	100	100
Working on computer	100	100	100	100	100
Overall Average	**89.7**	**84.27**	**94.2**	**82.4**	**91.1**

This recognition result is better than the one given in our referenced paper. We think the main reason is that we changed joint state number from 3 in the paper to be 5. The reason that we made the change was that we could not explain that three states should give the optimum performance, while intuitively some other states could also yield a same, if not better performance. The following figure shows an approximate relationship between recognition accuracy and several different numbers of joint states or pose states. The joint number chosen from 3 to 7, and pose number selected as 2,3,or 4, is more likely to yield a higher accuracy.

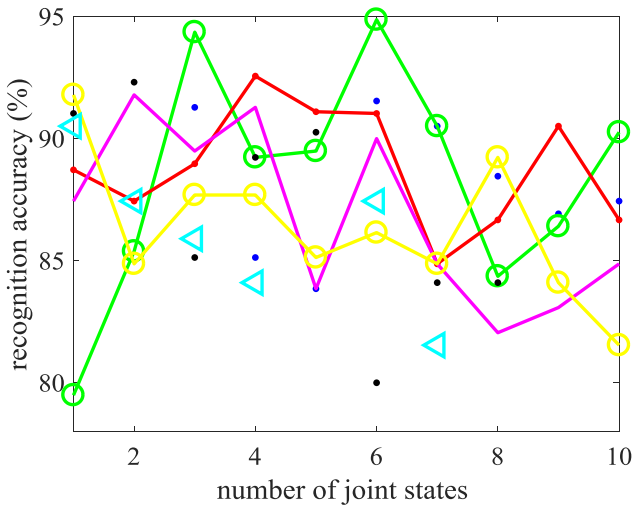

Fig. name… (add legend in the figure)

Another factor we noticed that might affect the recognition accuracy is the selection of skeleton joints. Table II gives the recognition accuracy of individuals using measurement from the second skeleton joints. The experiment was conducted with the same parameters as used in Table I. By inspection Table I and Table II, we find the result is surprisingly better than the first one, where we have an average recognition accuracy of 95.6%. One possible explanation is this skeleton probably conveys more information of activity features so that features themselves could have better distinctions.

Table II Recognition accuracy with respect to individuals (2nd joint data)

Activity	Test on person 1	Test on person 2	Test on person 3	Test on person 4	Test on randomly
Still	100	100	100	100	100

Talking on the phone	100	100	100	100	100
Writing on white board	100	100	100	100	100
Drinking water	100	100	100	100	100
Rinsing mouth with water	100	100	100	100	85.5
Brushing teeth	83	100	100	100	80.5
Wearing contact lense	100	100	100	100	100
Talking on the couch	100	100	100	100	100
Relaxing on the couch	100	100	100	100	75.5
Cooking (chopping)	100	100	100	100	100
Cooking (stirring)	100	100	100	100	100
Opening pill container	100	100	100	100	100
Working on computer	100	100	100	100	100
Overall Average	**98.7**	**100**	**100**	**100**	**95.6**

However, in the 'unseen case' as demonstrated in the referenced paper, our averaged recognition result (ranging from around 20% to 30%) is inferior to their results (78%). Although we had a low averaged recognition accuracy, the recognition accuracy could achieve 100% for some activities, such as 'Rinsing mouth with water' and 'Cooking (stirring)'. Future work could turn to analyze the relationship of the same activity amongst different individuals so that we could achieve higher classification accuracies.

Conclusion

In this report, we studied the GMM-HMM model for human activity recognition. Instead of using features from all skeleton joints mentioned in the referenced paper, we could achieve even higher activity recognition accuracy with a single skeleton node with a different joint state number. Then we conducted experiments to further investigate the relationship between recognition accuracy and different choices of state numbers. In our case, the joint number from 3 to 7, and pose number such as 2,3,or 4,tends to give a higher performance. Finally, we explored recognition performance with respect to different choices of skeleton joints—some skeleton joint simply shows an obviously superior performance (skeleton 2 in our case) than the other joints. This could provide insight into selecting more suitable joints for recognition purposes.

Notes

This model is an extension of HMM learnt in EECE 562 course. In addition to HMM and GMM models, in this project we were also involved with topics such as EM for parameter estimation, K-means clustering and forward algorithms to compute likelihoods in Markov chains. Although we met plenty of difficulties in this project, such as how to build models, extract features, initialize parameters in faced with this brand-new area, we could eventually find a way to the answer with background knowledge and group efforts.

Codes attached.

```
% Matlab code for EECE 562 project
% This code is to implement human activity recognition
%
% notes:
%       1. For table 1, use case1 and respectively change Itest from [1] to
%       [2],[3],[4], and [1 2 3 4]
%       2. Uncomment case 2 instead, and test unseen case
%       3. change node from 1 to 11 to try different skeletons

close all;
clear;clc

%% para. setting
Ifull=[1 2 3 4];      % full dataset of person 1 to 4

%## case 1 START ##
Itrain=[ 2 3 4 ];    % full dataset used for training,repace it with other set if necessary
Itest=[1];              % change this value respectively to 2,3,4
%## case 1 END ##

% %## case 2 START ##
% Itrain=[2 3 4];    % full dataset used for training,repace it with other set if necessary
% Itest=[1];
% %## case 2 END ##

nSamples=200;            % # of random samples to test

C={'still','talking_on_the_phone','writing_on_the_whiteboard','drinking_water',...

'rinsing_mouth_with_water','brushing_teeth','wearing_contact_lenses_2','talking_on_couch',...

'relaxing_on_couch','cooking_chopping','cooking_stirring','opening_pill_container','working_on_computer'};
```

```
T=200;                    % length of time slices, first half for training, seond for testing
O = 3;    % 3d position
nex = length(Itrain); % eg. 3 person's data
M = 5;    % state# of a single joint
Q = 3;    % state# of activity    % change    Q to other number
node=1; % skeleton node
cov_type = 'full';
Data=zeros(3*length(C),T,length(Ifull));
A_all=zeros(Q,Q,length(C));       % prob. trans. mat. of each activity
B_all=zeros(Q,M,length(C));       % obser. prob. trans. mat.
p_all=zeros(Q,length(C));
mu_all=zeros(3,Q,M,length(C));
Sigma_all=zeros(3,3,Q,M,length(C));

loglik=zeros(1,length(C));
label_true=(1:length(C))'*ones(1,nSamples);    % true label
label_est=zeros(length(C),nSamples);       % recognitized label

%% training GMM-HMM model for each activity
% load and transform data for easier manipulation
for i=1:length(Ifull)
      load (['person_',num2str(i),'.mat']);
      for j=1:length(C)
            Dtmp=eval(cell2mat(C(j)));          % Dtmp: tempory variable, transform cell-named data
to variable
            Data((j-1)*3+1:j*3,:,i)=Dtmp(1:T,1:3)';
      end
end

% train GMM-HMM model
for j=1:length(C)
      clc;
% initial guess of parameters
      p0 = normalise(rand(Q,1));
      A = mk_stochastic(rand(Q,Q));

      data=Data((j-1)*3+1:j*3,1:floor(T/2),Itrain);
      [mu0, Sigma0] = mixgauss_init(Q*M, data, cov_type);
      mu0 = reshape(mu0, [O Q M]);
      Sigma0 = reshape(Sigma0, [O O Q M]);
      B = mk_stochastic(rand(Q,M));

 % train the model using EM
      [LL, p_est, A_est, mu_est, Sigma_est, B_est] = ...
```

```
            mhmm_em(data, p0, A, mu0, Sigma0, B, 'max_iter', 50);
  % save para.
       A_all(:,:,j)=A_est;
       B_all(:,:,j)=B_est;
       p_all(:,j)=p_est;
       mu_all(:,:,:,j)=mu_est;
       Sigma_all(:,:,:,:,j)=Sigma_est;
end

    fprintf('recognization in process, please wait: \n');
  %% testing stage -- random selected samples (unseen data)
  for j=1:length(C)
    % generate random testing subject
        subj_idx=randi(length(Itest),1,nSamples);
    % generate random starting point given subject
        start_idx=randi(floor(T/4),1,nSamples);     % for the 2nd half T/2, using slideing window
of .5 overlapping
    % generate random testing samples
        for k=1:nSamples

data=Data((j-1)*3+1:j*3,floor(T/2)+start_idx(k):floor(T/2)+start_idx(k)+floor(T/4)-1,Itest(subj_id
x(k)));
            for kk=1:length(C)                                  % matching over all activities
                A_tmp=reshape(A_all(:,:,kk),size(A_est));
                B_tmp=reshape(B_all(:,:,kk),size(B_est));
                p_tmp=p_all(:,kk);
                mu_tmp=reshape(mu_all(:,:,:,kk),size(mu_est));
                Sigma_tmp=reshape(Sigma_all(:,:,:,:,kk),size(Sigma_est));
                loglik(kk)  = mhmm_logprob(data, p_tmp, A_tmp, mu_tmp, Sigma_tmp, B_tmp);
            end

            [~,label_est(j,k)]=max(loglik);       % recognized activity

        end

  end

  % get accuracy
     accuracy=mean(label_est==label_true,2);
     clc;
     fprintf('recognition accuracy is: \n');
     disp(accuracy);
     fprintf('overall average accuracy is: \n');
     disp(mean(accuracy));
```

```
%% END of the main code
```

张赫后面代码是Kevin toolbox里面的，可写可不写；

```
function [LL, prior, transmat, mu, Sigma, mixmat] = ...
        mhmm_em(data, prior, transmat, mu, Sigma, mixmat, varargin)
% LEARN_MHMM  Compute the ML parameters of an HMM with (mixtures of) Gaussians
output using EM.
% [ll_trace, prior, transmat, mu, sigma, mixmat] = learn_mhmm(data, ...
%     prior0, transmat0, mu0, sigma0, mixmat0, ...)
%
% Notation: Q(t) = hidden state, Y(t) = observation, M(t) = mixture variable
%
% INPUTS:
% data{ex}(:,t) or data(:,t,ex) if all sequences have the same length
% prior(i) = Pr(Q(1) = i),
% transmat(i,j) = Pr(Q(t+1)=j | Q(t)=i)
% mu(:,j,k) = E[Y(t) | Q(t)=j, M(t)=k ]
% Sigma(:,:,j,k) = Cov[Y(t) | Q(t)=j, M(t)=k]
% mixmat(j,k) = Pr(M(t)=k | Q(t)=j) : set to [] or ones(Q,1) if only one mixture component
%
% Optional parameters may be passed as 'param_name', param_value pairs.
% Parameter names are shown below; default values in [] - if none, argument is mandatory.
%
% 'max_iter' - max number of EM iterations [10]
% 'thresh' - convergence threshold [1e-4]
% 'verbose' - if 1, print out loglik at every iteration [1]
% 'cov_type' - 'full', 'diag' or 'spherical' ['full']
%
% To clamp some of the parameters, so learning does not change them:
% 'adj_prior' - if 0, do not change prior [1]
% 'adj_trans' - if 0, do not change transmat [1]
% 'adj_mix' - if 0, do not change mixmat [1]
% 'adj_mu' - if 0, do not change mu [1]
% 'adj_Sigma' - if 0, do not change Sigma [1]
%
% If the number of mixture components differs depending on Q, just set the trailing
% entries of mixmat to 0, e.g., 2 components if Q=1, 3 components if Q=2,
% then set mixmat(1,3)=0. In this case, B2(1,3,:)=1.0.

if ~isempty(varargin) & ~isstr(varargin{1}) % catch old syntax
  error('optional arguments should be passed as string/value pairs')
end

[max_iter, thresh, verbose, cov_type, adj_prior, adj_trans, adj_mix, adj_mu, adj_Sigma] = ...
```

```
    process_options(varargin, 'max_iter', 10, 'thresh', 1e-4, 'verbose', 1, ...
                    'cov_type', 'full', 'adj_prior', 1, 'adj_trans', 1, 'adj_mix', 1, ...
                    'adj_mu', 1, 'adj_Sigma', 1);

previous_loglik = -inf;
loglik = 0;
converged = 0;
num_iter = 1;
LL = [];

if ~iscell(data)
  data = num2cell(data, [1 2]); % each elt of the 3rd dim gets its own cell
end
numex = length(data);



O = size(data{1},1);
Q = length(prior);
if isempty(mixmat)
  mixmat = ones(Q,1);
end
M = size(mixmat,2);
if M == 1
  adj_mix = 0;
end

while (num_iter <= max_iter) & ~converged
  % E step
  [loglik, exp_num_trans, exp_num_visits1, postmix, m, ip, op] = ...
      ess_mhmm(prior, transmat, mixmat, mu, Sigma, data);



  % M step
  if adj_prior
    prior = normalise(exp_num_visits1);
  end
  if adj_trans
    transmat = mk_stochastic(exp_num_trans);
  end
  if adj_mix
    mixmat = mk_stochastic(postmix);
  end
  if adj_mu | adj_Sigma
    [mu2, Sigma2] = mixgauss_Mstep(postmix, m, op, ip, 'cov_type', cov_type);
```

```
    if adj_mu
      mu = reshape(mu2, [O Q M]);
    end
    if adj_Sigma
      Sigma = reshape(Sigma2, [O O Q M]);
    end
  end

  if verbose, fprintf(1, 'iteration %d, loglik = %f\n', num_iter, loglik); end
  num_iter =  num_iter + 1;
  converged = em_converged(loglik, previous_loglik, thresh);
  previous_loglik = loglik;
  LL = [LL loglik];
end


%%%%%%%%%

function [loglik, exp_num_trans, exp_num_visits1, postmix, m, ip, op] = ...
    ess_mhmm(prior, transmat, mixmat, mu, Sigma, data)
% ESS_MHMM Compute the Expected Sufficient Statistics for a MOG Hidden Markov Model.
%
% Outputs:
% exp_num_trans(i,j)   = sum_l sum_{t=2}^T Pr(Q(t-1) = i, Q(t) = j| Obs(l))
% exp_num_visits1(i)   = sum_l Pr(Q(1)=i | Obs(l))
%
% Let w(i,k,t,l) = P(Q(t)=i, M(t)=k | Obs(l))
% where Obs(l) = Obs(:,:,l) = O_1 .. O_T for sequence l
% Then
% postmix(i,k) = sum_l sum_t w(i,k,t,l) (posterior mixing weights/ responsibilities)
% m(:,i,k)   = sum_l sum_t w(i,k,t,l) * Obs(:,t,l)
% ip(i,k) = sum_l sum_t w(i,k,t,l) * Obs(:,t,l)' * Obs(:,t,l)
% op(:,:,i,k) = sum_l sum_t w(i,k,t,l) * Obs(:,t,l) * Obs(:,t,l)'


verbose = 0;

%[O T numex] = size(data);
numex = length(data);
O = size(data{1},1);
Q = length(prior);
M = size(mixmat,2);
exp_num_trans = zeros(Q,Q);
exp_num_visits1 = zeros(Q,1);
```

```
postmix = zeros(Q,M);
m = zeros(O,Q,M);
op = zeros(O,O,Q,M);
ip = zeros(Q,M);

mix = (M>1);

loglik  = 0;
if verbose, fprintf(1, 'forwards-backwards example # '); end
for ex=1:numex
   if verbose, fprintf(1, '%d ', ex); end
   %obs = data(:,:,ex);
   obs = data{ex};
   T = size(obs,2);
   if mix
      [B, B2]  = mixgauss_prob(obs, mu, Sigma, mixmat);
      [alpha, beta, gamma,     current_loglik, xi_summed, gamma2] = ...
      fwdback(prior, transmat, B, 'obslik2', B2, 'mixmat', mixmat);
   else
      B = mixgauss_prob(obs, mu, Sigma);
      [alpha, beta, gamma,     current_loglik, xi_summed] = fwdback(prior, transmat, B);
   end
   loglik = loglik +     current_loglik;
   if verbose, fprintf(1, 'll at ex %d = %f\n', ex, loglik);  end

   exp_num_trans = exp_num_trans + xi_summed; % sum(xi,3);
   exp_num_visits1 = exp_num_visits1 + gamma(:,1);

   if mix
      postmix = postmix + sum(gamma2,3);
   else
      postmix = postmix + sum(gamma,2);
      gamma2 = reshape(gamma, [Q 1 T]); % gamma2(i,m,t) = gamma(i,t)
   end
   for i=1:Q
      for k=1:M
         w = reshape(gamma2(i,k,:), [1 T]); % w(t) = w(i,k,t,l)
         wobs = obs .* repmat(w, [O 1]); % wobs(:,t) = w(t) * obs(:,t)
         m(:,i,k) = m(:,i,k) + sum(wobs, 2); % m(:) = sum_t w(t) obs(:,t)
         op(:,:,i,k) = op(:,:,i,k) + wobs * obs'; % op(:,:) = sum_t w(t) * obs(:,t) * obs(:,t)'
         ip(i,k) = ip(i,k) + sum(sum(wobs .* obs, 2)); % ip = sum_t w(t) * obs(:,t)' * obs(:,t)
      end
   end
end
```

```
if verbose, fprintf(1, '\n'); end
```