EMHMM-Toolbox: Eye-Movement analysis with Hidden Markov Models (HMMs)

Antoni B. Chan, City University of Hong Kong
Janet H. Hsiao, University of Hong Kong
Tim Chuk, University of Hong Kong
Cynthia Y.H. Chan, University of Hong Kong

Copyright (c) 2019, City University of Hong Kong & University of Hong Kong
2019-Feb-22, v0.75

DESCRIPTION

This is a MATLAB toolbox for analyzing eye movement data with hidden Markov Models (HMMs). The major functions of the toolbox are:

1) Estimating HMMs for an individual's eye-gaze data.
2) Clustering individuals' HMMs to find common strategies.
3) Visualizing HMMs and fixation data.
4) Statistical tests to see if two HMMs are different.
5) Using representative Holistic and Analytic models to compute HA scale value.

You can find more information about the methodology in the below reference papers.

REFERENCES

If you use this toolbox, please cite the following papers:

- For learning HMMs for eye gaze data: Tim Chuk, Antoni B. Chan, and Janet H. Hsiao, "Understanding eye movements in face recognition using hidden Markov models", *Journal of Vision*, 14(11):8, Sep 2014.
- For clustering HMMs with the VHEM algorithm: Emanuele Coviello, Antoni B. Chan, and Gert R.G. Lanckriet. "Clustering hidden Markov models with variational HEM". *Journal of Machine Learning Research*, 15(2):697-747, Feb 2014.
- For representative Holistic/Analytic models and HA Scale: Cynthia Y.H. Chan, Antoni B. Chan, Tatia M.C. Lee, and Janet H. Hsiao, “Eye Movement Patterns in Face Recognition are Associated with Cognitive Decline in Older Adults.” *Psychonomic Bulletin & Review*, to appear 2018.

CONTACT INFO

Send comments, bug reports, feature requests to Antoni Chan (abchan@cityu.edu.hk).

SETUP

The toolbox includes a script for setting up the working environment of EMHMM. In MATLAB, change to the main directory of the toolbox and run the following to setup

EMHMM.

```
>> setup
*** EMHMM - Eye-Movement anaysis with HMMs ***
  - current version: v0.75
  - checking for new version...
  - version up-to-date.
  - checking MEX files...
  - all OK
```

This will perform the following: 1) setup the MATLAB path to include the toolbox; 2) check if there is a new version of EMHMM available (requires an Internet connection); 3) check the MEX files and if necessary, compile and test them.

MEX files

EMHMM uses MEX files to speed up computations of some of the algorithms. MEX files are compiled functions that are native to specific computer platforms. *EMHMM includes the compiled code for macOS (High Sierra 10.13), Windows 7, and Linux (Ubuntu). Hence for those systems you do not need to compile the MEX files.*

The toolbox will check if there are compatible MEX files when running "setup". If MEX files are not found, the toolbox will attempt to compile the code for your computer automatically. Here are some tips:

- If you have never compiled a MEX file on your system before, you may need to run "mex -setup" to pick the compiler.
- To compile on Mac, you need to download and install Xcode from Apple. There could be some issues compiling on older versions of MATLAB when using a newer macOS. These problems can be fixed by modifying a few files in Xcode – more info:
  - https://www.mathworks.com/matlabcentral/answers/246507-why-can-t-mex-find-a-supported-compiler-in-matlab-r2015b-after-i-upgraded-to-xcode-7-0
  - https://www.mathworks.com/matlabcentral/answers/243868-mex-can-t-find-compiler-after-xcode-7-update-r2015b#comment_407066
- If you need to re-compile the MEX files, run the command: `emhmm_mex_check('all')`.

Finally, you can still use EMHMM without compiled MEX files. However, the algorithms will run about 10-20 times slower.

MATLAB toolboxes

EMHMM uses the following MATLAB toolboxes:

- Statistics toolbox
- Parallel computing toolbox (optional)

Parallel Computing Toolbox

If the Parallel Computing toolbox is available, EMHMM will use "parallel for loops" (parfor) to speed up the calculations. The speed-up will depend on the number of CPU cores in your computer. To use this feature, you need to have a "parallel pool" created on your computer.

MATLAB can be set to create pools automatically. Go to the MATLAB "Preferences", and select "Parallel Computing Toolbox" and enable "automatically create a parallel pool". You can also manually start a parallel pool by clicking on the small icon  on the bottom-left of the MATLAB window, and selecting "Start parallel pool".

DEMO

In this section, we will go through one of the demo programs to show the functions of the toolbox. Change to the "demo" folder, run "demo_faces" to run a demo script.

```
>> cd demo
>> demo_faces
```

Reading data

This script will first load the fixation data from an Excel spreadsheet, demodata.xls. The spreadsheet has 4 columns: SubjectID, TrialID, FixX, and FixY, which correspond to the subject ID, trial number (ID), and X and Y fixation locations. The fixation data is automatically separated according to the subject ID and trial number. In each trial, it assumes that the fixations occur in order. The output will look like this:

```
Reading demodata.xls
- found SubjectID in column 1
- found TrialID in column 2
- found FixX in column 3
- found FixY in column 4
- found 10 subjects:
1 2 3 4 5 6 7 8 9 10
    * subject 1 had 56 trials
    * subject 2 had 56 trials
    * subject 3 had 56 trials
    * subject 4 had 56 trials
    * subject 5 had 56 trials
    * subject 6 had 56 trials
    * subject 7 had 56 trials
    * subject 8 had 56 trials
    * subject 9 had 56 trials
    * subject 10 had 56 trials
```

The program found 10 subjects with 56 trials each.

Estimating and viewing individual HMMs

Next, an individual HMM will be estimated from each subject's fixation data. We use the variational Bayesian EM algorithm to estimate the HMMs. It can automatically select the number of ROIs, as well as estimate all the hyperparameters from the data, by finding the model with highest log-likelihood of the data. In this case, we try K={1, 2, 3} ROIs. For each K, the VB algorithm is run 50 times with random initializations, and then the hyperparameters are estimated for each unique result. The run with the highest log-likelihood is kept. The best model over all the candidate Ks will be selected. The `vbhmm_learn_batch` command will take data for all subjects, and learn an individual HMM for each subject. The command looks like this:

```
>> [hmms] = vbhmm_learn_batch(data, K, vbopt);
```

where `vbopt` is a structure containing the options for running the VB-EM algorithm (see the demo script, and details below). `hmms` is a cell array containing the HMM for each subject.

The output for one subject looks like this (when Parallel Computing is not available):

```
=== running Subject 10 ===
[Subject 10] -- vbhmm K=1: (seed=1000).
[Subject 10] (K=1) optimizing trial 1: -2504.04.........................................LL=-2482.75
[Subject 10] (K=1) best run=1; LL=-2482.75
[Subject 10] (K=1)  {alpha0=0.1; epsilon0=0.1; v0=2026; beta0=Inf; W0=5.408e-07; mu0=[243.9;197]; }
[Subject 10] -- vbhmm K=2: (seed=1000)..........10..........20..........30..........40..........50
[Subject 10] (K=2) optimizing trial 1: -2442.56.........LL=-2417.82
[Subject 10] (K=2) optimizing trial 2: -2456.42...................LL=-2437.71
[Subject 10] (K=2) optimizing trial 3: -2487.89.........................LL=-2437.66
[Subject 10] (K=2) best run=1; LL=-2417.72
[Subject 10] (K=2)  {alpha0=0.0004094; epsilon0=0.4214; v0=1.515; beta0=61.71; W0=0.001876; mu0=[249.4;200.5]; }
[Subject 10] -- vbhmm K=3: (seed=1000)..........10..........20..........30..........40..........50
[Subject 10] (K=3) optimizing trial 1: -2406.12............LL=-2384.54
[Subject 10] (K=3) optimizing trial 2: -2404.26................LL=-2384.38
[Subject 10] (K=3) optimizing trial 5: -2444.18............LL=-2424.72
[Subject 10] (K=3) optimizing trial 10: -2450.17.................LL=-2432.57
[Subject 10] (K=3) optimizing trial 11: -2413.29................LL=-2384.28
[Subject 10] (K=3) optimizing trial 14: -2460.72.....................LL=-2436.11
[Subject 10] (K=3) optimizing trial 15: -2405.07............LL=-2384.53
[Subject 10] (K=3) optimizing trial 18: -2469.77.....................LL=-2446.19
[Subject 10] (K=3) optimizing trial 21: -2447.04...................LL=-2422.02
[Subject 10] (K=3) optimizing trial 47: -2440.01.......................LL=-2415.15
[Subject 10] (K=3) best run=11; LL=-2384.28
[Subject 10] (K=3)  {alpha0=2.86e-06; epsilon0=0.3668; v0=4.756; beta0=1.237; W0=0.0006825; mu0=[244.5;200.9]; }
[Subject 10] best model overall: K=3; L=-2382.49
[Subject 10]   {alpha0=2.86e-06; epsilon0=0.3668; v0=4.756; beta0=1.237; W0=0.0006825; mu0=[244.5;200.9]; }
```

Different values of K are tried, and then the hyperparameters are optimized for a few trials. Finally the best model with highest log-likelihood (LL) is selected. The dots and numbers indicate the trials being run, or each iteration in hyperparameter estimation.

When Parallel Computing is available, then the `vbhmm_learn_batch` function will learn HMMs on several subjects in parallel to save time. The output will look like this:

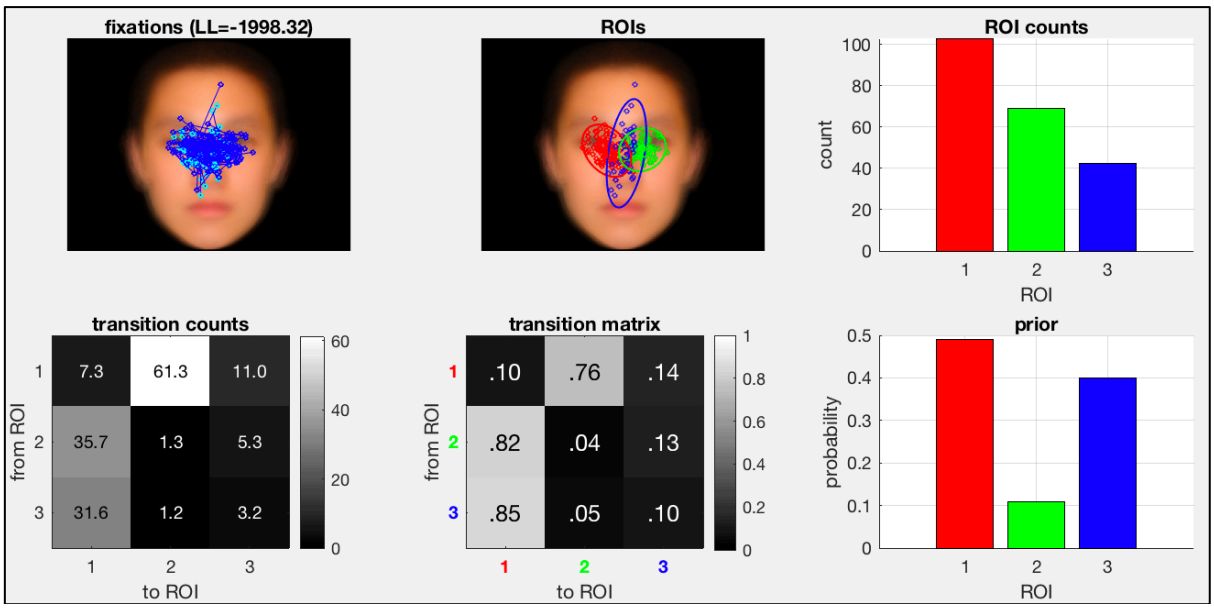
```
Starting parallel pool (parpool) using the 'local' profile ...
connected to 6 workers.
=== running Subject 1 ===
=== running Subject 2 ===
=== running Subject 3 ===
=== running Subject 4 ===
=== running Subject 5 ===
=== running Subject 6 ===
[Subject 1] -- vbhmm K=1: (seed=1000).
[Subject 1] (K=1) optimizing trial 1: -2065.4.........................LL=-2048
[Subject 1] (K=1) best run=1; LL=-2048
[Subject 1] (K=1)  {alpha0=0.1; epsilon0=0.1; v0=40.37; beta0=3.426e+51; W0=3.121e-05; mu0=[254.9;203.7]; }
[Subject 2] -- vbhmm K=1: (seed=1000).
[Subject 2] (K=1) optimizing trial 1: -2882.31.....................................LL=-2859.61
[Subject 2] (K=1) best run=1; LL=-2859.61
[Subject 2] (K=1)  {alpha0=0.1; epsilon0=0.1; v0=657.1; beta0=1.621e+101; W0=1.15e-06; mu0=[267.2;186.5]; }
...
[Subject 3] -- vbhmm K=2: (seed=1000)..........10..........20..........30..........40..........50
[Subject 3] (K=2) optimizing trial 1: -2537.95.............LL=-2512.64
[Subject 4] (K=2) optimizing trial 1: -2285.37.....................LL=-2262.5
[Subject 5] -- vbhmm K=2: (seed=1000)..........10..........20..........30..........40..........50
[Subject 6] -- vbhmm K=2: (seed=1000)..........10..........20..........30..........40..........50
[Subject 1] -- vbhmm K=2: (seed=1000)..........10..........20..........30..........40..........50
...
```

Since there are 6 workers in the pool, then 6 subjects will be run in parallel, and the output messages are mixed together.

To visualize an HMM and the data, we use the following 2 commands (in the demo script):

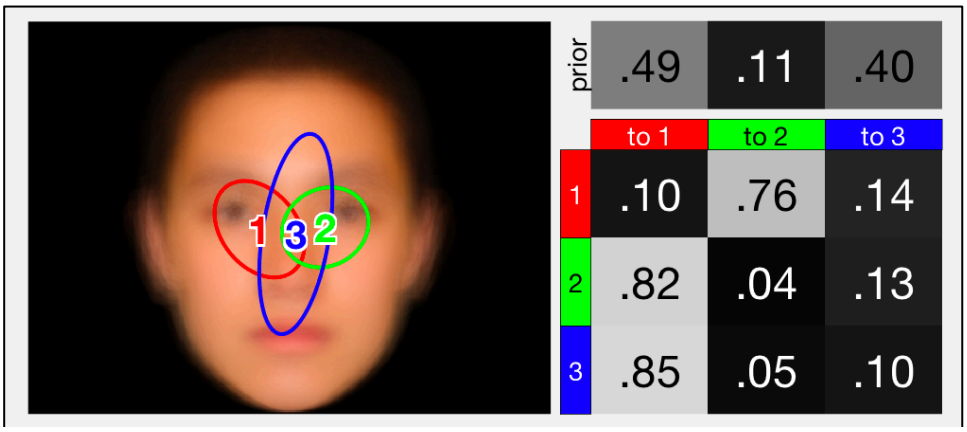
```
>> vbhmm_plot(hmms{1}, data{1}, faceimg)
>> figure, vbhmm_plot_compact(hmms{1}, faceimg)
```

The first command looks at the HMM of the first subject. "`faceimg`" contains the name of the image file of the face to put in the background. The figure from `vbhmm_plot` looks like this (Figure 1 from the demo):



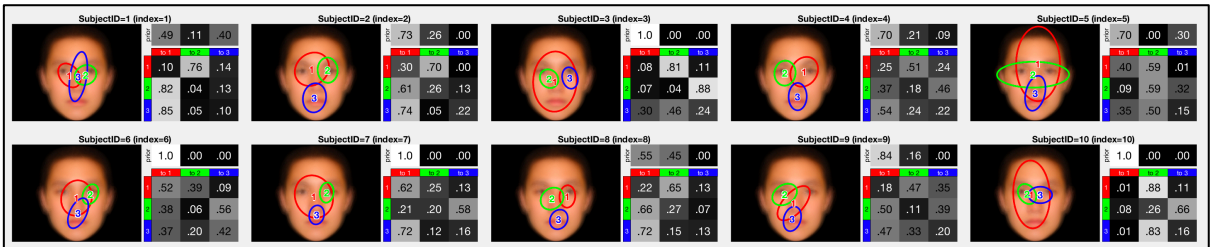
The top-left plot shows the fixation sequences, and the cyan "o" is the first fixation. The top-middle plot shows the emission densities (ROIs) and the corresponding fixations. The top-right plot shows the total number of fixations in each ROI. On the bottom, the left and middle plots show the transition counts and transition matrix. In the transition matrix (bottom-middle), each row is a probability distribution. I.e., each row is normalized to sum to 1. Finally, the bottom-right shows the prior probability, i.e., the probability of the ROI of the first fixation. The ROIs are automatically sorted according to the most likely fixation path: ROI 1 is the most probable initial fixation; ROI 2 is the most likely next fixation, given a fixation in ROI 1; etc. In this example, the subject has 50% probability to look at ROI 1 (red region). Then has a 76% of transitioning to the green ROI 2. In the green ROI 2, the subject will look back at ROI 1 (82% probability). So the subject tends to look back and forth between the eyes. In ~40% of the cases, the subject will first look in the middle of the face at ROI 3, and then transition to the red ROI 1.

The second command makes a compact plot of the HMM and doesn't show the data. The figure looks like this (Figure 2 from the demo):



On the left side are the ROIs (labeled with colors and numbers). The right-top shows the prior probabilities, and the right-bottom shows the transition matrix. The rows/columns are labeled with colors and numbers to indicate the corresponding ROIs.

The demo script also plots all the subjects in one figure (Figure 3 from the demo):



The demo script saves the individual HMMs to the file: models_demo_faces_individual.mat

Clustering Individuals' HMMs

Given the 10 individuals' HMMs, the demo script then computes the overall eye gaze pattern by clustering the individuals into one group. Here the variational HEM (VHEM) algorithm is used to cluster the HMMs. The algorithm tries 3 different initialization methods, running each one 100 times. The run with the highest log-likelihood is kept. Here is the command used in the demo:

```
>> [all_hmms1] = vhem_cluster(hmms, 1, 3, hemopt)
```

First we use K=1 groups (i.e., put all into one group) and S=3 hidden states. `hemopt` is a structure containing the options for running VHEM. The output of the demo script looks like this:

```
=== Clustering (1 group) ===
Checking input HMMS: done
+ set seed to 1001
auto initialization: trying baseem: VHEM Trial: ..............................
.....................................................................

Best run is 5: LL=-1012.11
auto initialization: trying gmmNew: VHEM Trial: ..............................
.....................................................................

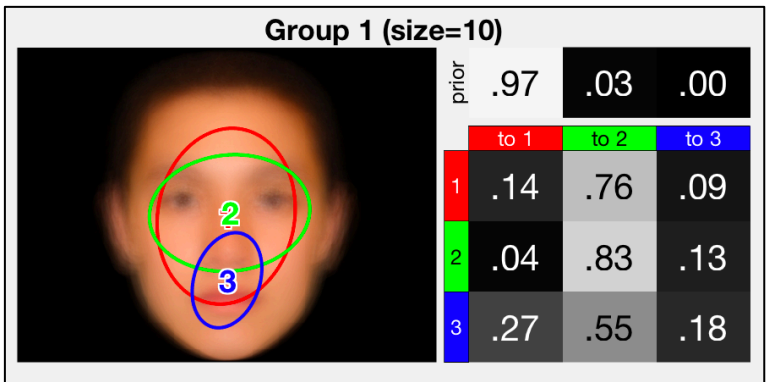
Best run is 65: LL=-1012.26
auto initialization: trying gmmNew2: VHEM Trial: ..............................
.....................................................................

Best run is 1: LL=-1014.07
best init was baseem; LL=-1012.11
```

We can plot the group HMM using the following command:

```
>> vhem_plot(all_hmms1, faceimg);
```

The corresponding plot of the group HMM is here (Figure 4 from the demo):



For the overall HMM for all subjects, two of the ROIs are large and cover most of the face features. This is mainly because there are different strategies used by each subject, and merging them all together yields a "washed out" HMM, which lacks specific details. One interesting thing is that subjects' mainly look to the eyes for the second fixation (green ROI), after looking somewhere on the face (red ROI). This indicates that a common strategy of subjects looking somewhere on the face first, and then turning to the eyes. The mouth (blue ROI) is looked at after the eyes with small probability.

To discover the different strategies among the subjects, next the demo script clusters the individual HMMs into 2 groups using VHEM, by changing to K=2.

```
[group_hmms2] = vhem_cluster(hmms, 2, 3, hemopt)
```

Here is the output:

```
=== Clustering (2 groups) ===
Checking input HMMS: done
+ set seed to 1001
auto initialization: trying baseem: VHEM Trial: ..............................
.....................................................................

Best run is 95: LL=-1007.19
auto initialization: trying gmmNew: VHEM Trial: ..............................
.....................................................................

Best run is 65: LL=-1008.85
auto initialization: trying gmmNew2: VHEM Trial: ..............................
.....................................................................

Best run is 22: LL=-1007.28
best init was baseem; LL=-1007.19

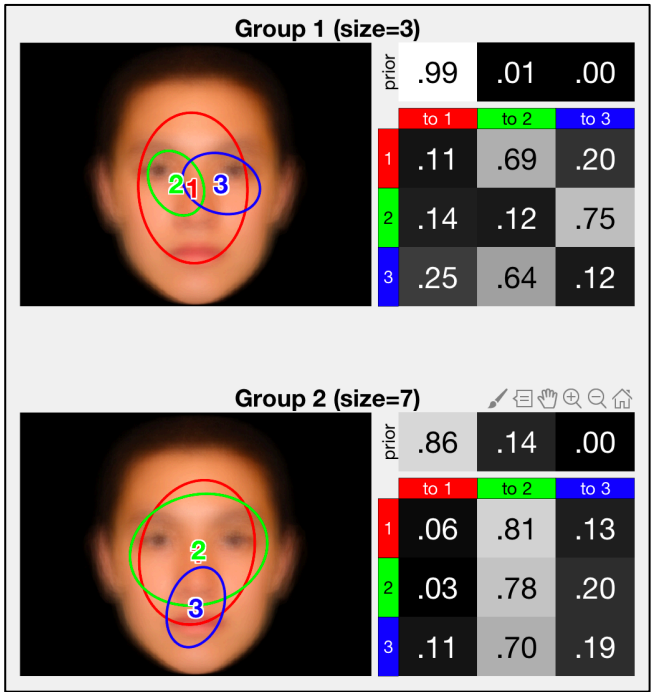
group_hmms2 =

  struct with fields:

                Z: [10×2 double]
            LogLs: [1×53 double]
             LogL: -1.0072e+03
            label: [1 2 1 2 2 2 2 2 2 1]
           groups: {[1 3 10]  [2 4 5 6 7 8 9]}
       group_size: [3 7]
             hmms: {[1×1 struct]  [1×1 struct]}
           hemopt: [1×1 struct]
    emhmm_version: 'v0.75'

Group membership (indicies):
  group 1 = [1 3 10]
  group 2 = [2 4 5 6 7 8 9]
Group membership (SubjectID):
  group 1 = 1, 3, 10,
  group 2 = 2, 4, 5, 6, 7, 8, 9,
```

Here is the plot of the two group HMMs (Figure 5 from the demo):

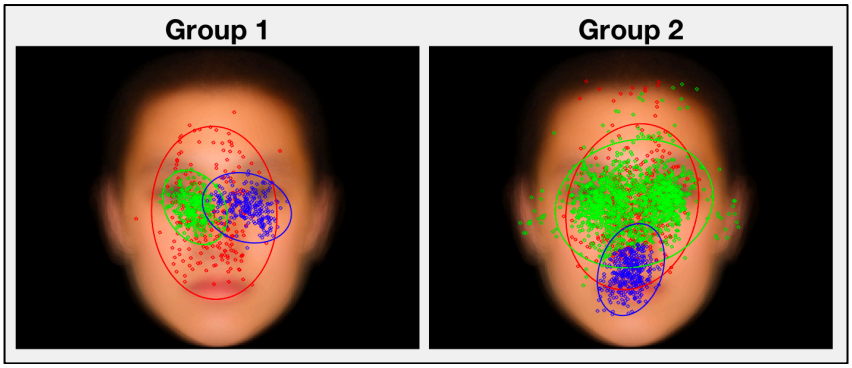


After clustering into two groups, we see two separate strategies. For Group 1 (top), they tend to look between the two eyes (blue and green ROIs). For Group 3 (bottom), they tend to look at the eyes (green ROI), and the mouth (blue ROI).

Finally, we can look at the fixations of subjects assigned to each group using the function `vhem_plot_fixations`:

```
vhem_plot_fixations(data, group_hmms2, faceimg);
```

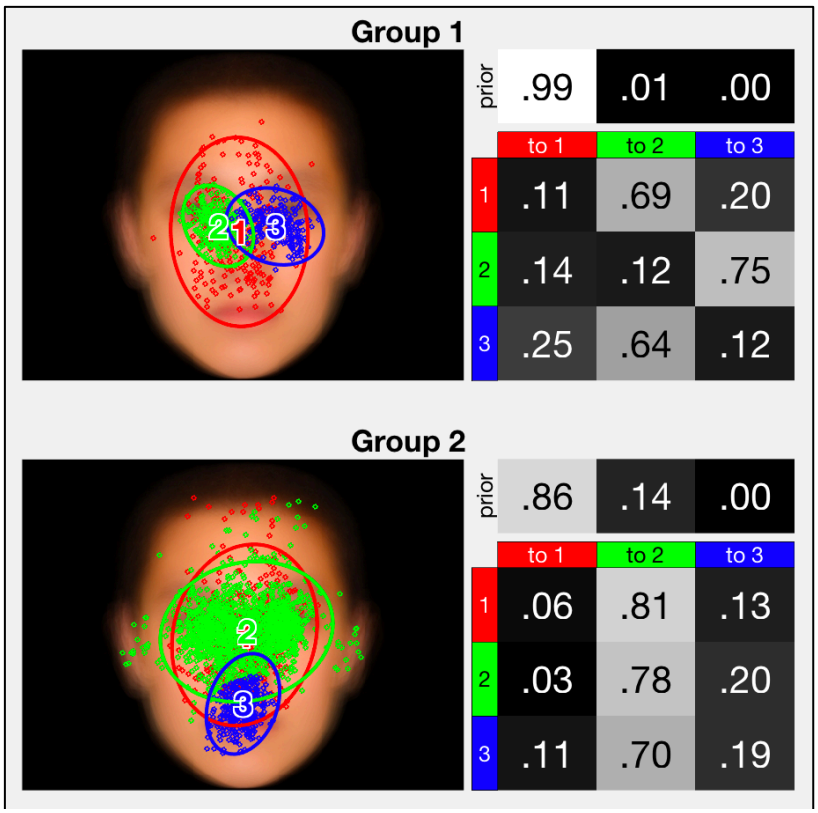
Here is the example figure (Figure 8 from the demo)



Adding the 'c' option will also show the transition matrices:

```
vhem_plot_fixations(data, group_hmms2, faceimg, 'c');
```

Here is the example figure (Figure 9 from the demo):



Analysis of HMMs

Next, the demo script analyzes the two group HMMs. The following command finds the most probable ROI sequences from the first group HMM, using length=3 sequences:

```
>> [seqs, seqs_pr] = stats_stateseq(group_hmms2.hmms{1}, 3);
```

Here is the output from the demo:

```
*** top-5 most probable ROI sequences ***
prob   : hmm1 ROI seq
----------------
0.5070 : 1 2 3
0.1280 : 1 3 2
0.0922 : 1 2 1
0.0793 : 1 2 2
0.0728 : 1 1 2

prob   : hmm2 ROI seq
----------------
0.5407 : 1 2 2
0.1364 : 1 2 3
0.0858 : 2 2 2
0.0793 : 1 3 2
0.0384 : 1 1 2
```

For hmm1, the most likely ROI sequence is 1-2-3, with probability 50.7%, followed by 1-3-2 (12.8%) and 1-2-1 (9.2%).

Next, we test whether the two group HMMs are different. We take the data for subjects in Group 1, and calculate the log-likelihood of this data under the Group 1 HMM and under the Group 2 HMM. The average difference between the log-likelihoods is an estimate of the Kullback-Leibler (KL) divergence, which is a dissimilarity measure for probability

distributions. A KL divergence of 0 means that the two distributions (HMMs) are the same. We run a paired t-test on the two lists of log-likelihoods to see if the average log-likelihood difference is significantly different from 0. Since KL divergence is not symmetric, we need to do this twice: once using the data from Group 1, and once using the data from Group 2. The code for computing test on hmm1 is:

```
>> data1 = {data{group_hmms2.groups{1}}};
>> [p, info, lld] = stats_ttest(group_hmms2.hmms{1}, ...
      group_hmms2.hmms{2}, data1);
```

The first line collects the data for subjects in group 1. The second line computes the test.

The test results are below:

```
*** t-test ***
- test group hmm1 different from group hmm2: t(2)=10.3738; p=0.00458241
- test group hmm2 different from group hmm1: t(6)=3.89294; p=0.00402468
```

In both cases, the two HMMs are significantly different (p=0.0045 and p=0.0040).

Finally, the script calculates the mean log-likelihood of each subject under the two group HMMs. A correlation test can be used with the log-likelihood data and behavioral data, to see if a particular eye fixation strategy is correlated with behavior. The command for calculating the mean log-likelihoods for hmm1 is:

```
[mLL1] = stats_meanll(group_hmms2.hmms{1}, data);
```

The output of the demo shows both hmms:

```
*** get mean LL for each subject ***
- mean LL for each subject under group hmm1:

mLL1 =

   -9.3626  -10.4290   -9.5695  -10.6841  -11.2984  -10.1845  -10.2214   -9.6876  -10.6591   -9.3781

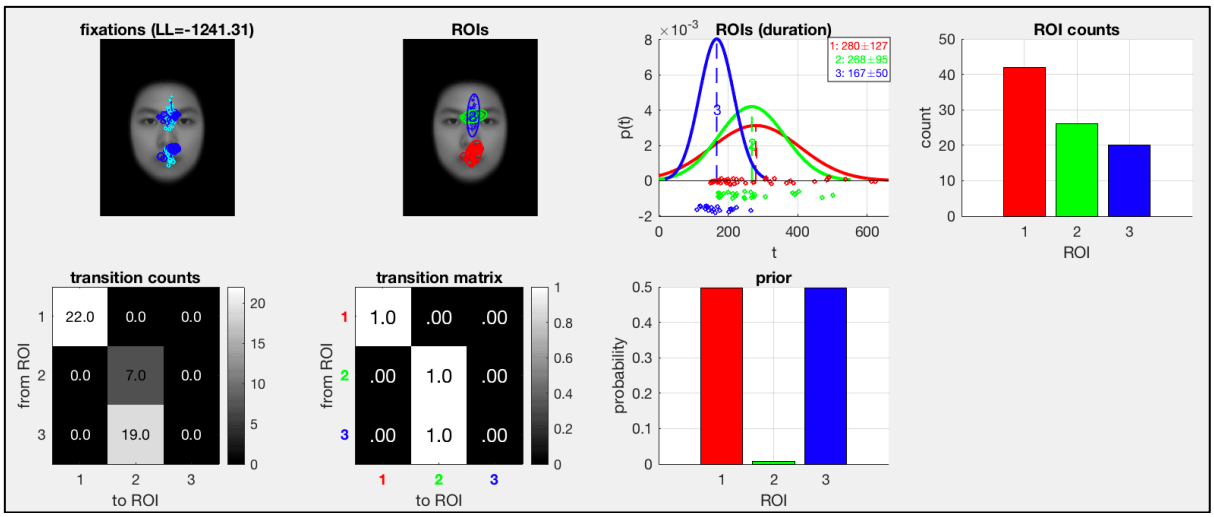
- mean LL for each subject under group hmm2:

mLL2 =

   -9.7846  -10.0976  -10.1602  -10.1141  -10.7487   -9.8083   -9.9147   -9.8250  -10.1173   -9.9160
```

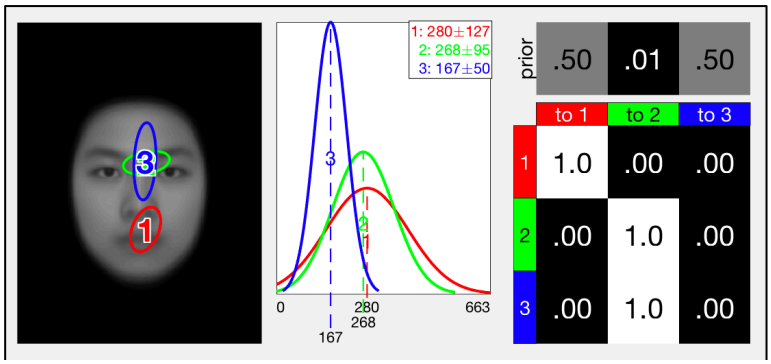
Eye fixation data with duration

The toolbox can also learn HMMs for eye fixation data consisting of both fixation location (x,y) and duration (time in milliseconds). An example can be found in the "demo_faces_duration.m" script. The script loads the data from "jov_duration.xls". The Excel spreadsheet now has a 5th column with the header "FixD", which is for the fixation duration. The commands used to learn a subject's HMM and group HMMs are the same. Here is an example of the 6th subject's HMM using fixation location and duration:



The 3rd subplot shows the distribution of fixation duration for each ROI. For this subject, either they will have a long fixation on the mouth (red ROI), or a quick fixation to the upper face (blue ROI) and then looking at the nose for longer (green ROI). The fixation duration data is shown at the bottom of the ROI duration plots. Note that the x-axis values are the durations, and the y-values are randomly selected to better visualize the data. The legend in the upper-right of the subplot shows the mean and standard deviation of the durations for each ROIl.

Here is a compact plot of the same HMM:



More details can be found in the demo script “demo_faces_duration.m”.

Other Demos

There are more demos for clustering and comparing HMMs in the demo scripts "demo_faces_jov_clustering.m" and "demo_faces_jov_compare.m".

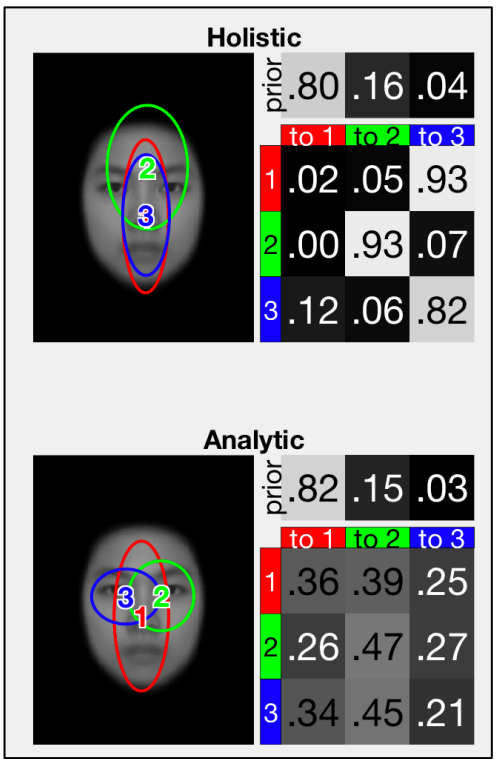
DEMO: Representative Holistic/Analytic Models and H-A Scale

In this demo we show the representative models and H-A scale used in the paper:

> Cynthia Y.H. Chan, Antoni B. Chan, Tatia M.C. Lee, and Janet H. Hsiao, “Eye Movement Patterns in Face Recognition are Associated with Cognitive Decline in Older Adults.” *Psychonomic Bulletin & Review*, to appear 2018.

The representative model and demo script are in the “models” directory. Change to the “models” directory, and then run `demo_PBR_model`.

The script first loads the representative models and plots them. The top is the Holistic representative model, where fixations tend to be on the center of the face. The bottom is the Analytic representative model, where fixations tend to be on the eyes and mouth.



Next, we load our new data, and then normalize the coordinates to be the same as the images used to learn the representative model (in the demo, the coordinates are the same, so no normalization is actually needed). For more information about converting coordinates between face images, please see the next demo.

We compute the log-likelihoods of the new data under the representative models.

```
*** get mean LL for each subject ***
Column 1 refers to the LL of the Holistic pattern
Column 2 refers to the LL of the Analytic pattern

LL_all =

  -10.3345   -9.3578
   -9.8063   -9.1211
   -9.5015   -9.2958
   -9.3297   -9.2799
   -8.5411   -9.0032
   -9.6745  -10.6560
   -9.2844   -9.0184
   -9.6761   -9.2645
```

We also compute the Holistic-Analytic (HA) scale, which is:

$$HA = \frac{LLH - LLA}{|LLH| + |LLA|}$$

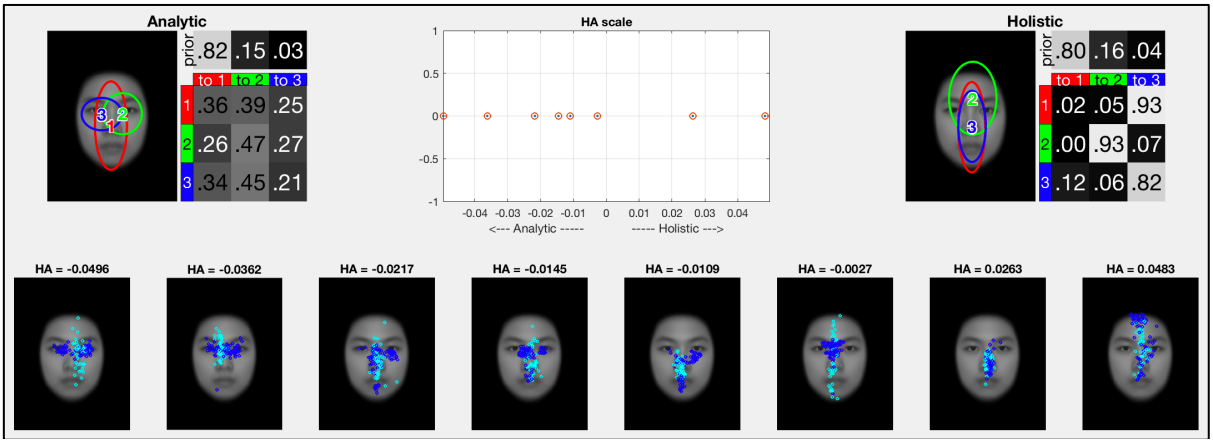
where LLH is the log-likelihood of the subject's data under the Holistic model, and LLA is the log-likelihood under the Analytic model.

```
*** H-A scale ***

HAscale =

   -0.0496
   -0.0362
   -0.0109
   -0.0027
    0.0263
    0.0483
   -0.0145
   -0.0217
```

Here is the generated plot showing the Analytic and Holistic representative models (top-left and top-right), the HA scale (top-center). A few subjects on the scale are plotted on the bottom. These are highlighted in the HA scale plot with red circles. Positive HA values indicate a more holistic pattern, while negative HA values indicate a more analytic pattern.



Finally, we compute the cluster assignments of the subjects.

```
*** subject assignments to HMMs ***

*** groups ***
group 1 - Holistic: 5 6
group 2 - Analytic: 1 2 3 4 7 8
```

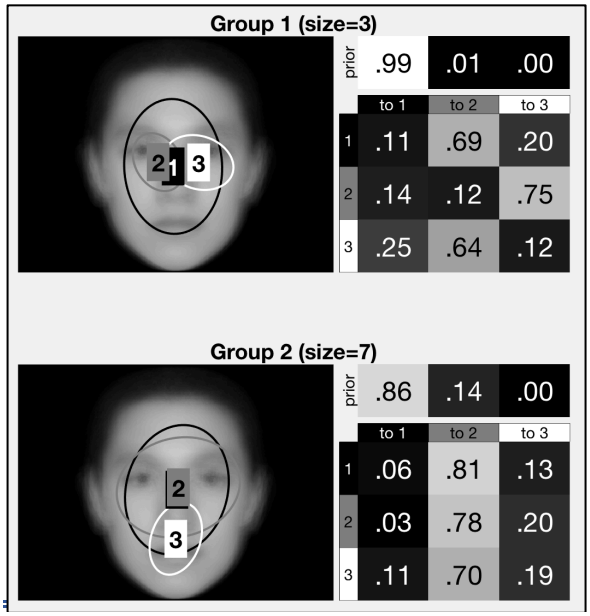
In the demo script, the "datamode" variable can be changed to select one of the other datasets: all JOV data and the demo dataset.

Grayscale Figures

By default EMHMM will produce color figures. Grayscale figures (e.g., for printing) can created by toggling the color mode:

```
>> emhmm_toggle_color
** emhmm is set to grayscale figures
```

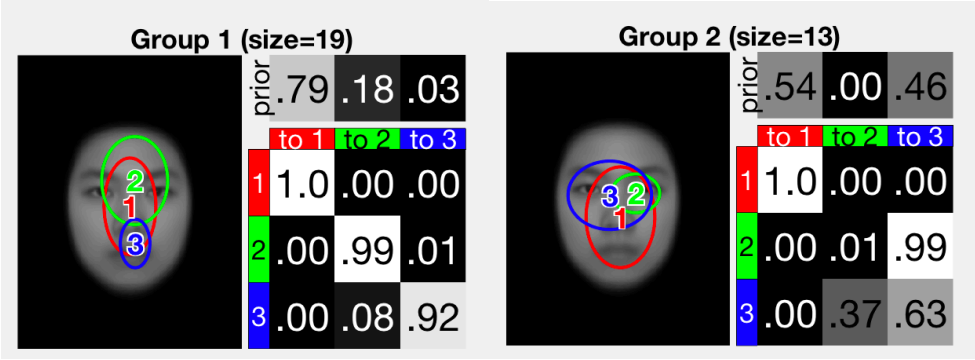
EMHMM will now generate denote the ROIs using different gray levels. Here is an example figure generated from `demo_faces`:



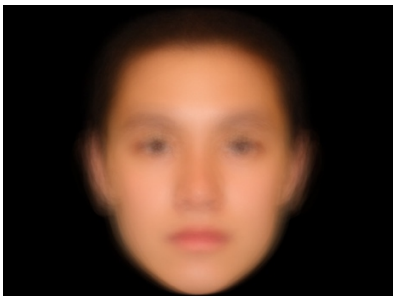
DEMO: Converting coordinates between face images

The learned HMMs depend on the coordinate system of the images used to collect the data. For example the location of the eyes, mouth, and nose are particular to one dataset, and could change for another dataset. The demo `demo_conversion` shows how to convert from the coordinates of one face image to another face image, so that the HMMs can be used to analyze data collected from other image sets.
IMPORTANT NOTE: normalization of new data to an old model is only valid when the presentation sizes (visual angle) of the old and new data are the same.

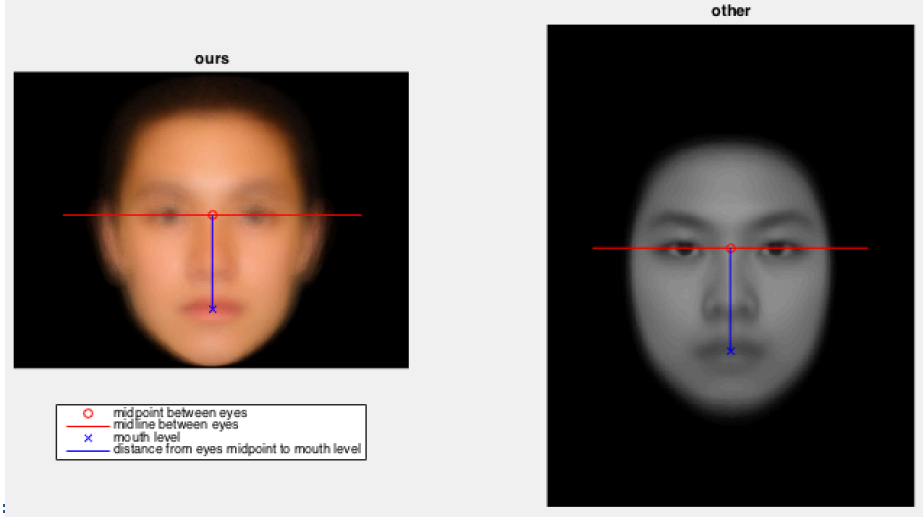
The group HMMs learned from the JOV data look like this:



However, our new data is collected from images with a different size:



To convert from our data to the JOV data, we first define two things on the images: 1) the coordinates of the midpoint between the eyes (called C); 2) the distance between the eyes midpoint and the mouth level (called D). Here is a picture showing these points:



For these images, we have:

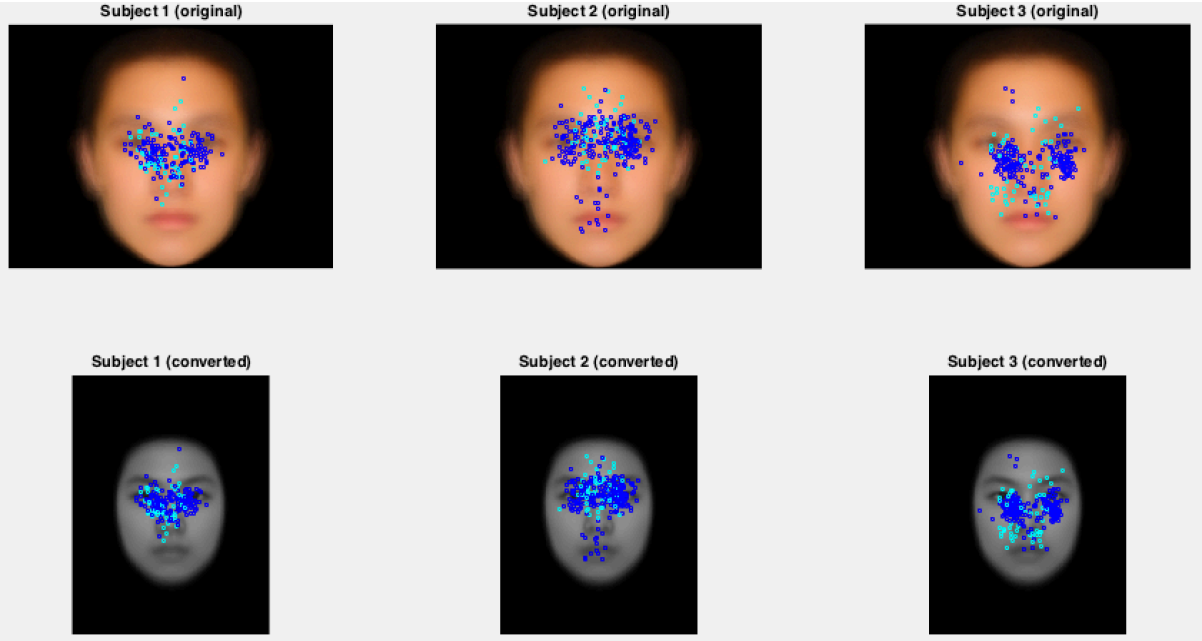
```
% for our images
our_C  = [258, 186];  % midpoint between eyes
our_D  = 121;         % distance from eyes midpoint to mouth level

% for the other images from JOV:
other_C = [160,195];  % midpoint between eyes
other_D = 90;         % distance from eyes midpoint to mouth level
```

Using this information, we convert the coordinates from our images to the other images:

```
>> data_s = convert_face_coord(data, our_C, our_D, other_C, other_D);
```

The following figure shows the fixations using the original coordinates and the converted coordinates on their respective images:



Finally, we use the converted data and compute the log-likelihoods with the JOV models:

```
>> [mLL1] = stats_meanll(model.group_hmms2.hmms{1}, data_s);
>> [mLL2] = stats_meanll(model.group_hmms2.hmms{2}, data_s);
```

Here is the result from the demo script:

```
- mean LL for each subject under hmm1:

mLL1 =

   -9.4891   -9.8699  -10.2966  -10.3456  -11.2027   -9.5222  ...
   -9.7196   -9.6563  -10.2170   -9.6101

- mean LL for each subject under hmm2:

mLL2 =

   -8.9928   -9.6833   -9.5378   -9.9843  -10.9729   -9.2440  ...
   -9.5662   -9.1458   -9.8409   -9.2563
```

FUNCTION USAGE

In this section, we provide more details about the main function calls and their input parameters. More detailed help can be found in MATLAB for each function by using the help command on the function (e.g., `help vbhmm_learn`).

Options are passed to the main functions, `vbhmm_learn`, `vbhmm_learn_batch`, and `vhem_cluster`, using a structure where each field represents one parameter setting. Values are not specified in the structure will be automatically filled-in using default values.

vbhmm_learn

estimate a single HMM from eye fixation data.

Usage:

```
[hmm,L] = vbhmm_learn(data,K,vbopt)
```

INPUTS	Description
`data`	Nx1 cell array, where each element is a fixation sequence and N is the number of sequences. data{i} is a TxD matrix, where T is the sequence length, and D is the dimension of a single fixation point (typically 2 for x and y coordinates).
`K`	The number of hidden states (ROIs) to use. If given a vector, the K will be automatically selected among the entries in the vector.
`vbopt`	structure containing options (see below)
`vbopt.alpha0`	The Dirichlet distribution concentration parameter for the prior distribution of the initial fixation. Large values encourage a uniform prior, while small values encourage a concentrated prior (default=0.1). It should be a positive number. Another way to think of it is in terms of "virtual" samples. A typical way to estimate the probability of something is to count the number of samples that it occurs and then divide by the total number of samples, i.e. P = (# times it occurred) / (# samples). The alpha parameter of the Dirichlet adds a "virtual" sample to this estimate, so that the probability estimated is P = (# times it occurred + alpha) / (# samples). Hence, for small alpha, the probability estimate will just follow the data, while for large alpha, it forces all the probabilities to be very similar (i.e., uniform).
`vbopt.epsilon0`	The concentration parameter for the Dirichlet distribution on the rows of the transition matrix (default=0.1). The meaning is similar to alpha above, but for the probabilities in the transition matrix.
`vbopt.mu0`	prior mean of the ROIs (default = [256;192]). Typically, it should be at the center of the image. Alternatively, you can use the average fixation location. When fixation duration is included in the data, then the default is [256;192;250].
`vbopt.W0`	Inverse variance of the inverse Wishart distribution (default=0.005). This parameter determines the variance (width) of the ROI prior. For example, W=0.005 means that the variance of the ROI prior is 1/0.005=200. Hence, the ROI prior has a standard deviation of 14 pixels, i.e., the ROI ellipses have width of 14*4 = 56 pixels. A vector can be specified, in which case the (x,y,d) widths of the ROIs can be different.

vbopt.v0	The degree-of-freedom of the inverse Wishart, v > D-1. (default=10). Larger values give preference to diagonal covariance matrices.
vbopt.beta0	The Wishart concentration parameter (default=1). Large values encourage estimated ROIs to be similar to the prior ROI (mean & W), while small value ignores the prior.
vbopt.seed	Seed for random number generator used when initializing the model. The seed is required for reproducible results.
vbopt.learn_hyps	estimate the hyperparameters (alpha0, epsilon0, etc) for each subject by maximizing the log-likelihood of the data. Use "1" to indicate all hyperparameters should be estimated.
vbopt.numtrials	The number of trails to run when using 'random' initialization (default=50).
vbopt.showplot	1 means show a plot of the HMM (default). 0 means don't show a plot.
vbopt.bgimage	Filename of the background image for plotting.
vbopt.verbose	0 means don't show any messages. 1 means show a few messages showing progress. (default) 2 means show lots of messages.

OUTPUTS	Description
hmm	Structure containing all the HMM parameters and other information. Use "help vbhmm_learn" for more details.
L	The log-likelihood of the data for the hmm.

vbhmm_learn_batch

Given a batch of data (data for several subjects), estimate an HMM for each subject. The options are largely the same as vbhmm_learn.

Usage:

```
[hmm_batch,Ls] = vbhmm_learn_batch(data_batch,K,vbopt)
```

INPUTS	Description
data_batch	Sx1 cell array, one cell for each subject. Each cell entry contains a cell array of N fixation sequences. I.e., the i-th trial of the s-th subject is: data{s}{i} = [TxD] matrix, where each row is a fixation (x,y) or (x,y,t). Duration t should be in milliseconds. Note: the sequence length T, and number of sequences N can be different for each subject/trial.
K	The number of hidden states (ROIs) to use. Same as vbhmm_learn.
vbopt	structure containing options. Most of the options are the same as vbhmm_learn, except the below ones.

vbopt.seed	Seed for random number generator used when initializing the model. The seed is required for reproducible results.
vbopt.learn_hyps	Estimate the hyperparameters (alpha0, epsilon0, etc) for each subject by maximizing the log-likelihood of the data. When used with vbhmm_learn_batch, each subject will have its own set of hyperparameters. This option assumes that each subject has a distinctive eye gaze strategy, and thus each subject should have its own set of hyperparameters.
vbopt.learn_hyps_batch	Estimate common hyperparameters for all subjects, i.e., all subjects share the same set of hyperparameters. Using this option assumes that all subjects have similar eye gaze strategies, and thus their HMMs are from the same distribution.

OUTPUTS	Description
hmm_batch	Structure containing all the HMM parameters and other information. Use "help vbhmm_learn" for more details.
Ls	Vector of data log-likelihood of the data for each hmm.

vbhmm_set_hyperparam

This is a helpful function to automatically set the hyperparameters mu and W for vbhmm_learn /vbhmm_learn_batch using the data.

Usage:

```
[vbopt] = vbhmm_set_hyperparam(vbopt, data, img, opt)
```

INPUTS	Description
vbopt	Existing vbopt structure
data	the same format as vbhmm_learn or vbhmm_learn_batch.
img	the template image (or filename)
opt	Option settings; 'c' = set mu as the center of the image, and W so that the prior ROI is 1/8 the width of the image. For duration, set the mean as 250ms and standard deviation as 25ms. 'd' = use a data-driven approach. Set the mu as the mean fixation location and duration. Set W according to the inverse variance of the data. Assumes that the ROI is circular in x-y dimensions.

OUTPUTS	Description
vbopt	New options structure with mu and W set.

vhem_cluster

Cluster HMMs into groups using VHEM and estimate representative HMMs for each group.

Usage:

```
[group_hmms] = vhem_cluster(hmms, K, S, hemopt)
```

INPUTS	Description
hmms	Nx1 cell array of HMMs, learned with vbhmm_learn_batch or vbhmm_learn.
K	The number of groups to cluster the data
S	The number of states (ROIs) in each group HMM. Use [] to automatically select the number of states as the median number of states in the input HMMs.
hemopt	structure containing options (see below)
hemopt.tau	virtual sequence length (default=10). It can be be set as the median sequence length in the data. Use "get_median_length(data)" to get the median length.
hemopt.trials	number of trials with random initialization to run (default=100)
hemopt.seed	Seed for random number generator used when initializing the model. The seed is required for reproducible results.
hemopt.verbose	0 means don't show any messages. 1 means show a few messages showing progress (default) 2 means show lots of messages.

OUTPUTS	Description
group_hmms	A structure containing the group HMMs and other information. Use "help vhem_cluster" for more details.
group_hmms.hmms	A 1xK cell array, where each entry is a group representative HMM.
group_hmm.LogL	The log-likelihood score of the group HMMs.
group_hmm.label	A 1xN vector, where each entry is the cluster assignment for the i-th input HMM.
group_hmm.groups	A 1xK cell array, where each entry contains the group members for that group.

TOOLBOX CONTENTS

The contents of the emhmm toolbox are listed below. Here only the important functions for the user are highlighted.

Directory/Filename	Description
setup.m	Setup MATLAB path for the toolbox, and check for updates.
demo/	
demo_faces.m	Simple example for learning individual HMMs and clustering HMMs.
demo_faces_duration.m	Simple example using fixation location and duration.
demo_faces_jov_clustering.m	Another example of learning and clustering HMMs.
demo_faces_jov_compare.m	Example of comparing HMMs learned from correct and incorrect face recognition trials.
demo_conversion.m	Example showing conversion of fixation coordinates between two images.

models/	
demo_PBR_model.m	Example computing HA scale using representative models from the PBR paper.
src/hem	
vhem_cluster.m	Cluster HMMs with VHEM algorithm
vhem_plot.m	Visualize the group HMMs
vhem_plot_clusters.m	Visualize the group HMMs and cluster members
vhem_plot_fixations.m	Visualize the fixations belonging to group HMMs
src/hmm	
vbhmm_learn.m	Learn HMM from data
vbhmm_learn_batch.m	Learn a batch of HMMs from batch of data.
vbhmm_plot_compact.m	Visualize an HMM using a compact format
vbhmm_plot.m	Visualize an HMM
vbhmm_kld.m	Calculate the KL divergence between 2 HMMs
vbhmm_ll.m	Calculate the log-likelihood of data for an HMM
vbhmm_set_hyperparam.m	Automatically set some hyperparameters using the data
vbhmm_remove_empty.m	Remove empty ROIs (those with < 1 fixation) in an HMM.
vbhmm_map_state.m	Compute the most probable state (ROI) sequence from fixation data
vbhmm_permute.m	Permute (re-order) the states of an HMM.
src/stats	
stats_ttest.m	Run a t-test to compare log-likelihoods of two HMMs.
stats_meanll.m	Calculate mean log-likelihood of a subject's data
stats_stateseq.m	Calculate the probability of all state sequences of a particular length.
src/util	
emhmm_version.m	Toolbox version number and history
emhmm_check_updates.m	Check if a new version of EMHMM is available
emhmm_mex_check.m	Check MEX files and compile & test if necessary.
Emhmm_toggle_color.m	Toggle between grayascale and color figures.
get_median_length.m	Compute the median length of sequences in data.
convert_face_coord.m	Convert fixation data to a new image size.
read_xls_fixations.m	Read fixations from an Excel spreadsheet in standard format.

ACKNOWLEDGEMENTS

This research was supported by the Research Grant Council of Hong Kong SAR: #17402814 and HKU 745210H for J.H. Hsiao; CityU 110513 and G-CityU109/14 for A.B. Chan. We also thank the HKU Seed Funding Programme for Basic Research (Project number 201311159131 to J.H. Hsiao.