Notes on Shay Ohayon's Mouse Tracking Code

Kristin Branson

All the tracking and analysis code can be accessed through the `Repository` function in `Repository/Repository.m`. This will open up a dialog that lets you choose files and functions to run them on.

The "Videos" panel shows all movies currently known by the program. The video files known to `Repository` are stored in the xml file `Config/Repository.xml` under the `<VideoList>` heading. The video list can be modified using the commands under the File menu. To the right of the "Videos" panel is a preview of the selected movie.

The "Identities" panel shows the mat files containing learned per-mouse identity classifiers, learned head/tail classifier, and learned mouse/not mouse classifier. These are all mat files in the `IdentitiesFolder` directory. `IdentitiesFolder` is also defined in `Repository/Repository.xml`. To the right of the "Identities" panel are snapshots of the mice defined in the selected mat file.

1 Pre-Processing

1.1 Pre-Processing → Track Single Mouse

To learn the classifiers in the identity mat files, we use movies in which there is only one mouse. We select these files in the "Videos" panel and choose "Track Single Mouse". In each of the selected videos, the single mouse is tracked and an ellipse is fit. HOG features are extracted in a rectangle around the fit ellipse. These feature vectors are stored for each frame in a file in the `ResultsRootFolder` directory, as defined in `Config/Repository.xml`. For each movie, a directory with the moviename (minus the extension) is created. In this directory will be a mat file named Identities.mat.

1.1.1 Repository::hLearnSingleMouseIdentityCallback

"Track Single Mouse" is implemented by the subfunction `hLearnSingleMouseIdentity_Callback` of `Repository`. This function first prompts the user to determine whether the tracking should be run on the Janelia cluster. If not, then it calls the function `fnLearnMouseIdentity` on each video sequence (Section 1.1.2). If the user chose to run on the Janelia cluster, then a job is created for each video sequence ... TODO ...

1.1.2 fnLearnMouseIdentity

`fnLearnMouseIdentity` first learns a median-based background model for the input video sequence. This is learned using the function `fnLearnBackgroundAux` (Section 1.1.3) to estimate the median of a small subset of randomly chosen frames (`iNumImagesForBuffer = 50`).

It then iterates through all frames in the video. For each frame, it:

1. Reads in the current frame.
2. Thresholds the absolute deviation from the background image at `fMotionThreshold` (hard-coded to $0.36 \cdot 255$).
3. Removes the timestamp printed on the video frame within the rectangle hard-coded to `[y = 1:24, x = 1:176]`.
4. Performs a morphological open with a box of size 4×4.
5. Finds connected components.
6. Computes the areas of each connected component using `fnLabelsHist`.
7. Chooses the largest connected component.
8. If there are no foreground pixels after thresholding and opening, then it labels the frame as not valid in `abValidFrame`, and goes on to the next frame.
9. Otherwise, it fits an ellipse to the foreground pixels in this largest connected component (a Gaussian is fit to the pixel locations with `fnFitGaussian` and converts the Gaussian covariance matrix to the (a, b, θ) representation.
10. It stores the ellipse position in `strctIdentity.m_af{X,Y,A,B,Theta}`.
11. Using the function `fnRectifyPatch`, it extracts a rectangle of length $22 \cdot 2.5 \cdot 2 + 1$ and width $17 \cdot 1.5 \cdot 2 + 1$ around the center location of the fit ellipse with orientation defined by the orientation of the fit ellipse (The semi-major axis length is hard-coded to 22 and the semi-minor axis length is hard-coded to 17. The length of the rectangle extracted is hard-coded in fnRectifyPatch to $A \cdot 2.5 \cdot 2 + 1$ and the width is hard-coded to $B \cdot 1.5 \cdot 2 + 1$. These sizes must match the size of the patches allocated in `fnLearnMouseIdentity` hard-coded to be of size 111×52). The function `fnFastInterp2` is used to interpolate the image and get pixel values at the non-integer pixel locations.
12. It extracts HOG features from the rectangular patch using the function `fnHoGfeatures` and stores them in `strctIdentity.m_a3HOGFeatures`.
13. It also extracts HOG features from the rectangular patch rotated by 180 degrees and stores them in `strctIdentity.m_a3HOGFeatureFlipped`.

For each interval of frames in which no connected component of foreground pixels is detected, it sets the ellipse positions by linearly interpolating between the positions before and after the interval. HOG features corresponding to rectangular patches around these interpolated values are extracted as above.

For each single-mouse video, a head/tail classifier is also learned. It resolves the head/tail ambiguity by classifying whether the mouse is in the canonical orientation or 180 degrees rotated from that. To do this, frames in which the speed in pixels per frame from the previous to the current frame is high ($>$ `fHighVelocityThresholdPix` which is hard-coded to be 6). The head/tail classification is assumed to correspond to the velocity direction in these frames. LDA is used to find a dimension onto which to project the data that well-classifies this identity versus all other identities. It inputs the entire vector of HOG features. The projected values are histogrammed to estimate the probability of the observation given head/tail class using the function `fnEstimateDistribution`. The per-class histogram is stored in `strctClassifier`, where field `afX` are the bin centers, `afHistPos` are the fraction of positives (head is forward) falling in each bin, `afHistNeg` is the fraction of negatives (tail is forward) falling in each bin, and

Once the head/tail classifer is learned, it is applied to every frame of the video and the soft score is

stored in afHeadTailValue. It is used, along with the velocity direction and ellipse orientation, to find a smooth series of orientations for all frames such that orientation agrees with velocity direction when the speed is sufficiently large and the appearance of the mouse agrees with the head/tail classifier in the function fnCorrectOrientation (Section 1.1.4). While fnCorrectOrientation can return values other than θ and $\theta + \pi$, after fnCorrectOrientation is run, the closest of θ and $\theta + \pi$ to the returned angle is chosen and stored.

1.1.3 fnLearnMouseIdentity::fnLearnBackgroundAux

This function randomly selects iNumImagesForBuffer (hard-coded in fnLearnMouseIdentity to 50) from all frames is the video, reads all these frames into a buffer, and computes the median.

1.1.4 fnCorrectOrientation

This function uses the Viterbi algorithm to choose the orientation at every frame. The number of states is 361, one for each degree, as well as a duplication 0 = 360. The likelihood for each state in each frame is set as follows. Let θ be the orientation of the ellipse fit and α be the velocity direction. For the two states corresponding to $round(\theta)$ and $round(\theta + \pi)$, the log-likelihood is initialized to the log of $P(head|image)$ and $1 - P(head|image)$, respectively, which are estimated using histogramming in fnLearnMouseIdentity and stored in strctClassifer. Note that from a Bayesian perspective, this is reasonable if we assume the priors $P(head) = P(tail)$. For all other frames, it is initialized to fLogZero = -5000. Next, for frames in which the speed is sufficient, 50 times the log-density of a Gaussian centered at $round(\alpha)$ and standard deviation hard-coded to fAlphaStd = $5 \cdot \pi/180$ is added to every state. This multiplying by 50 is equivalent to suing a Gaussian with standard deviation $5/\sqrt{(}50) \cdot pi/180$, or $0.707 \cdot pi/180$. Note that the density of this Gaussian will be much smaller than fLogZero = -5000 for many states. The transition probability follows a Laplace distribution with scale parameter hard-coded to 1000. The log probability for staying in the same state is hard-coded to 1/fMu/100 = 0.004 for some reason. Given these log-likelihoods and log-transition matrices, fndllViterbi is called to find the optimal sequence of states.

1.2 Pre-Processing → Verify Tracking on Single Mouse

TODO

1.3 Pre-Processing → Train Classifiers on Selected Sequences

To train per-mouse classifiers that distinguish identities based on the appearance of the mouse, select the single-mouse videos for all mice in the final video(s) and then choose "Train Classifiers on Selected Sequences" from the "Preprocessing" menu. This will call the function fnTrainClassifiers (Section 1.3.1) to learn a one-vs-all classifier for each mouse identity.

1.3.1 fnTrainClassifiers

fnTrainClassifiers assumes that the mat files containing the HOG features for each frame of each single-mouse movie are stored in [ResultsRootFolder]/[singlemousemoviename]/Identities.mat. These are the files created using the "Track Single Mouse" option (Section 1.1). If any of these files don't exist, the user is alerted and the function exits. This check occurs in the function fnCollectExemplars (Section 1.3.2).

fnTrainClassifiers calls fnCollectExemplars to choose and load in a sample of the HOG feature vectors for each single-mouse movie. Then, it calls fnTrainIdentities (Section 1.3.3) to learn the one-vs-all classifiers. It then calls fnCollectExemplars again to collect both flipped and not-flipped data. A new head/tail classifier is trained on data from all mice using the function fnTrainHeadTail. Finally, it prompts the user for a matfile to save the resulting classifiers to, and stores the results.

1.3.2 fnCollectExemplars

This function first checks to make sure that all of the mat files containing the HOG features created by "Track Single Mouse" exist. For each mouse identity. it chooses at most iMaxSamplesForIdentityTraining = 10000 (hard-coded in fnTrainClassifiers) to train the classifier on. This function also hard-codes the number of HOF features at iHOG_Dim = 837. Within fnCollectExemplars. the size of the image patch is again hard-coded to 52×111.

For each mouse identity, it loads in all the HOG feature vectors for all frames. It looks only at frames in which the semi-major axis length is greater than fMinA = 20 pixels and the semi-minor axis length is greater than fMinB = 10 pixels (hard-coded in fnCollectExemplars). It chooses the HOG vectors from the first aiNumSamplesTaken "good" frames. It stores and returns all sampled HOG vectors for all movies.

This function also finds the representative image patch from the frame in which the major and minor axis lengths are as close as possible to the median major and minor axis lengths.

1.3.3 fnTrainIdentities

For each mouse identity, fnTrainIdentities calls fnFisherDiscriminantTrainTest to learn a classifier. For a given mouse identity, this function uses linear discriminant analysis (LDA) to find the one-dimensional linear basis on which to project the HOG feature vectors to best discriminate the current identity from all other identities. Then, it histograms all the projected samples into 100 bins and computes the fraction of positive and negative samples within each bin, as well as the fraction of all samples within a bin that are positive. The histogramming is done by fnEstimateDistribution, and there is significant smoothing of the histograms going on. These values are stored in strctIdentityClassifier.

1.3.4 fnTrainClassifiers::fnTrainHeadTail

This function trains a head/tail classifier in much the same way that fnTrainIdentities trains a single-mouse identity classifier. It uses linear discriminant analysis (LDA) to find the one-dimensional linear basis on which to project the HOG feature vectors to best discriminate the current identity from all other identities. Then, it histograms all the projected samples into 100 bins and computes the fraction of positive and negative

samples within each bin, as well as the fraction of all samples within a bin that are positive. The difference, of course, is that this classifier is trained on the head-facing vs the tail-facing data.

2 Processing → Submit

We begin tracking multiple mice in a single video by first selecting the desired video in the Videos panel and the mat file with the trained classifiers in the Identities panel, then choosing "Submit" from the "Processing" menu. This calls `fnSubmitMovieToProcessing` (Section 2.1 to do the first step of tracking. In this step, we compute the positions of each mouse, but we do not concentrate on getting the identities correct, and do not use the mouse identity classifiers.

We split the video into multiple jobs that can be run in parallel by finding key frames where tracking is "easy" and we can trust tracking on the first frame. For each segment of video, we create a tracking job with a corresponding job file, stored in [JobFolder]/[moviename], where [JobFolder] is defined in Repository[Unix].xml.

The results of this tracking step will be stored in the directory [ResultsRootFolder]/[moviename].

2.1 fnSubmitMovieToProcessing

This function first pops up the `BackgroundDiffGUI` to compute the background model and set background subtraction parameters (Section 2.2). The parameters of the tracker are stored in the file [JobFolder]/[moviename]/Setup.mat by `fnCreateSetupFile`. Next, it pops up the `ReliableFramesGUI` GUI to choose frames at which to split the video to parallelize tracking.

2.2 BackgroundDiffGUI

The left side of the GUI shows the video and the right side allows the user to set parameters of the background modeling and subtraction. To learn the background model, the user clicks the "Learn Background" button.

This results in the `BackgroundDiffGUI::fnLearn` function being called. [Num Frames] frames are randomly sampled between frames [Start Frame] and [End Frame]. The background is estimated as the median of the sampled frames.

Then, k-means with $k = 2$ is called on all the pixel intensities in the background median image to find the intensity of the floor and the intensity of the wall. The larger intensity (whiter) cluster is assumed to correspond to the floor and the smaller intensity (darker) cluster is assumed to correspond to the wall. The frame timestamp may get clustered as foreground, and is forced to be clustered as wall. The location of the timestamp is hard-coded to rows $[1, 24]$ and columns $[1, 176]$.

After classifying the pixel locations as floor or wall based solely on pixel intensity, `bwlabel` is called to find all connected components labeled floor. The largest connected component of floor is found, and `imfill` and `imclose` (with a 10×10 square) are called to remove holes in the classification: `imfill( imclose( a2bBackground, ones(10,10) ) > 0,'holes' )`.

The segmented current frame is then plotted in the GUI. The background is shown dimmed and the foreground (as labeled using `fnSegmentForeground2`) (Section 2.3) is shown brightened. The wall

pixels are shown as purplish.

2.3 fnSegmentForeground2

This function performs background subtraction and thresholding. First, the absolute difference between the frame and the background image is computed. This difference is zeroed out over the timestamp, hard-coded here to be rows $[1, 24]$ and columns $[1, 190]$. If a pixel is on the floor, it is initially classified as foreground if the absolute difference is more than "Motion Threshold" (`m_fMotionThreshold`) and the image pixel intensity is less than "Intensity Threshold" (`m_fIntensityThresholdInFloor`). If the pixel is on the wall, then if it is initially classified as foreground if the absolute difference is more than "Motion Threshold Outside" (`m_fMotionThresholdOut`) and the image pixel intensity is less than "Intensity Threshold (Outside)" (`m_fIntensityThresholdOutsideFloor`). Then image closing is performed with a 5×5 square.

All connected components are found, and those with area at least "Large CC" pixels (`m_fLargeCC`) are kept.

Next, a hysteresis step is applied. Pixels that are within "Dilation" (`m_fDilation`) of a foreground pixel that satisfy the same absolute difference condition but a weaker image pixel intensity condition are classified as foreground. The image intensity threshold is increased by a factor of "Intensity Multiplier" (`m_fMoreMult`).

Finally, `bwlabel` is called to label the different connected components of foreground pixels.

3 Remember

Bug fixed.

- Looks like when run on the cluster, the identities output mat file is [ResultsRootFolder]/strFile/Identities.mat, but when run locally, the name is [ResultsRootFolder]/strFile/strFile_Identities.mat.
- When choosing frames with high speed for head/tail classifier learning, if the first or last frame is not valid, then the speed will be high at the beginning or end of the valid frames, as position is set to 0 here.
- For "problematic frames", π is added to the angle (fnLearnMouseIdentity:115, `strctIdentity.m_afTheta(aiNeedSwap) = strctIdentity.m_afTheta(aiNeedSwap) + pi;`), but the angle is never normalized again to be within $[0, 2\pi)$. In `fnCorrectOrientation`, the rounded angle is then set to 361 degrees if the swapped angle ends up being more than 360 degrees. The angle is only flipped if the speed is sufficiently large, and so I'm guessing it turns out somewhat okay most of the time because it will result in (much of the time) basically the velocity direction alone being used in these frames to set the likelihood. However, doing this right will change the relative weightings of all the factors and give different performance.
- The likelihood for the flipped angle can be much less than all other angles if $P(tail|image)$ is very small (e.g. 0), while other values are all thresholded at $\log P = -5000$.
- Why train on the HOG vectors from the first frames? You would get more independence by sampling from the entire trajectory.

True, but then it is more problematic to have a reliable test set