Documentation of the Automatic Behavior Analysis Software

1 Overview

The aim of this software is automatically detecting the paw in the videos, extracting the grasping sequences and implementing a representation which describes the paw appearance during the grasping action. With this representation it is possible to perform a detailed analysis of posture changes during the grasping task without the need for any manual annotations.
The detection of the paw on a query frame is accomplished by combining the foreground map (motion segmentation) with a low-level feature descriptor and powerful exemplar-based classifiers. Each extracted paw posture (bounding box around the detection) is represented using a dictionary of paws, thus establishing a posture embedding. The embedding then yields a similarity measure between postures.
A sequence matching algorithm finds the best matching between two grasping trials. A dictionary of sequences then constitutes a separate sequence-level embedding as follows. A video is subdivided into consecutive grasps using the detections. The sequence embedding of an individual grasp is then determined as the distance to each sequence belonging to the dictionary, where distances are calculated using the sequence matching algorithm.
For the behavior analysis the sequence embedding is applied on the sequences collected from all videos. The sequences are grouped based on the cohort and the time of recovery, from before the surgery (=Baseline), to 35 days after the surgery. We have implemented an approach in order to compare different recording sessions using the sequence embedding. Finally, each recording session is localized on a 2D plot which relates the recovery of different cohorts. The state of recovery at a specific time after stroke is therefore related with respect to the good behavior at baseline and the impaired behavior at 2 days.

2 Detection and Embedding

To run the paw detection use the function *graspingAnalysis_processVideo.m* on the example video (./video/00013.avi). This function can process each video separately, so that multiple videos can be processed in parallel by running multiple instances simultaneously.
The pipeline can be divided in three main parts: preprocessing, paw detection/posture embedding, and sequence matching/sequence embedding.

2.1 Preprocessing

Before starting with the preprocessing phase the frames of the video are extracted. During the execution of the source code, it is necessary to frequently load the original frames. Extracting the frames once at the beginning instead of loading the video several times during execution accelerates the whole process.

The following sub-tasks are executed during the preprocessing phase:

(1) - Finding the position of the shelf
The animals need to grasp through a vertical aperture in the Plexiglas wall before they can reach the sugar pellet, which is lying on a horizontal shelf in front of the wall. The position of the wall gives us the beginning of the shelf and is a useful prior information for the next steps. The function *./preprocessing/shelf_detection.m* detects the shelf using only a small subset of the extracted frames. Figure 1 shows an exemplary result after applying the function on a query video.

(2) - Finding the location of the sugar pellet
The coordinates of the paw are later on detected relatively to the position of the grasping target, which is the sugar pellet. Function *./preprocessing/detect_sugar_position.m* finds the location of the pellet. Since the camera position and settings may change between recording sessions, the paw detections are then made comparable based on the sugar position. Figure 2 displays an exemplary result of the source code.

(3) - Computing the mean frame
The mean image of a query video is a useful prior for the generation of

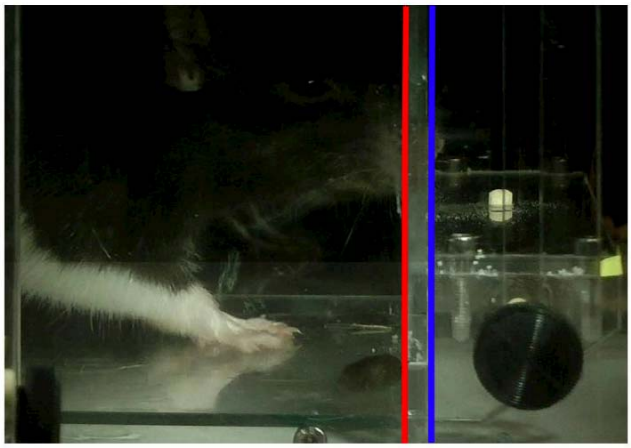

Figure 1: Detecting the vertical Plexiglas wall of the shelf - superimposed by a red (left) and blue (right) line

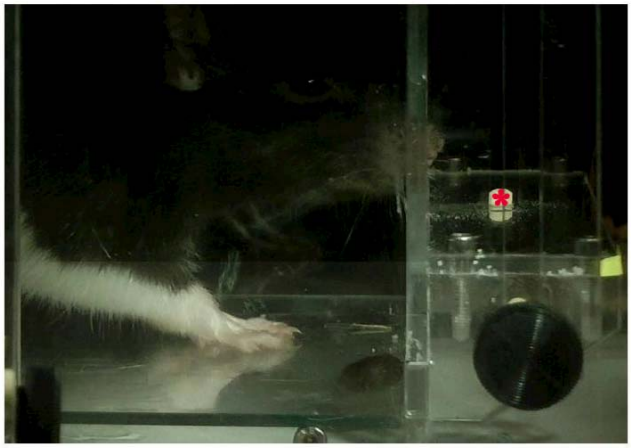

Figure 2: Sugar detection - superimposed by the red star

the feature descriptors. Due to the mean frame it is possible to remove the disruptive vertical Plexiglas wall in the feature space. It is computed by sum-

ming up all frames of a query video and a following division by the number of frames (exemplary result see Figure 3).

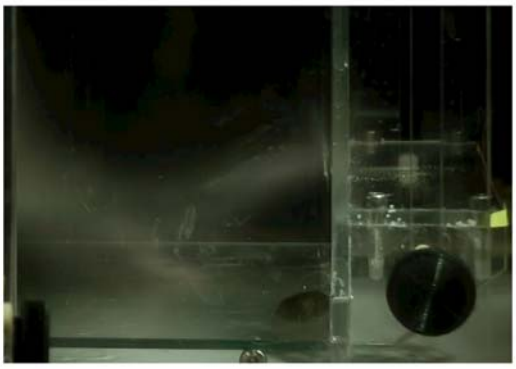

Figure 3: Exemplary mean frame of a video.

(4) - Generating the foreground map

The foreground map is generated by using robust PCA (J. Wright, Robust principal component analysis, NIPS 2009). The library is provided by the University of Illinois (http://perception.csl.illinois.edu/matrix-rank/sample_code.html). To be sure that the motion is estimated satisfactorily, the frames are processed in a random order. The resulting foreground maps reliably depict the foreground motion for every frame of a query video. Figure 4 shows some exemplary results gained by executing the function *./preprocessing/getForeground.m*.

(4) - Computing the HOG descriptor

The Histogram of Oriented Gradients (HOG) is a low-level feature descriptor, which describes the appearance and shape of objects shown on a query image (N. Dalal, Histograms of oriented gradients for human detection, CVPR, 2005). The algorithm determines normalized local histograms of image gradient orientations.HOG has shown to be an efficient yet reliable feature descriptor for subsequent classification using Support Vector Machines (SVM). VLFeat
(http://www.vlfeat.org/matlab/vl_hog.html) contains an implementation which

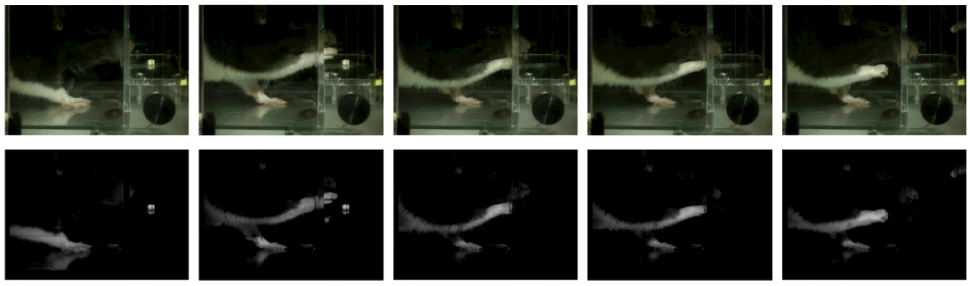

Figure 4: First row: query frames, Second row: resulting foreground map after applying robust PCA

computes the feature descriptor. VLFeat also provides the possibility of visualizing the results (see Figure 5). The oriented edgelets represent the calculated gradient orientation and strength per cell.

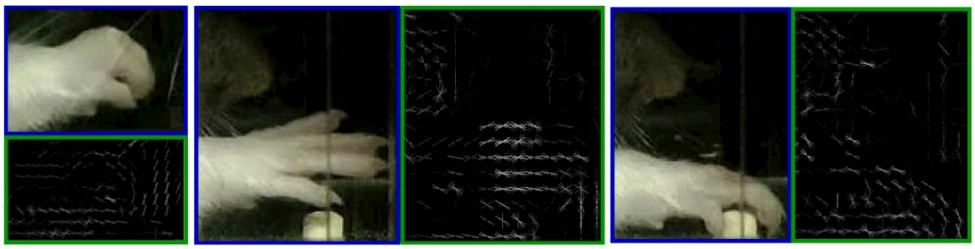

Figure 5: Examples of HOG descriptors extracted from three frames. In blue the original image and in green a sketchy visualization of the extracted HOG.

After generating the HOG for a specific frame with the function *./preprocessing/getHog.m*, the mean frame is utilized for erasing the vertical lines of the shelf in the HOG space. Figure 6 shows the HOG descriptor of an query image and the corresponding modified HOG after removing the shelf lines. Note that the visual difference is small, but experiments have shown that removing these occluding lines improves the detection results significantly. Besides the HOG a binary map which has the same size as the descriptor is created in the function *getHog.m*. This map is based on the foreground map and is also utilized during the subsequent detection process.

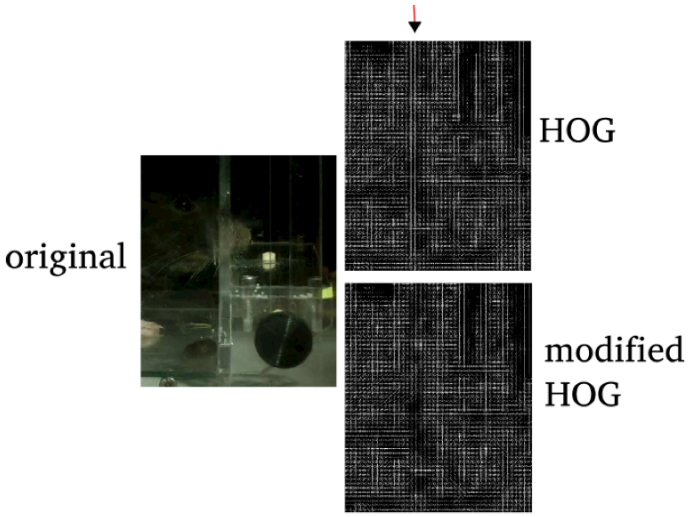

Figure 6: Illustration of the effect after removing the shelf lines with the help of the mean frame.

2.2 Detection

For detection we have proposed a max-projection of randomized versions of exemplar classifiers in the manuscript. Each discriminative exemplar classifier is trained using one positive and many randomly sampled negatives. The max-projection selects the most confident classifiers for each dimension. This compensates the unreliability of particular classifiers. Calibrating the classifiers with logistic regression assures the comparability of individual scores. Figure 7 displays a subset of the dictionary $\mathcal{D}$ of positive exemplar samples. Function *./preprocessing/patchDetector.m* performs the detection of the paw on a query video. The classifiers are applied densely over the individual frames. The final score of a specific location is computed by averaging over the highest k exemplars scores. Non-max suppression extracts then the highest three local maxima. Figure 8 shows some query frames which are superimposed by the detection scores received from the previously explained procedure.

The final detection (location with maximum score) of a query frame is determined by temporal smoothing and incorporating the a-priori distribution of paw locations (*./preprocessing/getMax.m*). Figure 9 displays the final detection of a query image, denoted by the blue bounding box. The shown frame

Figure 7: Subset of the Dictionary $\mathcal{D}$ of exemplars.

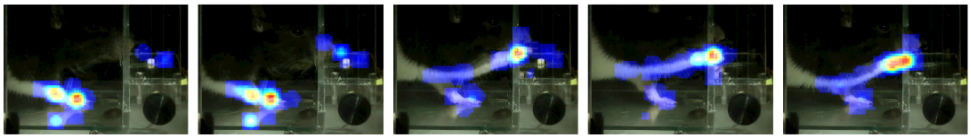

Figure 8: Illustration of the detection scores. Blue means low score, red means high score

is extracted from the video *detection/grasps.avi* (the video was created using the function *detection/createVideo.m*).

In the next step the detections are divided in individual grasps. A grasp

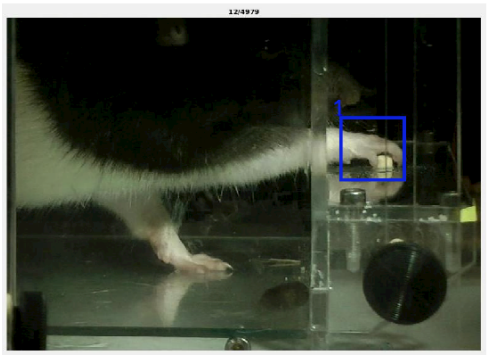

Figure 9: Detection of a query frame. The number above the bounding box denotes to which grasp the current frame belongs.

usually consists of a forward motion, the grasping moment and a backward

motion, whereas the grasping moment is characterized by a minimal absolute distance to the sugar pellet and a high x-coordinate value. Function *./preprocessing/getGrasps.m* finds these time points of grasping. Figure 10 depicts the oriented distance to the sugar location for around 200 frames (negative sign indicates a paw left of the pellet, positive on the right). The algorithm finds, in this case, four grasping points (blue) and the corresponding beginning and end (both in red) for each grasping trial. The red line represents the location of the shelf relative to the sugar position.

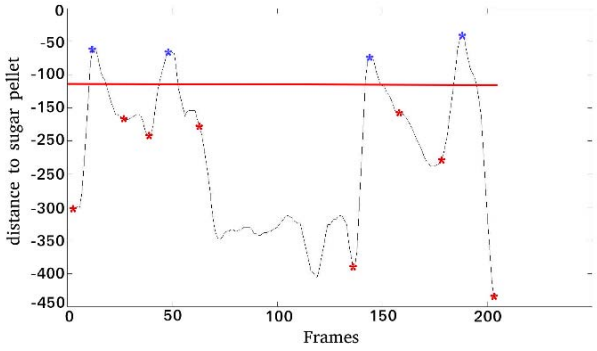

Figure 10: Oriented distances of paw to sugar pellet of a sequence of around 200 frames. The blue stars mark individual grasping points and the red ones the beginning and end of each grasping trial.

The **posture embedding** e of a specific detection x (p. 29 of manuscript) is a 120-dimensional vector of detection scores given by the exemplar classifiers $\{w_i\}_{i\in\mathcal{D}}$:

$$e := [< w_1, x >, ..., < w_{|D|}, x >].$$

It is a non-parametric representation of the hand posture. Figure 11 shows the nearest neighbor of some query frames calculated by using the cosine distance in the embedding space. It underlines that the detection scores provide a similarity measure for postures. The nearest neighbors are created using the function *./detection/plotNearestNeighbor.m*.

Figure 12 illustrates the resulting embedding of several detections after applying a distance preserving low-dimensional projection (t-SNE (L. Van der Maaten, Visualizing data using t-sne, JMLR, 2008)). Figure 3 in the submission shows all sequences per cohort, while individual grasps are linked using a

Figure 11: Nearest Neighbor of five query frames. All patches belong to the same video, but various grasping sequences.

polygonal chain. In Figure 12 below we depict only a subset of detections to also show individual postures (baseline video (before the stroke), where the animal belongs to the 'Delayed Training' cohort). The red boxes correspond to the areas in Figure 3B of the submission. Time is encoded from cyan to yellow. The Figure is created using *./detection/plotPatchesInTSNESpace.m.*

2.3 Sequence Matching

For a detailed behavior analysis of the grasping movement it is necessary to compare individual sequences with each other (find a similarity measure). Since the sequences are not temporally aligned, a sequence matching algorithm is required. The method finds an optimal matching between the individual frames of sequence j and sequence j' by considering certain constraints. A detailed description of the sequence matching algorithm can be found in the appendix of the original submission (Materials and Methods, section Spatiotemporal Parsing). The sequence matching is performed with the function *sequence_embedding/matchAllSeqs.m.* Note that a valid license of the cplex software of IBM is required to run this function.

The **sequence embedding** F_j of sequence j consists of the distances between j and all dictionary sequences $S_q \in D_{seq}$:

$$F' := [d(S', S_1), ..., d(S', S_Q)],$$

where the distance is given by the sequence matching algorithm.

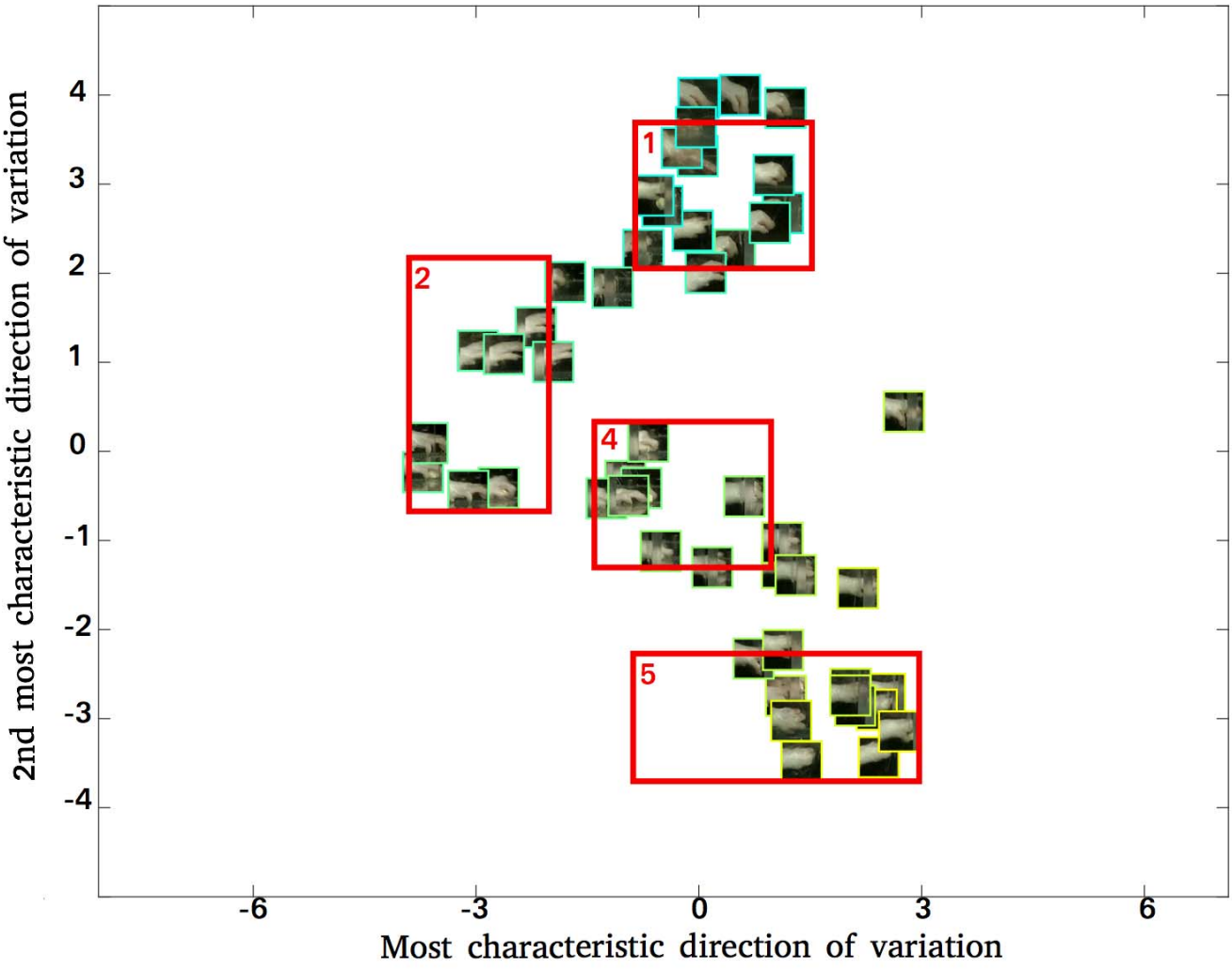

Figure 12: Illustration of the posture embedding from Figure 3B in the manuscript for a subset of detections in a video.

Figure 13: Subset of the sequence dictionary. Note the variability in appearance and color.

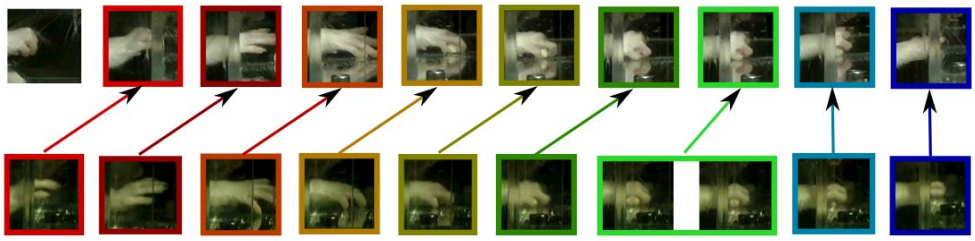

Figure 14: Exemplary result of the sequence matching algorithm. The algorithm matches each frame of the target sequence (bottom) to the frames of the query sequence (top). The colors indicate which frames are matched. Notice how the matching aligns the starting frame and it assigns two frames of the target sequence to one frame of the query sequence (light green) to align the speed of the grasps.

2.4 Recovery Analysis

(run file *RecoveryAnalysis/RecoveryAnalysis.m*)

The sequence embedding gives a visual representation for sequences useful for behavior analysis. Specifically, this experiment shows the improvement of animals during recovery. For this task, each recording session is com-

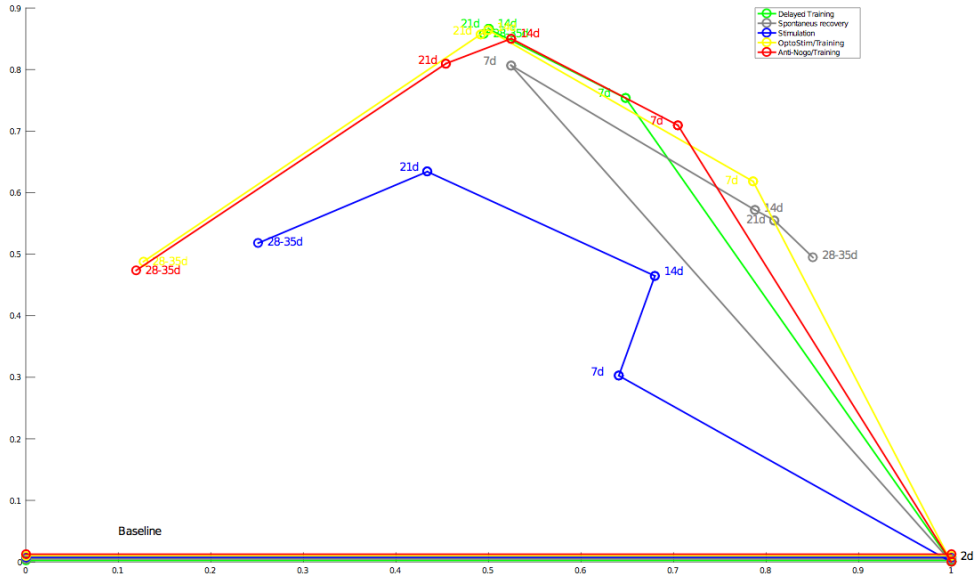

Figure 15: Result of *RecoveryAnalysis/RecoveryAnalysis.m*, which is identical to Figure 2C of the original manuscript. Each point is a recording session (cohort/time). From right to left, the figure shows the relative improvement of the animal behavior during recovery compared to 2days and Baseline

pared with healthy (Baseline) animals and impaired (2days) animals. In this way it is possible to relate the animal behavior at a specific time after stroke with respect to a good behavior and an impaired behavior. In Figure 15, for example, the recording of the OptoStim/Training cohort starting from impaired behavior (2days), become more similar to a healthy behaviour (Baseline) over time. This does not happen in the Spontaneous recovery cohort where the animals behavior stays close to impaired even after few weeks, except for an outlier (7days). For this analysis, it is clear that a similarity measure between recording sessions is needed. Given a specific recording session, the embeddings of all sequences form a multidimensional distribution. The similarity between two distributions is measured using the p-value resulting from the two sample KS-test. For computing the KS-test we need to project our data onto a single dimension. In order to reduce the dimensionality without losing the Baseline/2days relation, we apply Fisher LDA to find the projection which maximizes the distance between healthy

and impaired animals. The pairwise p-values between all cohort and days are computed in the function *./RecoveryAnalysis/distributionSimilarity.m*. Given p-values between all distributions, each recording session is localized in a two-dimensional plot using the similarity to Baseline and 2 days as references (function plot_triangulation.m). In order to visualize each point on the 2D plot, while preserving the distance information, a triangulation method is applied: the position of each point is calculated as the (upper) intersection point of two circles. The first circle has its counterpoint at Baseline and the radius is the distance to Baseline, the second at 2 days. In the resulting Figure 15, sessions appearing towards the left exhibit behavior closer to baseline, those at the right are closer to the impaired state right after stroke.