Introduction

Despite the established role of social behavior in the field of behavioral neuroscience, it is still a black box on how social context interact with biological functions to produce protracted and chronic sequalae. Difficulties with a gold standard of reliable scoring on complex social interactions has hinder the preclinical study of complex social behavior. Scoring are usually done by a few very experienced researchers for appropriate inter-rater-reliability validation, observing social behaviors and manually scoring previously defined events and their durations. This process is very tedious; prone to human error; long analysis times; and poor inter-rater-reliability. These impede detailed research on the complex social behaviors on larger datasets.

To overcome this, there are a range of open-source tools that use various forms of computer vision with synchronized RFID-tracking data and/or depth camera or multi-camera 3D systems have been developed for automated and precise classifications of animal behavior. However, they are very expensive as they do require significant investments in specialized hardware and significant computational and programming knowledge. Learning to run open-source libraries such as Pytorch, TensorFlow, OpenCV, sklearn, and many more packages requires time to learn how to use them and some sort of programming and computational background. Moreover, getting the libraries installed is also another challenge.

Last year, I built an open-source tools with a built-in graphical user-interface (GUI) called SimBA (Simple Behavioral Analysis, that is used to generate supervised decision ensembles of complex social behaviors from basic video recordings of mouse and rat dyadic encounters. SimBA uses features generated from pose-estimation tracking data together with experimenter-made annotations to generate random forests algorithms that accurately classify behavioral patterns in experimental videos.

SimBA is capable of predicting behavior with the current pose-estimation tracking data. Unfortunately, it is still not good enough if it were to use to predict behavior with minimum movements. Besides, the tracking data sometimes tend to jitter and becomes unreliable. Recently, we want to measure the breathing rates of the mice by using machine vision approaches and possibly predict death. This approach failed because it takes the pose-estimation of the rib cages of a stationary mouse and the movement of the tracking data is small and the tracking data is jumpy. In order to achieve that, the tracking has to be smoothened so the frequency of breathing can be measured by machine vision approach.

Here I proposed to use Kalman filtering to smoothened the tracking data and to correct the outlier for absolute pose estimation. I will incorporate a python library, pykalman into SimBA to smoothen the tracking data. Pykalman is a Kalman filter, Kalman smoother, and EM library for Python. Kalman filters are ideal for the pose estimation data which are in a continuous state space. The underlying model is a hidden Markov model (HMM) in which everything is multivariate normal, the hidden variables are continuous, rather than discrete. The Kalman filter is just the forward algorithm, except that each step can be computed analytically due to the magic of Gaussians. On the other hand, the backward algorithm is referred to as the smoother algorithm. The smoother allows one to refine estimates of previous states, in the light of later observations. As in the case of discrete-state HMMs, the results of the Kalman filter and smoother can also be combined with expectation-maximization to estimate the parameters of the model. They are all light on memory as they do not keep any history other than the previous state. Besides, they are very fast, making them well suited for time series problems and embedded systems.

Methods

Depends on the video, the user can choose between two algorithms for the tracking data. They are the Kalman Filter and Kalman Smoother. Traditionally, the model parameters are specified by hand, but this module allows the model parameters to be learned by the implemented Expectation-Maximization (EM) algorithm without any labeled training data. In this case, I would like to measure the breathing rate of a stationary mouse. Therefore, it is more optimal if I apply the Kalman Smoother. There are two steps to be done before applying the Kalman Smoother/Filter and they are described in paragraph two and three.

The first step is to pre-process the videos using SimBA. SimBA's video pre-processing pipeline is powered by ffmpeg in the backend. It allows the user to shorten, crop, apply CLAHE (Contrast-limited adaptive histogram equalization) enhancement, overlay frame numbers, and apply filters, video format conversion, and extract frames.

The second step is running the videos through animal pose-estimation tracking pipeline. Video frames is first extracted randomly using k-means algorithm that cluster based on visual appearance. The user will then have to label the frames with defined body parts. The labelled data is then feed into DeeperCut network with ResNet as backbone to learn the pose estimation. Once the training is finished, the model will be saved for future usage. Once the model is well trained, videos are feed through the model to predict the body parts and the tracking data will be generated (.csv). This step can be repeated using active learning workflow to increase pose estimation performance. The user will look at the overlay of pose estimation data on the video and correct the pose. Then, the data will be feed into the previous model to retrain until it reached a satisfied performance.

After getting the tracking data, Kalman Smoother/Filter is applied to the estimated pose to smoothen the tracking. For Kalman Smoother application, the user only has to define the initial state mean and observation space of the tracking data in matrices. On the other hand, Kalman Filter assumes that model parameters are known by the user beforehand. However, with the built-in EM algorithm, it can learn the parameters and the hidden sequence of states can be predicted using the Kalman Smoother. The pipeline that I build is to run Kalman Smoother first with a transition matrix and observation matrix that changes based on the tracking data. It will then generate a smoothed tracking data (x position, y position), and a plot and video for the user to judge if it the data is good enough.

If the plot and videos are choppy and the user thinks that the estimated position should be smoother, the user can increase the observation covariance so that it is higher than the state covariance. This means that we have less confidence in our measurements than dynamics. This can be done by creating a new Kalman Smoother with the previous transition matrix, observation matrix, and initial state mean. The only different parameters this time is to define and set the observation covariance to 10 times the value estimated from the previous Kalman Smoother. The user can adjust the smoothness of the tracking data by adjusting the observation covariance matrix.

The frequency of breathing of the mouse can be measured by the movement of the rib cages over time. The movement of the rib cages can be determined by finding out the pixel per millimeter of the video and calculate the millimeter moved over time.

Results

Figure 1 shows a plot of a subset of the tracking data that was smoothed by Kalman Smoother. The red dots are the original data and the dash plots are the smoothed data. As you can see, although it is smoother than the original data, it is still kind of bumpy and the pose estimation in the video is jumpy.

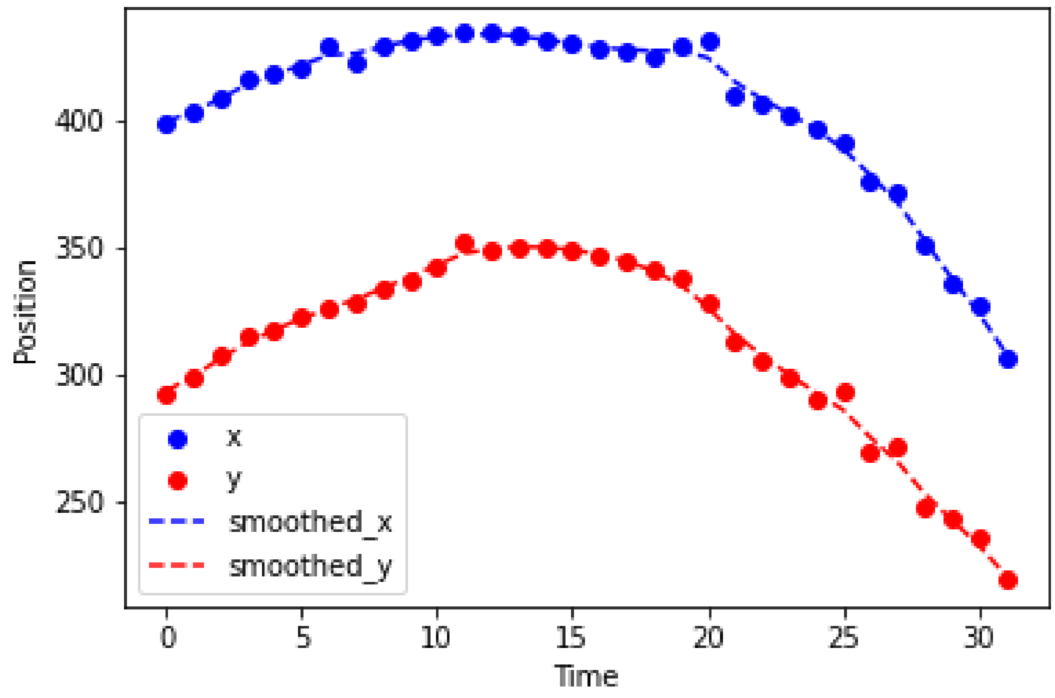

Figure 1: (x,y) position over time after running Kalman Smoother once.

Hence, I created a new Kalman Smoother with the previous parameters and change the observation covariance to be 10 times the observation covariance estimated previously. As you can see in Figure 2, the dash plots for smoothed_x and smoothed_y are much smoother and less bumpy. This is because we have a higher observation covariance compared to the state covariance.

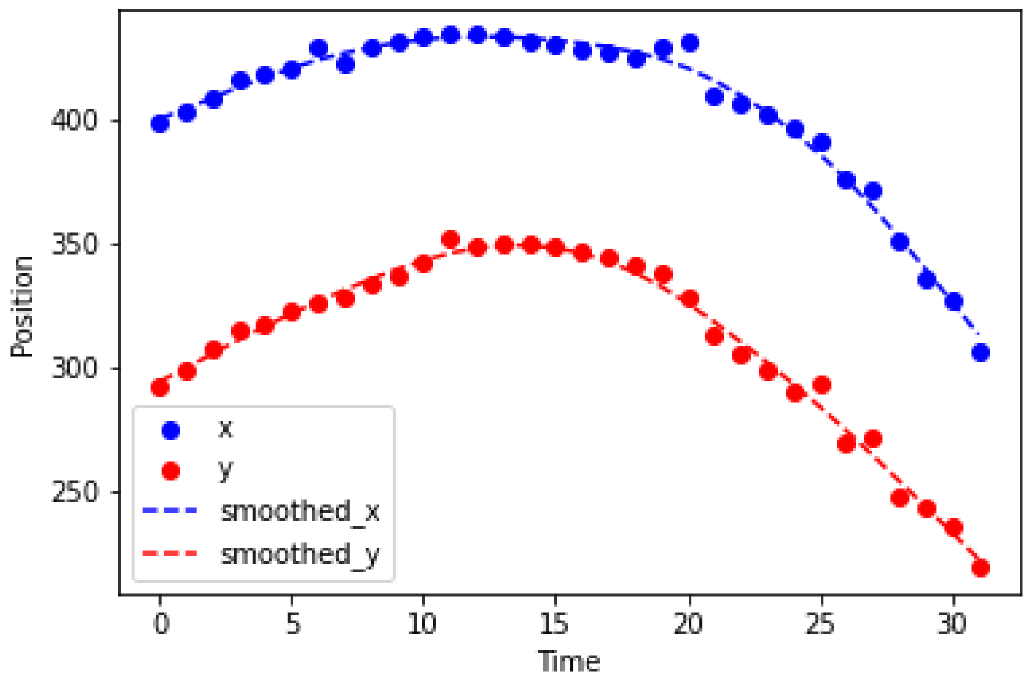

Figure 2:(x,y) over time after running Kalman Smoother with 10x observation covariance of the previous Kalman Smoother

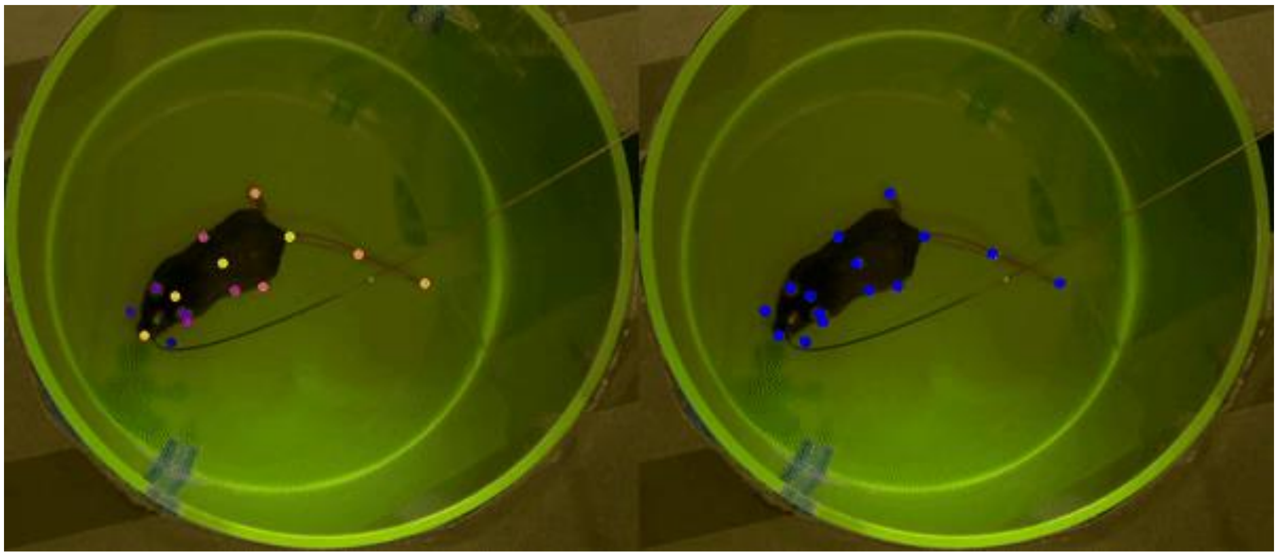

Figure 3: Video with pose estimation(left), and Video with pose estimation after Kalman Smoother (right)

The left side of the Figure 3 shows the pose estimation video after running through DeeperCut. The right side of Figure 3 shows the smoothed pose estimation data (after running Kalman Smoother) overlay onto the video on the left. In the video provided “dlc_level10.mp4”, it shows that the DeeperCut estimated pose (light color) jitters a lot, and the smoothed pose (blue color) does not jitter as much and are more stable. In the video, if you look at the right side, the yellow circles tend to move a lot and the blue circles are more stable. If both cases are stable, the user should not be able to see the light color circles as it is covered by the blue circles. However, as the pre Kalman Smoother data jitters a lot while the post Kalman Smoother data does not, yellow circles can be seen jumping around the blue circles.

Conclusions

Kalman Filter and Kalman Smoother improves the tracking data, resulting a more stable and accurate pose estimation data. However, the user will have to fine tune the settings to make it for best for their experiment. In order to make sure that this and the measurement of breathing frequency are accurate, the user can compare the results with the data obtained from rodent plethysmography.