Data Analysis:

Each flyGrAM run will produce a folder that is labeled with the Date and Time of the experiment. This folder contains two types of files: a recorded .avi video file of the experiment, which can be used for offline post-processing and .csv files for each ROI which denotes the number of flies moving in each frame for the duration of the experiment. Here we will explain how to create a summary .csv file of one or multiple experiments that contain flies of the same genotype and identical manipulations as well as a summary line graph plotting "Percent Group Activity" across time which can be saved as a .pdf using a custom written program in Python: "flyGrAM _analysis.py".

1. Create an experiment condition folder that contains all of the individual folders produced from each FlyGrAM experiment.

2. Download "flyGrAM _analysis.py" from Kaunlab github.

3. Start by creating an experiment key file in notepad (.csv) with "Datetime", "ROI", "Num_Flies" and "Treatment" columns. flyGrAM _analysis.py will use this key file to find the folders of each flyGrAM run, identify the genotype of the flies (i.e. GAL4 control, UAS control, and Experimental), and calculate the percentage of flies moving based on the number of flies in each ROI. Copy each folder name carefully into the "Datetime" column without any spaces. Inappropriate spaces will produce errors when running the analysis program.

4. Place the key file into the experimental condition folder.

5. Open flyGrAM _analysis.py and click "run". You will first be prompted to "Choose a Directory". Navigate to the appropriate experimental condition folder that contains the key files and folders for each individual experiment. Click "okay".

6. Next you will be prompted to "Choose your Key File". Select the .csv file with the experimental condition folder and click "okay". If your key file is not formatted correctly, you will typically receive an error saying "Error reading in roi. csv! Could not find roi with" {insert name of the folder that is problematic}." If this occurs, first make sure all of the folders listed in the key file are found within the experimental condition folder and then open the key file and correct for extra spaces surrounding each name or input before proceeding.

7. Once flyGrAM _analysis.py is running, you will next likely get an alert that Python "could not sort by 'Air:Ethanol flow rate, trying alternative sorting." You can ignore this alert. The alternative sorting will be by the "condition" you provided in your key file. After sorting, you will be promoted to "Please enter a stimuli label for the plot". Enter a label that indicates what the flies were exposed to (i.e. frequency and light for optogenetic experiments or concentrations of an odor exposure).

8. Once your folder has been sorted and binned, the program will ask if you would like to save the raw binned FlyGrAM data to a specific location. Click Yes. These data can be further processed in Python, R, or Matlab to create custom graphs for data visualization. The output is .csv for each designated group of flies that were identified under the "Treatment" column in your key file.

9. Lastly, the program will ask if you would like to save a pdf results file to a specific location. If so, you will be prompted to enter a desired name for the plotted pdf file.

Data Visualization and Statistics using R:

Using flyGrAM _analysis.py you were able to save a plotted pdf file of your data, however, often times, you might want to create a custom line graph with more visualization features. Instead of changing the graphing features in the flyGrAM _analysis.py code, we recommend inputting your binned data into RStudio for customization. This section will outline how to read .csv files and create custom line and boxplot graphs using ggplot2. This is by no means the only way to plot your data and run statistics. Feel free to explore other data visualization software as well.

1. Download FlyGrAM _Plot.R from the Kaun lab Github.

2. If you have never used RStudio before, you will need to install the appropriate packages that are listed at the beginning of the code. These have hashtags in front them to comment them out. “install.packages(“ “) with the package name within the quotes will download each package and install it on your device. Remove the hashtag and run the installation. You only need to do this once so once you are done you can comment them out again.

3. After installing each package, you will need to load the library in order to use it. Load each library using “library()”.

4. Once your libraries are loaded, it’s time to load your data. Load each .csv as a separate dataframe using the read.csv function and then bind each dataframe into one single dataframe with the cbind function.

5. If you click on the your new dataframe, you will notice that the columns have lengthy titles that aren’t easily grouped together. In order to group your data, create a list of genotype labels and rename the columns in your dataframe with the list using the colnames function. (genotype <- c(rep("A",4),rep("B",4),rep("C",4)) colnames(data)=genotype). It is easiest to use an A, B, C format. This is because ggplot automatically alphabetizes groups before plotting them so by using A, B, C, you can force the order of your groups in your plot. Just be sure to note which genotype corresponds with each label and make sure the repetition of labels in your list corresponds with the number of individuals in each group. In this case each group has an N of 4.

6. Individual data within each group is important for statistics, but in order to create the custom line graphs, you will need to calculate average data within each group and standard errors for those groups. Use the rowMeans function to calculate average group activity for each time point within each group and the custom written function to calculate standard errors for each group. Again, make sure the number of columns that you are averaging correspond with the number of individuals in each group. Lastly bind the average group activity into a single dataframe and the standard errors in a single dataframe.

7. Use the sequence function to create lists of 10 second bins for each group. (time<- c(rep(seq(10,1190,by=10), 3))). In this case, I have three groups so I repeated the sequence three times.

8. The ggplot package requires that your data is in melted format, which has "like measurements" in a single column. Your current data frame has several columns with similar measurements. Use the melt() function to convert your data frame into a data frame with only one set of observed values in each row. The resulting frame will contain two columns, group activity in one (value) and group labels in the second(variable). Next, create a data frame that includes your melted data, time, and standard error. A common error is that a data frame can't be made because the time is not the same length as data and standard error. If you receive this error, look at the lengths of your list and melted data to ensure they are consistent. If not, you can adjust your time list.

9. Now it's time to plot your data. Use ggplot to construct your initial plot and then map aesthetics as individual layers. Set your ymin, ymax, xmin, and xmax for your plot.

10. The annotate function allows you to layer on rectangles to denote the stimulus region as well as text to describe the stimulus.

11. Geom_line plots a line graph by variables, as dictated in the ggplot aesthetics. Using the scale_color_manual and hex codes you can tailor the color of your lines for each group. You can also label each of your groups that correspond to A, B, C groups previously used. Geom_ribbon plots the ribbon for standard error around each line. You can tailor the color of your ribbon for each group using hex codes in Scale_fill_manual.

12. If you are not happy with how ggplot breaks your x and y axes you can specify this using scale_x_continuous and scale_y_continuous. For example, scale_x_continuous(breaks=seq(200, 1200, 200) gives a range of times in seconds from 200 to 1200 by 200 seconds. scale_y_continuous(breaks=seq(0, 100, 25), gives a range for percent activity from 0 to 100 by 25. It's important to place these adjustments last because ggplot layers aesthetics on a plot so if this were to appear earlier it would be masked by the other x and y axes. Use ylab and xlab to custom label your axes.

13. The last bit of code within FlyGrAM _Plot.R customizes the position of the legend, removes the white background so it doesn't occlude the data and customizes the axes in terms of text size and color. Feel free to adjust these as you see fit.

14. Once you are satisfied with your plot you can save it as a pdf, png, or jpg using ggsave.

You can also use R to plot of group activity across genotypes for a single data point. Organizing your data in this way is especially helpful if you would like to run statistics. In this part of data visualization protocol, we will go over creating boxplots of group activity for different time ranges and overlay raw data.

15. In order to plot group activity for a specific time range, you need to create a new dataframe that extracts the group activity for just that time range. We can do this by limiting the rows that are included in the new dataframe. For example, if we wanted to capture the baseline (300 seconds before the onset of ethanol) we would simply run: data_baseline<-data[1:30,] and then average group activity for each individual N using colMeans.

16. Since we will be using ggplot to generate boxplots, you will again need to reshape your data in a melted format. You will notice that the code for making a boxplot is very similar to a line graph, however, instead of using geom_line, we will use geom_boxplot. You can

also layer the individual average group activity using geom_jitter. Feel free to experiment with different features to create your own customized graph.

17. Lastly, now in R you can also run statistics of group activity with different time ranges. In this case we can run an ANOVA using the aov function. In the example melted baseline data frame you have two columns, one that denotes the “value” or average group activity and one that denotes the “variable”. Test.aov<- aov(value~variable, data=mdatabase) will runs an ANOVA on the mdatabase dataset comparing value by variable. You can retrieve your results using summary(Test.aov).

18. You can also run a Tukey Posthoc on the ANOVA output using TukeyHSD(Test.aov).

19. The remaining code in FlyGrAM _Plot using the capture function allows you to save your stats results in a text file.

Optional Post-processing for High-Content Behavior Analysis:
The utility of flyGrAM largely lies in the automated real-time quantification of group locomotor activity. However, because one of the output files of flyGrAM is an .avi, opportunities exist for more advanced post-processing offline. Here we will explain how to format your video file using ffmpeg for post-processing with FlyTracker or Ctrax, and JAABA.

1. Download ffmpeg (https://www.wikihow.com/Install-FFmpeg-on-Windows). If you are using the same computer that you run flyGrAM, you should already have ffmpeg downloaded.

2. Open your c-terminal and navigate to your current working directory where you .avi files are located. It is recommended that you move all of your video files into a single folder for processing. Set your working directory using “>cd” followed by the path to your directory. If you need to back up into a folder use “cd..\”

3. First you need to reformat your .avi to preferred format (nv12). This expands your .avi to a raw video file so you can format it again in an appropriate format. You might want to delete this file once you have moved to the next step to save space. This will be about a 15GB file for 20 minutes or so of flyGrAM. “>ffmpeg -i videofilename.avi -pix_fmt nv12 -f avi -vcodec rawvideo newvideofilename.avi”

4. Next reformat your raw.avi file as a .mp4. “>ffmpeg -i newvideofilename.avi -pix_fmt yuv420p newvideofilename.mp4”

5. Now that your video file is appropriately formatted crop your video into 4 different ROIs so you can track individual flies separately. This is especially important if you have different number of flies in each ROI. The filter crop feature gives “width”:”height”:”x starting point”:”y starting point”. Given that the flyGrAM apparatus is not always in the same position, you will have to adjust these values to what works best for your video. Use ffplay to check your cropping before saving them.
    a. Crop RO1- top left quartile: >ffmpeg -i in.mp4 -filter:v "crop=240:240:75:0" -c:a copy outroi1.mp4
    b. Crop RO2- bottom left quartile: >ffmpeg -i in.mp4 -filter:v "crop=230:240:75:240" -c:a copyroi2.mp4

c. Crop RO3- bottom left quartile: >ffmpeg -i in.mp4 -filter:v "crop=230:240:380:0" -c:a copyroi3.mp4
d. Crop RO4- bottom left quartile: >ffmpeg -i in.mp4 -filter:v "crop=230:240:380:240" -c:a copyroi4.mp4

6. Lastly you adjust different filter corrections (https://ffmpeg.org/ffmpeg-filters.html#eq). Use ffplay to preview your file with adjusted parameters.