Behavior Analysis Pipeline v2

**Raghuveer Parthasarathy**

Last modified: August 28, 2024

# History

Version 1; based on Estelle Trieu’s code and my (major) modifications. Notes May 21-23, 2023

**Version 2:** Major revisions of everything – Re-writing all the code! Correcting errors; 7x speedup; etc. Begun May 2023. See “Behavior Code Revisions May 2023” … “… November 2023” documents.

# Python code

In progress versions in \Zebrafish behavior\Code\_current

Stable versions in \GitHub\Social-Behavior\BehaviorAnalysis

Documentation (incomplete) in \GitHub\Social-Behavior

# Required Python packages

numpy matplotlib pickle yaml csv pandas os xlsxwriter scipy time

# Required MAT and Excel files (ZebraZoom output and experiment description)

**MAT files.** There must be one or more .MAT files containing zebrafish trajectory outputs, from ZebraZoom. Each MAT file may contain multiple behavior datasets. The .MAT files are typically named something like “results\_SocPrefInf\_2b\_t1.MAT”.

**Excel file.** The Excel file should contain headers like “Trial Trial\_ID, Pair\_ID, BodySize, Cond\_Code, PixelSize, Include”. The “Trial\_ID” and “Pair\_ID” are important for identifying the appropriate trajectory data in the .MAT file. “PixelSize” is the image scale, μm/px. “Cond\_Code” or something similar is an integer code indicating the experimental condition. The Excel file is typically named something like “SocPrefInf\_2b\_RaghuAnalysis.xlsx”.

**Location.** Place the above files in a folder, probably a sub-folder of “…\Zebrafish behavior\MAT files\”

# Processing tracking output (MAT → CSV)

## Generate CSV files

ZebraZoom’s output of fish body positions and headings in each frame is saved in MAT files as structured MATLAB arrays. We must convert this data to CSV files, one per dataset, for further analysis.

The MATLAB program **convert\_fishTrackingData\_MATtoCSV.m** reads MAT files and writes data as a CSV file of the fish body positions, one per movie. There can be multiple MAT files, and each MAT file can contain multiple movies. Running this takes about 1 second per movie dataset. The output CSV files are in the same folder as the MAT file – see the next section about moving these files.

There are various inputs, shown below; the “Nfish” input is required, and is the number of fish in each movie of the behavior assay (in all data so far, this is either 1 or 2). The program checks that the number of fish in the datafiles matches this.

The program also reads the image offset positions, which are in the videoDataResults.wellPositions{k} structure for movie k as fields (topLeftX, topLeftY, topLeftX, topLeftY), and writes these to a separate CSV file (given by the wellOffsetPositionsCSVfile variable, probably “wellOffsetPositionsCSVfile.csv”), used in later analysis to determine the arena centers (see below).

**Example (MATLAB code):**

Nfish = 2;

dataDir = 'C:\Users\Raghu\Documents\Experiments and Projects\Misc\Zebrafish behavior\MAT files\2 week old - infected versus non-infected, solitary versus co-housed pairs';

wellOffsetPositionsCSVfile = 'wellOffsetPositionsCSVfile.csv';

cd 'C:\Users\Raghu\Documents\Experiments and Projects\Zebrafish behavior\Code\_current'

convert\_fishTrackingData\_MATtoCSV(Nfish, dataDir, [], wellOffsetPositionsCSVfile, []);

## Move CSV files

When finished, CSV files will be in the dataDir directory. We must move them to a new location, either a single folder or subfolders based on group or condition code.

Either manually move the CSV files to a folder (or group-based subfolders) in “\CSV files and outputs” or use process\_excel\_and\_csvs() in sort\_CSVfiles.py to sort by group or condition code into separate sub-folders. Note that process\_excel\_and\_csvs() also copies **wellOffsetPositionsCSVfile.csv**, copies the Excel file of experiment information, and makes a CSV version of this Excel file in each subfolder that contains only the row information for that subfolder. (It’s fine if **wellOffsetPositionsCSVfile.csv** contains information about datasets that aren’t present in that subfolder.)

This code isn’t very robust. If pair\_ID identification fails, ask for user input – probably the last number in the trial\_ID string. Also, if there aren’t subgroups, it’s easier to move/copy by hand.

CAUTION: Check manually that all CSV files were moved, and that “wellOffsetPositionsCSVfile” was copied. (Copy manually if not.)

**Example (Python code):**

Type the following at “Replace these” in the end section of sort\_CSVfiles.py .

excel\_fileName = r'SocPrefInf\_2b\_RaghuAnalysis.xlsx'

sourceMATpath = basePath + r'\MAT files\2 week-old - pairs exposed to water conditioned by infected 6-dpf larvae'

destination\_mainCSV\_path = basePath + r'\CSV files and outputs\2 week-old - pairs exposed to water conditioned by infected 6-dpf larvae'

group\_code\_label = 'Cond\_Code' # may be 'Group\_Code'

include1\_label = 'Include' # Header of the "include" column

include2\_label = None # use None to avoid additional filtering, or 'Filter' to filter

subfolder\_name = 'Condition' # will append the group code to this for sub-folders

wellOffsetPositionsCSVfilename = 'wellOffsetPositionsCSVfile.csv' # Probably don't need to change

excludeCSVList = ['wellOffsetPositionsCSVfile.csv',

'SocDef\_Shank3\_AnalysisRaghu.csv'] # ignore these CSVs

Then run sort\_CSVfiles

# Required data and locations

Each dataset must have:

* **A CSV file of positions.** See “Generate CSV files,” above. The CSV filenames are used for the “dataset” variable names.
* **A CSV file containing experiment parameters, including the image scale.** Required for “Image Scale” and “Arena Offsets” below. Probably an Excel file originally; save as CSV. Automatically copied and split into subsets (groups) by process\_excel\_and\_csvs() – see above. The Excel and CSV files have a header row, but some earlier subsets might not – this should not cause problems. E.g. For 2023 datasets, **ArenaCenters\_SocPref\_3456.xslx** (**.csv**). For March 2024 Social / Co-housed, **SocDef\_Solitary\_AnalysisRaghu.xlsx (.csv).** Note that the CSV filenames of trajectories must match the names in the first column of this CSV, except for “\_light” and “\_dark” and other strings (like 'results\_', 'SocDef\_') specified in get\_ArenaCenter() in toolkit.py. For now, may need to adjust these strings manually.
* **An entry in the configuration file, all\_expt\_configs.yaml** . This must contain the frames per second (fps, almost certainly 25), arena radius (almost certainly 25 mm), the filename of the CSV file containing well offset positions (almost certainly wellOffsetPositionsCSVfile.csv), and more. See “experiment configuration file,” below.
* The configuration file also specifies the file in which the image scale is noted, imageScalePathName and imageScaleFilename, as well as the column index (0==first) that contains the image scale, imageScaleColumn.
* **Image scale.** The location of the image scale parameter is saved the location indicated in the experiment configuration file. For 2023 datasets, the image scale is noted in fifth column of **ArenaCenters\_SocPref\_3456.xslx** (**.csv**), mean ±- std = 56.7 ± 0.6 um/px . For 2024 cholic acid, for example, the scale is in “**SocPrefBA\_3b\_AnalysisRaghu.csv**” and is (mean ±- std = 96.4 ± 1.1 um/px). The column number is an image parameter that must be specified in the experiment configuration file (see below).
* **Arena offsets (well offsets).** These indicate the location of the arena (well) in the camera field. Saved as a row in a CSV file. Each row: CSV filename (without .csv) | topLeftX | topLeftY | lengthX | lengthY . The list “datasetRemoveStrings” in the experimental configuration file specifies strings to remove to find matches between the first column and the CSV file name (see below). The location of this information (filename, column) is noted in the experimental configuration file. (Likely default “wellOffsetPositionsCSVfile.csv”).
* **Arena centers.** These are the x and y positions of the centers of the arenas. The position values need the arena offsets to be subtracted to make sense. However, if arena centers aren’t noted they can be estimated from the well offset positions as topLeftX + lengthX/2 and topLeftY + lengthY/2 . For 2023 datasets, arena centers are in “ArenaCenters\_SocPref\_3456.csv” in the main folder, in columns 5 and 6 (0==first column). Note that this csv file has a header row. For most other datasets, the arena centers are estimated by the program. See arenaCentersLocation and arenaCentersColumns in the experimental configuration file; use arenaCentersLocation=None indicating that the center should be estimated.
* **Arena radius.** For 2-week-old fish, arena radius is always 25mm (50mm diameter). This parameter is saved in the experimental configuration file as arena\_radius\_mm .

# Folder structure

* Each experiment will have a “CSV files” folder that contains subgroup folders with the subgroup CSV files; if there aren't subgroups, the CSV files will be in “CSV files.” The folder doesn’t have to be called “CSV files.”
* Each “CSV files” folder must contain an experiment configuration file, called “expt\_config.yaml” . This configuration file describes this experiment only.
* Each “CSV files” folder must also contain the configuration files that describe CSV structure and analysis parameters: analysis\_parameters.yaml, CSVcolumns.yaml
* When behaviors\_main.py is run, the user need only specify the location of the "CSV files" folder. The user can type the full folder path or leave it empty to select it using a dialog box. (Caution: the dialog box may be hidden.) The program checks that “CSV files” is in the lowest-level part of the path name, and checks that “expt\_config.yaml” , “analysis\_parameters.yaml” and “CSVcolumns.yaml” are all in this “CSV files” folder.

# Experiment Configuration File

Each experiment must have an experiment configuration file, written in YAML, in its “base” directory. This file must be called “expt\_config.yaml” and must have the following (note that some are optional, and the experiment name doesn’t matter):

**EXPERIMENT NAME:**

**fps:** # Frames per second (same for all in this experiment)

**arena\_radius\_mm:** # arena radius, mm (same for all in this experiment)

**imageScale:** # (optional) image scale, um/px, global (for all in this experiment) or leave empty to load from file for each dataset

**imageScalePathName:** # Optional; leave empty if location is this folder (basePath). Path name of folder containing the CSV with image scale for each dataset; will append to basePath

**imageScaleFilename:** # Optional; file name of CSV with image scale for each dataset, if loading from file (i.e. if imageScale is empty)

**imageScaleColumn:** 4 # (optional) column (0-indexed) in CSV containing image scale for each dataset, if loading from file (i.e. if imageScale is None)

**datasetColumns:** [0,1] # columns to concatenate in the imageScale CSV to match dataset name (cat with "\_" between). For example, if [0, 1], will combine col 0 “SocDef\_Shank3\_1a\_t1” and col 1 “6” to be “SocDef\_Shank3\_1a\_t1\_6”

**datasetRemoveStrings:** # strings to remove from filenames and first column of the imageScale CSV to find matches. Also applies to the ArenaCenters CSV, if that's used. For example, file name may be “results\_SocDef\_Shank3\_1a\_t6\_6.csv”, and this will match dataset name “SocDef\_Shank3\_1a\_t1\_6” if datasetRemoveStrings is ['SocDef\_', 'results\_'] Removes these strings in the order of the list.

**offsetPositionsFilename:** # CSV file with well offset positions, almost certainly “wellOffsetPositionsCSVfile.csv”

# Arena center location information (optional; empty to estimate from well offset positions)

**arenaCentersPathName:** # # Path name of folder containing the arena centers file; will append to basePath. Leave empty to estimate from well offset positions

**arenaCentersFilename:** # CSV file name containing arena centers information

**arenaCentersColumns:** [5,6] # columns (0-indexed) with x,y arena centers

**subGroups:** # strings of subFolder names with data from experimental groups with the same overall parameters, e.g. ['Genotype 1', 'Genotype 2', 'Genotype 3']

# CSV column information file

There must be a configuration file that describes the columns of the CSV files, written in YAML. This file is “CSVcolumns.yaml” …\Zebrafish behavior\CSV files and outputs\ ; path and file name are hard-coded in behaviors\_main.py.

The contents are:

# Specify columns of the CSV files with fish trajectory information

**CSVcolumns:**

N\_columns: 26 # total number of columns in CSV file

head\_column\_x: 3 # head position x column (first col == 0)

head\_column\_y: 4 # head position y column (first col == 0)

angle\_data\_column: 5 # angle (radians) column (first col == 0)

body\_column\_x\_start: 6 # starting column for body x positions (first==0)

body\_column\_y\_start: 16 # starting column for body x positions (first==0)

body\_Ncolumns: 10 # number of body datapoints

# Behavior Analysis Procedure

Analysis of a single experimental condition, consisting of multiple datasets.

Analyses all CSV files of body positions in a folder; one CSV file per dataset.

## Edit all\_expt\_configs.yaml

If this experiment set hasn’t been examined before, name it and add its image parameters to the configuration file all\_expt\_configs.yaml.

See “Experiment Configuration File” above.

## Edit CSVcolumns.yaml

In the unlikely event that the structure of the CSV output has changed, edit the CSV column configuration file.

## Edit analysis\_parameters.yaml

Edit the configuration file with all the behavior analysis parameters (if you’re modifying these). Make sure the correct configuration file is written in the behaviors\_main.py file, at “# Get behavior analysis parameter info”.

Note that this includes the names of the output files, probably “behavior\_count.csv” etc., and the output subfolder.

## Run behaviors\_main.py

This is the main program that loads CSV files, analyzes them, and outputs behavior characterizations. It takes about 4 seconds per CSV file.

**To use:**

When prompted, enter or select the “base” path name. The “base” path name is the higher-level folder that contains other folders that contain CSV fish trajectory files and outputs.

The “base” folder must contain the configuration files (all\_expt\_configs.yaml, analysis\_parameters.yaml, CSVcolumns.yaml).

Run behaviors\_main.py and follow the prompts:

Select the experiment. This will load parameters from the experiment configuration file.

*Optional, recommended:* Input the name of a .pickle file to write, containing all the analysis variables and outputs, e.g. “all\_2week\_light” – the program will append .pickle to the file name.

## Outputs of behaviors\_main.py

The program will generate several files in the output\_subFolder directory (probably called “Analysis”):

* A summary text file for each CSV file with information including the mean inter-fish distance and list of all frames with detected behaviors. Filename: same as the dataset name, .txt .
* An Excel workbook “behaviors\_in\_each\_frame.xlsx” with a sheet for each dataset, indicating with an “X” behaviors (columns) found in each frame (rows).
* A CSV file for each dataset that gives the basic measurements for each frame. Name: dataset + “\_basicMeasurements”.csv:
  + Head-to-head distance (mm) ["head\_head\_distance\_mm"]
  + The closest distance between fish (mm) ["closest\_distance\_mm"]
  + Speeds of each fish (mm/s) ["speed\_array\_mm\_s"].
  + Relative orientation angle of each fish with respect to head-to-head vector (radians) ["relative\_orientation"]
  + Distance to edge of each fish (mm)
  + Flag (0 or 1) for any fish being near the edge.
  + Flag (0 or 1) for any fish having bad tracking
* An Excel file with summary statistics of each behavior for each dataset, indicating the number of events, duration (number of frames), and relative duration of each of the behaviors. Each row is one dataset. Each column is one behavior; the first few columns are general dataset properties. In addition, separated by a blank row, each sheet contains statistics over the datasets (mean, std. dev., and s.e.m.). The Excel file is probably named “behavior\_counts” + “\_” + [subGroup name] + “.xlsx” . The base file name is specified in the analysis parameter configuration file, and the subgroup name if it exists is appended. This file is the most important output, and you may wish to rename it in the configuration file or afterwards.
* A .yaml file called all\_params.yaml with all the parameters used, merging expt\_config, analysis\_parameters, and the analysis/output path (dataPath).

Note: also returns “datasets”

# Comparison of Experiments

Comparison of datasets from two experimental conditions or groups, analyzing the single-experiment analysis outputs.

Makes plots that compare the relative durations of behaviors in two different experiment groups, such as *2 week old light* and *2 week old dark*. Plots each behavior versus each other (log-log plot) and ratio of behaviors in the groups. Allows excluding various behaviors (columns).

**To use:**

Recommended: Close all figure windows. (They can cause problems with the dialog box.)

Run compare\_experiment\_behaviors.py. From the dialog boxes, select the first and then the second “behavior\_counts.xlsx” file – i.e. the Excel file in which relative durations of each behavior, along with statistics, are stored in a “Relative Durations” sheet.

When prompted either type the name of the output folder for plots, or leave this blank for a dialog box.

When prompted, enter the “base” name of the output files for the plots, including the extension that specifies the image file type. The program will add '\_relBehaviorPlot' and '\_relBehaviorRatios' to this, so for example “this\_expt.eps” gives outputs ““this\_expt\_relBehaviorPlot.eps” and “this\_expt\_relBehaviorRatios.eps”

We exclude various columns/keys (i.e. behaviors) from the comparison. There is a separate list for the log-log plot and the ratio lot. These are hard-coded; you may wish to re-run this with different excluded columns.

# behaviors\_main.py

Contains the main analysis functions. Run this program for single-experiment analysis.

Writes a pickle file containing all datasets (optional)

## main()

*Main function for calling data reading functions, basic analysis functions, and behavior analysis functions for all CSV files in a set*

Load experiment configuration file; call load\_expt\_config(); dialog box if file not found.

Get CSV column info from configuration file; dialog box if file not found.

Get behavior analysis parameter info from configuration file

Get names of all CSV datasets: all CSV data files with names that begin with “Results”. Stores the file names in a list: “allCSVfileNames” . Calls get\_CSV\_folder\_and\_filenames()

Modify Excel output filename if there are subgroups

Get pickle output filename

Set “showAllPositions” for displaying positions (diagnostic; typically False). If True, plotAllPositions() will be called to show all head positions, dish edge in a separate figure for each dataset.

Call load\_all\_position\_data(). For all CSV files in the list, this calls load\_data() to load all position data and determine general parameters such as fps and scale. Optional (if showAllPositions==True): make a figure with all head positions for the full dataset, calling plotAllPositions()

For each dataset:

* Identify frames in which fish are close to the edge or in which tracking is bad; calls get\_edge\_frames(), get\_bad\_headTrack\_frames(), get\_bad\_bodyTrack\_frames()
* Also call make\_frames\_dictionary() to store this frame information.

For each dataset, identify characteristics or behaviors that involve single fish. Wrapper function get\_single\_fish\_characterizations(). This calls:

* Fish lengths from get\_fish\_lengths();
* Speed from get\_fish\_speeds(); also “isMoving” characterization, for each fish, and for “any” and “all” fish.
* Average speeds, for all good tracking frames and for the subset that “isMoving”
* Tail angle from getTailAngle()
* C-bends, J-bends from get\_Cbend\_frames(), get\_Jbend\_frames(). For C- and J-bends, exclude frames in which fish are close to the edge or in which tracking is bad; create a frames dictionary.

For each dataset, perform “basic” two-fish characterizations such as inter-fish distance and , relative orientation, if Nfish > 1. Wrapper function get\_basic\_two\_fish\_characterizations(). This calls:

Get the inter-fish distance – both the distance between head positions and the closest distance between any points – in each frame. Calls get\_interfish\_distance() .

Get the sliding window cross-correlation of heading angles for the two fish

For each dataset calls extract\_behaviors() to identify and tabulate events; for each behavior, a dictionary containing frames.

Write count information to individual text files, individual Excel sheets, and summary CSV file. The CSV file that combines information from all datasets: allDatasetsCSVfileName . Writes each behavior counts and durations, and proximity-to-edge frames, and bad tracking (body) frames.

## extract\_behaviors()

Calls functions to identify frames corresponding to each behavior.

[write]

# datasets: a list of dictionaries

I save information in a list of dictionaries called “datasets”. There is one dictionary for each experiment. Dataset *j* for example will have its CSV filename (see Keys below) in datasets[j]["CSVfilename"].

**Keys for each experiment’s dataset:**

## Keys containing general experimental information or input data

* arena\_center: tuple of (x,y) positions of the Arena Center, from get\_ArenaCenter(), which reads center and offset information from CSV files.
* all\_data : a numpy array (Nframes x Ncolumns x Nfish) containing all the trajectory data for this experiment, from load\_data()
* CSVfilename : the name of the CSV file containing all trajectory info.
* dataset\_name: the dataset name, extracted from the CSV filename by “get\_dataset\_name()”. (E.g. “3c\_2wpf\_k2” from “results\_SocPref\_3c\_2wpf\_k2\_ALL.csv” .)
* frameArray : 1D array containing all the frame numbers, from load\_data()
* image\_scale : the image scale (um/px), extracted from the appropriate CSV file by get\_imageScale()
* Nframes : the number of frames, set equal to the length of frameArray
* Nfish : the number of fish, set equal to datasets[j]["all\_data"].shape[2]
* total\_time\_seconds: total time (max frames – min frames + 1)/fps, seconds
* fps: frames per second, copied from expt\_config["fps"]

## Keys containing single-fish Information

* bad\_headTrack\_frames : array of frames in which one or more fish head positions are bad (zero). From get\_bad\_headTrack\_frames().
* bad\_bodyTrack\_frames: array of frames in which one or more fish body positions are bad (zero). Get frame array from get\_bad\_bodyTrack\_frames(), then convert to a list of dictionaries (see **behavior dictionaries**, below) to also calculate runs, durations
* Cbend\_Fish{j}, Cbend\_any: frames in which Fish *j* (*j* = 0, 1, 2, …, Nfish-1), or any fish shows a sharp bend. Each is a list of dictionaries (see **behavior dictionaries**, below).
* d\_to\_edge\_mm: Nframes x Nfish array of distance of each fish to dish edge, mm
* edge\_frames : frames in which any fish is close to the edge. Get frame array from get\_edge\_frames(), then convert to a list of dictionaries (see **behavior dictionaries**, below) to also calculate runs, durations.
* fish\_length\_array\_mm : length of each of the two fish in each frame (sum of all segments); mm; Nframes x Nfish array; from get\_fish\_lengths().
* fish\_length\_mm\_mean: mean fish length, mm, ignoring bad-tracking frames; averaged over both fish.
* fish\_length\_Delta\_mm\_mean and fish\_length\_Delta\_std: mean over frames of absolute difference in fish length, mm, ignoring bad-tracking frames, and standard deviation.
* Jbend\_Fish0, Jbend\_Fish1, Jbend\_any: frames in which Fish 0, Fish 1, or either fish shows a J-shaped bend, with a straight anterior half and a curved posterior. Each is a list of dictionaries (see **behavior dictionaries**, below).
* speed\_array\_mm\_s : speed of each fish, frame-to-frame, mm/second using scale and fps information; Nframes x Nfish ; from get\_fish\_speeds(). This is the frame-to-frame displacement, with speed[j] being position[j]-position[j-1], and the first element of speed set to zero so that there will be Nframes elements. (Note that zero for the first frame makes it easier to compare speed[j] and tail-angle[j] .
* speed\_mm\_s\_mean: mean fish speed, mm/s, ignoring bad-tracking frames; averaged over both fish.
* speed\_whenMoving\_mm\_s\_mean: mean fish speed only for frames that meet the isMoving criterion, mm/s, ignoring bad-tracking frames; averaged over both fish.
* tail\_angle\_rad : tail angle of each fish, radians, calculated as the difference between the angle of the position 8-to-9 segment and the heading angle. (Use positions 8-9 rather than the final 9-10 because the latter segment is often very short.)
* isMoving\_any : frames in which any fish has speed above threshold, i.e. is moving. Note that this only excludes bad Tracking frames (dilated +1),, not near-edge frames.
* isMoving\_all : frames in which all fish have speed above threshold, i.e. are moving. Note that this only excludes bad Tracking frames (dilated +1),, not near-edge frames.
* isMoving\_Fish{j}: frames in which Fish *j* (*j* = 0, 1, 2, …, Nfish-1) has speed above threshold, i.e. is moving. Each is a list of dictionaries (see **behavior dictionaries**, below Note that this only excludes bad Tracking frames (dilated +1), not near-edge frames.

## Keys containing behavior information

* approaching\_Fish{0,1}: Array of frames in which Fish 0 (resp. 1) is approaching Fish 1 (resp. 0). From get\_approach\_flee\_frames(). See **behavior dictionaries**, below.
* AngleXCorr\_mean, AngleXCorr\_std, AngleXCorr\_skew : mean, standard deviation, and skew of the xcorr\_array array. Only the mean is saved in the output CSV file.
* closest\_distance\_mm: the inter-fish distance calculated as the closest distance between any inter-fish positions in each frame; mm; array of length Nframes. From get\_interfish\_distance() in toolkit.py.
* closest\_distance\_mm\_mean: mean closest inter-fish distance, ignoring bad-tracking frames.
* contact\_any: frames in which there is any contact behavior; a list of dictionaries (see **behavior dictionaries**, below).
* contact\_head\_body: frames in which there is head-body contact behavior, a subset of any contact; a list of dictionaries (see **behavior dictionaries**, below).
* contact\_larger\_fish\_head: frames in which there is head-body contact behavior and in which the head of the larger fish (only) is making contact with the other fish, a subset of head-body contact; a list of dictionaries (see **behavior dictionaries**, below).
* contact\_smaller\_fish\_head: frames in which there is head-body contact behavior and in which the head of the smaller fish (only) is making contact with the other fish, a subset of head-body contact; a list of dictionaries (see **behavior dictionaries**, below).
* contact\_inferred: frames corresponding to inferred contact, in which tracking is bad (zeros values) but inter-fish head-to-head distance was decreasing over some number of preceding frames and was below-threshold immediately before the bad tracking. The returned frame is the one immediately before the bad tracking. Make this a list of dictionaries, though this is technically unnecessary because by construction it can’t extend > 1 frame.
* fleeing\_Fish{0,1}: Array of frames in which Fish 0 (resp. 1) is fleeing from Fish 1 (resp. 0). From get\_approach\_flee\_frames(). See **behavior dictionaries**, below.
* head\_head\_distance\_mm: the inter-fish distance calculated as distance between head positions in each frame; mm; array of length Nframes. From get\_interfish\_distance() in toolkit.py.
* head\_head\_distance\_mm\_mean: mean inter-fish head-to-head distance (px), ignoring bad-tracking frames.
* 90deg\_noneSee: frames in which there is perpendicular orientation behavior in which the orientation is such that neither fish sees the other; a list of dictionaries (see **behavior dictionaries**, below).
* 90deg\_oneSees: frames in which there is perpendicular orientation behavior in which the orientation is such that one fish sees the other; a list of dictionaries (see **behavior dictionaries**, below).
* 90deg\_bothSee: frames in which there is perpendicular orientation behavior in which the orientation is such that both fish see the other; a list of dictionaries (see **behavior dictionaries**, below).
* 90deg\_larger\_fish\_sees: frames in which there is perpendicular orientation in which the larger fish (only) “sees” the smaller fish. A subset of 90deg\_oneSees. A list of dictionaries (see **behavior dictionaries**, below).
* 90deg\_smaller\_fish\_sees: frames in which there is perpendicular orientation in which the smaller fish (only) “sees” the larger fish. A subset of 90deg\_oneSees. A list of dictionaries (see **behavior dictionaries**, below).
* tail\_rubbing: frames in which there is tail-rubbing behavior, a subset of any contact; a list of dictionaries (see **behavior dictionaries**, below).
* xcorr\_array: Heading angle cross-correlation of the two fish over a sliding window for all frames. Value at a given frame is the normalized cross-correlation for the window frame ending at that frame. Nframes x 1 array.

## Deleted

* circling : frames in which there is circling behavior; a list of dictionaries (see **behavior dictionaries**, below).

## Behavior dictionaries

For each behavior, make a sub-dictionary of datasets[j] that contain frames without “bad” frames, total durations, etc., as illustrated here for the tail-rubbing behavior:

datasets[j]["tail\_rubbing"]["behavior\_name"]: Name of the behavior (e.g. “tail\_rubbing”). The name of the behavior is the same as the key name, but this could be changed if desired.

datasets[j]["tail\_rubbing"]["raw\_frames"]: 1D array of frame numbers in which this behavior was detected, or the first frame if it requires a frame window.

datasets[j]["tail\_rubbing"]["edit\_frames"]: 1D array of frame numbers, the resulting of removing from “raw\_frames” any frames found in input lists of frames to remove using remove\_frames(), for example in which fish were close to the dish edge, or bad tracking frames.

datasets[j]["tail\_rubbing"]["combine\_frames"]: 2D array in which the “edit\_frames” frames are replaced with the first frame of any run of sequential frame numbers (row 1 of the array) and the duration of the runs (row 2). Note that the length along axis==1 (columns) gives the number of runs, and therefore the number of events.

datasets[j]["tail\_rubbing"]["N\_events"]: scalar. The number of events or runs (simply the length of the second row of combine\_frames).

datasets[j]["tail\_rubbing"]["total\_duration"]: scalar, sum of the durations in row 2 of “combine\_frames”. This is therefore the number of total frames in which this behavior was detected, after deleting “bad” frames (dish edge, or bad tracking).

datasets[j]["tail\_rubbing"]["relative\_duration"]: scalar, relative duration, i.e. total\_duration / Nframes. Again, deleting “bad” frames (dish edge, or bad tracking).

The rest of this module NOT YET MODIFIED BELOW HERE

Defines position, angle variables

Identifies circling events. Calls get\_circling\_wf → output circling\_wfs

Identifies 90-degree (perpendicular) events, including “one,” “both,” “none” subsets. Calls get\_90\_deg\_wf → output orientation\_dict and subsets “one,” “both,” “none” [should rename]

Identifies contact events, including “any” and “head-body” subsets. Calls get\_contact\_wf → output contact\_wf and subsets “any,” “head-body” [should rename “any”!]

Identifies tail-rubbing events. Calls get\_tail\_rubbing\_wf → output tail\_rubbing\_wf

Various outputs: text file, diagram, excel file:

Calls get\_txt\_file

Calls get\_diagram

Calls get\_excel\_file

**load\_data() in toolkit.py**

Returns the requested data array for fish 1, fish 2 from the dataset

[rows are hard-coded; comment more, or change this.]

**get\_circling\_wf() in circling.py**

“Returns an array of window frames for circling behavior. Each window frame represents the ENDING window frame for circling within some range”

[Terminology is odd. Should be clearer what frames, etc., we’re returning – e.g. ending frame of a window.]

Goes through all frames

Fits a circle to fish 1 and 2 head positions in a given frame window. Calls TaubinSVD → output taubin\_output, which is (xc, yc, r) for this window.

Determine RMSE for head positions to best-fit circle. Calls get\_distances → output distances\_temp to get distances between head positions and best-fit circle. Calls get\_rmse → output rmse, [Comment “Fit the distances…” makes no sense; fix]

Evaluate the angle between fish headings, and the antiparallel criterion. Calls check\_antiparallel\_criterion → output [not saved]

If rmse is below threshold rmse\_thresh and antiparallel criterion is met, append this window frame to circling\_wf .

Returns window frames appended to circling\_wf using combine\_events ,

**get\_90\_deg\_wf() in ninety\_deg.py**

Gets positions and angles for a given frame window

Calls get\_antiparallel\_angle to get angle (or cosine of angle) between fish headings; takes absolute value; compares this to the cos(theta) threshold.

Calls get\_fish\_vectors to get average heading angles over frame window.

Calls get\_connecting\_vector to get the normalized vector between fish head positions, connecting fish 1 and 2.

Calculate signs of cross product of fish vectors, and then call get\_orientation\_type in get\_90\_deg\_wf.py. to determine orientation type.

Evaluate criteria for 90 degree orientation: cos(theta) < threshold [rename variables, as noted elsewhere] and head-to-head distance < threshold. If so, append orientation type and frames.

**get\_contact\_wf() in contact.py**

“Returns a dictionary of window frames for different types of contact between two fish...”

Goes through all frames.

Checks if min distance between head of fish i and any point on fish j (j = 1, 2) is less than threshold; if so count as both “head-body” and “any” contact

Checks if min distance between any points on fish i and any point on fish j (j = 1, 2) are less than threshold; if so, count as “any” contact

If criteria are met, append frames.

**get\_tail\_rubbing\_wf() in tail\_rubbing.py**

Identifies tail-rubbing events.

Gets the 4 posterior-most body position markers for each fish. Also head positions, angles.

Calls get\_min\_tail\_distances “to get the minimum tail distances between two fish for two different body markers”

Evaluate the angle between fish headings, and the antiparallel criterion. Calls check\_antiparallel\_criterion → output [not saved]

If min tail distances and antiparallel criteria are met, append frames.

**Other functions in get\_90\_deg\_wf.py**

get\_fish\_vectors in get\_90\_deg\_wf.py. Returns a vector in the form (cos(theta), sin(theta)) for fish1 and fish2; average of heading angles over window size. [Simplify description to state this.]

get\_connecting\_vector in get\_90\_deg\_wf.py. Returns the normalized vector between fish head positions, connecting fish 1 and 2. [Check that it’s head positions.]

get\_orientation\_type in get\_90\_deg\_wf.py. Returns the orientation type of two fish given the sign of their respective (a, b, c) vectors.

# Write – behavior identification

## Contact functions

get\_contact\_frames()

get\_inferred\_contact\_frames()

## Perpendicular degree orientation functions

get\_90\_deg\_frames()

get\_orientation\_type()

## Tail-rubbing functions

get\_tail\_rubbing\_frames()

**Other functions in tail\_rubbing.py**

get\_min\_tail\_distances . Returns the minimum tail distances between two fish for two different body markers.

# toolkit.py

## Data loading and handling functions

get\_basePath:

Ask the user whether to use basePathDefault as the basePath; if not, get a path string either as text input or from a dialog box.

get\_valid\_file(fileTypeString = 'Config File'):

Check if the file+path exists; if not, dialog box.

load\_expt\_config(config\_path, config\_file):

Loads the experimental configuration file. Asks user for the experiment being examined

get\_CSV\_folder\_and\_filenames():

Asks user for the folder path containing CSV files; returns this and a list of all CSV files whose names start with "results" .

load\_all\_position\_data():

For all CSV files in the list, call load\_data() to load all position data, and determine general parameters such as fps and scale

load\_data(CSVfileName, N\_columns):

Loads position data from a CSV file and returns a single array containing both fish's information (position, angle, body markers, etc.)

Checks that frame numbers are the same for each array, and that there are no gaps in frames, and that the first frame is 1

get\_dataset\_name(CSVfileName):

Extract the "dataset name" from the CSV filename. Delete"results\_SocPref\_", "\_ALL.csv"; keep everything else. E.g. file name "results\_SocPref\_3c\_2wpf\_k2\_ALL.csv" gives dataset\_name = "3c\_2wpf\_k2""

make\_frames\_dictionary(frames, frames\_to\_remove):

Make a dictionary of raw (original) frames, frames with "bad" frames removed, and combined (adjacent) frames + durations

combine\_events(events):

Given an array of frame numbers, return an arrays of frame numbers with adjacent frames combined and duration numbers corresponding to the duration of adjacent frames.

## Coordinate determination functions

get\_ArenaCenter(dataset\_name, arenaCentersFilename, offsetPositionsFilename):

Extract the x,y positions of the Arena centers from the arenaCentersFilename CSV -- previously tabulated. Image offsets also previously tabulated, first and second columns of offsetPositionsFilename

get\_edge\_frames(dataset, params, arena\_radius\_mm, xcol=3, ycol=4):

identify frames in which the head position of one or more fish is close to the dish edge (within threshold)

get\_imageScale(dataset\_name, imageScaleFilename):

Extract the image scale (um/px) from imageScaleFilename CSV -- previously tabulated

def estimate\_arena\_center(alldata, xcol=3, ycol=4):

Estimate the arena center position as the midpoint of the x-y range.

**Note:** This function is not currently used (July 4, 2023)

## Basic properties

get\_interfish\_distance(all\_data, CSVcolumns):

Get the inter-fish distance, calculated both as the distance between head positions and as the closest distance, in each frame

get\_fish\_lengths(all\_data, CSVcolumns):

Get the length of each fish in each frame (sum of all segments)

## Bad Tracking identification and notation functions

def get\_bad\_headTrack\_frames(dataset, params, xcol=3, ycol=4, tol=0.001):

identify frames in which the head position of one or more fish is zero, indicating bad tracking

**Note:** This function is not currently used (July 4, 2023)

get\_bad\_bodyTrack\_frames()

identify frames in which tracking failed, as indicated by either of:

(i) any body position of one or more fish is zero, or

(ii) the distance between positions 1 and 2 (head-body) is more than

3 times the mean distance between positions j and j+1 for j = 2 to 9

remove\_frames(frames, frames\_to\_remove):

Remove from frames values that appear in frames\_to\_remove, and optionally dilate the set of frames to remove (e.g. for speed assessments).

## Visualization and diagnostic functions

plotAllPositions(dataset, CSVcolumns, arena\_radius\_mm, arena\_edge\_mm = 0):

Plot head x and y positions for each fish, in all frames also dish center and edge

visualize\_fish(body\_x, body\_y, frameArray, startFrame, endFrame, dataset\_name):

Plot fish body positions (position 1 == head) over some range of frames body{x, y} are Nframes x 10 x Nfish=2 arrays of x and y positions

## Data export

Note that the summary CSV of all datasets is created in behaviors\_main.py

write\_output\_files(params, dataPath, datasets):

Calls other functions to write the output files (several) for all datasets

Hard-codes a superset of all keys (behaviors) to write, and then removes any keys that are not in the first dataset, for example two-fish behaviors if that dataset was for single fish data

write\_behavior\_txt\_file(dataset, key\_list):

Creates a txt file of the relevant window frames and event durations for a set of social behaviors in a given single dataset

Output text file name: dataset\_name + .txt

mark\_behavior\_frames\_Excel(markFrames\_workbook, dataset, key\_list):

Create and fill in sheet in Excel marking all frames with behaviors found in this dataset

## Correlation functions

calculate\_autocorr()

calculate\_block\_autocorr()

calculate\_crosscorr()

calculate\_block\_crosscorr()

calculate\_value\_corr\_oneSet()

calculate\_value\_corr\_all()

## Probability distributions and histograms

plot\_probability\_distr()

make\_2D\_histogram(): Create a 2D histogram plot of the values from two keys in the given datasets. Can specify ranges, dilation of “bad frames” – always avoid bad frames. Uses combine\_all\_values\_constrained() to combine across all datasets the values of some characteristics, removing (dilated) bad frames, etc. See August 2024 notes, including August 27, 2024 notes – can use particular fish for values, and for constraints.

## Combining and plotting data

combine\_all\_values\_constrained(): Combine all values of one characteristic, like speed, across all datasets. Optional: can combine values of one characteristic subject to a mask on another characteristic, for example all speed values when inter-fish distance is close. Ignore, in each dataset, "bad tracking" frames. For example: get all speed values for frames in which inter-fish-distance is below 5 mm.

*August 27, 2024:* Allow for constraints properties to be arrays with more than one fish, e.g. for speed, either taking values for one of the fish, or an average, or max or min.

Can then use combined values in, for example, plot\_probability\_distr().

**Aligning and merging bout trajectories**

August 21-22, 2024

average\_bout\_trajectory\_oneSet(): Tabulates speed information from dataset["speed\_array\_mm\_s"] around each onset of a bout ("isMoving" == True) in the time interval specified by t\_range\_s. Averages these for all bouts for a given fish; considers each fish in dataset. Optional: only consider bouts for which the value of constraintKey at the start frame (first isMoving frame) is between constraintRange[0] and constraintRange[1].

average\_bout\_trajectory\_allSets(): For all datasets, call average\_bout\_trajectory\_allSets() to tabulate speed information around each onset of a bout ("isMoving" == True) in the time interval specified by t\_range\_s. Averages these for all bouts, each fish and each dataset. Optional: only consider bouts for which the value of constraintKey at the start frame (first isMoving frame) is between constraintRange[0] and constraintRange[1]. See also Aug. 27, 2024 notes – can use particular fish for constraints.

plot\_function\_allSets()

behaviorFrameCount\_one()

behaviorFrameCount\_all()

# behavior\_identification.py

## All zebrafish pair behavior identification functions

# behavior\_identification\_single.py

## All behavior identification or characterization functions that apply to single fish

get\_fish\_lengths()

get\_fish\_speeds()

getTailAngle()

getTailCurvature()

get\_Cbend\_frames()

get\_Jbend\_frames()

combine\_all\_values():  Loop through each dataset, get values of some numerical property, and collect all these in a list of numpy arrays. Ignore, in each dataset, "bad tracking" frames. If "dilate\_plus1" is True, dilate the bad frames +1; do this for speed values, since bad tracking affects adjacent frames! List contains one numpy array per dataset (possibly with multiple columns corresponding to fish). Output can be used, for example, for making a histogram of speeds or inter-fish distance.

plot\_probability\_distr(): Make a histogram of values from list of values, for example all speeds as generated by combine\_all\_values():.

# Abandoned functions

Move to an “abandoned” folder

get\_reorientation\_wfs() in matching.py: reorientation “matching” behavior

get\_mirroring\_wfs(): in mirroring.py: "mirroring" behavior.

get\_circling\_frames(): “circling” behavior – lots of failed attempts