Behavior Analysis Pipeline v2

**Raghuveer Parthasarathy**

Last modified: June 13, 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.

# 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.

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):**

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\Misc\Zebrafish behavior\Code\_current'

convert\_fishTrackingData\_MATtoCSV(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.)

**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 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 .

# Experiment Configuration File

Each experiment must have an entry in the experiment configuration file, written in YAML. This file is “all\_expt\_configs.yaml” …\Zebrafish behavior\CSV files and outputs\ ; the path and file name are hard-coded in behaviors\_main.py.

Each experiment entry must have the following (note that some are optional):

**EXPERIMENT NAME:**

**dataPath:** # Path with CSV files; leave empty for user-input

**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) path to CSV with image scale for each dataset, if loading from file (i.e. if imageScale is None)

**imageScaleFilename:** # (optional) file name of CSV with image scale for each dataset, if loading from file (i.e. if imageScale is None)

**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:** C:/Users/Raghu/Documents/Experiments and Projects/Misc/Zebrafish behavior

**arenaCentersFilename:** ArenaCenters\_SocPref\_3456.csv

**arenaCentersLocation:** # leave empty to estimate from well offset positions

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

**subExpts:** # strings of subFolder names of sub-experiments 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:**

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:**

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 behaviors found in each frame.
* A CSV file for each CSV dataset that gives the basic measurements for each frame: head\_head\_distance, closest\_distance, speeds, distance to edge, and flags (0 or 1) for any fish being near the edge and any bad tracking. Name: dataset + \_basicMeasurements
* A file with summary statistics for all the datasets, indicating the number and duration of each of the behaviors. (Probably named “behavior\_count.csv” , specified in the analysis parameter configuration file) This file is the most important, and you may wish to rename it in the configuration file or afterwards.

## Run relative\_duration.py

Reads the summary behavior analysis CSV (probably named “behavior\_count.csv) and returns the relative durations for each behavior for each dataset (total time relative to total experiment duration), as well as the mean and std. dev. across datasets. (Last rows.)

Function calc\_relative\_durations\_csv() in relative\_duration.py

**Inputs:**

Running relative\_duration.py will ask for the following inputs

Directory name in which csv files are located

input\_file : The name of the .csv file in which behavior counts and durations for each dataset are recorded, created by the main behavior program. Likely name 'behavior\_count.csv', but may be useful to rename (manually).

output\_file : filename for the output CSV file. Again, each row = each dataset. Columns = each behavior. Last rows are the summary mean and standard deviation. Suggested / default name: 'behavior\_relDurations.csv'

fps : frame rate of experiments (frames/sec), for proper normalization of durations. Default 25.0

**Output:**

Writes a CSV file.

**To use:**

Run relative\_duration.py and follow the prompts for filenames, etc.

# Comparison of Experiments

Comparison of two experimental conditions, analyzing the single-experiment analysis outputs.

For example: Makes plots that compare the relative durations of behaviors in two different datasets, such as *2 week old light* and *2 week old dark*.

**To use:**

Manually modify main() in compare\_experiment\_behaviors.py to specify the path and file names corresponding to each experiment name

Then run compare\_experiment\_behaviors.py

Functions

Function plot\_comparison() in compare\_experiment\_behaviors.py

See Behavior Code Revisions November 2023 for notes on making the function.

# 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 reading functions, basic analysis functions, and behavior extraction functions for all CSV files in a set*

Load experiment configuration file; call load\_expt\_config()

Get CSV column info from configuration file

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()

For each dataset:

* Get general properties such as image scale and arena center and loads all position data into each dataset’s dictionary.
* 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: fish lengths from get\_fish\_lengths(); 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.

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==2) 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
* 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 head positions are bad (zero). From get\_bad\_headTrack\_frames().
* bad\_bodyTrack\_frames: array of frames in which one or more 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\_Fish0, Cbend\_Fish1, Cbend\_any: frames in which Fish 0, Fish 1, or either of the two fish shows a sharp bend. Each is a list of dictionaries (see **behavior dictionaries**, below).
* d\_to\_edge\_mm : Nframes x Nfish=2 array of distance of each fish to dish edge, mm
* edge\_frames : frames in which either 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 : length of each of the two fish in each frame (sum of all segments); Nframes x 2 array; from get\_fish\_lengths()
* fish\_length\_mean: mean fish length, ignoring bad-tracking frames; averaged over both fish.
* fish\_length\_Delta\_mean and fish\_length\_Delta\_std: mean over frames of absolute difference in fish length, 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 of the two fish, frame-to-frame, mm/second using scale and fps information; Nframes x 2 array; from get\_fish\_speeds()

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: 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\_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: 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\_mean: mean inter-fish head-to-head distance (px), ignoring bad-tracking frames.
* perpendicular\_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).
* perpendicular\_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).
* perpendicular\_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).
* perpendicular\_larger\_fish\_sees: frames in which there is perpendicular orientation in which the larger fish (only) “sees” the smaller fish. A subset of perpendicular\_oneSees. A list of dictionaries (see **behavior dictionaries**, below).
* perpendicular\_smaller\_fish\_sees: frames in which there is perpendicular orientation in which the smaller fish (only) “sees” the larger fish. A subset of perpendicular\_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 (now deleted) circling behavior:

datasets[j]["circling"]["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]["circling"]["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(), e.g. in which fish were close to the dish edge, or bad tracking frames.

datasets[j]["circling"]["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).

datasets[j]["circling"]["total\_duration"]: scalar, sum of durations.

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.

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

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

## 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()

# toolkit.py

## Data loading and handling functions

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\_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 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

## 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\_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

# 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