Project Overview

This repository contains code for the filtering of data surrounding the motion of Colloids, given that the data has been extracted via ImageJ and Speckle TrackerJ. These files aim to provide accurate filtering, as well as reasonable automation.

This document outlines each of the included files, how to use them, and what they do. All code herein uses Python type hinting, which may be unfamiliar. If code modification is needed and this is unfamiliar, refer here (or contact me at jedehmel@mavs.coloradomesa.edu).

These filters should be stable and Windows-compatible, but they have been tested on a machine running Arch Linux. If a bug is found, report it to jedehmel@mavs.coloradomesa.edu.

Jordan Dehmel, 2023 - present, jdehmel@outlook.com or jedehmel@mavs.coloradomesa.edu

Workflow

1) Receive raw `*.avi` files from FIU via Globus
2) Re-encode and downscale these using `python3 reformat_all_avis.py /path/to/avis`
3) Re-organize output files from previous step
4) Use speckle tracker to analyse organized files, output speckle files
    1) Open `Fiji` or `ImageJ`
    2) Open a file explorer
    3) Drag the downsized file into `Fiji` / `ImageJ`. An `AVI Reader` window should pop up
    4) Make sure that `Convert to Grayscale` is checked, and `Use Virtual Stack` is **not** checked
    5) Click `OK` to exit the `AVI Reader` window. A preview window should open
    6) Click `Plugins` -> `Speckle TrackerJ` to open the speckle tracker window
    7) Use `+` and `-` to adjust the zoom
    8) Click `models` -> `Adjust Parameters`
        - The two relavent parameters are `Intensity factor` and `Search Size`
        - Lower on either of these makes this go faster, but can cause artifacts
        - For `512x512` pixel videos, `4.0` search size is good enough
        - If you leave it on `12.0` pixels, it will take forever
        - If you notice major tracking issues, adjust parameters
    9) Click `Accept` to go back to the tracking window
    10) Click `locate` -> `Locate Speckles` to open the speckle finding window
    11) Adjust `threshold` until all good particles are selected

12) Optionally, adjust `minimum distance` to eliminate clusters
13) Click `Accept` to go back to the tracking window
14) Click `track` -> `Auto-track All`
    - If this takes longer than 30 seconds, adjust parameters
    - For `512x512` pixel videos on a medium-grade computer, this takes ~1 second is parameters are correct
15) Scrub around the video using the left and right arrow keys
16) Switch between tracking models using the up and down arrow keys
17) Select a track by clicking on it
    - After selection, you can delete the track w/ `d`
    - You can auto-track the selected speckle track w/ `a`
    - You can manually select the next position w/ `t`
18) Make sure no conjoined colloids remain in the final data
19) Make sure particles are not too jittery
20) Make sure particles are in frame for long enough to get good readings
21) When done, click `File` -> `Save Speckles` and save wherever you the downsized video file is
22) Close the speckle tracking window, then close the preview window
23) Repeat on next video

5) Change output speckle files to track files using Python scripts
6) Use track file resources to extract velocities and whatnot
7) Use local graphing resources to visualize data

`simple.py`

This is a simple file doing only the bare minimum. This has no error handling, so over-filtering is likely in some circumstances. Additionally, you must manually enter all filepaths by hand. If these are deal-breakers, you should instead use `filterer.py`.

Options

You can change the following items in this file.

- `to_capture` This is the variable to extract. This should be set to `MEAN_STRAIGHT_LINE_SPEED`, but can be changed if need be.
- `filepaths` This is a list containing the filepaths which will be operated upon. For `simple.py`, you must manually enter each filepath.
- `frequencies` This is a list of the frequencies in Hertz which were applied. The first entry should correspond to the first filepath in `filepaths`, and so on.

If these variables are properly set, the protocol listed at the head of `simple.py` will be executed. However, overfiltering is likely to be an issue, and this program does not have the capability to recover from that.

`bulk_rename.py`

This script takes one command-line argument: The folder to rename. This is to be used after the automated reorganization of output files. It strips artifacts of name disambiguation, like `.avi.avi`.

`reformat_all_avis.py`

This script takes two command-line arguments: The first is the directory inside which to save a copy of the reformatted `avi` files, and the second is the directory to recursively search for said files. It will iterate through all the subfolders of this second argument, re-encoding and re-sizing all `*.avi` files that it finds. It will also save a copy of this output file in the output directory specified in the first argument. This script will save the reformatted videos in their original location and in the output folder. **It will *not* overwrite the original files.** When saved in the output folder, the names will be disambiguated to include their fully-qualified path.

`filterer.py`

This is the more complicated version of `simple.py`. This program has many more options, and many more advanced features. The options of this program are listed below.

Options

- `folder` This is a string containing the **folder to be filtered**. All files in this folder will be filtered and saved according to the Regular Expressions listed later in the program. These expressions should not need to be modified, but if you are having trouble with name matching they may need to be.
- `do_std_filter_flags` This is a list of boolean values. If the first value in this list is `True`, then the data will by internally filtered by excluding any outliers on the first value of `col_names`- in this case, `'TRACK_DISPLACEMENT'`. An outlier will be deemed **any value which is more than two standard deviations below the mean** for a given statistic. This holds true for the remaining values in the list- The `n`th value applies an internal filter to the `n`th item in `col_names`. A copy of `col_names` can be found for reference just above this option in `filterer.py`. If this value is `None`, none of these internal-standard-deviation filters will be applied.
- `do_iqr_filter_flags` This is identical to `do_std_filter_flags`, but designates an outlier slightly differently. With these flags, an outlier is any value which is more than 1.5 inner-quartile-ranges below the mean. This is marginally better at identifying outliers. As above, if this is `None`, no internal-IQR filters are applied.

- `do_quality_percentile_filter` If this is set to `True`, any particles below the `quality_percentile_filter`-th percentile in track quality (as determined by ImageJ) will be dropped.
- `quality_percentile_filter` If `do_quality_percentile_filter` is `True`, this is the minimal percentile that tracks must possess in order to remain.
- `conversion` This is the coefficient which, when applied, turns a measurement from pixels per frame to micrometer per second.
- `do_speed_thresh` This is a boolean value denoting whether or not to do the Brownian mean-straight-line filtering. If `True`, any value below `brownian_mean` + (`brownian_standard_deviation` · `brownian_multiplier`) will be filtered out.
- `brownian_multiplier` This is the number of standard deviations above the Brownian mean straight line speed a track must be in order to survive the filter activated by `do_speed_thresh`.
- `do_displacement_thresh` If `True`, filters out any tracks below the Brownian mean displacement. This should probably be left `False`, since straight line speed is a better measure of mobility.
- `do_linearity_thresh` If `True`, filters out any tracks below the Brownian mean linearity. This should probably be left `False`, since straight line speed is a better measure of mobility.
- `do_duration_thresh` If `True`, filters any tracks with durations (in frames) less than `duration_threshold`. **This should be left `True`, since shorter tracks introduce more error.**
- `duration_threshold` If `do_duration_thresh` is `True`, this is the minimal number of frames a track must have in order to survive filtering.
- `secondary_save_path` If not `None`, this is a string representing a secondary save location. All files produced by this program will be saved in this location.
- `silent` If `False`, this option causes the program to produce much more detailed output. This is useful to turn off for debugging purposes, but otherwise should be left `True`.
- `do_speed_thresh_fallback` This option controls whether or not to use the `brownian_speed_threshold_fallback` as the Brownian straight line speed for later filtering if no Brownian file can be detected. This should not be necessary (so long as the Regular Expressions are working properly), and should be left `False`.
- `brownian_speed_threshold_fallback` If `do_speed_thresh_fallback` is `True`, this is the Brownian straight line speed which will be used if no Brownian file can be found. This allows the program to keep filtering if no control value is found, but must be fine-tuned to your sample.
- `do_filter_scatter_plots` If this option is `True`, scatter plots detailing which particles were kept and which particles were dropped will be produced and saved. This should be left on.
- `do_extra_filter_scatter_plots` If this option is `True`, extra plots will be produced detailing the filtering process. These plots are less useful.

- `save_filtering_data` If this option is `True`, histograms representing the filtered data will be saved.

If there are issues with the automatic detection of files, it is likely that the naming scheme used does not match the existing Regular Expressions. If it is only a few files, you can change the naming scheme. However, if it is many files, the expressions should be modified, and you should contact jedehmel@mavs.coloradomesa.edu or someone else who knows RegEx.

Process

For a single file:

Initialization:

- Load file from .csv
- Drop Items which are not of use to us

Filtering:

- Do duration threshold
- Do Brownian mean straight line speed threshold
- Do Brownian displacement threshold
- Do Brownian linearity threshold
- Do quality threshold
- Do Standard Deviation filtering
- Do IQR filtering

Output:

- Calculate mean and standard deviation from remaining
- Output graphs for this file
- Yield data

Warning: If, after a given filter is applied, no tracks remain, that filter will be discarded and the data reverted to before that filter.

For a group of files:

Initialization:

- Load desired folder
- Find files within folder which match the frequency Regular Expressions

Iteration:

- Load Brownian file (with ONLY internal filtering)
- Load non-Brownian files (with Brownian and internal filtering)

Reconstruction:

- Save graphs

- Save .csv file(s)

`name_fixer.py`

This file contains only utilities for automatic name detection via Regular Expressions. This is what allows the other programs to work. You should not need to do anything with this file.

`reverser.py`

This file contains only utilities for plotting straight line speeds with respect to their crossover frequencies. You should not need to do anything with this file.

`comparisons.py`

This file creates height-wise comparison graphs of all data in the current working directory, assuming that it can be broken down by height. This is true for Clark's initial glycerol data, but not KCL or the waveform experiments.

The `legacy` folder

This folder contains old code which is no longer relevant. It can be ignored.

The `scripts` folder

This folder contains Linux scripts for multi-folder automation. These cannot be run on Windows (except through WSL), and must be specially written for a given dataset. These can most likely be ignored.

Resources

- TrackMate Manual
- Regular Expressions in Python
- Typing in Python
- Python Style Guide