Euglena Matlab Module

Zahid Hossain (zhossain@stanford.edu), Honesty Kim (honestyk@stanford.edu)

We compiled a Matlab module that automatically tracks Euglena swimming trajectories. The tracing is based on computer vision and is suitable for statistical measures, (it may not be very accurate for an individual track – but more than suitable for our tasks at hand).

1. Quick Overview: (Details below if you need to).

i. Download the Matlab module for your OS from https://goo.gl/Ob99Mw. (download the entire folder - it is ~130MB!) and unpack it
   a. You may need to do some extra configuration steps depending on your operating system – see section 1 below. (If demo01 below fails you need to make these configurations.)

ii. Change your Matlab directory to where the module is downloaded.

iii. There are three demos in the Matlab module package that you have downloaded. Each can be run individually but its better to run them in the following order from within the package folder. **These demo's use data from us – read also the comments inside.**
   a. demo01_basic_load.m: This shows the very basics of how to load data, get basic information, generate a video and eventually save the video on disk. (The first time you run this it might take 1-5 minutes – next time it is fast.)
   b. demo02_trackid.m: This demo shows how you can investigate a single Euglena track of your choice, extract geometry data and compute point velocities of that track. It also shows how to visualize a track via a video.
   c. demo03_multitrack.m: This demo shows how to select tracks between frames and deal with multiple tracks at the same time.

iv. Now work with your own data:
   a. Download your own experimental data, unzip it, which produces a folder (let's call it 'images' from now on) with two files, movie.mp4 and lightdata.json
   b. Now modify the data path in demo01_basic_load.m to 'images' and run (first run may again take 1 to 5 minutes)
   c. Now you should see a movie file called "tracks_thresholded_10.avi" and some other files (which you can ignore but should not delete). Watch this movie - you can see all tracks and stimuli, furthermore scalebar, frame number and timer. The number on each track is its "TrackID." (Note that many IDs are missing – as it only shows you cells that were traced successfully over at least 10 frames),
   d. Now use and modify starter codes problem_2d.m and problem_2e to do these problems.

v. Repeat steps in iv if you want to analyze more experiments

2. Potential additional configuration steps (Only works for 64-bit Platforms):

Please download the folder that is appropriate for your current OS from https://goo.gl/Ob99Mw. **Note that on all platforms, video generation on Matlab R2012x might be very slow! Besides, on R2012x, some text annotation on the videos will be visually less distinguishable, so consider running a later version if possible.** For any difficulty please contact me at zhossain@stanford.edu.

2.1. Windows

OS version tested: Windows 10, Windows 8.x
Matlab version tested: R2012a to R2015b.

1.1. OSX (Mac)

OS version tested on: Yosemite (10.10), El-Capitan (10.11), but should also work on 10.9 (Maverick).
Matlab version tested: R2013b to R2015b.

1.2. Linux

OS version tested: Ubuntu 14.XX to 15.XX with GCC 4.7 runtime
Matlab version tested: R2013b to R2014b
NOTE: On newer Ubuntus, Matlab's VideoReader module spits out an error regarding libgstreamer. Unfortunately this is Matlab's own problem and you can try the following steps to fix it.

- Install gstreamer0.10: Run the following command in the **system's console:**
  - sudo add-apt-repository ppa:mc3man/trusty-media && sudo apt-get update && sudo apt-get install ffmpeg gstreamer0.10-ffmpeg

  Restart Matlab!

- If the problem persists, you have to do the following from within **Matlab's console (including the '!').** The exact version of libstdc++ may be different, use the appropriate version number as found in the (matlabroot)/sys/os/glnxa64 folder.
  - cd (matlabroot)/sys/os/glnxa64/
  - !mv libstdc++.so.6 to backuplibstdc++.so.6
  - !mv libstdc++.so.6.0.10 to backuplibstdc++.so.6.0.10

  Restart Matlab!

3. The Matlab Module Details

You can perhaps skip this and rather read the demo files (see above) which has lots of comments that explains everything.

Adding path in Matlab

This only matters if you want to run the module from any arbitrary directory. I will assume that the module folder is at **"C:\Users\Zahid Hossain\Documents\Windows".** Now we will add a path in Matlab by typing in the following in the matlab console.

```
addpath 'C:\Users\Zahid Hossain\Documents\Windows'
```

Note this is different from adding to the **Systems** PATH variable for Windows.

Loading Data

Let suppose you downloaded an experimental data from http://euglena.stanford.edu and uncompressed it at "C:\Users\Zahid Hossain\Downloads\images", i.e. the files inside this folder are **"lightdata.json"** and **"movie.mp4"**

Now in matlab, you can load this data by the following

```
exp = EuglenaTracks('C:\Users\Zahid Hossain\Downloads\images',10);
```

EuglenaTracks is the class module that encapsulates an experimental data and computes all the tracks for you. From here, we will only run different queries on the object "exp" to extract data. The first argument is the folder (can also be a relative path) that contains lightdata.json and movie.mp4, while the second argument is a threshold, which means all the track with atleast 10 points (or samples) are loaded while the anything smaller are discarded. Setting this threshold to 0 with load all the tracks.

The first time you run EuglenaTracks on a data folder it may take a while to compute all the tracks and a couple of videos that I will talk about shortly. But these results are cached and will not be computed again as long as you don't change the "threshold" parameter. During the first run it will produce two videos, **debug.avi** and **tracks_thresholded_<threshold>.avi inside the data folder**. The **debug.avi** video will have all the tracks (regardless of threshold) in it, while the other video will only have the tracks that have atleast "threshold" number of points. The numbers written on the each of the track is its **"TrackID."** Every track that is visible in the **tracks_thresholded_<threshold>.avi** video can be further investigated and therefore this video is the best place to start.

NOTE: Unfortunately, Windows version of the module doesn't show a progress meter when the Tracks are computed.

Selecting and Extracting Data from a Single Track

Say a track with ID 90 seems quite interesting in the **tracks_threshold_<threshold>.avi** video. We can extract only that track by the following.

```
track = exp.getTrackByID(90);
```

A single track is basically a collection of rectangular boxes that are oriented at an angle. We can get all the geometry information of this track by simply:

```
[x,y,width,height,angles,frames] = exp.extractTrackData( track );
```

The output x,y,width,height,angles and frames are x and y coordinates of the center of all the boxes, width and height of the boxes, angle of orientation and frame numbers where each boxes appear. Frames are the video frame and the numbering starts from 1 like anything else in Matlab. If there are N number of boxes (points) in this track, the the size of each of the output variables will be 1xN.

Visualizing Tracks

You can visualize this single track by making a video out of it:

```
M = exp.movieFromTracks( track );
```

Where M will is a giant matrix containing all the video frames. You can save this video on the disk by

```
exp.movieSave('test',M);
```

which will save a video named **test.avi** (the extension .avi is automatically added) in the current directory. Or, you can also view this video immediately within Matlab by

```
implay(M);
```

Among other self explanatory annotations on the video, there will 4 numbers on each side of the video. These numbers (0-100) shows the LED values at that point in time.

NOTE: The annotation on the video are unfortunately much less distinguishable on matlab version R2012x. Also, `exp.movieFromTracks( … )` also take a list of tracks (Matlab Cells) and generates a movie from all of them at once.

Track Selection

Euglena tracks can be selected by the function exp.findTracksBetweenFrames(<startFrame> , <endFrame>), for example,

```
selectedTracks = exp.findTracksBetweenFrames( 100 ,200 );
```

This will return a list of tracks (in Matlab cell format) that have appeared atleast once between frame number 100 and 200 inclusive. Meaning some track may have started earlier than frame 100 or some may have ended long after 200 but as long appeared between 100 and 200 they will be selected.

If you make a movie out of these tracks by M = exp.movieFromTracks(selectedTracks), you may notice that the video starts from a frame number before 100 and ends at a frame number higher than 200 because it draws the full length of all the tracks.

You can select frames between actual times by using the helper function exp.getFPS() which returns the value of frames per second. This can be used to convert times to frame numbers and vice-versa.

Clipping

Tracks can be clipped along frames, i.e. with the previous selectedTracks variable, if we do the following

```
selectedTracks_clipped = exp.clipTracksBetweenFrames(track,100,200);
```

This time, the output selectedTracks_clipped will have the same selected tracks excepted they will be clipped exactly at the frame boundaries of 100 and 200. You can verify this by making a video out of it, exp.movieFromTracks(selectedTracks_clipped).

Finally, you can also extract all geometry information from the clipped tracks, for example the following is extracting the geometry from the first clipped track

```
[x,y,width,height,angles,frames] = exp.extractTrackData( selectedTracks_clipped{1} );
```

This is one way of extracting all the Euglena geometries between two frames precisely and compute various statistics.

4. Helper Functions

There are a bunch of helper functions but here are some of the important ones:

- exp.getNumTracks() : returns the total number of tracks
- exp.getNumFrames() : return the total number of frames in the experiment video.
- exp.getTotalTime() : returns the total run time (in seconds) of the experiment.
- exp.getFPS() : returns the frames-per-second; can be used to convert frame number to real time and vice and vice-versa.
- exp.getUMPP(): returns the length scale in micro meter (um) per pixel.