Real-Time Tracking

By Carolyn Silverman

Abstract

Hydra vulgaris is a small freshwater cnidarian with an elementary nervous system, making it a useful model for receiving an optical readout of neural activity. This project's long-term goal is to use optogenetic tools to target individual neurons with light to determine causal influence on the activity of specific neurons in *Hydra*. In order to do so, I have built a system to track individual objects in real time. The program first receives video input from a Hamamatsu camera that is connected to a microscope. The MATLAB code then determines vital information about each frame, including the brightest pixels, so that blue laser light can be sent to those areas for channelrhodopsin-2 stimulation. These pixels will correspond to individually marked neurons in channelrhodopsin-2 transgenic *Hydra vulgaris.* I am also refining a code that is more effective for multiple neuron tracking. The code creates tracks in order to associate an object in a particular frame to the same object in the previous frames using a nearest-neighbor approach. Thus, multiple neurons can be tracked simultaneously and differentiated. Overall, the use of optogenetics and real-time analysis could lead to new information about the function of specific neurons in *Hydra vulgaris*.

Overview of the Tracking Programs

SingleObjectTracker.m
This program works to identify and track a single object in real time using a brightness threshold and a number of size and shape based filters. After the brightest object in the frame has been detected, the program determines its centroid, bounding box, and pixels. The output video displays the acquired input video along with the annotated object. The program repeats the process for every frame at a rate of 30 frames per second until it is instructed to terminate. This tracker also tracks multiple objects, but it does not have the capability to link the same object from one frame to another based on a trajectory. The labels that appear on the output stream are assigned using a basic for loop. Thus, they correspond only to the object's location in space in any given frame. This program is therefore recommended for tracking one object at a time.

NearestNeighborTracker.m
The nearest neighbor multiple object tracker uses the same basic framework as the previous program: it creates a binary image based on a brightness threshold, filters out noise, and determines important properties of the brightest objects (centroids, bounding boxes, and pixels). It is more effective for tracking multiple objects because it retains an array of centroids from the previous frame and compares them to the centroids in the current frame to assign a "blob ID". It also keeps an array of centroids, an array of pixels, and an array of bounding boxes for all of the objects in the current frame in order to match the centroid of a tracked object to its corresponding pixel list and bounding box. The program calls the nearestneighborlinker function, which matches source and target points based on Euclidean distance and returns an array of target indices. Each index corresponds to the location of blob k in the centroids array, where k is the

blob ID. It also returns an array of unassigned targets, which contains the indices of the objects that cannot be linked to objects in the previous frame. A new blob ID is assigned to each of these objects. Finally, the program retrieves the pixels and bounding boxes that are associated with the blobID based on the index. The objects are once again annotated on the output video stream.

Problems:
1. The program is not robust when objects enter or exit the frame.
2. It is not capable of dealing with gap closing: if an object disappears for a frame, the program does not correctly identify it when it returns to the frame. Instead, it assigns a new blob ID.
3. The nearestneighborlinker function ensures local optimization but not global optimization.

HungarianLinkTracker.m
The Hungarian algorithm is similar to the nearest neighbor algorithm but it ensures global optimization by minimizing the sum of the square distances between the source points (the centroids of the objects in the previous frame) and the target points (those in the current frame). It is less efficient than the nearest neighbor approach—$O(N^3)$ opposed to $O(N^2)$, where N is the number of source points. The rest of the program operates identically to NearestNeighborTracker.m, so it has the same the same difficulties dealing with gap closing and objects entering and exiting the field of view. It is worth using this tracker over the nearest neighbor tracker if only a few objects are being tracked.

Outlook

While the current programs can identify and track single and multiple objects with high accuracy under favorable conditions, many improvements can be made to make them more efficient and robust. The single object tracker is the most effective for its specific purpose and performs very well on the test objects, assuming proper lighting. The next step is to track a neuron in real-time to ensure it can deal with the additional noise. If not, more filters can be easily added to the current code.

There are many possible ways to improve the multiple object tracker. One promising method is the Kanade-Lucas-Tomasi (KLT) algorithm, which is a feature-based tracker that detects Harris corners and links motions vectors in successive frames to get a track for each Harris point.
The zip file includes a version of a KLT tracker, but this tracker will not work for the current project because it is unable to associate the correct pixel list with the tracked object. Perhaps the most effective use of the KLT algorithm would be to use it to track object contours. Unfortunately, this is a challenging task and there is no MATLAB implementation of it.

A simpler way to improve the multiple object tracker is to manipulate Jean-Yves Tinevez‘s simple tracker program to work in real-time in order to combine the nearest neighbor linker and the Hungarian linker and to deal with gap closing. As it currently stands, the program accepts as input a cell array of all the objects’ points (centroids in our case), with one cell found per frame. Once it has this information, it builds a track for each of the particles and outputs a new cell array with each cell corresponding to a different track. Each track contains an integer array corresponding to the index of the object in every frame. In order to use it for our purposes, it needs to create these tracks as it is acquiring the particles’ positions in real-time.

Resources Used

Nearest-neighbor linker by Jean-Yves Tinevez
http://www.mathworks.com/matlabcentral/fileexchange/33772-nearest-neighbor-linker
Hungarian based particle linker by Jean Yves Tinevez
http://www.mathworks.com/matlabcentral/fileexchange/33968-hungarian-based-particle-linking
How to Detect and Track Red Colored Object in live Video by Arindam Bose
http://in.mathworks.com/matlabcentral/fileexchange/40022-how-to-detect-and-track-red-colored-object-in-live-video

Resources for Future Use

Simple Tracker by Jean-Yves Tinevez
http://www.mathworks.com/matlabcentral/fileexchange/34040-simple-tracker

Kanade-Lucas-Tomasi (KLT) algorithm
MATLAB's Point Tracker:
http://www.mathworks.com/help/vision/ref/vision.pointtracker-class.html
UCF Lecture: https://www.youtube.com/watch?v=tzO245uWQxA
http://crcv.ucf.edu/courses/CAP5415/Fall2013/Lecture-10-KLT.pdf
Implementation for multiple face tracking:
http://www.mathworks.com/matlabcentral/fileexchange/47105-detect-and-track-multiple-faces

Additional Methods Worth Investigating
http://smal.ws/pdf/mie2012.pdf (page 6)