# Beemon: Bee ID

Eric Russo

April 2019

## 1 Introduction

Bee ID is an online interactive application that allows users to upload and select from sample videos of bees, allowing them to discern, and track them among several frames. Bee ID uses several well known open source algorithms included as part of the OpenCV.js API, namely Cascade Classifiers using Haar-like features, Lucas Kanade Iterative method for Optical Flow, and the Shi-Tomasi method for corner detection in images.

## 2 Classifier Training

Hundreds of positively identified bees, as well as negative samples of non-bees were manually annotated by Appalachian State graduate student Patrick Beekman, and used as input for OpenCV's train cascade classifier utility, which was set to learn Haar-like rectangular features in the images. This was performed for every major hive that the data was sampled from, along with one classifier created using data taken from all of the hives. The output was a set of cascade classifier XML files that were used to perform the detection stage for BeeID.

## 3 Algorithms

The algorithm consists of two main phases, detection, and tracking. For the initial detections, a pre-trained cascade classifier is run on the video frames to identify the most probable instances of bees. One filtering step is taken to remove bees whose detection completely contains another bee. The corner detector is also run on the frame to determine what OpenCV calls "good features to track." The set of detected corners are associated with bee rectangles that contain them. For every off-frame in between detections (a detection interval of 5 frames was used in the app), optical flow is performed on the set of corners, giving a (possibly smaller) set of feature points that were tracked between frames. For every bee, the set of points associated with it during the detection phase are corresponded with their (possibly) tracked points in the next frame. The average x and y offsets are computed, which gives an estimate to the overall displacement of the "bee rectangle". If the set of positively tracked points is below a certain threshold (30% in the app), then the rectangle is temporarily invalidated and not displayed on the next frame update. On subsequent detection phases, if a tracked, "active bee" has a upper left corner near enough to a newly detected bee (40 pixels Manhattan distance in the app), they are considered to be the same bee and the active bee is updated with the newly detected coordinates and corner points. Otherwise, that active bee is marked inactive and temporarily hidden from the display. Every frame, bees that have not been detected in over 10 frames are removed from the list of active bees and are no longer able to be updated or tracked.

### 3.1 Haar-like Feature Cascade Classifier

The features used for detection are part of a popular algorithm that has been widely used for face detection applications[1]. The classifier uses differences in rectangular areas of intensity. A 2 feature would consist of the difference between two rectangles, and so on. These can be computed very rapidly by precomputing a summed integral image, where the value of every pixel is the sum of pixel intensities from the upper left corner of the image. The sum of a specific rectangle can then be computed using only four subtractions. The cascade part functions similar to a decision tree, where a subrectangle can quickly be rejected if it fails the threshold on an early stage, whereas the later stages require more time to compute.

### 3.2 Shi Tomasi Corner Detector

In order to find good tracking pixels for every bee detection, OpenCV's goodFeaturesToTrack function is utilized, which uses the Shi Tomasi algorithm for detecting corners[2]. The algorithm computes the autocorrelation matrix for each pixel in the image, which is a Taylor series approximation for the sum of the squared differences between image patches. Good corners should have high variance in both horizontal and vertical directions, which corresponds to the eigenvalues of the computed matrices. If one of the two eigenvalues is low, this indicates a probable edge which should not be detected. The algorithm employed by Bee ID filters corners by their minimum eigenvalue, and then performs non maximum suppression to keep only the pixels with the highest response compared to their neighbors.

### 3.3 Lucas Kanade Optical Flow

For the tracking stage, pixels are corresponded between frames by approximating a solution to the Optical Flow equation at each pixel:

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1)$$

$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$

$$\vdots$$

$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$

Where $I_x, I_y, I_t$ are the horizontal, vertical, and time based derivatives for image intensity. Lucas Kanade uses least squares to approximate an equation for the resulting linear system, weighted using a gaussian window to give more importance to pixels closer to the center of the image[3]. The resulting velocity vector $[V_x, V_y]$ at each pixel gives the location of the corresponding pixel in the new frame.

## 4 Implementation

The implementation of the app was done in Javascript using the OpenCV.js and Canvas API's. Vue, a frontend Javascript framework, was also utilized to manage the state and different components of the application. After the OpenCV.js javascript library is loaded and the user starts the detection, a class is created to manage the current list of detected bees and their history, including the frames and timestamps for which they were tracked. A video HTML element is shown when the user hits the page, which is swapped out for an active canvas once the stream starts. Every frame that is read is either subjected to a detection or tracking phase as explained above. Bees are assigned random colors upon detection, which is to help the user visually confirm that the same bee is being tracked. After 40 frames of continuous tracking, a bee is stored to the "archive", which is a javascript list of Bee objects that is bound to list of html canvas elements, that each display the image data to their canvas upon being created and mounted to the DOM.

### 4.1 Bee Replays

If a user wishes to see when a bee was tracked and monitor it, he/she can select it from the bee archive by clicking. This temporarily switches the state of the class to a "Bee Focus" mode, where the video is rewinded to the point where the bee first occurred. Initially, the bee rectangle was updated to match its history according to frame count, but using real time to know when to advance the rectangle state gave more accurate results. No image processing is done during replays, just displaying the rectangle coordinates as stored when the bee was initially tracked and detected. After the history is fully replayed, the Bee Focus class calls its reset callback which restores the state of the video / canvas to before the replay was launched.

## 5 Room for Further Improvement

Next steps for BeeID would be a metric for quantitatively assessing the detection and tracking phases of the algorithm, perhaps using some manually labeled data sets. This would allow for hyperparameter tuning and experimentation with different algorithms to produce better results. In addition, the app is currently lacking any "recognition" component to determine if a bee was previously spotted or is in fact new. The current system for establishing correspondence (rectangle proximity) is good for short series of frames, relatively stationary bees, and isolated bees, but does not perform well for longer and busier video sequences. If the tracking instead utilized a more sophisticated model, such as a Kalman filter, then less false positives during tracking could be achieved. The current algorithm, which associates all features within a rectangle as belonging to a bee, then doing optical flow on those is prone to "identity theft", where a bee that enters another bee's rectangle controls the direction of optical flow and runs off with the rectangle. Perhaps a way to address that would be to ignore new feature points whose optical flow vectors differ highly from the set of previous feature points.

### 5.1 Recognition and AI

Once accurate sequences of an individual bee are captured and stored, further image processing can be done on those images to determine information about the Bee and the hive as a whole. One idea would be to use a neural network trained on different bee types (drone, worker, queen, etc.) and use that as an additional classification step. This information could be used to quantify hive diversity (e.g. percent of drones), and also to help Hive owners monitor a queen bee by identifying instances where she appears in the video.

## References

[1] P. Viola and M. Jones, "Robust real-time object detection," in *International Journal of Computer Vision*, 2001.

[2] J. Shi and C. Tomasi, "Good features to track," pp. 593–600, 1994.

[3] J.-Y. Bouguet *et al.*, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel Corporation*, vol. 5, no. 1-10, p. 4, 2001.