Final Project

Kevin Hardin

CS7367 – Machine Vision

April 23, 2025

“Introduction and Related Works”

Understanding honey bee activity is crucial for monitoring hive health and ensuring optimal environmental conditions for colony survival. Traditional beekeeping methods rely heavily on manual inspections, which are time-consuming and intrusive. With advancements in computer vision, it is now possible to monitor hives unobtrusively and automatically by analyzing visual data. This project explores the use of classical computer vision techniques for detecting, tracking, and analyzing honey bee movement at the hive entrance using a top-down camera setup.

The primary objective of this work is to develop a pipeline that uses classical techniques for robust background subtraction, bee detection, orientation estimation, and persistent ID tracking. The intended application is to assist beekeepers by providing real-time data on bee ingress and egress behavior without relying on computationally expensive neural network models.

The inspiration for this work comes from a variety of existing studies that utilize different computer vision methodologies for similar applications. Among the notable contributions, Lowe [1] introduced the Scale-Invariant Feature Transform (SIFT), which has proven robust for feature detection in insect imagery. Sledevič et al. [2] built on this by using keypoint-based methods for bee orientation detection, though their methods rely on higher quality images. Similarly, Dalal and Triggs [3] proposed Histogram of Oriented Gradients (HOG) to detect object orientations, though HOG is more tuned to object detection rather than body orientation.

Contour and shape-based methods also offer insight into bee pose estimation. Active contours ("snakes") as proposed by Kass et al. [4], and ellipse fitting by Fitzgibbon et al. [5], help delineate object boundaries and approximate orientation. However, these methods may falter in low-contrast or noisy environments. Lindeberg [6] and Grigorescu et al. [7] addressed such challenges by using blob and texture detection methods, though computational efficiency remains a concern.

Motion-based analysis, particularly optical flow techniques introduced by Horn and Schunck [8], enables orientation and velocity estimation by analyzing frame-to-frame pixel shifts. Lingenfelter et al. [9] explored insect-inspired motion estimation using optic flow, reinforcing the applicability of such methods in bee tracking. While learning-based models, such as those by Robinson et al. [10] and Sanaullah et al. [11], show promise in adapting to complex environments, they exceed the intended scope of this project, which favors classical techniques.

Template-based approaches (Brunelli [12]) and Hough Transforms (Duda et al. [13]) offer additional classical tools for shape detection. Behavioral studies such as those by Blut et al. [14] and datasets provided by Chaudhary et al. [15] support the development of bee behavior tracking systems. Though Cheng et al. [16] and Stürzl and Möller [17] focus on other insects or panoramic imaging, their methods contribute conceptual foundations for insect orientation analysis.

Together, these studies provide a foundation for building a real-time, classical computer vision-based honey bee monitoring system. The unique contribution of this project lies in integrating these classical techniques into a cohesive tool for real-time bee detection, orientation estimation, and velocity visualization.

“Project Design”

The architecture of the proposed system consists of five core components, however only four will be considered for this project: (1) video acquisition (the code will be tested using a pre-recorded video), (2) background subtraction, (3) contour extraction and feature estimation, (4) bee tracking with persistent IDs, and (5) orientation and velocity visualization.

Video is captured from a top-down camera mounted above the hive entrance. The video stream is first converted to grayscale and smoothed using Gaussian blur. A background model is created using a running average method (`cv2.accumulateWeighted`) to dynamically adapt to changing lighting conditions and environmental noise.

Foreground segmentation is performed by subtracting the background model from the current frame and thresholding the result. Morphological operations help abate noise. Contours are then extracted and filtered based on area to isolate bees from small debris or other background artifacts.

Centroids of the filtered contours are passed to a `BeeTracker` class, which uses a distance-based matching algorithm to maintain persistent IDs. Each bee’s trajectory is stored for visualization and analysis. Orientation is estimated using PCA on each bee’s contour, and velocity is approximated using optical flow vectors. All outputs will be capable of being rendered as overlays on the video feed in real time (for the operational version).

“Implementation”

The implementation is done entirely in Python using OpenCV and NumPy. A sample video (`bees.mp4`) is used as input. The background subtraction method employs a weighted running average, which balances adaptiveness with robustness against transient movement such as wind-blown grass.

Contours are extracted using `cv2.findContours`, and bees are filtered based on contour area. Centroids are computed via image moments, and stored in a dictionary with unique IDs. If a new centroid is close to an existing one from the previous frame, it is assigned the same ID.

To visualize trajectories, each bee’s position over time is stored and drawn as a polyline. The PCA-based orientation estimation uses the major eigenvector of the contour's point cloud to determine the bee’s heading. Velocity vectors are estimated using the Farneback method for dense optical flow, and directional arrows are drawn for visual feedback.

Several parameters can be tuned to improve the performance. First, an alpha value can be augmented according to the following relationship:

$$background = (1 - alpha) * background + alpha * current\_frame$$

Higher values of alpha increase the rate at which the background “refreshes”, and hence can mitigate “ghost traces”, or instances wherein tracked items persist beyond their presence in frame. However, increasing this value inordinately can cause parts of the tracked bees to be lost, and therefore confuse the tracker into believing that multiple items are present when they’re actually a single item. If the value of alpha is too low, the background will not refresh fast enough and outlines of tracked bees will be maintained even when the bee is not present in the outline any longer

Another tunable parameter is the minimum area required for a contour to be tracked. Each contour extracted is compared with this value and if the area is too low, it is ignored. This helps mitigate tracking of very small, background objects and limits tracking to only bees.

The minimum time (or number of frames) until a track is dropped when not present can also be configured. Through experimentation, it was found that a low value is useful in minimizing the number of errant objects that persist on screen. However, a value of 1 causes the item ids of the correctly tracked bees to constantly flicker to different values, which is not ideal when bees are supposed to retain the same item id when in frame.

One additional parameter that can be adjusted is a maximum distance value that defines how far a tracked item can move between frames before being tracked as a separate object. This is useful in minimizing any potential confusion by the tracker when an object moves either too quickly and in maintaining the separateness of individually tracked items.

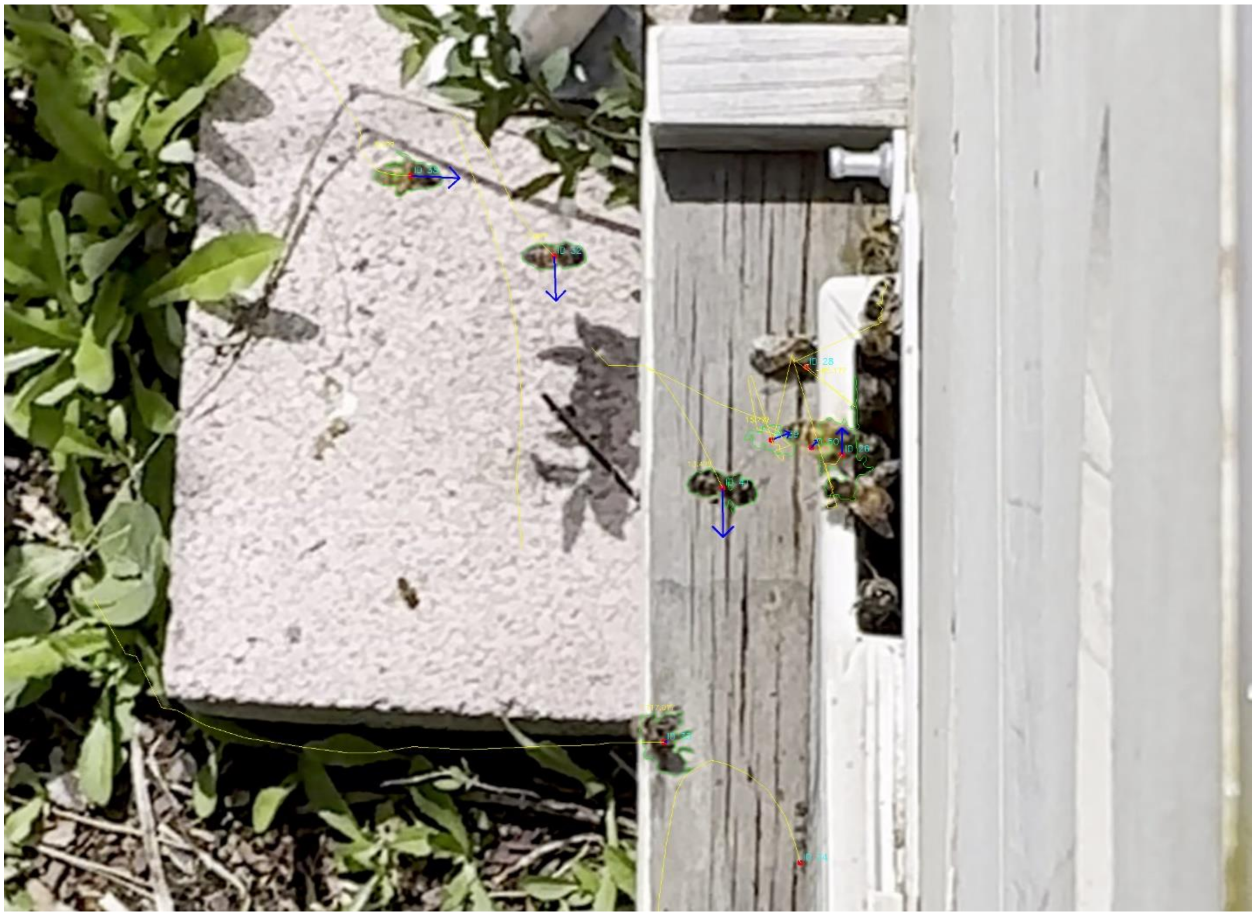
Finally adjustable in the model is the scaling of velocity arrows, which can be adjusted higher or lower (or off) if the screen becomes too cluttered.

“Results”

The evaluation was qualitative. The system correctly isolated and tracked individual bees, providing consistent IDs across frames. Trajectories and velocity vectors aligned with observable movement, and orientation angles were stable. The modularity of the code supports easy parameter tuning (e.g., minimum contour size, flow threshold) for adapting to other lighting conditions or hive configurations.

Although the approach is effective, there are several drawbacks. First, it occasionally suffers from ID switching during occlusions or rapid bee movement. Additionally, small debris can be misclassified tracked items if not filtered carefully. Any movement of the camera causes errant tracking of background objects and therefore for best background subtraction and rejection performance, the camera must be rigidly secured and immune to movement induced by things like wind.

Another problem is that often the shadows of bees are identified and tracked as regular bees. This is less than ideal, but not necessarily problematic. Since the trajectories of the shadow and the bee creating it are the same, velocity and pointing vectors will also be the same. This would simply inflate the number of bees entering or leaving the hive, but would ostensibly balance out over time. A greater problem lies in the difficulty the tracker has when the background is complex or variegated. Performance is much better over the fairly uniform rock as opposed to the plants further from the hive entrance, as seen below. If the background could be made to be more uniform or the hive porch be painted a light color, performance would be better.



Performance is not quite as close to real-time on consumer hardware as originally desired. Even a desktop computer was not fast enough to achieve 30 fps as desired. When using pre-recorded video, processing was limited to ~2 fps. This can cause major issues for the deployed product, as the hardware will be less powerful and a frame rate that is too slow will limit the effectiveness of the tracking. The background subtraction will ostensibly still operate, but if bees move too much between frames, the tracker will get confused and a cohesive track will not be achieved. There are several solutions to this; first, performance will likely be better when not having to display rendered frames in real time i.e. if either frames are rendered and stored for later viewing or if the information is instead distilled out into a text or csv file for use in post-data processing. Second, because the tracker is very modular, parts of it (such as the optical flow velocity determination) could be inactivated or deferred to post-processing, with only the background subtraction and object identification being performed in real time.

“Conclusion”

This project demonstrates the effectiveness of classical computer vision techniques for honey bee tracking at the hive entrance. The integration of background subtraction, contour-based detection, PCA orientation estimation, and optical flow velocity estimation creates a robust, real-time monitoring system.

While the system is limited in dealing with heavy occlusions or overlapping bees, it performs well under typical field conditions and requires minimal computational resources. This makes it suitable for deployment on embedded systems or edge devices. By avoiding reliance on neural networks, the solution remains interpretable, tunable, and lightweight.

Future improvements could include incorporating lightweight tracking filters (e.g., Kalman filters), integrating environmental context (e.g., weather data), or expanding to 3D orientation models. Additionally, exporting trajectory data for offline analysis would further enhance its utility for scientific or beekeeping purposes.

While the novelty of this project (in this AI-focused age) lies in its non-neural network-based approach, it’s utility could be increased by pairing it with a neural network classifier. The two approaches together yield a lighter weight, more reliable, and tunable solution than just relying on a neural network alone to track and classify. Further, pairing the solution with a neural network could increase the efficacy and speed at which the network can track and classify, enabling its use in a greater variety of (and potentially cheaper) edge devices.

"References"

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] T. Sledevič et al., "Keypoint-Based Bee Orientation Estimation and Ramp Detection at the Hive Entrance for Bee Behavior Identification System," *Agriculture*, vol. 14, no. 11, p. 1890, 2024.

[3] N. Dalal and B. Triggs, "Histograms of Oriented Gradients for Human Detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[4] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models," *International Journal of Computer Vision*, vol. 1, no. 4, pp. 321–331, 1988.

[5] A. Fitzgibbon et al., "Direct Least Square Fitting of Ellipses," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 476–480, 1999.

[6] T. Lindeberg, "Detecting Salient Blob-like Image Structures," *International Journal of Computer Vision*, vol. 11, no. 3, pp. 283–318, 1993.

[7] S. E. Grigorescu et al., "Comparison of Texture Features Based on Gabor Filters," *IEEE Transactions on Image Processing*, vol. 11, no. 10, pp. 1160–1167, 2002.

[8] B. K. P. Horn and B. G. Schunck, "Determining Optical Flow," *Artificial Intelligence*, vol. 17, no. 1–3, pp. 185–203, 1981.

[9] B. F. Lingenfelter et al., "Insect Inspired Vision-Based Velocity Estimation," *Bioinspiration & Biomimetics*, 2023.

[10] B. S. Robinson et al., "Online Learning for Orientation Estimation During Translation in an Insect Ring Attractor Network," *Scientific Reports*, vol. 12, p. 5798, 2022.

[11] Sanaullah et al., "Exploring Spiking Neural Networks," *Frontiers in Computational Neuroscience*, 2023.

[12] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, 2009.

[13] R. O. Duda et al., "Use of the Hough Transformation to Detect Lines and Curves," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[14] C. Blut et al., "Automated Detection of Encounter Behaviors," *Scientific Reports*, 2017.

[15] P. Chaudhary et al., "Honey Bee Image Dataset," Mississippi State University Libraries, 2024.

[16] Y. Cheng et al., “Estimating Orientation of Flying Fruit Flies,” *PLoS One*, 2015.

[17] W. Stürzl and R. Möller, “Insect-Inspired Active Vision Approach,” SpringerLink, 1970.

Source code printout.

```
import cv2

import numpy as np

from collections import OrderedDict


class BeeTracker:

    def __init__(self, max_disappeared=3, max_distance=500):

        # max_dissappeared is threshold number of frames for dropping a track if no movement is
detected.

        # max_distance is maximum allowable distance to match a new detection to an existing
track.

        self.next_object_id = 0

        self.objects = OrderedDict()

        self.disappeared = OrderedDict()

        self.trajectories = OrderedDict()

        self.max_disappeared = max_disappeared

        self.max_distance = max_distance


    def register(self, centroid):

        self.objects[self.next_object_id] = centroid

        self.disappeared[self.next_object_id] = 0

        self.trajectories[self.next_object_id] = [centroid]

        self.next_object_id += 1


    def deregister(self, object_id):

        del self.objects[object_id]

        del self.disappeared[object_id]
        del self.trajectories[object_id]

    def update(self, input_centroids):
        if len(input_centroids) == 0:
            for object_id in list(self.disappeared.keys()):
                self.disappeared[object_id] += 1
                if self.disappeared[object_id] > self.max_disappeared:
                    self.deregister(object_id)
            return self.objects, self.trajectories

        if len(self.objects) == 0:
            for centroid in input_centroids:
                self.register(centroid)
        else:
            object_ids = list(self.objects.keys())
            object_centroids = list(self.objects.values())
            D = np.linalg.norm(np.array(object_centroids)[:, None] - input_centroids, axis=2)
            rows = D.min(axis=1).argsort()
            cols = D.argmin(axis=1)[rows]
            assigned_rows = set()
            assigned_cols = set()
            for row, col in zip(rows, cols):
                if row in assigned_rows or col in assigned_cols:
                    continue
                if D[row, col] > self.max_distance:
                    continue
                object_id = object_ids[row]
                self.objects[object_id] = input_centroids[col]
                self.trajectories[object_id].append(input_centroids[col])
                self.disappeared[object_id] = 0
                assigned_rows.add(row)
                assigned_cols.add(col)
            unused_rows = set(range(D.shape[0])) - assigned_rows
            unused_cols = set(range(D.shape[1])) - assigned_cols
            for row in unused_rows:
                object_id = object_ids[row]
                self.disappeared[object_id] += 1
                if self.disappeared[object_id] > self.max_disappeared:
                    self.deregister(object_id)
            for col in unused_cols:
                self.register(input_centroids[col])
        return self.objects, self.trajectories


def estimate_orientation(contour):
    if len(contour) < 5:
        return None
    data_pts = np.array(contour).reshape(-1, 2).astype(np.float32)
    mean, eigenvectors = cv2.PCACompute(data_pts, mean=None)
    angle = np.arctan2(eigenvectors[0, 1], eigenvectors[0, 0]) * 180 / np.pi
    return angle


def main():
    alpha = 0.15 # determines how quickly the background adapts to changes per equation:
background = (1 - alpha) * background + alpha * current_frame.  Higher alpha adapts faster
```

```
    min_area = 1500 # min area for a contour
    tracker = BeeTracker()
    cap = cv2.VideoCapture("bees.mp4")
    background = None
    prev_gray = None

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray_blurred = cv2.GaussianBlur(gray, (7, 7), 0)
        if background is None:
            background = gray_blurred.copy().astype("float")
            prev_gray = gray_blurred.copy()
            continue
        cv2.accumulateWeighted(gray_blurred, background, alpha)
        background_uint8 = cv2.convertScaleAbs(background)
        diff = cv2.absdiff(gray_blurred, background_uint8)
        _, thresh = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)
        thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, np.ones((3, 3), np.uint8))
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        centroids = []
        for cnt in contours:
            if cv2.contourArea(cnt) < min_area:
                continue
            M = cv2.moments(cnt)
            if M["m00"] == 0:
                continue
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])
            centroids.append(np.array([cx, cy]))
        objects, trajectories = tracker.update(np.array(centroids))
        for object_id, centroid in objects.items():
            cv2.circle(frame, tuple(centroid), 5, (0, 0, 255), -1)
            cv2.putText(frame, f"ID {object_id}", (centroid[0] + 5, centroid[1] - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 0), 1)
        for cnt in contours:
            if cv2.contourArea(cnt) < min_area:
                continue
            orientation = estimate_orientation(cnt)
            if orientation is not None:
                rect = cv2.boundingRect(cnt)
                x, y = rect[:2]
                cv2.putText(frame, f"{orientation:.1f}°", (x, y - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 255, 255), 1)
                cv2.drawContours(frame, [cnt], -1, (0, 255, 0), 1)


        flow = cv2.calcOpticalFlowFarneback(prev_gray, gray_blurred, None,
                                            0.5, 3, 15, 3, 5, 1.2, 0)
        for object_id, centroid in objects.items():
            x, y = centroid
            if 0 <= y < flow.shape[0] and 0 <= x < flow.shape[1]:
```

```
                fx, fy = flow[y, x]
                if np.hypot(fx, fy) > 1:
                    end_point = (int(x + fx * 2.5), int(y + fy * 2.5)) # 2.5 corresponds to velocity arrow
length
                    cv2.arrowedLine(frame, (x, y), end_point, (255, 0, 0), 2, tipLength=0.3)


        prev_gray = gray_blurred.copy()


        for object_id, points in trajectories.items():
            if len(points) < 2:
                continue
            for i in range(1, len(points)):
                if points[i - 1] is None or points[i] is None:
                    continue
                cv2.line(frame, tuple(points[i - 1]), tuple(points[i]), (0, 255, 255), 1)


        cv2.imshow("Bee Tracking", frame)
        if cv2.waitKey(30) & 0xFF == 27:
            break


    cap.release()
    cv2.destroyAllWindows()


if __name__ == "__main__":
    main()
```