Individual Bee Identification of Solitary Bees using Computer Vision

Gregory Glatzer, Natalie Boyle, Christina M. Grozinger

***Abstract*—**

***Index Terms*—**

I. Introduction

There are many benefits that come from being able to track the movement of solitary bees. Being able to see when and how long bees are working on their tubes, as well as how long a bee is out foraging, tells us a lot about the health of the bee. Having data about bee activity also allows us to see how other factors, such as precipitation, temperature, time of day, and humidity affects their activity. Often, tracking bee activity is done by marking the bees and monitoring their movement manually [CITATION NEEDED]. This is tedious and time consuming, which makes tracking bees prohibitive for many bee keepers, especially for the backyard beekeeper. Having a way to automate the tracking of your bees would allow many beekeepers to benefit from the findings that come with understanding your bees' daily activities. This is the motivation behind developing a computer vision-based method for tracking bee movement.

II. Related Works

There have been several past studies that have used video feeds as a method to analyze solitary bee behavior, as well as how solitary bee activity is affected by external factors. McKinney and Park (2012) used video feeds from 4 infrared cameras to monitor various intranest and collecting activities of *Osmia cornifrons*. They used these video feeds to measure the effect of time of day, temperature, and precipitation on bee activities.

Another paper that attempts the automate the tracking of solitary bees with computer vision is Knauer et al. (2022). This paper uses a combination of 4 deep learning networks to perform bee detection, nest cavity detection, identify bee marker tags, and read bee marker tags. There are several limitations to this approach, however - a human must physically mark each bee, and there are limits on the kind of bee and number of individual bees: a maximum of 24 unique tags, currently only on Osmia cornifrons. This paper elaborates on how to extend their models to work with a higher count of individual bees, more colors of tags, and different species, however this would require extensive work including gathering high quality images of tags and bees to generate the dataset, let alone the difficulties that can come with retraining their neural networks on the new data.

Our approach presents several contributions compared to these existing approaches: (1) only one camera is needed per setup, (2) bees do not need to be marked ahead of time, and (3) can support other bee species by adjusting motion detection parameters (further work needed on quantifying the success of this approach on other bee species).

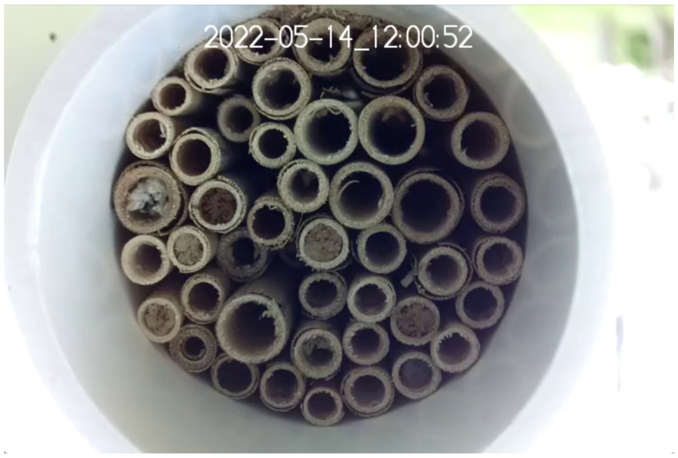

Fig. 1. The bee hotel setup, including a timestamp added by the Raspberry Pi.

Date	Start Time	End Time
05/14/2022	10:00:00	10:59:00
05/14/2022	11:00:00	11:59:00
05/14/2022	12:00:00	12:59:00

TABLE I
RECORDING DATES AND TIMES.

III. Methods

A. Data Collection

To collect videos of solitary bee activity, a bee hotel was set up with 44 tubes made of natural phragmites reeds with diameters varying from approximately 5 to 10 mm. These tubes were places in a PVC tube with a diameter of 3 inches. A picture of this setup can be seen in Figure 1.

To record the bees, a 3B+ Raspberry Pi with a HD lens/camera sensor attachment recording at a frame rate of 50fps was used. Although the frame rate was set to this, we observed that it was inconsistent. This led to the technique to extract the timestamp, which is explained in III-B5. Each recording session was for an hour, and saved as an mp4. Table I shows the full list of recordings.

Event Component	Description
Frame	The frame at which the event happened.
Timestamp	The timestamp (in real time, not seconds in the video) that the event happened.
Bee ID	The ID of the individual bee. This is also associated with an individual tube.

TABLE II
COMPONENTS OF AN EVENT FOR INDIVIDUAL BEE IDENTIFICATION.

B. Individual Bee Identification

Each video can be represented by a set of events over time that describe the activities of the bees in the target PVC tube. An event consists of 3 components, described in Table II. As a video is processed, these events are calculated in real time.

To calculate these events, we used a computer vision-based approach implemented with the popular Python library for computer vision, OpenCV. Each frame (after frame 30 to give the camera some time to adjust to the light and get in focus) is read into memory, preprocessed, and it is determined if motion was detected (relative to the last frame). If motion is detected, we compare this motion to the next couple frames to reduce false positives. Finally, an event is calculated and logged for this motion. Let us now dive into these steps in further detail.

1) Reading the Video Feed: Reading the frame into memory is fairly simple. OpenCV provides a standard way to read a video capture, frame by frame, into memory for processing. Currently, this is being used to read an mp4 file from disk, however this could also be configured for real-time processing of a camera feed.

2) Image Preprocessing: To optimize the performance of motion capture, we must preprocess the image. First, the image is converted from BGR color channels to RGB. This is simply because OpenCV reads the image as BGR, but we want to display it correctly. We then convert the image to gray-scale to speed up processing (since the actual processing doesn't rely on color). This reduces the image from three channels down to two, reducing the data that needs to be processed per frame by a third. Finally, the image is slightly blurred using a Gaussian Blur with a kernel size of 5x5 and a standard deviation of 0. This helps eliminate noise and small objects such as dirt in the image which may be picked up as false positives for detecting bees.

3) Detecting Motion: To detect motion, we compare each frame to the previous frame by calculating a "difference frame". This difference frame is calculated by taking the absolute difference of every pixel in the frame to every pixel in the previous frame. This difference frame is then dilated using a kernel size of 5x5, which makes the difference frame more suitable for contour detection. Then, we only keep pixels above a given motion threshold. This is a tunable parameter to adjust how much motion is needed to trigger a detection. We used a motion threshold of 10.

Once this difference frame is calculated, we find contours in the frame. The OpenCV documentation describes a contour as "a curve joining all the continuous points (along the boundary), having same color or intensity" (Bradski, 2000). Contours can be used to find groups of pixels in the difference frame, which we can infer to be an object.

For each detected contour, we calculate the area of its bounding box (smallest non-rotated rectangle you can draw encompass the entire contour with). To reduce false positives of very small (ants, dirt) and very large (people and cars in the background) detected objects, we will only consider objects with a bounding box area between a range of 400 and 1000 pixels. This range can be tuned, as needed, to adjust to varying setups and types of bees.

4) Individual Bee Identification and Contour Window: Once we have our set of contours that have been determined to be bees, we must associate each contour with an individual bee. To do so, we number each tube in the hive, and calculate the Euclidean distance from the center of each tube to the center of the contour. The contour is then associated with the tube number it is closest to. We call this the "Bee ID". If a contour is not within a certain distance threshold of ANY tube, then we ignore it. This helps prevent background movement outside the hotel from causing false positives.

One final step must be considered to make sure that we are not associating a bee with another tube during its flight path on the way to its final tube. *Osmia bicornis* will also "probe" the other tubes to find theirs, only to finally enter their own tube and disappear from view (Knauer et al., 2022). To account for this, we have developed a method to ignore contours detected during the flyover and probing phase, and only on the entry phase. We define a "contour window" that contains *N* sets of contours across *N* consecutive frames, where *N* is the size of the contour window. For each contour detected, we wait *N-1* frames to see if there are any more contours that appear that overlap with this contour. If there are any, we ignore this contour. If there are not, we can consider this the end of the flyover or probing phase, and that the bee has found their tube and entered it.

Figure 2 illustrates this process. Notice how frames *t=0* through *t=4* are red, indicating that there is an overlapping contour somewhere in the window of the next *N* frames. It is not until the contour found during frame *t=5* that there are no more contours that overlap in the next *N* frames. We now calculate the closest tube to the center of this contour, which ends up being Tube #4. Thus, we associate this detected bee with Bee ID=4 at the timestamp extracted at frame *t=5*.

5) Extracting the Timestamp: Associating a timestamp to each event allows us to quantitatively measure the performance of our method in comparison to annotated ground truths of events by a human observer. Additionally, timing of events can provide information on foraging rate, which may vary with landscape quality and weather, and can thus serve as an indicator of stress (McKinney and Park, 2012).

We tried several methods for converting the frame number at which motion was detected to a timestamp, using the frame rate as a conversion constant. This failed to be reliable, as it appears that the Raspberry Pi recorded at an inconsistent frame rate. This called for a more novel, albeit more fragile, approach. Our current approach is to use Optical Character Recognition (OCR) to extract the timestamp in the frame (see Figure 1) that motion was detected on. This requires the user of the tool to provide the coordinates of the bounding box of the timestamp in the video, so the OCR knows where to

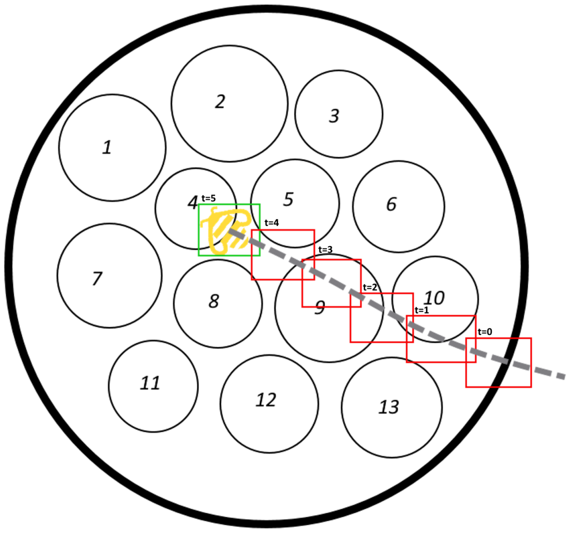

Fig. 2. The contour window in action. Contours in time steps *t=0* to *t=4* have an overlapping contour somewhere in the next *N=10* frames (red boxes), so they are ignored. The contour at *t=5* has no overlapping contours in the next *N=10* frames (since the bee enters tube 4), so the bee is given a Bee ID of 4.

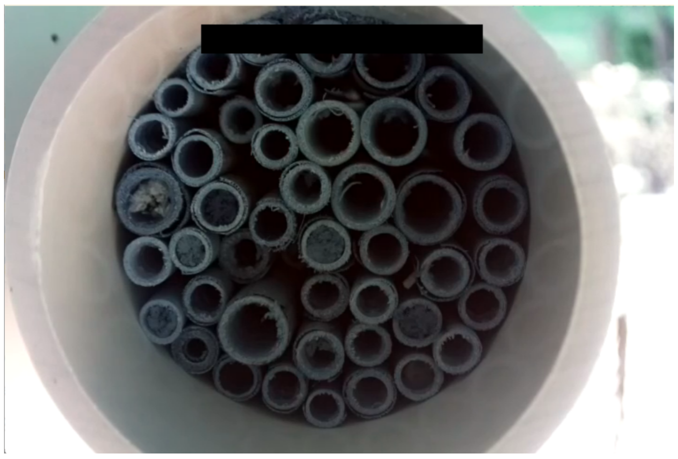

Fig. 3. The masked timestamp after reading it with the OCR (see top middle black rectangle).

grab the text of the timestamp from. Additionally, we mask the portion of the image where the timestamp is from the motion detection algorithm, so the changing timestamp is not detected as motion in the image. See Figure 3, which illustrates the final masked timestamp as a black rectangle on the image.

We used the popular pytesseract Python package for our implementation of OCR, which is based on Google's Tesseract OCR Engine. To optimize the OCR's performance, some preprocessing (of the raw frame) must be done. Like before, we convert to RGB and then grayscale. We then invert the image (so the text is black) and add 10 black pixels of padding on all sides. Pytesseract OCR takes parameters to tune the performance, namely values for OEM and PSM. We used OEM=3 and PSM=6. This works without error about

Bee ID	Start Timestamp	End Timestamp	Event Type
16	10:01:18	10.01.31	work
16	10:01:31		exit
27	10.02.30		enter
13	10.02.50	10.02.50	enter
27	10.03.18	10.03.21	turn around

TABLE III
GROUND TRUTH EVENTS

77.8% of the time, with the other 22.2% of the time the timestamp needing some light cleaning. This is primarily due to the background hurting the OCR's performance, since the timestamp is overlaid on the image. Ideal video feeds would place the timestamp on a black background.

IV. RESULTS

To evaluate the tool, we collected ground truths for an hour of footage by having a human log events. The annotator was asked to log the timestamp seen in the video for whenever a bee appeared, which bee it was (based on a predetermined numbering of the tubes), as well as the what the bee was doing. The bee could either "enter", "exit", "work", or "turn around" (in the tube) during an event. If an event lasted for a period of time, the annotator was asked to make the entire period of time the event occurred as a single event, with a start and end timestamp. Events that do not have a notable duration have the same start and end timestamp. An example of ground truth events is shown in Table III.

With these ground truths we can determine how many of the detected events overlap with the ground truth. To account for slight errors in alignment of detected and ground truth events, we allow a buffer of 3 seconds. If a detected event either overlaps or is within the buffer of a ground truth event, and is the same Bee ID, we call it a True Positive. With this matching methodology, we can calculate the number of True Positives, False Positives, and Precision for each bee.

Figure 4 illustrates the performance against the ground truths for each bee. In general, a bee either does very well or very poor. Some Bee IDs get triggered very often in a myriad of false positives (see Bee ID 36, for example). By observing the tool at work, we have found that these false-positive-prone Bee IDs are most often due to either pieces of tube debris flapping in the wind nearby, or another active bee that is in a neighboring tube, whose body often triggers the incorrect Bee ID.

Using the formula for precision, which is given by $precision = \frac{TP}{TP+FP}$, where TP is the number of true positives and FP is the number of false positives, we can calculate the distribution of precision for all bees as $\mu = 19.297\%$ and $\sigma = 25.385\%$. The highest precision achieved was 86%. If we disregard WHICH BEE was detected, and just that A BEE was detected, we can detect bees with 71% precision. This means that if a human observer were to use the logs of this tool as a guide, they would find a bee at a timestamp 71% of the time.

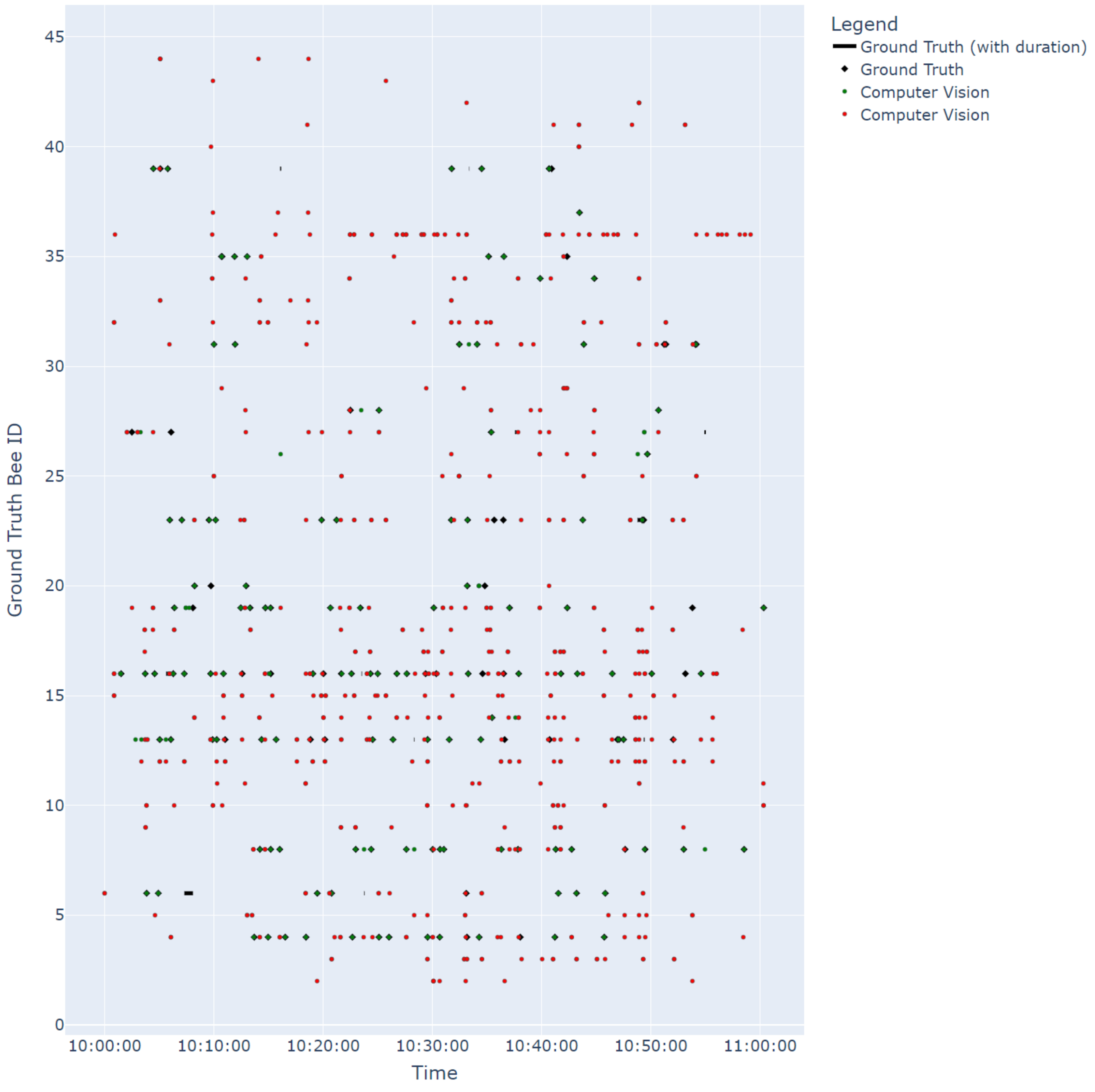

Fig. 4. Comparing Ground Truth events against the events detected by the computer vision application.

V. Discussion

A drawback of this approach is that our bee detection relies solely on motion detection. Although some precautions were put in place (such as looking for certain contour sizes, and ignoring contours beyond a certain distance from tubes) to reduce false positives, there were still many false positives noticed, mainly caused by ants, dirt, and loose particles such as fraying parts of tubes blowing in the wind. Our method assumes all motion that passes our contour check to be a bee. A smarter system that combines deep learning with traditional computer vision approaches is needed which can more intelligently reduce these false positives. There is a growing body of work in identifying insects in video feeds, including work on bee hotels (Knauer et al., 2022), and a more recent work in identifying insects in natural environments using YOLOv5 (You Only Look Once) (Bjerge et al., 2023). The integration of neural net-based tools, and their fine-tuning to adjust to bees, could greatly improve performance.

Currently, one of the most difficult problems with this approach is fighting with the limitations that come with the setup of our bee hotel. The close proximity of the tubes means that bees are often flying over other tubes, or crawling over them on the way to their tube and/or while working on their hive. This causes lots of false positives that are very hard to filter out. A better setup, with farther spaced tubes, could greatly reduce the false positives that come from bees incorrectly triggering other Bee IDs.

VI. Future Works

Reflecting on the current performance of a solely computer vision-driven tool, there is much work to be done in implementing other, more complex methods on top of the computer vision base. An alternative to the contour window technique could be the Kalman filter, which has been tried and tested in tracking multiple objects over time in real-world environments (Li et al., 2010). Another alternative could be the DeepSORT object tracking algorithm introduced by Wojke et al. (2017).

References

Bjerge, K., Alison, J., Dyrmann, M., Frigaard, C. E., Mann, H. M. R., and Høye, T. T. (2023). Accurate detection and identification of insects from camera trap images with deep learning. *PLOS Sustainability and Transformation*, 2(3):1–18.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Knauer, A. C., Gallmann, J., and Albrecht, M. (2022). Bee tracker—an open-source machine learning-based video analysis software for the assessment of nesting and foraging performance of cavity-nesting solitary bees. *Ecology and Evolution*, 12(3):e8575.

Li, X., Wang, K., Wang, W., and Li, Y. (2010). A multiple object tracking method using kalman filter. In *The 2010 IEEE International Conference on Information and Automation*, pages 1862–1866.

McKinney, M. I. and Park, Y.-L. (2012). Nesting activity and behavior of ¡i¿osmia cornifrons¡/i¿ (hymenoptera: Megachilidae) elucidated using videography. *Psyche*, 2012:814097.

Wojke, N., Bewley, A., and Paulus, D. (2017). Simple online and realtime tracking with a deep association metric.