A Computer Vision Approach to Monitoring beehive Activity

Maxime de Belloy
Stanford University
belloy@stanford.edu

1. Abstract

This paper introduces a non invasive beehive monitoring approach by quantifying entrance activity for a given hive through video analysis. Our approach uses a three stage pipeline for counting. It first detects the hive entrance ramp and resizes the image to only look at the ramp and then it detects individual bees in each frame. Finally, it tracks each bee's trajectory using a Kalman filter to distinguish arrivals from departures. We create a pipeline capable at running live at over 60 fps while counting individual bees' entrances and exits at above 85% accuracy.

2. Introduction

Bees are essential to the proper pollination of many plants that are a key part of the worldwide agricultural production. The common honey bee, Apis Mellifera, is dying at unprecedented rates, in particular because of colonies collapsing because of attacks from Varroa mites. Traditional beekeeping and even most monitoring systems require opening the hive and pulling out individual frames to assess a hive's health. This stresses the bees and should be done as infrequently as possible. This project therefore aims to use computer vision to un-intrusively monitor a hive's health by keeping track of a hive's activity.

This system takes video footage of the entrance of a hive as input. Then, we look at tallying the number of bees that enter or exit a hive over the course of the video. If the video quality is good enough, this can then be integrated with other pre-existing models for pollen and Varroa mite detection. The result is then a histogram through time of the number of bees that enter or exit the hive, as well as the percentage of arriving bees carrying pollen or a mite.

In particular, this requires detecting the hive's take-off ramp from the video and then detecting for each frame all bees on the ramp. The ramp is a piece of wood sticking out from the hive where bees can take off and land from. Then, across frames, bees' trajectories have to be computed to track their movements and determine if the bee is leaving or arriving into the hive. For the ramp detection, both a basic CNN and a finetuned YOLO11[4] were tested. Then, the frame is cropped to the ramp, and another finetuned YOLO model is used to find the bees. Finally, a Kalman filter is used to predict each bee's trajectory and determine if it is entering or exiting the hive.

3. Literature review

Some research has already been published in the hive monitoring space. The most important for this project is a paper by Kongslip et al[5]. In their paper, the authors set up a beehive with clear paneling so that they could continuously film the inside of the frame. They then implemented a bee detection model, and used a Kalman filter tracking system to track bee movements across the frame. The basis of their algorithm will be essential in this project, although the environment is quite different. In this paper, they were looking at frames with hundreds of bees in each frame, where each bee moves very slowly since they do not fly inside the hive. Moreover, the whole frame was in shot the entire time, under seemingly constant lighting conditions. In our case, we are filming the entrance of the hive, with fewer, but much faster moving bees.

For a more general overview of the field, we turn towards research from Bilik et al[2]. They surveyed over 50 papers related to automated beehive monitoring using computer vision, categorizing approaches into conventional techniques, CNN-based classifiers, and object detectors. They found that there is a steep increase in publications in this field since 2016, and in particular an increased adoption of object detection for bee traffic monitoring since 2021. They split their survey into four primary categories: pollen detection, Varroa mite detection, traffic monitoring, and general bee inspection. Their research particularly notes the importance of relatively compact models that can run on in the field embedded systems using Raspberry Pi or NVIDIA Jetson platforms for real-time processing. They posit that detection approaches like YOLO and SSD are the most common since they can handle multiple fast-moving objects simultaneously. For this reason, we decide to fine tune a YOLO model for our needs. Their paper also gives an overview of the available datsets in the field, which we can pick and choose from.

It is also important to look at Mahajan et al's[7] paper. They present a beehive monitoring system called NeuralBee. Their study compares object detection models like YOLOv5, v7, v8, and SSD for detecting Varroa mite infestations. They found that their finetuned YOLOv5 model achieved a precision of 0.962 and mAP@0.5 (metric explained later) of 0.974. We note that they also include interesting non-vision approaches. They built an audio analysis system that classifies a hive as strong or weak with 99.8% accuracy using Mel spectrograms and Mel-frequency cepstral coefficients (MFCCs) as input features. The audio component is a smart, non invasive monitoring method that would be cheaper to implement, and works well in tandem with vision methods.

We also note that there are a few pre-trained, open source models for Varroa mite detection like the Varroa Detector model[1] or the Varroa Mites Detector model [12] with performances of 97.4% mAP@0.5 (97.0% precision) and 79.4% mAP@0.5 (79.7% precision) respectively, offering accessible solutions for beekeepers who may not have the resources to train their own models from scratch.

While these require very close up images of particular bees, we note that these models could be a good addition to our model if it is deployed on real hives for a better overall understanding of hive health.

4. Dataset

We use a detection dataset from Sledevic et al[10], that contains 7,200 annotated still frames captured from eight different beehives with bounding boxes for each bee, an example of which (cropped to just show the hive ramp) can be seen with plotted labels in Figure 1. The images are sized at 1920 by 1080. This dataset also includes 156 images of hive ramps / entrances with bounding box annotations, which will help us train a detection model for both the ramps and bees. To test our model, we will use a 2 minute video of the entrance of one of these hive, labeled with bee trajectories. To speed up training and match the YOLO defaults, we start by preprocessing these images and reformatting them to 640 by 640. The image ratio is kept the same, and padding is added to fill the square.

5. Methods

5.1. Ramp Detection

We start by training a baseline model for the hive's ramp detection on our 156 images, using a basic 3 layer CNN we train from scratch. It's architecture is shown in Figure 2.

We then decide to finetune a YoloV11[4] nano model on the same dataset following the results found from the Mahajan[7] paper described above. We use a batch size of 8, image size of 416px, and train for 200 epochs. We pick the smallest available model (nano) since the task is single class and relatively easy. We also increase the box loss weight (15.0), and decrease the classification loss weight (0.25). This is because this is a single class problem where we care more about getting the exact correct bounds for the ramp rather than being 100% confident we have found a ramp. This is especially true since we assume that there will always be a ramp present in the image.

Getting the bounding box for the ramp lets us focus only on the part of the image where bees are entering and exiting the hive. We don't want to track or count bees that are simply flying in frame or exiting the frame unless they are exiting the ramp into the beehive.

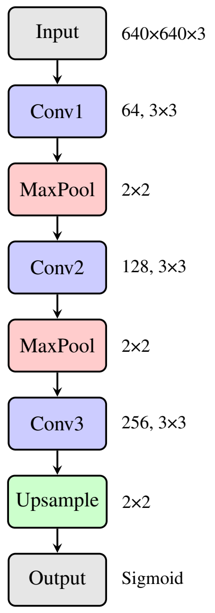

Figure 2. CNN architecture for submarine detection.

5.2. Ramp detection results

Having trained our baseline CNN and fine tuned the YoloV11 model for ramp detection, we show in Figure 3 that the models accurately detect the ramp from the image. From immediate visual inspection, both models seem to be working well.

We compare our models on Mean Average Precision (mAP). This metric relies on the Intersection over Union (IoU) metric defined as:

$$IoU(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \quad (1)$$

We use two versions in particular:

- **mAP@0.5**: Average precision calculated at an IoU threshold of 0.5. A predicted bounding box is considered a true positive if its IoU with ground truth exceeds 0.5.

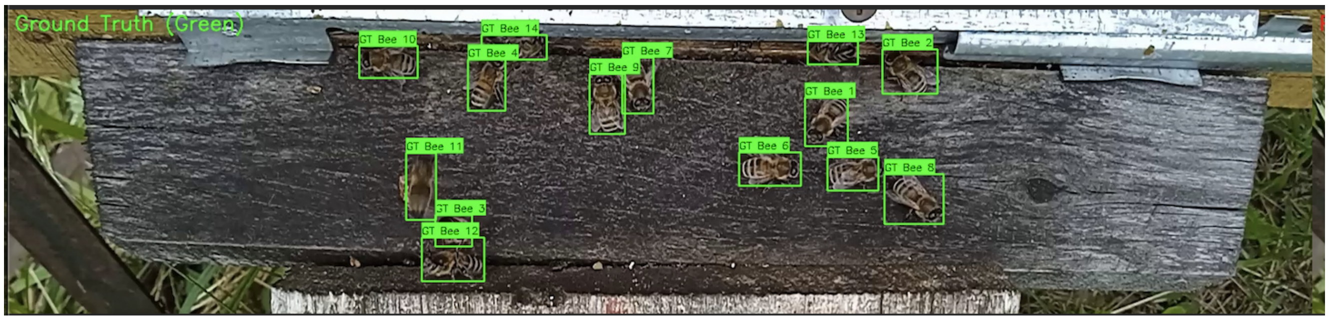

Figure 1. Labelled bee dataset example

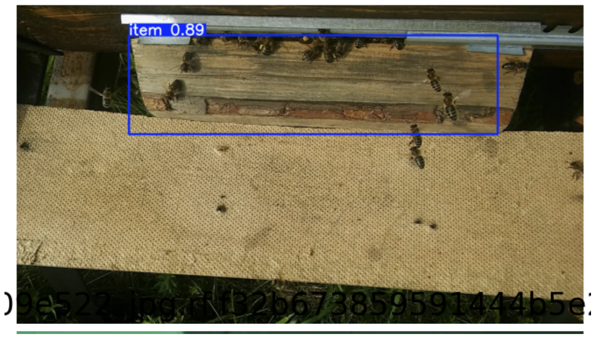

Figure 3. YOLOv11 Ramp Detection

- **mAP@0.5:0.95**: Average of AP values computed across multiple IoU thresholds from 0.5 to 0.95 in 0.05 increments. This metric penalizes poor localization more severely than mAP@0.5.

Table 1. Ramp Detection Model Performance Comparison

Metric	Baseline CNN	YOLOv11
mAP@0.5	0.7119	0.9950
mAP@0.5:0.95	0.2533	0.9810

We see that the finetuned Yolo model greatly outperforms our simple CNN baseline. While both models are able to pick out the ramp in the image, the baseline usually uses a too large or poorly centered bounding box, causing its scores to decrease. We decide to move forward with the former for our detection pipeline.

5.3. Bee Detection

5.3.1 Cropping then training

For bee detection, we start by cropping the 7200 images from the Sledevic dataset[10] using our ramp detection model to focus on the area of interest. The Yolo model at inference is very quick, taking 0.3ms per image. This gives us cropped images as seen in Figure 1. Our images get cropped from 1920 by 1080 to around 1700 by 380 depending on the bounding boxes from the detection model. We also have to change the labels to match the cropped image. We apply

$$\begin{cases} x' = \frac{x \cdot W - x_1}{x_2 - x_1} \\ y' = \frac{y \cdot H - y_1}{y_2 - y_1} \\ w' = \frac{w \cdot W}{x_2 - x_1} \\ h' = \frac{h \cdot H}{y_2 - y_1} \end{cases} \quad (2)$$

with x, y, w, h the original normalized bee bounding box coordinates, (x_1, y_1, x_2, y_2) the detected ramp region's top left and bottom right corner, and W / H the original image width / height respectively. We get transformed coordinates (x', y', w', h') in the cropped image space. We clip these from $[0, 1]$, which removes the labels that are outside the new image space.

Again, we decide to finetune another YOLOV11 model here. Since we have a much larger dataset, we train for 15 epochs (3 of which dedicated for warmup) with a batch size of 32 on the small model. We also use automatic mixed precision (AMP) for faster training.

5.3.2 Training then cropping

However, we note that YOLO models can only be trained on square images. Our cropped images have ratio around 5:1 depending on the ramp, with image sizes of around 1700 by 380. When scaled down and padded, the resulting image used for training is mostly gray padding vertically, with the bees being squashed in the center rows of the image. To combat this issue, we decide to also test fine-tuning our model before cropping to the ramp space. These images however, also contain the grass around the hive and the hive itself in frame, meaning that the model will have to learn to find bees in more diverse environments.

Again, we train our YOLO small model for 15 epochs and use AMP, freezing the top 10 layers and downsampling the image to 640 by 640.

5.4. Bee Detection Results

For each model, we also calculate precision and recall:

$$\text{Precision} = \frac{TP}{TP + FP} \qquad \text{Recall} = \frac{TP}{TP + FN} \quad (3)$$

where TP, FP, and FN represent true positives, false positives, and false negatives, respectively.

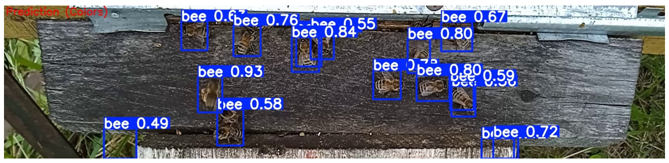

Figure 4. Bee Detection Predictions

We get the results shown in Table 2, and we plot an example of the model's output in Figure 4. From this and other visualized predictions, we note that the model accurately counts many different target bees on the ramp, but also seems to be flagging some false positives in the grass around the ramp and in some of the striated wood grain. We see from the table that cropping the image before training yields better results in our reported metrics excluding precision, with a 10% performance increase in mAP@0.5:0.95 and a 6% increase in recall.

Metric	No Cropping	Ramp Cropping
Precision	0.955	0.948
Recall	0.883	0.935
mAP@0.5	0.949	0.970
mAP@0.5:0.95	0.647	0.712

Table 2. Bee Detection Performance: Impact of Cropping On Ramp Before Training

Given the noticed improvement, we decide to use the model finetuned on pre-cropped images for the final pipeline.

6. Bee Tracking

Now that we have both a ramp and bee detection model, we need to track bee trajectories. This will be tested on a labeled video with 5973 frames. The labels are for all bees in the video, not just the ones on the ramp. To define our ground truth, we use our ramp detection model to get a bounding box for the ramp in the video. We then count the number of labeled trajectories that pass through the ramp bounding boxes, and that end with the bee crossing the ramp towards the hive (top of the bounding box since the hive is above the ramp in these videos). These are marked as entrances. All other trajectories that go through the bounding boxes are counted as exits.

In our test video, we get ground truth values of 346 entrances and 185 exits for a total trajectory count of 531 bees.

6.1. Baseline

We define a very simple baseline. In each frame, we use our bee detection model to count the number of bees on the ramp. To decrease noise from flickering, we average bee counts over 2 frames. Between each pair of frames, we find the difference of number bees on the ramp, and assume that half of them are entrances and half are exits.

6.2. Trajectory Tracking

We start by detecting the ramp by running our ramp detection model over the first 100 frames of the video and taking the median for each of the four corners of the bounding box. This gives us a good estimate of the ramp.

Then, for each frame, we use our bee detection model to find the bees currently in frame. To reduce detection flickering, we buffer k consecutive frames and cluster detections where $\|\text{center}(d_i) - \text{center}(d_j)\| < 30$ pixels. Each cluster is averaged as $\bar{d} = \frac{1}{|C|}\sum_{d\in C} d$ before feeding to the tracker. Like above, we run two tests, feeding in both the cropped and non cropped image to the model for bee detection.

6.3. Kalman Filter Tracking

If we are on the first series of averaged frames, then we initialize Kalman filters for each of the detection. These are set up with state vector $\mathbf{x} = [x, y, \dot{x}, \dot{y}, w, h]^T$ encoding position, velocity, and bounding box dimensions. The model is defined by:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

For following frames, we start by predicting every Kalman filter forward one step. For each currently active track τ_i, we compute:

$$\begin{cases} \mathbf{x}^i_{k|k-1} = \mathbf{F}\mathbf{x}^i_{k-1} \\ \mathbf{P}^i_{k|k-1} = \mathbf{F}\mathbf{P}^i_{k-1}\mathbf{F}^T + \mathbf{Q}. \end{cases} \tag{5}$$

where $\mathbf{Q} = 0.01\mathbf{I}_6$ represents process noise.

Now we have predicted locations for each bee that was previously in frame, and we need to either match the detected bees from the current frame to previous tracks or have them start their own tracks.

6.4. Hungarian Algorithm for Data Association

To do this, we implement the Hungarian Algorithm [6]. With n existing tracks and m new detections, we create a cost matrix $\mathbf{C} \in \mathbb{R}^{n\times m}$ such that

$$C_{ij} = 1 - \text{IoU}(\text{track}_i, \text{detection}_j) \tag{6}$$

with IoU as defined in equation 1 above.

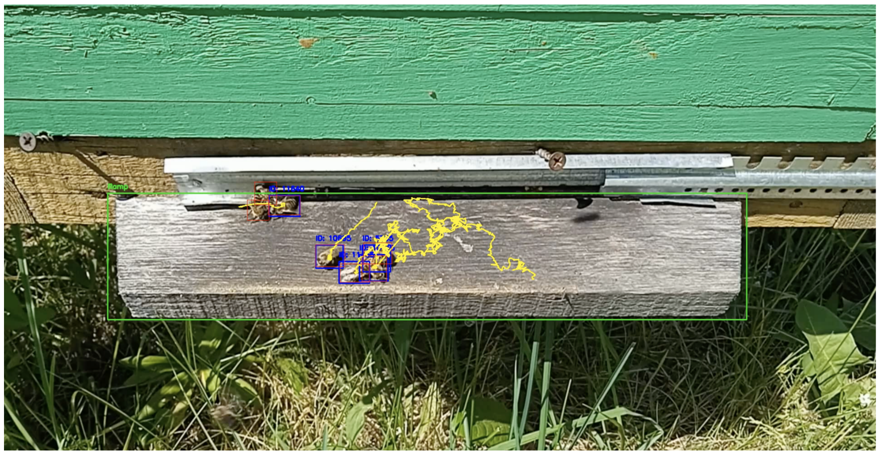

Figure 5. Bee Trajectory Tracking

To find an optimal assignment between new tracks and old tracks, we solve:

$$\min_{\mathbf{X}} \sum_{i=1}^{n} \sum_{j=1}^{m} C_{ij} X_{ij} \quad \text{s.t.} \quad \sum_{j=1}^{m} X_{ij} \le 1, \sum_{i=1}^{n} X_{ij} \le \tag{7}$$

with $X_{ij} \in \{0, 1\}$

This optimization problem solves for the minimum total assignment cost between tracks and detections. The Hungarian algorithm solves this problem in polynomial time, namely in $O(\max(n, m)^3)$ time through augmenting paths in the bipartite graph, processing the dual problem to keep track of the optimality conditions, and clever row and column reduction.

Matched pairs with IoU values of 0.3 or greater count as Kalman updates, while unmatched detections create new tracks with initial covariance $\mathbf{P}_0 = \text{diag}[50^2, 50^2, 1, 1, 10^2, 10^2]$.

When a detection is matched, the respective Kalman filter is updated appropriately: The measurement $\mathbf{z} = [x_{center}, y_{center}, w, h]^T$ updates the state via:

$$\begin{cases} \mathbf{K} = \mathbf{P}_{k|k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R})^{-1} \\ \mathbf{x}_k = \mathbf{x}_{k|k-1} + \mathbf{K}(\mathbf{z}_k - \mathbf{H}\mathbf{x}_{k|k-1}) \end{cases} \tag{8}$$

where measurement noise $\mathbf{R} = 0.1\mathbf{I}_4$.

To make sure that we are not tracking noise, we require that each track have a detection in its first 3 averaged frames. Moreover, tracks that exist for 5 frames without any new detections get terminated. When a track is stopped, the whole trajectory $\{(x_1, y_1), (x_2, y_2), ..., (x_f, y_f)\}$ gets saved for entrance / exit classification. Since we don't have depth information from our video, bees that fly above the ramp but don't actually land on the ramp are also counted as exits or entrances even though they should count as neither. This is a non-issue once we add averaging, since flying bees don't stay in frame long, so they wouldn't be picked up over multiple frames.

Figure 5 shows an example of an intermediate frame where the yellow lines are the plotted bee trajectories, blue bounding boxes show the detected bees, red boxes the ground truth for bees, and the green box shows the ramp.

We classify trajectories based on whether or not they end close to an edge of the ramp's bounding box. We call a trajectory an entrance if its endpoint is within the horizontal bounds of the ramp, and ends within 30% of the top bound of the ramp. This is a good substitute for the bee entering the hive and going out of sight. Trajectories that pass through the ramp, but end outside of the ramp bounds are classified as exits.

The output of the pipeline is a histogram of entrances and exits to the hive as seen in Figure 6

7. Results

We run the complete pipeline on the 5973 frames with and without cropping and get the results shown in Table 3. We see that averaging over 4 or 5 frame yields the best results, and, not cropping the image to just include the ramp makes counting the number of exits fail dramatically. This is due to the fact that we are looking for tracks that end outside the ramp boundary, and so we can't properly detect tracks ending near the edges since the frame is cropped down to the ramp boundary already. We also clearly see, unsurprisingly, that the baseline completely overestimates the number of bees since it doesn't know to differentiate between detection errors, flickering, and clumps of bees.

When not cropping the input model and settling for averaging over four frames, we get a promising indicator of the true number of entries and exits. In fact, if we don't care to distinguish between entries and exits, and use a frame averaging of five for entrances and four for exits, we get a total number of tracked trajectories within 7% of the ground

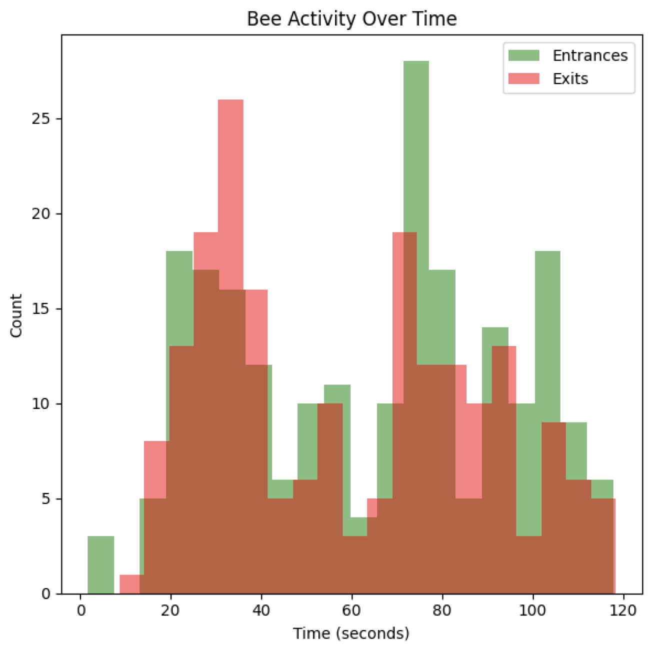

Figure 6. Bee Movements Histogram

Method	Entrances		Exits	
Truth	346		185	
Baseline	898 (+160%)		898 (+385%)	
Frame Avg	**No Crop**	**Crop**	**No Crop**	**Crop**
1	133 (-62%)	170 (-51%)	104 (-44%)	10 (-95%)
2	138 (-60%)	183 (-47%)	118 (-36%)	13 (-93%)
3	161 (-53%)	223 (-36%)	165 (-11%)	10 (-95%)
4	219 (-37%)	269 (-22%)	**201 (+9%)**	11 (-94%)
5	**295 (-15%)**	**347 (+0%)**	271 (+46%)	11 (-94%)
6	545 (+58%)	564 (+63%)	569 (+208%)	15 (-92%)

Table 3. Bee tracking results with different frame averaging values

truth.

8. Discussion

The importance of this work is also that it is usable in the field. As such, it needs quick inference and compute times. We decide to use a frame averaging value of four. These timing results are from a run on an M2 MacBook Air. We get, for inference:

$$\begin{cases} \text{With cropping} \rightarrow \mu = 12.31ms, \quad \sigma = 0.551ms \\ \text{Without cropping} \rightarrow \mu = 20.52ms, \quad \sigma = 0.723ms \end{cases}$$

Inference times are consistent independently of the number of bees in frame since the same computation with the same model weights from the fine tuned model is applied in all cases. We also note that our pre processing cropping strategy (with cropping) is quicker since the resulting image passed to the model is smaller.

However, the number of detections impacts the Kalman update computation steps. Figure 7 shows compute times for the Kalman Filter update step with respect to how many bees were detected in frame. We know that the Hungarian algorithm solves this problem in $O(\max(n, m)^3)$ with n and m the number of existing tracks and new detections respectively. time. However, since we are dealing with such small values, the cubic polynomial doesn't get a chance to grow very large. Even with 12 bees in frame, the update steps takes less than 10ms.

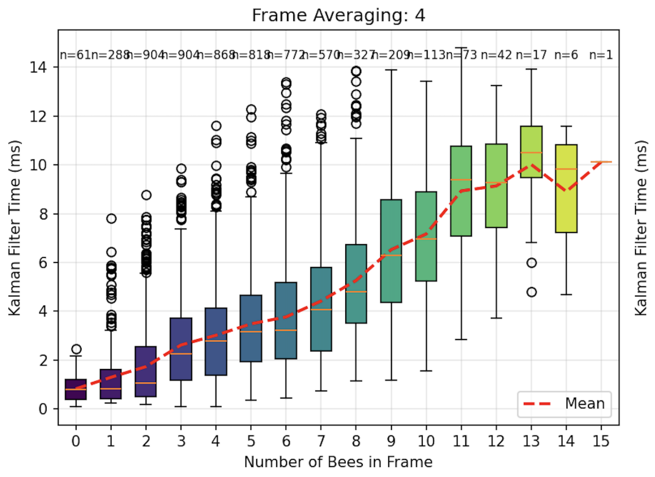

Figure 7. Kalman Filter Compute Time

Using cropping and assuming an average of five bees per frame, we get an average compute time per frame of $12.31 + 3.49 = 15.8ms$. This lets us process videos live at up to 63 frames a second on an M2 MacBook Air. We note that if we wanted to deploy this pipeline into the field, we would need to reduce frame rate or train a smaller model to run on smaller devices like a Raspberry Pi.

9. Future Work

While this project has shown promise, there are several key areas for future work. Firstly, and the easiest next step, is to train the fine-tuning for the bee detection model much longer or on a much larger image size. This will help the model become better at detecting bees in the video frames, which will help avoid trajectory dropouts from detection errors or flickering. Another area for improvement would be to look at a more sophisticated way of dealing with detection dropouts and errors than simply averaging over multiple frames. While this seems to work relatively well, it comes as a trade off with the accuracy of the Kalman matching, since more averaging decreases detection errors, but makes the bees move further during each update step. If many bees are closely clumped together, it will be hard to accurately match each bee to its previous track when frame rate decreases.

10. Conclusion

Overall, we've created a pipeline that detects a beehive's ramp, counts bees on the ramp, and computes trajectories for each bee currently on the ramp with greater than 85% accuracy. It relies heavily on a series of Kalman filters that are matched by optimizing a version of the assignment problem. The whole pipeline can run locally at over 60 frames per second, and provides valuable insight for beehive monitoring in a non invasive fashion. The real time performance makes the system practical enough to deploy in field on multiple hives. The entrance/exit counts our system reports enables beekeepers to have early signals and warning signs for colony health issues such as population decline, robbing events, or unusual foraging patterns that require a beekeeper's immediate attention.

11. Contributions & Acknowledgments

This project was done alone, solely for this class. I'd like to thank the CS231N teaching team for providing an enjoyable and well-taught class.

References

[1] Appicultor. Varroa detector computer vision project - https://universe.roboflow.com/appicultor/varroa-detector, 2023. 2

[2] S. Bilik, T. Zemcik, L. Kratochvila, D. Ricanek, M. Richter, S. Zambanini, and K. Horak. Machine learning and computer vision techniques in continuous beehive monitoring applications: A survey. *Computers and Electronics in Agriculture*, 217:108560, 2024. 1

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] G. Jocher and J. Qiu. Ultralytics yolo11, 2024. 1, 2

[5] P. Kongsilp, U. Taetragool, and O. Duangphakdee. Individual honey bee tracking in a beehive environment using deep learning and kalman filter. *Scientific Reports*, 14(1):1061, Jan 2024. 1

[6] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. 4

[7] Y. Mahajan, D. Mehta, J. Miranda, R. Pinto, and V. Patil. Neuralbee - a beehive health monitoring system. In *2023 International Conference on Communication System, Computing and IT Applications (CSCITA)*, pages 84–89, 2023. 2

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[10] T. Sledevič and D. Matuzevičius. Labeled dataset for bee detection and direction estimation on entrance to beehive. *Data in Brief*, 52:110060, 2024. 2, 3

[11] B. J. Spiesman, C. Gratton, R. G. Hatfield, W. H. Hsu, S. Jepsen, B. McCornack, K. Patel, and G. Wang. Assessing the potential for deep learning and computer vision to identify bumble bee species from images. *Scientific Reports*, 11(1):7580, 2021.

[12] Varroa Virus Detection. Varroa mites detector computer vision project - https://universe.roboflow.com/varroa-virus-detection/varroa-mites-detector, 2023. 2

[13] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.